INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

EVALUATION OF FLIGHT SIMULATION SOFTWARE DEVELOPMENT TOOLS

Reza Ghassemian

A Thesis

in

The Department

of

Mechanical Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

2002

ABSTRACT

EVALUATION OF FLIGHT SIMULATION SOFTWARE DEVELOPMENT TOOLS

Reza Ghassemian

One of the CAE product lines is commercial aircraft flight simulators for which over 400 engineers develop the simulation codes. Currently, aircraft design documents are converted into source code manually. This approach is time consuming, generates large number of errors in the code, and the code generated is very hard to debug. An alternative approach is to use commercially available software development tools to implement the design documents into a visual environment and automatically generate the simulation code for the implemented model. This approach is expected to reduce the software development process, minimize the number of errors in the generated simulation software, and provide user-friendly environment for debugging the code more easily and efficiently, and plus many more advantages. This thesis contributes in the development of such an approach. It addresses the new software development method using $MATRIX_X$ which is one of the leading commercial software development tools widely used in aerospace industries. Two aircraft system, medium commercial jet's flight warning computer and a generic autopilot, have been chosen to evaluate the use of $MATRIX_X$ as software development tool. This thesis will explore the use of $MATRIX_X$ and its advantages over manual coding, and will identify if there are any evaluation criteria or implementation issues that will make the use of $MATRIX_X$ impractical. In addition, it will be examined if there is any need for post-processing utility to adapt the generated code to flight simulation software environment.

ACKNOWLEDGEMENTS

The author would like to take this opportunity to thank his supervisor Dr. J. V. Svoboda for his continuous guidance and support. The author also likes to acknowledge his appreciation to CAE for providing the opportunity to do this project, particularly, to Mr. P. Jarvis for his continuous assistance. I also like to thank Sorin Busuioc for his invaluable help throughout this project. And last, but not least, the author wishes to express his sincere thanks to his family and friends for their invaluable encouragement and support.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF VARIABLES

C_D	Total drag coefficient
C_{D0}	Basic drag coefficient
$C_{D\alpha}$	Drag coefficient due to angle of attack
$C_{D\delta_e}$	Drag coefficient due to elevator angle
C_L	Total lift coefficient
C_{L0}	Basic lift coefficient
C_{Lq}	Lift coefficient due to pitch rate
$C_{L\alpha}$	Lift coefficient due to angle of attack
$C_{L_{\dot{\alpha}}}$	Lift coefficient due to rate of change of angle of attack
$C_{L\delta_e}$	Lift coefficient due to elevator angle
C_m	Total pitch coefficient
C_{m0}	Basic pitching moment coefficient
C_{mq}	Pitching moment coefficient due to pitch rate
$C_{m\alpha}$	Pitching moment coefficient due to angle of attack
$C_{m_{\dot{\alpha}}}$	Pitching moment coefficient due to rate of change of angle of attack
$C_{m\delta_e}$	Pitching moment coefficient due to elevator angle
$\bar{c}$	Mean aerodynamic cord
D	Airplane drag
F_{Ax}	Aerodynamic force along X body axis
F_{Ay}	Aerodynamic force along Y body axis

F_{Az}	Aerodynamic force along Z body axis
F_{Tx}	Trust force along X body axis
F_{Ty}	Trust force along Y body axis
F_{Tz}	Trust force along Z body axis
g	Acceleration of gravity
g_0	Gravity at sea level
h	Altitude
I_{xx}	Moment of inertia about X body axis
I_{yy}	Moment of inertia about Y body axis
I_{zz}	Moment of inertia about Z body axis
I_{xz}	Product of inertia about Y body axis
L	Aircraft lift
L_A	Aerodynamic moment about X body axis
L_T	Trust moment about X body axis
M	Aircraft moment
M_A	Aerodynamic moment about Y body axis
M_T	Trust moment about Y body axis
m	Airplane mass
N_A	Aerodynamic moment about Z body axis
N_T	Trust moment about Z body axis
P	Pressure
P_{SL}	Pressure at sea level
p	Angular velocity along X body axis

$\dot{p}$	Angular acceleration along X body axis
q	Angular velocity along Y body axis
$\dot{q}$	Angular acceleration along Y body axis
$\bar{q}$	Dynamic pressure
R	Gas Constant
r	Angular velocity along Z body axis
$\dot{r}$	Angular acceleration along Z body axis
S	Wing area
s	Laplace domain variable
T_S	Sampling rate
T	Temperature
T_{SL}	Temperature at sea level
U_0	Reference velocity
u	Velocity along X body axis
$\dot{u}$	Acceleration along X body axis
V_T	True air speed
v	Velocity along Y body axis
$\dot{v}$	Acceleration along Y body axis
w	Velocity along Z body axis
$\dot{w}$	Acceleration along Z body axis
z	Z transform variable

α	Angle of attack
β	Angle of sideslip
δ_a	Aileron angle
δ_e	Elevator angle
δ_r	Rudder angle
ϕ	Roll angle
$\dot{\phi}$	Roll rate
λ	Lapse rate
θ	Pitch angle
$\dot{\theta}$	Pitch rate
ρ	Air density
ρ	Density
ρ_{SL}	Density at sea level
τ	Filter time constant
ψ	Yaw angle
$\dot{\psi}$	Yaw rate

LIST OF ABBREVIATIONS

A/T	Auto-Throttle
ADC	Air Data Computer
ADV	Advisory
AFCS	Automatic Flight Control System
ALT	Altitude
AP	Autopilot
BRT	Brightness
CAS	Control Augmentation System
CLR	Clear
DMC	Display management computer
DU	Display Unit
E/WD	Engine Warning Display
ECAM	Electronic Centralized Aircraft Monitor
EFIS	Electronic flight instrument system
EIS	Electronic Instrument System
FWC	Flight Warning Computer
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IAS	Indicated Air Speed
ICAO	International Civil Aviation Organization
INOP	Inoperative
IRS	Inertial Reference System

LNX	Linked Executable
ND	Navigation Display
PC	Personal Computer
PFD	Primary Flight Display
PGUI	Programmable Graphical User Interface
RAM	Random Access Memory
RCL	Recall
RTV	Run Time Variable
SAS	Stability Augmentation System
SBA	SystemBuild Access
SD	System Display
SDAC	System Data Acquisition Concentrator
SEL	Select
STS	Status
T.O	Take Off
TPL	Template Programming Language
UCI	User Callable Interface
XFR	Transfer

1 INTRODUCTION

Flight simulators have proven to be practical devices in the aerospace industry throughout their history. One of the first flight simulators was the Sander Teacher, created in 1910. Using this device, students learned how to balance the simulator in reaction to the wind disturbances by the appropriate movement of pulleys connected to its wings. A variation to the Sander Teacher simulator was Walters' machine where disturbances were introduced by the trainer. Around 1929, one of the first modern simulators was developed by Link, which only allowed for the independent simulation of control surfaces. During the 1930's, the Link trainer was further developed to include altitude, rudder/aileron interaction, and course plotter to allow the instructor to monitor the simulation of the flight. Introduction of analog computers enabled the developers to design electronic simulators to solve the equation of the motion of an aircraft. in which the differential equation was represented by a series of electronic components such as resistors, transistors, and capacitors [1].

After the introduction of digital computers, the development of flight simulators entered a new era. Digital computers enabled the development of more sophisticated flight simulators by introducing motion systems and visual systems. Programming languages replaced the electronic components that were used to model the aircraft dynamics and other various systems. FORTRAN was one of the earliest programming languages used to develop simulation software for flight dynamic models. There have been a vast number of languages developed since the creation of digital computers, of which FORTRAN, C, and ADA have been widely used to write flight simulation software. Rapid development of these programming languages and of digital computer

components has made it possible to meet the requirements of more powerful simulators at a low cost, and has led to the development of state of the art full flight simulators containing visual systems and motion systems with six degrees of freedom [2].

Nowadays, flight simulators are used for a variety of purposes: flying qualities testing, simulation software development, stability and control, flight control development, crew training, flight test support, human factors, flight control research, etc [2]. In order to meet the requirements of growing demands for flight simulators, engineers are working hard to minimize development costs and to shorten production time. Unfortunately, the manual coding of the aircraft systems is costly and requires a great deal of time to enable the engineers to develop, test, verify, and implement simulation software. To overcome this problem, many software packages such as Matlab and $MATRIX_X$, have been developed to allow the designers to concentrate more on the system development process rather than on the manual coding process. In the next section, the manual coding of aircraft systems is briefly discussed. In addition, the disadvantages of manual coding will be addressed. The subsequent section explores available software development packages, their advantages, and the reasons behind selecting $MATRIX_X$ for evaluation as a software development tool.

1.1 Current Method Used to Develop Flight Simulation Software

One of CAE's product lines is the development of state of the art commercial and military flight simulators. CAE builds from thirty to fifty simulators a year for which more than four hundred engineers develop software. The manual software development technique currently used at industries will be discussed briefly in the next section. In addition, the disadvantages of this approach will be addressed.

1.1.1 Manual Code Generation

In order to simulate an aircraft system, its schematic design document is converted to real-time source code. In the current approach, engineers translate the schematic diagrams of the aircraft systems into source code manually. This method is explained with an example; Figure 1.1 shows a small sample of a logical block diagram and the corresponding manually written code is shown in figure 1.2.

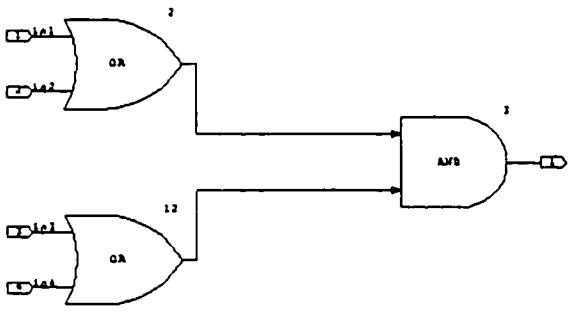


Figure 1.1 Sample design document.

```
{
or_2   = in1 || in2;
or_12  = in3 || in4;
and_3  = or_2 && or_12;
}
```

Figure 1.2 Manual coding of sample design document.

In manual coding, the code for each diagram is written as a separate C equation. The blocks with the highest execution priority are coded first. Normally, each block is

coded in a new line. Each aircraft system could contain up to 50000 lines of source code. Although this method has been used for many years, it is not the most efficient way to develop software for flight simulators today. The disadvantages of this method are explained in the next subsection.

1.1.2 Disadvantages of the Current Approach

The current software design method results in a large number of errors in the developed source code. For example, around 3000 errors were generated in the airbus 330 flight warning computer simulation software. One of the reasons for these errors could be due to typing mistakes. Another reason might be the result of cutting and pasting source code from one model to another that has similar content and neglecting to modify the variables that should be changed for the new model. Moreover, the execution order of blocks could be set wrong while coding the diagram manually. Due to the fact that codes are less visual than the system schematic, generated errors are hard to debug and this in turn slows down the design development process. Furthermore, there is no guarantee that the manual code will directly correlate all blocks in the design document to the hand written code. The generated code in Figure 1.2 can clarify this situation. Some software developers may combine the last two lines of the code into one line or even possibly combine all four lines into one line of code. Doing this will not create any problems in the result of the simulation, but the direct representation of the blocks in the generated code will be eliminated and could create problems when debugging the code or comparing it with the actual design documents. Finally, in manual coding, standardization of the code among different design teams is difficult to introduce and it is also difficult to reuse software components within various projects.

1.2 Alternative Approach Using Software Development Tool

An alternative method for developing simulation software for flight simulators is by using a software development tool. Currently, there are various advanced system modeling and software development packages available. Some of the packages that are commonly used in aerospace and automobile industries are Matlab, $MATRIX_X$, Foresight, Easy5, etc. One question that arises, is which software development package could possibly satisfy the software design requirements for the flight simulation software environment. To answer this question, an extensive study was performed that proposed set of criteria that a tool should meet in order to be considered a good candidate to be evaluated as a software development tool for commercial aircraft simulator product line. Unfortunately, the exact information of why it has been decided to evaluate $MATRIX_X$ can not be provided due to confidentiality issue. However, a general argument can be made to clarify why the $MATRIX_X$ product family has been selected for evaluation as a software development tool.

1.2.1 Advantages of the New Approach

One of the features that these software development tools offer is the ability to implement the design documents in the provided graphical environment, and to generate the corresponding real-time code automatically using their automatic code generator. In other words, these tools provide the designers with a user-friendly environment to implement the aircraft design documents in a more functional and visual manner in contrast to hand coding. After designing a model in their visual environment, the graphical representation of the system model will function as an interface to the

simulation code automatically generated. Implementing the system in a visual manner also allows the designers to compare the original design data with the implemented system more intuitively rather than comparing the design documents with hand written code. This is due to the direct correlation between the design data and the corresponding model implemented graphically that may not exist when coding the system manually. In addition, these tools minimize the dependency on the language that is used on the target machine, since the code is generated automatically and the debugging of the system is performed on the models built in a visual environment rather than manually generated code.

1.2.2 Requirements for Selecting a Software Development Tool

As an initial proposal for this project, a set of evaluation criteria to be used to evaluate $MATRIX_X$ and a set of implementation issues to be resolved during the evaluation period, had been defined. A general argument can be made for the candidacy of $MATRIX_X$ based on the criteria and implementation issues for the flight simulation software environment. Based on the initial proposal and from experience gained during the project, consideration for evaluation of $MATRIX_X$ would be according to the following properties: editing capability, implementation ability, debugging capability, automatic code generator, automatic document generator, and others.

The first requirement that the tool should support is the editing capabilities. The tool should provide utilities to extract information from the model built in its graphical environment. This means that the user should be able to write scripts to extract, modify, and / or automate the creation of any information such as block parameters, input/output connections, etc. The tool should also provide the ability to search for certain parameters

in the designed model. Furthermore, it should support comparison of entire or part of two models built in its schematic environment and it should allow the block icon to be customized or modified on or after creation.

The second requirement is that it must meet satisfactory implementation capabilities. The tool should primarily enable the designer to model a system in a hierarchical fashion. In addition, it should allow multiple instances of the same block to be implemented in the model and in such a way that whenever one of the blocks is modified, the corresponding instances are also modified. Moreover, it should support implementation of state transition hierarchy.

Another requirement the tool should provide, is a complete set of blocks or capabilities that allow users to define customized blocks and organize them in user-modifiable arrangement. The tool should provide condition blocks such as if clause, for loop, while loop, and case container blocks that are only executed when the user-defined condition is satisfied. The tool should also provide a block that allows the user to insert an existing C algorithm into the model.

In addition to the above requirements, the tool should provide the user with an interactive debugger such that the user may input any signal to a block, group of blocks, and / or the entire model, and be able to monitor the output of the simulation. The user should also be able to block step through the model to verify the block execution order in the model. Moreover, the tool should provide data handling of the simulation to initialize certain data, write and read the data from a host program, save the output data for future use, and be able to graphically represent the output of the simulation.

The most important requirement is a real-time C code generator and document generator. The code generated should be easy to follow and well commented such that the corresponding block or blocks of a code segment can be easily recognized from the model. The code generator should also generate the code fast and in an efficient manner by providing the user with options to optimize or minimize the generated code. The generated code should be capable to be linked to other utilities in the flight simulation software environment. Furthermore, the user should also have the option of creating functions, macros, or inline coding of a block or collections of blocks. The document generator should allow the user to generate a user-specified format which would include RTF or HTML format, to insert user-defined parameters and comments in the documents, and to customize or program the document generator easily and efficiently. Finally, the tool should have a user-friendly interface, quality on-line help or documentation, fast execution speed, and a high level of stability [3].

1.3 Why MATRIX$_X$

In the previous section, the characteristics that a tool should contain to be practical for simulation software development were presented. One tool that satisfies all these requirements, and is widely used in aerospace and automobile industries for advanced modeling, simulation, and in software development, is the MATRIX$_X$ product family. A brief overview of the MATRIX$_X$ product family is helpful in order to clarify how MATRIX$_X$ supports the requirements. The MATRIX$_X$ family consists of five products: Xmath, SystemBuild, AutoCode, DocumentIt, and RealSim. These products and their inter-correlation are illustrated in figure 1.3. The application of these products

will be explained in more detail, except for the RealSim application which was not part of requirements for the evaluation of $MATRIX_X$ in this project.

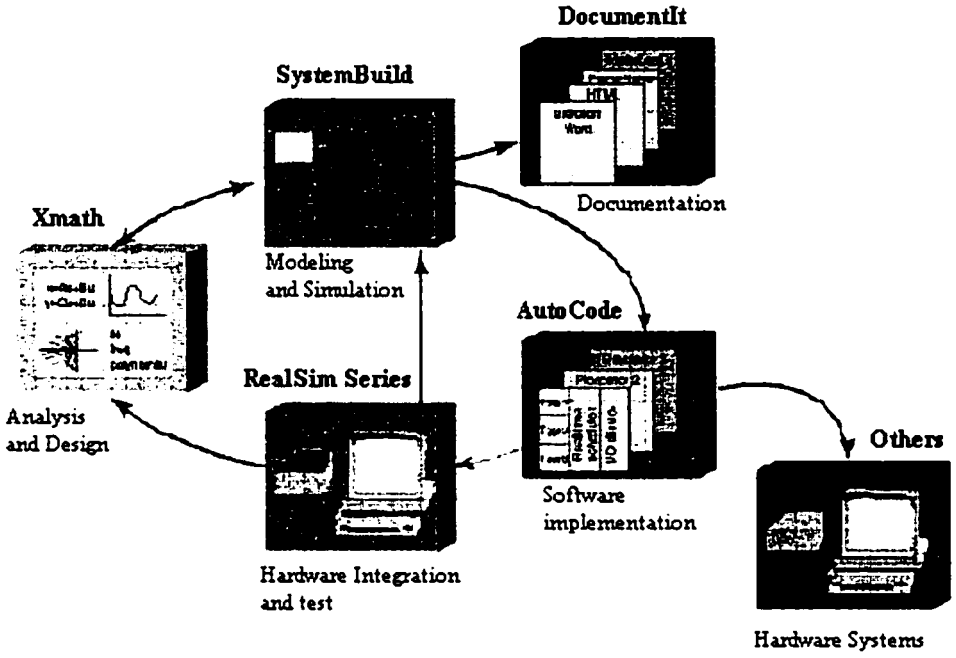


Figure 1.3 $MATRIX_X$ product family overview [4].

1.3.1 Xmath

The Xmath environment provides mathematical analysis, system modeling, and visualization and scripting packages, with over 700 hundred predefined commands and functions. These functions and commands cover applications from basic mathematical operations to more sophisticated levels of graphical user interface (GUI) tools such as the Interactive Control Design tool. In addition, Xmath is an object-oriented design tool that includes numeric, non-numeric, and user-definable objects. These objects provide

suitable data handling, fast and optimized program execution through optimal algorithms, and ease of use due to their intuitive syntax.

Xmath's MathScript programming language facilitates function or command programming and design analysis. Data handling can be automated and customized using MathScript language. Functions and commands can be written in the same way as predefined ones to handle the various types of data objects in an intuitive manner. Users are also provided with MathScript object programming to define and create custom Xmath objects.

Xmath offers an open architecture environment of SystemBuild using SystemBuild Access (SBA) commands. SBA commands can be used to create any SystemBuild module and to modify or query any parameter of the existing module. They may also be used to automate many tasks such as an initialization of parameters included in the SystemBuild models.

Xmath also supports a linkage of its environment to external programs, thereby allowing users to call external routines into the Xmath environment or vice versa. For this purpose, Xmath uses the linked executable (LNX) program to call external C, C++, and FORTRAN routines into Xmath, and the user callable interface (UCI) program to allow external programs to designate Xmath as a server.

Additionally, Xmath supports a fully programmable graphical user interface (PGUI) that allows users to create their own windows, dialogs, result displays, etc. By using PGUI and MathScript scripting languages, users can design powerful user-friendly tools such as a pre-designed interactive filter design tool. For example, this tool can be used to design various filters and is capable of displaying the results interactively in the

same window as the type of design approach and transformation between filters is selected [5].

1.3.2 SystemBuild

SystemBuild is a graphical design environment that can be used to model and implement systems such as dynamic systems. SystemBuild is also used for validation and implementation of systems through non-real time simulation and real-time code generation respectively. SystemBuild supports both top-down and bottom-up design approaches using a hierarchical structure level group of fundamental blocks called a SuperBlock, as well as allowing hierarchical state transition implementation, thus providing an environment for the design of more complex systems.

The SystemBuild environment consists of three main windows: Catalog Browser, SystemBuild Editor, and Palette Browser. The Catalog Browser is a tool that manages the structure of the designed models and handles tasks such as saving or opening the entire or part of the model into or from a file. The Palette Browser is a fully customizable library of over 80 pre-defined blocks. Users may define any number of custom blocks and organize them in an arbitrary manner in the Palette Browser. The SystemBuild Editor is a visual design environment allowing users to design their model by dragging and dropping the blocks from the Palette Browser and Catalog Browser.

SystemBuild is a flexible architecture design environment that allows the user to modify it for specific design requirements. SystemBuild Access commands allow the user to automate the required design specification. Designers may alter the Palette Browser in any way to satisfy the various design environments and to provide team members partial or full access to all blocks defined in the palette. SystemBuild also

provides a components block that can be used to archive, distribute, and license any designed SystemBuild model hierarchy, thus protecting the data and design details from other group members while the overall design is shared among various groups.

SystemBuild also allows the engineers to validate their design in the early phases of a project in order to save time and money. The validation is performed using the SystemBuild non-real time simulation tool called the interactive simulator. The interactive simulator provides a graphical user interface with block and time step debugging capabilities, Run Time Variable (RTV) modification of system parameters, etc. It also supports ten various integration algorithms for high accuracy simulation [6].

1.3.3 AutoCode

AutoCode is a tool that automatically generates real time C or ADA code from the models built in the SystemBuild environment. The generated code is traceable and commented accordingly. AutoCode supports customized code generation using a configuration option and template file. AutoCode also provides modular code generation such as reusable standalone functions using the procedure SuperBlocks.

There are six various procedure SuperBlocks that can be chosen to meet a specific requirement: Standard, Inline, Macro, Startup, Interrupt, and Background procedure SuperBlocks. Standard procedure SuperBlocks are used to generate reusable or stand-alone functions. Inline procedure SuperBlocks are used to generate inline code as if the code is generated for a normal block. AutoCode generates a macro call in the generated code for the macro procedure SuperBlocks. Startup procedure SuperBlocks are used to initialize the required data in the startup phase. Interrupt and Background procedure

SuperBlocks are used to generate code for a specified interrupt routine and for tasks to be executed in the background respectively.

AutoCode supports propagation of block attributes such as signal labels and data types. If signal labels are specified, then they will be used as signal names in the generated code, otherwise the AutoCode generates signal names according to the parent SuperBlock's name and the ID of the block the signal is coming from. Data types, block output scaling, and signal scoping are propagated exactly as they are defined in the SystemBuild environment.

The AutoCode advance option provides a graphical user interface to set the desired options prior to code generation. The code generated by AutoCode is almost fully customizable using a template programming language. AutoCode uses a default template file to generate the code. However, users may define their own template program, or modify the existing template, in order to generate the code that meets their specific requirements.

In summary, some of the most important applications of code generated by AutoCode are as follows: Non-real time and real time simulations, rapid prototyping, stand-alone simulation, SystemBuild user code block, etc [7].

1.3.4 DocumentIt

DocumentIt enables the user to extract the desired information from the graphical presentation of the system in SystemBuild, and to generate software specifications and design documents in a user-defined structure and format. DocumentIt provides a template programming language to tailor the formatting and the structure of the generated documents to satisfy the requirements of the software design documentation.

Furthermore, the template program allows the users to insert any appropriate comments at a user-defined location in the generated document. It also supports templates for various texts format, namely: FramMaker, InterLeaf, Rich Text Format for Microsoft Word, and HTML format [8].

1.3.5 Summary of Advantages For Using $MATRIX_X$

Using $MATRIX_X$, the aircraft system is implemented in the SystemBuild environment, tested and validated using SystemBuild interactive simulation, and finally the corresponding code is generated automatically using AutoCode to satisfy the software design requirements. For this purpose, AutoCode, the real time code generator is used to generate a C code with direct correlation between the generated code and the model built in SystemBuild. The only effort to be made for the code generation process is the development and customization of a template program such that the automatically generated code satisfies the specifications of the target applications. This template program is used as an interface between the SystemBuild models and the generated code to tailor the code to the requirements needed for the target machine. Moreover, the template program needs to be developed only once for any application, and it can be reused for other applications as it is, or with slight modifications.

Another capability that $MATRIX_X$ offers is an automatic document generation from the models built in SystemBuild. The benefit of this facility is more apparent when one thinks of all the changes that will be made to the model in SystemBuild during a debugging process. Any changes made will be reflected in the design documentation, automatically eliminating the need for modifying the documentation manually whenever some part of the model is modified.

Consequently, with use of the $MATRIX_X$ product family, the number of errors that occur in the manual code will decrease dramatically since the design environment is more visual. Secondly, it will maximize productivity by reducing the time required for coding a system. In addition, it provides an environment that simplifies the communication between various design teams and will provide reusability of common models between various design projects. Finally, it will help new engineers to adapt to the working environment much faster and easier [9].

2 THESIS OBJECTIVES

The main objective of this thesis is to evaluate the usefulness of $MATRIX_X$ as a software development tool for flight simulators. The objective of the evaluation is divided into two parts. The first objective is to implement the selected aircraft system in SystemBuild and determine if there are any implementation issues which make the use of $MATRIX_X$ impractical in the flight simulation software environment. The second objective is to comment on the criteria that will be used to evaluate $MATRIX_X$ to determine if it is beneficial to use $MATRIX_X$ to develop software for flight simulators. For this purpose, two different aircraft systems, a reduced scope of medium commercial jet flight warning computer system and a typical generic autopilot system, have been selected for evaluation in order to consider the nature of various kinds of typical systems implemented at industries, namely the Boolean and dynamic control systems. The first part of this chapter covers the method of implementation of the systems in SystemBuild, and the second part discusses the evaluation criteria used to evaluate $MATRIX_X$.

2.1 Implementation of the Selected Systems Using $MATRIX_X$

Implementation of the chosen aircraft systems in SystemBuild is divided into three major steps. The first step is to develop a component library of all the necessary blocks used to model each aircraft system in the SystemBuild environment. Once the component library is defined, each system will be modeled appropriately in SystemBuild to determine whether a suitable simulation code can be generated automatically from the developed model. Finally, if necessary, an appropriate template program will be developed to translate the nature of the each model in SystemBuild into a real-time C

code in such a way that the generated code satisfies the flight simulation software design requirements.

2.1.1 Development of the Component Library

A component library must necessarily be developed for each Aircraft system due to three reasons. The first reason is that the predefined library that is provided in $MATRIX_X$ is not complete, and certain blocks, such as a flip-flop block, have to be developed. The second reason is that normally, when a block is created, its parameters such as name, output type, and icon, have to be initialized. Obviously, it would take long time to initialize these variables each time an instance of any block is created. To solve this problem, a custom block can be created when necessary to simplify the task of the designer by automatically initializing the appropriate parameters upon the creation of each block. The last and the most important reason is that by using custom blocks, users can control the code generation from the models built in SystemBuild. As an example, when using a custom macro procedure SuperBlock, designers can reuse any already defined macro, and by using BlockScript blocks, the code generation for any block can be tailored to meet the design specifications.

2.1.2 Modeling the Boolean System (Flight Warning Computer)

The flight warning computer has been selected to represent a typical Boolean system implemented in flight simulators. The purpose is to implement the flight warning computer in SystemBuild and determine if $MATRIX_X$ is a suitable tool for use in modeling Boolean systems. An appropriate block should be defined for previously developed macros to facilitate reusability and controllability of the code generation.

Furthermore, care should be taken during the implementation of the systems in SystemBuild using properly predefined or customized blocks in order to minimize the amount of generated code and to reflect the schematic of the original design documents, thereby simplifying the comparison of the design documents with their corresponding models in SystemBuild. Since SystemBuild does not provide automatic transfer of input information such as input names and labels from a child SuperBlock to its parent SuperBlock, a utility should be developed to perform this task automatically. The main task in implementing a flight warning computer system in the SystemBuild environment will be to model the system in such a way that a suitable real-time C code can be generated from the implemented model using the AutoCode code generator.

2.1.3 Modeling the Dynamic Control System (Autopilot)

A typical generic autopilot was chosen for the purpose of evaluation of $MATRIX_X$ to reflect a typical dynamic control system implemented in flight simulators. The modeling of the autopilot consists of the implementation of many dynamic control loops and control laws. An appropriate method should be chosen to model the autopilot mode of operation in order to ensure that only the corresponding engaged mode of operation is simulated during simulation. Since flight test data has not been provided for this project due to confidentiality issues, an approximate mathematical model will be developed to calculate the motion variables and the aerodynamic forces and moments acting on the aircraft in order to simulate the autopilot system. Furthermore, a standard atmosphere model should also be implemented to determine the dynamic pressure that is used in aerodynamic forces and moments calculations.

2.1.4 Generation of the Simulation Software Automatically

AutoCode generates a real time C code for the models built in SystemBuild using a template program language. The purpose is to write a template program to generate a code such that the code is as close as possible to a hand written code. Even though the template programming language can be used to tailor the generated code, there are some restrictions that cannot be avoided. A utility should be written to initialize the model appropriately, prior to code generation. In addition, any necessary post processing utilities should be developed to delete undesirable portions of the generated code that can not be eliminated using customized template program.

2.2 Evaluation Criteria

The second objective of this thesis is to use the identified evaluation criteria to evaluate MATRIX$_X$ as a software development tool for flight simulators. The evaluation criteria used are based on the initial proposal given by CAE. However, they have been changed according to experience gained during the project. The evaluation criteria are divided into four groups: general, implementation, unit testing, and code generation. The purpose of each criteria is discussed briefly in the next subsections.

2.2.1 General

One of the reasons for using MATRIX$_X$, is that it provides a suitable working environment that new employees can easily adapt to and learn quickly. For this purpose, a set of criteria has been selected to evaluate MATRIX$_X$ for its general performance. First, it is required to determine the ease of learning and use of the software. After that, the quality of the online help and documentation will be evaluated. Furthermore, the

criteria will evaluate which type of system is best suited for MATRIX$_X$. Finally, a suitable method should be recommended for identifying the differences between revisions of the same system.

2.2.2 Implementation

The main purpose of the implementation criteria is to find out how much time will be saved during the software development process using MATRIX$_X$ compared to manual coding of the system. It will also be used to evaluate the accuracy of the modeling of the systems in the MATRIX$_X$ environment.

2.2.3 Unit Testing

The objective of unit testing criteria is to evaluate whether MATRIX$_X$ provides a friendly user interface for its products (Xmath window, SystemBuild editor, etc.). In addition, the execution speed of the MATRIX$_X$ family and the ease of use of scripting for system simulation will be evaluated.

2.2.4 Code Generation

Template programming language is used to extract the desired section of the implemented model in SystemBuild and to generate the corresponding real time source code automatically. The purpose of the code generation criteria is to evaluate the ease of template programming in addition to the quality and accuracy of the generated code.

3 BOOLEAN SYSTEM

One of the aircraft system that was chosen for evaluation of $MATRIX_X$ as a software development tool is the flight warning computer (FWC) system. The FWC system represents a Boolean system that primarily does not contain any dynamic blocks or dynamic loops. The FWC has been selected to reflect a typical Boolean system implemented in flight simulators. This chapter discusses briefly the operation of the typical flight warning computer system and describes how the warning messages are arranged and provided to the pilots.

3.1 Flight Warning Computer

The flight warning computer is responsible for computing warnings and cautions occurring during the various flight envelopes, and providing the crew with operational assistance for both normal and abnormal configurations of the aircraft systems. Two identical flight warning computers provide this operational assistance through electronic centralized aircraft monitor (ECAM) display units that display visual messages and use loudspeakers to announce aural alerts and synthetic voice messages.

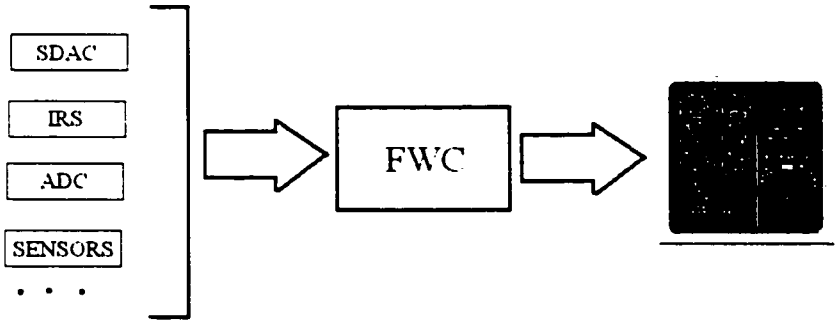


Figure 3.1 Basic overview of FWC operation.

