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.

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

DYNAMIC SIMULATOR OF A FLIGHT MANAGEMENT SYSTEM FOR COMMERCIAL JETLINER – DEVELOPMENT, INTEGRATION AND APPLICATION

Ming He Liu

A Thesis

In

The Department

Of

Mechanical and Industrial Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

October 2002

©Ming He Liu, 2002

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-77701-4

Canada

ABSTRACT

Dynamic Simulator of a Flight Management System

for Commercial Jetliner –

Development, Integration and Application

Ming He Liu

Flight management of modern civil aircraft is realized by a system of sophisticated computerized devices and commands, which provides the pilot with navigation, aircraft status and other dynamic flight information enabling him to perform his task correctly. However, rigorous testing of the flight management system (FMS) is needed to discover and remedy any flaw in the hardware or software components so as to ensure proper running of the system, as well as to reveal potential for further improvement. A joint project between the CMC Electronics Canada Inc., world leader in civil aircraft FMS, and the Concordia University was initiated to develop a dynamic test bed (DTB) for achieving this aim, culminating in an application-level testing tool that could meet specified requirements yet remain within affordable range. This paper describes achievements gained in phase 1 and 2 of the project, focussing on the A429-PC hardware setup and fabrication of a breakout box with its circuit board on the hardware side, as well as the real-time architecture design, an executive program, a device driver for A429 and a client-server Ethernet network on the software side. Accessory tools created such as an ARINC switch graphic user interface (GUI), ARINC label GUI and discrete signal

GUI are also described. Also given is a rather comprehensive overview of the principles relevant to the project together with suggestions for future improvement of the DTB.

ACKNOWLEDGEMENTS

The author wishes to pay special tribute to his supervisors Dr. H. Hong and Dr. J. V. Svoboda, for their guidance in completing this thesis.

To Mr. Gilles Huard the author is greatly appreciative for his assistance concerning the custom electronic hardware.

The author is especially indebted to his father and wife for providing the help and encouragement to undertake this interesting task.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

ACT	-	Approach/Terminal Control
ADC	-	Air Data Computer
ADF	-	Automatic Direction Finder
AGL	-	Above Ground Level
AEEC	-	Airlines Electronic Engineering Committee
AHRS	-	Attitude Horizon Reference System
AIGND	-	Analog Input Ground
AISENSE	-	Analog Input Sense
ANP	-	Actual Navigation Performance
ARINC	-	Aeronautical Radio Incorporation
ATC	-	Air Traffic Control
ATM	-	Air Traffic Management
CDU	-	Control Display Unit
CIC	-	Center for Industrial Control of Concordia University
CMA	-	Canadian Marconi Avionics
CMC	-	CMC Electronics Canada Inc.
CRS	-	Course to Waypoint
DA	-	Drift Angle
DIO	-	Discrete Input / Output
DLL	-	Dynamic Link Library
DME	-	Distance-measuring Equipment

D/R	-	Dead Reckoning
DSP	-	Digital Signal Processor
DTB	-	Dynamic Test Bed
DTG	-	Distance to Go
EPU	-	Estimate of Position Uncertainty
ETA	-	Estimated Time of Arrival
FAF	-	Final Approach Fix
FLSIM	-	Flight Simulator
FMC	-	Flight Management Computer
FMS	-	Flight Management System
FMU	-	Flight Management Unit
FOM	-	Figure of Merit
GIAL	-	GPS Integrity Alarm Limit
GPS	-	Global Positioning System
GS	-	Ground Speed
GUI	-	Graphic User Interface
HDG	-	Heading
HDOP	-	Horizontal Dilution of Precision
HIL	-	Horizontal Integrity Limit
HOL	-	Higher Order Language
HSI	-	Horizontal Situation Indicator
IAS	-	Indicated Air Speed
ICAO	-	International Civil Aviation Organization

ILS	-	Instrument Landing System
INS	-	Inertial Navigation System
IRS	-	Inertial Reference Sensor
LNAV	-	Lateral Navigation
LRU	-	Line Replace Unit
MAP	-	Missed Approach Point
MCP	-	Mode Control Panel
MFC	-	Microsoft Foundation Classes
NM / nm	-	Nautical Mile, 1 nm = 1.853 km
NRSE	-	Non-referenced Single-ended
OOP	-	Object-oriented Programming Language
OS	-	Operating System
PGIA	-	Programmable Gain Instrumentation Amplifier
PPI	-	Programmable Peripheral Interface
RAIM	-	Receiver Autonomous Integrity Monitoring
RNP	-	Required Navigation Performance
RSE	-	Referenced Single-ended
RTA	-	Required Time of Arrival
SDI	-	Source / Destination Identifier
SID	-	Standard Instrument Departure
SPDT	-	Single-pole Single-throw
SSM	-	Sign / Status Matrix
STAR	-	Standard Terminal Arrival Routes

TAS	-	True Air Speed
TCP/IP	-	Transmission Control Protocol / Internet Protocol
TK	-	Ground Track Angle
TKE	-	Track Angle Error
TOGA	-	Take off Go Around
VHF	-	Very High Frequency
VNAV	-	Vertical Navigation
VOR	-	VHF Omni-directional Range System
VPI	-	Virtual Prototypes Incorporation
WD	-	Wind Direction
WS	-	Wind Speed
XTK	-	Cross-track Distance
ZIF	-	Zero Insertion Force

CHAPTER 1

INTRODUCTION

1.1 Background of Navigation Technology

Navigation is the determination of following a predetermined path of a moving vehicle. The three components of position and the three components of velocity make up a six-component state vector that fully describes the translational motion of the vehicle. Navigation data are usually sent to other onboard sub-systems, for example, to the flight control, flight management, engine control, communication control and crews displays computers. *Navigation sensors* may be located in the vehicle, in another vehicle, on the ground, or in air space. When the state vector is measured and calculated onboard of a moving vehicle, the process is called *navigation process.*

Essentially, aircraft instruments are extensions of a pilot's senses, refinements of his/her vision and other sensory perceptions that supply the information he or she needs to feel at home in the sky at all times. Avionics (shorthand for aviation electronics) and instrumentation are commonly used terms describing those systems, equipment, and instruments that enable the pilot to control and monitor the performance of the aircraft and that provide vital information to the ground-based air traffic control system. Today's advancing technology, with its increase in automation and function integration, suggests that these terms might have become obsolete. Further, future aircraft development, directed toward even greater integration of the aircraft's systems, will fundamentally change the role of the pilot within this environment [1].

In the past, much emphasis was placed on acquiring and displaying to the pilot an ever-increasing amount of information required during a flight. For example, during takeoff and landing, the pilot reviews his checklist, constantly scans his instruments, especially noting the attitude of the aircraft, rate of climb (rate of descent when landing), airspeed, and engine functions, maintains direction, looks outside to avoid collisions, maintains communication with the Approach/Terminal Control (ACT) when necessary, and monitors a host of other parameters. All these necessary actions result in increased workload by forcing the pilot to evaluate highly complex sets of input data, decide upon courses of action, and then implement these actions in minimal times. Such situations are seen as undesirable since they increase the pilot's chances of making errors, consequently reducing the intelligent, computer-aided decision-making system of the modern aircraft, which works cooperatively with the pilot in order to ensure the safety of the aircraft and its crew and passengers.

The role of the pilot may now and in the future be viewed as that of a flight manager, one who monitors the aircraft's performance but maintains override control of all automated and/or integrated functions associated with the aircraft. A more appropriate and perhaps more descriptive term for this broad area of concern is "flight management."

A flight management system, in the context of this definition, comprises the sensors, displays, and associated subsystems used to process and record flight information. The flight information includes data for status, control, communications, navigation, propulsion, landing, alerting, self-protection, armament support systems (in military aircraft), and other flight parameters involved in the air and on the ground. The

flight management system, working as a synergistic unit, processes and records the data as well as integrates them, working in consonance with the pilot's knowledge, experience, and judgment.

1.1.1 Flight Instruments

By the early 1930s, the developing aviation community considered as being essential three general groups of: 1) flight instruments (such as the pitot-static tube) to indicate the aircraft's attitude, 2) navigation instruments (such as compass and autopilot) to inform the pilot of his position in air space, and 3) engine and airframe instruments (such as fuel flow meter and tachometer) to indicate how the aircraft is operating.

Even though some systems are no longer representative of modern technology, a brief description of the systems will be presented here due to two reasons. First, a belief that by knowing the line of development from past to present, it would be far easier to comprehend that from present to future. Second, airport systems are built to extremely high standards and are consequently very expensive. Some old equipment is still in operation, meeting the original specifications and still performing an excellent job.

1.1.2 Navigation Systems

Navigation systems can be grouped into three branches termed airborne navigation system, ground-based navigation system and space-based navigation system.

1.1.2.1 Airborne Navigation Systems [27]

A typical application of the airborne navigation system is the Inertial Navigation System (INS). The early INS generally included a computer and a stable platform

composed of gyroscopes and accelerometers suspended in a gimbal arrangement. The gyros, together with the computer, generated signals to maintain the platform's accelerometers in a level position with relation to the earth's surface. In this way, they accurately measured aircraft accelerations, which were then converted to velocities. These velocities, when modified by the computer with appropriate corrections, provided extremely accurate navigational information. In addition, the inertial platform provided attitude outputs of roll, pitch, and platform heading as analog signals, which could be used by other systems in the aircraft.

The conventional inertial navigation system, after an injection of initial position information, is now capable of continuously and accurately updating displays of position, ground speed, attitude, and heading. In addition, it provides guidance or steering data for the autopilot and other flight instruments. However, the accuracy of the inertial navigation system will degrade over a period of time in flight, and therefore the system must be periodically updated.

The accelerometer is the basic measuring instrument of the inertial navigation system. Accelerometers mounted at right angles to one another within the gimbal system of a vertical gyro can measure the acceleration of the aircraft horizontally and vertically. While acceleration by itself is of little importance, its integration with time results in a measure of aircraft velocity. A second integration provides distance form the aircraft's take-off point. The accelerometer is basically a pendulum device that swings from its null position when the aircraft accelerates. A signal-pickoff device determines how far the pendulum has moved, and the resultant signal is sent to an amplifier. The signal from the amplifier activates a torquer in the accelerometer that restores the pendulum to the

null position. The amount of signal (electrical current) sent to the torquer is a function of the aircraft's acceleration.

1.1.2.2 Ground-based Navigation Systems [28]

Typical applications of ground-based navigation systems are the VHF Omni-directional Range System (VOR) and Distance-measuring Equipment (DME).

VOR is the currently International Civil Aviation Organization (ICAO)-approved short-distance navigation system. It may be visualized as a huge wheel, the spokes of which are courses extending from the hub out to line-of-sight distance around the station. Ideally, an infinite number of these spokes are available to the pilot for his selection [2]. The courses are perfectly straight and pass through a central point at the hub, with each course in the form of a vertical plane.

Operation of the VOR is based on comparing the phase difference between two 30 Hz signals transmitted from a ground station in the 112-118 MHz frequency band and received on a VHF receiver in the aircraft. One transmitted signal R (the reference-phase signal) has a constant phase regardless of the position of the aircraft. The phase of the second transmitted signal V (the variable-phase signal) is a function of the bearing of the aircraft from the transmitting station. The two signals are in phase when the receiving position is due north (magnetic) from the facility. For any other bearing, the phase difference between the two signals is equal to the bearing of the aircraft from the ground station.

Basically, the VOR ground transmitter facility has two antenna systems. One, the "carrier antenna," radiates a radio-frequency signal with an omni-directional pattern (in the horizontal plane), amplitude-modulated (AM) with a 9,960 Hz subcarrier, which is in turn frequency-modulated (FM) at 30 Hz (the reference-phase signal). This carrier is also

simultaneously modulated, either with the tone identification from the facility, or by voice for communication purposes. The second antenna system, the "sideband antenna," when combined with the first, produces a space modulation rotation at 30 Hz, the variable-phase signal. A monitoring system checks the station output continuously and automatically and, in the event of transmitter trouble, actuates warning alarms.

In the aircraft, the composite signal is received on a conventional VHF receiver. After demodulation, the two audio signals are separated by suitable filters (30 Hz and 9,960 Hz), and the 9,960 Hz subcarrier is passed through a discriminator to recover the 30 Hz reference signal. The phase difference between the two 30 Hz signals is then measured to determine the bearing of the receiving point from the facility. The principle of VOR receiver can also be represented in Figure 1.

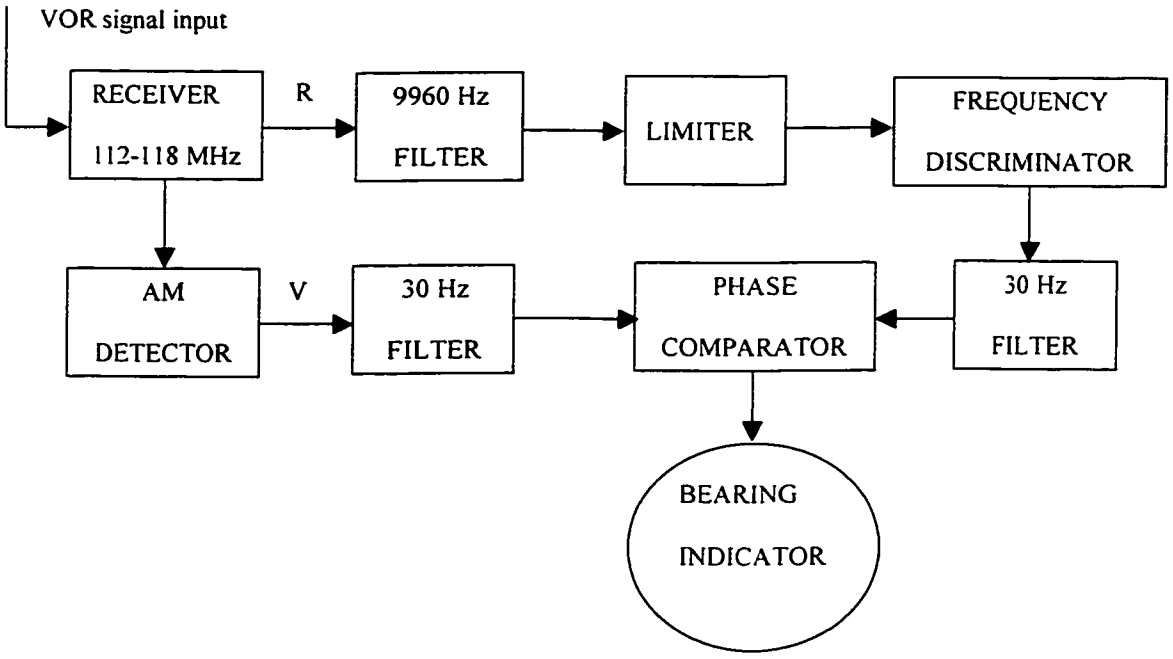


Figure 1: Block Diagram of VOR Receiver

Distance-measuring Equipment (DME) [5] is a system combining ground-based and airborne equipment to measure the distance of the aircraft from the ground facility. DME is used primarily for fixing an aircraft's position, approaching an airport, avoiding protected air or ground space, holding at a given position, or figuring ground speed. The DME ground station is usually co-located with other navigation or distance-measuring systems. DME operating principle is shown in Figure 2.

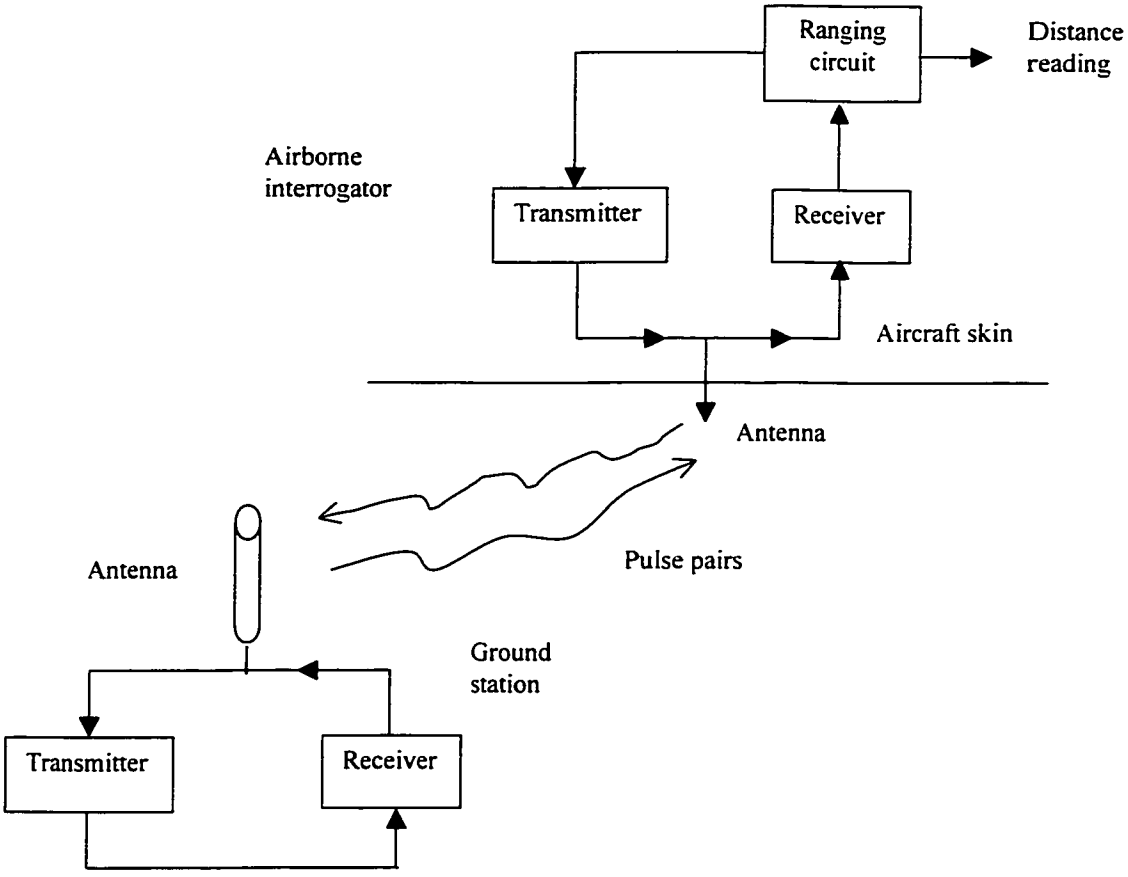


Figure 2: DME Operation

The airborne DME consists of a transmitter-receiver, a control unit, a distance indicator, and an antenna. The ground-based portion of the DME system consists of a transmitter-receiver and an antenna, but operates only on a single frequency.

The transmitter section of the airborne equipment contains the circuits to generate, amplify, and transmit interrogating pulse pairs. The receiver section contains the circuits required to receive, amplify, and decode the received reply pulses. Computation circuits then determine the validity of the reply pulses and calculate the distance by comparing the elapsed time between transmission and reception.

1.1.2.2.1 Space-based Navigation Systems [29]

Global Positioning System (GPS) has been a worldwide navigational aid with an accuracy of a few meters. The basic principle is that a number of satellites in orbit each radiate a series of precisely timed radio signals. The user notes the time at which the signals are received and from the delay of each due to transmit time and knowledge of the position of the satellite at the moment of transmission, calculates the distance from each and thus the position.

GPS can achieve position accuracy of 16 meters in three dimensions, the correct time within one-millionth of a second, and the user's velocity to the nearest one-tenth of a meter per second. GPS employs a network of 21 satellites (plus 3 operational spares) in six orbital planes, each broadcasting precise time and location information. The satellites operated in circular 20,200 km orbits at an inclination angle of 55 degrees, and with a 12 hour period. They are precisely arranged so that a minimum of 5 satellites is in view at all times at any single location in the world.

GPS consists of three segments: the Space Segment, a constellation of satellites circling the earth; the Control Segment, a series of ground stations that monitor the

satellites and transmit corrected position and time data to the satellites; and the User Segment, all sea, land, and air equipment that receive the satellite signals and calculate user position, time, and velocity. Each satellite transmits two l-band signals (1,575.42 and 1,227.6 MHz). Both signals contain data that include satellite locations; thus, the range to the satellite can be determined with the user's receiver by comparing the time delay between satellites. The precise latitude, longitude, altitude, velocity, and time can therefore be determined for an unlimited number of users.

When the GPS is turned on, the pilot enters an estimate of his present position, velocity, and time. The GPS equipment then begins to search for and track satellites. The data from the satellite signals identifies the satellite number, locates the satellite in space, and establishes system time. The GPS receiver then calculates the range to the satellite by measuring the time of receipt of the signal and multiplying that time by the speed of propagation of a radio signal. This calculation locates the user on a sphere of radius R1, whose center is the transmitting satellite. With the range of one satellite known, a range measurement is made to a second satellite to define a second sphere of range R2. In a like manner, R3 is then determined by measuring the range to a third satellite. Using the three range measurements and elementary geometry, the GPS receiver determines the user's precise position in terms of latitude, longitude, and altitude. Range to a fourth satellite is required to determine the time-offset from the user's crystal clock with respect to the GPS atomic time standard. The velocity measurement is determined by counting the Doppler shift from the GPS center frequency.

The effects of GPS on future generations of air traffic management systems and on civil and military users of airspace, both domestic and international, will undoubtedly be quite substantial.

1.2 Flight Management System

Flight management system (FMS) has three specific capabilities relevant to Air Traffic Management (ATM) [3]. It can guide the aircraft accurately along a predetermined path defined by a sequence of two-dimensional (latitude and longitude) or three-dimensional (adding altitude) waypoints. The FMS also can minimize the cost of the flight by selecting optimum speeds and/or altitudes along the predetermined path. The optimization is based on a pilot-selected cost index that weighs fuel cost relative to flight time cost. When the cost of flight time is weighted heavily, the FMS will complete the trip more quickly, burning more fuel in the process. Finally, the FMS has a required time of arrival (or 4D navigation) capability that assures the aircraft will arrive at selected waypoints within small time windows (e.g., within ten seconds of the prescribed times).

Extensive data bases are resident in the FMS, as well as the current flight plan and the wind velocities and air temperatures expected along the route of flight. The FMS provides pitch, steering, and autothrottle commands to the aircraft guidance systems, as well as essential flight data, including estimated times of arrival at waypoints, fuel remaining and fuel flow, current position and speed, and current winds. The FMS integrates the various avionic subsystems, including the navigation sensors (Figure 3). It will automatically tune VOR/DME receivers according to the position of the aircraft and the navigation aid frequencies stored in the database.

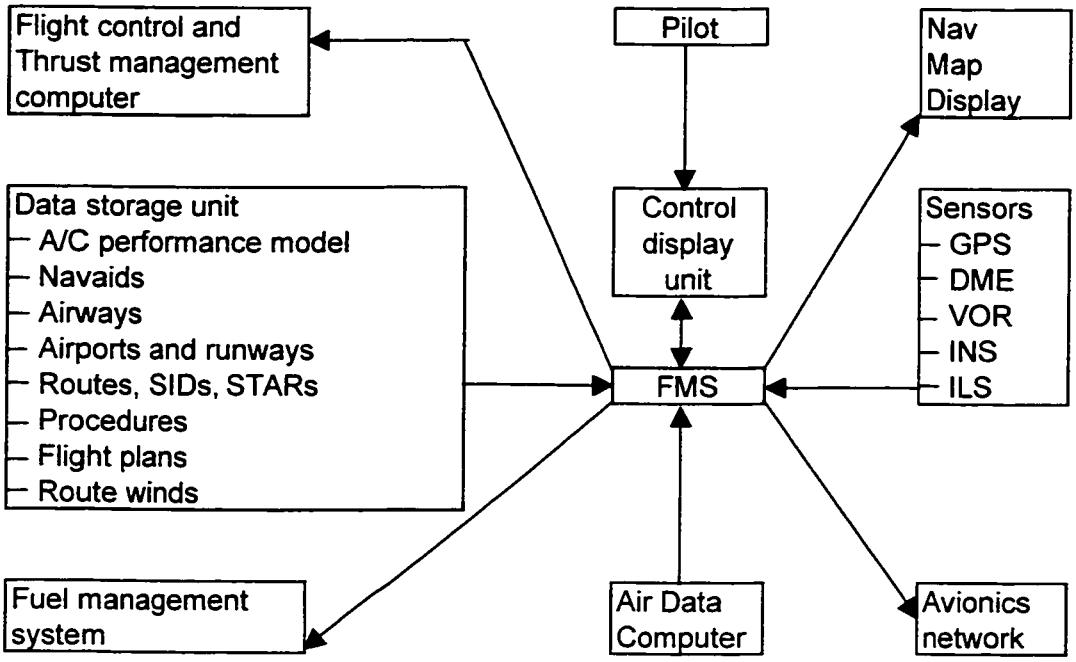


Figure 3: Flight management system [3]

Modern civil transport airplanes are often equipped with a Flight Management Computer (FMC), typically part of the Flight Management System (FMS), which optimizes the performance and/or flight path of the aircraft in terms of some parameter such as flight time or the plane fuel cost. The most common interface between the crew and the FMC is a Control Display Unit (CDU) shown in Figure 4. In addition to the alphanumeric keyboard, there are special-purpose keys that support the flight-planning and management processes. There are typically three CDUs in the aisle stand of the cockpit, one each for the captain and first officer in the front of the aisle stand, and one at the rear of the aisle stand intended primarily for use by maintenance personnel to operate the on-board maintenance system. The third CDU also is a backup for the other two.

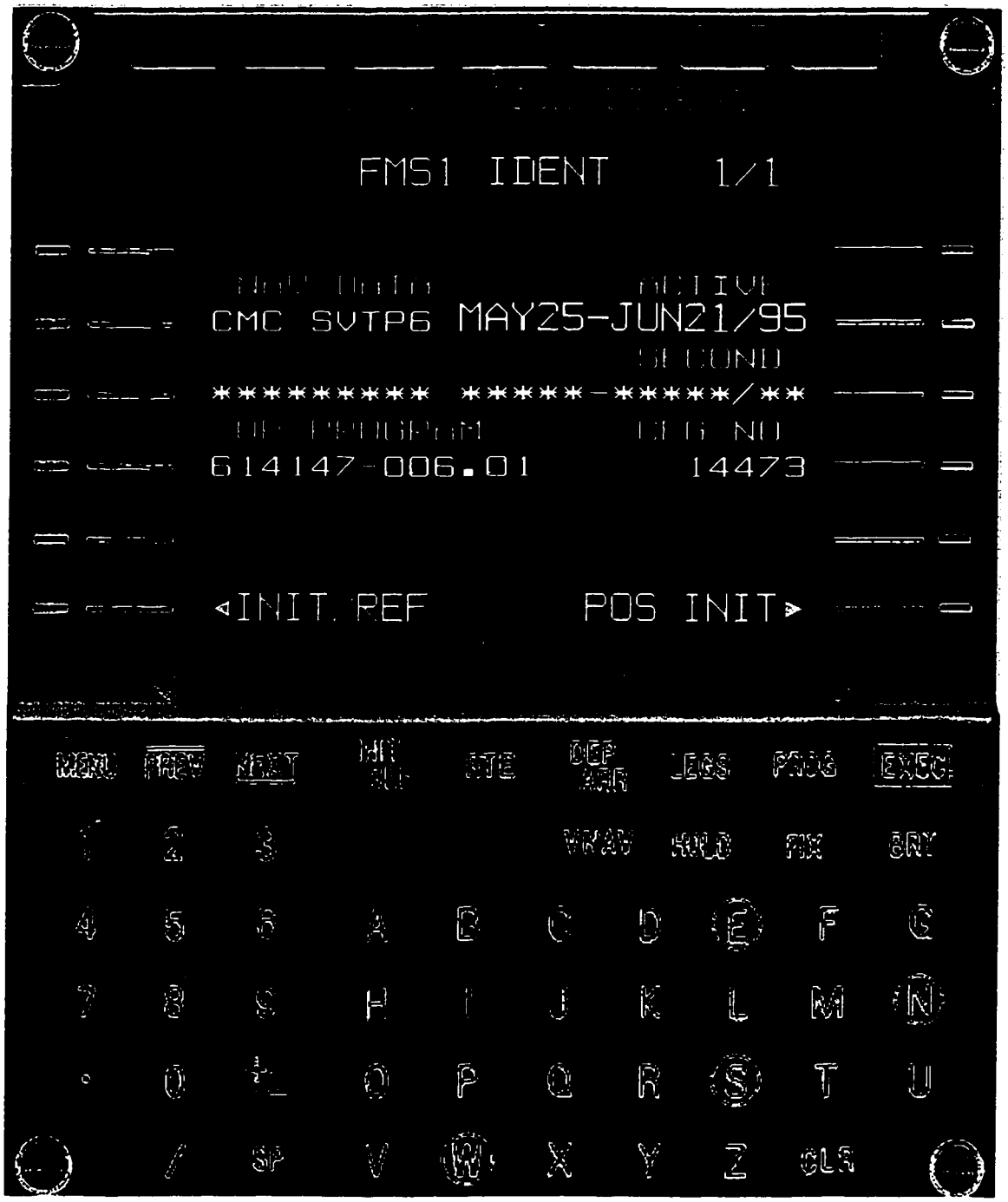


Figure 4: Control display unit (CDU) [30]

1.3 Test Bed Review

Over the last 10 years, hundreds of hours of flight simulation have been expended in the US, Canada and Europe in efforts to revise the handling quality requirements for flight management systems. In the same time, very few flight simulation experiments addressing handling qualities at FMS dynamic interface have been reported. There has been no significant use of flight simulation of the dynamic interface in test and evaluation programs [21]. A NASA Dryden Flight Research Center program explored a practical application of real-time adaptive configuration optimization for enhanced transport performance on an L-1011 (Lockheed Corporation, Burbank, California) aircraft in 1998. This adaptive configuration optimization system is better than benefits available with a preprogrammed flight management system [31]. However, the testing was very expensive since they were all performed in actual aircraft.

Most of dynamic interfaces are software-based. As an example, the mobile spaceplane simulation facility being developed for Phillips Laboratory by Boeing provides a test bed for assessing operations before prototype vehicles and their support systems are built [22]. Also, Boeing reported the project "Open Control Platform for Uninhabited Air Vehicles" in 2001. The Open Control Platform (OCP) will be integrated with popular and useful hybrid system control design tools such as Matlab/Simulink, and Ptolemy II to enable the rapid design, simulation, and the test of reliable embedded software for unmanned air vehicles [32]. This will lower cost and reduce effort as operations-related problems are uncovered and resolved through simulation before hardware prototypes are built. However, this design process is not an "end-to-end" process since there is no integration of the avionics hardware.

CMC Electronics Canada Incorporation, as the world leader of FMS design, was promoting a project of dynamic test bed (DTB) to reach new stage in order to get more credibility and replicate the real world. This DTB will have the capability to interface and communicate between the high-fidelity six-degree-of-freedom flight simulator and the CMA-900 flight management system of real time. DTB as one of the important steps in FMS development is its testing in the dynamic environment of the specific aircraft type. While such testing is mostly performed in the actual aircraft, a more economical approach would be carry out the bulk of the testing using a laboratory-based simulated environment or test bed, for which the basis is a reconfigurable, real-time flight dynamic model. This distributed approach to the development of FMS is based on a modular concept; simulator modules are allocated to individuals PCs, which communicate via Ethernet packets transmitted at a certain rate. Similar approach is reported by a few research organizations throughout the world including the engineering simulator in Cranfield University, College of Aeronautics, UK [33].

1.4 Thesis Objectives

This thesis is written for the navigation engineer, whether user or designer, who is concerned with the practical application of newly developed FMS. It is also the intention of the author to provide enough information for future students to carry on similar task in this field.

This thesis will focus on the author's contribution to the joint project sponsored by the CMC Electronics Canada Inc.(CMC) and the Center for Industrial Control (CIC) of Concordia University – Dynamic Test Bed for Flight Management System, including

the description of the real-time software architecture design, system software implementation and breakout box design. The author will describe not only procedures followed in carrying out the project, but also provide salient insight for each component of the test bench, particularly from the hardware point of view.

The first chapter presents a historical overview of navigation technology relative to the DTB and its mission for civil and military aircraft. Chapter 2 gives the whole picture for the DTB by means of purpose of DTB, hardware components and software architecture for the system. Chapter 3 continues to explore the detailed explanation of the hardware components by means of ARINC 429-PC16 card interfacing, PCI-6025E card interfacing, breakout box design, and the general principles and application of the FMS. Chapter 4 discusses some software assistant tools to monitor flight simulation parameters in real run time performance. Chapter 5 covers all stages for verifying the system by means of loop-back, Lateral Navigation (LNAV) and Vertical Navigation (VNAV). The last chapter summarizes all work along the DTB and provides some recommendation for the future work of the coming phase.

CHAPTER 2

DYNAMIC TEST BED OVERVIEW

2.1 Purpose of the Dynamic Test Bed

The primary function of DTB is to evaluate and test the dynamic performance of the CMA-900 Flight Management System (FMS). The DTB has as core software (VPI FLSIM version 7) of a flight simulator (B737-200) model to simulate with all interfaces of the FMS, including a flight model, environment model, Air Data Computers (ADC) model, GPS model, etc. DTB is in effect a simplified aircraft simulator, which recreates the flight simulation signals for FMS. The FMS will act as if it is in an aircraft. "Dynamic" means all signals changing simultaneously as they do in a real aircraft. Therefore, DTB gives testing of the FMS in a real-time environment and provides a useful tool to develop the new VNAV capabilities on CMA-900 FMS for CMC.

