N89-16354

# APPLYING ADA TO BEECH STARSHIP AVIONICS

David W. Funk
Rockwell International
Cedar Rapids, Iowa

## Abstract

Rockwell International's Collins Avionics Group has been active in the Ada* language development since 1978 when we participated in the design evaluation. As the language design solidified, it became evident that Ada offered advantages for avionics systems because of its support for modern software engineering principles and real-time applications. Starting in 1983, Collins developed an Ada programming support environment for two major avionics subsystems in the Beech Starship. The two subsystems include electronic flight instrument displays and the flight management computer system. Both these systems use multiple Intel 80186 microprocessors. The flight management computer provides flight planning, navigation displays, primary flight display of attitude as well as engine instruments and multi-function displays of checklists and other pilot advisory information. Together these systems represent nearly 80,000 lines of Ada source code and to date approximately 30 man years of effort. The Beech Starship avionic systems are in flight test now with expected FAA certification by the end of 1986.

## Background

The Beech Starship is an entirely new turboprop airplane that will combine high performance and excellent fuel economy (see Table 1). The Starship with its composite construction, unconventional design and advanced avionics architecture presented a unique opportunity for Collins Avionics to pioneer the use of Ada in an airborne application.

Applying Ada to major subsystems of the Starship avionics offered several software engineering challenges. Except for the use of proven software design and testing methods all the other elements of software development were new. We started with new system/software requirements, a new development team, a new host computer environment (VAX), a new target computer environment (Intel 80186) and of course a new HOL Ada. It would be misleading to think the selection of Ada was the cause of all this newness because it was not. Given the new system requirements, all the other elements would be new, independent of the language.

* Ada is a registered trademark of the Department of Defense (AJPO).

TABLE 1

Starship Operating Characteristics

| | |
|---|---|
| Max. Takeoff Weight | 12,500 lbs. |
| Max. Altitude | 41,000 feet |
| Cruise Speed | 400 mph |
| Max. Occupants | 10 |
| Single Pilot IFR | |

## Starship APSE

The Ada Programming Support Environment (APSE) that Collins established for the Starship applications include a compiler, assembler/linker/loader, symbolic debugger, configuration manager, text editor and command language interpreter. The APSE is hosted on VAX (VMS) computers targeted to the Intel 80186 (see Figure 1). The components of the APSE are discussed below:

### Compiler

Developed by Irving Compiler Corporation (formerly the Irvine Computer Science Corporation), the ICC compiler front end accepts an Ada source program and performs all lexical, syntactic, and semantic analysis. Under license to ICC, Collins developed the code generator for the Intel 8086 family of microprocessors. While the compiler is not validated by the Ada Joint Program Office, it only accepts valid Ada statements. At the beginning of the Starship projects it was determined that the ICC compiler was more than adequate to support the design constructs used in avionics software. This conclusion was reached by careful comparison with other HOL compilers in use at Collins. The compiler produces an assembly source file at the rate of 800 to 1000 Ada source lines per minute on a VAX 785. When the option is selected the compiler also produces a symbol table file for use by the symbolic debugger.

### Configuration Manager

When the code is developed there must then be a method of keeping track of its evolution. This is partially the task of a configuration management tool. Source Tools by Oregon Software is used to manage a project's files by storing them in a library, tracking changes, and monitoring access to the library that contains the files. The Make function controls the efficient building of a software system by determining which components in the system have changed and then updating, or creating new versions of, only those files that depend on the changed components.

## Symbolic Debugger

The testing phase is especially difficult for embedded computer applications. Because of this, Collins Avionics has developed a symbolic debugger which allows a developer to test a program at the Ada source code level on the target computer rather than at a lower machine code level. The debugger uses a database that is generated in part by the compiler, assembler and linker. This database is separate from the user program. This means that the user program need not be altered (it could even be in ROM) in order to use the debugger for testing.

The symbolic debugger which is also written in Ada is hosted on an IBM personal computer. The personal computer is fitted with cards that connect it via a cable to the target computer. These circuit cards provide additional RAM memory as well as control functions including signals to reset, halt, run and step the target computer, as well as facilities to examine and modify target memory location, an execution history buffer and address matching logic for breakpoints. The user interface to the debugger via the personal computer include commands for file manipulation, execution control, breakpoint, data manipulation as well as show and help commands.