As shown in figure 3.1, the flight warning computer receives the required data directly from aircraft sensors and systems for generating red warnings, or from two identical system data acquisition concentrators (SDAC) for generating amber cautions. SDAC acquires data from aircraft sensors and sends the appropriate signals to the display management computer for display of status pages and engines parameters, or to the flight warning computer to display ECAM messages or aural alerts. The FWC also provides radio altitude callout, decision height callout, landing distance and landing speed increment during approach by synthetic voice messages [10][11].

3.1.1 Types of Failures

There are three different kinds of failures: Independent, Primary, and Secondary. In an independent failure, an isolated system or part of equipment has been affected without influence on the performance of other systems in the aircraft. Primary failure occurs when an affected system or part of equipment degrades the performance of other systems or parts of equipment. Finally, the secondary failure is a failure of a system or a part of equipment because of primary failure [11].

3.1.2 Warning/Caution Classification

Messages generated by the flight warning computer can be divided into two groups: failure messages and information messages. Failure messages are subdivided into three groups: level 3 red warning, level 2 amber caution, and level 1 amber caution. Level 3 red warning occurs when the aircraft is in a dangerous configuration, a flight condition limit has been reached, or there is a system failure alerting the system safety. In these cases, the configuration or failure needs immediate action from the crew. Level

2 amber caution indicates a failure that does not directly affect flight safety and does not require the immediate action of the crew, but alerts the crew to the configuration or failure. Level 1 amber caution only requires crew monitoring and it is the result of failures leading to loss of redundancy or system degradation. Information messages are subdivided into two groups: advisory and memo. Advisory messages provide the crew with system parameters, and memo messages inform the crew of normal or automatic selections of functions that are only temporarily used [11].

3.1.3 Priority Rules

Level 3 warning messages that require the immediate action of the crew have priority over level 2 caution messages, which in turn have priority over level 1 caution messages. There are a set of rules within each level of messages which are taken into account when multiple messages of the same priority level have to be sent to the display units.

3.2 Electronic Instrument System (EIS)

The components of the electronic instrument system are shown in the following block diagram. The figure shows the flow of the data from the aircraft sensors and systems to the flight warning computer. After these data are analyzed by the flight warning computer, they are sent to the display units through the display management computers or to the loud speakers directly from the FWC to announce the appropriate messages. The operation of each component of electronic instrument system will be described in more detail in the following sections.

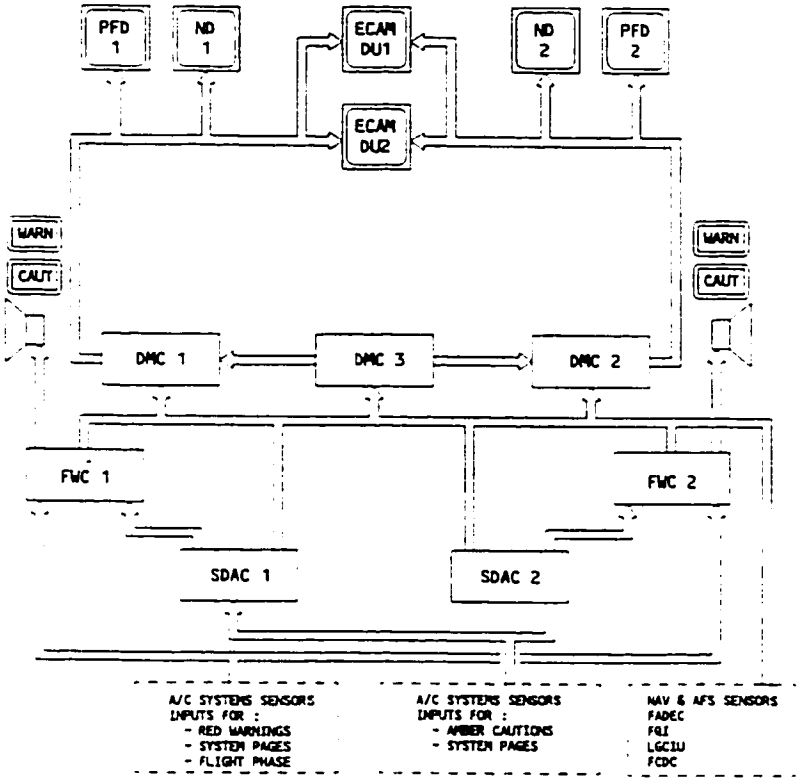


Figure 3.2 Electronic instrument system components [11].

3.2.1 Display Units (DU)

The electronic instrument system display units consist of six identical full colored cathode ray tube display units, which provide the crew with flight and aircraft system information. The EIS is divided into two subsystems: Electronic flight instrument system (EFIS) and Electronic centralized aircraft monitor (ECAM). EFIS consists of four display units, two primary flight display (PFD) units that provide mostly flight parameters, and two navigation display (ND) units to display navigation data. ECAM consists of two display units: Engine/warning display unit (E/WD) which provides engine indications, warning and caution messages and memos, and system display (SD) which displays a synoptic diagram of aircraft systems, status messages and permanent data [11].

3.2.2 Display Management Computer (DMC)

Three identical display management computers receive data from the aircraft sensors and other computers. They process the data and generate the appropriate signals to be displayed on the display units. Each computer consists of two independent EFIS and ECAM channels, and is capable of feeding one PFD, one ND, and either EW/D or SD at the same time.

3.2.3 System Data Acquisition Concentrator (SDAC)

Two identical SDACs receive data from aircraft system sensors and feed the DMC and FWC. DMC uses the signals to generate system pages and engine parameters and the FWC uses the signals to generate amber caution messages and aural alerts.

3.2.4 Attention Getting Devices

There are two sets of attention getting devices; one for the captain and one for first officer located under the glareshield. Each set consists of one "Master Warn" light that flashes for red warning, and one "Master Caut" light that flashes for amber cautions.

3.2.5 Loudspeakers

Aural alerts and voice messages are announced using the loudspeakers provided in the cockpit even when set to off.

3.3 Electronic Centralized Aircraft Monitor (ECAM)

ECAM consists of two display units; Engine warning display (E/WD) and System/status display (SD). The E/WD displays warning/caution messages or memos. The SD displays system synoptic or status.

3.3.1 ECAM Control Panel

The ECAM control panel is shown in figure 3.3. The Operation of each button is explained in more detail.

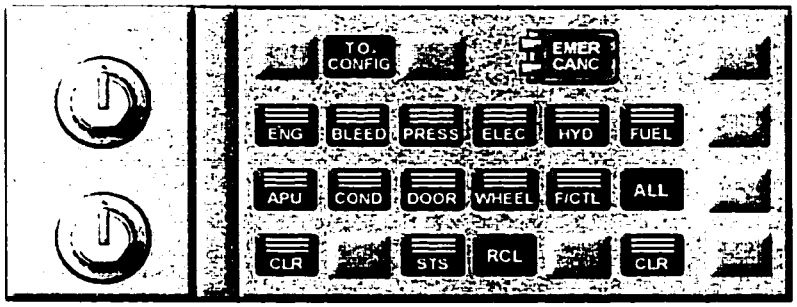


Figure 3.3 ECAM control panel [12].

The OFF / BRT knobs control respective ECAM display unit brightness and turn them on or off. If the upper display unit is turned off, E/WD is automatically displayed on the lower ECAM display unit.

Pushing the T.O. CONFIG button performs takeoff power application simulation test to trigger a warning in case the airplane is not in proper takeoff configuration. The "TO CONFIG NORMAL" message is displayed on the E/WD if the test has been passed.

When the EMER CANC button is pushed, the next step taken depends on the type of situation. In a warning situation, the current aural warning is cancelled and the MASTER WARNINGS lights are extinguished without affecting the ECAM message display. In a caution situation, any current caution messages and aural signals are first canceled for the rest of the flight, and then the STATUS page automatically displays the message "CANCELLED CAUTION" and the title of the inhibited failure.

There are eleven system page buttons that when pushed, display the related system page on the SD and light up after being manually pushed, or when an advisory is detected. If same button is pushed a second time, it calls up the related current flight phase or current warning system page.

If the ALL button is pressed and held, the lower ECAM displays all the system pages in sequence at one-second intervals. If the ECAM control panel fails, this button can be pushed and held until the desired page is shown, releasing the button will select the page.

The CLR buttons illuminate when a warning, caution, or status message is displayed on the E/WD. Pushing an illuminated button will change the ECAM display.

When the STS pushbutton is pushed it lights up and the STATUS page is displayed on the SD. If there are no status messages to be displayed, the message "NORMAL" will appear on the STATUS page for 5 seconds. Pressing the STS button a second time or the CLR button will clear the STATUS page.

Pushing the RCL button recalls suppressed warning or caution messages that have been obscured by use of the CLR button or caused by the flight phase inhibition. If there are no suppressed messages the "NORMAL" message is displayed on the E/WD for five seconds. If this button is pushed and held for three seconds the caution messages, canceled by pushing the EMER CANC button, will be displayed on the E/WD [11].

3.3.2 Color Code

Warning and caution messages on ECAM are color coded to indicate the importance of each message. Red is used for warnings that need the immediate action of the crew. Amber is used for caution messages, which indicate awareness but no

immediate action is required. Green indicates normal operation. White indicates title and remarks. Blue messages are limitations or actions that need to be carried out. Finally, magenta is used for special messages such as inhibition messages [14][15].

3.3.3 Failure of ECAM DU

In case the upper ECAM DU fails, the E/WD automatically replaces the SD on the lower ECAM DU. System or status messages can be displayed either on the ND unit by using the ECAM/ND XFR switch, or they can be temporarily displayed on the lower ECAM DU for a maximum of 30 seconds by pushing and holding the required system page button on the ECAM control panel. Alternately, if the lower ECAM DU fails, the system/status messages can be displayed either on the ND unit using the ECAM/ND XFR switch or they can be temporarily displayed on the upper ECAM DU for a maximum of 30 seconds by pushing and holding the required system page button on the ECAM control panel. If both ECAM DUs fail, the engine warning messages can be displayed on the ND unit using the ECAM/ND XFR switch and system/status messages can be displayed on the ND unit for a maximum of 30 seconds by pushing and holding the required system page button on the ECAM control panel [11].

3.3.4 Aural Indication

Red warnings are announced by continuous repetitive chimes as long as the warning exists, unless the pilot pushes the "EMER CANC" button on the ECAM control panel or the "MASTER WARN" button located under the glareshield (this excludes 'over speed' or 'landing gear not down' warnings). Amber caution warnings are announced by a single chime for a duration of half a second.

3.3.5 ECAM Procedure

Whenever a failure occurs in a system, ECAM performs the following sequence. The corresponding warning and caution messages are displayed on the E/WD. Except for level 1 amber caution messages, the master warning or master caution lights on the side panels light up. Aural indication is sounded except for level 1 amber cautions. The SD unit displays the system page corresponding to the failed system to help the crew identify the affected system. The CLR button lights up on the ECAM control panel. When all necessary actions have been performed to overcome the failure, the CLR button should be pushed repeatedly until only MEMO messages are displayed on the E/WD, the SD displays the present flight phase system page, and the CLR button light on the ECAM control panel is off [11][13].

3.4 Engine/Warning Display (E/WD)

The engine/warning display unit is in the upper ECAM display unit and is divided into four parts as shown in figure 3.4. On the upper left side the engine parameters are displayed, and on the upper right side the total fuel on board and slat / flap positions are displayed. The lower left part, in case of a failure, displays primary or independent warnings and cautions and in normal operation it displays memo messages. The lower right part displays secondary failure, memos, special lines (e.g. "AP OFF"), or the title of the affected system by an independent or primary failure when there is an overflow on the lower left part of the E/WD.

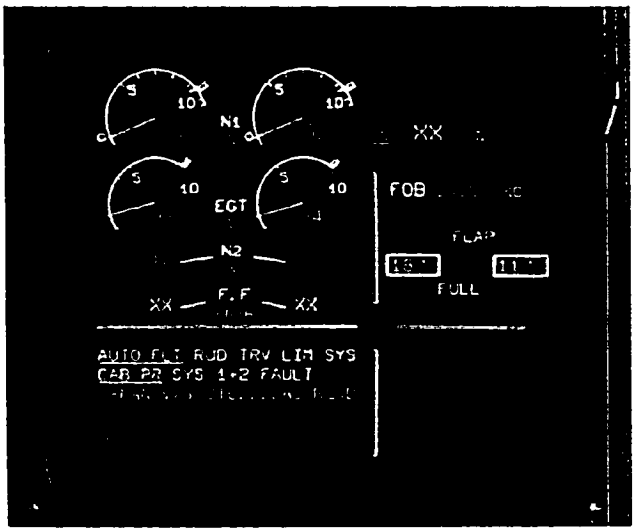


Figure 3.4 Engine/Warning display unit [17].

3.4.1 E/WD Message Management

There is a priority order defined for the display of messages on the E/WD. The priority order is as follows: special lines (e.g. red warnings or limitation), level 3 red warnings, level 2 and level 1 amber cautions, secondary failure messages, and memos.

Messages are displayed on the E/WD from top to bottom in Z fashion. As soon as a system failure is detected and no flight phase inhibition has been activated, the title of the failure and the corresponding actions that should be taken are displayed on each line separately. Each action line, except in some cases, is cleared as the flight crew has performed the required action. Once each line is deleted, the empty line is filled by upward transfer [10].

Memo messages that were displayed on the lower left part of the E/WD prior to the occurrence of the failure reappear when the CLR button on the ECAM control panel is pushed once. In addition, if there is a secondary failure after executing each action, it

will be displayed on the lower right part of the E/WD. Also, system pages corresponding to the secondary failure are displayed automatically on the SD to indicate the affected area. After the CLR button is pushed a second time, normal operation memo messages are displayed on the lower right part of the E/WD and the status page is called up on the SD to provide the crew with operational assistance to complete the flight with the affected area. Pushing the CLR button for a third time results in an automatically call up of the system page for the current flight phase and a status reminder appears on the bottom of the E/WD.

In order that different failures can be easily identified, the primary failure title is boxed and the secondary failure is shown with a star in front of the title of the affected area. If there are too many messages to be displayed in the space provided on the lower part of the E/WD, a green arrow pointing downwards will appear at the bottom of the display unit to indicate the overflow of the messages off screen. The crew can view that information by pressing the CLR button on the ECAM control panel [11].

3.4.2 Flight Phase Inhibition

The flight envelope has been divided into ten phases from starting point power on to engine shut down. According to these phases, the flight warning computer's functions have been divided in order to improve the efficiency of its operation. In some phases, such as landing or takeoff, the flight warning computer inhibits some warnings and cautions that would be displayed on the ECAM display units to decrease the pilot's workload. For example, consider certain failures are inhibited in phase three, if the failure occurs before entering phase three; the warning will be displayed immediately and will be displayed as long as the failure persists even during phase three. But if the failure

occurs during phase three, the warning is not displayed until the airplane exits phase three and enters phase four where the warning is not inhibited. Then the warning is displayed as long as the failure persists [11].

3.4.3 Memos

Memo messages indicate a temporarily used system or function under normal situations. They are displayed on the lower part of the E/WD and are normally green. In this section, only take-off and landing memos are discussed.

Take-off memo messages are displayed automatically two minutes after the second engine has been started, or manually when the pilot pushes the CONFIG TEST button if at least one engine is on. They are blue before the required actions are performed and are green afterwards. Figure 3.5 shows a T.O memo after the take off configuration test has been performed and everything is in normal configuration. These memo messages are erased at the moment of take off power application [11].

Figure 3.5 Take off memo messages.

The landing memo is displayed when the aircraft's altitude is below 1500 ft if the landing gears are down, or below 800 ft if the landing gears are not down. The color of the message is blue if the required action is not performed, and green if the action is

performed. After touch down, this memo is erased. The landing memos are shown in figure 3.6 [16].

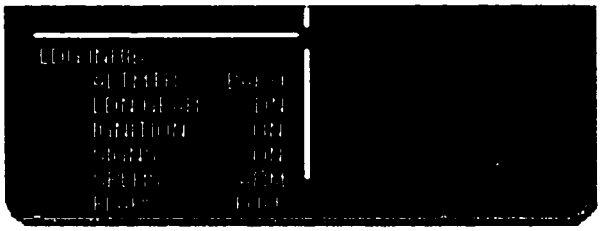


Figure 3.6 Landing memo messages.

3.5 System Display (SD)

The system display is the lower ECAM display unit and is used to display a synoptic diagram of system pages or the status page. As an example, the system status that corresponds to the warning messages in figure 3.4 is shown in the following figure.

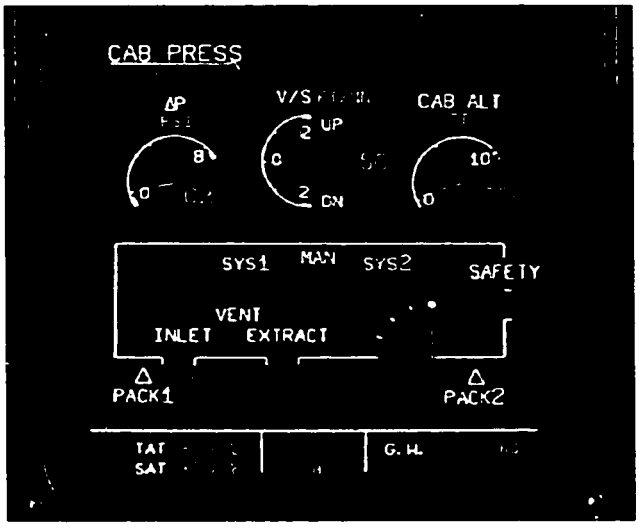


Figure 3.7 System Display Unit [17].

3.5.1 System Pages

There are twelve different system pages that can be displayed on the SD. The status pages are secondary engine parameters, air bleed, cabin pressurization, electric power, hydraulic, fuel, auxiliary power unit, air conditioning, doors/oxygen, wheel, flight control, and cruise. System pages are displayed either automatically or can be called up by the flight crew to be displayed on the SD.

The pilot can call up each system page, except the cruise page, by pushing the appropriate button on the ECAM control panel. The chosen system page will be replaced automatically in case of warning or advisory messages.

Automatic call up of a system page can be caused by a failure, advisory message, or flight phase mode. If a failure is detected the appropriate system page is immediately displayed on the SD.

System page relating to the phase of operation is automatically displayed if no warning/caution or advisory messages are detected.

Advisory call up occurs when a parameter is out of its normal range. In this case, the appropriate system page is displayed with the out of range parameter flashing in green until the value is back to normal. If only one ECAM display unit is operating, the relevant button on the ECAM control panel flashes to indicate the defect area and a boxed advisory message is displayed on the upper part of the message area on the E/WD as shown in Figure 3.8. In some flight phases the advisory call up is inhibited [10][11].

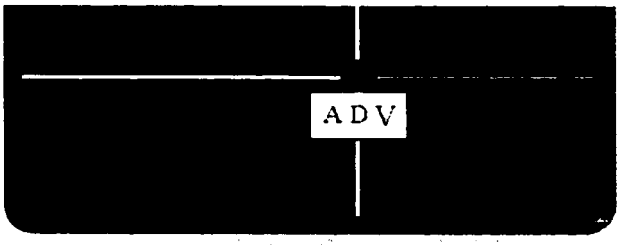


Figure 3.8 Advisory message on E/WD.

3.5.2 Status Page

The status page is filled from top to bottom. This page displays Limitations, Approach procedures, Procedures, Information, Cancelled Cautions, Inoperative systems, and Maintenance status. The first four are displayed on the left part of the SD from the top with a line separating each group. Cancelled cautions are displayed on the lower left side, INOP SYS are displayed on the upper right side, and Maintenance status is displayed on the lower right side of the SD. In case of an overflow, a downward arrow is displayed on the lower part of the SD to indicate information off screen. By using the CLR button on the ECAM control panel the pilot can view the overflow information.

The status page is displayed automatically when the crew performs all required actions in case of failure or when, during descent, the slats are extended and there is no information or only maintenance messages are available. The status page may be displayed manually by pressing the STS button on the ECAM control panel. The E/WD shows the status reminder "STS" when there is other information than "CANCELLED CAUTION" or MAINTENANCE available. If there is still MAINTENACE information at the time of engine shut down, the status reminder "STS" will flash on the E/WD [11].

3.5.3 Permanent Data

There are five parameters that are displayed on the bottom of the SD called permanent data. These parameters are shown in figure 3.7. Total air temperature and static air temperature are displayed in green on the bottom left side of the SD. The value of the load factor if it is out of limit is displayed in the lower middle part of the SD or, in case it is not displayed, the selected altitude from the flight control unit is displayed in green if it is selected in metric units. Universal time coordinated is also displayed in the middle part of the SD. Finally, on the lower right part of the SD, the gross weight is displayed in green [11].

4 DYNAMIC CONTROL SYSTEM

A typical generic autopilot system was selected to evaluate the use of $MATRIX_X$ as a software development tool for flight simulation software. The autopilot system represents a typical dynamic control system developed in flight simulators. This chapter provides the user with a brief theory about the autopilot system. It concentrates primarily on the pitch channel autopilot since only this part of the autopilot has been implemented and tested using $MATRIX_X$.

4.1 Automatic Flight Control Systems

The automatic flight control system (AFCS) provides the pilot with the basic piloting functions that reduce unnecessary or dangerously high workloads that pilots must perform in order to allow the pilots to concentrate on the more important tasks of the flight envelope [18]. The detailed concept behind the design of automatic flight control systems is beyond the scope of this project. Therefore, only a brief overview of the concept is discussed. In deriving the equations of motion of the airplane many assumptions have been made to simplify the derivation of the equations [19][20]. Using these assumptions, the equations of motion of the aircraft can be divided into two groups: longitudinal motion and lateral motion, each containing three equations.

The solution of longitudinal motion for a general aviation aircraft results in two different modes: the short period mode and the phugoid mode. The short period mode is a high frequency heavily damped oscillatory response, which is felt as a bump by the flight crew or passenger. The phugoid mode is a lightly damped long period oscillatory response, which is very annoying if not controlled.

The solution of Lateral motion for conventional aircrafts results in three different modes: spiral mode, roll subsidence mode, and dutch roll mode. The spiral mode is slowly divergent implying instability. The roll subsidence mode is highly damped, and finally the Dutch roll mode is a damped oscillation with various damping ratios and frequencies for different types of aircrafts.

If the high frequency unstable modes have a higher frequency than the pilot bandwidth, it proves to be very difficult for the pilot to control the airplane in these modes. Therefore, closed loop automatic control systems have been designed to sufficiently dampen these modes to ease the pilot's workload [21]. Normally, AFCS can be divided into three groups; stability/control augmentation system, autopilots, and flight guidance systems [18].

4.1.1 Stability/Control Augmentation Systems

The stability augmentation system (SAS) is a closed-loop control system that uses the control surfaces to damp the unwanted high frequency angular motions in pitch, roll and yaw [22][23]. The control augmentation system (CAS) controls the unwanted motion and also provides the pilot with a specific type of response to the control inputs, such as pitch rate or normal acceleration. Examples of these systems are the yaw damper system and pitch-rate augmentation system [22].

4.1.2 Autopilots

The autopilot system is a closed loop system that is used to establish and/or maintain a required flight condition. The autopilots are designed to perform various tasks from single axis control to the more sophisticated three axis control [21].

4.1.3 Flight Guidance Systems

The flight guidance system either provides the pilot with the necessary information to fly and control the airplane manually, or combines the information with autopilot to control the airplane automatically. The former is called the flight director system which provides the information through a flight director indicator on the primary flight display units. The latter system is called the flight management system and it couples the function of autopilot with data provided by the flight director to control the airplane automatically [18].

In early airplanes the flight director and autopilot had separate computers to perform their tasks, and generally the autopilot computers were more sophisticated than the flight director computer. In contrast, in modern airplanes today, the autopilot computer also performs the task of the flight director. Consequently, the pilot can let the autopilot fly the airplane in the desired mode and use the flight director to monitor autopilot performance [24].

4.2 Autopilot

The autopilot is used to maintain and/or establish the required flight condition. Autopilots are designed in various degrees of operation and control functions. They can be very simple and only control the airplane in single axis, or can be fully equipped and control the airplane in pitch, roll, and yaw axis.

4.2.1 Single-Axis Autopilot

The single-axis autopilot normally controls the airplane only about the roll axis using ailerons control surfaces. This kind of autopilot is very simple in concept and it

uses only the ailerons in the control loop to maintain the airplane's lateral stability more commonly known as the wing level. In addition, it has the ability to track and maintain the heading of the airplane by using signals from the navigation system. It is also able to turn the airplane automatically using the command signals from the pilot [25].

4.2.2 Two-Axis Autopilot

The two-axis autopilot primarily controls the airplane in a roll and pitch axis. There are two control loops with ailerons and elevators in the loop to control the airplane in roll and pitch. Normal modes of operation of this autopilot are heading-hold, manual turn control, and in some cases, altitude-select and altitude-hold modes [18].

4.2.3 Three-Axis Autopilot

Three-axis autopilot is capable of controlling the attitude of the aircraft in roll, pitch and yaw using ailerons, elevators and rudder respectively. It is also able to combine the ailerons and the rudder to perform co-ordinate turns.

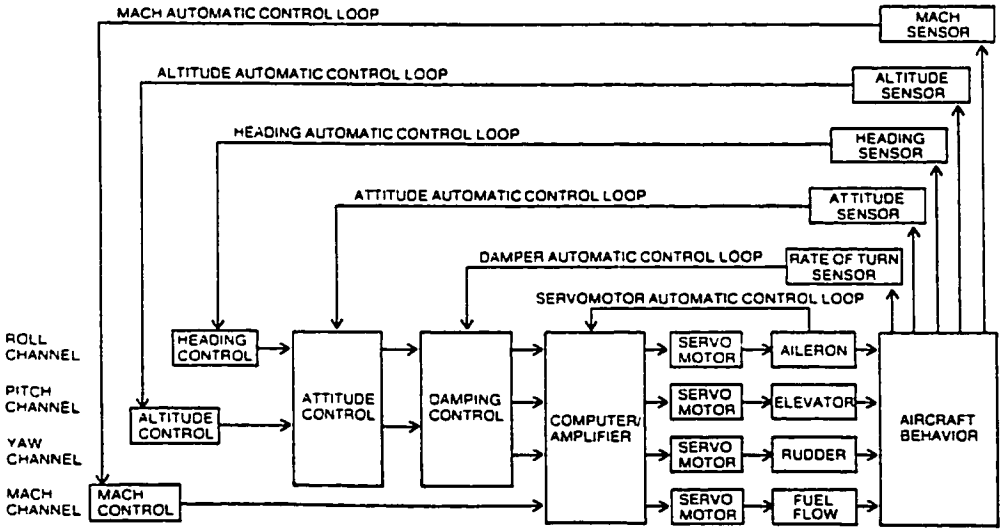


Figure. 4.1 Typical three-axis autopilot [26].

Figure 4.1 shows the block diagram of a typical three-axis autopilot used for commercial aircraft. It also illustrates the autopilot's ability to maintain the mach number or speed in addition to altitude control and heading control [26].

4.2.4 Autopilot Control Panel

The automatic flight control system must be capable of letting the pilot interact and send commands to the system in order to change the aircrafts attitude. This is done through the autopilot control panel. Figure 4.2 shows a typical large commercial jet autopilot control panel that is located under the glare shield. For this aircraft, autopilot computers are also used as flight director computers and the same knobs are used for autopilot and flight director modes as opposed to other aircrafts that use different buttons and knobs for the autopilot and flight director. In addition to mode select panel there is also a controller panel containing turn control knobs and pitch wheel controls.

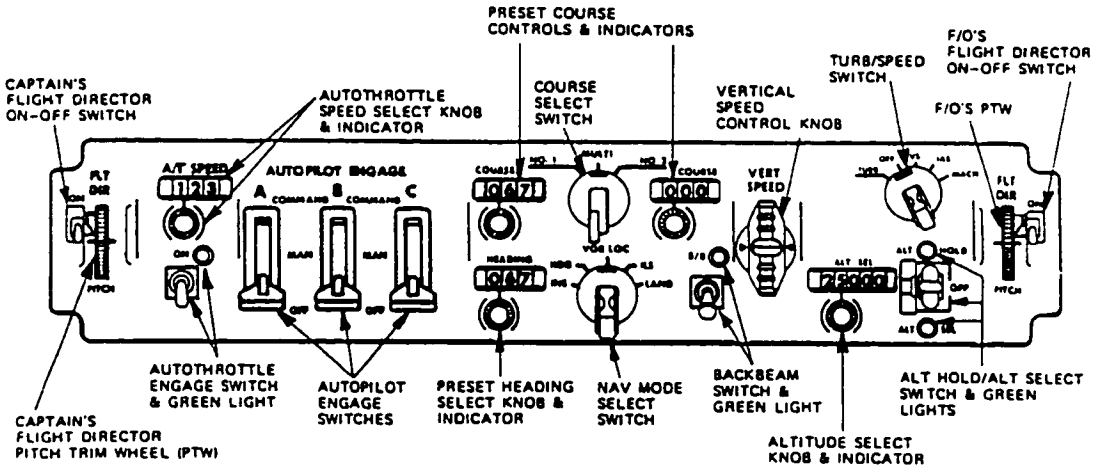


Figure. 4.2 Typical mode select panel for large commercial jet [25].

The two flight director switches engage or disengage the corresponding flight director channel. When the auto-throttle (A/T) system is on, it operates the throttle to maintain the selected speed during an automatic landing. The autopilot engage levers are locked in "off" position until certain interlocked circuitry is fulfilled in order to ensure that the autopilot is able to safely take control of the aircraft. When the levers are in manual position, only the turn knob control and heading hold modes are accessible for the roll channel, and the pitch wheel control can be used for pitch attitude hold mode except for altitude hold mode. There are two course select knobs that control the course select bugs on the captain's and first officer's horizontal situation indicators. The mode of both autopilot and flight director are controlled through the mode select switch in the middle of the panel. If the autopilot is set to manual mode, the mode select switch only controls the flight director modes. When the switch to the right of the altitude select knob is on the "ALT SEL" position, the airplane will capture the selected altitude shown on the digital read out and hold the desired altitude after capture. Finally, an automatic landing is possible when all the three (for category III landing), or at least two autopilots (for category II landing) are engaged and operational, and two flight directors are also operational [24].

4.2.5 Autopilot Control Loops

Each axis of the autopilot system commonly consists of two loops as shown in figure 4.3. The first loop is the inner stability loop that controls the stability of the airplane by sensing any variations or disturbances to the airplane's attitude, and then returns the disturbed attitude to an acceptable stable position.

The outer loop receives data such as airspeed and heading from the airplane's sensors, pilots, and the navigation radio in order to modify the currently stabilized attitude by performing some maneuvers to reach a new stable path. The data required to change the current attitude to a specific flight path depends on the mode of operation. The number of modes of operation depends on the type of autopilot designed. The modes are selected by the pilot through the mode control panel, except in some cases when they are automatically selected. For example, the glide slope mode is activated automatically as soon as the airplane is in the approach mode [27][28].

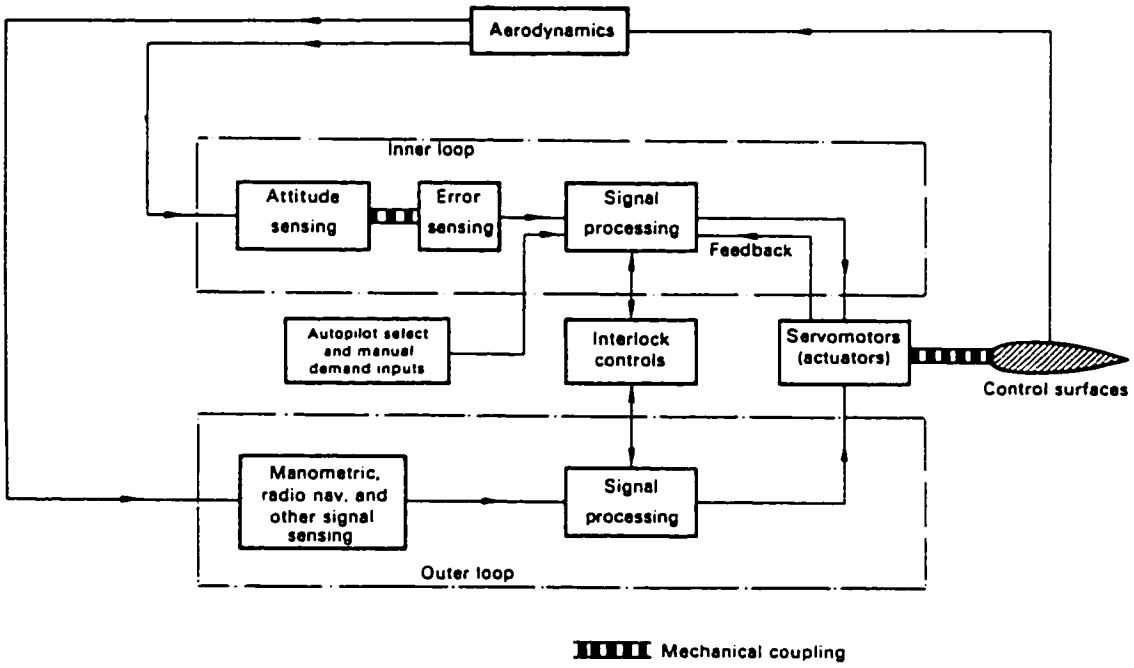


Figure 4.3 Inner and outer loops of autopilot system [25].