In order to complete the tasks, DTB provides the following functions:

a) Evaluation of the CMA-900 Lateral (LNAV) and Vertical Navigation (VNAV) accuracy requirements.

b) Closed loop testing of the FMS for the development, verification and demonstration of LNAV and VNAV requirements.

c) Crew interface evaluation in a dynamic simulation mode.

d) Gradual introduction of DTB as a certificate tool for Transport Canada requirements.

2.2 Hardware Components

The DTB consists of two PC computers: the Simulation PC and the Interface PC. The Simulation PC holds the simulation software (FLSIM 7.0) which is provided by Visual Prototypes Inc. (VPI). The Interface PC holds the two interface cards A429-PC16 and PCI-6025E, that takes the flight information generated by the simulation PC and sends it to the FMS in the ARINC and discrete protocol, respectively. The general layout of hardware components is shown in Figure 5.

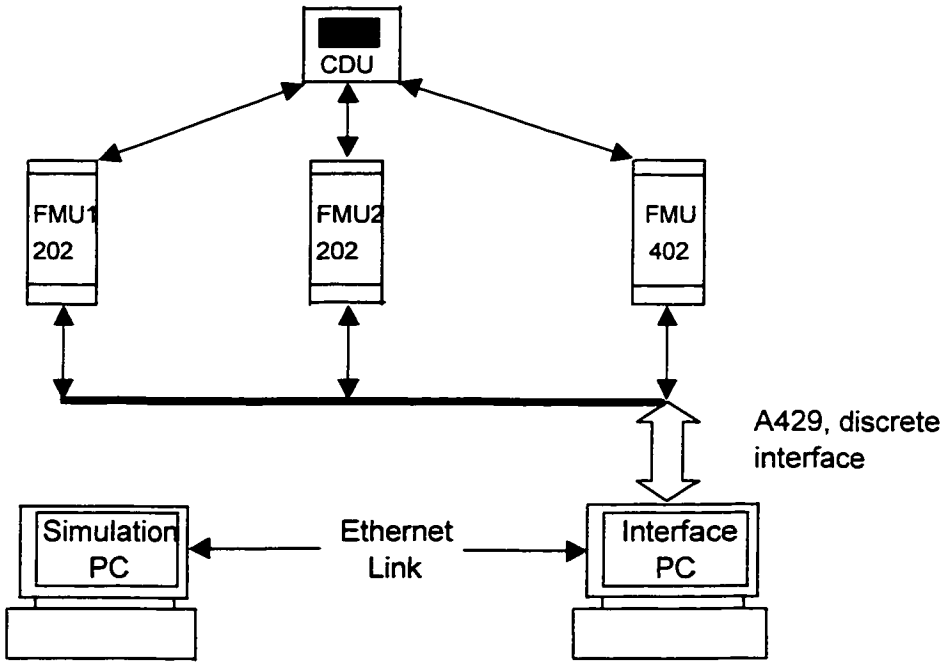


Figure 5: Hardware components of DTB

Figure 5 shows that the DTB has the flexibility for interfacing two kinds of CMA-900 FMS. Model 202, which was originally donated by CMC, has no VNAV

functionality and is used for initial testing at the Concordia Aerospace lab; model 402, still under VNAV development but presently possesses Glide Slope functionality mode.

Hardware block diagram is shown in Figure 6, which offers more detailed information about the hardware set-up. The DTB hardware includes two computers, one A429-PC16 card, one PCI6025-E card, one breakout box and the cables to connect these to each other and the FMS being simulated. The Simulation PC, with a network card installed, performs most of the simulation and display tasks during execution of the simulation. The Interface PC also has a network card to communicate with the Simulation PC during the simulation, and it holds one ARINC429 card and one discrete I/O card. The interface PC runs all interface software.

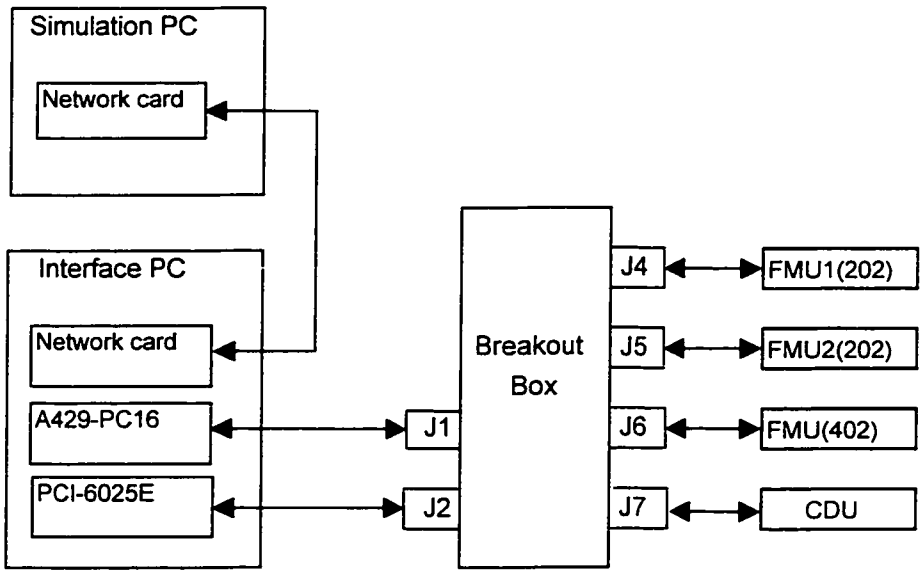


Figure 6: Hardware block diagram of DTB

J1, J2, J4, J5, J6 and J7 are connectors providing the interconnect points for all wiring between the FMS and the DTB. The wiring within is semi-permanent, thus making changes possible. Another connector J3 inside the breakout box simply provides common joint (node) for all avionics.

2.3 Software Architecture

2.3.1 Simulation Software

The simulation software includes real time software, the operating system being Windows NT 4 (work station) and the compiler is Microsoft Visual C++ (version 6.0). The simulator software overview is shown in Figure 7.

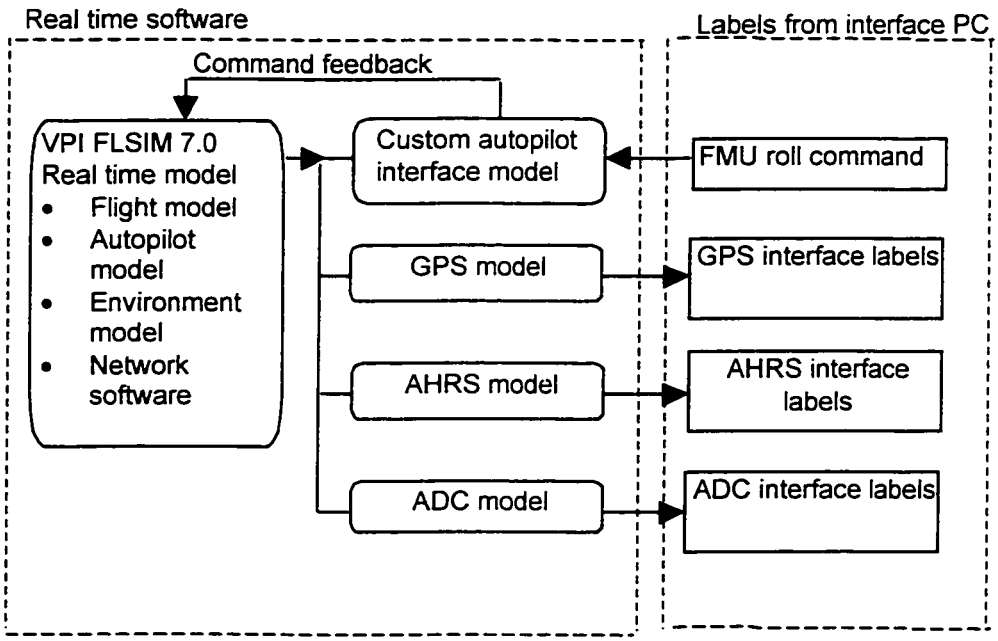


Figure 7: Simulator real time software overview

The real time software contains the core simulation software (VPI FLSIM model) which includes the flight model, autopilot model, environment model and network software. The network software bundles all interface labels, which contain all flight performance information such as Heading and True Air Speed in ARINC words format, and transfers them to and from the interface computer. The ARINC words format will be presented in Section 3.1.8 in this thesis.

The custom autopilot interface model will act on commands from a Mode Control Panel (MCP). Unlike an autopilot in a real aircraft, DTB has the following custom autopilot modes:

- Speed hold mode (selectable speed)
- Heading hold mode (selectable heading)
- Altitude hold (selectable altitude)
- Vertical speed mode (selectable vertical speed)
- LNAV mode (roll command from a FMU)
- VNAV mode (vertical deviation from a FMU)

The GPS model takes basic aircraft position information and maps it into A429 words that are transmitted by the GPS.

The AHRS model receives information from the flight model and sends the True Heading and Magnetic Heading to the FMU via A429 words.

The ADC model takes information from the flight model, the environment model and the altimeter barometric settings and transmits A429 words in standard ADC format.

2.3.2 Interface PC Software

2.3.2.1 Synchronization Mechanism

For the system to implement synchronization, the basic concept of multithreaded programming is introduced. A process is a running program that has its own memory, file handles, and other system resources while a thread is an individual process that contains independent execution paths. Threads are execution streams and form the basis of multitasking [10]. That is to say, each program that runs in Windows has its own main thread to launch the program, and which may also start additional threads. Each new thread has its own procedures and executes the code in that procedures at the same time that other threads are working. Threads are managed by the operating system, each thread having its own stack. In other word, many threads can access the same global variables, even a single function.

There are two kinds of threads in Windows, user interface threads and worker threads. A user interface thread has windows and its own message loop which sends and receives messages for user's application. A worker thread has no windows, and therefore it is relatively easier to achieve the synchronization between the FMS performance and flight simulator reaction in DTB since all worker threads have the same characteristics.

In the system referred to in Figure 8, the main function defines an instance of the *DTBExecutiveThread* first, all other working threads beginning in the order *DTBVPIIOThread*, *DTBArincIOThread*, *DTBVADIOThread* and passing data to the double buffered container. After performing their functions, they will stop until being woken up at every 30 Hz period which is predefined from the flight simulator. At the beginning of each cycle, the *DTBExecutiveThread* first copies everything from the

writable layer into the readable layer of the double buffered container, then it will wake up the threads in a fixed order. The *flags* (logic TRUE or FALSE) in each thread are used to synchronize all of the working threads in the double buffered container when updated by the *DTBExecutiveThread.*

2.3.2.2 DTBExecutive Workspace

The Interface PC software takes all flight simulation information from the Simulation PC via a network packet and transfers it to the interface cards smoothly in real-time. This module controls the ARINC-429 and discrete information, the software being put under the workspace of "*DTBExecutive*". The design is described at the Class/Object level and the architecture is Object Oriented. The software architecture consists the executive workspace which runs on the Interface PC housing the IO cards, and the VPI link & interface running on the Simulation PC along with the VPI flight simulator. The "*DTBExecutive*" software structure can be seen in Figure 8. The solid line format represents control signals; the dash line format is used to show data signals.

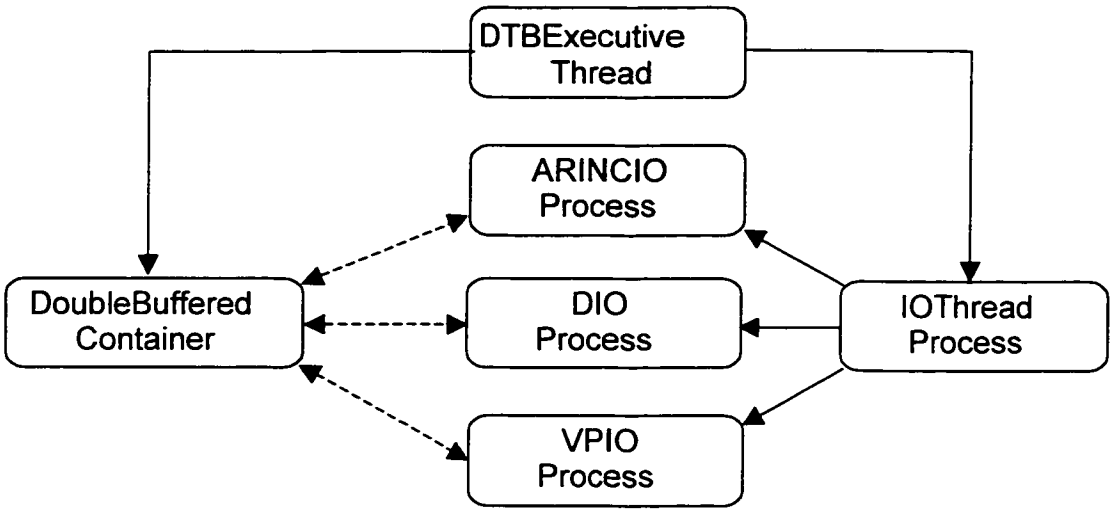


Figure 8: DTBExecutive Components

The main *DTBExecutiveThread* object is used to execute the rest of the objects in the workspace of "*DTBExecutive*". It runs periodically (at 30 Hz), and in each cycle of execution, it invokes operations in the other objects of this component.

The *IOThreadProcess* object is a generalized interface that encapsulates the general functionality that must be provided by every one of the IO card processes including performing the IO, waiting to be awaken by the executive thread, and updating the double buffered container data.

The *ARINCIOProcess* object implements all the operations needed to control, transmit, and receive data to and from the ARINC 429 cards. The thread associated with this object is triggered periodically to continue its execution by the Executive Thread object. When it is triggered, the *ARINCIOPProcess* object performs the necessary checking for available data in the ARINC buffers to be read, reads available data into the double buffered container, and transmits data from the double buffered container to the FMS via the ARINC channels.

The *DIOProcess* object implements all the operations needed to control, read, and write digital and analog data from/to the National Instruments IO card. The thread associated with this object is triggered periodically to continue its execution by the Executive Thread object. On being triggered, the *DIOProcess* object performs digital/analog read/write operations with the National Instrument card and updates the corresponding fields in the double buffered container.

The *VPIOProcess* object is a TCP/IP client-server for communication with the other component running on the VPI Simulation PC. The thread associated with this object is triggered periodically to continue its execution by the Executive Thread object.

When it is triggered, the *VPIOProcess* object sends data from the double buffered container to the VPI simulator and receives the data sent by the VPI simulator into the double buffered container.

The *DoubleBufferedContainer* object implements a double-buffered system bus. Double buffering enables access synchronization between the various threads in the software architecture. The first buffering layer in this container is available at any time for any thread to read data from while the second buffering layer is available for the other threads to perform synchronized write operations into it [11]. The Executive Thread periodically locks the two buffering layers and copies the second one into the first before releasing the mutual exclusion lock. The purpose of double buffering is to cancel the effect of the sequence of execution of the threads on the performance of the whole system and to provide a standard synchronization mechanism according to a validated real-time software design pattern.

2.3.3 Communication Tool

Visual C++ Transmission Control Protocol / Internet Protocol (TCP/IP) client-server two–way communication module was designed for transferring data between the Simulation PC and the interface PC. This is a 32-bit synchronous Winsock application and thus supports multithreaded programming.

There are three classes in this module- the main class *CblockingSocket*, two helper classes *CsockAddr* and *ChlockingSocketException*. The *CblockingSocket* was designed for synchronous use in a worker thread, and embodies the major functionality of the module. The *CsockAddr* achieves conversion between network byte order (big

endian) and host byte order (little endian). The *CblockingSocketException* handles the exceptions on errors that occur and time-outs thrown by the *CblockingSocket* when sending or receiving data [9] [46].

2.3.4 Graphic User Interface (GUI)

There are four major GUIs developed in the DTB, namely Simulation control, ARINC labels, DIO and Channel Switch. The Simulation control GUI contains all simulation control features such as reposition, freeze, recording and graphing, aircraft flight controls settings and Automatic Flight Control systems. The feature of the MCP GUI is presented in Figure 9. ARINC labels GUI is used to trace down the real time performance of the simulation parameters in standard ARINC format and their engineering values. DIO GUI is used to check up the status of the simulation discrete signals. Channel Switch GUI is used to provide the flexibility of changing different channel setting for the existing ARINC property file in the specific A429-PC 16 card.

The author has contributed to the development of the last three GUIs in the DTB and the detailed description will be given in Chapter 4.

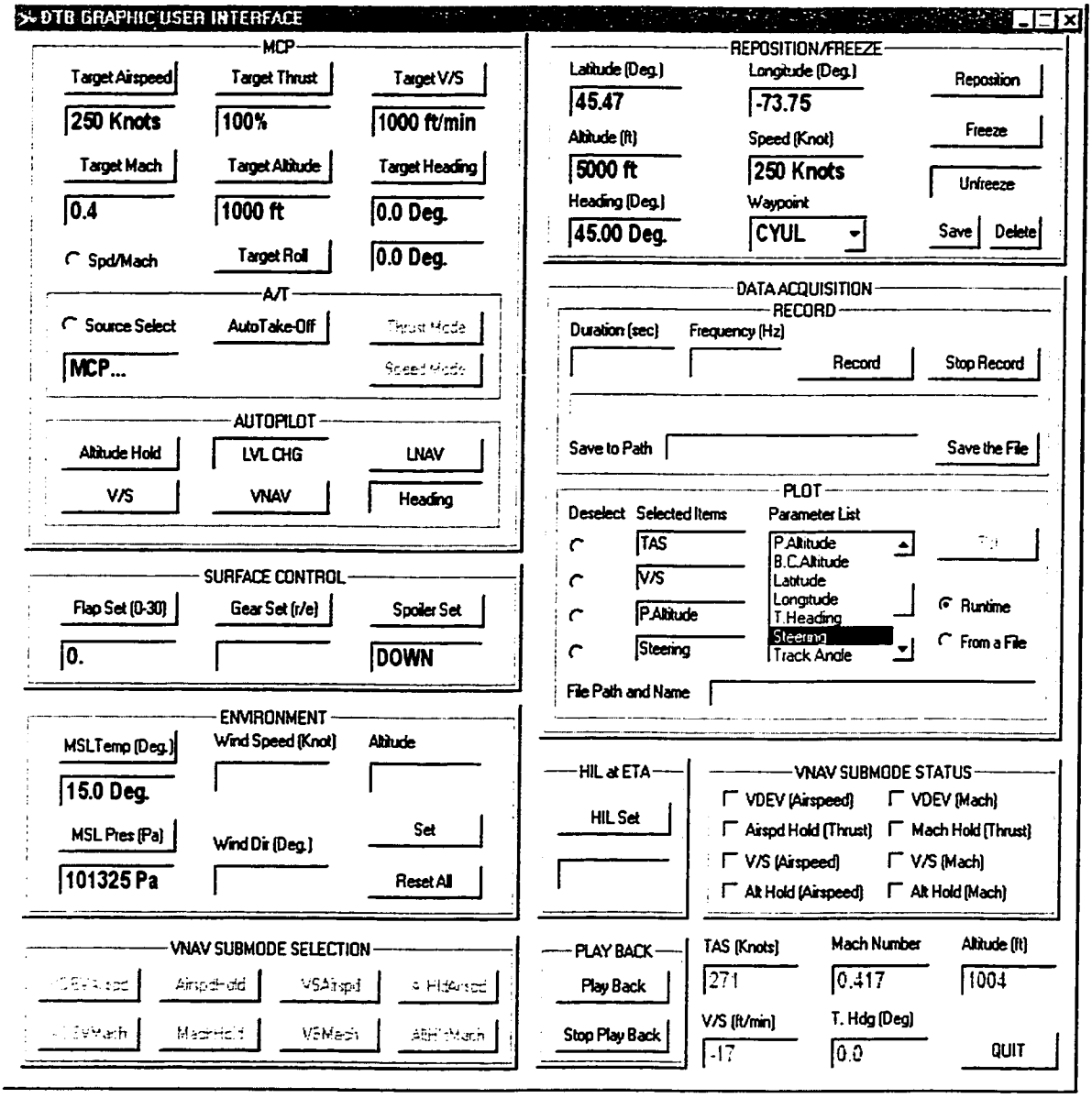


Figure 9: Simulation Control GUI

CHAPTER 3

HARDWARE INTEGRATION

The technologies that dominate systems design shift with hardware miniaturization and concomitant architectural integration strategies. Years ago, systems engineers relied upon functional encapsulation in dedicated hardware units (subsystems) and standard communication channels, especially ARINC 429. Specialized computer hardware teams designed subsystems and software engineers often formulated requirements after hardware design. Now and in the future, highly integrated systems designs blur the distinction between system and subsystem. Therefore, the systems engineer should have background not only in software technology area, but also in data buses and processors.

3.1 ARINC 429-PC16 Interface

3.1.1 ARINC Standards

Advances in microelectronics, information processing, and displays, and the integration of those technologies into the modern flight management system, are reaching the stage when almost any operational demand can be met. While technically and economically beneficial, the integration philosophy can also make life difficult for some airline buyers who may want to make their own choice of equipment, for example, to maintain commonality with systems already in use or to protect their national industry. National and international standards, such as those recommended by Aeronautical Radio, Inc.(ARINC) and other organizations, all contribute to interface definitions, interference

management, safety, materials use, tests, and other fields and define the operational, dimensional and signal interface characteristics to ensure interchangeability between equipment of different manufacturers and compatibility with other equipment [4].

The ARINC organization is the technical, publishing and administrative support arm for the Airlines Electronic Engineering Committee (AEEC) group. The purpose of the standards published by this organization is to meet the operational and technical requirements of individual airline users, while establishing the most general solution so that a market force may develop. ARINC standards are currently used in military avionics primarily for special mission and transport category military aircraft. However, as ARINC standards begin to address integrated avionics, they are likely to have impact on fighter / attack avionics. This relationship is also dual track, as military hardware and software technologies find their way into transport avionics [8]. ARINC Specification 429 is a serial data bus used for point to point communication. Multicast operation is also possible with a line for each receiver, but two-way communications (full duplex) is not possible in this manner. In other words, transmission of information occurs from a port on the Line Replace Unit (LRU) that is designated for transmission only. Similarly, receive ports are used for receive only, and information cannot flow into a port designated for transmission. This is the basic definition of a simplex bus.

Data transfers are clocked at either 100,000 bits per second or 12,000-14,500 bits per second with a tolerance of 1%. This data bus functions most commonly for transmitting navigation information. An attachment to ARINC 429 contains a digital interface standard that provides data word formats for radio navigation information such as frequencies for Very High Frequency Omni-Range (VOR), Instrument Landing

System (ILS), Automatic Direction Finder (ADF), and transponder, as well as Distance Measuring Equipment (DME).

3.1.2 A429-PC16 Design Review

The design of the A429 incorporates an open systems philosophy. The A429 is a generic processing engine that can be configured through various application programs. Processing for each of sixteen channels is performed independently through host-defined Transmit, Receive, and Monitoring data structures.

The A429-PC16 appears as two independent 8-channel devices, namely Device 1 and Device 2. Device 1 contains Channel 1 through 8; Device 2 contains Channel 9 through 16. Each channel can be configured either as transmitter or receiver, and all the receiver channels being always assigned to the lower channel numbers. Each device has 256K of internal RAM. Figure 10 shows the basic block diagram of the A429 module.

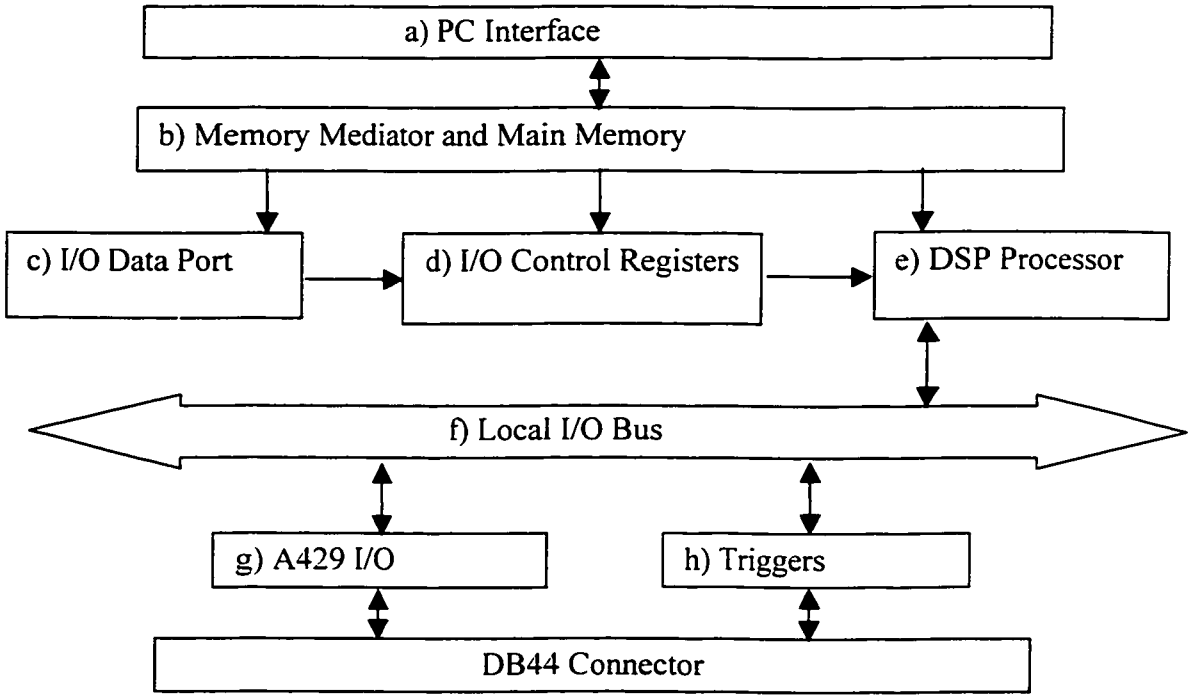


Figure 10: A429-PC Block Diagram

From top to bottom, block (a) shows the host interface where the circuit block contains the 16 data slave interface and the interrupt logic. Block (b) provides a main device memory of 256k byte SRAM and a high-speed bus mediator for memory access between the device and the host. Block (c) provides the organism with access to the device RAM. Block (d) gives the elementary control functions of the A429-PC, which includes firmware start/stop, PC interrupt control, memory base address, memory window control, and memory access control. Block (e) is a general-purpose digital signal processor that executes the firmware code to afford the low level ARINC functionality running under Windows NT operating system. Block (f) is a general local bus, which offers the correspondence for A429 channels, hardware clock, and LED logic. Block (g) provides the transmitter or receiver function for the actual A429 buses. Block (h) provides a coupled latching 48-bit, and a 1-μsec timer, the latching of a 48-bit time stamp being controlled by the Digital Signal Processor (DSP) through discrete logic and ensuring accurate and consistent time marks for each ARINC 429 message.

3.1.3 ARINC 429 Physical Aspects

It is often easier to understand the system if the physical aspects of ARINC 429 are described first. These include transmission media, wiring topology and transmission characteristics.

3.1.3.1 Transmission Media

The transmission media for the ARINC bus is a 78 Ω twisted-shielded pair. Each bus has only one source, but a maximum number of 20 sinks can be connected. A source can handle a maximum load of 400 Ω. A receiver sink must have minimum effective

input impedance of 8 kΩ. The bus length in the most applications is designed for less than 175 feet.

3.1.3.2 Wiring Topology

There are two topologies - Star and Bus Drop, which are shown in Figure 11 and Figure 12 respectively.

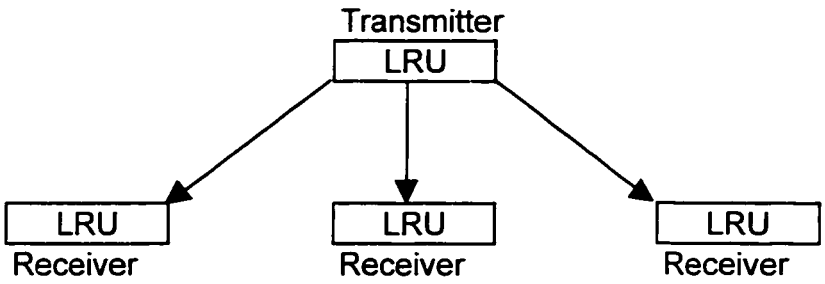


Figure 11: Star Topology for LRU Wiring

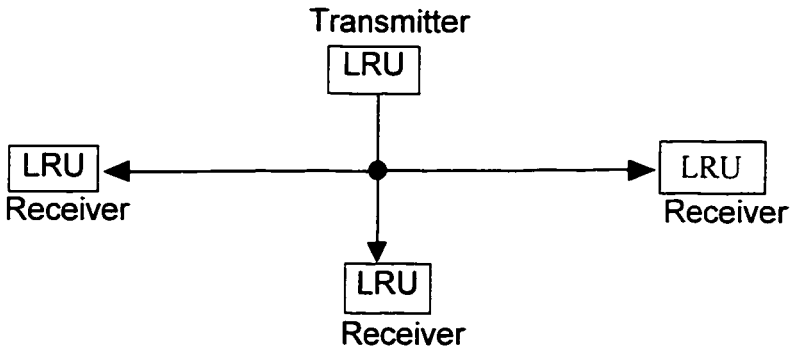


Figure 12: Bus Drop Topology for LRU Wiring

The choice of wiring topology depends on the distance of the sinks to the source. Safety is the advantage of Star topology since each LRU has its own connection to the

source, and any break along bus length results in loss of only one listener. However, the Star topology requires much more wire, thus adding more weight. On the other hand, although the Bus Drop topology uses the same number of connections, there is significant reduction in weight.

3.1.3.2.1 Transmission Characteristics

ARINC 429 has two speeds of operation – 12.5 kHz and 100 kHz. Transmission of sequential words is separated by at least 4 bit times of NULL (zero voltage). This eliminates the need for a separate clock signal wire. That's why this signal is known as a self-clocking signal. Most ARINC 429 transmitters are designed using RC circuit to control the rise time and this implementation is preferred in order to minimize overshoot ringing.

3.1.4 ARINC 429 Architecture Overview

The multi-channel ARINC 429 card (A429) provides simultaneous interface of multiple transmit channels, receive channels, bus monitoring, advanced interrupt services, and high speed host operations. The A429 architecture uses a DSP processor to transmit or receive A429 messages and store them on the monitoring data buses. Low–level processes control all A429 messages within the monitoring buses for protocol verification. High-level processing is responsible for moving A429 packets in real-time to and from the defined data buffers. The general A429 architecture can be specified as Control Registers, Device Management, Transmit Management, Receive Management and Bus Monitoring.

The Control Registers allow the host application program to control hardware and data structures for A429 processing. They include the host command set syntax and key software control registers for managing A429 processing and the hardware registers that are directly accessible from the PC. Device Management includes the processes for setting up and general operation of A429 transmitters and receivers. Transmit Management includes the data structures associated with all A429 messages. Receive Management provides a sophisticated data structure for real-time bus monitoring of A429 traffic. Bus Monitoring gives information on how to receive, and it is useful when the host system has tight processing constraints (relative to ARINC 429 traffic), but must still receive every ARINC 429 message word.

Since the DTB is mainly involved in transmitting and receiving firmware, those two architectures will be described in detail in the following sections.

3.1.5 Address Map

Before going into the detailed description on transmit/receive management, the memory address map is described for the A429 memory RAM. As mentioned previously, each device has 256k bytes RAM. The detailed address map is shown on Figure 13. It is noted that the internal addresses between 00000h and 007FFh and 10000h and 107FFh are reserved and should not be accessed. There are 128k word addresses between 00000h and 1FFFFh, and each 16-bit word has two bytes. Hence, the memory size for each A429 device is 256k bytes.