## Editor

The text editor currently being used is the EDT editor developed by DEC. It allows editing to be done in either full screen or line mode.

## Interpreter

This task is currently performed by DEC's Digital Command Language (DCL) and its associated command processor. This provides a user with interactive program development, device and data file manipulation, and interactive and batch program execution and control.

## Assembler/Linker/Loader

For the 80186, a VAX hosted cross assembler/linker package was purchased from Microtec Research. Collins developed programs were added to this package to provide data for the symbolic debugger.

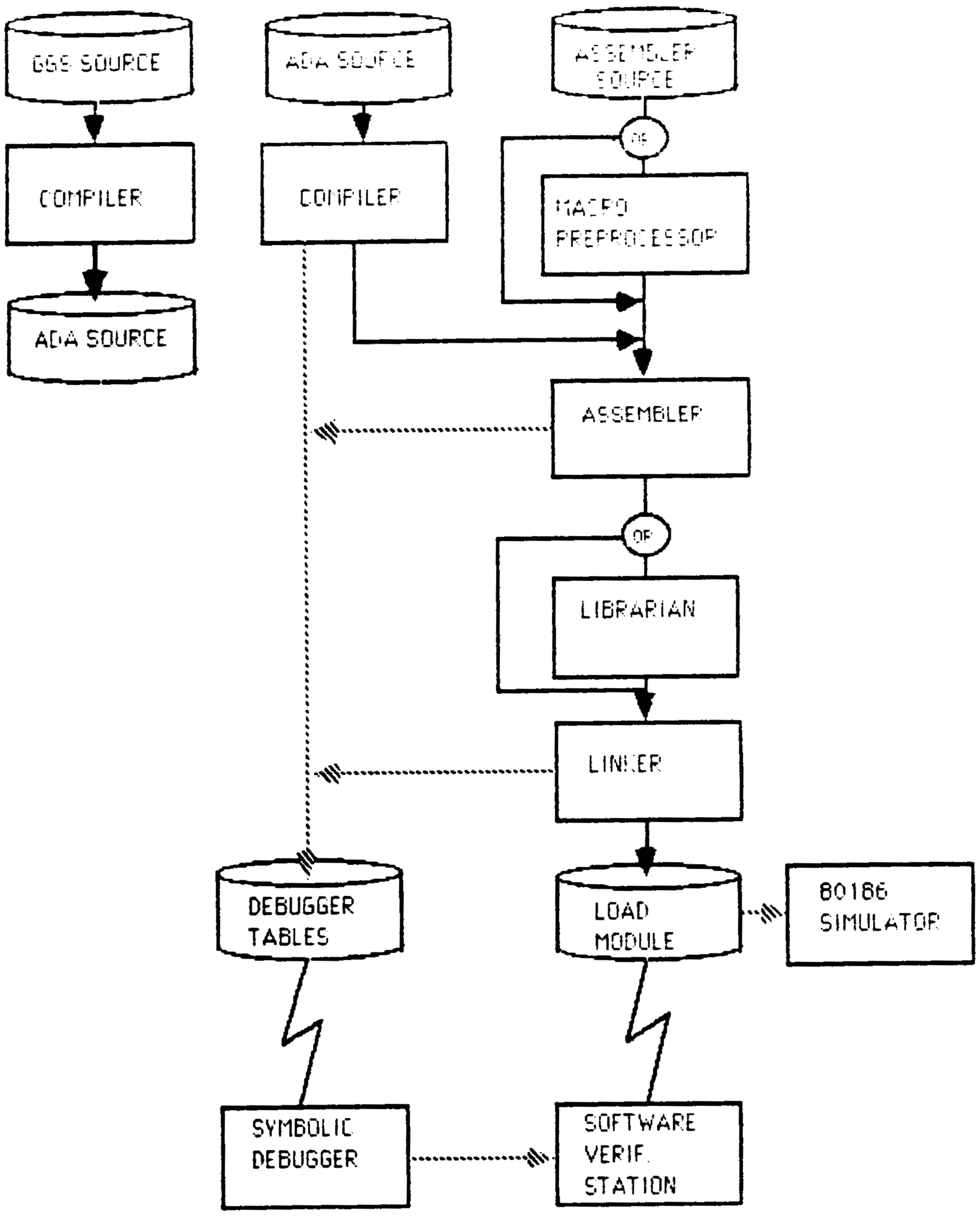
## GGS Compiler

In order to help automate the generation of electronic flight display page formats, a general graphics system (GGS) compiler was added to the Starship APSE. As shown in Figure 1, the compiler accepts GGS source code and translates it into Ada source code. The GGS source is expressed in a language that allows description of a graphics object such as scales, pointers, numbers and letters as well as raster fill areas and stroke written areas. The GGS compiler which is written in Ada also runs on the VAX host computer.

ORIGINAL PAGE IS
OF POOR QUALITY

Ada Program Development Flow

FIGURE 1



## Ada Applications

### RTE

In order to support the embedded applications, Collins developed a Real-Time Executive (RTE) which is written in Ada. While the RTE is compatible with Ada, it uses a slightly different tasking model than the one directly supported by the language. A different model was chosen for two reasons. 1) It is a tasking model that Collins has used in many other embedded systems. 2) The ICC compiler did not completely support all of the Ada tasking features.

The RTE provides an interface from the application programs to the 80186 processor. Included in the processor resources are the CPU execution resource, interrupts and the timers. From the viewpoint of the application program, the following functions are supported: tasking, based on the concept of independent tasks, which share the CPU resources; time-based execution of tasks; event-based synchronization between tasks; external interrupt based execution of tasks and controlled access to resources such that independent tasks do not interfere with each other accessing shared resources.

A task is identified by stack, outer scope procedure, priority, and a four character name. Tasks are prioritized and may be activated by cyclic timer, event based signal or an external hardware interrupt. The RTE can support cyclic execution of tasks up to 1000 hertz. An Ada package may contain the outer procedure and stack for zero, one or more tasks, and procedures in a package may be executed by many different tasks. Table 2 lists the executive service routines that an application uses to interface with the RTE. The size of the RTE target code is approximately 10,000 bytes.