4.2.6 Autopilot Modes of Operation

The autopilot automatically controls the airplane in the roll, pitch and yaw axis, depending on the mode of operation selected by the pilot. If the autopilot computers are

used as flight director computers, these modes of operation (except in some cases) are the modes of the flight director as well. Some common autopilot and flight director modes for roll and pitch channel are shown in table 4.1. In addition to the mode of operations given in the table, the autopilot is capable of controlling the rudder in conjunction with ailerons for turn coordination.

PITCH AXIS	ROLL AXIS
Altitude Hold	Heading Hold
Altitude Preselect Track	Heading Select
Altitude Preselect Capture	VOR Capture
Glideslope track	VOR Track
Glideslope Capture	Dead Reckoning
IAS	Turbulence
Mach	Back Course
Pitch	Localizer Capture
Take-off Vertical	Localizer Track
Vertical Speed	Take-off Lateral
Go-Around	Roll

Table. 4.1 Pitch and roll mode of operation [29].

4.3 Pitch Axis Autopilot

In this project, only the pitch axis autopilot has been implemented in SystemBuild to evaluate the use of $MATRIX_X$. Therefore, only pitch axis modes are discussed in more detail.

4.3.1 Altitude Hold Mode

In this mode, the autopilot controls and maintains the altitude of the aircraft from any deviation of the commanded altitude. This mode is usually engaged in cruise mode. Normally, the pilot controls the aircraft during the climb and descent, and once the desired altitude has been reached, the pilot engages this mode to let the autopilot control the aircraft's altitude. In more sophisticated autopilots, the pilot is able to select an altitude and the autopilot will automatically reach that altitude and engage the altitude hold mode to maintain the commanded altitude. Figure 4.4 shows a typical block diagram of the altitude hold autopilot.

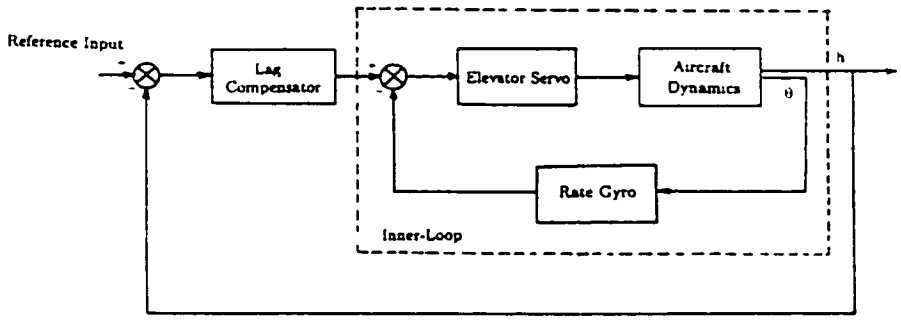


Figure 4.4 Typical schematic diagram of altitude-hold autopilot [30].

The lag compensator block is required to maintain the stability of the outer loop. The system is unstable for positive values of the lag compensator's gain when there is only altitude feedback, therefore pitch rate feedback is necessary to keep the system stable [30].

4.3.2 Altitude Preselect Capture/Track Mode

In this mode, the autopilot controls the aircraft to capture and track the preselected altitude. Figure 4.5 illustrates the operation of this mode.

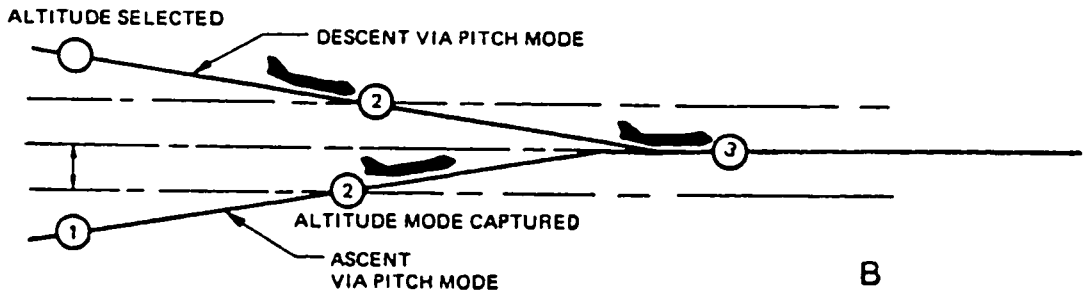


Figure 4.5 Altitude preselect capture/track mode [26].

At point 1 the pilot preselects the desired altitude while the pilot may select any vertical mode to fly the airplane to the capture point, except glideslope capture, go-around or altitude-hold modes. At point 2, which is the capture point, the autopilot automatically selects the altitude preselect capture mode to capture the selected altitude. The capture point depends on the rate of climb or descent at which the airplane is flying toward the selected altitude. It is also limited by the maximum distance from the desired

altitude. Once the selected altitude is reached, the track mode maintains and controls the selected altitude [24].

4.3.3 IAS Hold Mode

The speed control is used by more sophisticated autopilots and is normally integrated in high performance aircraft systems. The autopilot uses the indicated air speed hold mode or the mach hold mode to control the speed of the aircraft. The IAS hold mode is normally used during the approach and flight phase.

One way of controlling the speed is by using auto-throttles, which use feedback control loops to control the throttles. The block diagram of this approach is shown in figure 4.6.

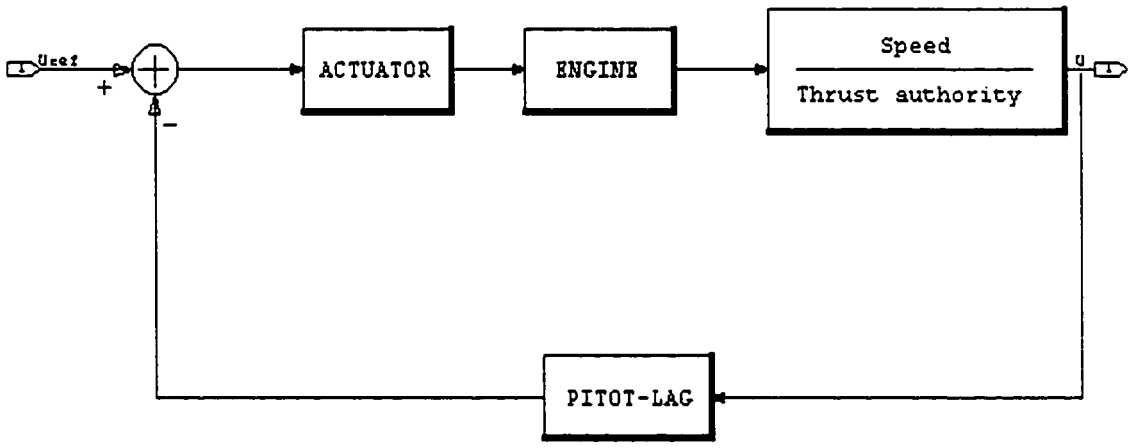


Figure. 4.6 Speed control using auto-throttle [31].

The analysis of the system (given in the reference) demonstrates that the performance of auto-throttles improve with faster engine response [31].

In the above approach, the engine-lag can have severe effects on the control of the speed. Therefore, in some aircrafts, auto-drag controls are used instead of auto-throttle. The auto-drag approach uses speed brakes as a feedback to control speed. The advantage of this approach is that it causes less damage to the engine because the speed is controlled at a constant throttle setting. The auto-drag block diagram is shown in Figure 4.7 [31].

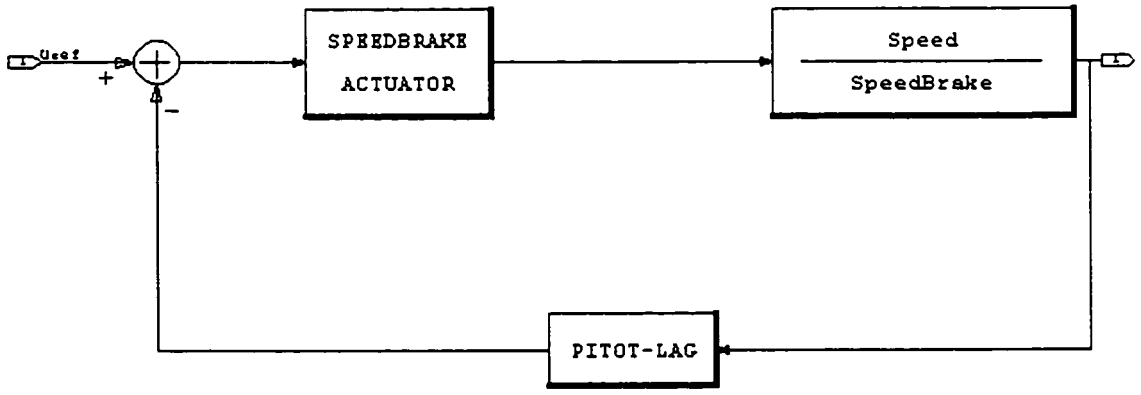


Figure. 4.7 Speed control using auto-drag [31].

4.3.4 Mach Hold Mode

This mode is used during the cruise flight phase and it controls and maintains the mach number at a constant value. Before this mode is engaged, the pilot should fly the airplane to a level flight path and reach the desired mach number. Once this mode is engaged, the autopilot maintains the desired Mach number. During the cruise, as airplane fuel decreases, the speed of the airplane increases. Therefore, the autopilot sends a

command to the elevator to cause the airplane to climb smoothly in order to keep the mach number constant. A sample block diagram of mach hold mode is shown in figure 4.8 [32].

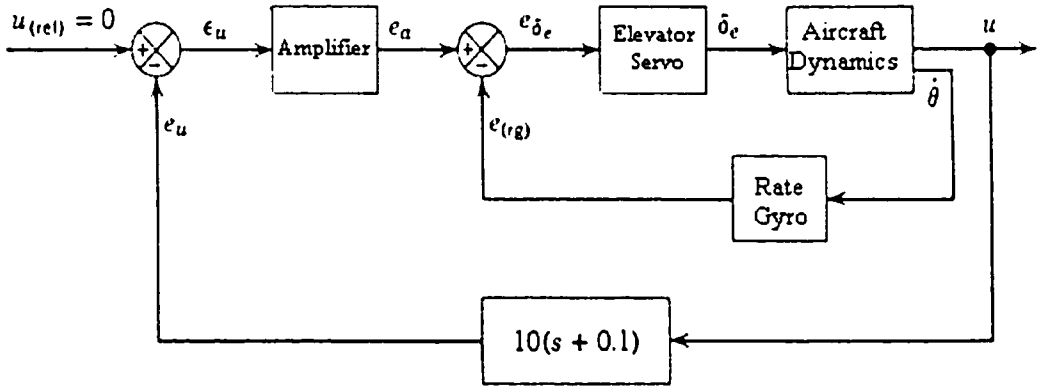


Figure. 4.8 Block diagram of mach hold mode [32].

4.3.5 Pitch Mode

This mode is usually used in a wing-level flight condition. In this mode, the autopilot controls and maintains the airplane's pitch attitude at the time of engagement. A sample block diagram of this autopilot is shown in figure 4.9. The dynamic compensation assures that the steady-state error of the system is small. The pitch rate feedback into the inner loop provides good overall damping of the system. In the diagram G_c represent dynamic compensator, k represent gain [33].

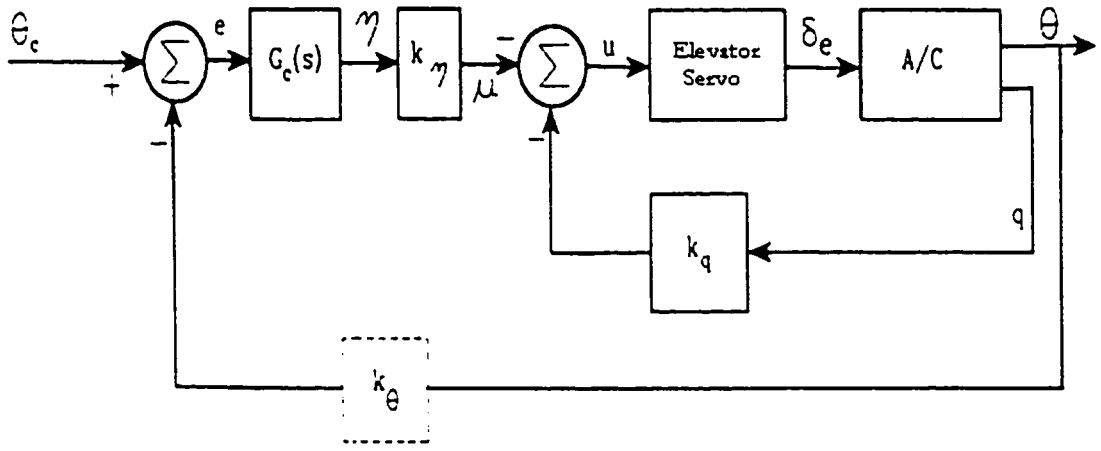


Figure. 4.9 Block diagram of pitch hold autopilot [33].

4.3.6 Vertical Speed Mode

In this mode, the autopilot controls and maintains the reference vertical speed. The reference vertical speed is the actual vertical speed of the airplane at the time of engagement, but it can be changed using the vertical speed control wheel located on the controller panel or on the mode control panel in some cases [34].

A typical vertical speed control block diagram is shown in figure 4.10. The gain control will assure a smooth dynamic response. The disadvantage of this system is that the vertical speed below the minimum drag speed cannot be controlled, and in this case the pilot must take control of the aircraft [35].

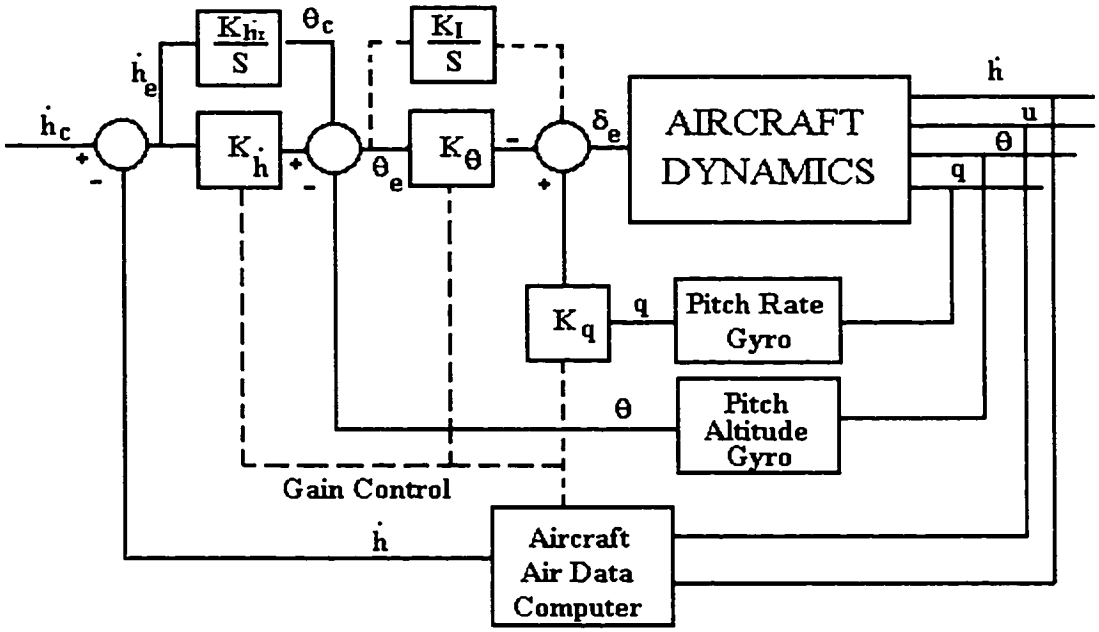


Figure 4.10 Vertical speed control block diagram [35].

4.3.7 Glideslope Capture/Track Mode

This mode is engaged automatically when the airplane is in approach mode. In this mode, the autopilot captures and tracks the glide slope beams radiated from the glide slope transmitter located near the runway threshold. The capture of the beams can be from above or below the glide slope beams, and as soon as capture occurs, the other mode of autopliot is automatically disengaged. The capture point depends on the closure rate. Figure 4.11 shows an airplane flying under the glide path. From the figure it can be observed that in order to capture the glide path, the value of the angle Γ should be zero. This concept is used to design a system to capture and track the glide slope beams and is shown in figure 4.12.

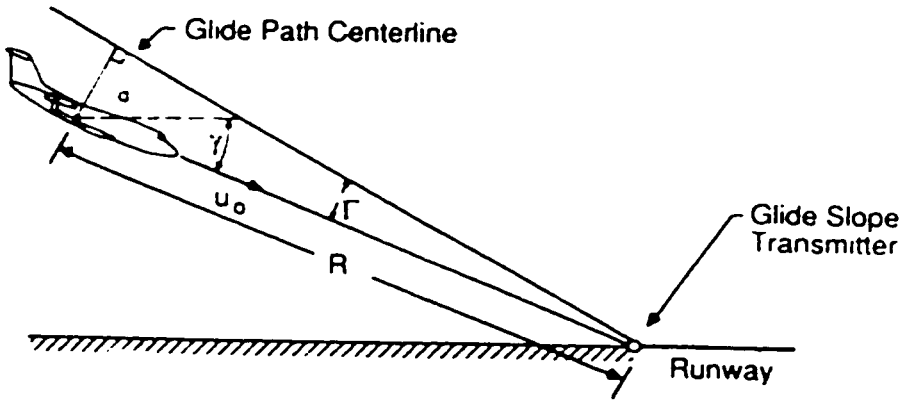


Figure.4.11 An airplane flying below the glide path [36].

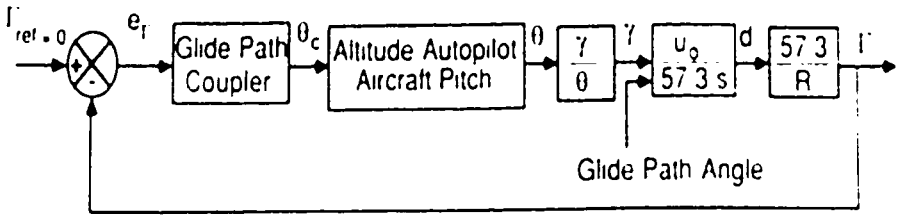


Figure 4.12 A block diagram of the glideslope capture mode [36].

5 MODELING AIRCRAFT SYSTEMS USING MATRIX$_X$

Two types of aircraft systems were chosen for evaluation in this thesis: the Dynamic Control System and Boolean System. For the purposes of evaluation of MATRIX$_X$ for the dynamic and Boolean systems, a typical generic autopilot and a medium commercial jet's flight warning computer were used respectively.

5.1 Useful Tools Developed for This Project

MATRIX$_X$ comes with a number of sample tools, in some cases incomplete or even non-functional, to show users the power of its math scripting language, especially when it is combined with its fully programmable user interface (PGUI) language. Even though these tools are not fully functional, they provide users with some ideas to develop their own tools to simplify many tasks during design development. In this project, it was useful to develop new tools, or to modify an existing non-functional tool, to facilitate certain tasks such as searching the whole model for a specific type of information. Nevertheless, it is necessary to mention that these tools were developed only for evaluation purposes and may not be fully functional for all system models, but can be modified in the same manner to work with any developed system.

5.1.1 Search Tool

A model built in SystemBuild, such as an aircraft system, may consist of thousands of blocks and SuperBlocks, each containing as many as hundreds of pieces of information like names, IDs, labels, user parameters, input signals and many others. Suppose a designer has to find a bug in a previously built model after working couple of weeks on a new project. Normally, there is a great chance that the designer does not

remember all the details that were used in the previously designed model. A search utility would be a good option for searching for a certain variable that was used in the model. For this purpose, a tool was developed to search for specific information in the entire model and provide the user with all of the blocks and SuperBlocks containing that information in a user-friendly interface.

This tool only searches for the common keyword of blocks or SuperBlocks. It does not search for all keywords that a block or SuperBlock may include. It neither supports search of State Diagram parameters nor DataStore Keywords. A complete search tool can potentially be developed, but would require more time and programming beyond the scope of this thesis and is left for future development.

The search tool GUI consists of four buttons, two pull down menus that allow the user to select a block or SuperBlock and the corresponding keyword to be searched for, a text field to enter the value of the keyword, and two scroll fields that display the results. The following example demonstrates the operation of the search tool. Assume a user wishes to search all of the blocks containing the user parameter variable named Ac.S (Aircraft wing area) in the autopilot model. Figure 5.1 shows the results of the search.

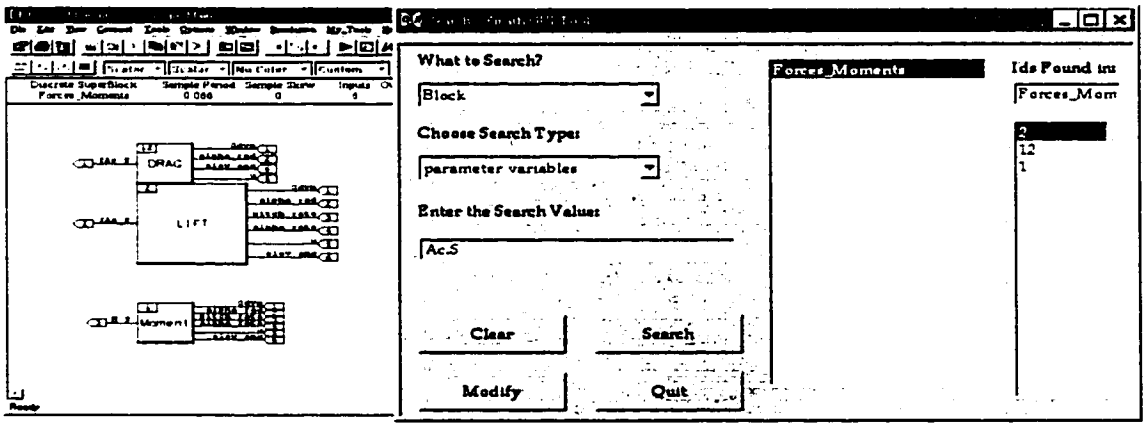


Figure 5.1 Search tool GUI and operation.

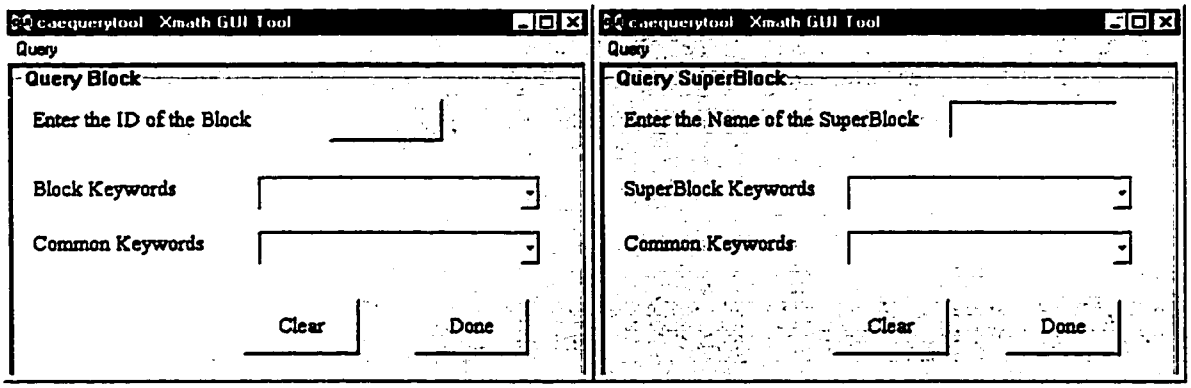
The field in the middle of the GUI displays the names of all of the SuperBlocks that contain the block with the specified keyword. The selection of any of the SuperBlocks in this field will first bring up the SystemBuild editor with the selected SuperBlock, and then it will display all of the ID's of the blocks containing the keyword in the selected SuperBlock in the right field. Double clicking any ID in the right field will double the size of that block in the SystemBuild Editor to simplify the recognition of the block in the editor. The clear button will initialize all variables used in the search tool and prepare the tool for a new search. The modify button has not been implemented and is left for future development.

5.1.2 Query Tool

Queryblock and querysuperblock are two of the SystemBuild Access commands that are used frequently during the development of scripts or tools to automate certain tasks that will simplify the workload of the designer. These commands are used to extract information from the blocks and SuperBlocks using their respective keywords. Consider that the designer wishes to write a script to extract the IDs of the blocks included in a SuperBlock, one must question which command, combined with which keyword, would yield the best results. To answer this question, one must try each command using various keywords. To do this, the query command and the keywords with the correct syntax must be typed into the Xmath command area. The problem with this solution, is that the designer either has to remember the correct syntax for all the keywords, or has to check the help file to find out the syntax each time this command is going to be used. For this reason, it is beneficial to develop a tool with a user-friendly

interface that will enable the user to use this command without having to remember the syntax for each keyword.

The graphical interface of this tool is illustrated in figure 5.2. This tool consists of a pull down menu from which the user may either choose to use the queryblock command or the querysuperblock command. Selecting either of the commands will change the window's appearance for use of the selected command. Many of the keywords are common to their respective class of blocks and some keywords are only used for specific type of blocks, therefore two combo boxes are available; one for common keywords and one for specific keywords. To query for a SuperBlock or a block keyword, the name of the SuperBlock or the ID of the block should be entered in the field provided for each, and any keyword may be chosen from either of the combo boxes to reflect the results in the Xmath results field.



(a) Query block GUI. (b) Query SuperBlock GUI.

Figure 5.2 Query tools.

This tool is also useful for non-advanced users of $MATRIX_X$ to explore the function of each keyword simply by choosing them from the combo boxes and observing

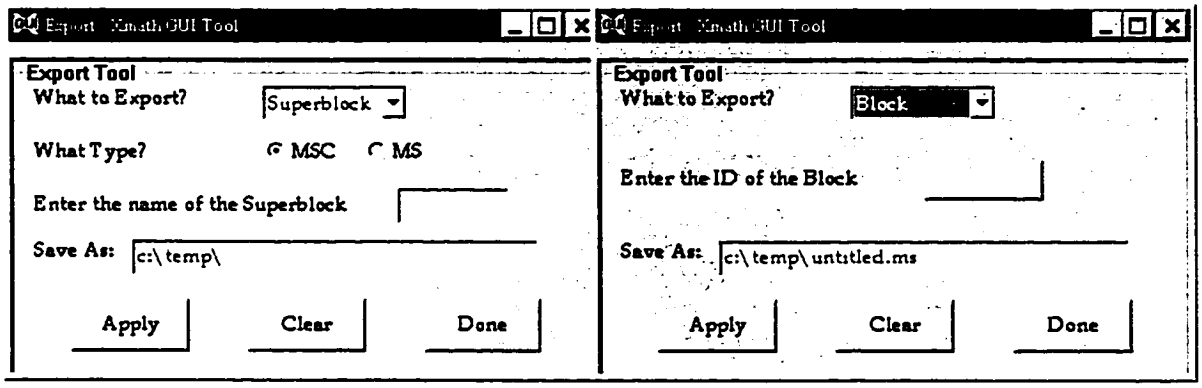
the results in the Xmath window. Note that this tool may not be fully functional since it has been designed solely to monitor the function of each keyword in order to simplify the design of the scripting programs written for this thesis. Further development of this tool is left for future work if found to be useful for users.

5.1.3 Export Tool

The export SuperBlock and export block are two predefined samples of math scripting tools that come with the $MATRIX_X$ product family. They are helpful tools that save the models built in the SystemBuild environment in the form of a math scripting language and SystemBuild Access (SBA) commands. This form of saving is more useful when it is used for viewing purposes in editors such as notepad. The export Scripts are also very helpful for designing custom blocks if these blocks need a callback function (used for creating and initializing a block whenever it is dragged and dropped from the Palette Browser). Normally, the callback function contains many SBA commands that can be extracted directly from the above saved files and used with a slight modification in the callback function. This will eliminate the need to remember all of the SBA commands and their syntax when writing the callback functions.

The problem with the existing export SuperBlock script was that if a SuperBlock contained another SuperBlock, the tool would export the child SuperBlock as a continuous SuperBlock without any blocks inside no matter what type of attribute the child SuperBlock had and whether or not it contained any blocks. In other word, it did not support the export of the SuperBlock hierarchy. Therefore, it was necessary to modify the tool to export the SuperBlock hierarchy correctly. A GUI was also developed for this tool to facilitate the use of this tool by providing a friendly interface for the user.

The interface of this tool is shown in figure 5.3. The default value for the export upon activation of the tool is the SuperBlock. The user may select to export a block by using the provided combo box. It also allows the user to enter the directory in which the file is to be saved. It is necessary to mention that this tool may not be fully functional, especially if the SuperBlock contains State Diagrams or DataStores. Future work is required to complete the development of this tool.



(a) Export SuperBlock GUI. (b) Export block GUI.

Figure 5.3 Export tools.

The usefulness of this utility is demonstrated with an example. Consider that a designed block needs to be customized with a callback function. First, using this utility, all the information about the block can be extracted in the form of a math script function and the required SBA commands. This is shown in figure 5.4. Then, the required keyword with the corresponding values can be cut and pasted from this math script file directly into the call back function. This procedure will not only save time but will also eliminate the need for remembering the SBA command's keywords and syntax.

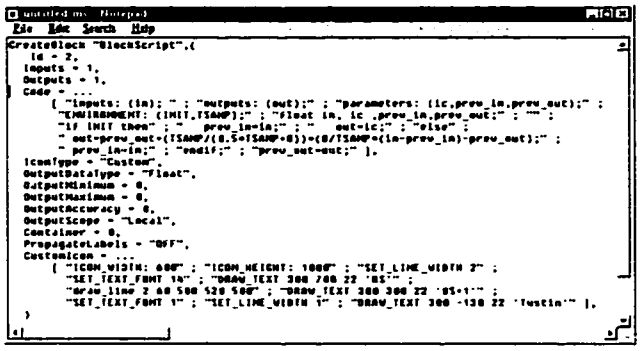


Figure 5.4 Output of export tool for sample washout filter.

5.1.4 Adding or Removing a Specific Input from a SuperBlock

This utility was developed to add or delete an input from a SuperBlock. Assume that a SuperBlock has ten inputs and input number five needs to be deleted. SystemBuild provides automatic deletion of the inputs from the last entry as is shown in figure 5.5. Therefore, one way of deleting input number five is by using the block property dialog to remove the last input and change all of the input's attributes and connections from input six to ten to input five to nine. This approach is very cumbersome especially if the SuperBlock has many inputs.

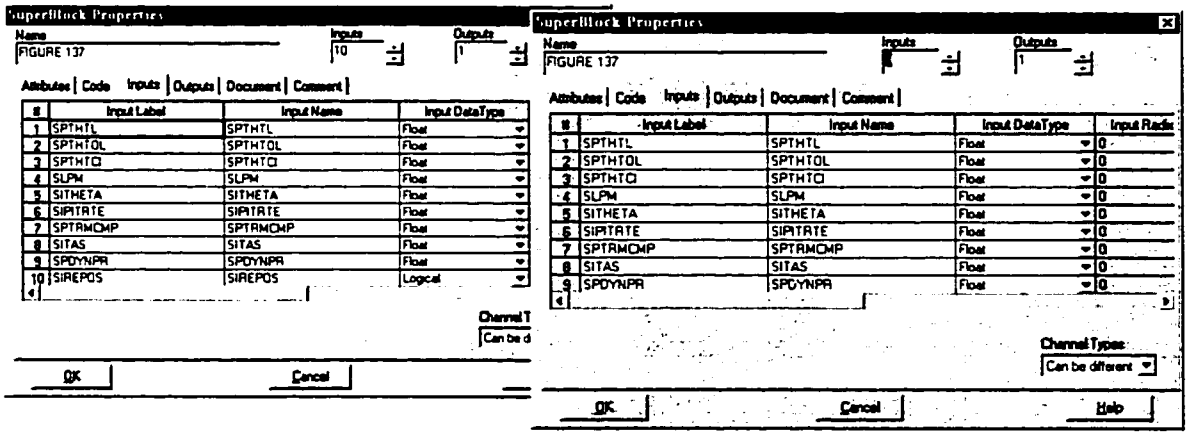


Figure 5.5 Deleting an input using the SuperBlock property.