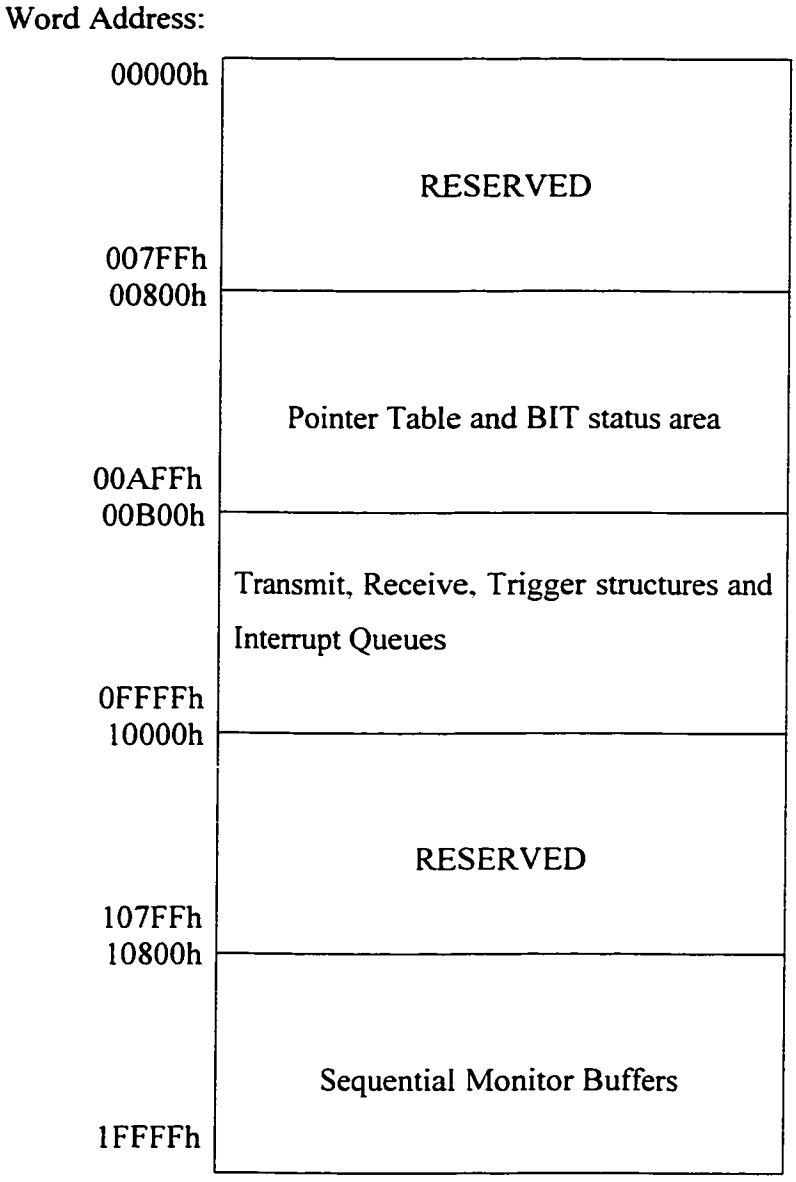


Figure 13: A429-PC16 Memory Map (Device 1)

The bit 0 of the 16-bit I/O Control/Status Register (CSR) selects the lower or upper 128k bytes. In the PC16, CSR1 (word address 0000h) bit 0 can be used to access the full 256k bytes of Device 1 memory and CSR2 (word address 0004h) bit 0 can be used to access the full 256k bytes of Device 2. Software control registers are compound with General Control Registers, System Clock Registers, Interrupt Registers, Trigger Registers, Sequential Monitor Registers, Transmit/Receive Operation Registers and Command Block Pointer Registers. The detailed software control registers address map is shown on Figure 14.

Address	**Name**
880h – 886h	General Control Registers
896h – 899h	System Clock Registers
89Bh – 8A0h	Interrupt Registers
8A3h – 8A6h	Trigger Registers
8A9h – 8ADh	Sequential Monitor Registers
900h – A00h	Transmit/Receive Operation Registers
0B00h - FFFFh	Command Block Pointer Registers

Figure 14: Software Control Registers

It is noted that the address range (900h – A00h) applies for both transmit and receive operation. Each channel control table has a length of 1Fh, in other words, there are 32 words for each channel transmit/receive control block. And those location used for transmitter or receiver is determined by the first word from control block structure – Channel Type. Also, it should be clear that the locations which are not included in Figure 14 are reserved as A429 internal use. Accessing these locations will produce unpredictable results.

3.1.6 Transmit Management

This section explains the definition and management of transmit command blocks and their associated data buffers. Command blocks are linked to allow for flexible and accurate A429 messages transmission.

3.1.6.1 Transmit Control Block Structure

A control block of each transmitter contains 32 words that govern command block processing. Note that, the "words" here have different bit number definitions with standard ARINC format that will be described in section 3.1.9. Figure 15 illustrates the major data structure for a control block. Each cell represents a word.

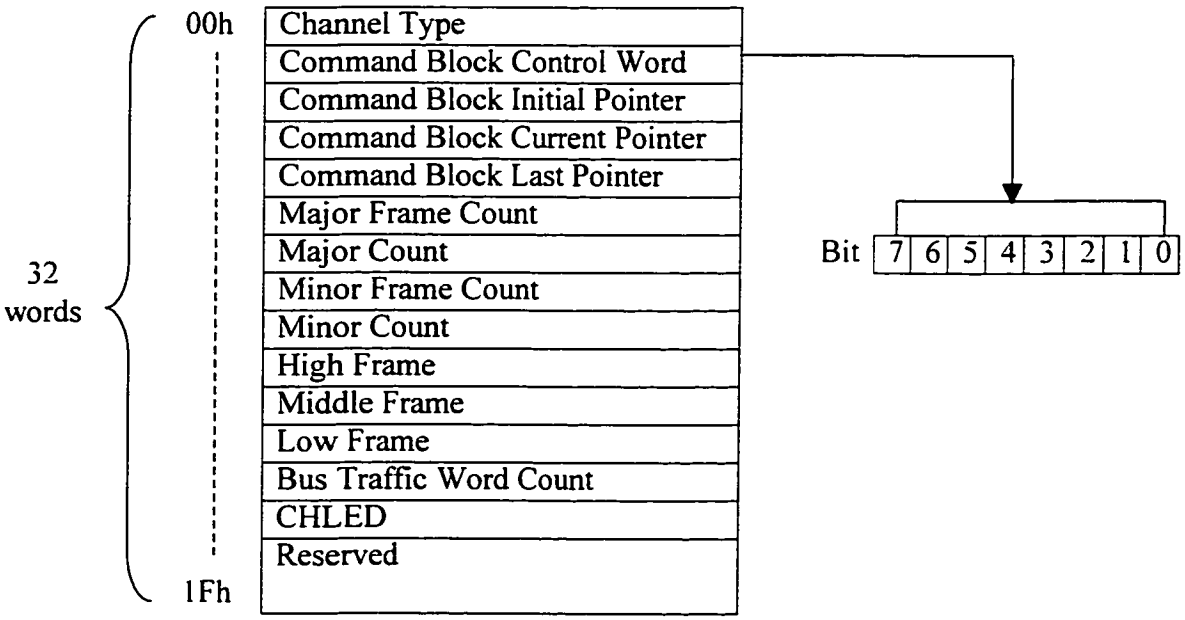


Figure 15: Transmit Control Block Structure

Channel Type This 16-bit word defines the type of channel, it contains FFFFh if it has been predetermined as a transmit-channel and 0000h if it has been designated as a receive-channel.

Command Block Control Word This word governs transmit and command block operations for the channel through the following bits:

- Bit 0 is the Halt bit, which is used to halt processing for the associated command structure when it is set to "1". The firmware completes processing the current command block, halts processing, and then clears this bit.
- Bit 1 is used to present the transmit-speed. For fast transmission (100 kHz). this bit is set to "1"; for slow transmission (12.5 kHz), this bit is set to "0".
- Bit 3 is Channel Wrap bit. The purpose of the channel wrap feature is to allow an operator to receive data on a channel and immediately transmit the same data out on a selected transmit channel. To enable Channel Wrap processing, this bit is set to "1"; for normal operation, as in the case of the DTB, it is set to "0".
- Bit 4 is used to enable or disable Channel Wrap Error Injection globally with "1" or "0" respectively. When masking data, the possible errors come from the word parity, adding a bit and subtracting a bit.

Command Block Initial Pointer (CBIPTR) This pointer location is constantly monitored by the A429's firmware. When CBIPTR is at non-zero, the firmware sets this pointer to zero and executes a chain of command blocks beginning at the offset defined in

CBIPTR. The interface between this pointer and the command block will be described in the next sub-section 3.1.6.2.

Command Block Current Pointer This pointer points to the current location in the command block structure. This pointer is at zero until processing of a command block structure starts. The interface between this pointer and the command block will be described in the next sub-section 3.1.6.2 as well.

Command Block Last Pointer This location is updated when a halt occurs, and indicates the last command block that was executed before the halt occurred.

Major Frame Count This value determines the number of major frames to execute before halting command block processing. For continuous operation, in the case of the DTB, it is set to "0". In time-division multiplexing, a frame is defined as one complete commutator revolution that includes a single synchronizing signal or code. Major Frame is the time period, during which all data of a multiplex are sampled at least once, including one or more Minor Frames. Major frame length is determined as (N)(Z) words, where N is the number of words and Z is the number of words in the longest sub-multiple frame. For more detail about the relationship between major frame and minor frame, see section 3.1.8 with actual example.

Major Count This value indicates the current major frame being processed, and it is updated by the A429 firmware.

Minor Frame Count This value determines the number of minor frames to execute for each major frame.

Minor Count This value indicates the current minor frame being processed, and it is updated by the A429 firmware.

High Frame This register contains the most significant 16 bits of the 48-bit time at which the last minor frame type command was processed, this value being used by the A429 firmware to keep track of minor frame times.

Middle Frame This register contains the middle 16 bits of the 48-bit time at which the last minor frame type command was processed, this value being used by the A429 firmware to keep track of minor frame times.

Low Frame This register contains the least significant 16 bits of the 48-bit time at which the last minor frame type command was processed, this value being used by the A429 firmware to keep track of minor frame times.

Bus Traffic Word Count In this value is the increment for each word transmitted. A word count is provided for each channel.

CHLED, Channel Activity Indicator This value is used by A429 firmware to indicate bus activity. It contains 0000h if there is no bus activity, 00FFh if the channel is active, and FF00h if the channel is transmitting errors.

The rest of the total 32 words are reserved and used for further application under ARINC development.

3.1.6.2 Transmit Command Block Structure

After setting up the control block, A429 transmission commands have to be specified in a series of linked-list command blocks. The length of the command block linked-list is limited only by the available internal 256k bytes of A429 device memory (SRAM). Each byte here has 8 bits according to the internal A429 address range (00000h-1FFFFh). Separate control areas for each of the eight channels allow the A429 device to perform each channel simultaneously. The transmit-register for a channel is pointed at offsets from the base address of that channel's control table. Channel control

tables begin at 00900h and have a length of 1Fh. Hence, the control table for channel 1 occupies 00900h to 0091Fh, the control table for channel 2 occupies 00920h to 0093Fh, and so forth.

Figure 16 illustrates the data structure for one command block linked-list chain. Each transmit-channel has one Chain ID Number, and there could be many command blocks linked to this Chain. Separating different command block depends on the label characteristics, and labels having the same transmitting speed should be grouped into the same command block. To program transmit operations on the A429, a linked-list data structure of command blocks has to be constructed and loaded into A429 memory first, then the 16-bit Command Block Initial Pointer (CBIPTR) has to be programmed with the address of the first word the first command block. When the firmware detects a nonzero value in CBIPTR, it uses this value as a pointer to the first command block of the linked list. It then sets CBIPTR to "0" and begins processing the command block.

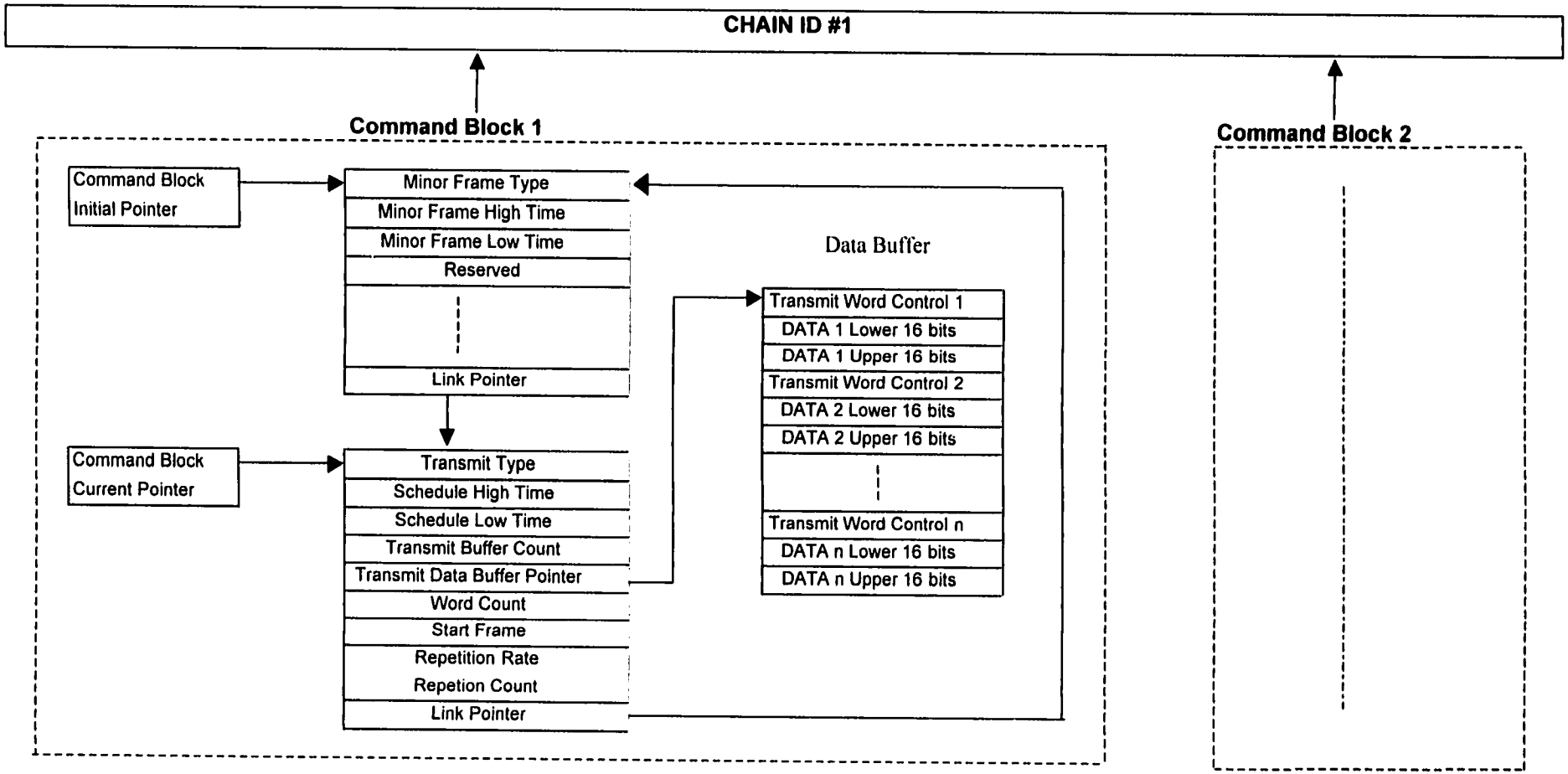


Figure 16: Transmit Command Block Data Structure

Minor Frame Type/Transmit Type This place defines a code for the type of operation and the way of performance. The accessible codes are shown in Table 1.

Table 1: Type Codes

Operation	Control (Bit 15)	Control (Bit 14)	Transmit Block (Bit 13)	Control (Bit 12)	Type Code (Bit 0)
Minor Frame	NO-OP	Interrupt at end of command	N/A	N/A	0
Transmit	NO-OP	Interrupt at end of command	1 = Single 0 = Block	N/A	1

- Bit 15 is set to "1" to skip the command or is set to "0" to process the command.
- Bit 14 is set to "1" to make an interrupt when the command is ended or is set to "0" to produce no interrupts upon end of the command.
- Bit 13 is set to "1" to transmit a single word each time the command block is processed when the next data word in the transmit-buffer is sent. To transmit the whole data buffer each time the command block is processed, set this bit to "0".
- Bits 1 – 11 are reserved as A429 device internal use.

Minor Frame High Time This register is associated with the minor frame type command and contains the upper 16 bits of the time, in microseconds, allotted for a minor frame.

Minor Frame Low Time This register is associated with the minor frame type command and contains the lower 16 bits of the time, in microseconds, allotted for a

minor frame. When the firmware encounters another minor frame, it checks both minor frame high and low times to determine when to start processing the next minor frame.

Schedule High Time This register is associated with the transmit type command and contains the lower 16 bits of the time, in microseconds, that must expire before the transmit command block is processed. The time is referenced from the last minor frame type command.

Schedule Low Time This register is associated with the transmit type command and contains the upper 16 bits of the time, in microseconds, that must expire before the transmit command block is processed. The time is also referenced from the last minor frame type command.

Transmit Buffer Count (TBCNT) This register is associated with the transmit type command and indicates the number of data words to transmit for the associated command block. Each value to be transmitted requires three 16-bit words in Hex. The first word is the control word, followed by the lower 16 bits of the ARINC data word, then by upper 16 bits of the ARINC data word, i.e., the Transmit Buffer Length is equal to 3 times that of TBCNT.

Transmit Data Buffer Pointer This value is associated with the transmit type command and points to the base of the transmit buffer for the associated command block.

Word Count This register is associated with the transmit type command and points to the current location in the transmit data buffer.

Start Frame This register is associated with the transmit type command and indicates which minor frame a transmit type command block will begin executing.

Repetition Rate (REPRTE) This register is associated with the transmit type command and indicates how often to process a transmit type command block after the START condition is met. If the REPRTE is equal to 2, processing of the command block will occur every other time that the associated command block is accessed.

Repetition Count The A429 firmware uses this value to control the Repetition Rate word transmission.

Link Pointer This value points to the next command block in the chain.

Transmit Word Control This is the first word of a three-word block which defines a transmit word.

3.1.6.2.1 Transmit Firmware Operation

During transmit operations, the firmware decodes each block of the linked-list program and takes the suitable action. If a minor frame type is used, the minor frame times are checked and the structure is set up for minor frame operation. If a transmit-type is indicated and schedule times, start, and repetition rates are met, the transmit-buffer is transmitted. The command block structure is continued until a halt condition happens which can be generated in one of these three ways:

- By setting bit 0 of Command Block Control Word in the control block structure to "1".
- By setting the Link Pointer of the command block data structure to "0".
- By completing the specified number of major frames.

3.1.7 Receive Management

In order to access the receiver operation from all sixteen channels of the board, the parameters and data storage structures must be defined first. Table 2 defines the data storage structure required for A429 PC16.

Table 2: Data Storage Structure

Data Storage Structures	Description
Current Value Buffer	Contains the latest received data words for a specific channel. They are arranged by label or both SDI and label.
Current Time Buffer	A time stamp buffer can be set up to provide a current time stamp for each received word.
Channel Sequential Monitor	Stores data received on a specific channel.
Global Sequential Monitor	Stores data from all receive-channels.

It should be noted that either the Channel or Global Sequential could be selected. Even though the DTB deals with Current Value Buffer and Channel Sequential Monitor only, it would still be a good idea to go briefly through all components starting with the receive data structure.

3.1.7.1 Receive Control Block Structure

The receive control block structure is comprised of 32 words. They are mainly the channel type word, the receive control word, the current value buffer pointer, the current time buffer pointers, the filter pointers, bus traffic word count, and channel activity indicator. Figure 17 illustrates the receive control block structure.

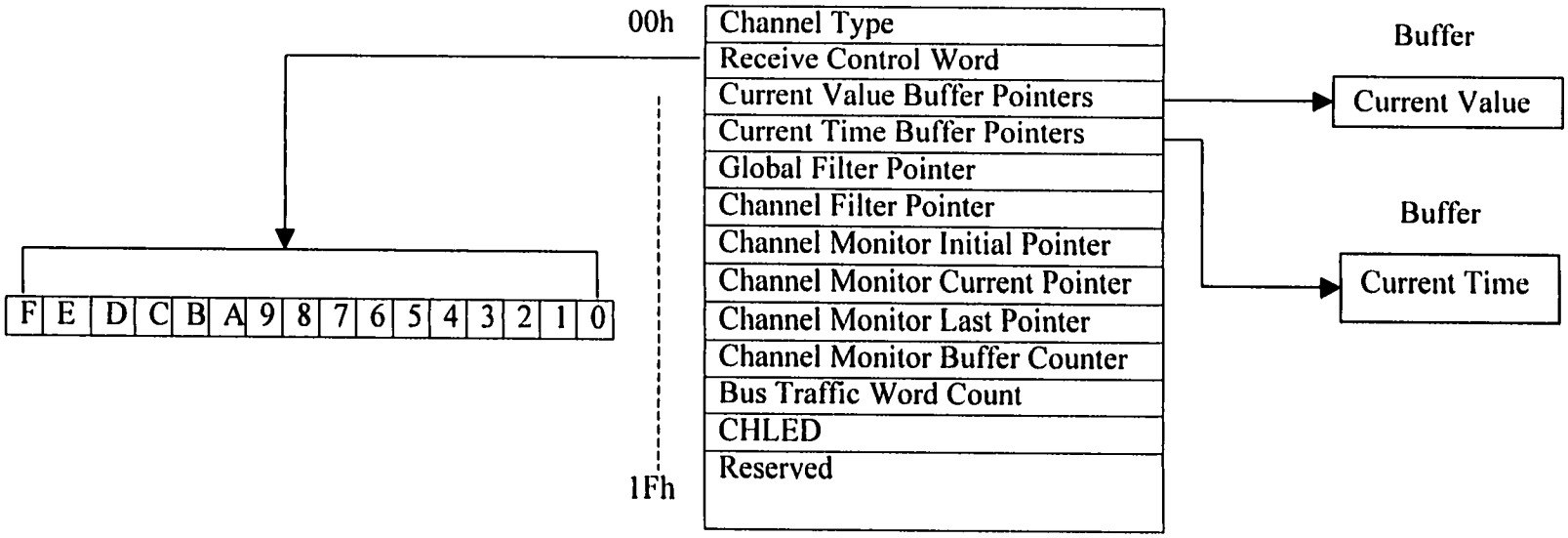


Figure 17: Receive Control Block Structure

Channel Type This word specifies the type of channel. The word contains FFFFh if the channel has been designated as a transmit channel and 0000h if designated as a receiver channel.

Receiver Control Word This word controls receiver operations through the following nine bits:

- Bit 0 is the Run/Halt bit. To start receive operations for the specified receiver, set this bit to "1". To halt receive operations, set this bit to "0".
- Bit 1 is Receiver Speed bit. To process 100 kHz ARINC data, set this bit to "1". To process 12.5 kHz ARINC data, set this bit to "0".
- Bit 2 is Interrupt on Error bit. To generate a host interrupt when a received data word contains an error, set this bit to "1".
- Bit 3 is SDI/Data bit. To sort received data by Source/Destination (SDI) bits and label (bits 0-9 of ARINC word), set this bit to "1" and to sort received data by label only (bits 0-7 of ARINC word), set this bit to "0". Detailed description of ARINC word will be presented in section 3.1.9.
- Bit 4 is Channel Monitor Halt bit which is set to "1" when halting the receiver data buffer. When the channel monitor is needed, the firmware resets this bit to "0".
- Bit 5 (Restart Channel Monitor bit) is set to "1" in order to restart the channel monitor after a halt has set.
- Bit 6 (Force Monitor Swap bit) is set to "1" for forcing a monitor buffer swap.
- Bit 7 (Interrupt on Swap bit) is set to "1" for producing an interrupt to the host when a monitor swap occurs.

- Bit 8 is Service Current Value Buffers bit. To cause the current value and time pointers to swap, set this bit to "1". The swap allows the current value and time buffers pointed out by their respective pointers to be read.

Current Value Buffer Pointers These 16-bit words point to the base offset of the current value buffers. Since per ARINC specifications, the label (least significant 8 bits) has a reversed bit order, labels are bit swapped prior to being stored in the current value buffer. Hence, for the 32-bit ARINC words arranged by label (least significant 8 bits) only, the word will be stored at offset times 2 for lower and upper 16 bits of ARINC data. In other words, the current value buffer size is a 512 ARINC word buffer (i.e., 2×2^8). Considering the words arranged by both SDI (bit 9 and 10 in ARINC word) and label number, the current value buffer size now becomes a 2048 ARINC word buffer (i.e., $4\times2\times2^8$). After the pointer is activated, the current values in the buffer can be read in ascending order from low data to high data. It should be clear that the current value buffers must be in device memory between B00h and FFFFh.

Current Time Buffer Pointers These words point to the base offset of the current time buffers. When a word is received, a 48-bit time stamp is stored in the active current time buffer. The time stamp consists of three 16-bit words, namely, low time, middle time and high time. The current time value buffers must be in device memory between B00h and FFFFh as well.

Global Filter Pointer This register contains an offset to a table of 128- or 512-ARINC words arranged by label, or both SDI and label. Entries in this table govern interrupts and sequential monitoring for each possible receive label, or both SDI and label. Entries are placed on 8-bit boundaries, even labels occupying the eight least

significant bits and odd labels occupying the eight most significant bits. It is noted that the global filter pointer is 16-bit represented, in order to convert to standard 32-bit ARINC word, 0.5 factor is taking into account for the buffer size (i.e. $0.5\times2^{8} = 128$ or $0.5\times2^{10} = 512$).

Channel Filter Pointer This register contains an offset to a table of 128- or 512-ARINC words arranged by label or both SDI and label. Entries in this table govern sequential monitoring on a channel level for each possible receive label or both SDI and label. Entries are placed on 8-bit boundaries. even labels occupying the eight least significant bits and odd labels occupying the eight most significant bits.

Channel Monitor Initial Pointer This register contains a pointer to the base of the channel monitor.

Channel Monitor Current Pointer This register contains a pointer to the currently active channel monitor buffer which stores received data.

Channel Monitor Last Pointer This register contains a pointer to the last monitor buffer which was filled. Data is this buffer is safe to read.

Channel Monitor Buffer Counter This register contains a value that indicates the monitor swap count, the value increasing incrementally each time a channel monitor swap occurs. It can initialize to any value.

Bus Traffic Word Count This value increase incrementally for each word received and can be used as a measure for bus loading.

CHLED This value is used by the A429 firmware to indicate bus activity. For example, the value is set to 0000 if there is no receiving bus activity; the value is set to

00FF if the receiving channel is active; value FF00 indicates that the channel is receiving errors.

3.1.7.2 Receive Firmware Operation

During receive operations, the firmware polls an I/O Control/Status Register (at 0000h address) to determine if new data has been received, or if a bus error has occurred on a specific receive channel. If data has been received or an error has occurred, the firmware reads the I/O receive buffer. It processes the data, fetches a time stamp, and stores the data in the current value buffer (if initialized). The firmware then stores the data in the sequential monitors. If an error occurs when receiving the data, the data and a time stamp are stored in the sequential monitor regardless of the bits set in the filter table. The current value buffer is not updated.

3.1.8 Transmitter Set-up

Transmitter set-up is a key application of DTB using A429 standard interface libraries. Even though all calling functions exist in the '*tx_mgmt.c*' file, every parameter inside each function should be set up very precisely according to the firmware arrangement (see section 3.1.6). This section is helpful for setting up ARINC label "property file" which will be discussed in chapter 4 as well.

In order to achieve this goal, it is best to clarify the concepts by showing a simple example as the following. This example shows how to transmit two ARINC messages at two different transmitting speeds, say 100ms (10 messages per major frame) and 50ms (20 messages per major frame) in channel 5 of device 1. Figure 18 shows the relationship among the property parameters in a transmitting control block. First, the

minor frame time is defined as the Greatest Common Divider (GCD) of the different speeds in the same transmitting channel. In this example, it is equal to 50ms. Secondly, a transmit command control block is decided by the minimum speed for both messages. In this example, it is equal to 100ms. Finally, It is clear that there are 10 message 1 in the transmitting command block and each message 1 writes every other minor frame. Also there are 20 message 2 in the transmitting command block and each message 2 writes every minor frame.

There are three major functions that need to be called up, namely '*A429_create_tx_cb*', '*a429_add_mf_cmd_blk*' and '*a429_add_tx_cmd_blk*'. Those functions are defined in '*a429_incl.h*' of the libraries. The descriptions are given below, and Table 3 shows all parameters definition inside the functions.

- '*A429_create_tx_cb*' creates a control block for a specified transmit-channel. It sets the transmitter speed, minor frame count, major frame count, and the inter-word gap time.
- '*a429_add_mf_cmd_blk*' adds a minor frame to an existing chain of minor frame and transmit command blocks. It sets the minor frame time in microsecond, loop flag and command type.
- *a429_add_tx_cmd_blk*' adds a transmit command block to an existing chain of minor frame and transmit command. It sets the transmitting ARINC word count, frame start and repeat rate.

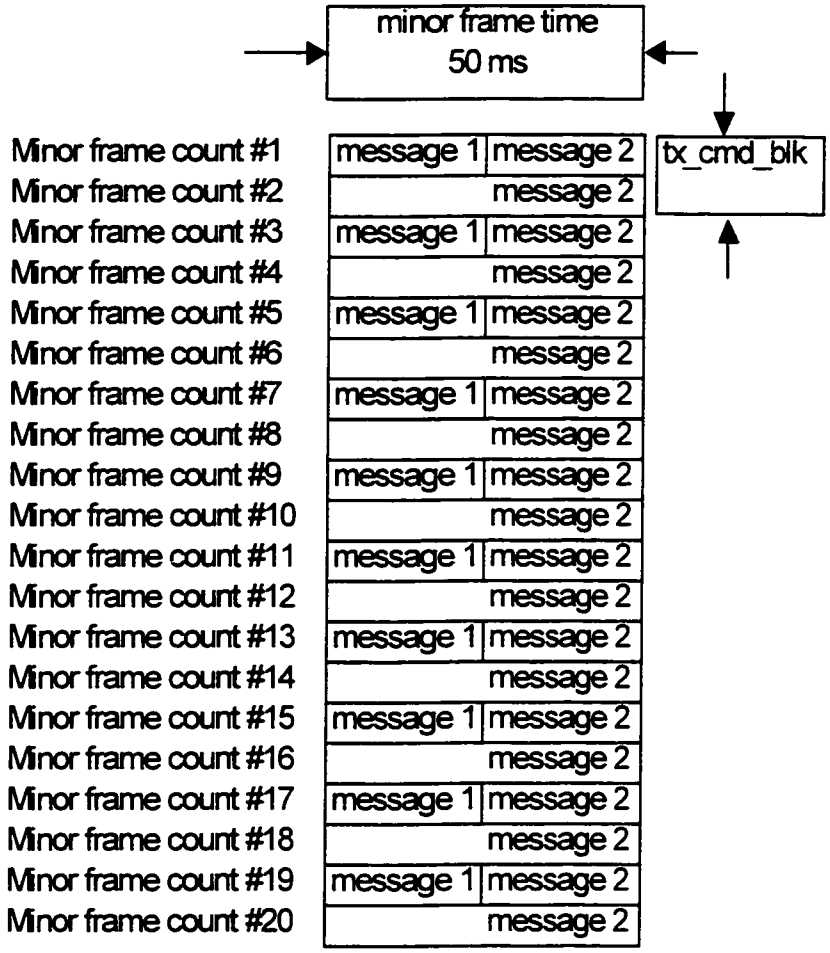


Figure 18: Parameters Relationship of a Transmitter

Table 3: Definition of Transmitter Parameters

Parameter	Description
device_number	The device on which to add the transmit command block.
Channel_number	The channel on the ARINC device.
Speed	The frequency of the transmitter. SLOW means 12.5 kHz. FAST means 100 kHz.
Minor frame time	The Greatest Common Divider (GCD) of the different speeds in the same transmitting channel. In this example, it is equal to 50ms.
minor_frame_count	The number of minor frames to execute for each major frame. Here it is 20.
Major frame time	20 * 50ms = 1 sec
major_frame_count	The number of major frames to execute before halting command block processing. For continuous operation, set to "0".
gap_time	The inter-word gap time in μs. Note that even if set to "0", the minimum inter-word gap time at which ARINC 429 words can be transmitted is four bit times.
chain_id	The number of chain to create.
loop_flag	Set it to TRUE to loop through the chain for continuous transmission.
command_type	Minor Frame Command type. It is set to match the firmware. See details in Table 1.
scheduled_time	The time in μs that must expire before the transmit command block is processed.
xmit_count	The number of data words to transmit for the associated command block.
start_frame	Which frame the transmit command block will begin executing.
repeat_rate	How often to process a transmit command block after the **start_frame** is met. It is also a relative frequency to a Major Frame.
transmit_type	A transmit type command block to match the firmware. See details in Table 1.
*pData	A pointer to the associated data buffer for the transmit command block.

Finally, the solution to the problem in this example is as follows:

Set up:

```
a429_create_tx_cb (1, 5, A429_SLOW, 20, 0, 0 )
        /* (device_number, channel_number, speed,
            minor_frame_count, major_frame_count, gap_time) */
a429_add_mf_cmd_blk (1, 1, 50000, TRUE, 0x0000)
            /* (device_number, chain_id, minor_frame_time,
                loop_flag, command_type) */
```

for message 1:

```
a429_add_tx_cmd_blk (1, 1, 0, 2, 1, 1, TRUE, 0x0001, buffer)
    /* (device_number, chain_id, scheduled_time, xmit_wordcount,
start_frame,
        repeat_rate, loop_flag, transmit_type, *pData) */
```

for message 2:

```
a429_add_tx_cmd_blk (1, 1, 0, 2, 1, 2, TRUE, 0x0001, buffer)
        /* (device_number, chain_id, scheduled_time, xmit_wordcount,
            start_frame, repeat_rate, loop_flag, transmit_type, *pData) */
```

It is noted that the "repeat_rate" is assigned different values for synchronization purposes, according to Figure 18. Since there are 20 message 2 and 10 message 1 in a same major frame, the relative frequency between message 2 and message 1 is 2. Also, to distinguish message 1 from message 2 as shown in the above shown example, two transmit command blocks have to be assigned to channel 5 of device 1. First, message 1

is configured by Transmit Command Block 1 (TCB1), then message 2 is configured by Transmit Command Block 2 (TCB2).

The DTB involves many similar function set-ups, where the transmitter set-up is one of the typical applications. It is necessary to point out that function set-ups are extremely important since even a single wrong parameter set-up could fail the whole device I/O functionality or cause run time errors.

