A BUILT-IN-SUPPORT-FUNCTION (BISF)
FOR ADA SOFTWARE

MARC J. PITARYS, 1LT, USAF

Air Force Wright Aeronautical Laboratories (AFWAL/AAAF)
Wright-Patterson AFB, OH

ABSTRACT

The introduction of the Ada programming language has resulted in considerable development of software design methodologies and tools. However, technology for debugging, integrating, verifying and validating, and maintaining Ada avionics software has not kept pace. In addition to the programming language, the architectures of future avionics systems have out paced our software support systems. The Air Force Wright Aeronautical Laboratories' (AFWAL) Avionics Laboratory has developed and demonstrated a Built-In-Support-Function (BISF) for Ada software. This paper will describe the need for the BISF, the implementation of the specific BISF instructions, and methods for using the BISF with Ada software. Experiences using the BISF will also be reported.

The development and maintenance of avionics software is assisted with instrumentation for real-time software monitoring. This instrumentation is limited to current architectures and programming languages such as JOVIAL. The introduction of Ada, in addition to new avionics architectures such as PAVE PILLAR, have placed new demands on software monitoring technology. An Ada Built-In-Support-Function (BISF) has been developed and demonstrated to support the design and maintenance of distributed Ada avionics software.

There are several features of Ada that current real-time software monitors cannot support. Dynamic allocation of memory for variables and other data types is one example. Local variable addresses are unknown prior to execution since they are stack based. For monitoring variables, current software monitoring devices need to know what memory locations to watch before the software executes.

Ada tasking presents another challenge for software monitoring. Several tasks calling the same entry before the corresponding accept statement is reached causes the calls to be queued. Debugging Ada tasks requires knowledge of how long the calls are queued, when the rendezvous occur and in what order. Therefore, the indeterminacy of Ada tasking requires monitoring techniques that can support nonprobabilistic code. [1]

New avionics systems will consist of multiple processors, distributed Ada application software, and a distributed operating system. Each of these characteristics demand additional capability from the software support tools.

Verifying that the application software works correctly will be a complex effort. The software monitoring equipment will need to trace the execution of the avionics program in real-time. In a distributed system, program tracing must take place for multiple programs running on multiple processors. With the software running in a dynamic, fault-tolerant environment the location of variables and code will be unknown prior to execution. Coupled with Very High Speed Integrated Speed (VHSIC) speeds and limited physical signal access, existing software monitors are at a handicap.

To overcome the obstacles of monitoring Ada avionics software in real-time; Mr Duane O. Hague, an engineer working at the Avionics Laboratory's System Evaluation Branch, proposed an Ada Built-In-Support-Function (BISF). Mr Hague called for a special class of software hook instructions that are embedded into the application and/or operating system software. The BISF instructions would transfer software hook information to a specialized BISF integrated circuit resident on the processor module that functions as an output port to the external support equipment. To the processor/system, execution of a BISF instruction would be equivalent to the execution of a "NO-OP" instruction.

Three BISF instructions are needed to provide debugging support for multiprocessor/Ada avionics architectures. These instructions are the External Flag (XFLG), External Flag Register Dump (XFLGR), and the External Trace (XTRC).

The XFLG instruction outputs a unique flag value to the specialized BISF output port, thus making the value available to the external support equipment. This provides the capability for tracing code execution. By time tagging the flag output software performance can be measured and timing and intermittent errors can be located.[2]

The XFLGR instruction outputs a unique flag value and the contents of a register to the BISF output port. This provides the capability to monitor stack based variables. Software test equipment can use the value of the stack pointer to add with the compiler generated relative address of the program variable to determine the absolute address for monitoring.[3]

The XTRC instruction triggers the test equipment to take a limited size snap-shot of sequential processor internal bus activity such as instruction or data reads and/or writes.[4]

The System Evaluation Branch of the Avionics Laboratory (AFWAL/AAAF) initiated an in-house effort for full BISF proof-of-concept. The Fairchild F9450 microprocessor was selected for the demonstration. Besides implementing the MIL-STD 1750A instruction set architecture, the F9450 supports the optional user defined BIF instruction (opcode 4F).