Another method is to use the SBA command to extract all of the information about the connections and input attributes in the form of a matrix, and then delete line five and restore the results back in the SuperBlock property. This approach is the obvious choice, but it requires some programming. Consequently, it is handy to create a utility to avoid performing the same task every time an input needs to be erased from a SuperBlock.

The interface of this utility is shown in figure 5.6. Users can choose to add or remove an input, to or from a specific row by entering the name of the SuperBlock and the number of the input that needs to be added or erased. This utility also keeps track of the external connections; whenever an input is added or erased from a super block, the external connections are also shifted accordingly.

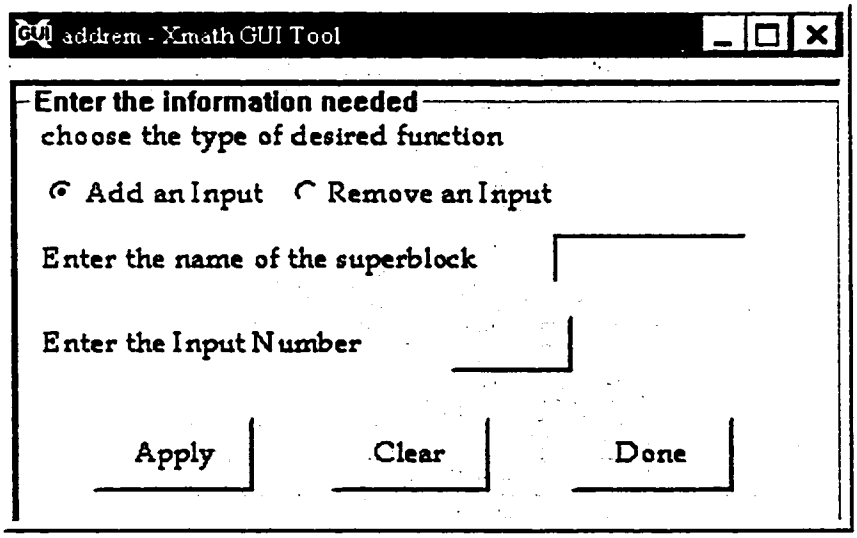


Figure 5.6 Addrem tool.

5.1.5 Searching for a BlockID in the SystemBuild Editor

This is a very small but helpful utility that identifies a block in the SystemBuild Editor. Whenever an error occurs during a simulation, the cause and the hierarchical location of the block that caused the error is displayed in the Xmath log area. The SuperBlock containing this block sometimes has many other blocks, and the more blocks a SuperBlock contains, the smaller the blocks appear in the editor, and the harder it is to recognize the block ID by just looking at the editor. Normally, it is necessary to zoom in on the editor's contents and move around the editor using the cursors to find the block. This utility will recognize the block using its ID and will double the size of the block in the editor, eliminating the need for zooming in and searching through the editor. The operation of this utility is shown in figure 5.7.

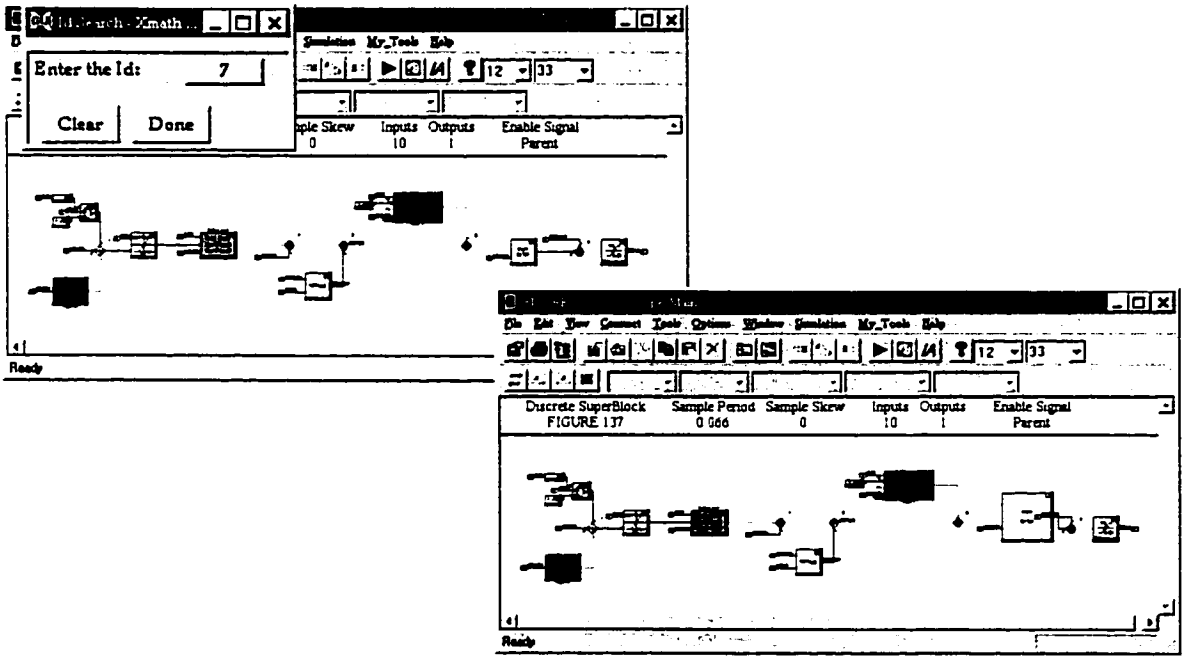


Figure 5.7 ID search utility and its operation.

5.2 Modeling Flight Warning Computer Using MATRIX$_X$

Flight warning computer (FWC) messages cover various aircraft systems such as autoflight, hydraulics, engines, etc. In this thesis, only part of the engine model was chosen to evaluate MATRIX$_X$. The main interest was to generate a C code using MATRIX$_X$ so that the generated code would be as similar as possible to a manually written code. This approach, if achieved, would yield reusability of the existing CAE dispatcher and predefined macros and functions. In addition, the generated code would be more readable, easier to follow, and most importantly, more controllable. The following subsections will demonstrate the design of custom blocks, initialization utility, complete customized template file, and all other necessary programs needed to achieve the generation of the desired code.

5.2.1 Approach Used for Modeling Flight Warning Computer

The MATRIX$_X$ open architecture environment offers the designer various approaches for achieving the design requirements. However, the problem is to decide which approach should be taken in order to obtain maximum benefits from MATRIX$_X$. To answer this question, designers should consider all options and try each of them, analyzing all results and finally decide on what technique would yield the highest success. This procedure requires time and experience with MATRIX$_X$. In spite of that, the approach used to model the FWC system in this thesis was based on experience gained during the short time given to evaluate MATRIX$_X$.

The approach used was to model the FWC and generate a C code using AutoCode in such a way that the generated code would look exactly like the manually written code. However, even though MATRIX$_X$ offers vast user customization of its environment,

there are many restrictions to modification of its environments. For this reason, an initialization utility was needed to pass some variables from the SystemBuild environment into the generated code using template programming language, a post processing utility was needed to eliminate undesired parts of the generated code, and many custom blocks needed to be created to manipulate the nature of the generated code.

5.2.2 Why Design Custom Blocks?

One of the advantageous features of the SystemBuild environment is the ability to design custom blocks. SystemBuild comes with various predefined block libraries, but these block libraries are not complete and users need to design custom blocks such as confirmation and flip flop blocks. There is another reason for designing custom blocks, and that is to control the generated code for blocks. In some cases, it is important and sometimes critical to control the size of the code due to the memory that is reserved for each subsystem's simulation code on the flight simulator. As an example, consider the AND block that comes with $MATRIX_X$; the code generated for this block is shown in figure 5.8. Essentially, there is nothing wrong with the code and it definitely represents an AND gate operation, but as shown in the illustration, there are three lines of code generated for this block that could be replaced with just one line of code. This does not mean that the shorter code runs faster, but simply means that less memory would be needed to store the generation code. This would be more evident if the reader would think of all the AND blocks that are used in the entire model. If AutoCode generates two more lines of code for each of these blocks, then the simulation code will contain thousands of lines that could be simply eliminated. The same applies to the other predefined logical blocks. To solve this problem one could design a custom AND block

by using a logical expression block. AutoCode generates only one line of code for this block as illustrated in fig 4.8, thus large amounts of code would be removed from the simulation code.

```
/* ---------------------------- Logical Operator -- AND-OR-NOT */
    /* {test.b_and_1.1} */
    test = U->input1;
    test = test && U->input2;
    Y->b_and_1 = test;
/* ---------------------------- Logical Expression */
    /* {test.b_and_2.2} */
    Y->b_and_2 = U->input1 && U->input2;
```

Figure 5.8 Comparison of the generated code for predefined and customized AND gate.

Another reason to use custom blocks, is to control the generation of the source code. This is due to the fact that the users do not have any access to manipulate the code generation at the block level for any blocks. However, using customized BlockScript blocks, Macro procedure SuperBlocks, user-code blocks and logical expression blocks, the designers can alter the code generation in some manner. In addition, customized blocks could also optimize the generated code for increased execution speed.

Finally, customizations of blocks are a good choice if a block needs to be initialized in the same manner every time it is created. Consider some parameters of a

block such as its icon and the output data type that follows the same convention, each time a new block of this type is dragged and dropped from the Palette Browser to the SystemBuild editor, these parameters need to be set to the same values. Consequently, it is helpful to customize this block with a callback function in order to automate the initialization of any necessary parameters upon the creation of the block. This task will save time and will reduce the amount of repetitive work.

5.2.3 Designed Custom Blocks for FWC Model

In the following sections, all blocks that needed customization in order to implement the selected FWC model in the SystemBuild environment are discussed in more detail.

5.2.3.1 Logical Blocks

Logical blocks were customized using logical expression blocks. Figure 5.9 shows all of the logical blocks that were needed for the selected FWC model. Even though SystemBuild comes with predefined logical blocks, there are three main reasons to use customized logical blocks.

The first reason is to minimize the size of the code as was explained in the previous section (how customized blocks affect the size of generated code). The second reason is for initialization of some parameters such as block name, output name, and output label, so that all logical blocks follow the same naming convention automatically upon creation of the block. For example, for an AND gate the name of the block, output name and label would be b_and_2, in which b stands for Boolean, "and" for AND gate, and 2 is the block ID.

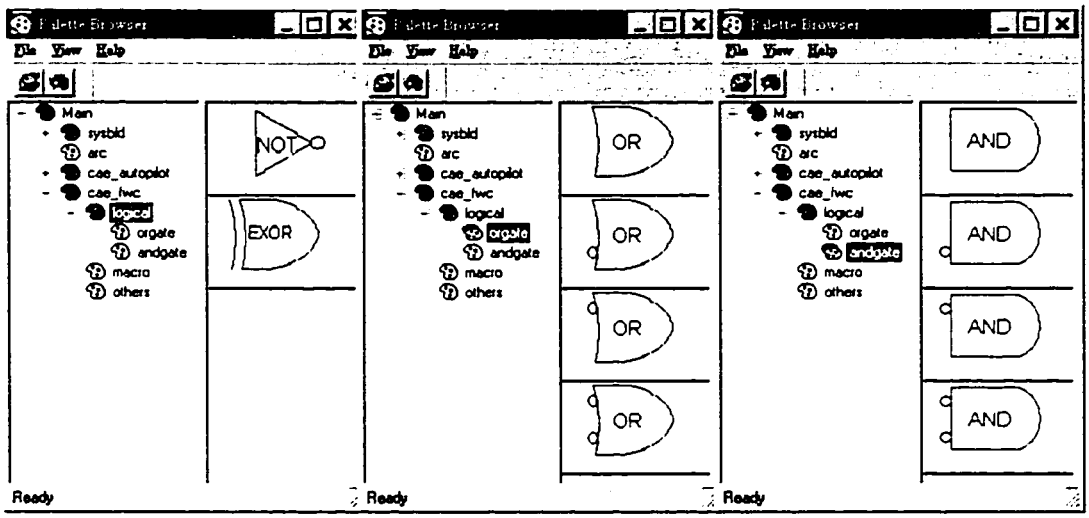


Figure 5.9 Customized logical blocks.

Finally, the icon of the block has been created in such a way that it resembles the blocks as they appear on the design document. The circle in front of any input of the block means the negation of that input prior to the operation of the block. This will prevent any confusion when comparing the original design documents with the model build in the SystemBuild editor.

5.2.3.2 Custom Macro SuperBlocks

AutoCode generates a macro call for a macro procedure SuperBlock. Therefore, choosing this SuperBlock is beneficial for use in a model where the code for a block representing a macro call has already been developed. By customizing these blocks, not only the already defined macros will be reused, but also the generated code for these blocks will be exactly the same as the manually written code, and this is the objective of this part of the project. Each custom block contains a callback function to automate the initialization of the parameters, icon, and the macro code. On the other hand, in order to

simulate the model containing macro procedure SuperBlocks in the SystemBuild environment, the designer should build a block that reflects the nature of the macro inside the macro SuperBlock. This can be done using BlockScript blocks or user-code blocks. Figure 5.10 shows the macros that were developed during this project.

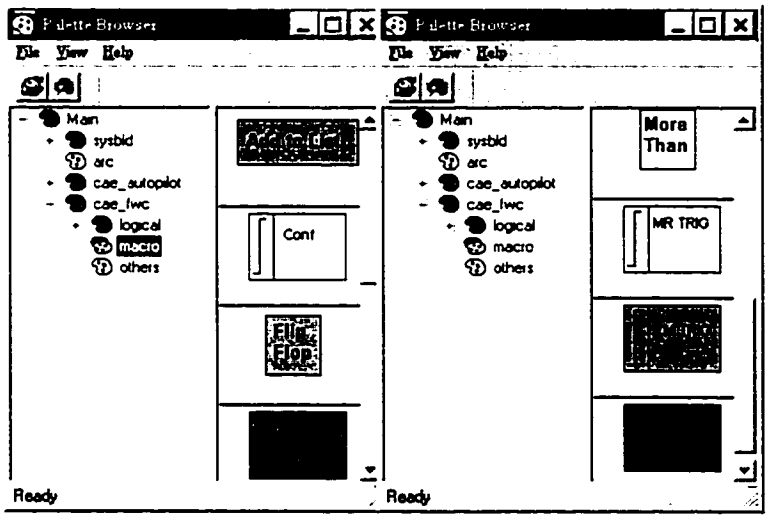


Figure 5.10 Customized macro procedure SuperBlocks.

Each custom block also contains a graphical user interface (GUI). The GUI is helpful for entry of some variables that are used for various parameters. The callback function takes the values from the GUI and sets the block parameters to their appropriate values. This method decreases the designer's workload by automatically performing many tasks at the same time. Consider the example of dragging and dropping the confirmation macro from the Palette Browser to the SystemBuild editor. Immediately after dropping the block in the editor, the confirmation GUI (as shown in figure 5.11) appears.

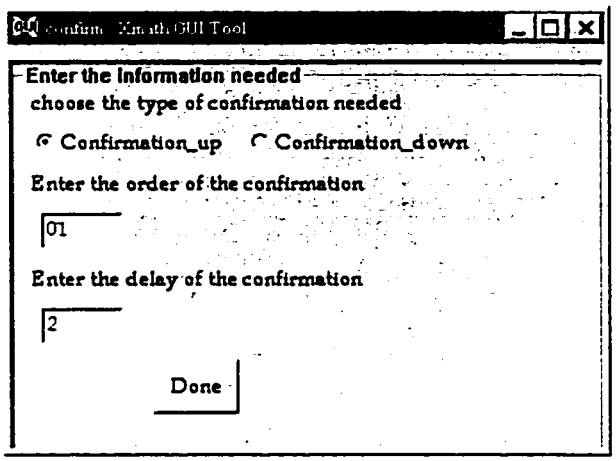


Figure 5.11 Confirmation GUI.

The GUI contains a toggle button to select, either the confirmation up or confirmation down macro, and two text fields to enter the order and the delay of the confirmation. Figure 5.12 shows the icon of the block created for both cases with the same value of confirmation order and delay. It also shows the code created for confirmation up.

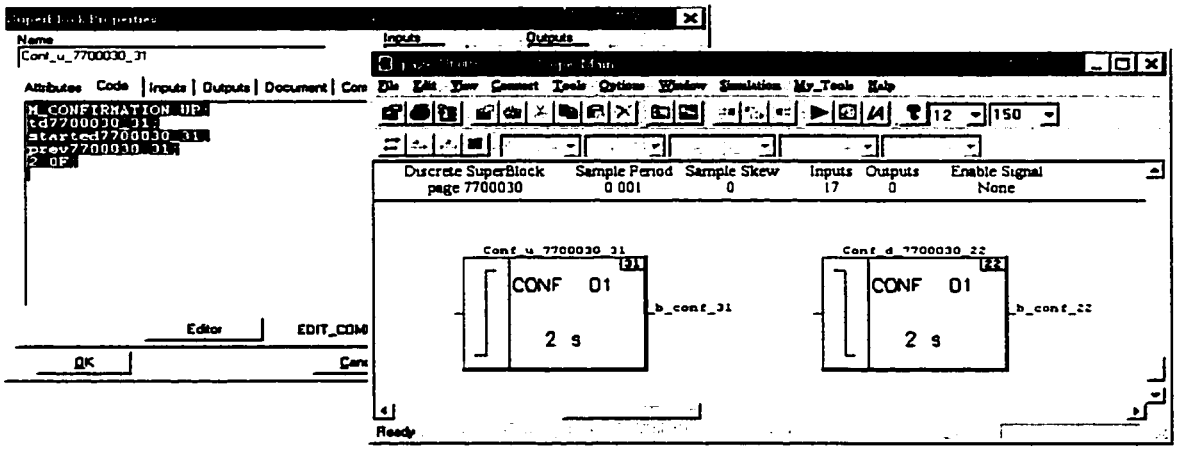


Figure 5.12 Sample confirmation SuperBlock.

Observe that the icon for each block is different and the values of the confirmation delay and order are shown on the icon as well. The reader should pay attention to the name and the output label of the blocks that are created automatically. In the name Conf_u_7700030_31, conf stands for confirmation, u for up, 7700030 has been extracted from the parent SuperBlock name seen in the SuperBlock property field in the editor.

If one looks inside the confirmation macros, he should verify that a BlockScript block has been automatically created in the macro SuperBlock. In addition, by looking at the code of the BlockScript block, the reflection of the confirmation delay in the code will be apparent. This is shown in figure 5.13.

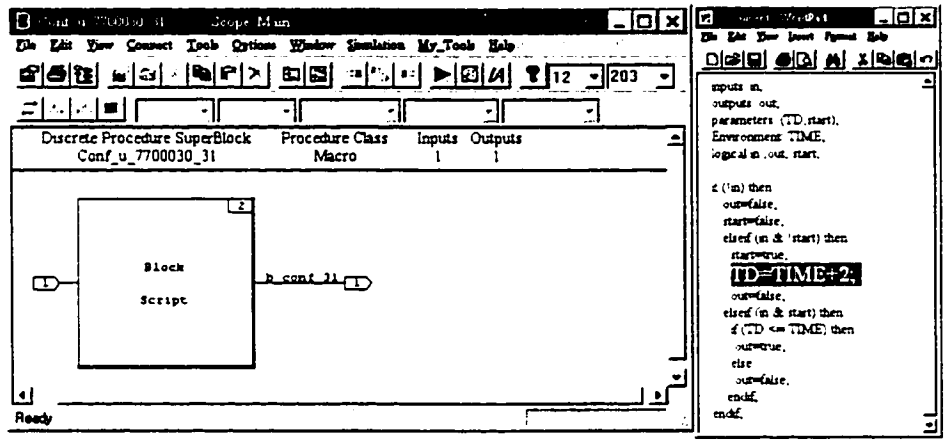


Figure 5.13 BlockScript block representing confirmation macro.

Another important issue that should be mentioned, is the automatically generated code that is highlighted in figure 5.12. The first line is the name of the macro, and the rest are the parameters of the macro that will be reflected in the generated code as shown in figure 5.14.

```
/* {Conf_u_7700030_31.31} */
M_CONFIRMATION_UP(td7700030_31,started7700030_31,
prev7700030_31,2.0F,input,b_conf_31);
```

Figure 5.14 AutoCode generated code for sample macro procedure SuperBlock.

Let's summarize the process to see how much work has been done using a custom block with a callback function combined with a GUI. The following figure shows all the programs that have been entered automatically in the macro SuperBlock and BlockScript block properties. Note also that the value of the confirmation delay entered in the GUI has been used in the icon, BlockScript block code, and in the macro SuperBlock code. Clearly, entering this information for each SuperBlock of this type would require much time and repetitive work which has been eliminated by using these customized blocks.

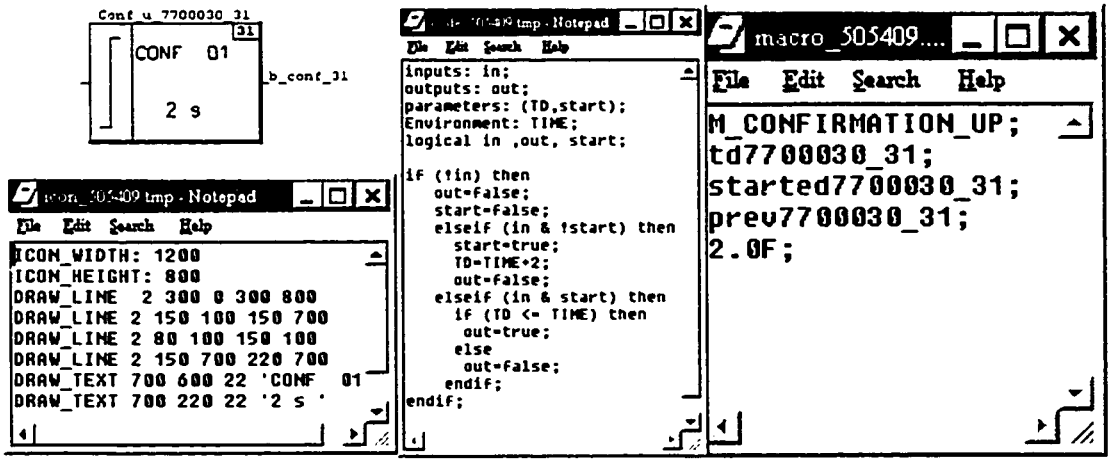


Figure 5.15 The codes and icon generated automatically for the confirmation macro.

Another important customized macro SuperBlock is the add-to-the-fault-list. Each logic page of the engine system consists of a message list which should be sent to the display units if the engagement logic is set to true. The function of the add-to-fault-list is to add the fault messages to the correct list. Each macro is set to true if the logic is engaged and false otherwise. Thus, if each page contains several add-to-fault-list macros, this block should be dragged and dropped twice to messages that exist in the page. This is a very cumbersome job because more than a hundred logic pages exist for just the engine model, and each page consists of several memos. To solve the problem, this block was customized using a callback function and GUI; shown in figure 5.16. As soon as the block is dropped into the editor, a dialog box pops up and asks for the number of messages. A GUI will be built based on that number and the user is asked to enter the messages in the text box provided. Consequently, this block needs to be created only once for each logic page. Observe that with this solution, an inline SuperBlock containing a macro procedure SuperBlock for each message with the appropriate name and icon has been created.

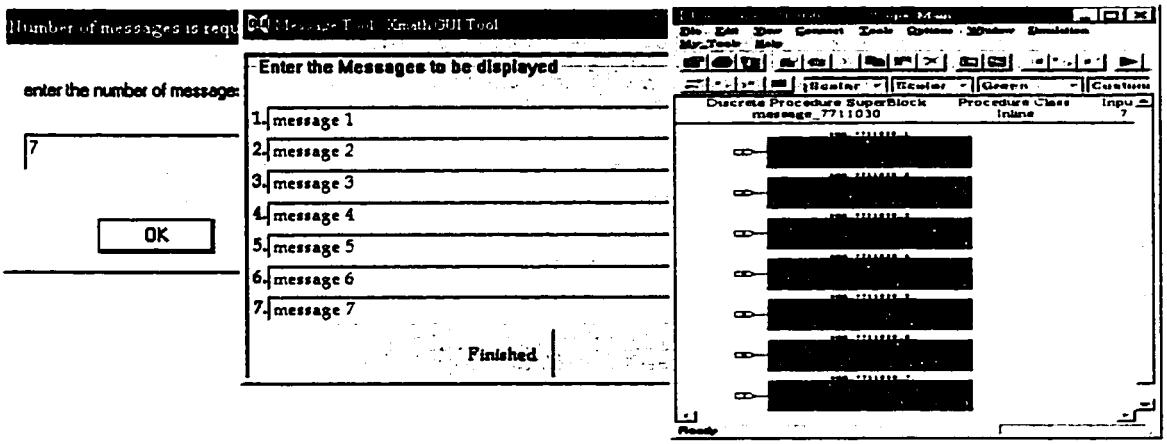


Figure 5.16 Operation of customized add_to_fault_list macro.

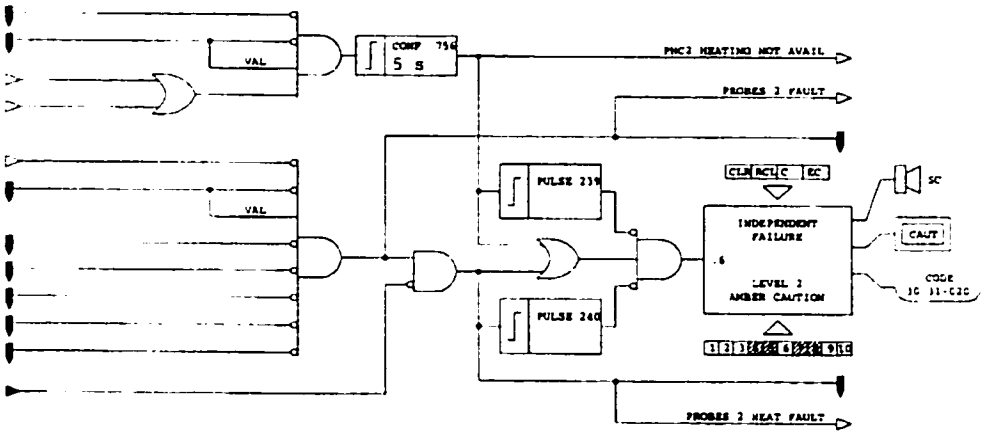
On the other hand, sometimes the messages for consecutive logic pages are almost identical, therefore, messages entered in the GUI are saved for the next time the block is created. This will further simplify the design of the model in SystemBuild. It is necessary to mention that this block is created only for purpose of code generation and it is not intended to be operative in the SystemBuild simulation environment. However, if one wishes to simulate the model in SystemBuild, a block that performs the function of the add-to-fault-list macro should be inserted into each one of these SuperBlocks.

5.2.4 Implementation of Flight Warning Computer in SystemBuild

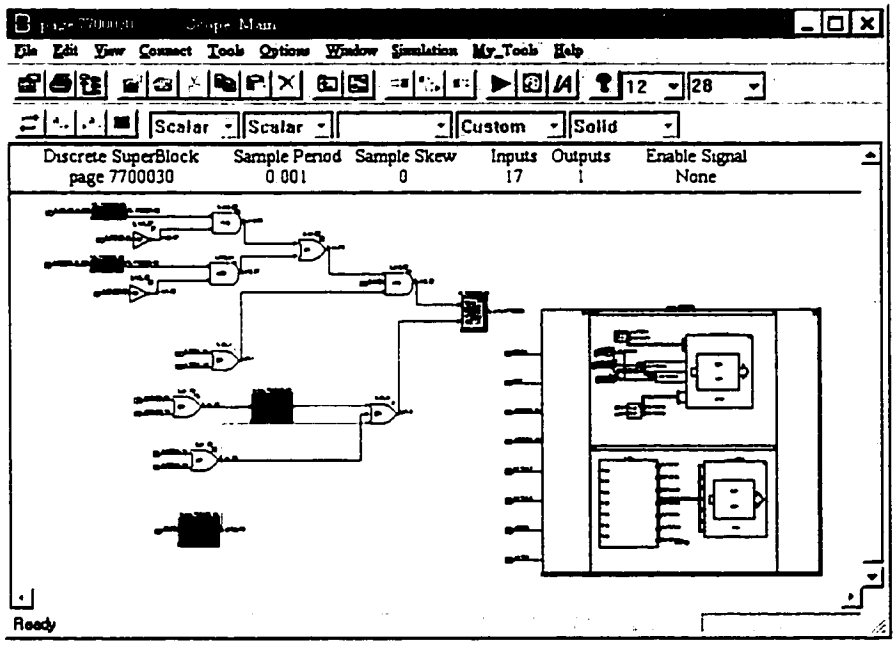
In order to understand the implementation of the FWC in the SystemBuild environment, it is helpful to take closer look at a sample logic page. Figure 5.17a shows a typical logic page for FWC. For purposes of comparison, the implemented logic page is shown in figure 5.17b. The input entries to this page come from various aircraft systems such as the air data computer (ADC) or system data acquisition concentrator (SDAC). Based on these inputs the appropriate messages, if applicable, are sent to the display units. If the output to the biggest box in the figure is true, this page's messages will be added to the list of messages that should be displayed on the display units. However, some restrictions such as flight phase inhibition for these messages may apply.

SystemBuild supports both top-down and bottom-up design approaches, from which the bottom-up approach was chosen for this project. In this approach, each logic page has been designed separately as a SuperBlock. Then, all designed SuperBlocks are grouped in a top level SuperBlock to represent existing messages for an aircraft system such as an engine system. In other words, for each system there are collections of logic pages covering all of the messages for the system. Each of the aircraft system messages

have been represented as a top level SuperBlock with all pages as child SuperBlocks as shown in figure 4.18.



(a) Design document [37]



(b) MATRIX$_X$ model

Figure 5.17 Sample warning page and MATRIX$_X$ model.

Consider the inline SuperBlock, the biggest block in figure 5.17b. This SuperBlock contains an if-else-then construct block that takes care of the messages that should be sent to the display units if the output of the flip-flop block is true, otherwise there is no message to be displayed. Moreover, all of the messages are grouped in the if-else-then block as an inline SuperBlock. This organization of blocks enables the AutoCode to generate very easy to follow codes similar to manual codes.

A problem that one encounters in a bottom-up approach is that SystemBuild supports only automatic flow of input labels from the higher SuperBlock hierarchy to lower SuperBlocks. In this approach, the input should be entered for parent SuperBlock as one goes from lower to higher in the SuperBlock hierarchy. In addition, all parent SuperBlock inputs should be connected to their appropriate input entries of the child SuperBlocks. This would be very cumbersome work if the designer does it manually. A utility called "ini", which is presented in appendix 3, was developed to facilitate the designer work. This utility automatically extracts all input labels of all children SuperBlocks, sorts them in alphabetical order, deletes the inputs that fill multiple lines in the name list, modifies the parent SuperBlock input label and name list, and connects all the input entries from the parent SuperBlock to the child SuperBlock. This utility also performs another task that will be explained in the next section. The operation of this utility is shown figure 5.18.

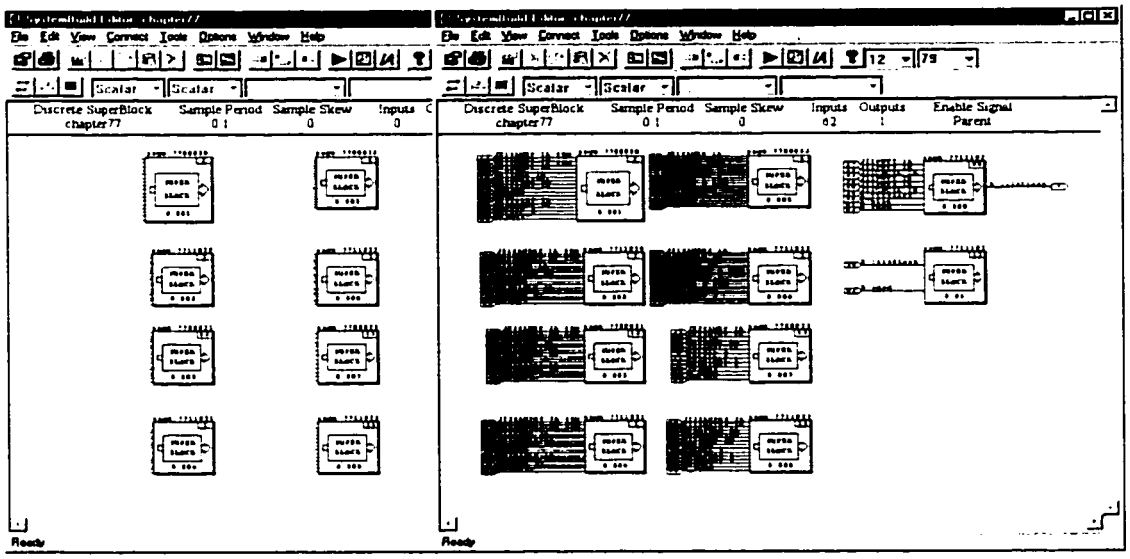


Figure 5.18 Operation of "ini" utility.