3.1.9 ARINC Word & Conversions

It is important to note that each A429 word consists of 32 bits with the first 8 bits (label) octally encoded to represent the type of information contained within the 32 bit word. Figure 19 shows the organization of the 32 Bit ARINC 429 word, a typical 32-bit word having five parts:

- 8-bit label
- Source / Destination Identifier (SDI)
- Data area
- Sign / Status Matrix (SSM)
- Odd parity bit

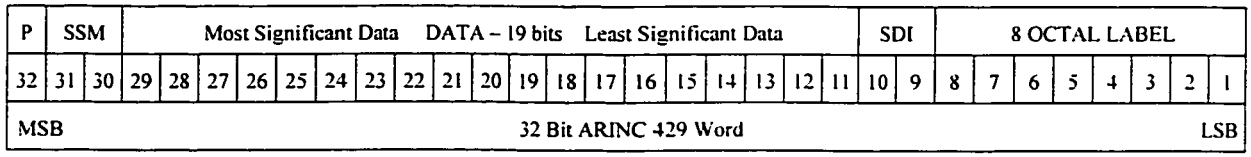


Figure 19: 32-bit ARINC Word

The least significant bit of each byte except the label is transmitted first, and the label is transmitted ahead of the data in each case. The order of the bits transmitted on the ARINC bus is as follows: 8, 7, 6, 5, 4, 3, 2, 1, 9, 10, 11, 12, 13 ... 32.

When a 32-bit ARINC word is transmitted on the bus, in the case of the label, the most significant bit is transmitted first. This reverse order is in contrast to the transmission order of the other bits in the ARINC word.

3.1.9.1 Label

It is necessary to point out that the label has a reversed bit order. The most significant bit of the octal word is located in the least significant ARINC 429 bit and is transmitted first out onto the bus. Since the LSB of the ARINC word is transmitted first, this effect causes the label to be transmitted onto the bus in reverse bit position order. For example, octal label 116 is transmitted as 72h. Details are shown in Table 4. Appendix E also shows some sample ARINC label information.

Table 4: Label Conversion

7				2				Hex.
8	4	2	1	8	4	2	1	
8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Bit**
0	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**Value**
1	2	4	1	2	4	1	2	
6			1			1		Octal

3.1.9.2 SDI

The Source / Destination Identifier (SDI) is optional and when selected, occupies bits 9 and 10 of the ARINC word. When used, the SDI is considered as adding an

extension onto the ARINC word's Label, and ARINC firmware is expected to decode the Label / SDI combination as a different label from an ARINC word with the same Label but no SDI implementation. There are two functions for SDI - to identify which source (FMU) of a multi-system installation (FMUs) is transmitting the data contained; or identify which destination (FMU) on a multi-system installation (FMUs) should distinguish the data contained within the ARINC word. For example, SDI (00) represents to FMU; SDI (01) represents to FMU1; SDI (10) represents to FMU2; SDI (11) represents to FMU3. In the CMC project, SDI for different FMU is preset by configuring certain discrete SDI port lines from the FMU connectors.

In actual use, the basic structure of the ARINC 429 word is very flexible. The only two parts of the word needing to stay intact are the label and the parity bit. For example, some data like label 076 (GNSS Altitude) is out of the range if using 19-bit representation. Hence, bit 10 of ARINC word (SDI bit) is used as extension data area for special 20-bit data presentation.

3.1.9.3 Data Area

The data element can be categorized into many application groups. In the DTB application, it involves two kinds of data, namely, binary code (BNR) data and binary coded decimal (BCD) data.

In the DTB application, all conversion between Engineering Value and standard ARINC 429 word has to be done before transmitting data from the Simulation PC to the Interface PC, and vise versa. The principle of the conversion can be explained through simple examples, and which applies to the coding as well.

BNR Data Encoding

BNR or "binary" encoding is a very common ARINC data format. This type of encoding simply stores the data as a binary number, much in the same format that is used on virtually every modern-day computer. Table 5 shows the general BNR format. Bit 29 is the sign bit and bit 28 is the most significant bit of the data field.

Example: Converting the FLSIM output values to 32-bit ARINC word

Label = 116

Format = BNR

Cross Track Distance = 73.2 NM

Maximum Value = 128 NM

Minimum Value = -128 NM

N bits = 15

LSB = 14

Solution:

Step 1: Calculate the Scaled Value with following formula.

$$\frac{X-\min}{\max-\min}\times 2^{n}$$

Where X is the sending engineering value, n is number of binary digits available for the representation. Hence,

$$\text{Scaled Value} = \frac{73.2-(-128)}{128-(-128)}\times 2^{15} = 25753 \qquad \text{(fraction ignored)}$$

Step 2: Convert the Scaled Value into binary format.

$(25753)_d = (1100100100111001)_2$

Step 3: Shift left (LSB-1) bits so that the DATA starts the Least Significant Bit (LSB) and ends at the Most Significant Bit (MSB).

Step 4: Add 8-bit label.

Table 6 illustrates the conversion procedure.

BCD Data Encoding

BCD, or binary-coded decimal, is also a common data format in ARINC 429 and many other engineering applications. In this format, four bits are allocated to each decimal digit. A generalized BCD message is shown in Table 7. Its data fields contain up to five sub-fields. The most significant sub-field contains only three bits. so that its maximum decimal value can be 7. If the maximum decimal value is greater than 7. bits 29 through 27 are padded with zeros and the second sub-field becomes the most significant. Again, Bit 29 is a sign bit.

Example: Converting the FLSIM output values to 32-bit ARINC word

Label = 001

Format = BCD

Distance to way point = 1234.5 NM

Maximum Value = 3999.9 NM

Minimum Value = 0.0 NM

N bits = 18

LSB = 11

Solution:

The procedure is almost the same as that of the BNR format except for the method used to calculate the Scaled Value. Table 8 illustrates the conversion procedure.

Here, Scaled Value $= 5\times16^{0}+4\times16^{1}+3\times16^{2}+2\times16^{3}+1\times16^{4}=74565$

It is noted that each ARINC label has its own detailed word description. In order to distinguish the maximum value and minimum value with decimal point, the detailed bit arrangement is shown on Appendix F.

Table 5: Generalized BNR Word Format

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
P	SSM		Data																			SDI		Label							

Table 6: Converting an Engineering Value to ARINC Word (BNR format)

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	1	Scaled Value = 25753
0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Shift left 13 bits
																								0	1	1	1	0	0	1	0	Add label
0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	1	1	1	0	0	1	0	Final Value = C932072 HEX

Table 7: Generalized BCD Word Format

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
P	SSM		CHAR 1			CHAR 2				CHAR 3				CHAR 4				CHAR 5				SDI		Label							

Table 8: Converting an Engineering Value to ARINC Word (BCD format)

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1	Scaled Value = 74565 Dec
0	0	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	Shift left 10 bits
																								1	0	0	0	0	0	0	0	Add label
0	0	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	Final Value = 48D1480 Hex

3.1.9.4 Sign/Status Matrix

The Sign/Status Matrix (SSM) field is used to report hardware equipment condition or the signs. For BNR data, the Sign function can be described by bit 29, "0" will stand for Plus / North / East / Right / To / Above while a "1" will indicate Minus / South / West / Left / From / Below. Bits 30 and 31 are typically assigned to the SSM for the status function only, "11" is used to indicate the Failure Warning, "01" is encoded for No Computed Data, "10" is set to Functional Test Indicator while "00" represents Normal Operation. For BCD data, The Sign function can be achieved by using bits 30 and 31. A "00" will stand for Plus / North / East / Right / To / Above while a "11" is indicating indicate Minus / South / West / Left / From / Below. The SSM bits can also be used to report the status function, "01" indicates No Computed Data while "10" indicates the Functional Test Indicator.

3.1.9.5 Parity Field

It should be noted that ARINC has a specific odd parity convention rule: if the number of 1's in the final ARINC word (without counting the parity bit 32) is even, the parity bit has to be assigned 1, then the final count would be odd.

3.1.10 ARINC I/O Driver Derived

DTB uses Integrated Avionics Programming Library (sbs_ver6.10) and runs Firmware '*F030H.DAT*'. One ARINC429-PC16 card has been installed in the Interface PC and it is the main media to communicate with the FMU's. The driver for each ARINC bus (transmit or receive) was set up with a Microsoft NT 4.0 (workstation) operating system and Microsoft Visual C++ (6.0) compiler.

3.1.10.1 Installing the ARINC429 IO Card

The base I/O address was set to 390h as shown in Figure 20:

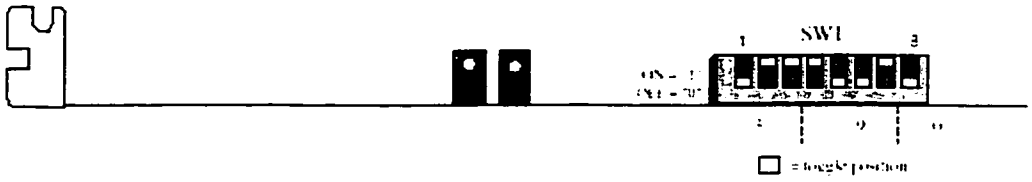


Figure 20: Base I/O Address 390h, Top View of PC16 as installed [12]

3.1.10.2 Installing the software

The support software from SBS Technologies Inc. (SBS) includes the Integrated Avionics Library (sbs_ver6.10) and Firmware (F030H.DAT). The library contains the C library source files, sample applications and Unit Test executable; The firmware contains the file that must be downloaded to the A429 PC16 card upon initialization [12].

3.1.10.3 Installing the SBS Driver Library

3.1.10.3.1 'sbs_dev.cfg' and 'sbsa429.ini'

Two library files have been modified in order to start the devices. One is '*sbs_dev.cfg*'; the second that is essential is '*sbsa429.ini*'. The details are presented on Appendix A and Appendix B respectively.

3.1.10.3.2 Required files of the device driver

Step 1: In order to achieve transmitting and receiving functions, a list of source files and header files have to be added to the application project [13] as Table 9 and Table 10 show respectively.

Table 9: Required .cpp Files

File	Description
ll_winnt.cpp	Low-level operating-system-specific file
crt_io.cpp	Display processing
dev_mgmt.cpp	High-level common device management
429_dm.cpp	Device management
429_mm.cpp	Monitor management
429_rcm.cpp	Receive management
429_txm.cpp	Transmit management

Table 10: Required .h Files

File	Description
low_lvl.h	Additional low-level files for NT system
crt_io.h	Customized video commands which are based upon the type compiler or video terminal being used
dev_mgmt.h	Declares device exceptions, constants, global variables, and function prototypes for common device management functions
sbs_sys.h	Defines the operating system type
429_def.h	Definitions
429_incl.h	Include files
429_prot.h	Function prototypes
429_type.h	Type declarations
429_dd.h	A429 NT kernel driver include files
dev_cfg.h	Device information to be used in lieu of sbs_dev.cfg
HardwareDefs.h	SBS Avionics Board Include File used by software intended for WinNT operating systems

Step 2: In order to initialize devices and start I/O, the following functions must be called in the order shown on Table 11.

Table 11: Required Functions for Initializing

Function	Description
sbs_parse_config_file( )	Parses the information in *sbs_dev.cfg* file in order to initialize the fields of the **SBS_DEVICE_RECORD** for each device.
sbs_init_device( )	Initializes the SBS device for access by the application program.
a429_create_global_sm_buffers( )	Creates global sequential monitor buffers.
a429_create_channel_sm_buffers( )	Creates the specified number of channel sequential monitor buffers in upper memory for each of the receive channels configured on the card.
sbs_start_io( )	Enables the device to perform avionics bus operations.
a429_create_tx_cb( )	Creates a control block for a specified transmit-channel.
a429-add_tx_cmd_blk( )	Adds a transmit-command block to an existing chain of minor frame and transmit command blocks.
a429-add_mf_cmd_blk( )	Adds a minor frame to an existing chain of minor frame and transmit command blocks.
a429_load_chain( )	Loads a chain for execution.
a429_write_type_word( )	Writes the minor frame/transmit type word of a command block.
a429_read_type_word( )	Reads the minor frame/transmit type word of a

	command block.
a429_create_rc_cb( )	Creates a control block for a specified receive-channel. It sets the channel speed and sort type. The sort type can be SDI and label or label only. It is set by label in our application program.
a429_set_rc_control( )	Sets the receive-control word.
a429_set_filter_table( )	Writes a value to a global (0) or channel (1) filter table for a single specified label.
a429_start_rc( )	Enables the receiver operations for a specified receive-channel.
a429_write_tx_cb_data( )	Gets the address of the data buffer from the transmit-control block and then write the data to a chain link transmit data buffer.
a429_convert_label( )	According to the ARINC specifications, the label has a reversed bit order. Labels are bit-swapped prior to being stored in the current value buffer. This function converts a label to or from a bit-swapped value and returns the converted value of the label.
a429_halt_cb( )	Halts the control block of an ARINC transmit/receive channel.
a429_stop_rc( )	Halts the receiver operations for a specified receive-channel.
sbs_stop_io( )	Stops input and output operations.
sbs_close_device( )	Closes access to the device. Since only one logical connection to the device can be active at a time within the application, we must close the device before reopening it.

3.1.10.4 Configuration Files

Two text files called '*A429Config1.prop*' and '*A429Config2.prop*' have been created in order to set up all the ARINC words properties such as frame time, scheduled time and repeat rate into the file without bothering other classes. The details can be found in Appendix C. The reason why it is set up in this way is for our design to match the operating firmware (Section 3.1.8 has more explanation).

3.2 PCI-6025E Interface

The National Instrument PCI-6025E card has 16 channels (eight differentials) of analog input, two channels of analog output, 32 lines of digital I/O and a 100-pin connector [5]. In the DTB application, analog I/O has been used for testing purpose only. Since at the beginning of the project, there were no FMS available in the CIC lab, in order to check if the communication between two PCs was working, an analog input from DC voltage generator had been added to the DTB. While tuning the voltages range, the reaction from the flight simulator could be seen, and vise versa. For the CMC requirement, digital I/O plays the software-programmable role in enabling or disabling discrete signals from certain FMU ports. Once the National Instrument driver library (NI-DAQ 6.5.1) has been installed properly, the PCI-6025E is ready for insertion into the 5V PCI slot of the Interface PC. The overview of the pin assignment is shown in Figure 21. The block diagram of PCI-6025E is shown in Appendix I.

3.2.1 Analog Input

3.2.1.1 Overview

The board has three different input modes: non-referenced single-ended (NRSE) input, referenced single-ended (RSE) input, and differential (DIFF) input. The single-ended input configurations provide up to 16 channels, whereas that of the DIFF provides eight. These three modes are described in Table 12. DTB uses the RSE mode.

Table 12: PCI-6025E Analog Input Modes

Configuration	Description
DIFF	A channel configured in DIFF mode uses two analog input lines. One line connects to the positive input of the board's Programmable Gain Instrumentation Amplifier (PGIA), and the other connects to the negative input ground (AIGND).
RSE	A channel configured in RSE mode uses one analog input line, which connects to the positive input of the PGIA. The negative input of the PGIA is internally tied to analog input ground (AIGND).
NRSE	A channel configured in NRSE mode uses one analog input line, which connects to the positive input of the PGIA. The negative input of the PGIA connects to analog input sense (AISENSE).

AIGND	1	51	PC7
AIGND	2	52	GND
ACH0	3	53	PC6
ACH8	4	54	GND
ACH1	5	55	PC5
ACH9	6	56	GND
ACH2	7	57	PC4
ACH10	8	58	GND
ACH3	9	59	PC3
ACH11	10	60	GND
ACH4	11	61	PC2
ACH12	12	62	GND
ACH5	13	63	PC1
ACH13	14	64	GND
ACH6	15	65	PC0
ACH14	16	66	GND
ACH7	17	67	PB7
ACH15	18	68	GND
AISENSE	19	69	PB6
DAC0OUT	20	70	GND
DAC1OUT	21	71	PB5
RESERVED	22	72	GND
AOGND	23	73	PB4
DGND	24	74	GND
D1O0	25	75	PB3
DIO4	26	76	GND
DIO1	27	77	PB2
DIO5	28	78	GND
DIO2	29	79	PB1
DIO6	30	80	GND
DIO3	31	81	PB0
DIO7	32	82	GND
DGND	33	83	PA7
POSITIVE 5V	34	84	GND
POSITIVE 5V	35	85	PA6
SCANCLK	36	86	GND
EXTSTROBE	37	87	PA5
PF10/TRIG1	38	88	GND
PFI1/TRIG2	39	89	PA4
PF12/CONVERT*	40	90	GND
PF13/GPCTR1-SOURCE	41	91	PA3
PF14/GPCTR1-GATE	42	92	GND
GPCTR1-OUT	43	93	PA2
PF15/UPDATE*	44	94	GND
PF16/WFTRIG	45	95	PA1
PF17/STARTSCAN	46	96	GND
PF18/GPCTR0_SOURCE	47	97	PA0
PF19/GPCTR0-GATE	48	98	GND
GPCTR0-OUT	49	99	POSITIVE 5V
FREQ-OUT	50	100	GND

Figure 21: I/O Connector Pin Assignment for the PCI-6025E

The board has a bipolar input range that changes with the programmed gain. Each channel may be programmed with a unique gain of 0.5, 1.0, 10.0, or 100.0 to maximize the 12 bit analog-to-digital converter (ADC) resolution. With proper gain setting, the full resolution of the ADC can be used to measure the input signal. Table 13 shows the input range and precision with respect to the gain used. DTB uses gain of 1.0.

Table 13: PCI-6025E Measurement Precision

Gain	Input Range	Precision
0.5	-10 to +10 V	4.88 mV
1.0	-5 to +5 V	2.44 mV
10.0	-500 to +500 mV	244.14 μV
100.0	-50 to +50 mV	24.41 μV

3.2.1.2 RSE Configuration

A single-ended connection is one in which the board analog input signal is referenced to a ground that can be shared with other input signals. The input signal is tied to the positive input of the PGIA, and the ground is tied to the negative input of the PGIA. Figure 22 shows a diagram of PGIA with RSE configuration for a floating signal source. A floating signal source is not connected in any way to the building ground system but, rather, has an isolated ground-reference point. In the DTB, the ground reference of a floating signal from the negative end of DC power supply output must be tied to the board's analog input ground (AIGND) in order to establish a local reference

for the signal. Otherwise, the measured input signal varies as the source floats out of the common-mode input range.

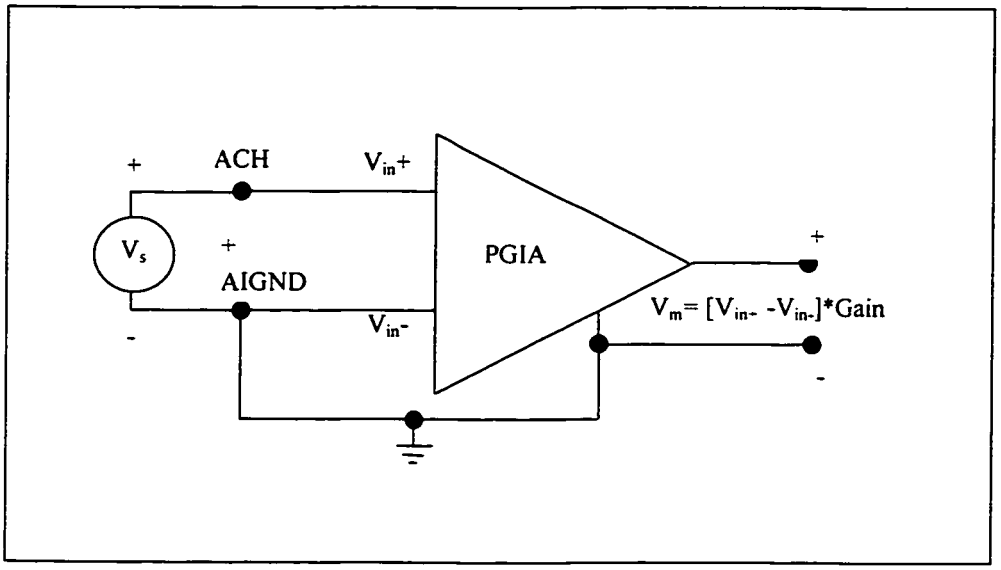


Figure 22: PGIA with RSE Configuration for Floating Signal Source

When every channel is configured for single-ended input, up to 16 analog input channels are available. In NRSE mode, signals connected to ACH<0..15> are routed to the positive input of the PGIA. The AIGND signal is connected internally to the negative input of the PGIA when their corresponding channels are selected.

In single-ended configurations, more electrostatic and magnetic noise couples into the signal connections than in differential configurations. The coupling is the result of differences in the signal path. Magnetic coupling is proportional to the area between the two signal conductors. Electrical coupling is a function of how much the electric field differs between the two conductors.

3.2.2 Analog Output

The board provides two channels of analog output voltage at the I/O connector. The bipolar range is fixed at +/-10 V. Figure 23 shows how to make analog output connections to the board.

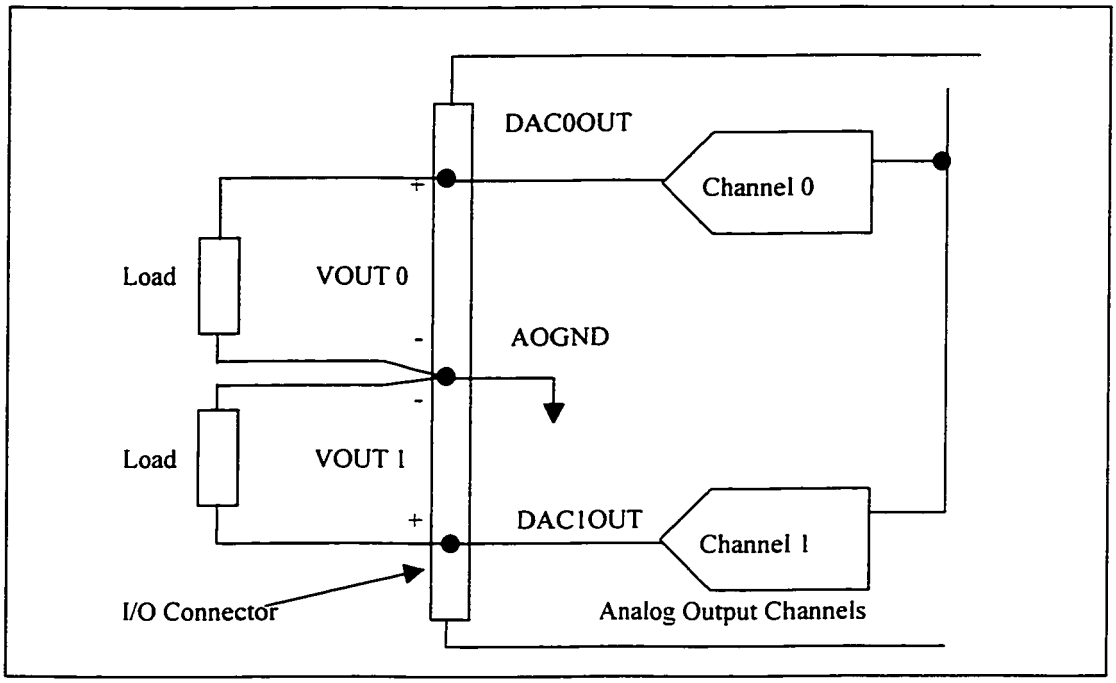


Figure 23: Analog Output Connectors

3.2.3 Discrete I/O

The port 0 of PCI-6025E board has 8 lines of discrete I/O (DIO0~DIO7) for general-purpose use. In this port, all eight lines can be individually software-configured as either input or output. At system start up and reset, the discrete I/O ports are all at high impedance.

In addition, the PCI-6025E card uses an 82C55A Programmable Peripheral Interface (PPI) to provide another 24 lines of discrete I/O, which represent three 8-bit ports: PA, PB, and PC. The 82C55A has 3 modes of operation: simple I/O (mode 0), strobed I/O (mode 1), and bi-directional I/O (mode 2). In modes 1 and 2, the three ports are divided into two groups: group A and group B. Each group has eight data bits, plus control and status bits from Port C. Modes 1 and 2 use handshaking signals from the computer to synchronize data transfers. The DTB just uses mode 0. Under this selection, it should be noted that, among the four ports, only port 0 can be configured with mixed lines, i.e., some lines can be configured as inputs whereas others as outputs in the same port. Ports PA, PB, and PC are different. They can be configured as input or output, but only as a port, which means all the lines in each port can be either inputs or outputs at the same time. Figure 24 depicts signal connections for three typical discrete I/O applications.

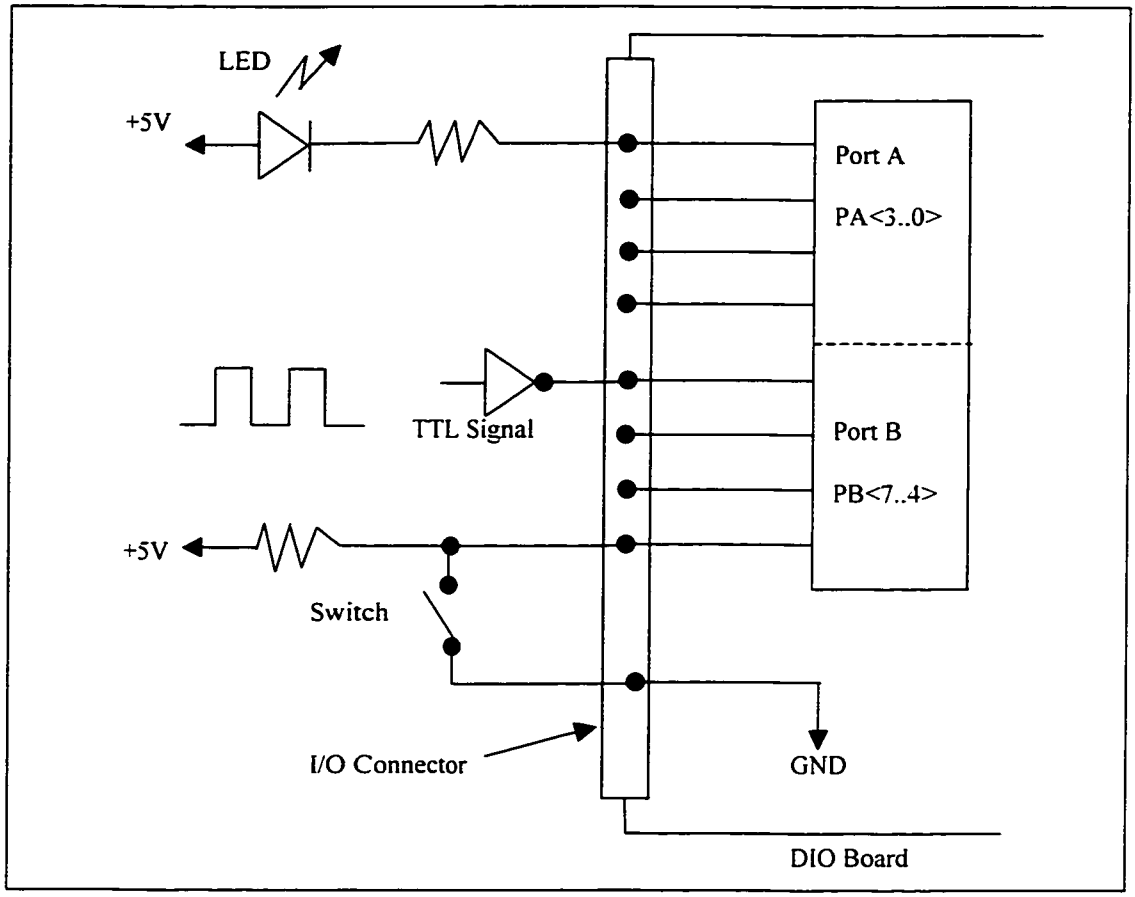


Figure 24: Discrete I/O Connections Block Diagram

In Figure 24, port A of the PPI is configured for discrete output, and port B is configured for discrete input. Discrete input applications include receiving TTL signals and sensing external device states such as the state of the switch. Discrete output applications include sending TTL signals and driving external devices such as the LED.

3.2.4 Discrete I/O Power-up State

The PCI-6025E contains bias resistors that control the state of the discrete I/O lines PA<0..7>, PB<0..7>, PC<0..7> at power-up. Each DIO line is initially pulled to Vcc (approximately +5 VDC) with a 100kΩ resistor. In the case of using low logic as initial discrete I/O power-up state, a pull-down resistor (R_L) has to be connected between that line and ground. Figure 25 shows the DIO configuration to pull down the DIO power-up state from high to low.

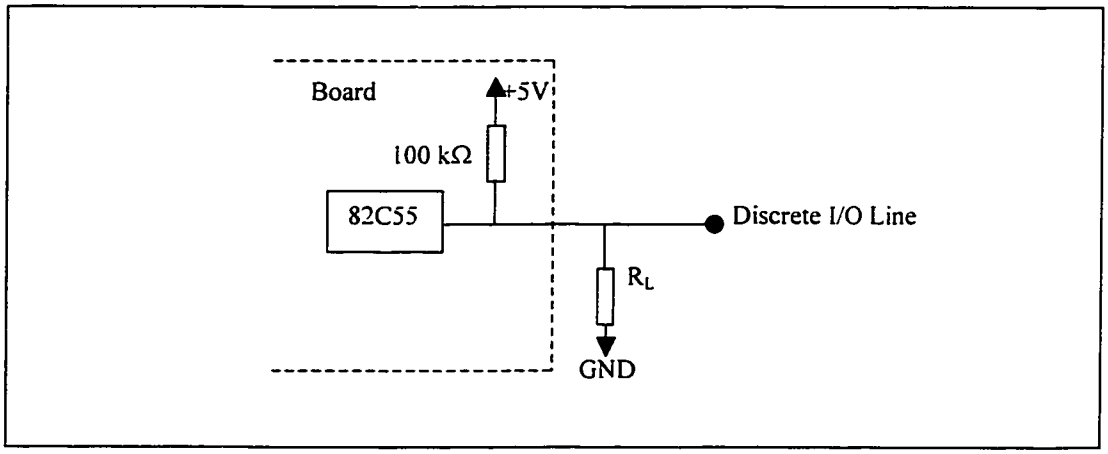


Figure 25: DIO Line Configured for High DIO Power-up State

However, make sure that the resistor's value is not so large that leakage current from the DIO line along with the current from the 100 kΩ pull-up resistor drives the voltage at the resistor above a TTL-low level of 0.4 VDC.

To pull it low on power-up with an external resistor, follow these steps:

Step 1. Install a load (R_L). Remember that the smaller the resistance, the greater the current consumption and the lower the voltage.

Step 2. Using the following formula, calculate the largest possible load to maintain a logic low level of 0.4 V and supply the maximum driving current:

$V = I^* R_L \Rightarrow R_L = V/I$, where:

$V = 0.4$ V ; Voltage across R_L

$I = 46\ \mu A + 10\ \mu A$; 4.6 V across the 100 kΩ pull-up resistor plus 10 μA maximum leakage current

Therefore:

$R_L = 7.1$ kΩ ; 0.4 V/56 μA

This resistor value, 7.1 kΩ provides a maximum of 0.4 V on the DIO line at power-up and achieves the low power-up state. Theoretically, the R_L can be replaced with a small value in order to lower the voltage or to provide a margin for VCC variations and other factors. However, smaller values will draw more current, leaving less drive current for other circuitry connected to this line.

3.3 CMA-900

The CMA-900 is the world's first GPS/FMS certified for GPS primary-means navigation in oceanic/remote areas. The system incorporates differential GPS navigation and wide and local area precision approaches. A full complement of analog and digital

interfaces is provided for other aircraft subsystems and displays, as well as company route and worldwide navigation databases [6].

The FMS accepts data from external navigation sensors, in order of priority: GPS, DME/DME, VOR/DME, INS/IRS. Information from these sensors is combined with heading and reference air data inputs of true air speed and altitude to determine aircraft position. This position is then used for navigating along a programmed flight plan created by selecting waypoints along a desired route from the navigation data base (flight planning). Comparison will also be made with a previously entered flight plan to indicate any course corrections necessary, or alternatively the FMS will be interfaced with the aircraft control systems to achieve this automatically. A comparison of positional information provided by the different sensors is also available to the pilot.

Distance to the next waypoint of the journey, together with the estimated time of arrival, will be displayed and, should it be desired to arrive by a given time, the FMS will indicate the necessary Indicated Air Speed (IAS) or MACH to achieve this. If a hold is required at the waypoint, the automatic guidance will ensure that is performed in accordance with published procedures. Typically up to 5.000 waypoints and 300 discrete routes have been stored within the navigation data bank of the system.

When operating in conjunction with ground based aids such as DME/DME, the appropriate frequencies will be automatically selected and tuned. Should a required frequency be unavailable for any reason, the equipment will indicate this fact and make an alternative selection. In the event of a temporary total loss of data from the ground-based aids, the FMS will maintain a Dead Reckoning (D/R) plot until data is again available.

In addition to navigation functions, the FMS also provide assistance in fuel management by computing the present consumption and estimating fuel remaining at destination, maintain an engineering log by recording the nature and time of events, and assist in similar housekeeping duties.

3.3.1 Features

CMA-900 features include:

- Multi-sensor navigation modes with available sensors
- Standard Instrument Departure (SID)
- Standard Terminal Arrival Routes (STAR)
- GPS instrument approaches
- Direct-to/intercept navigation, holding patterns, procedure turns, arcs, offset tracks and search patterns
- Automatic waypoint sequencing, with and without turn anticipation
- Required and actual navigation performance (RNP/ANP)
- Required time of arrival (RTA)
- Fuel management and navigation tuning functions

3.3.2 Waypoint Navigation

The FMS navigates from waypoint to waypoint sequentially, automatically changes the legs, and displays all required navigation parameters computed according to the relationships and direction sense illustrated in Figure 26. Those terminology definitions are described in Appendix H.