### EFD

The Electronic Flight Displays (EFD) developed for Beech use 6 x 7 inch color CRT and completely integrated display processing. One display unit type is used in four applications in the cockpit. Table 3 indicates the applications and their functions. Each display unit is programmed with two applications to allow better redundancy and reversionary modes. In a two pilot cockpit, six display units, (four PFD/ND units and two EICAS/MFD units) are used . The Ada based applications execute on an pair of 80186 microprocessors. The first 80186 is used for application specific functions (such as PFD or ND) as well as input/output functions. The second processor is used for display control functions that are common to all display applications. Each processor uses a copy of the RTE.

### FMS

The Flight Management System (FMS) provides a very flexible automatic multi-sensor navigation system which greatly reduces the pilot's workload. This system consists of a control display unit, a data base unit with a 3 1/2 inch floppy disk drive, and the flight management computer with a pair of 80186 processors. The Ada based applications implement the functions listed in Table 4. The software is partitioned on the two microprocessors in the following way. The first microprocessor provides control of the pages displayed on the CDU as well as control of the data base unit. The second microprocessor coupled with a floating point co-processor provides all of the navigation and performance computations. Some statistics about the EFD and FMS projects are summarized in Table 5.

TABLE 2

Executive Service Routines

| | | |
|---|---|---|
| Procedures: | | |
| | START TASK | defines a task procedure, stack, priority, and name |
| | CHANGE PRIORITY | raise or lower a task priority |
| | ABORT TASK | stop task execution |
| | SET TIMER INTERRUPT RATE | defines cyclic task execution rate in hertz. |
| | WAIT FOR TIMER INTERRUPT | stalls task until next cyclic timer interrupt |
| | WAIT FOR (EVENT) | stalls task until a signaled event or external interrupt |
| | SIGNAL (EVENT) | used to synchronize with another task |
| | RESERVE (RESOURCE) | used to dedicate a shared resource to the calling task |
| | RELEASE (RESOURCE) | frees a shared resource (opposite of RESERVE) |
| Functions: | | |
| | FULL SECONDS | returns current value of real time clock in seconds |
| | CENTI SECONDS | returns current value of real time clock in 0.01 seconds |