5.2.5 Code Generation

Once the model has been implemented and validated in the SystemBuild environment, the corresponding real-time C code can be generated using AutoCode. Template programming language (TPL) is used to customize the generated code. AutoCode comes with a predefined template program that is used as default to generate C code. Users may modify the template program accordingly to satisfy the requirement of their design. Designers may also rewrite a template program according to their needs and this is the approach chosen to generate the C code for FWC model. This approach may not be the best solution but it should be remembered that there was a time limit for this project, and most importantly, this approach was chosen based on experience gained during that short time.

5.2.5.1 Using Default Template Program

SystemBuild analyzer flattens the entire model prior to code generation such that all the blocks with the same timing attributes are grouped together in a subsystem. The analyzer also controls the order of execution for blocks contained in each subsystem. After the analyzer completes its job, AutoCode generates a real-time code based on the execution order that has been set by the analyzer. The generated code consists of various components that are shown in figure 5.19 [38].

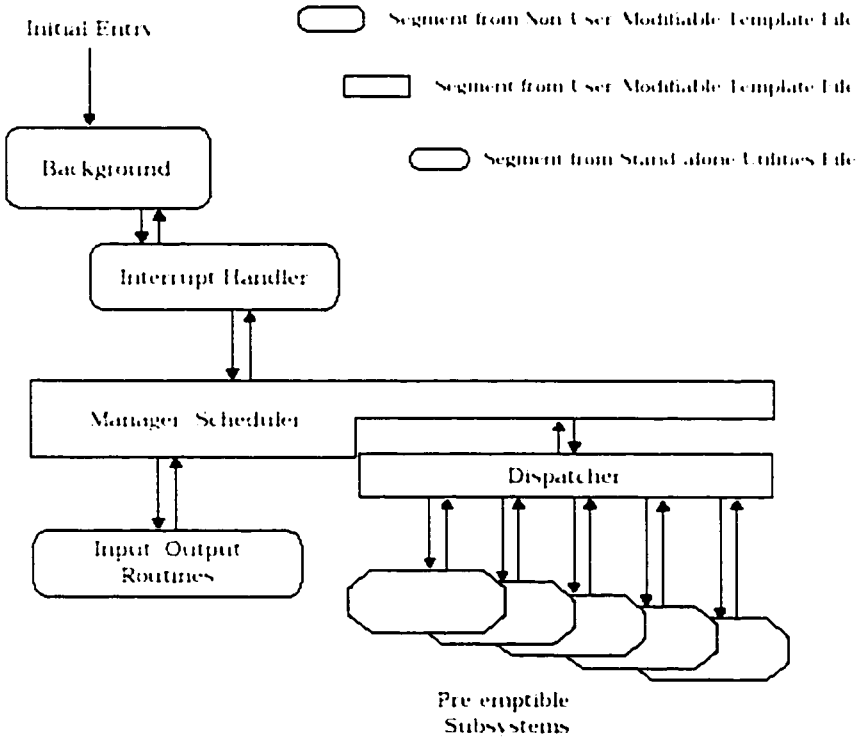


Figure 5.19 The components of generated code using default template program [39].

The scheduler is a time-critical routine and its main task is to take care of the execution of subsystems, data flow between the subsystems, and to perform functions such as handling input/output and interrupts routines. The dispatcher handles the

dispatch of each subsystem from the list created by scheduler according to the execution priority of all subsystems starting from the highest priority [39].

By using the template programming language, designers are able to modify or rewrite these routines except for the routine set by the analyzer that handles the execution order of the blocks inside each subsystem. This means that by using the template program, the user can neither change the order in which the blocks are executed, nor the algorithm used for coding each block inside a subsystem. However, the designer may use a sequencer block to change the execution order of the blocks in the SystemBuild environment. The sequencer performs its task simply by forcing the executions of the blocks on its left side to take place before the blocks on its right side. In the case that the sequencer block is used, care should be taken when dealing with read and write variable blocks, if-then-else block, and in some cases, standard procedure SuperBlocks to make sure that the order of execution of the blocks satisfies the expectations. In addition, the later restriction can be resolved using BlockScript block, and can be used to develop user defined algorithms using the BlockScript language. Note that these restrictions are resolved in the SystemBuild environment prior to code generation from the SystemBuild model and not using the template programming language [40].

5.2.5.2 Using User Defined Template Program

In the early phase of this project, it was believed that it would be difficult to adapt the AutoCode generated scheduler, the most critical part of the code, to the CAE environment. Therefore, due to the short time available for the project, the main effort made was to find out whether it was possible to generate a code such that the automatically generated code would be similar to the manually written code. Doing this

would not only eliminate the need for AutoCode hard to trace scheduler, but would also allow for the reuse of already existing routines such as the CAE dispatcher and macros. In the following paragraphs, the required code structure is first briefly explained, and then each problem encountered to satisfy this requirement and its respective solution is explained in more detail.

In order to understand the operation of the user defined template program the reader should recall that each FWC system is represented as a top level SuperBlock and each of its warning logic pages are designed as a child SuperBlock. The user defined template program translates this SuperBlock hierarchy into the generated code in exactly the same order as it appears in the SystemBuild model. The overall structure of the desired generated code is as follows. The top level SuperBlock has been coded as a module that handles the computation of the FWC warning messages. All necessary external and internal variables to this module have been declared in the beginning of the program. Then, each logic page has been translated into an equation in the module with declaration of all the variables used only in this equation. In addition, a comment according to the naming convention of each warning logic page is generated for each equation, and this simplifies the identification of each equation in the generated code. Consequently, in this manner the generated code is well-organized, well-commented, and easy to trace. For a comparison of the manual code versus the automatically generated code, refer to appendix 1.

In designing the FWC model, only SuperBlocks with same timing-attributes, macro procedure SuperBlocks, and inline procedure SuperBlocks have been used. Therefore, these SuperBlocks and their contents will be categorized as one subsystem

with the block execution order set by the analyzer. On the other hand, recall that template programming language does not support manipulation of the execution order inside a subsystem. Consequently, it is not guaranteed that the code for all blocks contained in each logic page would fall in the appropriate order inside a separate equation. This problem is solved using two different techniques, namely by using a sequencer block between each child SuperBlock in the model, or by changing the sampling period of each child SuperBlock appropriately prior to code generation. The sequencer block guarantees that the code for all blocks on its left be generated before the code for the blocks on its right. But, consider that for a model with ten children SuperBlocks, nine sequencer blocks would be needed for grouping the block codes for each page into separate equations. As a result, this approach does not create an orderly schematic diagram of the system model. The second technique ensures that each child SuperBlock and its content are coded separately in a subsystem. Note that the sampling rate is changed prior to the code generation using a MathScript utility and it does not effect the simulation of the model in the SystemBuild environment. Regardless of which approach is used, the generated code should be modified using a post-processing utility in order to be functional. The reason that the post-processing utility is needed is explained in more detail later in this section.

Another problem encountered is that the TPL does not provide a token to extract the parameters of the macro procedure SuperBlock, which are entered in the code tab of SuperBlock property into the generated code. These parameters need to be declared as internal variables inside each equation that contains a macro procedure SuperBlock. The solution for this problem is for these variables to be passed from the SystemBuild model

into the generated code using user-parameters. However, user-parameters cannot be vectors, which further complicates passing the parameters to generated code. In addition, it is cumbersome to define a user-parameter for all macro SuperBlock parameters that are used in the model. To simplify this task, the initialization program "ini" includes a routine that automatically extract the parameters of all the macro procedure SuperBlocks from each child SuperBlock and declares them using an appropriate naming convention in the child SuperBlock property. During code generation, these parameters are extracted using the written template program and are coded as internal variables in their proper place.

Finally, as mentioned earlier, AutoCode treats all SuperBlocks with the same timing attributes and their contents as a subsystem. Furthermore, each subsystem and its external inputs/outputs are coded automatically as a structure in the generated code, and for every structure, AutoCode generates a declaration of variables. Unfortunately, the template programming language does not allow the user to eliminate these structures and their variable declaration. Therefore, using math script programming language, a post-processing utility, which is presented in appendix 4, has been written in order to transform the generated code into a code similar to the hand written code by deleting the unwanted code that has been automatically generated.

5.3 Modeling Pitch Channel Autopilot Using MATRIX$_X$

A typical generic autopilot was chosen as the dynamic control system for the purpose of the MATRIX$_X$ evaluation. Due to confidentiality issues, the design documents (including label definitions, block descriptions, operation of autopilot modes, etc.) were not provided. Additionally, in the provided design documents some of the

engagement logic blocks were not meaningful, and some connections of blocks were missing. For example, there were "AND" gates with one or no input at all. There were no indications of where the inputs to specific engagement logic came from, or where the outputs were supposed to be connected to. Therefore, it was impossible to develop a model to represent the complete autopilot. Even if one could manage to implement the entire given model in the SystemBuild, there would be no guarantee that the model would work because of the missing connections and necessary documents in the system model. On the other hand, during this part of the project, the license for AutoCode was not provided, and this made it impossible to generate code for the autopilot model designed in the SystemBuild environment. Nevertheless, care was taken during the modeling of the autopilot in the SystemBuild such that a code should be automatically generated for the model if the license for AutoCode were available.

In spite of the facts mentioned above, only the pitch axes autopilot was chosen to evaluate the use of $MATRIX_X$. Although only part of the autopilot was modeled in the SystemBuild environment, this information was sufficient for the consideration of $MATRIX_X$ evaluation for dynamic control systems because the roll and yaw channel autopilots use essentially the same blocks as the pitch channel autopilot in their design. As a result, if the pitch channel autopilot was successfully implemented and validated in the SystemBuild environment, then the roll and yaw axis autopilots would also be implemented in the same manner. In the following sections, the custom blocks developed for this model are explained and then the design consideration to implement the control loops in the SystemBuild is discussed in more detail. Finally, the reader will have an understanding of the approach used to fly the aircraft in pitch axis and will also

observe the overall schematic diagram of the aircraft model implemented in the SystemBuild environment.

5.3.1 Design of the Blocks Used in Autopilot Model

The SystemBuild Palette Browser provides various dynamic blocks such as an integrator and time delay block. The parameters and icons of these blocks need to be set with the creation of each block in the SystemBuild editor. Moreover, recall that users cannot alter the AutoCode to generate user-defined algorithms for these blocks in the generated code. Therefore, it is practical to design these blocks as BlockScript custom blocks; first to automate the initialization of desired parameters such as block name and icon, and second for designing a desired routine to be reflected in the generated code. The dynamic blocks provided in SystemBuild are designed such that upon creation, if the parent SuperBlock is continuous the block will be created in continuous form, and if the parent SuperBlock is discrete the block will be in discrete form. Considering this is not necessary when designing the blocks for the autopilot model due to fact that the autopilot will be modeled in discrete form. In the next section, the methods used to transform continuous systems into discrete system are explained. Afterwards, the custom blocks created for the autopilot system are presented.

5.3.1.1 Transformation from Continuous to Discrete Domain

In autopilot schematic documents, filters are presented in the S domain namely continuous time domain. The difference between continuous and discrete time presentation of these blocks is that in continuous time, these blocks are presented as a differential equation, while in discrete time they are in forms of difference equations.

There are various techniques to transform continuous time filters into the discrete time domain. Even though there are optimal techniques available for this transformation, for the purposes of evaluation in this thesis, the filters are transformed from continuous time to discrete time using approximation techniques such as the Tustin or Trapezoidal method, Forward Euler method, and Backward Euler method [41]. This transformation is performed by substituting the S domain variable s into its equivalent form in the discrete domain as shown below:

$$\text{Forward Euler method} \qquad s \rightarrow \frac{z-1}{T_s} \tag{5.1}$$

$$\text{Backward Euler method} \qquad s \rightarrow \frac{z-1}{T_s z} \tag{5.2}$$

$$\text{Tustin method} \qquad s \rightarrow \frac{2(z-1)}{T_s(z+1)} \tag{5.3}$$

The transformation from continuous to discrete time domain is demonstrated with an example using the Backward Euler method. Consider a lag filter in the S domain, which is of the form:

$$\frac{Y(S)}{X(S)} = \frac{1}{\tau s + 1} \tag{5.4}$$

by substituting $\frac{z-1}{T_s z}$ for s and some algebra the lag filter becomes,

$$\frac{Y(Z)}{X(Z)} = \frac{1}{\tau\left(\frac{z-1}{T_s z}\right)+1} = \frac{T_s z}{\tau(z-1)+T_s z} = \frac{T_s}{\tau\left(1-z^{-1}\right)+T_s} \tag{5.5}$$

taking the inverse z transform from the above equation gives,

$$(\tau + T_s)y(n) - \tau y(n-1) = T_s x(n) \tag{5.6}$$

rearranging the terms and some algebra yields:

$$y(n) = \frac{1}{\tau + T_s}(\tau y(n-1) + T_s x(n)) \tag{5.7}$$

$$y(n) = \frac{1}{\tau + T_s}(\tau y(n-1) + T_s y(n-1) - T_s y(n-1) + T_s x(n)) \tag{5.8}$$

$$y(n) = y(n-1) + \frac{T_s}{\tau + T_s}(x(n) - y(n-1)) \tag{5.9}$$

The above equation indicates that the output of the lag filter at any time depends on the input to the filter at that time, the output from the previous cycle, and the sampling rate of the system. In addition, this equation is used to implement the lag filter using BlockScript language in a customized BlockScript block.

5.3.1.2 Customized BlockScript Blocks for Autopilot Model

Dynamic blocks have been customized in order to simplify the task of the designer, speed up the design process, and control the generated code for these blocks. Each block consists of a callback function with a GUI. Users are asked to choose the algorithm to be used for implementation of the block in discrete form, and to enter information such as the gain of the block. The information entered in the GUI is then

used to create the code for the BlockScript block, the icon of the block, and the initialization of variables that were provided. The customized blocks that were developed for this model are shown in the following figure 5.20. For the purpose of demonstration, the callback function for the lag filter is provided in appendix 5.

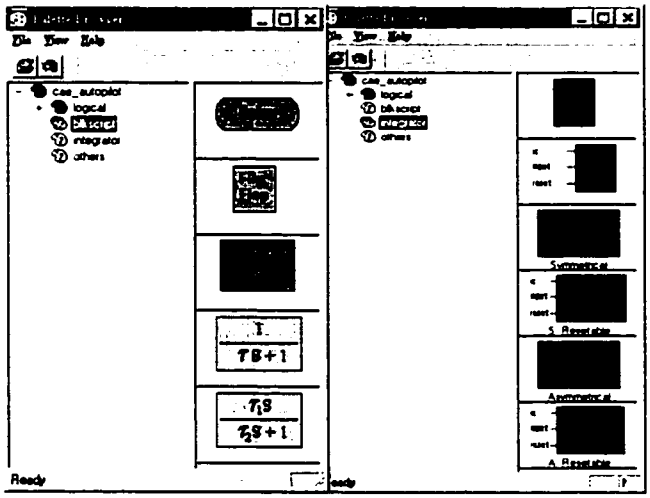


Figure 5.20 Customized BlockScript blocks.

For example, consider what happens when creating a lag filter. As soon as the block is dropped in the SystemBuild editor, a GUI (figure 5.21) pops up that allows the users to choose the type of transformation, as well as enter the value of the filter time constant. From the figure, it is also clear that the value of τ entered in the GUI is not only used in the generated code, but also appears in the icon of the block. The reader should also verify that equation 5.9, derived in the previous section using the Backward Euler approximation, is implemented in the generated code.

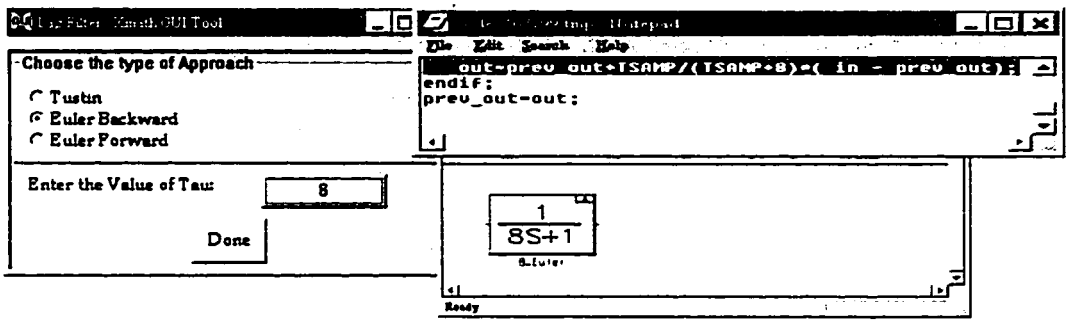


Figure 5.21 Customized lag filter.

5.3.2 Implementation of Autopilot system in SystemBuild

The pitch axis autopilot module simulates the necessary computation to generate an elevator command in order to control the aircraft along the pitch axis. It receives inputs such as mode of operation, appropriate gains for current mode control law, altitude, airspeed, and all other necessary information needed from the various aircraft modules to control the aircraft in the currently engaged mode of operation. The simplified pitch module is simulated in its proper order as shown in figure 5.22.

Each step in the figure has been represented as a SuperBlock in the SystemBuild environment, which is demonstrated in figure 5.23. Additionally, in order to assure that each SuperBlock is executed in the desired order, a sequencer block has been placed between each SuperBlock. The sequencer block guarantees that the outputs of the complementary filter are computed before they are sent to the control law computation blocks, and output of control law blocks are computed before they are used by the inner loop blocks.

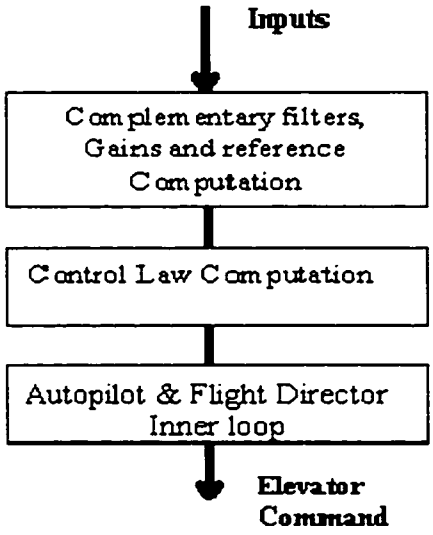


Figure 5.22 Pitch channel autopilot flowchart.

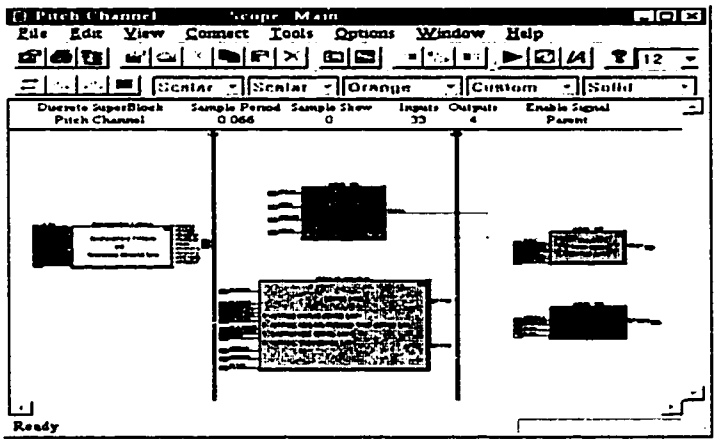


Figure 5.23 Pitch channel autopilot implemented in SystemBuild.

The largest block in the above figure represents the control law computation for the autopilot system. Depending on which mode of operation is engaged, the

corresponding control law should be simulated. To perform this task, two possible methods were evaluated.

The first possible implementation was to use condition SuperBlock. In this method, each control law is considered as either a macro or a standard procedure SuperBlock due to the restriction that only these two Superblocks can be nested inside the condition SuperBlock. AutoCode generates a subsystem for each SuperBlock of the standard procedure class and this coding structure is useful if a function is reused in a model repeatedly, but, not for the autopilot model because each control law model is simulated only once in each execution cycle. Therefore, passing variables from the main code to the subsystem will introduce an overhead in the execution time. On the other hand, the macro procedure SuperBlock will be hard coded as a macro call in the generated code. However, the macro procedures will eliminate the overhead of a calling function if the macro is small. Otherwise, the macro will not solve the overhead of the calling function. In any case, the macro procedure Superblocks are not an appropriate choice due to the fact that a macro has normally been used to represent the execution of a single block and not a dynamic control model such as control law computation. Consequently, condition block currently is not the obvious choice to model the autopilot system in the SystemBuild environment. However, the condition block could be considered useful if it supports inline procedure SuperBlock, in the future release of the $MATRIX_X$ family.

The second block that could be used to model the control law computation is an if-else-then construct block. This block supports standard, macro, and most importantly, inline procedure SuperBlock. As explained previously, the macro and standard

procedure Superblocks were not used for the condition block in the autopilot model implementation. Fortunately, AutoCode does not generate structures for this block and in contrast to a condition block, inline procedures are supported within this block, therefore, the generated code would be as expected.

Regardless of which approach is used, the main problem encountered in designing the autopilot model was due to algebraic loops not supported by AutoCode. An algebraic loop occurs in a model when an input to a block depends on its output in a loop within the same execution cycle of simulation. In this case, the simulator is not able to clearly identify which block in the loop should be computed first. Using a default integration algorithm, the simulator adds a delay block to the output of one of the blocks in the loop such that the input or output to the block causing the problem will be delayed by one execution cycle. By doing this, the simulator is able to set the execution order for the blocks that are within the loop. However, in some cases, this will have unexpected results on the output of the simulation.

Algebraic loop could be eliminated by using two techniques. The first technique is for the user to use a delay block in the loop similar in method to that of the simulator; the difference being that the user chooses where to place the delay block in the loop for satisfactory result. Even though, this method solves the problem of algebraic loops, it is not suited for use in all applications due to the delay it introduces into the execution of the system. The second technique is to use a pair of variable read and write blocks instead of a delay block in the model wherever an algebraic loop exists. In order to understand how this method solves the problem it is helpful to recall that the analyzer sets the execution order in each subsystem prior to simulation. The read from variable

block has the highest priority in the execution order and the last block executed is the write to variable block. Thus, using read and write blocks in a loop will not only assure that the analyzer is able to set an execution order in a loop without confusion, but it also ensures that there will be no delay in the execution process of the system. However, if read and write are used to eliminate algebraic loop, then the read from variable block should be initialized appropriately to guarantee that the values in the loop that pass through the read and write block are as expected within the same execution cycle time. Finally, the initialization of the read block variables can be done using either startup procedure SuperBlock, executed only once in the beginning of the simulation, or by using a mathscript script that is executed prior to the simulation of the system.

5.3.3 Simulation of Autopilot System

In order to simulate the autopilot system, it is necessary to receive inputs from various aircraft sensors and systems to maintain the desired attitude of the aircraft by the appropriate control surface movement to oppose the forces and moments that push the aircraft away from its current stable position. In practice, the autopilot system is coupled to the aircraft model, and by using aircraft flight test data and linear interpolation tables provided for the coefficients of aerodynamics forces and moments, the forces and moments acting on the aircraft are computed. Unfortunately, neither the flight test data nor the complete aircraft model were provided for this project due to confidentiality issues. Moreover, complete flight test data is hard to find elsewhere and if found, the data is not complete, roughly rounded, and is provided only for a specific flight condition. Consequently, the only choice to simulate the autopilot system was to develop an approximate mathematical model to calculate the aerodynamic forces and moments,

using data found for the coefficients of these forces and moments for given flight condition. Even though this approach is not used in practice, for the purpose of this thesis it was sufficient in order to simulate the autopilot system in the SystemBuild environment. In the following sections, the necessary mathematical models used to simulate the autopilot system are explained in more detail.

5.3.3.1 Axis Systems

Prior to derivation of the equation of motion and the aerodynamic forces and moments, it is necessary to define a clear axis system in which the motion of aircraft, and forces and moments acting on it are calculated. Figure 5.24 shows the inertial and body axis system. The body axis system which is used for motion calculation is fixed to the aircraft body with its origin attached to the center of gravity; X axis forward to the nose of the aircraft, Y axis along the right wing, and Z axis downward. The inertial axis is fixed to the earth with its positive direction of X axis pointing to the north, Y axis pointing to the east, and Z pointing to the center of the earth, and it is used for calculation of acceleration forces, aircraft position and orientation with respect to earth.

Normally the aircraft velocity in space does not coincide with any axis in body axis systems. The body axis velocity components can be obtained using angle of attack and angle of side slip (figure 5.25) that are represented by the following equations.

$$\text{Angle of attack}: \qquad \alpha = \tan^{-1}\frac{w}{u} \tag{5.10}$$

$$\text{Angle of sideslip}: \qquad \beta = \sin^{-1}\frac{u}{v} \tag{5.11}$$

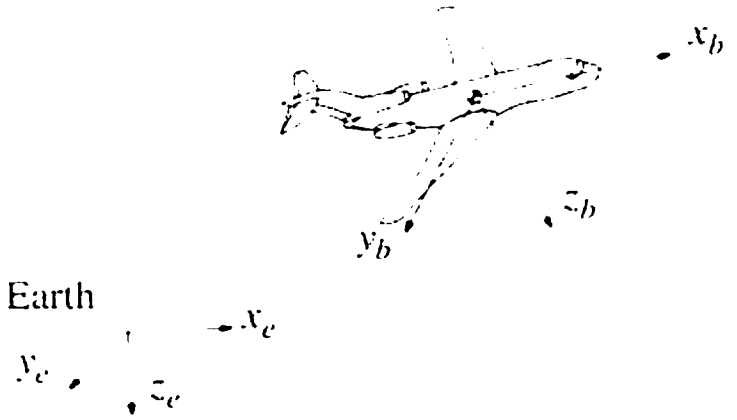


Figure 5.24 Inertial and body axis system [42].

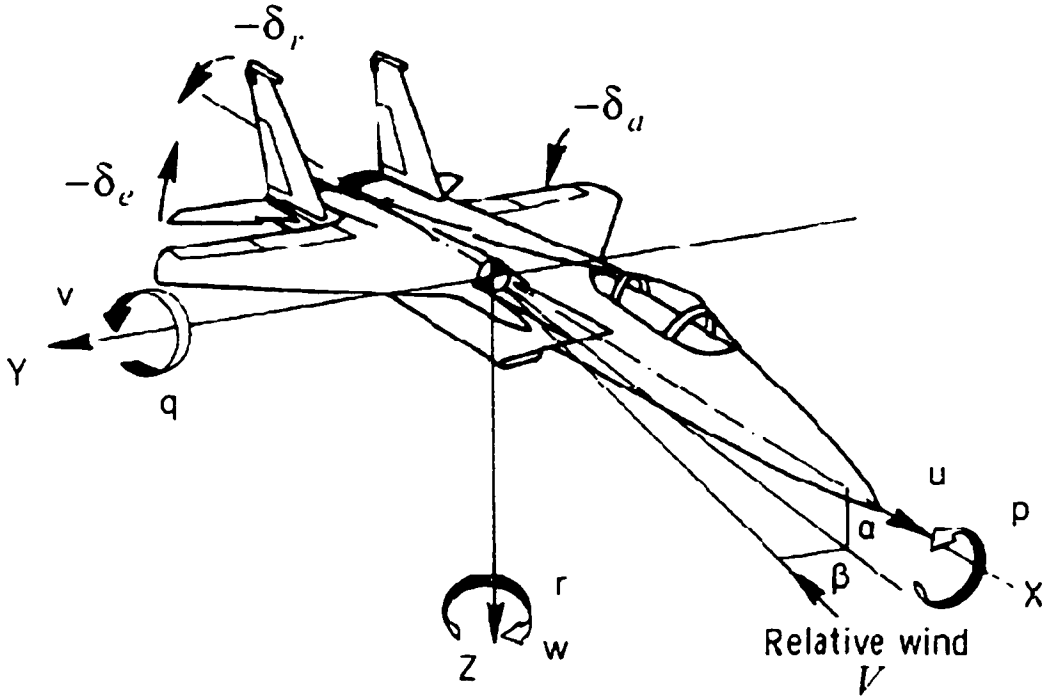


Figure 5.25 Angle of attack and sideslip [42].

Another important axis system is the stability axis system in which the aerodynamic forces and moments are calculated. The stability axis system is obtained from the body axis system by rotating the aircraft along the Y axis by angle of attack [20].

In Figure 5.24, the direction and notation of the aircraft's translational and rotational velocities can be observed. In addition, the signing convention of the control surface movement is also shown. The translational velocities are u, v, and w which represent velocities along the X, Y, and Z axis respectively. The rotational velocities are p, q, and r and represent roll rate, pitch rate, and yaw rate about the X, Y, and Z axis respectively. These notations will be used in the next section to introduce the equation of motion of the aircraft.

5.3.3.2 Equations of Motion

The equations of motion are derived using Newton's second law. Many assumptions have been made to simplify the derivation of these equations. The first assumption is that the aircraft is a rigid body which makes it possible to consider the motion of the aircraft to have six degrees of freedom; three rotational and three transnational motion equations. The second assumption is that the earth is fixed in space to make it an inertial reference point where Newton's second laws are valid. The complete discussion of these assumptions and their effect on driving the equations is beyond the scope of this project. However, these two assumptions form the basis to derive the aircraft's equations of motion [20].

The motion of the aircraft along the X, Y, and Z body axis systems can be obtained from the equations given by: [43]

Forces Along X Axes: $$m\left(\dot{u}-vr+wq\right)=-mg\sin\theta+F_{Ax}+F_{Tx} \quad (5.12)$$

Forces Along Y Axes: $$m\left(\dot{v}+ur-wp\right)=mg\sin\phi\cos\theta+F_{Ay}+F_{Ty} \quad (5.13)$$

Forces Along Z Axes: $$m\left(\dot{w}-uq+vp\right)=mg\cos\phi\cos\theta+F_{Az}+F_{Tz} \quad (5.14)$$

The motion of the aircraft about the X, Y, Z axis is given by: [43]

Rolling moment about X axis: $$I_{xx}\dot{p}-I_{xz}\dot{r}-I_{xz}pq+\left(I_{zz}-I_{yy}\right)rq=L_A+L_T \quad (5.15)$$

Pitching moment about Y axis: $$I_{yy}\dot{q}+\left(I_{xx}-I_{zz}\right)pr+I_{xz}\left(p^2-r^2\right)=M_A+M_T \quad (5.16)$$

Yawing moment about Z axis: $$I_{zz}\dot{r}-I_{xz}\dot{p}+\left(I_{yy}-I_{xx}\right)pq+I_{xz}qr=N_A+N_T \quad (5.17)$$

And the kinematic equations are given by: [43]

Roll rate about X axis: $$p=\dot{\phi}-\dot{\psi}\sin\theta \quad (5.18)$$

Pitch rate about Y axis: $$q=\dot{\theta}\cos\phi+\dot{\psi}\cos\theta\sin\phi \quad (5.19)$$

Yaw rate about X axis: $$r=\dot{\psi}\cos\theta\cos\phi-\dot{\theta}\sin\phi \quad (5.20)$$

5.3.3.3 Aerodynamic Forces and Moments

As was mentioned in chapter three, the equation of motions is divided into longitudinal and lateral equations. Since only the pitch axis autopilot has been implemented in the SystemBuild environment, it is sufficient to implement longitudinal equations of motion to simulate the autopilot system. In addition, note that these forces and moments are calculated in the stability axis system. However, forces and moments in the equations 5.12 to 5.17 are represented in the body axis system. The longitudinal forces and moments can be represented as:

Aerodynamic Force along the X axis: $F_{Ax} = -D$ (5.21)

Aerodynamic Force along the Z axis: $F_{Az} = -L$ (5.22)

Pitch Moment about the Y axis: $M_A = M$ (5.23)

Where,

$D = C_D \bar{q} S$ total drag force (5.24)

$L = C_L \bar{q} S$ total Lift Force (5.25)

$M = C_m \bar{q} S \bar{c}$ total Pitching Moment (5.26)

In the above equations, S is the wing area, $\bar{c}$ is the mean aerodynamic cord, and $\bar{q}$ is the dynamic pressure which is given by:

$$\bar{q} = \frac{1}{2}\rho V_T^2 \tag{5.27}$$

Where,

ρ air density

V_T aircraft true air speed

In equation 5.25, C_L is the total lift coefficient. The following equation represents the lift coefficient considering only the most significant factors that influence it [44].

$$C_L = C_{L0} + C_{L\alpha}\alpha + C_{L\dot{\alpha}}\frac{\dot{\alpha}\bar{c}}{2U_0} + C_{Lq}\frac{q\bar{c}}{2U_0} + C_{L\delta_e}\delta_e \tag{5.28}$$

where,

α angle of attack

q pitch rate

δ_e elevator angle

U_0 reference velocity

C_{L0} the basic lift coefficient

$C_{L\alpha}$ the lift coefficient due to angle of attack

$C_{L\dot{\alpha}}$ the lift coefficient due to rate of change of angle of attack

C_{Lq} the lift coefficient due to pitch rate

$C_{L\delta_e}$ the lift coefficient due to elevator angle

The drag coefficient shown in equation 5.24 can be computed using equation 5.29. Note that in this equation, only the most significant factors have been considered [44].

$$C_D = C_{D0} + C_{D\alpha}\alpha + C_{D\delta_e}\delta_e \tag{5.29}$$

where,

C_{D0} the basic drag coefficient

$C_{D\alpha}$ the drag coefficient due to angle of attack

$C_{D\delta_e}$ the drag coefficient due to elevator angle

The pitching moment coefficient used in equation 5.26 can be computed using the most significant factors affecting it, according to equation 5.30 [44].

$$C_m = C_{m0} + C_{m\alpha}\alpha + C_{m\dot{\alpha}}\frac{\dot{\alpha}\bar{c}}{2U_0} + C_{mq}\frac{q\bar{c}}{2U_0} + C_{m\delta_e}\delta_e \tag{5.30}$$

where

C_{m0} the basic pitching moment coefficient

$C_{m\alpha}$ the pitching moment coefficient due to angle of attack

$C_{m\dot{\alpha}}$ the pitching moment coefficient due to rate of change of angle of attack

C_{mq} the pitching moment coefficient due to pitch rate

$C_{m\delta_e}$ the pitching moment coefficient due to elevator angle

Recall that the aerodynamic forces and moments are calculated in the stability axis system, but in order to calculate the aircraft acceleration and position, these forces and moments should be transformed to the body axis and inertial axis system respectively. The transformation from one axis system to another is performed using the Euler angles shown in figure 5.26.