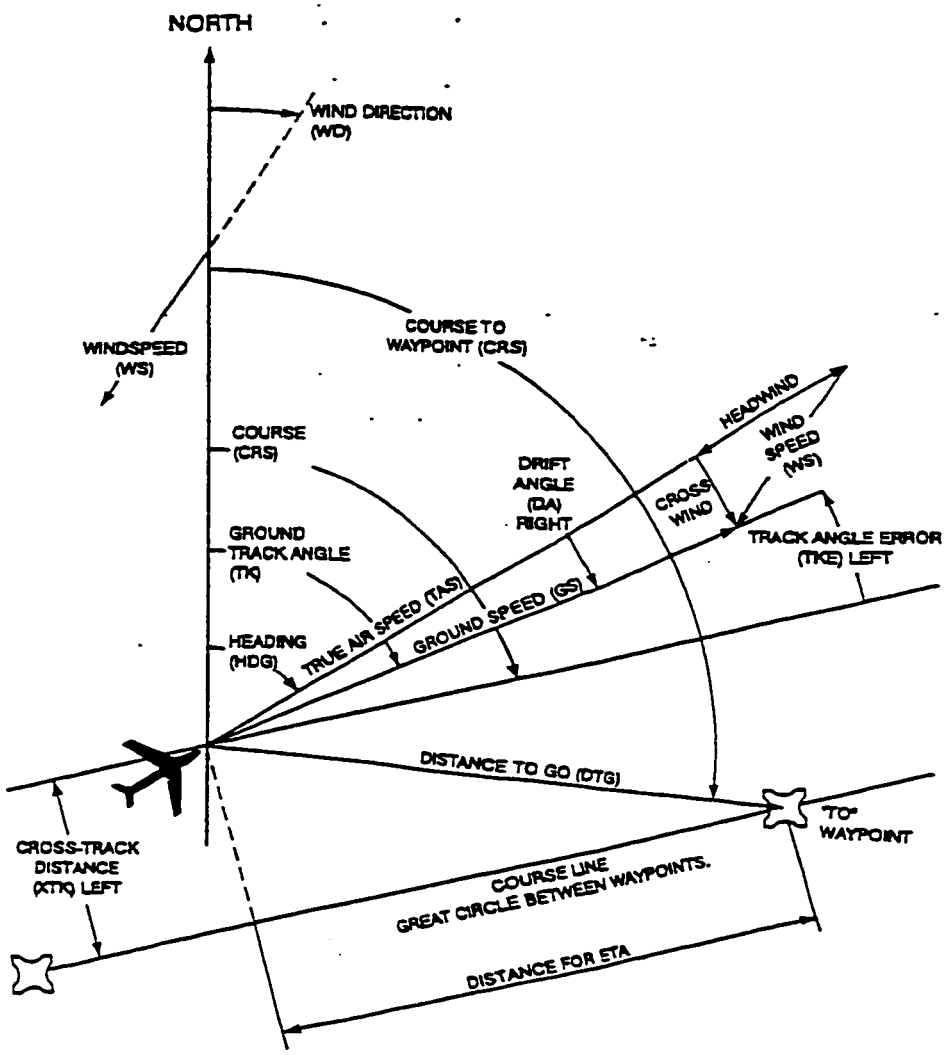


Figure 26: Navigation Relationships [7]

The FMS switches leg prior to reaching the active (TO) waypoint so that the aircraft turns are smooth without any overshoot. Waypoints may be defined as either fly-by (with turn anticipation) or fly-over (no turn anticipation). The aircraft is considered to have passed a given point when it has crossed the bisector of the course change angle. Thus, for leg sequencing to occur, the aircraft needs to pass the bisector of the course

change angle or the way-line of the "TO" waypoint whichever occurs first, depending on the flight plan geometry of whether the airplane is on/off track [7].

Where automatic leg sequencing has been inhibited, such as at the missed approach waypoint, the FMS will prevent the leg change from occurring, but will continue to provide guidance along the extension of the last leg until further operator action is taken. After the last defined waypoint is overflown, or when a route discontinuity is active, the FMS will provide guidance along the extension of the last leg, but will prevent autopilot-coupled navigation by disengaging the LNAV mode of the autopilot/flight director system.

3.3.3 Departures

The complete departure procedure, including Standard Instrument Departure (SID) and SID transition, can be loaded into the route at the same time or in segments, depending on the Air Traffic Control (ATC) clearance received. The segments are selected form lists of named procedures extracted from the navigation database for the specified departure airport. For example, in Montreal, the SIDs could be either Dorval or Mirabel airport.

When a SID is selected, the waypoints and procedural legs are extracted from the navigation data base, procedural leg types are decoded, and all resulting waypoints are inserted into the route in the correct order.

The SID transition is linked to the appropriate waypoint of the en-route portion. If there were no SID transition, the SID would be separated from the en-route portion by

a route discontinuity. For certain types of SID and transition, the waypoints may not be loaded into the route until the runway is selected.

SID transitions are appended to the route after the SID and are usually separated from it by a route discontinuity, unless the last waypoint of the SID and the first waypoint of the en-route portion of the route are identical.

ATC clearances that modify the selected SID procedures can be incorporated by selection of a new procedure, when the aircraft is not in flight. This results in the automatic deletion of the waypoints associated with the cancelled procedure.

3.3.4 GPS Navigation

The GPS sensor uses 12 independent channels and can track any combination of GPS satellites. The GPS sensor computes and outputs three-dimensional position and velocity components, time, groundspeed, and track. Both code and carrier phase tracking are used. Carrier phase tracking greatly reduces position and velocity errors under highly dynamic aircraft manoeuvres.

The GPS is selected and will remain selected as the main sensor for navigation when:

- GPS has Receiver Autonomous Integrity Monitoring (RAIM) with a Horizontal Integrity Limit (HIL) less than or equal to the GPS Integrity Alarm Limit (GIAL) defined for the current phase of flight;
- GPS has RAIM with a HIL greater than the GIAL, or has lost its RAIM function but the aircraft is in the approach phase and the Horizontal Dilution of Precision (HDOP) is less than 4.0, then GPS shall remain selected for 5 additional minutes;

- GPS has RAIM with a HIL greater than the GIAL, or GPS has lost its RAIM function and no other approved sensor (i.e., INS, DME, VOR) available.

Before a pilot gets airborne, he / she will need to obtain an approach RAIM prediction for the destination aerodrome. RAIM tells the pilot whether there are enough satellites in view to establish the accuracy of navigation information provided by GPS. If RAIM is unavailable, a GPS approach can not be flown. If RAIM is lost, a missed approach must be executed.

For each specific phase of flight – enroute, terminal and approach – there is an additional position tolerance called Horizontal Integrity Limit (HIL). HIL is the radius of a circle in the horizontal plane, with its center being at the indicated position, which describes the region which is assured to contain the true position. It is the horizontal region for which the missed alert and false alert requirements can be met. It is only a function of the satellite and user geometry and the expected error characteristics: it is not affected by actual measurements. Therefore, this value is predictable [34].

3.3.5 GPS Instrument Approaches

The implementation of the GPS instrument approach procedures has been based on the evolving Required Navigation Performance (RNP) airspace concept. The transition from en-route through terminal to non-precision approach is in effect a seamless series of waypoints/legs with progressive increases in Horizontal Situation Indicator (HIS) lateral deviation display sensitivity (to reduce flight technical error), reductions in RNP value, and appropriately-timed alert or advisory messages.

Approach transitions are appended to the route after the STAR and are usually separated from it by a route discontinuity, unless the last waypoint of the STAR and the first waypoint of the approach transition are identical. Approach transitions may include procedure turns. Missed approach procedures are loaded as part of the approach. Arrival waypoint may include speed and altitude constraint advisories.

Baro-corrected altitude is required to enable automatic sequencing of arrival or missed approach procedure legs with altitude terminations. If the aircraft installation provides only a pressure altitude input to the CMA-900, manual entry of the altimeter correction is necessary to convert this input to baro-corrected altitude. When required, entry of the altimeter correction for the arrival airport should be performed as soon as received, and prior to reaching a radial distance of 30 nm from, and an altitude below 15,000 feet above, the arrival airport.

When the approach transition includes a procedure turn, the CMA-900 creates two outbound legs based on the procedure turn reference fix, followed by a turn in the correct direction to intercept the inbound course to the final approach fix. The still-air lengths of the procedure turn outbound legs are computed based on a time of 1 minute and 45 seconds respectively at a speed of 180 knots.

The CMA-900 provides navigation and guidance in the normal fashion for the two outbound legs of the procedure turn, followed by a turn to intercept the final approach course inbound to the final approach fix. When the maximum allowable procedure turn distance is less than 10 nm, the length of the first outbound leg may be reduced. As a result, navigation and guidance may be sequenced to the second outbound leg almost immediately after crossing the procedure turn reference fix.

On entering the terminal area, the GPS instrument approach phase of flight will be armed automatically 2 nm prior to the Final Approach Fix (FAF) as shown in Figure 27.

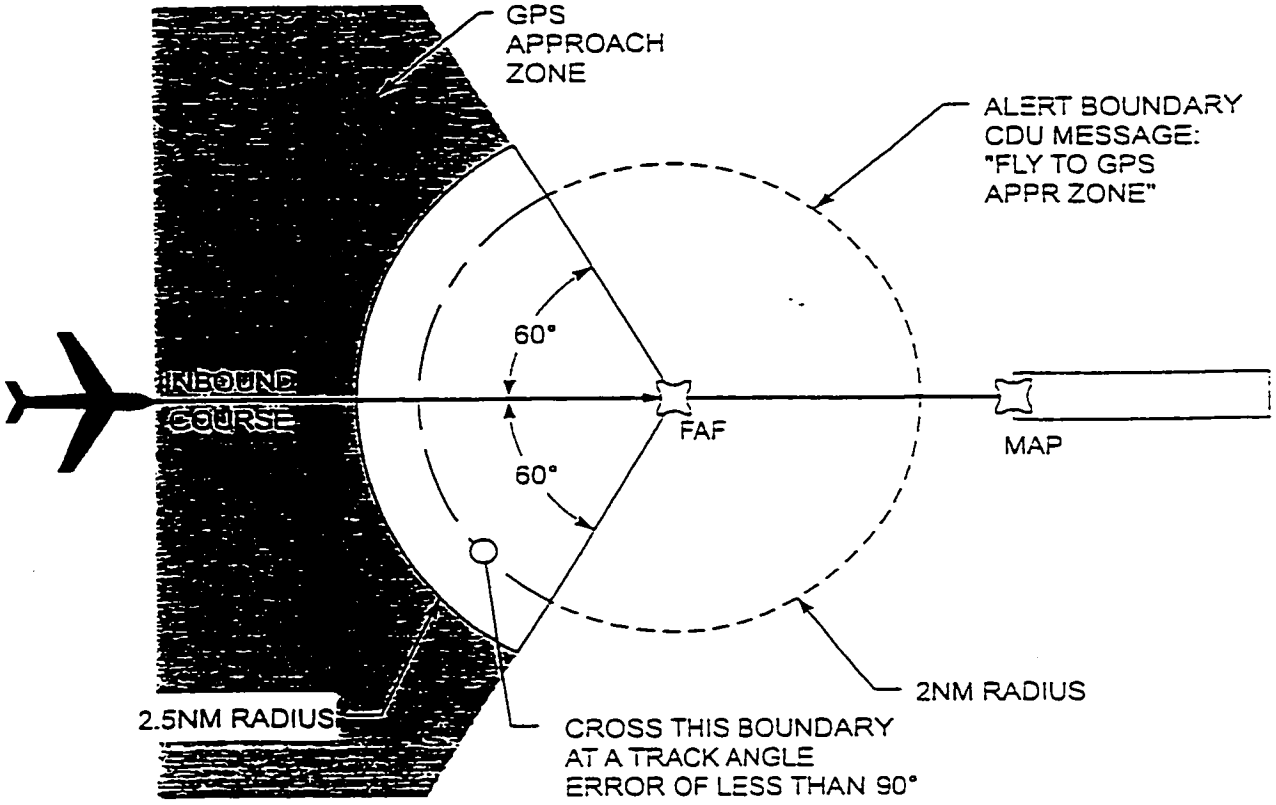


Figure 27: GPS Approach [7]

3.3.6 Required Navigation Performance

To optimize the flight performance such as fuel economics, landing safety etc., Required Navigation Performance (RNP) values are given on different flight phases. Those values are computed by aerodynamics, engine model etc. The RNP value for each leg is associated with the current phase of flight (refer to Table 14), or can be manually assigned by the operator. When the RNP value is included in the navigation database, it will be automatically extracted. The current estimate of position uncertainty (EPU), also known as actual navigation performance (ANP), is computed based on the GPS horizontal integrity limit (HIL) of figure of merit (FOM), or the quality factor of other navigation modes, and an alert message is generated when the ANP exceeds the RNP. In case of an alert generated, for example in the approach phase, the pilot must not land the plane using the malfunction FMS auto-landing navigation. Instead, the pilot should climb up the plane and use other supplemental navigation for landing.

Table 14: Required Navigation Performance

Phase of flight	Condition	Manual RNP entry	Position check limit	GPS integrity alert limit	HSI scale
EN-ROUTE	When not in terminal or approach phase of flight	None (2.0 nm default)	2.0 nm	2.0 nm	± 5.0 nm
		1.01 to 20 nm	2.0 nm	2.0 nm	± 5.0 nm
		0.31 to 1.0 nm	1.0 nm	1.0 nm	± 1.0 nm
		0.1 to 0.3 nm	0.3 nm	0.3 nm	± 0.3 nm
TERMINAL	<15,000 ft AGL and <30 nm from arrival airport but not in approach, or <16,000 ft AGL and <33 nm from departure airport	None (1.0 nm default)	1.0 nm	1.0 nm	± 1.0 nm
		0.31 to 20 nm	1.0 nm	1.0 nm	± 1.0 nm
		0.1 to 0.3 nm	0.3 nm	0.3 nm	± 0.3 nm
APPROACH	<15,000 ft AGL and within 2 nm of FAF with approach conditions satisfied	None (0.3 nm default)	0.3 nm		
		0.1 to 20 nm			

3.4 Control display Unit (CDU)

The CDU is the primary control and display operator interface of the CMA-900. The CDU keyboard, page layout and operating procedures have been designed to emulate the "full-size" FMSs widely used in "glass-cockpit" aircraft. This design approach offers operational commonality across mixed aircraft and equipment types. The simple, intuitive, scratchpad/line-select operating procedures and the flexible architecture eliminate the need for keyboard redesign to accommodate new functions. The CDU front panel is shown in Figure 28.

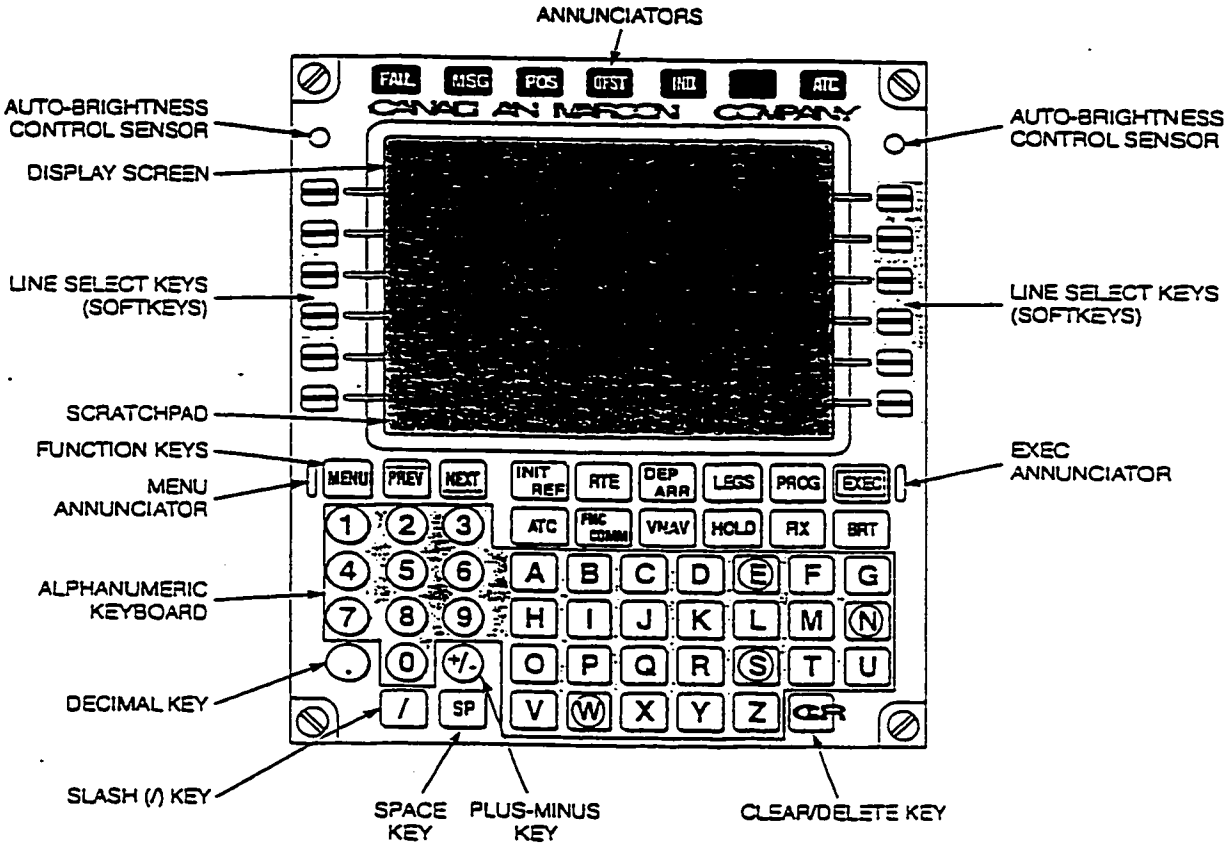


Figure 28: CDU Front Panel [2]

The unit has been designed for ease of use, with no multiple-function keys, immediate function-key access to its principal functions, and display of important information in well-formatted pages. System level functions are readily accessible through the MENU key. The pilot enters and views information on several different

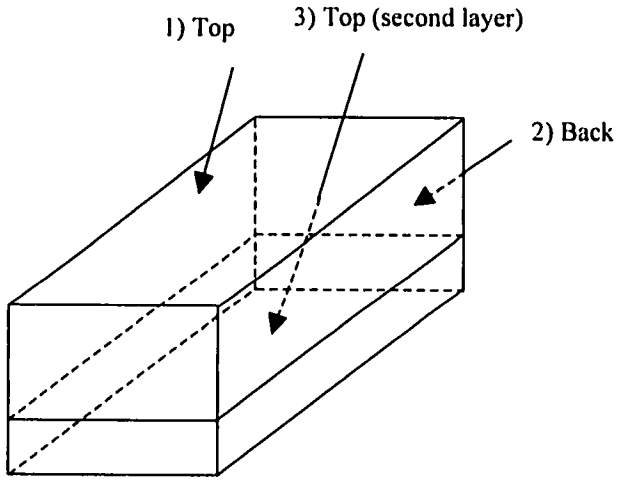
"PAGES". For example, on the Departure and Arrival page, the pilot can view all the names of available Standard Instrument Departures (SIDS), Standard Arrivals (STARS), departures and arrival runways.

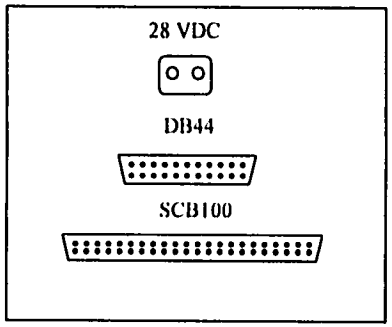
3.5 Breakout Box Design

Breakout box is a part of hardware design to integrate FMS into DTB system. It is used to interface two donated FMS CMA-900-202 or one new FMS CMA-900-402 with a CDU to the DTB. The breakout box contains all the interconnections between interfacing PC and avionics. All wiring connection can be found in Appendix D.

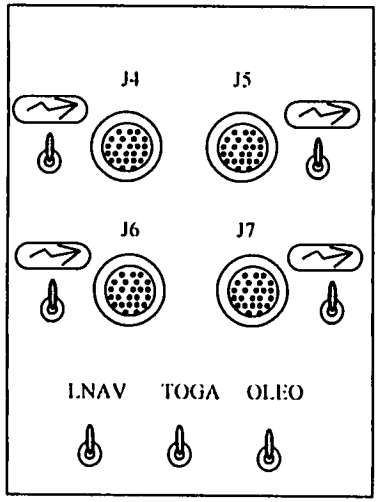
3.5.1 Dimension Design

One suitable box has been chosen as the CIC lab environment, and all the dimension design is done with an eye for convenient use and neat wiring arrangement. The dimension design includes all layouts for the box as shown on Figure 29 in details.

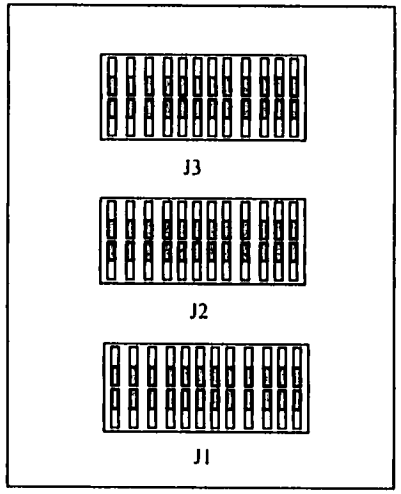




Back View



Top View



Top View (Second Layer)

Figure 29: Breakout-box Overview

In Figure 29, the top plane 1) contains connectors J4 (for FMU 202A), J5 (for FMU 202B), J6 (for FMU 402), J7 (for CDU) and the corresponding power switches and LEDs. It also includes three discrete Ground/Open input switches for the FMUs, namely OLEO, TOGA and LNAV. The back plane 2) provides three connectors for 28 VDC power supply, ARINC I/O DB44 and discrete I/O SCB100. The second layer of the top plane was arranged for J1 (for ARINC I/O signals), J2 (for discrete I/O signals), J3 (for additional common nodes for the Avionics) and the DIO internal circuit board.

3.5.2 Breakout-box Components

The breakout-box components include power supply, LED display circuit, internal circuit board, connectors and cable wiring.

3.5.2.1 Power Supply

The power supply provides the regulated voltage and current source needed to run the avionics. According to the CMA-900 specification, the maximum power of one FMU is 38 W at 28 VDC. From the CMA-2014 (CDU) specification, maximum power is 35 W at 28 VDC.

Hence, for DTB application, three FMU and one CDU will be used, and the total current draw will be 5.33 A (3*38/28+35/28). Finally, for the DTB application, a power supply (6 A, 28 VDC) with a safety margin has been selected.

3.5.2.2 LED Display Circuit

In order to monitor the activity of each power of the avionics, the schematic LED circuit is designed as shown in Figure 30. Once the position of the switch is set to ON, the FMU or CDU will be ON and the corresponding LED will be lit; when the switch position is set to OFF, the FMU or CDU will be OFF and the corresponding LED will become unlit. The maximum current for the standard LED is 20 mA, therefore a 1.5kΩ current-limiting resistor is chosen.

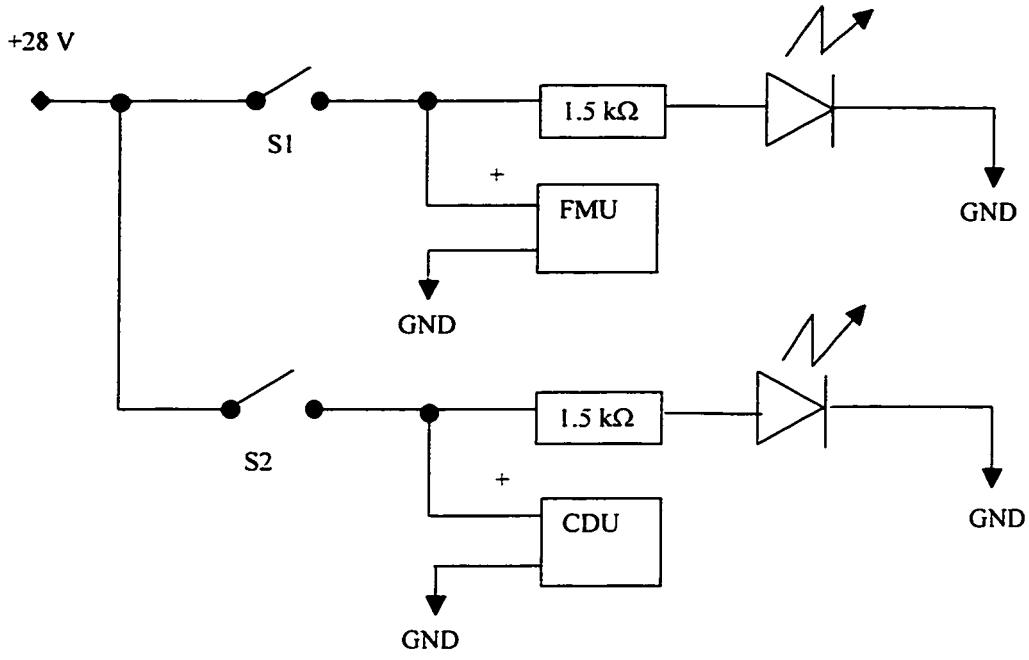


Figure 30: LED Display Circuit

3.5.2.3 Discrete Input Switches

In DTB application, all discrete signals are Ground / Open. Those signals can be controlled either by software through the PCI-6025E card or by single-pole single-throw (SPDT) ON-ON switches [15]. Since only three discrete signals are required for lab testing, namely OLEO, Take Off Go Around (TOGA) and LNAV, a simple switching device will be used as shown in Figure 31. Once it switches to the right, the discrete input signal is OFF, and when it switches to the left, the discrete input signal is ON.

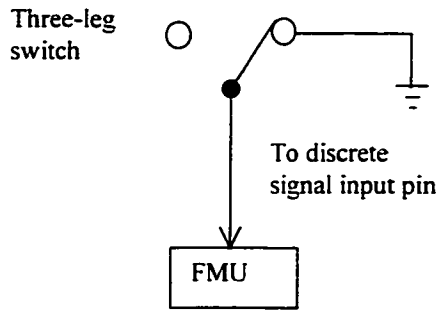


Figure 31: Discrete Input Switch

3.5.2.4 Internal Circuit Board

The purpose of the internal circuit board is to convert between standard low-level logic (5 V from DIO card) and high level logic (28 V for CMA-900).

3.5.2.4.1 Conversion (5V to 28V)

Figure 32 shows the circuit design for converting 5 V logic to 28 V logic. The principle can be explained with the simplified circuit in Figure 32.

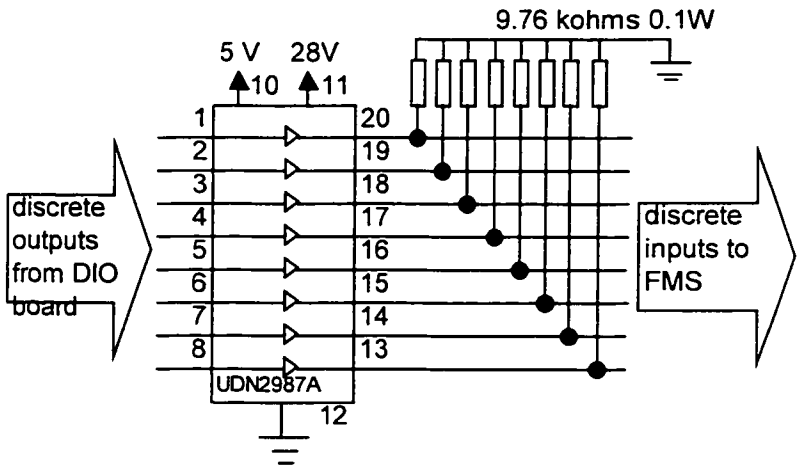


Figure 32: Logic Conversion (5 V to 28 V)

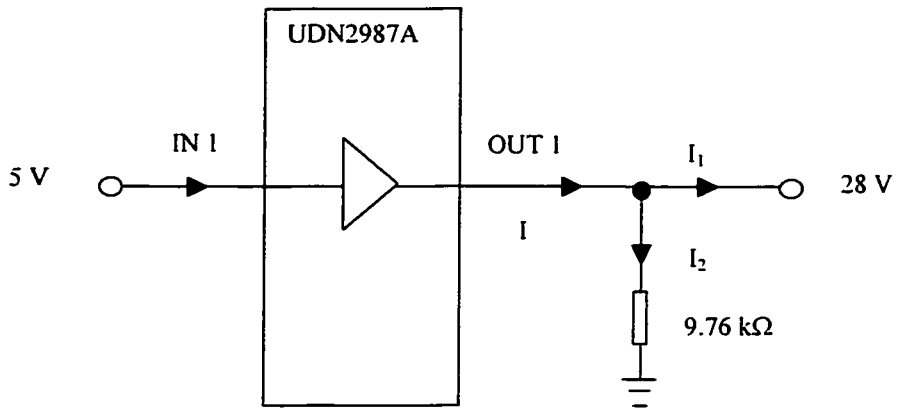


Figure 33: Simplified Logic Conversion Circuit

According to Kirchoff's current law (KCL), we have the relationship in Figure 33: $I = I_1 + I_2$.

Where the 9.76 kΩ resistor provides the necessary current sink in order to meet the specification of the FMU loading current. UDN2987A is 8-channel source driver amplifying voltage from 5V to 28V.

3.5.2.4.2 Conversion (28V to 5V)

On the other hand, FLSIM is needed to accept certain discrete valid signal from FMS for special performance such as Glide Slope guidance of VNAV mode. Therefore, another circuit is designed for converting FMS logic signal into standard logic level. The explanation of one line of circuit can be seen in Figure 34.

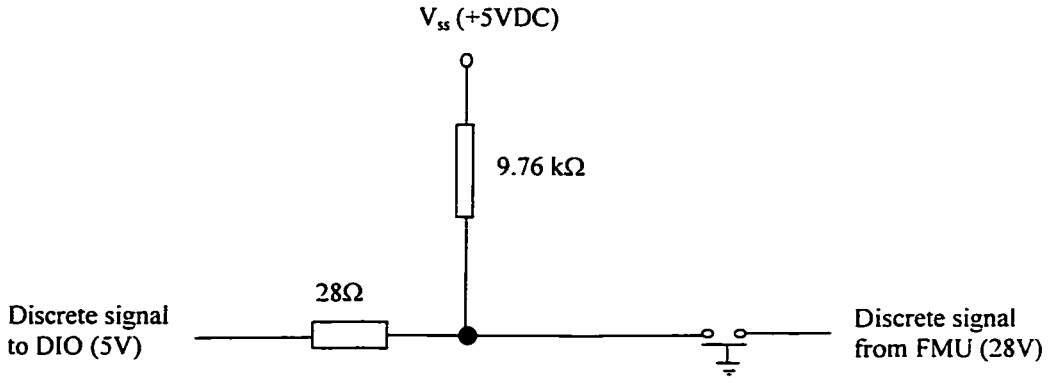


Figure 34: Logic Conversion (28V to 5V)

Where the 28 Ω and 9.76 kΩ resistors provide the necessary loading current for the DIO input, V_{ss} provides 5V voltage (logic 1) to DIO input when FMU output is 28V voltage (logic 1). It is noted that all discrete output signals from FMU are designed in Open/Ground fashion. "Open" means open circuit, logic 1.

3.5.2.4.3 Voltage Regulator

A voltage regulator circuit is added to the internal circuit board for two reasons. First, it provides 5V source to the logic conversion circuits for both directions. Second, it reduces the number of power supply outputs. The circuit is shown in Figure 35.

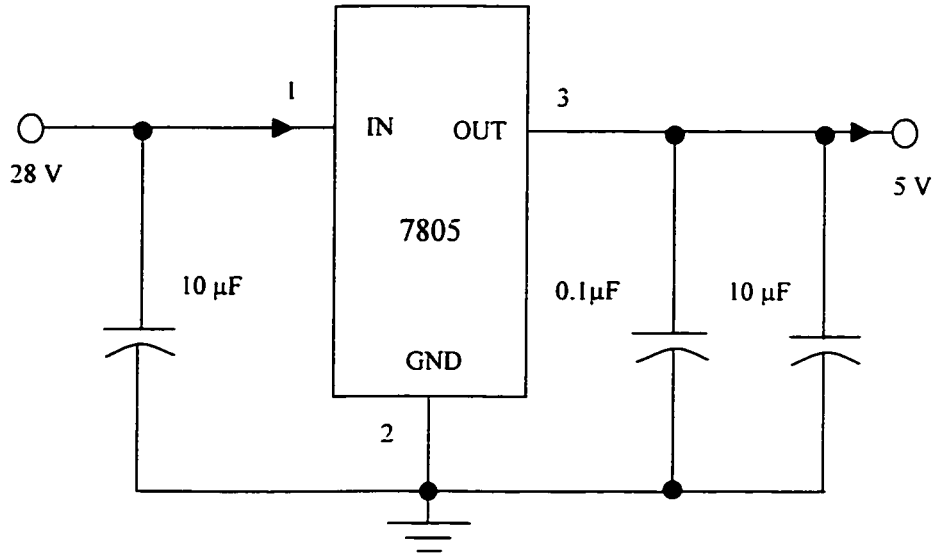


Figure 35: Voltage Regulator

Those capacitors in Figure 35 are called by-pass capacitor that acts as a reservoir and are used to reduce noise from the power supply.