TABLE 3

EFD Applications and Functions

PFD (Primary Flight Display)

Attitude
Flight Director
Lateral & Glideslope Deviation
Airspeed error
Alerts: marker beacon, decision height, altitude, ILS
Flight guidance modes (lateral & vertical)
Fault & off-normal annunciations
Reversionary "composite" PFD & ND

EICAS (Engine Instrument Crew Advisory System)
Torque
Prop RPM
Prop Sync
$N_1$
Fuel Flow
Oil temp, pressure
80 caution & advisory msg

ND (Navigation Display)

Heading
Selected heading/course/track
Lateral/vertical deviation
Bearings (ADF, VOR, WPT)
Distance/time to WPT
Groundspeed, windspeed
Flight plan with Navaids
Weather radar
Reversionary "composite" PFD & ND format

MFD (Multi-function Display)

Reversionary EICAS
Weather radar
Moving map (hdg up)
Planning map (north up)
Checklist (emergency & routine)
Nav status pages (pos, perf)
Diagnostic & maintenance data

TABLE 4

FMS Functions

(1) Statistical estimation of present position employing Kalman filtering techniques, utilizing all available sensor data.

(2) Automatic station selection, tuning, and management of a position fixing submode hierarchy, with provisions for pilot to intervene where appropriate.

(3) Adaptive leg-to-leg and off-course captures with g-limited steering law.

(4) Worldwide data base of VHF navigation aids, airport reference points, and published waypoints in numerous categories.

(5) Pilot creation of a large number of stored routes separate from the active flight plan, with provisions for off-aircraft creation and editing of the stored routes and ability to make trip planning calculations in flight.

(6) Calculations predictive of fuel remaining at destination

(7) Vertical Navigation function with deviation and steering outputs relative to fixed paths in space as may be defined in several ways.

TABLE 5

Ada Project Statistics

| PROJECT | SOFTWARE ENGINEERS | LINES OF SOURCE CODE | TARGET CODE SIZE (BYTES) | HOST DISK (MEGABYTES) |
|---|---|---|---|---|
| EFD | 8 | 31,000 | 318,000 | 164 |
| FMS | 12 | 51,000 | 470,000 | 189 |

## Lessons Learned

Two and a half years of using Ada in real-time embedded systems have taught us some lessons about the application of the language. We chose a "walk first approach" in using Ada. Instead of trying to embrace the entire language with all of its new features, we chose to use a subset of the language that was similar to features of other high order languages we have used. This approach appeared to be wise in terms of training and in terms of bounding the number of variables in building a new system.

Training was accomplished by a combination of classroom work, textbook study and running examples on the VAX hosted APSE. The twelve hours of classroom training which was prepared and delivered by an in-house Ada expert, concentrated on the Ada concepts and features to be used by the Starship project. To complement the classroom work, all students were given a copy of the book "Software Engineering with Ada" by Grady Booch. Despite the fact that nearly every element was new in the software development process for these projects, we found software engineering productivity to average 200 delivered Ada source lines per man month. This is approximately equivalent to what we have experienced on other projects using more established programming support environments.

We discovered that the demands on the host computer system were greater than we had expected. This is in terms of both processing time and storage space. While the compiler is quite fast at 1000 lpm, adding the steps for the assembler, linker and debugger table generation reduce the average to 250 lpm. Add to this the fact that the configuration manager enforces recompilation of dependent packages and the result is longer processing times on the VAX host than we originally planned. Table 5 indicates the amount of host disk space required to support each project which is about twice what we have experienced on other HOL based avionics projects. The increase is explained by the additional files for configuration manager revision history, for the debugger database and the symbol tables needed by the compiler for Ada package specs and bodies. As far as code density on the target, we found the average of nine to ten bytes per Ada statement to be the same as other HOLs for the 80186.

We feel that Ada has brought some real benefits to the subsystems. Ada has provided the discipline and checks to allow program builds to work the first time in laboratory equipment. This greatly reduced debug time on the target computer. Ada has helped offer us more portable code between host system VAX's, personal computers and our target computers. And with the aid of symbolic debugging tools, our verification tasks are simplified.