Euler angles are represented by heading angle (ψ), pitch angle (θ), and bank angle (ϕ). The procedure of transformation is to rotate one axis system by ψ, θ, ϕ respectively to make the first axis system coincide with second axis system. The order in which one axis system is rotated in order to be transformed into another system is very important since different rotation order, would yield different results.

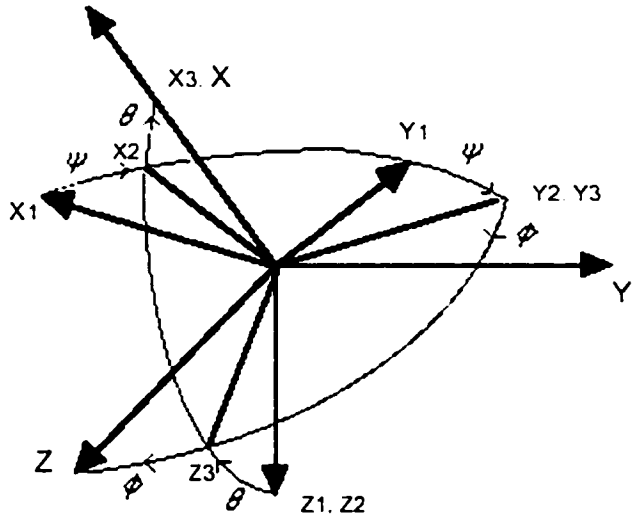


Figure 5.26 Euler angles [45].

Keeping in mind the above procedure and rotating the inertial axis system to coincide with body axis system, the following transformation matrix from body axis system to inertial axis system is obtained [46].

(Yaw Term) (Pitch Term) (Roll Term)

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \tag{5.31}$$

or

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\theta\sin\phi\cos\psi - \sin\psi\cos\phi & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\phi\cos\theta \\ \sin\theta\cos\phi\cos\psi + \sin\psi\sin\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\theta & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} \tag{5.32}$$

The equation 5.32 is used to transform body axis velocities into inertial system velocities from which the aircraft position is obtained.

5.3.3.4 The Atmosphere

International Civil Aviation Organization (ICAO) adapted a standard atmosphere model in 1952. Based on this model the atmosphere is divided into different regions from which only the first two, namely troposphere and stratosphere, are important for the operation of most aircraft. The temperature in these regions depends on altitude and its variation is shown in figure 5.27. Once the altitude is known the temperature, pressure, and density can be obtained for each region in the following manner.

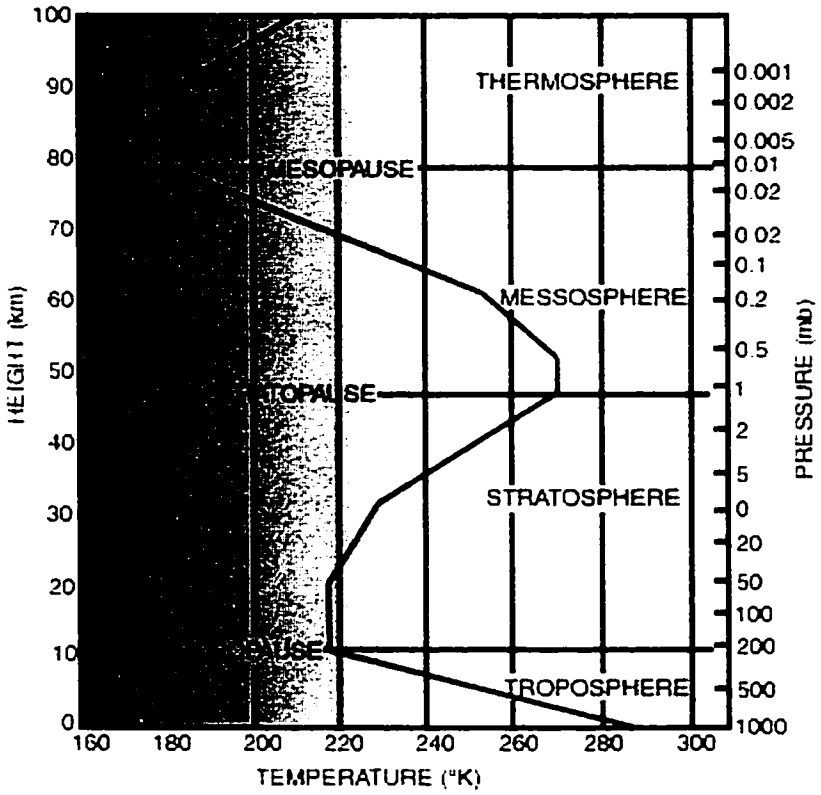


Figure 5.27 Variation of temperature with altitude [47].

In general, the relationship of pressure and density with temperature is given by the equations 5.33 and 5.34 [48][49].

$$\frac{P}{P_1}=\left(\frac{T}{T_1}\right)^{-g_0/(R\lambda)} \tag{5.33}$$

$$\frac{\rho}{\rho_1}=\left(\frac{T}{T_1}\right)^{-(1+g_0/(R\lambda))} \tag{5.34}$$

where,

P	Pressure.
ρ	Density.
T	Temperature.
R	Gas Constant.
λ	Lapse rate.
g_0	Gravity at sea level

The troposphere region is below 36.089 ft (11 km) in a standard atmosphere. In this region the temperature decreases as altitude increases at a rate of 6.875×10^{-6} (altitude in feet). Sea level values are used for the variables shown by subscript 1 in equations 5.33 and 5.34. These values are as follow; [50]

$$T_{SL} = 288.16\,K$$

$$P_{SL} = 2116.2\ lb/ft^2$$

$$\rho_{SL} = 0.0023769\ slug/ft^3$$

$$g_{SL} = 32.17\,ft/s^2$$

Consequently, substituting the sea level values in equations 5.33 and 5.34 yields:

$$T = T_{SL}\left(1 - 6.875 \times 10^{-6}\, h\right) \tag{5.35}$$

$$P = P_{SL}\left(1 - 6.875 \times 10^{-6}\, h\right)^{5.2621} \tag{5.36}$$

$$\rho = \rho_{SL}\left(1 - 6.875 \times 10^{-6}\, h\right)^{4.2621} \tag{5.37}$$

In the stratosphere, a range of up to 82025 ft (25 km) is of interest since most aircrafts operate under this altitude [49]. In this region, the temperature is constant and the reference values are calculated in the boundary of the troposphere and stratosphere region. By using equations 5.33 and 5.34 and substituting the reference values, the stratosphere equations in a standard atmosphere are obtained [50].

$$T = 216.66\,K \tag{5.38}$$

$$P = 471.913 e^{(-0.00004811\ (h-36089))} \tag{5.39}$$

$$\rho = 0.000705939 e^{(-0.00004811\ (h-36089))} \tag{5.40}$$

6 MATRIX$_X$ EVALUATION

The main purpose of this chapter is to discuss the evaluation criteria and evaluation of the results for both the flight warning computer and the autopilot systems. The first part will concentrate on the evaluation criteria used to evaluate MATRIX$_X$ as a software development tool. In the second part, the evaluation of the results for autopilot non-real time simulation in the sytembuild environment, and automatic code generation for flight warning computer system will be demonstrated.

6.1 Evaluation Criteria

The criteria used for the evaluation of MATRIX$_X$ in this chapter are based on work done during this project. First, the evaluation criteria for MATRIX$_X$ software in general are explained. Then, the evaluation criteria regarding the resolved implementation issue will be explored. After that, the unit testing within MATRIX$_X$ is discussed. Finally, the criteria regarding code generation and template programming language will be verified.

6.1.1 General

The general evaluation criteria will cover ease of learning and use of MATRIX$_X$, quality of its online help, the type of system that is best suited for the tool, and finally the recommendations for the version control.

6.1.1.1 Ease of Learning and Use

$MATRIX_X$ is a very comprehensive tool with various programming languages provided for design and implementation of a wide range of systems. It also supports an extensive range of predefined functions and tools. Therefore, to evaluate this criterion, it is necessary to identify by whom and how this product will be used. For users that do not need an advanced understanding of $MATRIX_X$'s various programming tools, this product is easy to learn and use. Couple of weeks of use is sufficient to become familiar with Xmath and SystemBuild environments, and to be able to use the basic features of this software. However, in order to have an advanced understanding of this software and learn most of Xmath and SystemBuild features, time and experience with the software is required. On the other hand, to learn most of its functionality and programming languages, to write scripts, develop graphical user interface, create customized blocks, alter the template programming language or write one, and become accustomed to other programming languages such as LNX, first the user need to have programming skill, second spend time and put lots of effort. In general, for users with some programming skills, most of the programming languages that $MATRIX_X$ offers are easy to learn and use, but time and experience are required to learn its full functionality.

6.1.1.2 Quality of Online Help

$MATRIX_X$ provides online help and an online manual. The online help is in HTML format, well organized, and provides an extensive library of the $MATRIX_X$ product family's features with their syntax and information on how to be used. Although the online help has very user friendly interface, it contains some bugs such as links that do not bring up the exact location of corresponding information, and whenever the help

window is enlarged or decreased in size, the location of the information that is being viewed is lost. Furthermore, in some cases the information provided is not complete and is confusing and at times too advanced for most users. In other cases, the online help refers the user to online documentation for more detail but there is no more information provided in the online documentation than was in the online help and vice versa. In contrast to online help, the online documentation is viewed using acrobat reader. It is very well organized and provides advanced topics for all $MATRIX_X$ features in more detail. It primarily concentrates on design documents and programming languages used by the $MATRIX_X$ product family. In some cases, the online documents lack the necessary detail to provide a good understanding of the topic. In other cases, the topic presented is good for use as a reference of system behavior for advanced users but not for beginners who need to understand how the system works. Altogether, the online help and documentation are useful and handy, but definitely require future enhancement.

6.1.1.3 Types of Systems Best/Least Suited for the Tool

Two types of systems were used to evaluate $MATRIX_X$; a typical generic autopilot system (for representation of a dynamic control system) and a typical medium commercial jet's flight warning computer (for representation of a Boolean system). The results of the evaluation confirm that both systems are well-suited for the tool. However, there are more restrictions when implementing the dynamic systems due to feedback loops which develop algebraic loops in many cases. Nevertheless, $MATRIX_X$ offers an open architecture environment to allow users to implement either of these systems through the customization of blocks, SuperBlocks, template programs, etc.

6.1.1.4 Recommendations for Version Control

Two various methods can be used to introduce revision history into the SystemBuild environment which in turn can be forwarded to the generated code and documentation. The revision history could contain the name of the person who does the modification, the date of modification, and the nature of the problem that has been solved. The first technique is to drop a text box into each SuperBlock to identify the version of the model and history of the revision each time a user modifies any property of the SuperBlock. The second method is to enter the revision history in the SuperBlock property using the user parameter keyword. In any case, a utility can be developed to perform this task and to provide the user with a user friendly interface for adding new comments to the revision history. This tool would first require users to add the new information in the corresponding text fields, and then it would automatically modify the existing revision history according to the information provided by the users. As an alternative, the user may modify the existing history manually by entering the information directly into the text block code tab or by modifying the user parameter in the SuperBlock property. Finally this information could be extracted using a template program and inserted in the generated code or documentation if required.

6.1.2 Implementation

The implementation criteria will be used to evaluate SystemBuild to verify how much time is saved using $MATRIX_X$ software, and how accurate the implemented model in the SystemBuild environment is compared to the existing approach.

6.1.2.1 The Time Required For Coding a Model in SystemBuild

The main effort made to implement a system in sytembuild has been the development of the component library. Each custom block from the library has been designed to automatically perform the initialization of each block. Consequently, this approach will dramatically minimize the time required for coding a model in SystemBuild since most of the job is done automatically prior to the creation of each block. The time required per page on a schematic basis could be anywhere from between one to two minutes depending on how many blocks are used in the page, and how fast the designer is accustomed to working with SystemBuild.

6.1.2.2 The Accuracy of the Modeling Using SystemBuild

Modeling a system using SystemBuild is much more accurate than a manual coding of the drawings. A manual drawing generates many mistakes. For example, the manual coding of Airbus 330 created mistakes in 3000 pages of the system schematic. The first reason could be due to the fact that manual coding is difficult to compare to the original schematic diagram. The second reason is that some designers cut and paste while programming two similar pages and forget to change the variables in the currently designed page appropriately. Another reason could be due to a typing mistake when someone tries to finish the job quickly. In addition to the above, manual formatting of the code is time consuming and delays the software development process. In contrast to manual coding, designing a model in SystemBuild is more accurate because it is easier to compare the schematic presentation of the model in the SystemBuild environment with the original schematic system design documents. Moreover, most of the work is performed automatically, and results in less typing mistakes. In addition, there is no need

to cut and paste because it would take about the same time to cut and paste a block as to create a block. Finally, since the code is generated automatically for the model in the predefined format, modeling the system in the SystemBuild environment will eliminate the need for formatting the code currently done by hand in a manually written code.

6.1.3 Unit Testing

The unit testing criteria is used to evaluate $MATRIX_X$ for friendliness of its user interface, execution speed, and the ease of scripting for system simulation. The evaluation of these criteria is done in the next subsections with their relevant comments.

6.1.3.1 Friendliness of User Interface

$MATRIX_X$ provides easy to use and friendly user interfaces for all of its products. Each product will be evaluated separately. The Xmath window consists of two parts: the command line and the result area. The command line allows users to enter predefined commands or executable files and; the result of this action will be displayed in the result log. The result area poses a bothersome problem at times. Normally, when the buffer of applications like the result area is full, the buffer will be emptied from the top of the window and the current result will be displayed at the bottom. This is not the case with the Xmath window. Whenever the result area's buffer becomes full the result will be displayed from the top and sometimes shows arbitrary locations of the result which causes confusion because it is not obvious where the last results are displayed. Therefore, to overcome this problem, users have to clear the result window before executing new programs, and this is unpleasant since the previous results are useful in most of cases. This problem is shown in figure 6.1.

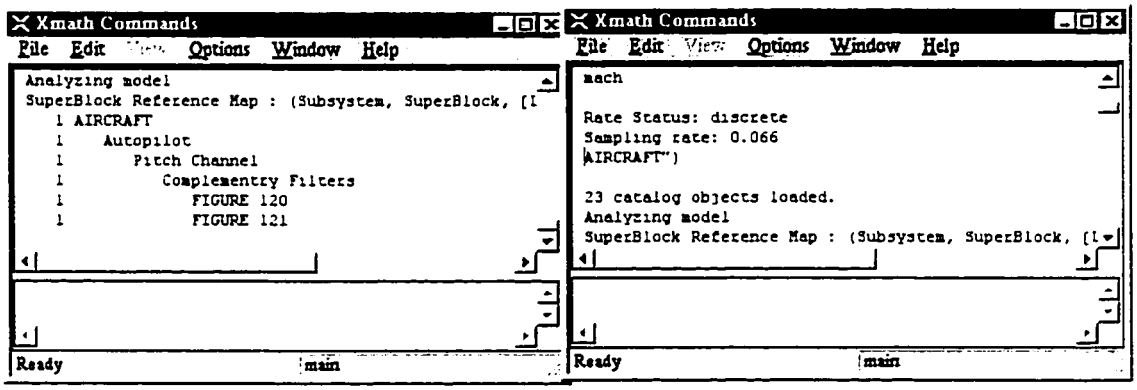


Figure 6.1 Xmath commands window.

The left picture in figure 6.1 illustrates how when the buffer becomes full in the middle of execution of a program, the result area displays the result at the bottom of the window up to the point that the buffer is not full. As soon as the buffer is full, the results start to be displayed from the top of the window, and stop when the execution is finished. From the right picture, it can be observed that the end of the last result, and middle of previous result, are not separated from each other and this makes it difficult to know where the last result has been displayed.

The SystemBuild Catalog Browser also has a user friendly interface with menu bars and a display of the hierarchy structure of the implemented model. However, it has one main drawback; it does not provide an undo function for delete function. This means that if a SuperBlock or SuperBlock hierarchy is deleted accidentally and the model has not been saved prior to deletion, then this SuperBlock and the efforts to build it are gone.

In contrast to the Catalog Browser, the SystemBuild editor provides an undo command, but it undoes all the actions that have been performed since the last time the editor was updated. Therefore, if the editor is not updated frequently, the undo function

will delete all the work that has been done since the last update and once a user presses the undo button, there is no way to go back, the work is gone and there is no redo function available. This is a major problem since sometimes the user needs to undo only the last action, and not all the work that has been done from the last time the editor was updated, or sometimes the undo bottom is pushed by mistake and half an hour or an hour of a job is lost. Furthermore, it does not allow the input and output connections of a block to be on the same side to create a schematic for a block that resembles the original design documents. Finally, it is not possible to line up the external input and output connections to have a clear view of all the external inputs and outputs in the created model.

Regardless of the problems mentioned above, the SystemBuild editor has a user friendly interface and it also allows the user to have two custom menu bars with buttons to perform repetitive actions, eliminating the need to enter the commands in the Xmath command line. This is demonstrated in figure 6.2 that shows the customized pull down menu for tools created in this project.

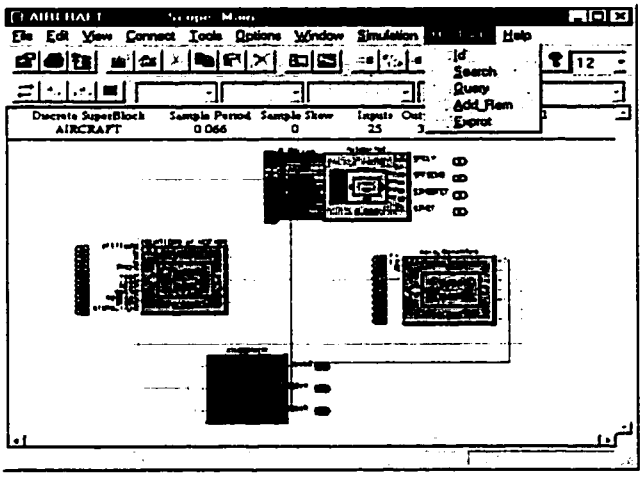


Figure 6.2 SystemBuild editor's customized pull down menu.

The Palette Browser window is very well organized, and displays the categorized group and blocks belonging to each group in same window, allowing the users to select a group on the left side and see the blocks for that category on the right side of the window. It also allows users to organize the customized blocks in any way they wish to place them in the browser.

The interactive simulation window provides a user friendly environment for non-real time simulation of the models built in SystemBuild. It provides a time and block step for debugging purposes. However, the time step only evaluates the next execution cycle outputs, and the block step shows only the next block that is executed. The drawback is that this window does not buffer any previous results which mean it is not possible to look back and forth at two or three consecutive time cycle outputs.

Finally, the size and location of the Xmath window can be saved for next time Xmath is run. This is not the case for other windows, and it would be beneficial if MATRIX$_X$ supports saving the size and location of the Catalog Browser, SystemBuild editor, and the Palette Browser in its future release.

6.1.3.2 Execution Speed

MATRIX$_X$ was evaluated for each aircraft system on a different machine. In the case of the flight warning computer, MATRIX$_X$ was evaluated on a Pentium I class PC, 200 MHz processor, 64 M RAM, and Windows NT operating system. A portion of the engine system warnings model was implemented in SystemBuild and the corresponding code was generated automatically using AutoCode. The time taken to generate the code and perform the post processing of the code using math scripting language was less than ten seconds for this model. The autopilot system was simulated on a Pentium II class PC,

400 MHz, 128 M RAM, and windows 98 operating system. Only the pitch channel autopilot was implemented, and it was linked to a simplified mathematical model of equations of motion, standard atmosphere model, and aerodynamic forces and moments model. The system was simulated for two thousand seconds with a sampling period of 0.066 of a second, and the time taken to perform this task was approximately 40 seconds which is very fast. Altogether, the execution speed of the $MATRIX_X$ family evaluated in this project was satisfactory, and the faster the host machine is, the faster the execution speed.

6.1.3.3 Ease of Use of Scripting for System Simulation

In many ways, the math script language is similar to other programming languages such as FORTRAN. Consequently, if the user is familiar with programming techniques, then learning the math scripting language combined with the SystemBuild access commands to write script for system simulation is a mater of learning the syntax. Furthermore, Xmath provides the linked executable (LNX) program to call external C and C^{++} routines into Xmath and the user callable interface (UCI) program which allows external programs to call Xmath as a server. These two programs are more likely similar to C language, and users familiar with C would learn them very quickly.

6.1.4 Code Generation

The automatic code generation of the models in SystemBuild is done through a template programming language (TPL). TPL programs can be used to reflect the desired portion of a schematic representation of the models into real time C code to meet a specific requirement. The following criteria will be used to evaluate the ease of use of

the template programming language and the quality of the code generated for the flight warning computer system built in SystemBuild.

6.1.4.1 Ease of Template Programming

The template programming language is a multi-purpose program. It controls the structure of the generated code and can be used to insert source code and comments into the generated code. It also allows users to extract the required data from the SystemBuild model and reflect it in the generated code. The template programming language has its own unique syntax, and its concept resembles programming languages such as FORTRAN. It uses TPL tokens to extract almost any data from the SystemBuild model by scoping into each SuperBlock or subsystem. Its syntax is easy to learn and each TPL token extracts specific information from the SystemBuild model in the same way that the SystemBuild access commands extract information. Therefore, if a designer is familiar with the Xmath scripting language and SystemBuild commands, then it should not be difficult to learn the template programming language.

6.1.4.2 Quality of the Code

In this project a specific template program was written to generate a code to resemble the hand written code for the flight warning computer system. However, it was not possible to perform this task completely due to some restrictions in the template programming language which resulted in the generation of undesired code. Consequently, to overcome this problem, a post processing utility was written using the math scripting language to delete part of the undesired code that was generated

automatically. Regardless of this issue, the quality of the generated code was exactly as required which will be illustrated in the evaluation of results in the next section.

According to the report presented by AverStar, the SystemBuild model and its corresponding generated code in C may have inconsistencies in the result. Divergences occur when the variables used in the BlockScript blocks are not initialized explicitly. This is due to the fact that the variables used in BlockScript block are initialized to zero or false if they are not explicitly initialized in the BlockScript code. Therefore it does not create any problems in the SystemBuild simulation. This is not the case when the code is generated for this block, because the default initialization of variables in the BlockScript block is not reflected in the generated code. On the other hand, if the variable is not explicitly initialized in the generated code, then it could either be undefined or have any arbitrary initial value depending on its scope. Therefore, all the variables with a zero initial value should be explicitly initialized in the BlockScript block to overcome this problem.

6.2 Evaluation of Results

In this section, the results obtained for the simulation of the autopilot system and the code generated for the flight warning computer will be discussed. There will be a complete demonstration of the customization of AutoCode to generate a code that satisfies the software design requirements. Additionally, it will be demonstrated that a system implemented in the SystemBuild environment can be validated using a non real-time interactive simulation engine prior to the code generation.

6.2.1 Manual Code Versus Automatic Generated Code for the FWC System

For the purposes of evaluation of $MATRIX_X$ for the flight warning computer, a portion of the aircraft warning system was selected. The primary effort made was to generate a code similar to the manually written code. Consequently, the flight warning computer system components were designed and implemented in the SystemBuild environment accordingly to achieve this goal. In appendix 1, the manually written code is compared to the automatically generated code using $MATRIX_X$ for the selected part of the generated code. The only differences between the codes are the comments and the revision history written in the manual code. These differences are not important since they do not play a role in the execution of the code, and they are presented solely for the clarity of the code and recognition of the modifications made to the code by various designers. Most important, is the similarity between the structure and the source code introduced in the codes. As can be observed, the structure of both codes follows the same conventions starting with the 'includes files' followed by a declaration of external and local variables for this system, and the presentation of each warning logic page as a C program equation with the appropriate comments for easy identification of the various logic pages contained in the system. Note also that the static variables inside each equation are declared and initialized as required, and the debugger code is also inserted in the automatic generated code the same as with manual code. The template program written to generate this code is provided in appendix 2. It is also important to mention that the code generated using MATRIXx has been tested on the flight simulator, which had exact same result as the manual code.

6.2.2 Simulation Results for Autopilot System

In order to simulate the autopilot system practically, flight test data such as aerodynamic coefficients are required for specific flight conditions to calculate the aerodynamic forces and moments coefficients as the aircraft attitude changes. However, due to confidentiality issues the flight test data was not provided. Many other resources were explored for these data namely the regulation given in advisory circular 25.1329-1A (Automatic Pilot System Approval), FAA publications, etc; however, there were no flight test data available for accurate simulation of autopilot. Nevertheless, for purpose of evaluation of MATRIX$_X$, the approach taken was sufficient to prove that the autopilot model built in SystemBuild is functional. A large commercial aircraft was used for this simulation, and the flight data used is shown in figure 6.3.

Transport aircraft: Boeing 747

Longitudinal	C_L	C_D	C_{L_0}	C_{D_0}	C_{m_0}	C_{L_α}	C_{m_α}	C_{L_q}	C_{m_q}	$C_{L_{\dot\alpha}}$	$C_{D_{\dot\alpha}}$	$C_{m_{\dot\alpha}}$	$C_{L_{\delta_e}}$	$C_{m_{\delta_e}}$
M = 0.25														
Sea level	1.11	0.102	5.70	0.66	−1.26	6.7	−3.2	5.4	−20.8	−0.81	0.0	0.27	0.338	−1.34
M = 0.90														
40,000 ft	0.5	0.042	5.5	0.47	−1.6	0.006	−9.0	6.58	−25.0	0.2	0.25	−0.10	0.3	−1.2
Lateral	C_{y_β}	C_{l_β}	C_{n_β}	C_{y_p}	C_{n_p}	C_{l_r}	C_{n_r}	$C_{l_{\delta_a}}$	$C_{n_{\delta_a}}$	$C_{y_{\delta_a}}$	$C_{l_{\delta_r}}$	$C_{n_{\delta_r}}$		
M = 0.25														
Sea level	−0.96	−0.221	0.150	−0.45	−0.121	0.101	−0.30	0.0461	0.0064	0.175	0.007	−0.109		
M = 0.90														
40,000 ft	−0.85	−0.10	0.20	−0.30	0.20	0.20	−0.325	0.014	0.003	0.075	0.005	−0.09		

Note: All derivatives are per radian

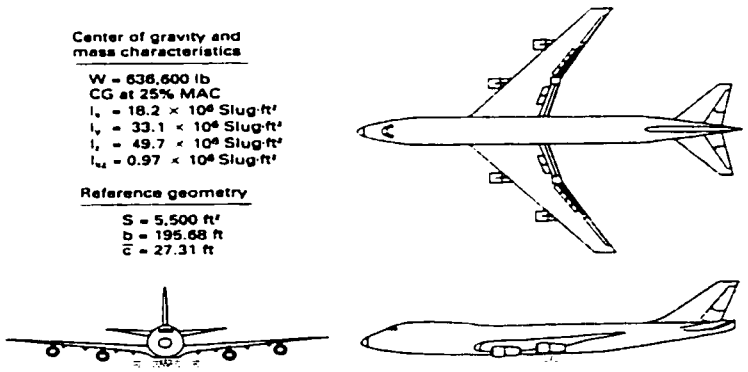


Figure 6.3 Sample large commercial jet data [51].

The data represents the coefficients of forces and moments of the aircraft in cruise, and the autopilot mode normally used for cruise is the altitude hold mode. Prior to the simulation, the parameters of the aircraft were initialized as given in figure 6.3 with the altitude hold mode selected. Consider that the aircraft is flying in a steady state condition without external forces and moments acting on the aircraft. Theoretically, the altitude of the aircraft will stay constant as the aircraft flies. The result of the simulation is consistent with the theory as expected. To illustrate the role of the autopilot in controlling the stability of the aircraft in pitch axis, a moment of the magnitude 500 ftlbs is exerted upwards about the Y axis at the time around 100 seconds. In the first scenario, shown in figure 6.4, the autopilot is disconnected from the aircraft. Note that the forces and moments acting on the aircraft are constant and equal to zero up to 100 seconds. Verify also that the altitude is constant with time for this period of simulation time. As soon as a moment is introduced, the aircraft starts to deviate from its steady state and with autopilot being disengaged, the attitude of aircraft changes in a growing harmonic manner which indicates instability.

In scenario 2, shown in figure 6.5, the autopilot is engaged and as long as no external forces and moments are acting on the aircraft and the motion of the aircraft remains steady. Once the external moment is exerted in the pitch axes, the autopilot changes the elevator control surface accordingly to oppose the external moments, and dampens out this unwanted moment and returns the aircraft motion to its normal stable condition. However, it can be observe that the time taken for the autopilot to perform its task is much longer than expected. This recovery time can be shortened by changing the autopilot gains. Figure 6.6 reflects this scenario where the gain of the autopilot has been

altered to decrease the time in which the autopilot brings the aircraft back to a stable attitude.

There are other factors that the autopilot does not respond rapidly in order to overcome the external moments. The first reason is that the mathematical model developed for this simulation of the aircraft is an approximate model and many factors have been neglected in calculating the forces and moments acting on the aircraft due to the unavailability of aircraft data. The second reason is that the available data for aerodynamic forces and moments coefficients are rounded roughly, and they are provided in the books only for the purpose of demonstration of aircraft behavior in certain conditions. For example, the total lift coefficient for the Boeing 747 is given as 0.52 in Roskam and as 0.5 in Nelson. For illustration purposes, this difference does not create any problems. However, for simulation of the system, this small amount is a significant factor in determining the aerodynamic forces and moments; since this value is multiplied by the wing area equal to 5500 ft^2 and the dynamic pressure at 222.8 lbs/ft^2 to give the lift of the aircraft. The difference for aircraft lift using these values is approximately 24000 lbs., a very large difference that has a great effect on the system's behavior. Therefore, for simulation of aircraft this data should contain more significant decimal points; basically the more decimal points would yield the more accurate results.

Nevertheless, the results shown in figures 6.4, 6.5, and 6.6 illustrate that the autopilot implemented in $MATRIX_X$ controls the aircraft stability in pitch axis. However, the system has not been tested for all pitch channel autopilot modes due to unavailability of aircraft data and incomplete autopilot design documentation provided for this project.