3.5.2.5 Connectors

The selection of connectors is based on three characteristics, namely contact type, pin pattern, and pin count. Two types of connector are used. The first type is the front panel connector used to bring power, ground, and ARINC information to the breakout box. For compatibility with FMS, Zero Insertion Force (ZIF) connectors are chosen. The second type is part of the module design and bridges the signals between breakout box and the avionics. Connector designs are important not only because of the assembly dimensions and the pin arrangement, but also the prevention from vibration and mechanical stiffness required [8].

3.5.2.6 Cables

Cables are used to provide the media for wiring. The only important aspect of it is the cable type. In DTB application, two sizes of cable are used, namely 22 AWG (0.0253 inches) and 24 AWG (0.0201 inches). AWG stands for American Wire Gauge. 22 AWG cables are easier to be clamped and maintained but they are heavier, whereas 24 AWG cables are easier to be broken while clamping but they are lighter. For ARINC information, it is preferable to use 22 AWG cable for 2 reasons. First, bigger cable provides greater stiffness while it is required in DTB application that the wiring is designed to be flexible and since the user might need to change channel configuration for further testing. Second, according to the basic concept of resistance ($R = \rho \frac{L}{A}$, where ρ is resistivity, L is length, A is cross-section area), for same material and length, small diameter wire provides larger resistance. This concept is important since larger resistance would make small signals unstable.

CHAPTER 4

SOFTWARE IMPLEMENTATION

Even though the general description of DTB software architecture has already been introduced in Chapter 2, more detailed background information of the architecture shall be expounded in this chapter, together with the author's contributions in the DTB software implementation.

4.1 Software Requirements

Software refers to the instructions and data which control running of computer hardware. This includes low-level software supplied by the computer vendor to provide a richer user environment, as well as application programs written for a specific user function. Modern applications are written in powerful Higher Order Language (HOL) by which software designers can achieve an impressive level of sophistication and abstraction. This maturing capability allows system functions to be moved from hardware implementation and control to software means, increasing system reliability and simplifying hardware upgrades.

Software technical requirements associated with development may include functionality, performance, reliability and maintainability. There are many program languages each having its advantage and disadvantage. The FORTRAN language is good for scientific computation and the C language for character processing and direct control of hardware program. But both FORTRAN and C language which are often called functional or structured programming are clumsy for large projects and both are not suitable for Internet and visual interface usage [14]. In functional programming, the

designer identifies and packages each function that a software must perform as a separate design entity. These entities, or units, interact to form a system. Visual C++ is another type of programming, which is likely to emerge as the common language for sensor electronics control systems being often called object, oriented programming. The design of object oriented programming comes from the I/O requirements, as well as notions of data encapsulation and data hiding. Each object must have a state and transitions among and between states must be defined. An informal method of identifying objects is to describe the system verbally. The nouns in the description are objects and the verbs are operations performed on the objects [9].

One problem with the functional design approach is that if the aircraft functions change during an upgrade, a major software redesign may be needed because of the complex interface and timing relationships within the code. In principle, object oriented design defines such relationships more generally and may therefore simplify such upgrades. Hence, from easier maintainability point of view, Microsoft Visual C++ compiler has been selected for DTB design.

4.2 DTB Realization

There are numerous problems in realizing the DTB software integration, which was programmed in an easy-to use environment. In order to deal with the complexity of such unstructured environments, a three layer software organization is utilized as shown in Figure 36, consisting of: 1) organization layer – application layer, 2) coordination layer – DLL layer, and 3) execution layer – device driver layer. Each layer is in turn subdivided into additional levels as needed.

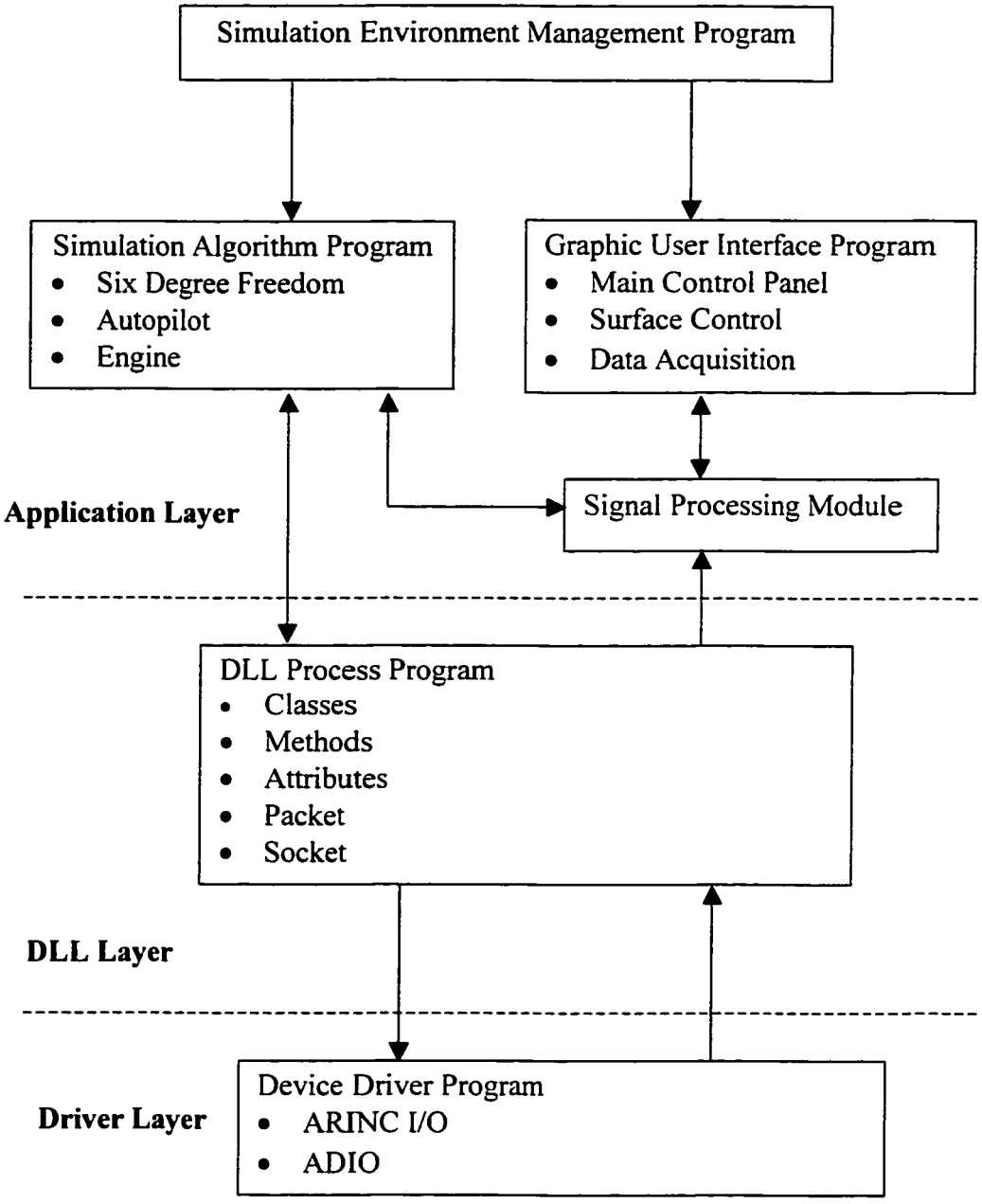


Figure 36: Block Diagram of DTB Software Integration

4.2.1 Device Driver Layer

In the Windows operating system, a programmer cannot access hardware directly from the application level. Hardware access is allowed only from within the operating system itself. In order to access a custom hardware device from the application level, a programmer must do the following:

- Learn how to write a device driver.
- Learn the internals of the operating system he / she is working on.
- Learn new tools for development / debugging in the Kernel Mode.
- Write the application in the User Mode, which accesses the hardware through the device driver written in the Kernel Mode.

Therefore developing a device driver for A429-PC16 hardware interface is not an easy task in the DTB application. A general configuration of the ARINC device driver is important to achieve objective architecture capabilities and keep costs within an affordable range. The device driver was written in Visual C++ and it provides transparent support for Windows NT, furnishing programmers with a function for real-time direct access to port I/O. The step by step function for obtaining the device driver can be found in section 3.1.10 of this paper.

4.2.2 Dynamical Link Library Layer

Threads are execution streams and form the basis of multitasking. That is, each program that runs in Windows has its own main thread to launch the program, and it may start additional threads. Each new thread has its own procedure and executes the code in

that procedure at the same time that other threads are working. Multi-thread programming was used to achieve synchronization application between FLSIM and FMS. The methodology was described in section 2.3.2. Visual C++ Microsoft Foundation Classes (MFC) provide a native class method to achieve the main goal of platform independence for different thread. When working with native methods, the application must load dynamic libraries first.

4.2.3 Application Layer

Many DTB functions are implemented in the application layer. To construct an efficient and practical real time simulation environment, the application layer is divided into three function modules – signal process module, graphic user interface (GUI) module, and algorithm of simulation module.

4.2.3.1 Signal Process Module

In DTB, there are many conventional sensors such as GPS, VOR, DME. INS etc. An application program has the control capability to present inputs to processes and the plants of subsystems so that only the desired outputs or actions are fulfilled.

4.2.3.2 GUI Module

Computer generated information displays provide a promising technology for successful display in GUI design. The Visual C++ dialog based AppWizard supplies much fancy control features for creating GUI display. A desirable display should associate information with the individual primitive activities, and letting an intelligent information manager combine on-line the information associated with each current activity. How to realize the desired display interface is not easy and in DTB. it depends upon customer demand only.

4.2.3.3 Simulation Algorithm Module

In DTB, this module is called *VPIGateway*. Besides using native functions in FLSIM libraries, user customizable properties are provided such as adding a new control law of flight. The conversion between ARINC words and the engineering values has been designed in this module.

4.3 Software Development Examples

In DTB, there are three major essential workspaces, namely *VPIGateway*, *VPIGUI*, and *DTBExecutive*. The author's contribution to the software development is on the Interface PC with *DTBExecutive* workspace. Two small GUIs have been attached to the *DTBExecutive* workspace for monitoring the real-time performance of ARINC-label and discrete-signal statuses, namely *ARINC label GUI* and *DIOGUI* respectively. Since both are assistance software and they are not in the requirement of the DTB, the detailed information would not be shown in the paper. There is another independent GUI which has been created to provide flexibility in changing the ARINC channel property, namely *ARINC Bus Channel Switch* workspace.

4.3.1 ARINC Channel Switch GUI

4.3.1.1 Introduction

The primary purpose of creating this GUI is to provide a flexible tool for switching the existing ARINC channel configuration. Since the DTB will be tested on different models of aircraft, whose different hardware configurations force the DTB to be flexible, the 'ARINC Bus Channel Switch' is essential. Also, in the real testing

environment, this GUI furnishes a faster way to reconfigure the ARINC bus property without specific training in ARINC standards.

4.3.1.2 Software Architecture

To implement this GUI, the first step is to put the complete ARINC channel property file into a certain directory on the Interface PC. The second step is to program the 'discrete' track bar control using the MFC library view class. Each track bar (slider) represents an ARINC channel, the static text to the left of each slider representing the name of the ARINC bus and the tick mark on the slider showing the current channel set up for a certain ARINC bus. The value is initialized at zero, the assumed default value. Any new position setting can be traced by "Now" column, which is just to the right of the slider. The layout of the GUI is shown in Figure 37. The final step is to program list-box control functionality. The first list box reads the original existing property file from a certain directory and puts it into a buffer, where the switching of channels is dealt with. Then the modified property file is copied into the second list box which finally writes and saves the modified property file into the 'DTBExecutive' working directory. This principle applies to "device 2" with other two list box controls as well. The schematic diagram is shown in Figure 38. Note that, to use this GUI, the combined channels (device 1, channel 3, 4, 5) have to be selected first in ascending order.

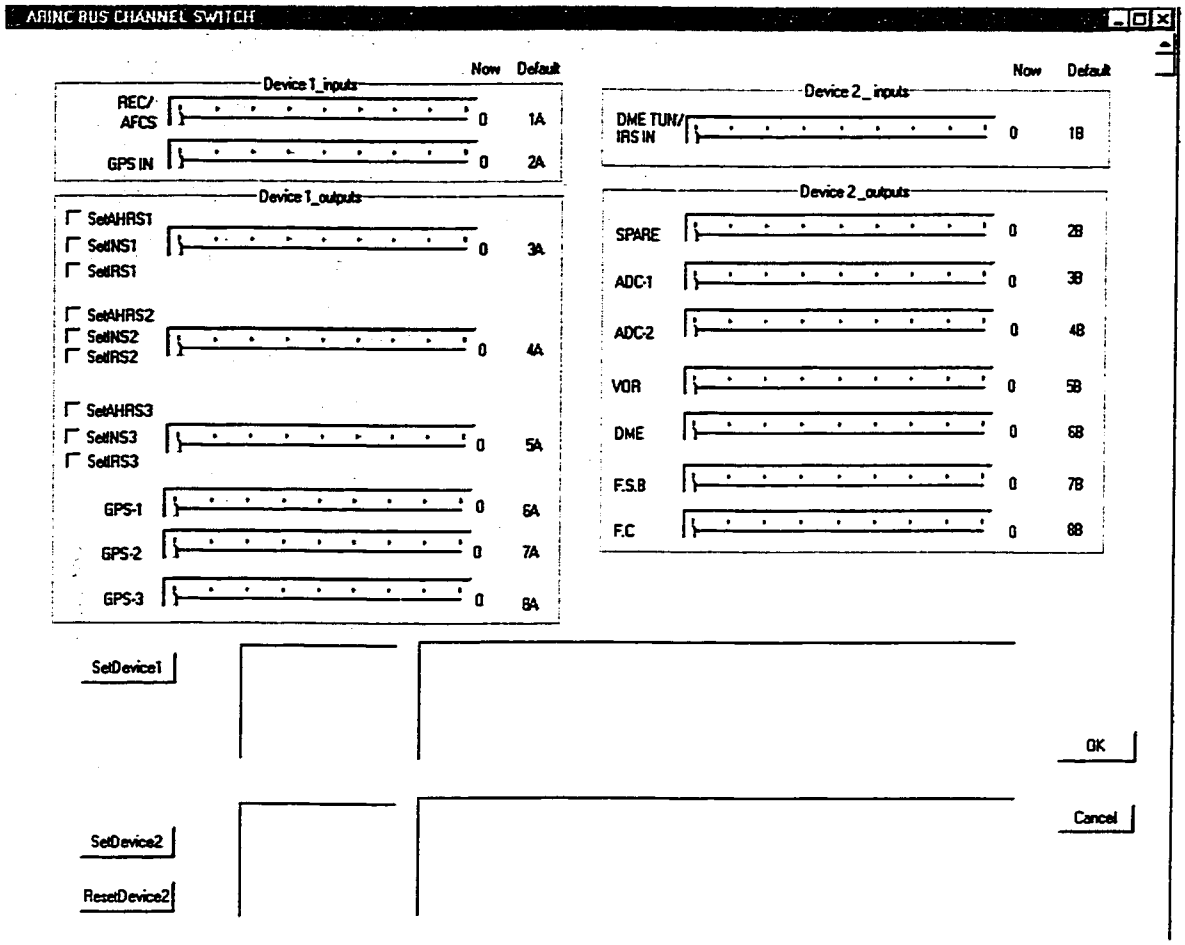


Figure 37: Layout of the ARINC Channel Bus Switch GUI

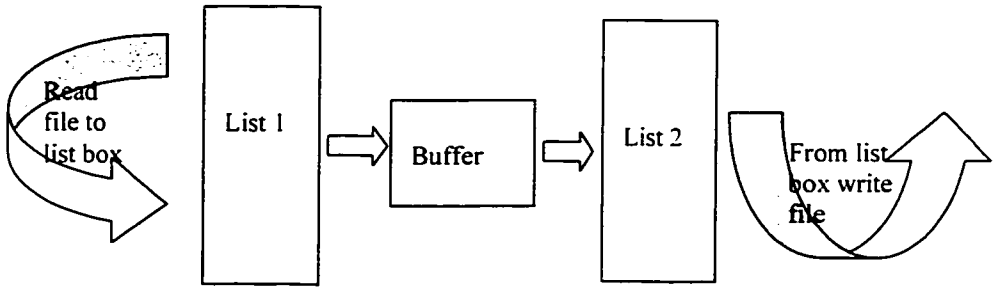


Figure 38: List Box Control Block Diagram

4.3.1.3 Software Components

The major GUI components are the class 'GUI1Dlg' and the header file 'fileload.h'. Class GUI1Dlg contains a program for setting up ARINC bus configuration. All calling functions for class 'GUI1Dlg'are included in 'fileload.h' and are presented in Table 15. The software implement flow chart can be simplified as shown in Figure 39.

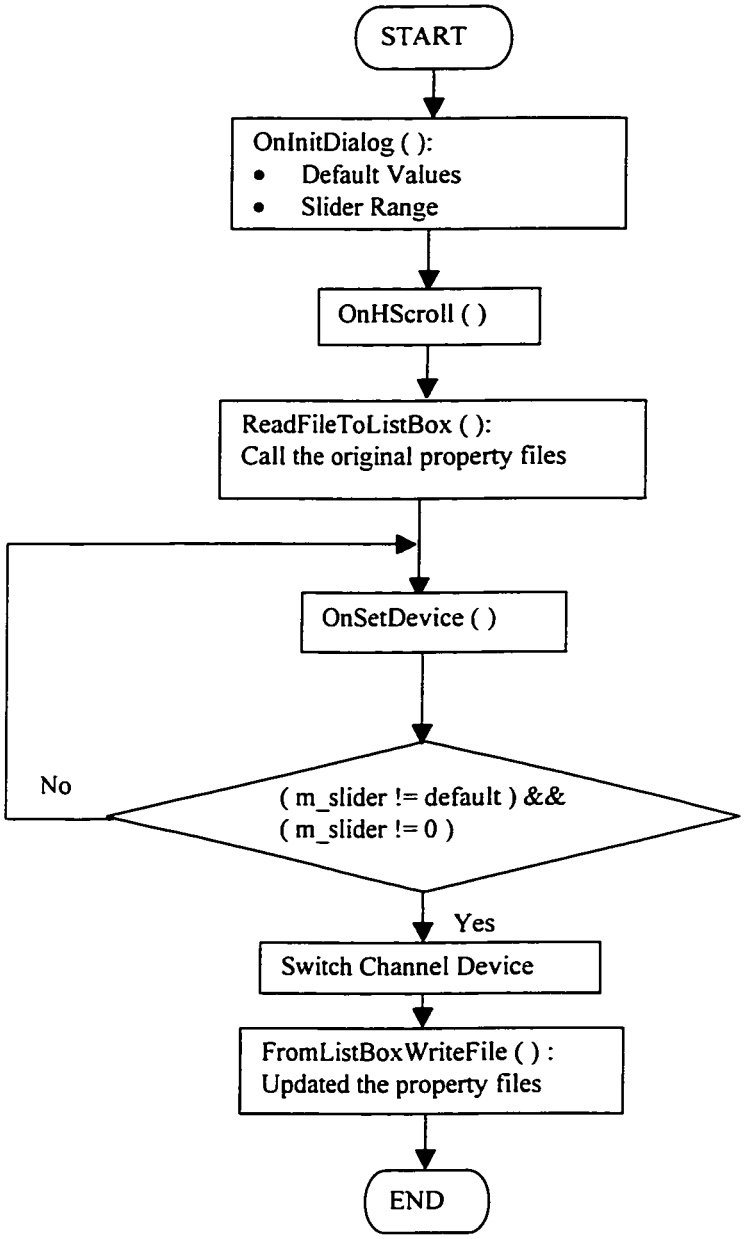


Figure 39: Flow Chart of Channel Switch

Table 15: Functions in Class GUI1Dlg

Functions	Description
OnInitDialog ( )	This function is called by the framework in response to the WM_INITDIALOG message, the message being sent to the dialog box during the **Create**, **CreateIndirect**, or **DoModal** calls, which occur immediately before the dialog box is displayed.
OnHScroll ( )	This method is called by the framework when the user clicks a window's horizontal scroll bar. The SB_THUMBTRACK scroll-bar code typically is used by applications that give some feedback while the scroll box is being dragged.
ReadFileToListBox ( )	Copies existing property files into list box 1 or 2 for device 1 or 2 respectively.
FromListBoxWriteFile ( )	Copies the edited property files into the desired DTBExcutive directory from list box 3 or 4.
FindStringInListBox ( )	Finds the specific string in a list box that contains the specified prefix, without changing the list-box selection.
SwitchChannelDevice1 ( )	Implements all channel resetting for device 1 with all options.
SwitchChannelDevice2 ( )	Implements all channel resetting for device 2 with all options.
OnSetAHRS ( )	For the combination channels 3A or 4A or 5A, selects AHRS bus for the specified channel.
OnSetINS ( )	For the combination channels 3A or 4A or 5A, selects INS bus for the specified channel.
OnSetIRS ( )	For the combination channels 3A or 4A or 5A, selects IRS bus for the specified channel.
OnSetDevice1 ( )	After selecting AHRS/INS/IRS, switches all necessary channels for device 1 and then saves to desired working directory.

OnSetDevice2 ( )	After selecting AHRS/INS/IRS, switches all necessary channels for device 2 and then saves to desired working directory.
OnResetDevice2 ( )	If there are no changes in device 2, copies the default setting for device 2 directly into the desired working directory.
AddString ( )	Adds a string to a list box.
GetDlgItem ( )	Retrieves a handle to a control in the specified dialog box.
GetPos ( )	Retrieves the current position of the slider in a slider control.
GetText ( )	Retrieves a string from a list box.
GetCount ( )	Retrieves the number of elements in this list.
SetRange ( )	Sets the range (minimum and maximum positions) for the slider in a slider control.
SetDlgItemText ( )	Sets the text of a control in a dialog box.
SetCurSel ( )	Selects a string and scroll it into view, if necessary. When the new string is selected, the list box removes the highlight from the previously selected string.
ResetContent ( )	Removes all items from a list box.

4.3.2 ARINC I/O Configuration Management

For easy maintenance, instead of hard coding, five new classes ('Arinc429Channel', 'Arinc429Device', 'Arinc429TCB', 'Arinc429RCB' and 'DTBProperties') have been inserted into the workspace of *DTBExecutive*. Also, two property files called 'A429Config1.prop' and 'A429Config2.prop' have been built under the same directory, thus all the ARINC words properties such as frame time, schedule time and repeat rate can be entered into the file without changing other classes. The details for setting up the property files can be found in Chapter 3.

4.3.2.1 Class Arinc429Channel

This class ("Arinc429Channel.cpp" under the workspace of "DTBExecutive") contains a program to initialize and set up the ARINC 429-PC16 channels for transmitting and receiving on the 429 buses. Functions of this class are shown in Table 16.

Table 16: Functions of Class Arinc429Channel

Functions	**Description**
Arinc429Channel ( )	Creates an Arinc429Channel object for construction.
InitializeAsTransmitter ( )	Initializes the transmitting channels and calls the specific library init routines.
InitializeAsReceiver ( )	Initializes the receiving channels and calls the specific library init routines.
haltChannel ( )	Halt the Transmit Control Block chain, disabling the receivers.
writeWord ( )	Gets the address of the data buffer from the Transmit Control Block and then writes the data to a chain link transmit data buffer. Remarks: there are three 16-bit words for each Transmit entry. The first word is the transmit control word, followed by the lower 16-bit of ARINC data word, then by upper 16-bit of the ARINC data word.
readWord ( )	Reads the last received word, followed by its three associated time stamp words and converts a label to or from a bit-swapped value. Remark: for ARINC specifications, the 8-bit label has a

	reversed bit order.
GetWordAsULONG ( )	Drops out the first 10 bits and reads the 19-bit data only.
GetArincWordAsULONG( )	Reads the current values of the last received 32-bit Arinc word.

4.3.2.2 Class Arinc429Device

This class ("Arinc429Device.cpp" under the workspace of "DTBExecutive") contains a program to parse the configuration file, initialize a device and declare certain functions at device level. Functions of this class are shown in Table 17.

Table 17: Functions of Class Arinc429Device

Functions	**Description**
Arinc429Device ( )	Creates an Arinc429Device object.
shutdown ( )	Stops the I/O, closes the devices and exits.
isConfigured ( )	Returns 'correctlyConfigured'.
writeWord ( )	Transmits data. Remark: the data does not include the label and SDI bits.
readWord ( )	Receives data.

4.3.2.3 Class Arinc429TCB

This class ("Arinc429TCB.cpp" under the workspace of "DTBExecutive") includes all class members to set up transmitting command blocks. Functions of this class are shown in Table 18.

Table 18: Functions of Class Arinc429TCB

Functions	Description
Arinc429TCB ( )	Creates an Arinc429TCB object for construction.
getTCB ( )	Returns the number of transmitting command block.
getChainID ( )	Returns the ID number of a chain.
hasLabel ( )	Returns TRUE if it is correctly configured.
writeWord ( )	Gets the address of the data buffer from the Transmit Control Block and then writing the data to a chain link transmit data buffer. Remarks: there are three 16-bit words for each Transmit entry. The first word is the transmit control word, followed by the lower 16-bit of ARINC data word, then by upper 16-bit of the ARINC data word.
isConfigured ( )	Returns logic state of "correctlyConfigured".

4.3.2.4 Class Arinc429RCB

This class ("Arinc429RCB.cpp" under the workspace of "DTBExecutive") serves to set up a receiver control block. Functions of this class are shown in Table 19.

Table 19: Functions of Class Arinc429RCB

Functions	Description
Arinc429RCB ( )	Creates an Arinc429RCB object for construction.
hasLabel ( )	Returns TRUE if it is correctly configured for a receiver.
isConfigured ( )	Returns logic state of "correctlyConfigured" for a receiver.
readWord ( )	Reads the last received word, followed by its three associated time stamp words and converting a label to or from a bit-swapped value. Remark: for ARINC specifications, the 8-bit label has a reversed bit order.

4.3.2.5 Class DTBProperties

This class includes all attributes and methods in order to match and read the 'A429Config.prop' text file. Functions of this class are shown in Table 20.

Table 20: Functions of Class DTBProperties

Functions	**Description**
DTBProperties ( )	Creates a DTBProperties object for construction.
getShort ( )	Returns Cstring key and passes the value to the pointer short.
getInt ( )	Returns Cstring key and passes the value to the pointer int.
getLong ( )	Returns Cstring key and passes the value to the pointer long.
getFloat ( )	Returns Cstring key and passes the value to the pointer float.
getDouble ( )	Returns Cstring key and passes the value to the pointer double.
getString ( )	Returns Cstring key and reads the value from the Cstring.
Put ( )	Inserts the character and dynamically creates "Property" text file.
Load ( )	Reads the "A429Config.prop" text file.
Save ( )	Writes the "A429Config.prop" text file.
Dump ( )	Dumps the contents of "A429Config.prop" and checks if it is correct.

4.3.2.6 Class DTBArincIOThread

This class is an individual running program within the main Executive Thread Loop, which contains a separate execution path. The general information about "Thread"

is introduced in Chapter 2. There are three main methods under this class as shown in Table 21.

Table 21: Functions of Class DTBArincIOThread

Functions	**Description**
getDataFromBus ( )	Sets a pointer to the readable double buffered bus and reads the contents from the layer.
performIO ( )	Receives data from the readable double buffered bus and transmits to required channels. REMARK: In order to cancel out the effect of delay when updating data in double buffer container, multi-rate flag "tickmarkIndex" is set so that the receiving data can be updated with different specific frequency.
SetDataIntoBus ( )	Writes the new data into the writable layer of the double buffer bus.

CHAPTER 5

SYSTEM VALIDATION

Since this DTB was mostly involved with technical implementation, particular in the area of software development, not much theory can be presented since verifying the whole DTB system depends mostly on data comparison or navigation working experience. To test both software and hardware in the DTB system, three stages have been passed through, namely loop-back testing, LNAV testing and VNAV testing.

5.1 First Stage: Loop-back Test

As described in Chapter 4, the software development can be divided into three levels. To ensure that the device driver layer is working, the loop-back test has to be passed first. Loop-back is a simplified system without breakout-box connection. The Simulation PC converts certain signals (say Heading and True Air Speed) from engineering values to ARINC 32-bit words and sends them into a double buffer container, then Interface PC picks up the data and transmits them on different ARINC channels (say channel 5 and 6). With loop-back connection, in this example, channel 5 and channel 6 were directly connected to channel 1 and channel 2 respectively. When the expected results are achieved, it means that all received information from channel 1 and channel 2 are perfectly identical with transmitted information from channel 5 and channel 6 respectively without any run-time errors.

Similar testing has been held with different channels including Device 2. It should be noted here again that A429-PC16 has two devices, each device having 8

channels. For the loop-back test, we always assign the upper 4 channels as transmitters and the lower 4 channels as receivers in each device.

At this stage, it should be ensured that the primary functions of the device driver layer are working properly, including ARINC channel configuration file, ARINC device initialization, transmitting and receiving.

5.2 Second Stage: LNAV Test

LNAV verification is one of the primary objectives of the DTB. To begin the test, besides having the device driver layer set up properly, all necessary input buses including the conversion between the FLSIM engineering values and ARINC words have to be all correct and stable. The pin-connections in the breakout box is a main concern at this stage, and loose contacts have to be eliminated totally. After making sure there is no poor contact on the wiring, the LNAV testing block diagram can be replaced with the sketch in Figure 40.

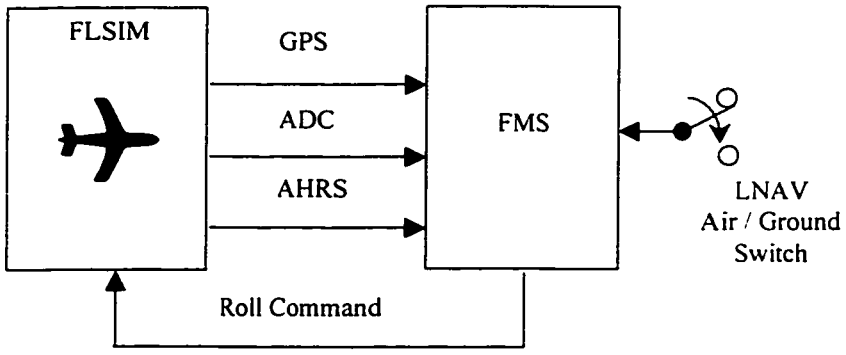


Figure 40: LNAV Test Block Diagram

Figure 40 indicates that the only input needed from the FMS for lateral navigation is the roll angle. However, the FMS needs other information to compute the proper steering command. It is necessary to make sure that the FMS is reading correctly the airspeed, the heading, the baro-corrected altitude and the position from the FMS. After verifying that the ground speed (GS) is greater than 140 knots and all GPS validity and integrity flags are as transmitted by the DTB, DTB is ready to have the LNAV test with a certain flight plan. Figures 40 to 43 show the active progress of LNAV with a flight plan from CYMX (Montreal) to CYYZ (Toronto).