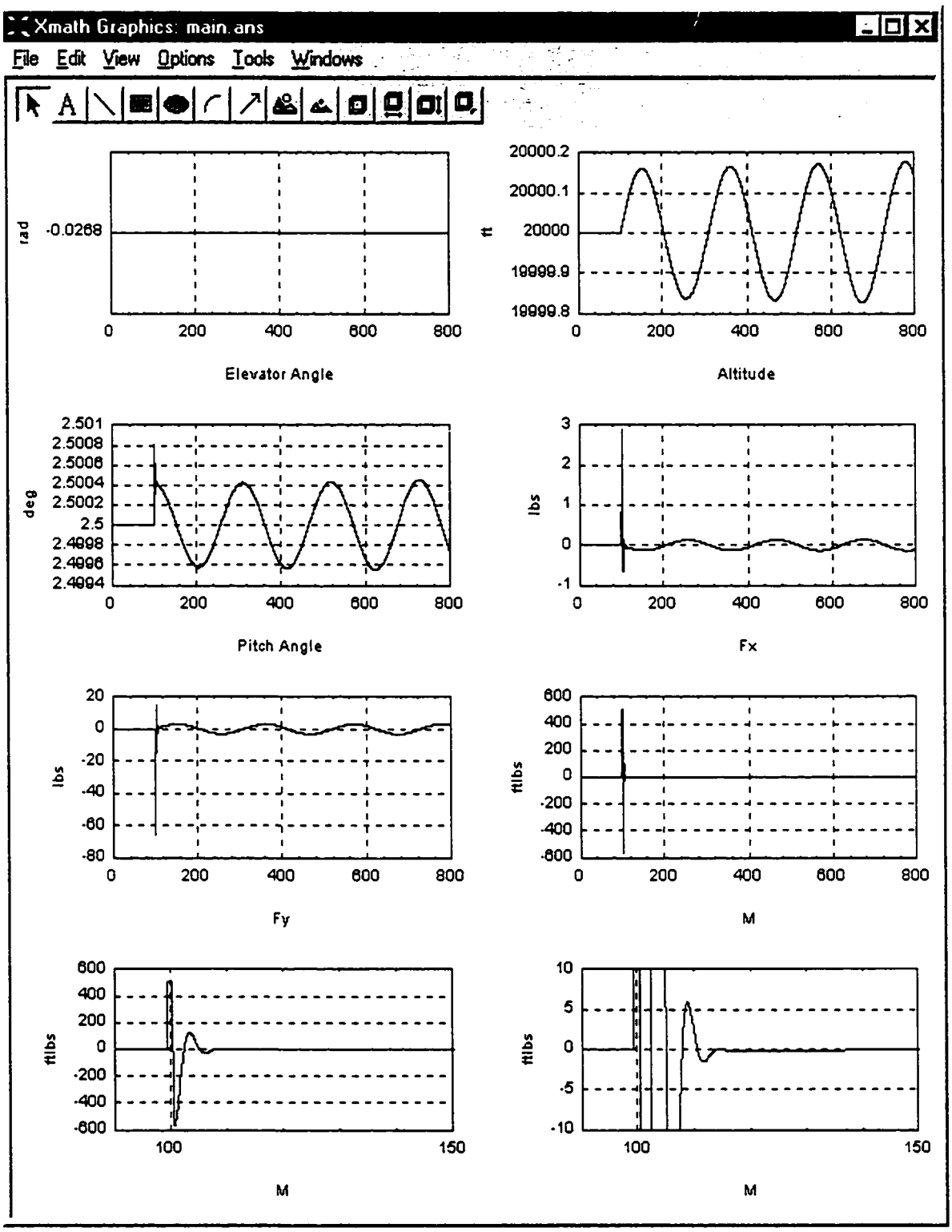


Figure 6.4 Simulation result with autopilot disengaged.

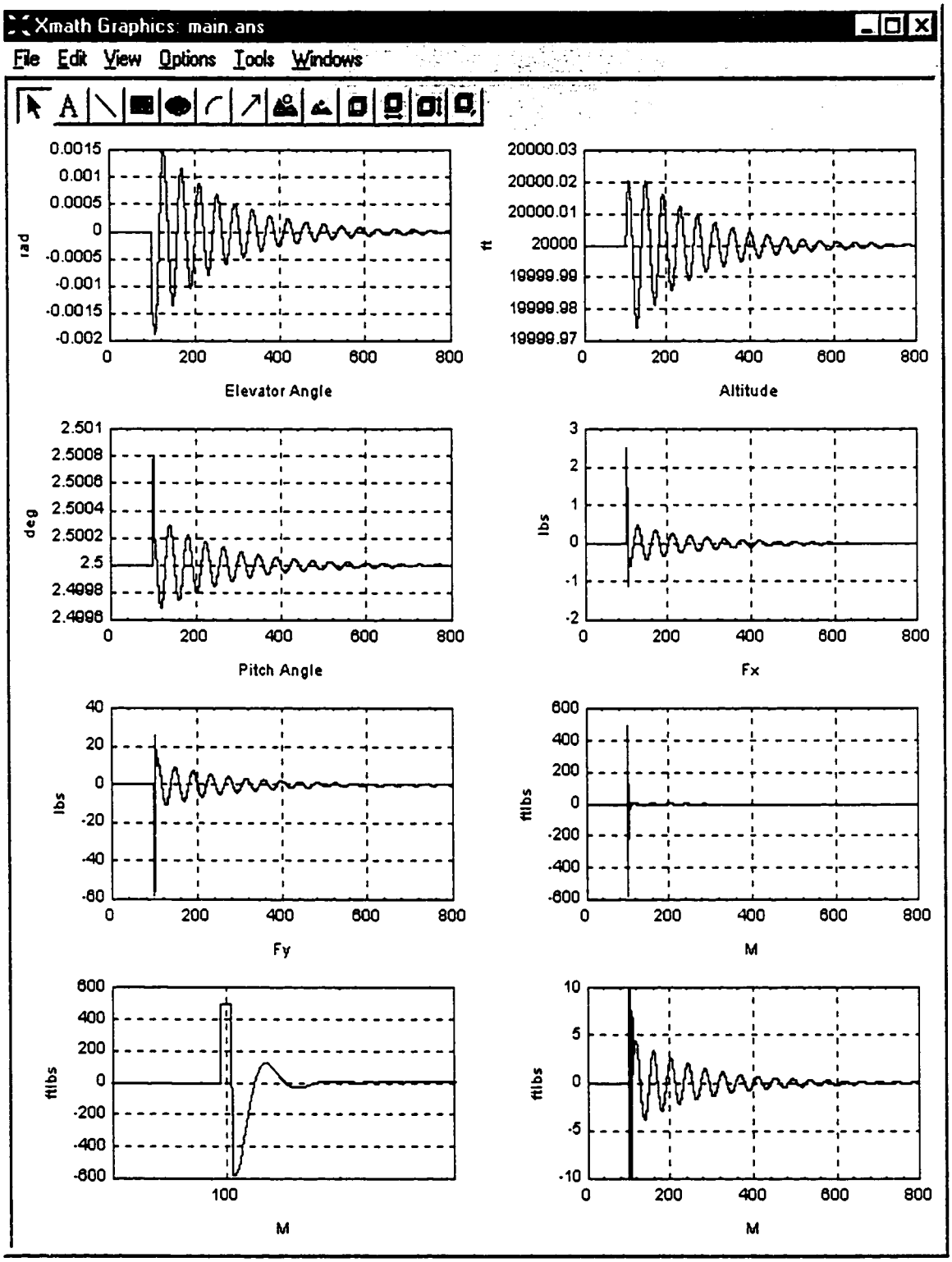


Figure 6.5 Simulation result with autopilot engaged.

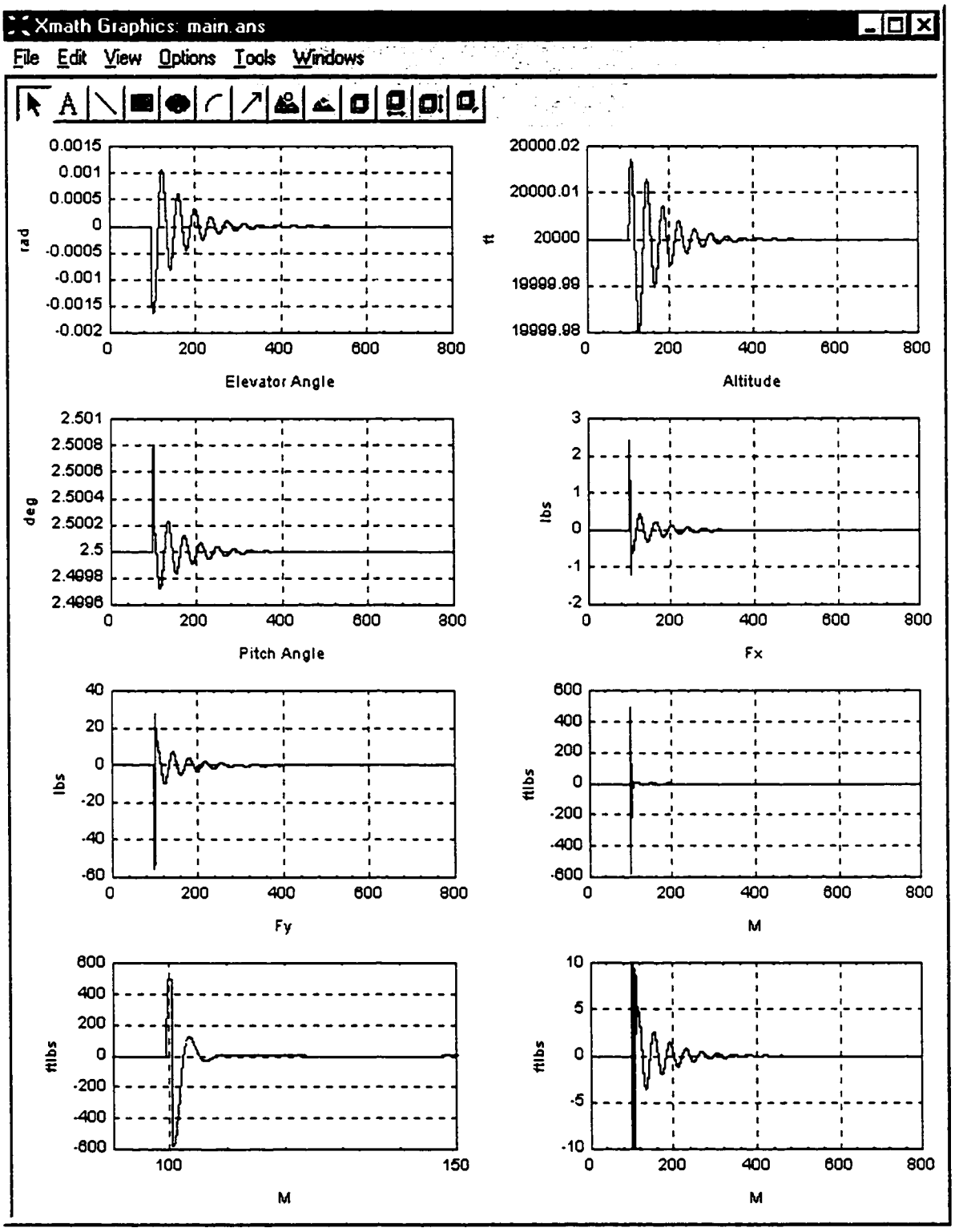


Figure 6.6 Result of changing autopilot gain.

7 CONCLUSION, RECOMMENDATIONS, AND FUTURE WORKS

7.1 Conclusion

The main purpose of this thesis was to evaluate the use of $MATRIX_X$ as a software development tool for flight simulators. The current hand coding approach is not only time consuming, but also generates many errors in the code. However, the $MATRIX_X$ product family provides a user-friendly environment to implement aircraft systems in a visual environment, and to generate the source code for the corresponding models automatically. This approach will speed up the software development process, provide better communication between different project teams, reduce coding errors, and provide a faster integration of new engineers into the work environment.

In this thesis, there were 2 objectives to achieve. The first objective was to implement the selected aircraft systems in the SystemBuild environment and to identify if there were any implementation issues that made the use of $MATRIX_X$ impractical. The second objective was to define set of criteria to evaluate $MATRIX_X$ and to determine if $MATRIX_X$ is a suitable tool to be used as software development tool. For this purpose the reduced scope of the typical flight warning computer system and a typical generic autopilot were chosen to reflect the nature of the various typical systems implemented in flight simulators. The flight warning computer was chosen to represent a typical Boolean system. Due to the time constraints for this part of the project, it was decided to see whether it is possible to generate a C code for the flight warning computer as similar as possible to the hand written code. The autopilot system was chosen to reflect a typical dynamic system. The complete autopilot model could not be implemented due to incomplete documentation provided for this project. Therefore, pitch channel autopilot

was chosen to evaluate $MATRIX_X$ for dynamic control systems. On the other hand, the license for AutoCode was not available for this part of the project. As a result, it was impossible to generate the simulation code for the autopilot system implemented in SystemBuild. However, care was taken during the autopilot design by choosing appropriate implementation approach as if the AutoCode license was available.

For flight warning computer, an appropriate component library for the blocks used in the selected model was developed. Logical expression blocks provided by SystemBuild were customized to represent the logical blocks used in the model. The logical blocks were customized to initialize certain variables upon creation of a block to reduce the generated code, and to resemble the models built in SystemBuild to the original design documentation data for the purpose of simple comparison. Furthermore, macro procedure SuperBlocks were customized to represent the already defined macros. For these blocks, AutoCode generated a macro call just as the macro is represented in the manual code. The flight warning computer was designed in a bottom-up fashion. Each warning page was represented as a SuperBlock, and all the warning pages belonging to an aircraft system were placed in a top level SuperBlock. A utility was developed to pass the external input parameters from all child SuperBlocks to their parent SuperBlock, sort the information, delete the multiple instances of same input, and automatically connect the external input of the parent SuperBlock to the corresponding child SuperBlock. A template program was written to automatically generate the corresponding real time C code as close as possible to hand written code from the flight warning computer model built in SystemBuild. Since some information could not be extracted from the SystemBuild model using the template programming language tokens, a utility was

developed to pass these variables on to the generated code automatically by using user defined parameters. Due to some AutoCode restrictions, the automatically generated code contained undesired code that could not be eliminated using the template programming language. Therefore, a post processing utility was developed to automatically delete the undesired portion of the generated code. The generated code from using the defined template program and post processing utility was exactly as a code that would be written manually. This automatically generated code was tested on flight simulator and the result obtained was satisfactory. Therefore, it can be concluded that the initial goal in this part of the project was achieved successfully.

In implementation of autopilot system, all dynamic blocks contained in the selected model, such as the integrator block, were customized using BlockScript blocks to take control over the AutoCode generated code for these blocks. The autopilot model was designed in a bottom-up fashion. The control laws for pitch channel mode of operation were placed in an if-else-then block in order to support inline code generation for these control laws, and to make sure that only the engaged mode is operated during the simulation. Unfortunately, due to confidentiality issues, the flight test data needed (such as aerodynamic forces and moment coefficient) to simulate the model was not provided. Therefore, an approximate mathematical model was developed to simulate the autopilot system using the rough data available for aerodynamic forces and moments coefficients. The model calculates the aircraft motion variables and aerodynamic forces and moments acting on the aircraft. A standard atmosphere model was also developed to calculate the Mach number needed as input to the autopilot system and to calculate the dynamic pressure needed for aerodynamic forces and moments calculations. The system

was tested for a typical large commercial aircraft flying in cruise with the altitude hold mode selected. An external moment was inserted for a specific period of time and the result of the simulation for autopilot being engaged and disengaged was observed. It was illustrated that while the autopilot is disengaged, the aircraft has unstable behavior. However, when the autopilot was engaged, the external moments and the resulting forces dampened out, and the aircraft returned to its normal operation over a period of time. The time taken to dampen out the external moments and forces might not be as fast as expected. This is due to the fact that the aerodynamic model built is approximate; the values of the aerodynamic forces and moments coefficients were rough data, and this data was assumed to be constant for small deviations of the flight condition which is not the case in practice. Nevertheless, the result obtained in this project was satisfactory for the purpose of evaluation of $MATRIX_X$.

As the second objective of this thesis, the evaluation criteria defined to evaluate $MATRIX_X$ was discussed in chapter five. The evaluation criteria used is divided into four categories: General, Implementation issues, Unit testing, and Code generation. In general, the $MATRIX_X$ product family proved to satisfy all the criteria. However, as discussed in chapter five, it needs future enhancement; namely to provide less restrictions for AutoCode and SystemBuild, to fix some bugs appearing in its GUIs, and to provide more command and utilities for its products such as an undo command for the Catalog Browser.

7.2 Recommendations and Future Works

The flight warning computer was implemented in SystemBuild for the purpose of code generation only. Due to time limits for this project, the goal was to generate a

simulation code automatically that would resemble the manually written code as close as possible. It is not possible to simulate the implemented model using the SystemBuild non-real time simulation. Because blocks, which were represented as macro calls in the manual code, were implemented using customized macro procedure SuperBlocks. In order to simulate the macro procedure SuperBlocks, its corresponding algorithm has to be implemented inside the SuperBlock using other blocks. For some macros such as confirmation, pulse, or flip-flop, their corresponding algorithm was implemented using BlockScript block inside the macro SuperBlock. Consequently, when the code is generated for the model, a macro call is generated in the code and when the model is simulated using interactive simulation, the BlockScript block representing the macro will be simulated. This is not the case for all macros implemented in the flight warning computer. Therefore, in future work, whenever a complete flight warning computer is going to be implemented using $MATRIX_X$, this task should be kept in mind.

The template program was written to generate the code for a selected portion of the flight warning computer and will not work for the complete model. This is due to the part of the code that searches for specific information of a macro block in the SystemBuild model and generates the appropriate code for that information. In the written template program, only the implemented macros for the selected model are searched for. As a result, in future work when a complete flight warning computer is implemented the corresponding part of the template program has to be modified accordingly.

On the other hand, it was not possible to implement and simulate a complete autopilot system due to incomplete design documents that were provided. Moreover, the

approximate mathematical model developed to simulate the aircraft is not mainly used in practice. In this project however, it was the only choice due to the available documentation and data. Consequently, the complete implementation of the autopilot system, generation of its corresponding code automatically, and simulation of this system in a more practical way is left for future work.

Finally, as was explained in section 4.1, due to personal interest some utilities were developed to simplify the creation of the component library and the task of the implementation of the systems in SystemBuild. These utilities proved to be useful tools during this project. They provide users with a user friendly interface to perform repetitive functions that should be entered in the command line with the correct syntax. These utilities eliminated the need for remembering all the syntax and keywords of the functions that are normally used during the modeling of a system. Even though these tools had satisfactory results during this project, they are not fully functional for the all the commands that are provided with $MATRIX_X$. In the future, any of these tools can be easily modified to be functional with more commands and keywords to simplify the task of the designer.

REFERENCES

1 J. M. Rolfe, K. J . Staples, Flight Simulation, Cambridge University Press, pp. 14-35.

2 E. Bruce Jackson, Results of Flight Simulation Software Survey, NASA – Langley Research Center, AIAA 95-3414.

3 CAE Electronics-Dept 78, Evaluation of MATRIX$_X$ as a Software Development Tool by Concordia University, Statement of Work.

4 Integrated Systems, Inc. , Getting Started (Windows), 000-0119-004, March 1999, p. 1.2.

5 http://www.windriver.com/products/html/xmath.html

6 http://www.windriver.com/products/html/systembuild.html

7 http://www.windriver.com/products/html/autocode.html

8 http://www.windriver.com/products/html/documentit.html

9 Integrated Systems, Inc. , How To Write Production Quality Code Graphically, 2,500 COLM0038, May 1998, pp. 1-12.

10 CAE Electronics Ltd., Software Design Documents A320 Flight Warning Computer, 10018900, February 1998.

11 A319/320/321 Flight Crew Operating Manual, Indicating/Recording Systems.

12 http://www.meriweather.com/a320/320_main.html

13 Peter Mellor, Computer-Related Factors in Incident to A320 G-Kman, Gatwick on 26 August 1993, University of Bielefeld, February 1995, p.13.

14 http://www.chipsplace.com/helpful/Airbus/Instrument.htm

15 Ian Moir, Allan Seabridge, Aircraft Systems, Longman Group UK Limited, 1989, p. 153.

16 D. h. Middleton,, Avionic Systems, Longman Group UK Limited, 1992, p. 75.

17 William A. Wainwright, Advanced Technology and The Pilot, Airbus Industry, Fast, Number 14, February 1993, p. 4.

18 Basics of Automatic Flight Control, Mech 609 Class Note, pp. 1-6.

19 Dune Mcruer, Irving Ashkenas, Dunstan Graham, Aircraft Dynamics and Automatic Control, Princeton University Press, 1973, pp. 204-205.

20 Donald Mclean, Automatic Flight Control Systems, Prentice Hall International (UK) ltd, 1990, pp. 16-27.

21 Bandu N. Pamadi, Performance, Stability, Dynamics, and Control of Airplanes, American Institute of Aeronautics and Astronautics, Inc. 1998, pp. 537-595.

22 Brian L. Stevens, Frank L. Lewis , Aircraft Control and Simulation, New York : Wiley, 1992, p.203.

23 M. V. Cook, Flight Dynamics Principles, Arnold, 1997, pp. 234-240.

24 United Air Lines, inc., Avionics Fundamentals, 1974, p. 252-301.

25 E. H. J. Pallett, Automatic Flight Control, BSP Professional Books, 1987, pp. 85-91.

26 Edward L. Safford, Aviation Electronics Handbook, Tab Books, 1975, pp. 92-101.

27 CAE Electronics Ltd., Avionics Systems, February 1996, pp. 4.17-4.18.

28 Pallett, pp.159-160.

29 CAE Electronics Ltd., CAE Generic Flight Control Computer, 10019685.

30 Pamadi, pp.616-618.

31 Jan Roskam, Airplane Flight Dynamics and Automatic Flight Controls Part II, Roskam Aviation and Engineering Corporation, 1979, pp. 1143-1148.

32 John H. Blakelock, Automatic Control of Aircraft and Missiles, John Wiley & Sons, Inc., 1965, pp.92-93.

33 Stevens, pp. 285-286.

34 Canada air Inc., Pilot Training Manual, p. 4.34.

35 A. Morris, Flight Control & Automatic Control, Course Notes, pp.113-116.

36 Robert C. Nelson, Flight Stability and Automatic Control, McGraw-Hill, Inc., 1989, pp. 219-220.

37 System Description Note, Flight Warning Computer, Aerospatiale, December 1992.

38 Integrated Systems, Inc., AutoCode Reference, 000-0155-005, February 1999, pp. 5.4-5.5.

39 Integrated Systems, Inc., AutoCode User's Guide, 000-0155-005, February 1999, pp. 1.5-1.6.

40 Integrated Systems, Inc., SystemBuild User's Guide, 000-0155-005, February 1999.

41 http://lorien.ncl.ac.uk/ming/digicont/digimath/

42 Eugene A. Morelli, Airplane Dynamics, Modeling, and Control, NASA Langley Research Center, May 1997.

43 Jan Roskam, Airplane Flight Dynamics and Automatic Flight Controls Part I, Roskam Aviation and Engineering Corporation, 1994, p. 21.

44 Peter Lawn. "The Enhancement of a Flight Simulator System with Teaching and Research Applications." M. A. Sc. , Concordia University, 1998.

45 Thomas S. Alderete1, Simulator Aero Model Implementation, NASA Ames Research Center, Moffett Field, California.

46 Louis V. Schmidt, Introduction to Aircraft Flight Dynamics, , pp. 93-97.

47 www.met-office.gov.uk/education/training/atmosphere.html

48 Steven A. Branddt et al, Introduction to Aeronautics: A Design Perspective, American Institute of Aeronautics and Astronautics, Inc., 1997, pp. 40-44.

49 Richard S. Shevell, Fundamentals of Flight, Prentice-Hall, Inc., 1983, pp. 63-71.

50 Mario Asselin, Operational Aircraft Performance, Concordia University, Cource Note, pp. 18-21.

51 Nelson, p. 260.

APPENDIX 1

COMPARISON OF MANUAL CODE AND AUTOMATIC GENERATED CODE FOR FWC

Manual Code 1/3

```
/* $ScmHeader: 9996497z72067C7z72y3999999978&16865|0 $*/
/* $Id: a320qww23.c,v 1.8 1.4 1998/06/18 15:08:33 inspec(MASTER_VERSION|CAE_MR) Exp
                                                                               $*/
/*
Title                   Simulation of the FWC : A320 and A321 BOOK 23 (communication)
Module ID               A320QWW23.C
Entry_Point             AVSQWW23
Customer                A32* SIMULATOR
Authors                 Daniel Leblanc
Date                    July 28, 1997
System                  Software Avionics
Subsystem               FWC
Documentation           FWC SDD
Process                 Synchronous
Iteration               266 ms
Version                 1.0

Compilation directives


Include_files_directives


Application

Abstract: Code for the warning logics of the A320 and A321 FWC for book 23


Description

References :

     Ref [1] : AEROSPATIALE GO.5 D31401706

Purpose

C'Revision_History
C
C  a320qww23.c.11 15Apr1998 14.17 a32x ab
C       < modified equation headers to include sheet numbers >
C
C  a320qww23.c.10 28Jan1998 19.51 ab32 meb
C       < add _f for false output >
C
C  a320qww23.c.9 23Jan1998 11.20 ab32 sg
C       < insertion of the FWC debugger >
C
C  a320qww23.c.8  8Dec1997 09.33 ab32 meb
C       < insert the ident part >
C
C  a320qww23.c.7  8Dec1997 08.55 ab32 meb
C       < change dispc for dispcom >
C
C  a320qww23.c.6  4Nov1997 14.44 1321 ll
C       < logic change in 22-00-040 for std e2 >
C
C  a320qww23.c.5 23Sep1997 11.10 sw76 ll
C       < added logic in 22-00-435,410,415 for std e2 >
C
C  a320qww23.c.4 23Sep1997 11:05 sw78 ll
C       < logic change in 23-00-040 for std e2 >
C
C  a320qww23.c.3 12Sep1997 09.19 1321 MEB
C       < CHANGE CONFIRMATION_UFX/INX BY UP/DN >
C
C  a320qww23.c.2 29Aug1997 10:51 1321 ss
C       < changed local type >
C
C  a320qww23.c.1 31Jul1997 11.43 1321 meb
C       < insert _f and _a variable >
C
C
C
C
C
C'
*/



/*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                          STANDARD C-LIBRARIES
                            & CAE LIBRARIES
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/


#include <math.h>                    /* !NOCPC */
#include <stdio.h>                   /* !NOCPC */
#include <stdlib.h>                  /* !NOCPC */
#include "cae.h"                     /* !NOCPC */
#include "qssmsdi.h"                 /* !NOCPC */
#include "a320qw_def.h"              /* !NOCPC */
#include "a320qw_cdb_labels.h"       /* !NOCPC */
#include "a320qw_fwclib.h"           /* !NOCPC */
#include "a320qw_smpc.h"             /* !NOCPC */
#include "a320qw_mnem.h"             /* !NOCPC */
#include "dispcom.h"                 /* !NOCPC */
#include "a320qw_indx_num.h"         /* !NOCPC */




/***************************************************************************/
/*                        EXTERN VARIABLES                                 */
/***************************************************************************/

/* Declare extern the variables that are already defined in another module */


unsigned char

/* page 23 00 435 */
        /* input */
                b_eemer,

/* page 23 00 410 */
        /* input */
                /* b_eemer, see 23 00 435 */

/* page 23 00 415 */
```

```
        /* input */
                /* b_eemat, see 23 00.435 */

/*****************************************************************************/
/*                      LOCAL VARIABLES                                      */
/*****************************************************************************/

/* Define the variables used in this module */



/* page 23 00 005 */
        /* output */
                b_output2300005,
                b_output2300005_f,


/* page 23.00.010 */
        /* output */
                b_output2300010,
                b_output2300010_f,


/* page 23.00 015 */
        /* output */
                b_output2300015,
                b_output2300015_f,


/* page 23.00.020 */
        /* output */
                b_output2300020,
                b_output2300020_f,


/* page 23.00.025 */
        /* output */
                b_output2300025,
                b_output2300025_f,


/* page 23.00.030 */
        /* output */
                b_output2300030,
                b_output2300030_f,
                b_cidsf,


/* page 23.00.050 */
        /* output */
                b_soundout2300050,

/* page 23 00 060 */
        /* output */
                b_soundout2300060,

/* page 23.00 062 */
        /* output */
                b_soundout2300062,

/* page 23 00 040 */
        /* output */
                b_output2300040,
                b_output2300040_f,
                b_cact,

/* page 23.00 430 */
        /* output */
                b_output2300430,
                b_output2300430_f,


/* page 23.00.440 */
        /* output */
                b_output2300440,
                b_output2300440_f,


/* page 23 00 450 */
        /* output */
                b_output2300450,
                b_output2300450_f,


/* page 23 00 435 */
        /* output */
                b_output2300435,
                b_output2300435_f,


/* page 23 00.400 */
        /* output */
                b_output2300400,
                b_output2300400_f,


/* page 23.00 405 */
        /* output */
                b_output2300405,
                b_output2300405_f,


/* page 23 00 410 */
        /* output */
                b_output2300410,
                b_output2300410_f,


/* page 23.00 415 */
        /* output */
                b_output2300415,
                b_output2300415_f
                ;  /* End of the local boolean variable list */


/* Flight warning computer simulation  Book 23 */

void avsqww23 (void)
```

```
{

/*
CIdent
*/
   static char rev[] =
      "$Source: a320-pew23 c.v $";


if ( qwlftz || qwlgftz || qwl23ftz ) return ;


/*****************************************************************/
/*                          Book 23                              */
/*****************************************************************/

/*****************************************************************/
/*                       Text Code 23 60.005                     */
/*                       Sheet 23.5                              */
/*****************************************************************/

/* eqd
       Q23_00010 COM   VHF1 EMITTING  (b_output2300005)
       Ref [1] 23 00 005
*/

{

  static long    td2200005_1,

  static unsigned char

         start2200005_1=FALSE,
         prev_c2200005_1=TRUE
         ;

  M_CONFIRMATION_UP (CVHF1EM,b_output2300005,td2200005_1,start2200005_1,prev_c220000
                                                                    5_1,60.0F),

  if (b_output2300005)
    { b_output2300005_f = TRUE,}
  else { b_output2300005_f = FALSE, }

  add_to_fault_list(I_INDEX2300005A, b_output2300005_f),

/* Debugger-Start */
  {
  int idx = 0,

  if      (qwidebeqn[0] == 2300005) idx=0;
  else if (qwidebeqn[1] == 2300005) idx=1;
  else if (qwidebeqn[2] == 2300005) idx=2;

  if ( idx < 3 )
    {
sprintf(qwidebtext[idx],"td2200005_1=%f,start2200005_1=%f,prev_c2200005_1=%f,\
CVHF1EM=%f,b_output2300005=%f,b_output2300005_f=%f,I_INDEX2300005A=%f,"\
,(float)td2200005_1,(float)start2200005_1,(float)prev_c2200005_1\
,(float)CVHF1EM,(float)b_output2300005,(float)b_output2300005_f\
,(float)I_INDEX2300005A);
    if (!qwidebeqn[idx])
      qwidebeqn[idx] = 0,
    }
  }
/* Debugger-End */
}

/*****************************************************************/
/*                       Text Code 23 60.010                     */
/*                       Sheet 23.10                             */
/*****************************************************************/

/* eqd
       Q23_00020 COM   VHF2 EMITTING  (b_output2300010)
       Ref [1] 23 00 010
*/

{

  static long    td2200010_1,

  static unsigned char

         start2200010_1=FALSE,
         prev_c2200010_1=TRUE
         ;

  M_CONFIRMATION_UP (CVHF2EM,b_output2300010,td2200010_1,start2200010_1,prev_c220001
                                                                    0_1,60.0F),

  if (b_output2300010)
    { b_output2300010_f = TRUE,}
  else { b_output2300010_f = FALSE, }

  add_to_fault_list(I_INDEX2300010A, b_output2300010_f),

/* Debugger-Start */
  {
  int idx = 0,

  if      (qwidebeqn[0] == 2300010) idx=0;
  else if (qwidebeqn[1] == 2300010) idx=1;
  else if (qwidebeqn[2] == 2300010) idx=2;

  if ( idx < 3 )
    {
sprintf(qwidebtext[idx],"td2200010_1=%f,start2200010_1=%f,prev_c2200010_1=%f,\
CVHF2EM=%f,b_output2300010=%f,b_output2300010_f=%f,I_INDEX2300010A=%f,"\
,(float)td2200010_1,(float)start2200010_1,(float)prev_c2200010_1\
,(float)CVHF2EM,(float)b_output2300010,(float)b_output2300010_f\
,(float)I_INDEX2300010A);
    if (!qwidebeqn[idx])
      qwidebeqn[idx] = 0,
    }
  }
/* Debugger-End */
}
```