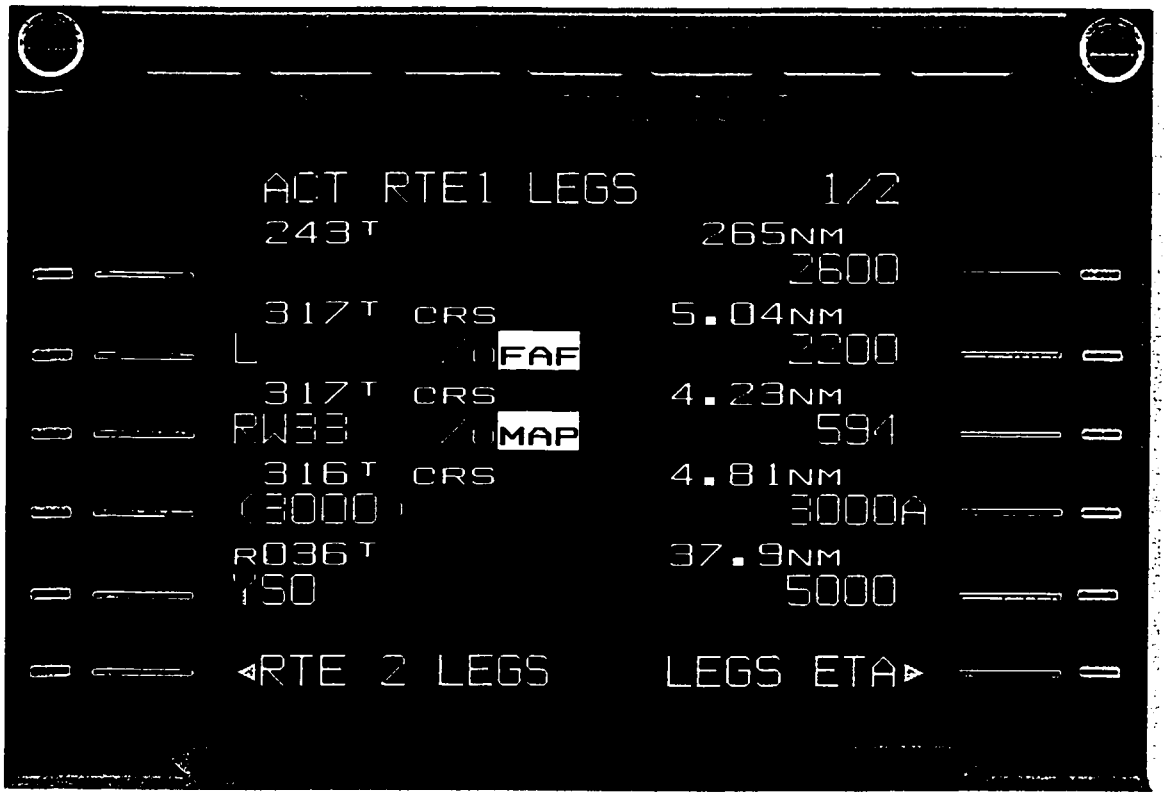


Figure 41: Flight Plan Along Legs (Waypoints)

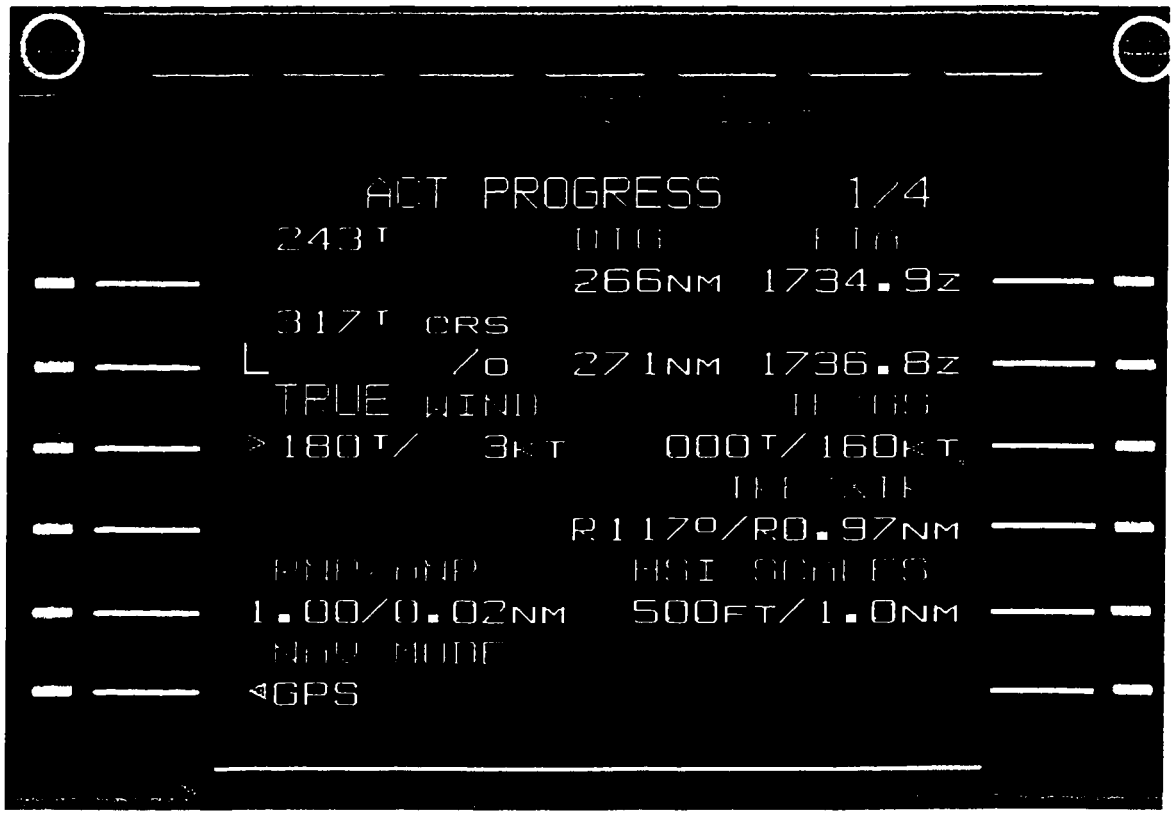


Figure 42: LNAV OFF Status

Figure 42 shows that the course of waypoint AGBEK is 243 (degrees True) with Ground Track Angle Error (TKE) Right 117 degrees before LNAV is engaged.

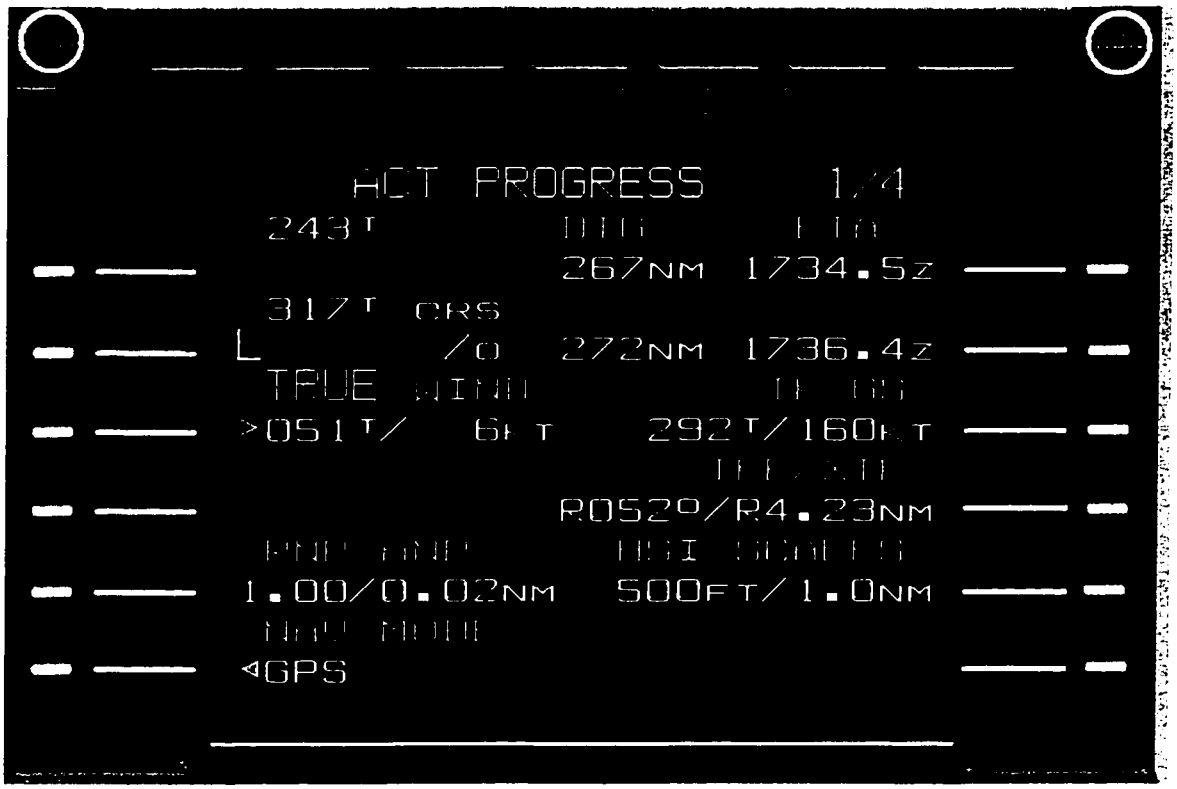


Figure 43: LNAV ON Status

Figure 43 indicates that once the LNAV is engaged, the Heading changes as the aircraft turns. It is suggested that the TAS should be set in the range of 160 – 170 knots to prevent navigation overshoot. This diagram also shows that the TKE is decreasing and the aircraft is tracing the prescribed path (Heading).

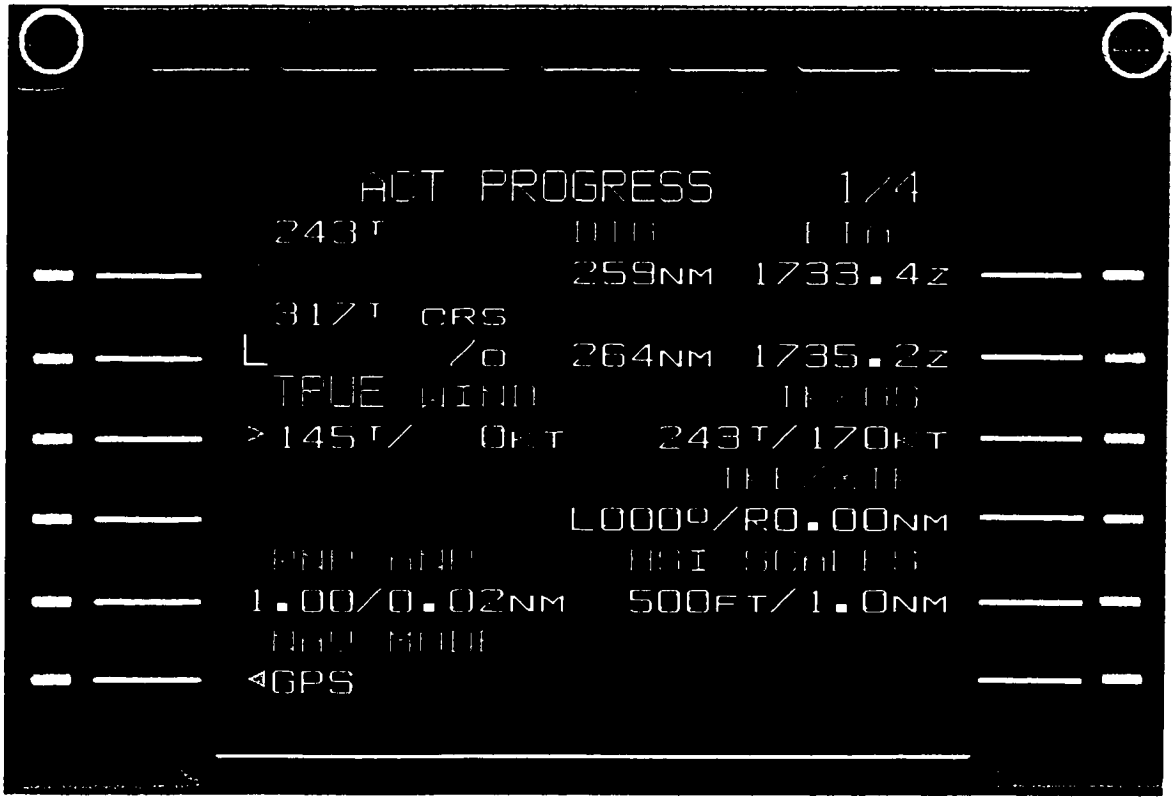


Figure 44: LNAV Completed

Figure 44 shows that the aircraft is following the roll command, and once the Heading reaches the expected course, the roll angle becomes zero and the Heading a constant. For easier comparison, the wind speed is set to zero. When LNAV is completed, it is verified by looking at both TKE and XTK which become zero and the Heading equal to 243 (Degrees True). To understand those navigation relationships, see Figure 26.

5.3 Third Stage: VNAV Test

The VNAV test is another primary objective of DTB. Instead of roll command from the FMS, the only input for FLSIM is FMS Vertical Deviation, and once the VNAV is engaged, FLSIM should follow the Glide slope control law during the approach phase. The components of the Approach / Landing phase are shown in Figure 45.

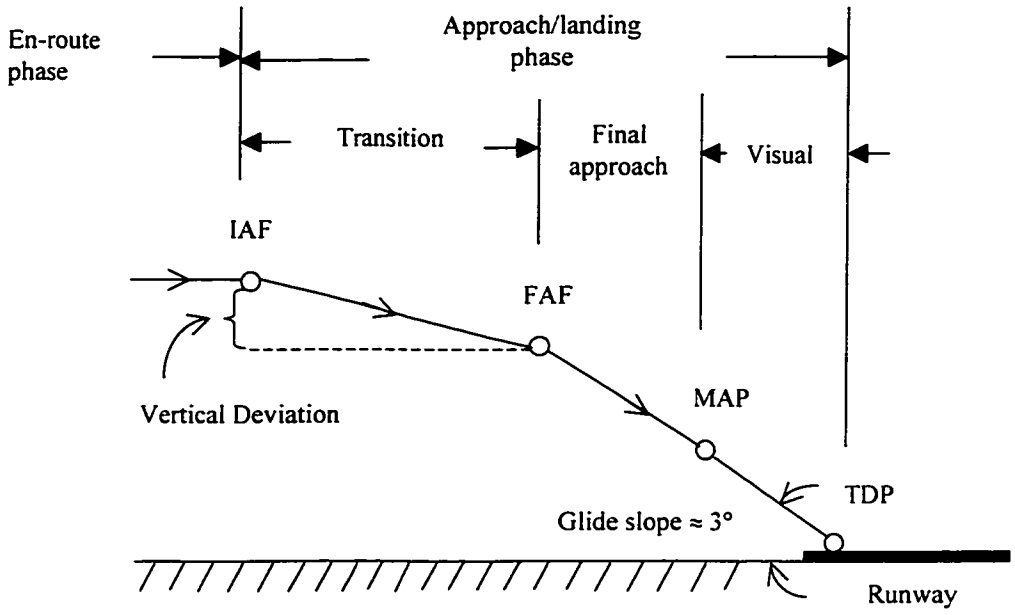


Figure 45: Components of Approach/Landing Phase (side view)

Where: IAF = initial approach fix

FAF = final approach fix

MAP = missed approach point

TDP = touch down point

Besides having all input verified, the LNAV should be activated correctly as Figure 46 shows and, in order to go into GPS approach mode, the FMS makes Predictive Real-time Receiver Autonomous Integrity Monitoring (RAIM) requests for the final approach fix (FAF) and missed approach point (MAP).

As part of the request, the DTB shall receive the Destination waypoint ETA label on the GPS input channel and acknowledge reception of this label by transmitting the Destination waypoint ETA label back and a Destination waypoint HIL label in particular format on the GPS output channel. The Label Destination waypoint HIL should contain a user-specified horizontal integrity limit. The default value should be 0.1 nm. After programming a certain format with counters properly (Appendix G has detailed information), its value should be entered on the navigation status window (Predict RAIM HIL) as shown in Figure 46.

Figure 46: GPS with RAIM

To further ensure that the VNAV is good for approach, it is essential to verify that the RAP is 0.3 nm when the aircraft is getting FAF within 2 nm as shown in Figure 47. For the specification of the Requirement of Approach Performance, see Table 14 in this paper.

Figure 47: Entering GPS Approach Zone

Up to this point, all work has been completed on the Interface PC side including software development, breakout box and hardware components. However, the result of the VNAV test is not perfectly matched as desired since the Glide Slope control law from FLSIM library does not support Vertical Deviation as input, accepting glide slope only. The discussion will be presented in Chapter 6 and the remaining task of implementing the new Glide slope control law will be left for future work.

CHAPTER 6

CONCLUSION

6.1 Background Review

In the future, aircraft design performance and control will be focused on the capabilities of the flight management system (FMS). Instead of aircraft flight instruments such as air speed indicator, altimeter, artificial horizon, automatic direction finder, climb rate indicator, compass, course indicator, heading indicator, horizontal situation indicator, omni-bearing indicator, radio magnetic indicator and turn coordinator, the FMS will be completely integrated to supply all instantaneous information. The pilot will no longer rely on mental calculations to determine the position or attitude of his aircraft. Instrumentation and displays of a FMS will show current status together with command information, such as the proper altitude and speed required for making good a predetermined line of flight. Most normal in-flight corrections will eventually be automated so that the pilot need not interpret the data to make the necessary corrections.

GPS holds the highest priority of a FMS external navigation sensor, approaching very closely the ideal worldwide navigation aid with an accuracy of a few meters. However, GPS is expensive to operate, costing nearly a billion dollars per year for the replacement of satellites and the maintenance of the ground-control and monitoring network [3]. The cost of collecting user charges would exceed the revenue that could be extracted from navigation-only users. Hence, in the next generation, GPS transmitters will be installed on low-cost communication satellites as a way to augment the GPS network or as a low-cost replacement for dedicated GPS satellites.

6.2 Hardware Summary & Discussion

ARINC 429-PC16 interface plays the most important role in DTB hardware integration. A429 provides concurrent simulation of multiple transmit channels, monitoring of multiple receive channels, sequential monitoring, advanced interrupt services, and high speed host operations. The device driver is the core of ARINC specification. Compared with A429, it should be noted that MIL-STD-1553 systems offer more robustness, simplicity of hardware and software interfaces, and better performance, but are more expensive [8].

Breakout box design is able to respond to changing demands and situations and is oriented towards satisfying many requirements at once. It is maintained not only within specified limits, but it is also done cheaply, quickly, and efficiently. The grade of connector selection, module insertion and extraction forces must be low enough to simplify maintenance procedures, yet high enough to assure good electrical contact even during highly dynamic force loading. In short, the usual commercial breakout box design with conventional contacts is unlikely to offer adequate performance for FMS. On the other hand, the military connectors are likely to be much too expensive for workstation designs, and in the near future it is unlikely that unique military connector designs will be available.

6.3 Software Summary & Discussion

A vigorous but disciplined approach to technology development is the key to both achieving Objective Architecture capabilities and keeping costs within affordable range. Three layers are designed to appear in DTB, namely, driver layer, DLL layer and

application layer. In DTB, a real flight management can be used and accessed by the application program through the DLL layer and device-driver hardware layer. The DTB execution is responsible for the interface to FMS and handling of the database through a bi-directional channel A429. During run time, FMS sends labels into ARINC channels with a specific command protocol. For real time interfacing with GUI, a dedicated Ethernet is used. This arrangement takes advantage of a computer's internal memory-to-memory transfer speed, especially when large amounts of data need to be transferred [16].

Object oriented design is an abstract, sophisticated method of integrating a large application project. An object has state, behavior, and identity: the structure and behavior of similar objects are defined in their common class [11]. This design originated with software developers, but is rapidly gaining acceptance as a systems design model as well, due to its close relationship to the process maturity model and the increasingly important role that software plays in systems design. It should be mentioned that Ada is superior to C++ in terms of safety and reliability [23]. Originally sponsored by the US Department of Defense, the Ada programming language is designed for applications where correctness, safety, and reliability are the prime goals. As an internationally standardized object-oriented programming language (OOP), Ada is well suited for developing reusable components, real-time and parallel processing systems, and interfaces with systems written in other languages. Also, it should be mentioned that the operating systems for embedded, real-time computers have rather harsh performance requirements. Window NT (4.0) is not a real real-time operating system, UNIX is not designed for real-time application either, but for multi-user, multi-tasking system [24]. The OS and Ada run

time system provide linkage between low-level hardware and application software. This software can have a tremendous impact on performance in terms of execution speed, memory capacity utilization, and determination.

6.4 Future Work

The DTB has reached an application level as a testing tool for FMS. The whole software architecture of DTB has been proven to be a solid system. The enhanced GUI has been developed on the Simulation PC and foundation breakout box designed. However, there are still some tasks remaining for future work.

6.4.1 Breakout box Modification

The breakout box provides hardware interface between the Interface PC and FMS. Major modification will be on the internal DIO circuit board. This circuit was primarily designed for both CMA-900-202 and CMA-900-402 models. However, the discrete logic has been modified in the 402 model and it is not compatible with the older design. Hence, an enhanced compatible internal DIO circuit board should be built so that FMS discrete output signals can be software-controlled through the Simulation PC. As an optional feature, power supply with a fan could be put into the breakout box in order to make a complete portable unit.

6.4.2 Software Development

At present the ARINC channel switch GUI has been programmed as a semi-automated tool to reconfigure ARINC channel properties from text files only. It could be

updated to a fully-automated tool by adding a few control buttons and updating the *DTBArincIOThread* simultaneously. Adding the corresponding CDU IO port numbers into the GUI is also desirable.

Although our aim is to develop a foundation dynamic test bed for FMS, the more flexible we make the DTB, the safer it would be for flight with FMS in the real world. Hence, a complete tactical model should be added in terms of terrain, weather, electromagnetic and infrared environment.

6.4.3 Testing Aspects

Under the VNAV mode from FMS, the development of a Glideslope control law has started, but the result did not come up to FMS expectation, and modification should be carried on in the next phase. Also, there are many control laws awaiting development once they are well defined by CMC. Furthermore, other lower priority navigation modes such as VOR, DME and IRS should be tested once sufficient commitments are obtained from CMC.

6.5 Concluding Remarks

By the end of the thesis, the author's contributions on the DTB are included as follows:

- Designed and implemented real-time software architecture for the DTB using OOP features, MFC, ActiveX controls, multithreaded programming with C and Microsoft Visual C++.

- Implemented the system software including a GUI, an executive program, device drivers for I/O cards of ARINC 429 PC-16 and National Instrument PCI-6025E, a client-server Ethernet TCP/IP network software and a data acquisition software.
- Designed and implemented a breakout box and internal circuit board (A/D, D/A).
- Verified the DTB by identifying and improving the software modules.
- Created a user guide, a programmer guide and an installation guide.
- Familiarized with electrical hardware units of GPS and FMS.

Even though each dynamic interface simulation does have limitations and it is not possible to optimize the DTB in all areas simultaneously, the DTB research has benefited from collaboration between CMC and Concordia University. The collaboration is a vehicle for exchanging working experience, information and models, as well as sharing the results of relevant research.

This thesis has presented an overview of the current work undertaken as a joint project between CMC and Concordia University – Dynamic Test Bed for Flight Management System. The following concluding remarks are indicated:

- The DTB as a demanding and capability adequate and worthwhile testing tool is just becoming available.
- The DTB could provide significant cost and operational benefits in verifying FMS dynamic performance.
- The advanced solid software architecture of the DTB is vital to the success of further research on development of this valuable tool for use in this field.

- The collaboration continues to be a valuable element in augmenting DTB's capability.

REFERENCES

1. S. B. Fishbein, Flight Management Systems, London: Westport, 1995.
2. B. Kendal, Manual of Avionics, 3rd ed., London: Blackwell Scientific Publications, 1993.
3. M. Kayton and W. R. Fried, Avionics Navigation Systems, 2nd ed., New York: John Wiley & Sons, 1997.
4. ARINC 429 Reference Manual, 2nd ed., SBS Avionics Technologies, 1999.
5. DAQ PCI-6025E User Manual, National Instruments, 1998.
6. CMA-900 Internal Reports, CMC, 1999.
7. Operator's Manual CMA-900/CMA-2014, CMC, 1999.
8. J. R. Newport, Avionic Systems Design, Florida: CRC Press, 1994.
9. Visual C++ 6.0 Programming,
10. Visual C++ 6.0 MSDN Documentation, 1999.
11. S. Holzner, Visual C++ 6.0 Programming, New York: John Wiley & Sons, 1998.
12. Getting Started with the ARINC 429 PC8/PC16, SBS Avionics Technologies, 1999.
13. What is New in the Integrated Avionics Libraries, SBS Avionics Technologies, 1999.
14. X. Qiu and F. Liao, "Visual Flight Simulation of UAVs in Real-time Programmed in JAVA Language", Beijing University of Aeronautics & Astronautics, 1999.
15. AMP, http://www.digikey.com
16. S. B. Anderson, "Historical Review of Piloted Simulation at NASA Ames", NASA Ames Research Center, 1995.
17. M. E. C. Roberts, "Simulated Visual Scenes – Where Are the Critical Cues?", Thomson Training & Simulation, 1995.

18. K. Alvermann and S. Graeber, "The RTSS Image Generation System", Institute of Flight Mechanics, Germany, 1995.

19. F. D. Heran, " Tactical Environment Servicers", SOGITEC Division Electronic, France, 1996.

20. H. A. J. M. Offerman, "Advancements in Simulator Technology and Application", National Aerospace Laboratory, Holland, 1994.

21. S. W. Paris, "Optimal Trajectories by Implicit Simulation", Boeing Final Report to the Air Force Flight Dynamics Lab, 1997.

22. J. Ball, et al., "Military Spaceplane Mobile Operations Test Bed", AIAA, Defense & Space Programs, 1997.

23. D. L. Bryan, Exploring Ada, NJ: Prentice-Hall, 1992.

24. Bell Labs Innovations, http://www.bell-labs.com.

25. CMA-900 GPS/FMS Technical Notes, CMC, 1999.

26. J.L. Tomsic, SAE Dictionary of Aerospace Engineering, 2nd ed., SAE International, 1998, p.418.

27. Minimum operational performance standards for airborne supplemental navigation equipment using global positioning system (GPS) / RTCA , Washington DC: RTCA, c1991.

28. Moura and Jose Manuel Fonseca de, Narrow-base passive systems theory with applications to positioning and navigation, Cambridge: Massachusetts Institute of Technology, Research Laboratory of Electronics, 1976.

29. Guidance and control, New York: Academic Press, 1962-1964.

30. CMA-900 PTT GUI, CMC, 1999.

31. G. B. Gilyard, J. Georgie and J. S. Barnicki, "Flight Test of an Adaptive Configuration Optimization System for Transport Aircraft",Dryden Flight Research Center, Edwards, California, 1999.

32. Boeing, "Open Control Platform for Uninhabited Air Vehicles", DARPA ITO Sponsored Research, 2001.

33. Engineering Flight Simulators, Cranfield University, College of Aeronautics, UK, 2000.

34. B. W. Parkinson and P. Axelrad, "Autonomous GPS Integrity Monitoring Using the Pseudorange Residual", *Navigation*, Vol. 35, No. 2, Summer, 1988.

35. G.B. Green, P.D. Massatt and N.W. Rhodus, "The GPS 21 Primary Satellite Constellation", *Navigation*, Vol. 36, No. 1, Spring 1989.

36. M.A. Sturza, "Navigation System Integrity Monitoring Using Redundant Measurements", *Navigation*, Vol. 35, No. 4, Winter 1989.

37. M.A. Sturza and A.K. Brown, "Comparison of Fixed and Variable Threshold RAIM Algorithms", *Proceedings of ION GPS-90.* Colorado Springs, CO., 1990.

38. R.G. Brown, "A Baseline GPS RAIM Scheme and a Note on the Equivalence of Three RAIM Methods", *Navigation*, Vol. 39, No. 3, Fall 1992.

39. M. Durand and A. Caseau, "GPS Availability, Part II: Evaluation of State Probabilities for 21 Satellite and 24 Satellite Constellations", *Navigation*, Vol. 37, No. 3, Fall 1990.

40. A.G. Gower, "Putting a number on GPS Integrity: The IBM GPS Integrity Study for the DoD", *Proceedings of ION GPS-91.* Albuquerque, NM. 1991.

41. B.S. Pervan, C.E. Cohen and B.W. Parkinson, "Autonomous Integrity Monitoring for Precision Approach using DGPS and a Ground-Based Pseudolite", *Proceedings of ION GPS-93.* Salt Lake City, UT., 1993.

42. J.L. Farrell and F. von Graas, "Statistical Validation for GPS Integrity Test", *Proceedings of ION GPS-91.* Albuquerque, NM. 1991.

43. J.M. Davis and R.J. Kelly, "RNP Tunnel Concept for Precision Approach with GNSS Application", *Proceedings of the ION 49th Annual Meeting.* Cambridge, MA., 1993.

44. P.W. McBurney and R.G. Brown, "Self-Contained GPS Integrity Monitoring Using a Censored Kalman Filter", *Proceedings of ION GPS-88.* Colorado Springs, CO., 1988.

45. S. Bancroft and S.S. Chen, "Integrity Monitoring Using Bayes' Rule", *Proceedings of ION GPS-91.* Albuquerque, NM. 1991.

46. N. F. Chui, "The Development of a Dynamic Test Bed for Flight Management Systems", Thesis (M. A. Sc), Department of Mechanical and Industrial Engineering, Concordia University, 2002.

APPENDIX

A. "Sbs_dev.cfg" File

```
;*=============================================================================
;* Copyright (c) 1999 by SBS Technologies, Inc.
;*              2400 Louisiana Blvd., NE
;*              AFC Building 5, Suite 600
;*              Albuquerque, New Mexico 87110
;*              Technical support (toll free)
;*              877-TECHSBS (877-832-4727)
;*=============================================================================
;* SBS INTEGRATED AVIONICS LIBRARY Version 6.1 (08 Mar. 1999)
;*=============================================================================
;*
;* NAME: sbs_dev.cfg
;*
;* DESCRIPTION:
;*   This file contains the information required to initialize one or
;*   more SBS device(s).
;*
;* FUNCTIONS:  None
;*
;*=============================================================================

[DEVICE=1]

; Base Memory Address in hex
base_address=D0000h

; PCMCIA DSP Bootloader program filename (default bootload.txt)
; bootloader=bootload.txt

; Device Type ( M1553_cPCI_3U,    A429_PC8,
;          M1553_cPCI_6U_1,  A429_PC16_1,
;          M1553_cPCI_6U_2,  A429_PC16_2,
;          M1553_FW5000,     A429_PC104,
;          M1553_PC104,      A429_PCMCIA,
;          M1553_PC3_1,      A429_V2_1,
;          M1553_PC3_2,      A429_V2_2,
;          M1553_PCI_1,      A429_cPCI_3U,
;          M1553_PCI_2,      A429_cPCI_6U_1,
;          M1553_PCMCIA,     A429_cPCI_6U_2,
;          M1553_PMC,        A429_PCI_8,
;          M1553_V6_1,       A429_PCI_16_1,
;          M1553_V6_2,       A429_PCI_16_2,
;          M1553_V6_3,
;          M1553_V6_4
;          M1553_V6_CC_1,
;          M1553_V6_CC_2,
;          M1773_PCI )
device_type=A429_PC16_1

; PCMCIA DSP BIT/Application program filename
; dsp_file=PCMDSP.TXT

; UNIX/NT Device Driver Name
```

```
; filename=sbspci320
filename=sbsa429_0

; Firmware filename
firmware=f030h.dat

; VME Interrupt Vector in hex (0-FFh)
; int_vector=FFh

; PC Base I/O Address in hex (300h-3FFh)
;io_base=390h
io_base=390h

; Interrupt Request Level in hex (0-0Fh)
irq_level=0Fh

; Number of Receive channels
num_receive=2

; Number of Transmit channels
num_transmit=6

; Product Type (ABI,ASF,A429)
; product=A429

; Relative position of PCI card under Windows 95
; relative_position=0

; PCMCIA Socket number (0-7)
; socket=0

; PC Memory Window Size (16k, 64k)
window_size=64k

; PCMCIA Xilinx downloadable filename (.bit extension)
; xilinx_file=00670002.BIT


[DEVICE=2]

; Base Memory Address Segment in hex
base_address=D0000h

; PCMCIA DSP Bootloader program filename (default bootload.txt)
; bootloader=bootload.txt

; Device Type
device_type=A429_PC16_2

; PCMCIA DSP BIT/Application program filename
; dsp_file=PCMDSP.TXT

; UNIX/NT Device Driver Name
; filename=sbspci320
filename=sbsa429_0
```

```
; Firmware filename
firmware=f030h.dat

; VME Interrupt Vector in hex (0-FFh)
; int_vector=FFh

; PC Base I/O Address in hex (300h-3FFh)
;io_base=390h
io_base=390h

; Interrupt Request Level in hex (0-0Fh)
irq_level=0Fh

; Number of Receive channels
num_receive=2

; Number of Transmit channels
num_transmit=6

; Product Type (ABI,ASF,A429)
; product=A429

; Relative position of PCI card under Windows 95
; relative_position=0

; PCMCIA Socket number (0-7)
; socket=0

; PC Memory Window Size (16k, 64k)
window_size=64k

; PCMCIA Xilinx downloadable filename (.bit extension)
; xilinx_file=00670002.BIT


; [DEVICE=3]

; Base Memory Address Segment in hex
; base_address=D0000h

; PCMCIA DSP Bootloader program filename (default bootload.txt)
; bootloader=bootload.txt

; Device Type
; device_type=A429_PC8

; PCMCIA DSP BIT/Application program filename
; dsp_file=PCMDSP.TXT

; UNIX/NT Device Driver Name
; filename=sbspci320

; Firmware filename
; firmware=f024k.dat

; VME Interrupt Vector in hex (0-FFh)
```

```
; int_vector=FFh

; PC Base I/O Address in hex (300h-3FFh)
; io_base=380h

; Interrupt Request Level in hex (0-0Fh)
; irq_level=0Ah

; Number of Receive channels
; num_receive=4

; Number of Transmit channels
; num_transmit=4

; Product Type (ABI,ASF)
; product=ABI

; Relative position of PCI card under Windows 95
; relative_position=0

; PCMCIA Socket number (0-7)
; socket=0

; PC Memory Window Size (16k, 64k)
; window_size=16k

; PCMCIA Xilinx downloadable filename (.bit extension)
; xilinx_file=00670002.BIT
```

B. "sbsa429.ini" File

```
\Registry\Machine\System\CurrentControlSet\Services\EventLog
   System
      SBSA429
         EventMessageFile = REG_EXPAND_SZ "%SystemRoot%\System32\IoLogMsg.dll"
         TypesSupported = REG_DWORD 0x00000007

\Registry\Machine\System\CurrentControlSet\Services\SBSA429
   Type = REG_DWORD 0x00000001
   Start = REG_DWORD 0x00000003
   Group = Extended Base
   ErrorControl = REG_DWORD 0x00000001
   SBSA429_0
      Parameters
         Interrupt = REG_DWORD 0x0000000f
         PortAddress = REG_DWORD 0x00000390
         PortCount = REG_DWORD 0x0000000e
```