Code Generated Using AutoCode 1/3

```
/*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                          STANDARD C-LIBRARIES
                            & CAE LIBRARIES
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/

#include <math.h>                      /* !NOCPC */
#include <stdio.h>                     /* !NOCPC */
#include <stdlib.h>                    /* !NOCPC */
#include "cae.h"                       /* !NOCPC */
#include "qssmsdi_matrixx.h"           /* !NOCPC */
#include "a320qw_def.h"                /* !NOCPC */
#include "a320qw_cdb_labels.h"         /* !NOCPC */
#include "a320qw_fwclib_matrixx.h"     /* !NOCPC */
#include "a320qw_smpc.h"               /* !NOCPC */
#include "a320qw_mnem.h"               /* !NOCPC */
#include "a320qw_indx_num.h"           /* !NOCPC */
#include "dispcom.h"                   /* !NOCPC */


/********************************************************************/
/*                          EXTERN VARIABLES                        */
/********************************************************************/

extern unsigned char

                      b_eemer
;

/********************************************************************/
/*                          LOCAL VARIABLES                         */
/********************************************************************/

unsigned char

/* page 2300005 */
   /* outputs */
                      b_output2300005
                     ,b_output2300005_t

/* page 2300010 */
   /* outputs */
                     ,b_output2300010
                     ,b_output2300010_t

/* page 2300015 */
   /* outputs */
                     ,b_output2300015
                     ,b_output2300015_t

/* page 2300020 */
   /* outputs */
                     ,b_output2300020
                     ,b_output2300020_t

/* page 2300025 */
   /* outputs */
                     ,b_output2300025
                     ,b_output2300025_t

/* page 2300030 */
   /* outputs */
                     ,b_output2300030
                     ,b_output2300030_t
                     ,b_cidst

/* page 2300050 */
   /* outputs */
                     ,b_soundout2300050

/* page 2300060 */
   /* outputs */
                     ,b_soundout2300060

/* page 2300062 */
   /* outputs */
                     ,b_soundout2300062

/* page 2300040 */
   /* outputs */
                     ,b_output2300040
                     ,b_output2300040_t
                     ,b_cact

/* page 2300430 */
   /* outputs */
                     ,b_output2300430
                     ,b_output2300430_t

/* page 2300440 */
   /* outputs */
                     ,b_output2300440
                     ,b_output2300440_t

/* page 2300450 */
   /* outputs */
                     ,b_output2300450
                     ,b_output2300450_t

/* page 2300435 */
   /* outputs */
                     ,b_output2300435
                     ,b_output2300435_t

/* page 2300400 */
   /* outputs */
                     ,b_output2300400
                     ,b_output2300400_t

/* page 2300405 */
   /* outputs */
                     ,b_output2300405
                     ,b_output2300405_t

/* page 2300410 */
   /* outputs */
                     ,b_output2300410
```

```
                    ,b_output2300410_f

/* page 2300415 */
   /* outputs */
                    ,b_output2300415
                    ,b_output2300415_f
;


/*  Flight warning computer simulation . Book 23 */


void avsqww23 (void)
{

/*
C'Indent
*/
    static char rev[] =
       "$Source: a320qww23.c,v $";

if ( qwlfrz || qwlyfrz || qwl77frz ) return;

/*****************************************************************/
/*                          chapter23                            */
/*****************************************************************/



/*****************************************************************/
/*                          page 2300005                         */
/*****************************************************************/

(

static unsigned character

                    prev2300005_2 = TRUE
                   ,started2300005_2 = FALSE

;

static long

                    td2300005_2

;


      /* ---- ------ --- ----- ------  User Macro Block */
      /* (Conf_u_2300005_2_2) */
      M_CONFIRMATION_UP(td2300005_2,started2300005_2,prev2300005_2,60.0F,
      CVHFIEM,b_output2300005),

      /* --------------------------  IfThenElse */
      /* (memo2300005, 1)) */
      if( b_output2300005 ) {

         /* -- -----  --                  Logical Expression */
         /* (memo2300005 logblk 12) */
         b_output2300005_f = 1;
         /* ----------  ---  -------- ---  User Macro Block */
         /* (adl_2300005_1_1) */
         add_to_fault_list(I_INDEX2300005A,b_output2300005_f),

      }
      else {

         /* ---------------  --------  -   Logical Expression */
         /* (memo2300005 logblk 10) */
         b_output2300005_f = 0;
         /* ----------  --  ------------  User Macro Block */
         /* (adl_2300005_1_1) */
         add_to_fault_list(I_INDEX2300005A,b_output2300005_f),

      }


/* Debugger Start */
  {
  int idx =1,

  if         (gwidebeqn[0] == 2300005) idx=0,
  else if    (gwidebeqn[1] == 2300005) idx=1,
  else if    (gwidebeqn[2] == 2300005) idx=2,

  if ( idx > 3 )
    {
sprintf(gwidebtext[idx],"td2300005_2=%f,prev2300005_2=%f,started2300005_2=%f,
CVHFIEM=%f,b_output2300005=%f,b_output2300005_f=%f,I_INDEX2300005A=%f,"
,(float)td2300005_2,(float)prev2300005_2,(float)started2300005_2
,(float)CVHFIEM,(float)b_output2300005,(float)b_output2300005_f,(float)I_INDEX230000
                                                                    5A);
    if (!gwidebmon[idx])
      gwidebeqn[idx] = 0;
    }
  }
/* Debugger-End */
}


/*****************************************************************/
/*                          page 2300010                         */
/*****************************************************************/

(

static unsigned character

                    prev2300010_1 = TRUE
                   ,started2300010_1 = FALSE
;

static long

                    td2300010_1

;


      /* ---- ---------- - - --------  User Macro Block */
      /* (Conf_u_2300010_1_1) */
```

Code Generated Using AutoCode 3/3

```
M_CONFIRMATION_UP(td2300010_1,started2300010_1,prev2300010_1,to OF,
CVHF2EM,b_output2300010),

/* ------------------------------  IfThenElse */
/* (memo2300010..2) */
if( b_output2300010 ) {

    /* ------------------------------  Logical Expression */
    /* (memo2300010.logblk.1) */
    b_output2300010_f = 1;
    /* ------------------------------  User Macro Block */
    /* (add_2300010_1.1) */
    add_to_fault_list(I_INDEX2300010A,b_output2300010_f);

}
else {

    /* ------------------------------  Logical Expression */
    /* (memo2300010.logblk.11) */
    b_output2300010_f = 0;
    /* ------------------------------  User Macro Block */
    /* (add_2300010_1.1) */
    add_to_fault_list(I_INDEX2300010A,b_output2300010_f);

}

/* Debugger-Start */
  {
  int idx =3;

  if      (qwidebeqn[0] == 2300010) idx=0;
  else if (qwidebeqn[1] == 2300010) idx=1;
  else if (qwidebeqn[2] == 2300010) idx=2;

  if ( idx < 3 )
    {
sprintf(qwidebtext[idx],"td2300010_1=%f,prev2300010_1=%f,started2300010_1=%f,
CVHF2EM=%f,b_output2300010=%f,b_output2300010_f=%f,I_INDEX2300010A=%f,"
,(float)td2300010_1,(float)prev2300010_1,(float)started2300010_1
,(float)CVHF2EM,(float)b_output2300010,(float)b_output2300010_f,(float)I_INDEX230001
                                                                    0A),

    if (!qwidebmon[idx])
      qwidebeqn[idx] = 0;
    }
  }
/* Debugger-End */
}
```

APPENDIX 2

TEMPLATE PROGRAM USED TO GENERATE THE CODE FOR FWC

```
@/*************************************************************************
@/      Written by:     Reza Ghassemian
@/      Date:           November 23, 2000
@/
@/      This template program is developed to generate a code for the selected FWC
@/      system such that the code resembles manually written code at CAE.
@/
@/*************************************************************************
@INT i,i1,i2,j,k@
@INC "c_intgr.tpl"@@
@INC "c_sched.tpl"@@
@INC "c_async.tpl"@@
@SEGMENT MAIN() STRING psarray[5], tdarray[5],array[100],s3,temp INT blank,
num_user_par, num_td, psind, index, count, flag@@
@ASSERT STRCMP("C",language_s)@@
@ASSERT not multiprocessor_b@@
@SCOPE PROCESSOR 0@@
@SCOPE SUPERBLOCK 0@@
/*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                                STANDARD C-LIBRARIES
                                    & CAE LIBRARIES
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/


#include <math.h>                       /* !NOCPC */
#include <stdio.h>                      /* !NOCPC */
#include <stdlib.h>                     /* !NOCPC */
#include "cae.h"                        /* !NOCPC */
#include "qssmsdi.h"                    /* !NOCPC */
#include "a320qw_def.h"                 /* !NOCPC */
#include "a320qw_cdb_labels.h"          /* !NOCPC */
#include "a320qw_fwclib.h"              /* !NOCPC */
#include "a320qw_smpc.h"                /* !NOCPC */
#include "a320qw_mnem.h"                /* !NOCPC */
#include "a320qw_indx_num.h"            /* !NOCPC */
#include "dispcom.h"                    /* !NOCPC */
```

```
/*************************************************************************/
/*                          EXTERN VARIABLES                             */
/*************************************************************************/

extern unsigned char

@j=0@@
@LOOPP i=0, i lt num_sb_in_i , i=i plus 1@@
@IFF STRCMP(sb_extin_dsc_ls[i], "external") eq 1@@
@IFF j eq 0@@
@22@ @sb_extin_name_ls[i]@
@j=1@@
@ELSE@@
@22@,@sb_extin_name_ls[i]@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
;

/*************************************************************************/
/*                          LOCAL VARIABLES                              */
/*************************************************************************/

unsigned char

@LOOPP k=1, k le ntasks_i, k=k plus 1@@
@SCOPE SUBSYSTEM k@@
@SCOPE SUPERBLOCK proc_sb_id_li[0]@
/* @sb_name_s@ */
  /* outputs */
@LOOPP i = 0, i lt num_uni_sb_i, i = i plus 1@@
@SCOPE SUPERBLOCK uni_sb_li[i]@@
@IFF sb_parent_id_i eq proc_sb_id_li[0]@@
@IFF STRSTR(sb_name_s,"memo")@@
@IFF k eq 1@@
@21@  @sb_extin_name_ls[0]@
@ELSE@@
@21@ ,@sb_extin_name_ls[0]@
@ENDIFF@@
@flag=0@@
@LOOPP i1=0, i1 lt num_blks_in_sb_i, i1=i1 plus 1@@
@SCOPE BLOCK i1@@
@IFF STRSTR(blk_name_s,"logb") and flag eq 0@@
@LOOPP j=0, j lt num_blk_out_i, j=j plus 1@@
@21@ ,@blk_out_name_ls[j]@
@ENDLOOPP@@
```

```
@flag=1@@
@ENDIFF@@
@ENDLOOPP@@
@ENDIFF@@
@IFF STRSTR(sb_name_s,"mess")@@
@21@ ,@sb_extin_name_ls[0]@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@SCOPE SUPERBLOCK proc_sb_id_li[0]@@
@LOOPP i2=0, i2 lt num_sb_out_i , i2=i2 plus 1@@
@IFF STRSTR(sb_extout_dsc_ls[i2], "loc")@@
@21@ ,@sb_extout_name_ls[i2]@
@ENDIFF@@
@ENDLOOPP@@
@ENDLOOPP@@
;


/*  Flight warning computer simulation : Book 23 */


void avsqww23 (void)
{

/*
C'Indent
*/
   static char rev[] =
     "$Source: a320qww23.c,v $";

if ( qwlfrz || qwlgfrz || qwl77frz ) return;
@SCOPE SUPERBLOCK 0@
/*************************************************************************/
/*                                    @sb_name_s@                        */
/*************************************************************************/

@IFF ntasks_i le 0@@RETURN 0@@ENDIFF@@
@FILECLOSE@@
@FILEOPEN("stdout","append")@@
        Generating subsystems definitions ...
@FILECLOSE@@
@LOOPP k=1, k le ntasks_i, k=k plus 1@@
@SCOPE SUBSYSTEM k@
@SCOPE SUPERBLOCK proc_sb_id_li[0]@
```

```
/*************************************************************************/
/*                                @sb_name_s@                            */
/*************************************************************************/

{

@index=0@@
@psind=0@@
@LOOPP i=0, i lt num_blks_in_sb_i, i=i plus 1@@
@SCOPE BLOCK i@@
@IFF blk_has_out_b@@
@LOOPP j=0, j lt num_blk_out_i, j=j plus 1@@
@IFF STRSTR(blk_out_name_ls[j],"b_")@@
@IFF STRSTR(blk_name_s,"dum") eq 0@@
@IFF not(STRSTR(blk_out_name_ls[j],"b_output") or
STRSTR(blk_out_name_ls[j],"b_sound"))@@
@array[index]=blk_out_name_ls[j]@@
@index=index plus 1@@
@ENDIFF@@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@ENDIFF@@
@ENDLOOPP@@
@LOOPP i = 0, i lt num_uni_sb_i, i = i plus 1@@
@SCOPE SUPERBLOCK uni_sb_li[i]@@
@IFF sb_parent_id_i eq proc_sb_id_li[0]@@
@IFF STRCMP(sb_attr_s,"Macro")@@
@IFF STRSTR(sb_name_s,"Conf")@@
@psarray[psind]=STRSTR(sb_name_s,"23")@@
@psind=psind plus 1@@
@IFF STRSTR(sb_extout_name_ls[0],"b_output") eq 0@@
@array[index]=sb_extout_name_ls[0]@@
@index=index plus 1@@
@ENDIFF@@
@ENDIFF@@
@IFF STRSTR(sb_name_s,"INV")@@
@array[index]=sb_name_s@@
@index=index plus 1@@
@ENDIFF@@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@count=index@@
@IFF index gt 0@@
unsigned character
```

```
@blank=0@@
@LOOPP i=0, i lt index , i=i plus 1@@
@IFF STRSTR(array[i],"and")@@
@blank=1@@
@count=count minus 1@@
@IFF count neq 0@@
@21@ @array[i]@,@
@ELSE@@
@21@ @array[i]@;@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@IFF blank eq 1@@

@ENDIFF@@
@blank=0@@
@LOOPP i=0, i lt index , i=i plus 1@@
@IFF STRSTR(array[i],"or")@@
@blank=1@@
@count=count minus 1@@
@IFF count neq 0@@
@21@ @array[i]@,@
@ELSE@@
@21@ @array[i]@;@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@IFF blank eq 1@@

@ENDIFF@@
@blank=0@@
@LOOPP i=0, i lt index , i=i plus 1@@
@IFF STRSTR(array[i],"not")@@
@blank=1@@
@count=count minus 1@@
@IFF count neq 0@@
@21@ @array[i]@,@
@ELSE@@
@21@ @array[i]@;@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@IFF blank eq 1@@

@ENDIFF@@
```

```
@blank=0@@
@LOOPP i=0, i lt index , i=i plus 1@@
@IFF STRSTR(array[i],"pulse")@@
@blank=1@@
@count=count minus 1@@
@IFF count neq 0@@
@21@ @array[i]@,@
@ELSE@@
@21@ @array[i]@;@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@IFF blank eq 1@@

@ENDIFF@@
@blank=0@@
@LOOPP i=0, i lt index , i=i plus 1@@
@IFF STRSTR(array[i],"conf")@@
@blank=1@@
@count=count minus 1@@
@IFF count neq 0@@
@21@ @array[i]@,@
@ELSE@@
@21@ @array[i]@;@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@IFF blank eq 1@@

@ENDIFF@@
@blank=0@@
@LOOPP i=0, i lt index , i=i plus 1@@
@IFF STRSTR(array[i],"INV")@@
@blank=1@@
@count=count minus 1@@
@IFF count neq 0@@
@21@ @array[i]@,@
@ELSE@@
@21@ @array[i]@;@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@IFF blank eq 1@@

@ENDIFF@@
```

```
@ENDIFF@@
@IFF psind gt 0@@
static unsigned character

@LOOPP i=0, i lt psind , i=i plus 1@@
@IFF i eq 0@@
@22@  prev@psarray[i]@ = TRUE
@22@ ,started@psarray[i]@ = FALSE
@ELSE@@
@22@ ,prev@psarray[i]@ = TRUE
@22@ ,started@psarray[i]@ = FALSE
@ENDIFF@@
@ENDLOOPP@@
;

static long

@LOOPP i=0, i lt psind , i=i plus 1@@
@IFF i eq 0@@
@22@  td@psarray[i]@
@ELSE@@
@22@ ,td@psarray[i]@
@ENDIFF@@
@ENDLOOPP@@
;
@ENDIFF@@

@define_subsystem()@

/* Debugger-Start */
  {
  int idx =3;
@SCOPE SUPERBLOCK proc_sb_id_li[0]@
@s3=STRSTR(sb_name_s,"23")@@
  if       (qwidebeqn[0] == @s3@) idx=0;
  else if   (qwidebeqn[1] == @s3@) idx=1;
  else if   (qwidebeqn[2] == @s3@) idx=2;

  if ( idx < 3 )
    {
sprintf(qwidebtext[idx],"@
@LOOPP i=0, i lt num_sb_in_i , i=i plus 1@@
@array[index]=sb_extin_name_ls[i]@@
@index=index plus 1@@
@ENDLOOPP@@
@/************************************************
```

```
@LOOPP i=0, i lt num_sb_out_i , i=i plus 1@@
@array[index]=sb_extout_name_ls[i]@@
@index=index plus 1@@
@ENDLOOPP@@
@LOOPP i = 0, i lt num_uni_sb_i, i = i plus 1@@
@SCOPE SUPERBLOCK uni_sb_li[i]@@
@IFF sb_parent_id_i eq proc_sb_id_li[0]@@
@IFF STRSTR(sb_name_s,"memo")@@
@array[index]=sb_extin_name_ls[0]@@
@index=index plus 1@@
@flag=0@@
@LOOPP i1=0, i1 lt num_blks_in_sb_i, i1=i1 plus 1@@
@SCOPE BLOCK i1@@
@IFF STRSTR(blk_name_s,"logb") and flag eq 0@@
@LOOPP j=0, j lt num_blk_out_i, j=j plus 1@@
@array[index]=blk_out_name_ls[j]@@
@index=index plus 1@@
@array[index]=blk_out_dsc_ls[j]@@
@index=index plus 1@@
@ENDLOOPP@@
@flag=1@@
@ENDIFF@@
@ENDLOOPP@@
@ENDIFF@@
@IFF STRSTR(sb_name_s,"mess")@@
@array[index]=sb_extin_name_ls[0]@@
@index=index plus 1@@
@array[index]=sb_extin_dsc_ls[0]@@
@index=index plus 1@@
@ENDIFF@@
@ENDIFF@@
@ENDLOOPP@@
@IFF psind gt 0@@
@LOOPP i=0, i lt psind , i=i plus 1@@
td@psarray[i]@=%f,@
prev@psarray[i]@=%f,@
started@psarray[i]@=%f,@
@ENDLOOPP@
@ENDIFF@@
@LOOPP i=0, i le index div 6 , i=i plus 1@@
@IFF i eq index div 6@@
@LOOPP j=i times 6, j lt i times 6 plus index mod 6, j=j plus 1@@
@array[j]@=%f,@
@ENDLOOPP@@
@ELSE@@
@LOOPP j=i times 6, j lt i times 6 plus 6, j=j plus 1@@
```

```
@array[j]@=%f,@
@ENDLOOPP@
@ENDIFF@@
@ENDLOOPP@"
@IFF psind gt 0@@
@LOOPP i=0, i lt psind , i=i plus 1@@
,(float)td@psarray[i]@@
,(float)prev@psarray[i]@@
,(float)started@psarray[i]@@
@ENDLOOPP@
@ENDIFF@@
@LOOPP i=0, i le index div 6 , i=i plus 1@@
@IFF i eq index div 6@@
@LOOPP j=i times 6, j lt i times 6 plus index mod 6, j=j plus 1@@
,(float)@array[j]@@
@ENDLOOPP@@
@ELSE@@
@LOOPP j=i times 6, j lt i times 6 plus 6, j=j plus 1@@
,(float)@array[j]@@
@ENDLOOPP@
@ENDIFF@@
@ENDLOOPP@);
   if (!qwldebmon[idx])
    qwidebeqn[idx] = 0;
   }
  }
/* Debugger-End */
}
@ENDLOOPP@@

  qwldone = TRUE;

} /* End of function avsqww23 */


@/
@FILEOPEN("stdout", "append")@@
Output generated in @output_fname_s@.

@FILECLOSE@@

@ENDSEGMENT@@
```

APPENDIX 3

INITIALIZATION OF THE FWC MODEL PRIOR TO CODE GENERATION

```
#*************************************************************************
#       Written by:     Reza Ghassemian
#       Date:           October 10, 2000
#
#       This utility is used to initialize the FWC model prior to code generation.  It passes
#       the external input variables from the lower level SuperBlock hierarchy to the
#       higher level.  In addition, it creates the required user parameters to be passed to
#       the template program which uses      these to generate the simulation code.
#
#*************************************************************************

Command ini sbn

in=[];
[ref1=references]=querysuperblock( sbn );
[rno1,]=size(ref1);
for j=1:rno1
sbn1=ref1(j);

[inl=inputlabel,inc=inputcomment]=querysuperblock( sbn1 )
in=[in;inl,inc]
out=[];
st=[];

editcatalog sbn1
[ref=references]=querysuperblock( sbn1 );
[rno,cno]=size(ref);
logic rno,sbn1
ltn sbn1
for i=1:rno
 [t1=procedureclass]=querysuperblock(ref(i));
 if t1=="Macro"
  [t=macro]=querysuperblock(ref(i));
   [r,c]= size(t);
   if r > 1
    [n=name]=querysuperblock(ref(i));
      if stringex(n,1,3) <> "VAL" then
      for j=2:r
      st=[st;t(j)];
      endfor
```

```
      endif
    endif
  endif
endfor

[num,]=size(st);


for i=1:num
   ind=index(st(i),";");
   st(i)=stringex(st(i),1,ind-1);
   if stringex(st(i),1,4) == "prev" then
     out=[out;st(i)+" = TRUE"];
   elseif stringex(st(i),1,4) == "star" then
     out=[out;st(i)+" = FALSE"];
     elseif stringex(st(i),1,2) == "td" then
     out=[out;st(i)];
      elseif stringex(st(i),1,3) == "lim" then
      out=[out;st(i)];
   endif
endfor

[out,]=sort(out);
[rsiz,]=size(out);

num=0;
for i=1:rsiz
if stringex(out(i),1,2)=="td" then
  num=num+1;
endif
endfor

for i=1:rsiz
out(i)="@s"+string(i-1)+"_s@ "+out(i);
endfor

out=[out;"@num_sb_user_par_i@  "+string(rsiz)] ;
out=[out;"@num_td_par_i@ "+string(num)];

if find(in=="external") == null then
 out=[out;"@num_external_i@  "+string(0)] ;
else
 out=[out;"@num_external_i@  "+string(1)] ;
endif
modifysuperblock sbn1 , {Comment=out}
endfor
```

```
in2=[];
flag=0;
[rsiz,]=size(in);
for i=1:rsiz
 for j=i+1:rsiz
  if in(i,1)==in(j,1) then
   in(j,1)="z";
   flag=1;
  endif
 endfor
endfor
if flag > 0 then
[in2,sortindx]=sort(in(:,1));
for i=1:rsiz
 in2(i,2)=in(sortindx(i),2);
endfor
ij=find(in2=="z");
ij1=ij(1,1)-1;
in2=in2(1:ij1,:);
else
in2=in;
ij1=rsiz;
endif
main.in=[],
main.in=in2;

modifysuperblock sbn ,
{inputs=ij1,inputname=in2(:,1),inputlabel=in2(:,1),inputcomment=in2(:,2)}
logic ij1,sbn

[bl=BlockList]=querysuperblock( sbn );
for j=1:rno1
sbn1=ref1(j);
bid=bl(j);
[in1=inputlabel]=querysuperblock( sbn1 )
[sizein,]=size(in1);
for i=1:sizein
ij2=find(in2(:,1)==in1(i));
 if ij2 then
 createconnection 0,bid,[ij2,i];
 endif
endfor
endfor

endcommand
```

APPENDIX 4

POST PROCESSING UTILITY DEVELOPED FOR FWC

```
#*************************************************************************
#       Written by:     Reza Ghassemian
#       Date:           November 23, 2000
#
#       This utility is used to delete the portion of the generated code that could not be
#       avoided using the template programming language.
#
#*************************************************************************

[sb_name=name]=querysuperblock()

creatertf sb_name , {rtf="c:\codegen\filez\"+sb_name+".rtf"}
autocode sb_name,
{options="c:\codegen\autostar.opt",rtf="c:\codegen\filez\"+sb_name+".rtf",file="c:\codeg
en\filez\"+sb_name+".c"}
#execute file="c:\codegen\Fix_code.ms"


# post processing of the generated code.

temp_name="c:\codegen\filez\"+sb_name+"_1.c";
if exist(temp_name, {file}) then
oscmd("del  temp_name")
endif

temp_name2="c:\codegen\filez\"+sb_name+".c";
myfile=[];
ind=0;
myfile=read(temp_name2);
len=length(myfile);
ind=index(myfile,"void subsys");
while ind <> -1
ind1=index(myfile,"/***** Output Update. *****/");
temp=stringex(myfile,1,ind-1);
temp2=stringex(myfile,ind1+28,len);
myfile=temp+temp2;
len=length(myfile);
ind=index(myfile,"void subsys");
endwhile

len=length(myfile);
```

```
ind=index(myfile,"U->");
while ind <> -1
temp=stringex(myfile,1,ind-1);
temp2=stringex(myfile,ind+3,len);
myfile=temp+temp2;
len=length(myfile);
ind=index(myfile,"U->");
endwhile

len=length(myfile);
ind=index(myfile,"> 0.0");
while ind <> -1
temp=stringex(myfile,1,ind-1);
temp2=stringex(myfile,ind+5,len);
myfile=temp+temp2;
len=length(myfile);
ind=index(myfile,"> 0.0");
endwhile


len=length(myfile);
ind=index(myfile,"Y->");
while ind <> -1
temp=stringex(myfile,1,ind-1);
temp2=stringex(myfile,ind+3,len);
myfile=temp+temp2;
len=length(myfile);
ind=index(myfile,"Y->");
endwhile

len=length(myfile);
ind=index(myfile,"!(!");
while ind <> -1
temp=stringex(myfile,1,ind-1);
myfile=temp+stringex(myfile,ind+3,ind+17)+stringex(myfile,ind+19,len);
len=length(myfile);
ind=index(myfile,"!(!");
endwhile

fprintf(temp_name,"%s",myfile)

display("******post processing is complete******
```

APPENDIX 5

CALL BACK FUNCTION FOR LAG FILTER

```
#*************************************************************************
#       Written by:     Reza Ghassemian
#       Date:           November 31, 2000
#
#       This program executes upon creation of a lag filter in SystemBuild editor.  This
#       program generates a GUI to allow users to enter the required information.
#       It then uses the values provided by users to create the lag filter's icon and its
#       code in the form of blockscript language.
#
#*************************************************************************

Command LagFilter {fragName, widget, instance}

  alias T      "LagFilter"
  alias MW     "MainWin"

if ( exist(fragName) )
    goto *fragName;
  else


  if ( !exist(_LagFilter, {partition}) )
        new partition _LagFilter
  endif

  if (uiExist(T, MW))
      void = uiShow(T, MW);
      void = uiWindowDeiconify(T, MW);
      void = uiWindowRaise(T, MW);
  else


    _LagFilter.choice=1;
    _LagFilter.tau=0;


    _LagFilter.tool = uiTooiCreate( T, {name="LagFilter", partition="_LagFilter."});

    _LagFilter.width   = 350;
    _LagFilter.height  = 210;
    MainWin = uiWindow( T,
```

```
{name = "MainWin",
type = "panel",
title = "Lag Filter",
visibility = 1,
width = _LagFilter.width,
height = _LagFilter.height,
xr=0,
yr=0});

_LagFilter.MainWin = MainWin;

Message = uiPanel(MainWin,
 {name = "message",
 title     = "Choose the type of Approach",
 height    = _LagFilter.height,
 width     = _LagFilter.width})

void = uivarchoice(Message,{
 flags     ="v",
 items     =["Tustin","Euler Backward","Euler Forward"],
 values    =[1,2,3],
 varname = "choice",
 xr=15,
 yr=30,
 width=130,
 height=60})

void = uiseparator(Message,{
 flags = "h",
 xr=0,
 yr=100,
 width=350})

void = uilabel(Message,{
 text      ="Enter the Value of Tau:",
 xr=15,
 yr=110,
 width=200,
 height=20})

void = uivaredit(Message,
 {
 varName = "tau",
 flags = "s",
 xr=200,
 yr=110,
```

```
      width=100,
      height=30})

    void = uibutton(Message,{
      type= "Button",
      text= "Done",
      xmath = "Done",
      yr=150,
      xr=145,
      width=60,
      height=40})

   endif

   void = uiShow(T, MW);
endif

   return;
##############################################################################
<Done>
if _LagFilter.choice==1 then
iconstring=[
"ICON_WIDTH: 600"
"ICON_HEIGHT: 1000"
"SET_LINE_WIDTH 2"
"SET_TEXT_FONT 14"
"DRAW_TEXT 300 700 22 '1'"
"draw_line 2 60 500 520 500"
"DRAW_TEXT 300 300 22 '"+string(_LagFilter.tau)+"S+1'"
"SET_TEXT_FONT 1"
"SET_LINE_WIDTH 1"
"DRAW_TEXT 300 -130 22 'Tustin'" ]
bcode =[
"inputs: (in); "
"outputs: (out);"
"parameters: (ic,prev_in,prev_out);"
"ENVIRONMENT: (TIME,TSAMP);"
"float in, ic ,prev_in,prev_out;"
"" ; "if TIME == 0 then"
"   prev_in=in;"
"   out=ic;" ; "else"
"   out=prev_out+(TSAMP/(0.5*TSAMP+"+string(_LagFilter.tau)+"))*(0.5*(prev_in +
in)-prev_out);"
"   prev_in=in;" ; "endif;"
"prev_out=out;" ]
elseif _LagFilter.choice==2 then
```

```
iconstring=[
"ICON_WIDTH: 600"
"ICON_HEIGHT: 1000"
"SET_LINE_WIDTH 2"
"SET_TEXT_FONT 14"
"DRAW_TEXT 300 700 22 '1'"
"draw_line 2 60 500 520 500"
"DRAW_TEXT 300 300 22 '"+string(_LagFilter.tau)+"S+1'"
"SET_TEXT_FONT 1"
"SET_LINE_WIDTH 1"
"DRAW_TEXT 300 -130 22 'B_Euler'" ]
bcode = ...
    [ "inputs: (in); " ; "outputs: (out);" ; "parameters: (ic,prev_out);" ; "ENVIRONMENT:
(TIME,TSAMP);" ; "float in, ic ,prev_out;" ; "" ; "if TIME == 0 then" ; "   out=ic;" ;
"else"
     "   out=prev_out+TSAMP/(TSAMP+"+string(_LagFilter.tau)+")*( in - prev_out);"
     "endif;" ; "prev_out=out;" ]
elseif _LagFilter.choice==3 then
iconstring=[
"ICON_WIDTH: 600"
"ICON_HEIGHT: 1000"
"SET_LINE_WIDTH 2"
"SET_TEXT_FONT 14"
"DRAW_TEXT 300 700 22 '1'"
"draw_line 2 60 500 520 500"
"DRAW_TEXT 300 300 22 '"+string(_LagFilter.tau)+"S+1'"
"SET_TEXT_FONT 1"
"SET_LINE_WIDTH 1"
"DRAW_TEXT 300 -130 22 'F_Euler'" ]
bcode = ...
    [ "inputs: (in); " ; "outputs: (out);" ; "parameters: (ic,prev_in,prev_out);" ;
"ENVIRONMENT: (TIME,TSAMP);" ; "float in, ic ,prev_in,prev_out;" ; "" ; "if TIME
== 0 then" ; "   prev_in=in;" ; "   out=ic;" ; "else"
      "   out=prev_out+TSAMP/"+string(_LagFilter.tau)+"*(prev_in - prev_out);"
      "   prev_in=in;" ; "endif;" ; "prev_out=out;" ]
endif

modifyblock _LagFilter.id, {Code=bcode,CustomIcon = iconstring, IconType =
"Custom"};

void = uidestroy(T)
delete _LagFilter.*
delete _LagFilter.
return;

endcommand
```