C. ARINC Label Property File

```
######################################################################
### lines starting with pound sign are comments!!!
# blank lines are ignored.
######################################################################


######################################################################
###########   Channel configuration  ##############################
######################################################################
######################################################################
###########   Device 1            ##############################
######################################################################

######################################################################
###########   Receive channels configuration  ######################
######################################################################


######################################################################
######################### channel 1 ##################################

################### channel parameters ###############################
# T for transmit, R for receive:
Dev1Chn1ChnDirection=R

# SLOW for 12.5KHz, FAST for 100 KHz:
Dev1Chn1Speed=SLOW

# time-out in milliseconds for read operation:
Dev1Chn1ReadTimeOut=1000

################# RCB parameters #####################################
# number of labels this channel will receive/transmit:
Dev1Chn1RCBWordCount = 7

# comma separated list of labels, labels are in decimal format:
Dev1Chn1RCBLabels = 136,131,132,1,78,86,81
################### channel configuration end ########################
######################################################################

#
#
#

######################################################################
######################### channel 2 ##################################

################### channel parameters ###############################
# T for transmit, R for receive:
Dev1Chn2ChnDirection=R

# SLOW for 12.5KHz, FAST for 100 KHz:
Dev1Chn2Speed=SLOW

# time-out in milliseconds for read operation:
```

```
Dev1Chn2ReadTimeOut=1000

################ RCB parameters ####################################
# number of labels this channel will receive/transmit:
Dev1Chn2RCBWordCount = 7

# comma separated list of labels, labels are in decimal format:
Dev1Chn2RCBLabels = 136,131,132,1,78,86,81

################## channel configuration end ######################
####################################################################

#
#
#

####################################################################
####################################################################
####################################################################
####################################################################
###########   Transmit channels configuration  ####################
####################################################################
####################################################################
####################################################################
################## Channel 3 ######################################
####################################################################

####################################################################
################## Channel parameters ##############################

Dev1Chn3ChnDirection=T
Dev1Chn3MinFrameCount=4
Dev1Chn3MajFrameCount=0
Dev1Chn3GapTime=0
Dev1Chn3Speed=SLOW
Dev1Chn3ChainID = 1

#frame period in microseconds:
Dev1Chn3FramePeriod=250000

# number of transmit command blocks:
Dev1Chn3TCBCount=1

################ TCB 1 config parameters ###########################
# schedule time in microseconds:
Dev1Chn3TCB1SchedTime=0
Dev1Chn3TCB1ChainID=1
Dev1Chn3TCB1WordCount=2
Dev1Chn3TCB1RepeatRate=1
Dev1Chn3TCB1StartFrame=1

# comma separated labels to transmit in this transmit control block:
Dev1Chn3TCB1Labels= 184,185
####################################################################
################## End channel configuration  ######################
####################################################################
```

```
#
#
#

######################################################################
######################## channel 4 ###################################

################### Channel parameters ##############################

Dev1Chn4ChnDirection=T
Dev1Chn4MinFrameCount=5
Dev1Chn4MajFrameCount=0
Dev1Chn4GapTime=0
Dev1Chn4Speed=SLOW
Dev1Chn4ChainID = 2

#frame period in microseconds:
Dev1Chn4FramePeriod=200000

# number of transmit command blocks:
Dev1Chn4TCBCount=1

################ TCB 1 config parameters ############################
# schedule time in microseconds:
Dev1Chn4TCB1SchedTime=0
Dev1Chn4TCB1ChainID=2
Dev1Chn4TCB1WordCount=2
Dev1Chn4TCB1RepeatRate=1
Dev1Chn4TCB1StartFrame=1

# comma separated labels to transmit in this transmit control block:
Dev1Chn4TCB1Labels= 28,146
######################################################################
################### End channel configuration ########################
######################################################################

#
#
#

######################################################################
################### Channel 5 ########################################
######################################################################

######################################################################
################### Channel parameters ##############################

Dev1Chn5ChnDirection=T
Dev1Chn5MinFrameCount=20
Dev1Chn5MajFrameCount=0
Dev1Chn5GapTime=0
Dev1Chn5Speed=SLOW
Dev1Chn5ChainID = 3

#frame period in microseconds:
```

```
Dev1Chn5FramePeriod=50000

# number of transmit command blocks:
Dev1Chn5TCBCount=2

############### TCB 1 config parameters ############################
# schedule time in microseconds:
Dev1Chn5TCB1SchedTime=0
Dev1Chn5TCB1ChainID=3
Dev1Chn5TCB1WordCount=2
Dev1Chn5TCB1RepeatRate=9
Dev1Chn5TCB1StartFrame=1

# comma separated labels to transmit in this transmit control block:
Dev1Chn5TCB1Labels= 136, 81

############### TCB 2 config parameters ############################
# schedule time in microseconds:
Dev1Chn5TCB2SchedTime=0
Dev1Chn5TCB2ChainID=3
Dev1Chn5TCB2WordCount=2
Dev1Chn5TCB2RepeatRate=2
Dev1Chn5TCB2StartFrame=1

# comma separated labels to transmit in this transmit control block:
Dev1Chn5TCB2Labels= 131,132

#####################################################################
################## End channel configuration ####################
#####################################################################

#
#
#

#####################################################################
################## Channel 6 ####################################
#####################################################################

#####################################################################
################## Channel parameters ###########################

Dev1Chn6ChnDirection=T
Dev1Chn6MinFrameCount=20
Dev1Chn6MajFrameCount=0
Dev1Chn6GapTime=0
Dev1Chn6Speed=SLOW
Dev1Chn6ChainID = 4

#frame period in microseconds:
Dev1Chn6FramePeriod=50000

# number of transmit command blocks:
Dev1Chn6TCBCount=2

############### TCB 1 config parameters ############################
```

```
# schedule time in microseconds:
Dev1Chn6TCB1SchedTime=0
Dev1Chn6TCB1ChainID=4
Dev1Chn6TCB1WordCount=1
Dev1Chn6TCB1RepeatRate=9
Dev1Chn6TCB1StartFrame=1

# comma separated labels to transmit in this transmit control block:
Dev1Chn6TCB1Labels= 136

############### TCB 2 config parameters ############################
# schedule time in microseconds:
Dev1Chn6TCB2SchedTime=0
Dev1Chn6TCB2ChainID=4
Dev1Chn6TCB2WordCount=2
Dev1Chn6TCB2RepeatRate=2
Dev1Chn6TCB2StartFrame=1

# comma separated labels to transmit in this transmit control block:
Dev1Chn6TCB2Labels= 131,132
#########################################################################
################### End channel configuration ####################
#########################################################################

#
#
#

#########################################################################
################### Channel 7 ####################################
#########################################################################

#########################################################################
################### Channel parameters ############################

Dev1Chn7ChnDirection=T
Dev1Chn7MinFrameCount=1
Dev1Chn7MajFrameCount=0
Dev1Chn7GapTime=0
Dev1Chn7Speed=SLOW
Dev1Chn7ChainID = 5

#frame period in microseconds:
Dev1Chn7FramePeriod=1000000

# number of transmit command blocks:
Dev1Chn7TCBCount=1

############### TCB 1 config parameters ############################
# schedule time in microseconds:
Dev1Chn7TCB1SchedTime=0
Dev1Chn7TCB1ChainID=5
Dev1Chn7TCB1WordCount=19
Dev1Chn7TCB1RepeatRate=1
Dev1Chn7TCB1StartFrame=1
```

```
# comma separated labels to transmit in this transmit control block:
Dev1Chn7TCB1Labels= 187,62,65,66,67,72,80,73,81,74,88,91,167,94,117,176,104,114,227
########################################################################
################### End channel configuration ###################
########################################################################

#
#
#

########################################################################
################### Channel 8 ###################################
########################################################################

########################################################################
################### Channel parameters ##########################

Dev1Chn8ChnDirection=T
Dev1Chn8MinFrameCount=1
Dev1Chn8MajFrameCount=0
Dev1Chn8GapTime=0
Dev1Chn8Speed=SLOW
Dev1Chn8ChainID = 6

#frame period in microseconds:
Dev1Chn8FramePeriod=1000000

# number of transmit command blocks:
Dev1Chn8TCBCount=1

############### TCB 1 config parameters ############################
# schedule time in microseconds:
Dev1Chn8TCB1SchedTime=0
Dev1Chn8TCB1ChainID=6
Dev1Chn8TCB1WordCount=19
Dev1Chn8TCB1RepeatRate=1
Dev1Chn8TCB1StartFrame=1

# comma separated labels to transmit in this transmit control block:
Dev1Chn8TCB1Labels= 187,62,65,66,67,72,80,73,81,74,88,91,167,94,117,176,104,114,227
########################################################################
################### End channel configuration ###################
########################################################################
```

D. Breakout Box Wiring Table

Source			ARINC Inputs	Destination		
Pins	LRU	LRU Pin (port)	Name	LRU	LRU Pins	Pins
J4 - 11	FMU1	MP-3A (#6)	AFCS-1	DTB	C1-1A (+)	J1-1
J4 - 12	FMU1	MP-3B (#6)	AFCS-1	DTB	C1-1A (-)	J1-2
J6 - 1	FMU	TP-7C (#1)	REC/AFCS	DTB	C1-1A (+)	J1-1
J6 - 2	FMU	TP-7D (#1)	REC/AFCS	DTB	C1-1A (-)	J1-2
J5 - 5	FMU2	MP-3A (#6)	AFCS-1	DTB	C1-2A (+)	J1-3
J5 - 6	FMU2	MP-3B (#6)	AFCS-1	DTB	C1-2A (-)	J1-4
J6 - 5	FMU	MP-2C (#4)	GPS RAIM REQUEST	DTB	C1-2A (+)	J1-3
J6 - 6	FMU	MP-2D (#4)	GPS RAIM REQUEST	DTB	C1-2A (-)	J1-4
J4 - 1	FMU1	TP-8C (#2)	GPS RAIM REQUEST	DTB	C1-1B (+)	J1-18
J4 - 2	FMU1	TP-8D (#2)	GPS RAIM REQUEST	DTB	C1-1B (-)	J1-19
J6 - 13	FMU	TP-8C (#5)	DME TUN/IRS ALIGN	DTB	C1-1B (+)	J1-18
J6 - 14	FMU	TP-8D (#5)	DME TUN/IRS ALIGN	DTB	C1-1B (-)	J1-19
Destination			ARINC Outputs	Source		
Pins	LRU	LRU Pins	Name	LRU	LRU Pin (port)	Pins
J1-5	DTB	C1-3A (+)	FSIM BUS-1	FMU1	TP-9C (#1)	J4-18
J1-6	DTB	C1-3A (-)	FSIM BUS-1	FMU1	TP-9D (#1)	J4-19
J1-5	DTB	C1-3A (+)	AHRS1/INS1/IRS1	FMU	TP-15C (#9)	J6-7
J1-6	DTB	C1-3A (-)	AHRS1/INS1/IRS1	FMU	TP-15D (#9)	J6-8
J1-7	DTB	C1-4A (+)	FSIM BUS-2	FMU2	TP-9C (#1)	J5-9
J1-8	DTB	C1-4A (-)	FSIM BUS-2	FMU2	TP-9D (#1)	J5-10
J1-7	DTB	C1-4A (+)	AHRS2/INS2/IRS2	FMU	MP-1A (#11)	N/A
J1-8	DTB	C1-4A (-)	AHRS2/INS2/IRS2	FMU	MP-1B (#11)	N/A
J1-9	DTB	C1-5A (+)	ADC-1	FMU1	TP-15C (#1)	J4-7
J1-10	DTB	C1-5A (-)	ADC-1	FMU1	TP-15D (#1)	J4-8
J1-9	DTB	C1-5A (+)	AHRS3/INS3/IRS3	FMU	TP-1C (#11)	N/A
J1-10	DTB	C1-5A (-)	AHRS3/INS3/IRS3	FMU	TP-1D (#11)	N/A
J1-11	DTB	C1-6A (+)	ADC-2	FMU2	TP-15C (#10)	J5-1
J1-12	DTB	C1-6A (-)	ADC-2	FMU2	TP-15D (#10)	J5-2
J1-11	DTB	C1-6A (+)	GPS-1	FMU	TP-14C (#8)	J6-3
J1-12	DTB	C1-6A (-)	GPS-1	FMU	TP-14D (#8)	J6-4
J1-13	DTB	C1-7A (+)	GPS-1	FMU1	TP-14C (#8)	J4-9
J1-14	DTB	C1-7A (-)	GPS-1	FMU1	TP-14D (#8)	J4-10
J1-13	DTB	C1-7A (+)	GPS-2	FMU	TP-14C (#8)	N/A
J1-14	DTB	C1-7A (-)	GPS-2	FMU	TP-14D (#8)	N/A
J1-16	DTB	C1-8A (+)	GPS-2	FMU2	TP-14C (#8)	J5-7
J1-17	DTB	C1-8A (-)	GPS-2	FMU2	TP-14D (#8)	J5-8
J1-16	DTB	C1-8A (+)	GPS-3	FMU	TP-14C (#8)	N/A
J1-17	DTB	C1-8A (-)	GPS-3	FMU	TP-14D (#8)	N/A
J1-22	DTB	C1-3B (+)	ADC-1	FMU	BP-1C (#19)	J6-15
J1-23	DTB	C1-3B (-)	ADC-1	FMU	BP-1D (#19)	J6-16
J1-24	DTB	C1-4B (+)	ADC-2	FMU	BP-1A (#16)	N/A
J1-25	DTB	C1-4B (-)	ADC-2	FMU	BP-1B (#16)	N/A
J1-26	DTB	C1-5B (+)	AHRS-1	FMU1	TP-15A (#9)	J4-5
J1-27	DTB	C1-5B (-)	AHRS-1	FMU1	TP-15B (#9)	J4-6
J1-26	DTB	C1-5B (+)	VOR	FMU	TP-14A (#7)	J6-11
J1-27	DTB	C1-5B (-)	VOR	FMU	TP-14B (#7)	J6-12
J1-28	DTB	C1-6B (+)	AHRS-2	FMU2	TP-15A (#9)	J5-3
J1-29	DTB	C1-6B (-)	AHRS-2	FMU2	TP-15B (#9)	J5-4
J1-28	DTB	C1-6B (+)	DME	FMU	MP-3C (#14)	J6-18
J1-29	DTB	C1-6B (-)	DME	FMU	MP-3D (#14)	J6-19

J1-31	DTB	C1-7B (+)	VOR	FMU1	TP-14A (#7)	J4-13
J1-32	DTB	C1-7B (-)	VOR	FMU1	TP-14B (#7)	J4-14
J1-31	DTB	C1-7B (+)	FSIM BUS	FMU	MP-4C (#13)	J6-9
J1-32	DTB	C1-7B (-)	FSIM BUS	FMU	MP-4D (#13)	J6-10
J1-33	DTB	C1-8B (+)	DME	FMU1	TP-13A (#5)	J4-15
J1-34	DTB	C1-8B (-)	DME	FMU1	TP-13B (#5)	J4-16
J1-33	DTB	C1-8B (+)	FUEL COMPUTER	FMU	TP-9C (#1)	N/A
J1-34	DTB	C1-8B (-)	FUEL COMPUTER	FMU	TP-9D (#1)	N/A
Source			**CDU & FMUs**	**Destination**		
Pins	**LRU**	**LRU Pins**	**Name**	**LRU**	**LRU Pins**	**Pins**
J7-3	CDU	J3 - 25	CDU Output ARINC Bus	FMU1	TP-11A (#3)	J4-34
				FMU2	TP-11A (#3)	J5-23
				FMU	TP-11A (#3)	J6-22
J7-4	CDU	J3 - 26	CDU Output ARINC Bus	FMU1	TP-11B (#3)	J4-35
				FMU2	TP-11B (#3)	J5-24
				FMU	TP-11B(#3)	J6-23
J4-36	FMU1	MP-2A (#4)	FMU Output ARINC Bus	CDU	J3-10	J7-5
J4-37	FMU1	MP-2B (#4)	FMU Output ARINC Bus	CDU	J3-11	J7-6
J5-25	FMU2	MP-2A (#4)	FMU Output ARINC Bus	CDU	J3-12	J7-1
J5-26	FMU2	MP-2B (#4)	FMU Output ARINC Bus	CDU	J3-18	J7-2
J6-20	FMU	MP-2A (#2)	FMU Output ARINC Bus	CDU	J3-10	J7-5
J6-21	FMU	MP-2B (#2)	FMU Output ARINC Bus	CDU	J3-11	J7-6
Source			**Cross-talk**	**Destination**		
Pins	**LRU**	**LRU Pins**	**Name**	**LRU**	**LRU Pins**	**Pins**
J4-38	FMU1	TP-7C (#1)	Cross-talk ARINC Output	FMU2	TP-12A (#4)	J5-42
J4-39	FMU1	TP-7D (#1)	Cross-talk ARINC Output	FMU2	TP-12B (#4)	J5-31
J5-11	FMU2	TP-7C (#1)	Cross-talk ARINC Output	FMU1	TP-12A (#4)	J4-40
J5-12	FMU2	TP-7D (#1)	Cross-talk ARINC Output	FMU1	TP-12B (#4)	J4-41
Source			**Discrete Inputs**	**Destination**		
Pins	**LRU**	**LRU Pins**	**Name**	**LRU**	**LRU Pins**	**Pins**
J4-26	FMU1	TP-1A	System Valid FMU1	DTB	ICB	D1 (J2-17)
J6-26	FMU	TP-1A	System Valid FMU	DTB	ICB	D1 (J2-17)
C1	ICB		System Valid FMU1	DTB	PB0	PCI-81
C1	ICB		System Valid FMU	DTB	PB0	PCI-81
J5-27	FMU2	TP-1A	System Valid FMU2	DTB	ICB	D2 (J2-18)
C2	ICB		System Valid FMU2	DTB	PB1	PCI-79
J4-27	FMU1	TP-1B	Wpt Alert, FMU1	DTB	ICB	D3 (J2-19)
J6-27	FMU	TP-9B	Wpt Alert, FMU	DTB	ICB	D3 (J2-19)
C3	ICB		Wpt Alert, FMU1	DTB	PB2	PCI-77
C3	ICB		Wpt Alert, FMU	DTB	PB2	PCI-77
J5-28	FMU2	TP-1B	Wpt Alert, FMU2	DTB	ICB	D4 (J2-20)
C4	ICB		Wpt Alert, FMU2	DTB	PB3	PCI-75
N/A	FMU1	TP-11D	Approach Capture, FMU1	DTB	ICB	D5(J2-21)
J6-28	FMU	TP-11D	Approach Capture, FMU	DTB	ICB	D5(J2-21)
C5	ICB		Approach Capture, FMU1	DTB	PB4	PCI-73
C5	ICB		Approach Capture, FMU	DTB	PB4	PCI-73
N/A	FMU2	TP-11D	Approach Capture, FMU2	DTB	ICB	D6(J2-22)
C6	ICB		Approach Capture, FMU2	DTB	PB5	PCI-71
J4-28	FMU1	MP-15A	LNAV Valid, FMU1	DTB	ICB	D7(J2-23)
J6-29	FMU	MP-15A	LNAV Valid, FMU	DTB	ICB	D7(J2-23)
C7	ICB		LNAV Valid, FMU1	DTB	PB6	PCI-69
C7	ICB		LNAV Valid, FMU	DTB	PB6	PCI-69
J5-29	FMU2	MP-15A	LNAV Valid, FMU2	DTB	ICB	D8(J2-24)

C8	ICB		LNAV Valid, FMU2	DTB	PB7	PCI-67
J6-30	FMU		VNAV Sub-mode	DTB	ICB	D11(J2-27)
C11	ICB		VNAV Sub-mode	DTB	PA0	PCI-97
	FMU		VNAV Sub-mode	DTB	ICB	D12(J2-28)
C12	ICB		VNAV Sub-mode	DTB	PA1	PCI-95
	FMU		VNAV Sub-mode	DTB	ICB	D13(J2-29)
C13	ICB		VNAV Sub-mode	DTB	PA2	PCI-93
J6-31	FMU	MP-8B	VNAV Sub-mode	DTB	ICB	D14(J2-30)
C14	ICB		VNAV Sub-mode	DTB	PA3	PCI-91
Source			**Discrete Outputs**	**Destination**		
Pins	**LRU**	**LRU Pins**	**Name**	**LRU**	**LRU Pins**	**Pins**
PCI-29	DTB	DIO2	OLEO Air Ground	ICB		A1
B1(J2-1)	ICB		OLEO Air Ground	FMU1	TP-4A	J4-29
				FMU2	TP-4A	J5-18
				FMU	TP-4A	J6-32
PCI-31	DTB	DIO3	TOGA	ICB		A2
B2(J2-2)	ICB		TOGA	FMU1	TP-9A	J4-30
				FMU2	TP-9A	J5-19
				FMU	MP-14C	J6-33
PCI-26	DTB	DIO4	TAS Valid FMU1, FMU	ICB		A3
B3(J2-3)	ICB		TAS Valid FMU1, FMU	FMU1	MP-6A	J4-31
				FMU	MP-6A	J6-34
PCI-28	DTB	DIO5	TAS Valid FMU2	ICB		A4
B4(J2-4)	ICB		TAS Valid FMU2	FMU2	MP-6A	J5-20
PCI-32	DTB	DIO7	VNAV Engage	ICB		A5
B5(J2-5)	ICB		VNAV Engage	FMU	MP-5A	J6-35
PCI-65	DTB	PC0	Altitude Valid FMU	ICB		A6
B6(J2-6)	ICB		Altitude Valid FMU	FMU	BP-3A	J6-36
PCI-61	DTB	PC2	Slave Free FMU1, FMU	ICB		A8
B8(J2-8)	ICB			FMU1	MP-14C	J4-32
				FMU	TP-7A	J6-37
PCI-59	DTB	PC3	Slave Free FMU2	ICB		A9
B9(J2-9)	ICB		Slave Free FMU2	FMU2	MP-14C	J5-21
PCI-30	DTB	DIO6	LNAV Engage	ICB		A16
B16(J2-16)	ICB		LNAV Engage	FMU1	TP-4B	J4-33
				FMU2	TP-4B	J5-22
				FMU	TP-4B	J6-38
Source			**Program Pin**	**Destination**		
Pins	**LRU**	**LRU Pins**	**Name**	**LRU**	**LRU Pins**	**Pins**
J6-49	FMU	TP-3A	SDI FMU	DTB	DC Gnd	DC Gnd
J5-32	FMU2	TP-3A	SDI FMU2	DTB	DC Gnd	DC Gnd
J6-50	FMU	TP-8A	Flight Simulator Program	DTB	DC Gnd	DC Gnd
Source			**Program Pin**	**Destination**		
Pins	**LRU**	**LRU Pins**	**Name**	**LRU**	**LRU Pins**	**Pins**
	DTB		28 VDC+	Switch		S0-1
S0-2	Switch		To FMU1	CB1		CB1-1
			FMU2	CB2		CB2-1
			FMU	CB3		CB3-1
			CDU	CB4		CB4-1
	DTB		28 VDC Ground	DTB	DC Gnd	DC Gnd
CB1-2	DTB		FMU1:28 VDC +	FMU1	BP-2	J4-55
DC Gnd	DTB		FMU1:28 VDC Gnd	FMU1	BP-3	J4-56
CB2-2	DTB		FMU2:28 VDC +	FMU2	BP-2	J5-55
DC Gnd	DTB		FMU2:28 VDC Gnd	FMU2	BP-3	J5-56

CB3-2	DTB		FMU:28 VDC +	FMU	BP-10A	J6-55
DC Gnd	DTB		FMU:28 VDC Gnd	FMU	BP-9A	J6-56
CB4-2	DTB		CDU:28 VDC +	CDU	J3-4	J7-55
DC Gnd	DTB		CDU:28 VDC Gnd	CDU	J3-6	J7-56

Index of Connectors

J1	ARINC 1	DB44
J2	DIO (Discrete I/O)	SCB-100
J3	2M50FC	Can be replaced by fixed wires
J4	FMU1 (202)	
J5	FMU2 (202)	
J6	FMU (402)	
J7	CDU	
J8	28 VDC Power Supply	

E. ARINC Output Words

ARINC Bus	Description	Label (Oct)	Speed (ms)
ADC	True Airspeed	210	100
ADC	Pressure Altitude	203	450
ADC	Baro-Corrected Alt	204	450
AHRS	True Headding	314	50
AHRS	Magnetic Heading	320	50
GPS	GPS Sensor Status	273	1000
GPS	GPS Altitude Word	076	1000
GPS	GPS HDOP word 1	101	1000
GPS	GPS HDOP word 2	102	1000
GPS	GPS Track Angle word	103	1000
GPS	Latitude word	110	1000
GPS	Fine Latitude word	120	1000
GPS	Longitude	111	1000
GPS	Fine Longitude	121	1000
GPS	Ground Speed	112	1000
GPS	GPS HIL word	130	1000
GPS	GPS VIL	133	1000
GPS	GPS HFOM word	247	1000
GPS	GPS VFOM word	136	1000
GPS	Vertical Velocity	165	1000
GPS	Date word	260	1000
GPS	UTC word	150	1000
FSIM	Simulator Command	270	250
FSIM	Simulator LRU Choose	271	250

F. Detailed ARINC Word (Label 001) Description

Word Name: 001 Distance to the "TO" Waypoint

APPROX. RESOLUTION: 0.1 NM
MAXIMUM VALUE : 399939 NM
MINIMUM VALUE : 0
UNITS : NM
DATA FORMAT: BCD
FULL SCALE : 3999.9

Field Name	Bit NO.		Description
Label	01	0	Code OCTAL = 001
	02	0	
	03	0	
	04	0	
	05	0	
	06	0	
	07	0	
	08	1	
SDI	09	0	Spares
	10	0	
Data	11	X	Tenths NM
	12	X	
	13	X	
	14	X	
	15	X	Units NM
	16	X	
	17	X	
	18	X	
	19	X	Tens NM
	20	X	
	21	X	
	22	X	
	23	X	Hundreds NM
	24	X	
	25	X	
	26	X	
	27	X	Thousands NM
	28	X	
	29	0	Positive
SM	30	0	Normal Operation
	31	0	
Parity	32	X	Odd

G. GPS WITH RAIM

In order to go into GPS approach mode, the FMS makes Predictive RAIM requests for the final approach fix (FAF). As part of the request, the Destination waypoint ETA is transmitted in label 152. This label is transmitted every 500 msec during 3-second burst, along with the other labels making up the request.

The DTB shall receive from FMS label 152 on the GPS input channel (low speed) and shall acknowledge reception of this label by transmitting the following two labels on the GPS output channel:

- Label 162 (Destination waypoint ETA) shall contain the same ETA as label 152.
- Label 343 (Destination waypoint HIL) shall contain a user-specified horizontal integrity limit. Its value shall be enterable on the navigation status window (PRAIM HIL). The default value shall be 0.1 nm.

These labels shall be transmitted every second until label 152 is no longer received on input channel #2. Label 343 shall be transmitted for all the sequence counter values.

The sequence of transmission is:

LABEL	SEQUENCE COUNTER	COMMENT
162	-	
343	0	ACK
343	1	HIL AT ETA
343	2	HIL AT ETA – 5 min
343	3	HIL AT ETA + 5 min
343	4	HIL AT ETA – 10 min
343	5	HIL AT ETA + 10 min
343	6	HIL AT ETA – 15 min
343	7	HIL AT ETA + 15 min
162	-	
343		
:		

H. Navigation Terminology Definitions

Course – The intended direction or path of flight in the horizontal plane measured in degrees from north. The actual course or path of an aircraft should be distinguished from its heading. (They may coincide, but usually do not, the difference being a function of heading, side-slip and drift.)

Cross Track Distance – The shortest distance between the present position of an aircraft and the desired track.

Cross Wind – The wind component measured in knots at 90 degree to the longitudinal axis of the flight path.

Drift Angle – The angle between the aircraft centerline and ground track; or the angular difference between true heading and ground track angle. Drift angle is "right" when ground track angle is greater than true heading; and "left" when ground track angle is less than true heading.

Heading – The direction, usually expressed in degrees relative to true or magnetic north, in which the longitudinal axis of an aircraft points.

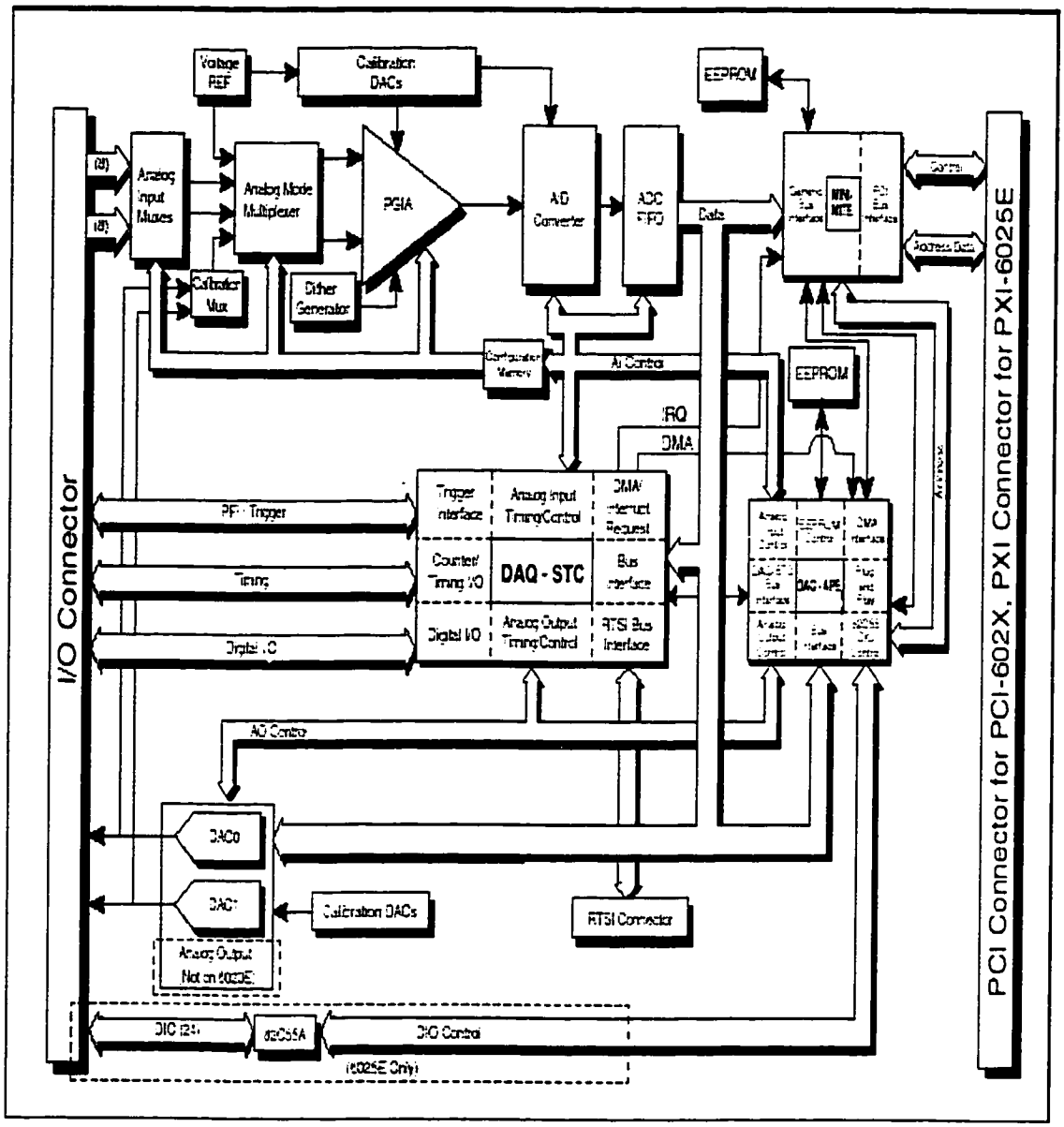
Ground Speed – The speed of an aircraft relative to the surface of the earth, typically expressed in knots.

Ground Track Angle – The clockwise angle from true north to an imaginary line on the earth's surface connecting successive points over which the aircraft has flown (ground track).

Ground Track Angle Error – The angular difference between ground track angle and desired track angle. Track angle error is "left" when the actual track angle is less than the desired track angle, and "right" when the actual track angle is greater than the desired track angle.

True Air Speed – Indicated airspeed adjusted for error from the installation of the sensing equipment, the compressibility of the air, and the density of the air.

I. PCI- 6025E Block Diagram


I/O Connector
PCI Connector for PCI-602X, PXI Connector for PXI-6025E
Voltage REF
Calibration DACs
EEPROM
Analog Input Muxes
Analog Mode Multiplexer
PGIA
AD Converter
ADC FIFO
Data
Control
Address Data
Calibration Mux
Dither Generator
Configuration Memory
AI Control
EEPROM
IRQ
DMA
PFI / Trigger
Timing
Digital I/O
Trigger Interface
Analog Input Timing Control
DMA Interrupt Request
Counter/ Timing I/O
DAQ - STC
Bus Interface
Digital I/O
Analog Output Timing Control
RTSI Bus Interface
AO Control
DAC0
DAC1
Calibration DACs
RTSI Connector
Analog Output Not on 6023E
DIO Control
(6025E Only)