The F9450 implements the BIF as a three word instruction. The instruction is treated as a co-processor escape where the extension field is written to a dedicated XIO address. The implementation of each of the BISF instructions for the F9450 is shown in figures 1, 2 and 3.

Using the BISF with Ada requires the compiler having the capability to insert machine code inline, and the assembler allowing for the BIF instruction or the capability for defining an instruction. In addition, pragma inline and pragma interface must be implemented. Defining the BISF instructions in the Ada package MACHINE CODE can simplify their insertion.

BISF instructions are inserted at points of interest in the Ada software. For example in Ada tasks, procedures, and exception handlers. A sample of Ada software using the BISF is listed in figure 4. Other areas consist of those sections of software that need performance tuning. XFLG instructions can be inserted by the compiler as part of the calling convention used for context changes. Using the BISF instructions requires no change to Ada (MIL-STD-1815A) or MIL-STD-1750A.

The XFLG and XFLGR instructions were demonstrated with Ada running on the F9450 computer in the fall of 1987. Real-time tracing of a multi-task Ada program was accomplished by using a logic analyzer to monitor the BISF port. Program timing for each Ada task was obtained by using two frequency counters. One counter measured the time between the output of the unique flag for each task. The other counter measured the average time after taking multiple samples. Using the XFLGR instruction, local variable and task control block locations were determined by dumping the program stack pointer to the BISF port.

To demonstrate the XTRC instruction and provide for a user friendly software test environment, a BISF Performance Monitor and Controller (PMAC) is being designed. In addition, a distributed Ada test bed will assist in demonstrating the BISF with Ada software executing on multiple 1750A computers.

There is no "free lunch" when using the BISF to support Ada software. Since the BISF is minimally intrusive, it does have a system impact. For a typical application, execution of 10,000 BISF instructions per second would use under one percent of the processor throughput. Furthermore, there is an increase to the program size and a hardware cost since an additional chip and/or output pins would be needed to fully exploit the BISF.[5]

In an Ada environment, BISF hook information is independent of virtual memory mapping, dynamic allocation, and dynamic multiprocessor multi-tasking. The BISF allows support equipment to determine what software is executing, on which processor, and at what time. When the cost of using the BISF is compared to existing software monitoring technology and the complexities of debugging distributed Ada, the BISF concept is seen as the best technical, least expensive answer to the issue of real-time performance monitoring.

REFERENCES

1. Pitarys, Marc J., Debugging Distributed Avionics Operational Flight Software, AGARD/NATO Avionics Panel Spring 1988 Symposium on "Software Engineering and its Application to Avionics", p. 13-1
2. Ibid, p. 13-2
3. Ibid
4. Ibid
5. Hague, Duane O. Jr, ESIP Built-In-Support-Function for V1750A briefing, 23 Jul 86

Figure 1.

Figure 3.

Figure 2.

FIGURE 4

```

with SYSTEM;
use SYSTEM;
package BISF_SUPPORT is

    procedure XFLG0;
    pragma interface(ASSEMBLER, XFLG0);
    pragma inline;

    procedure XFLG1;
    pragma interface(ASSEMBLER, XFLG1);
    pragma inline;

    procedure XFLG2;
    pragma interface(ASSEMBLER, XFLG2);
    pragma inline;

    --
    --
    --
end BISF_SUPPORT;

--This package is included with the Ada
--applications software.

with BISF_SUPPORT;
use BISF_SUPPORT;
with SYSTEM;
use SYSTEM;
with CALENDAR;
use CALENDAR;
procedure AVIONICS_TASKS

    task type TASK0_TYPE is
        entry START;
    end TASK0_TYPE;

    task type TASK1_TYPE is
        entry START;
    end TASK1_TYPE;

    --
    --
    -- additional task type declarations
    --
    --

    TASK0 : TASK0_TYPE;
    TASK1 : TASK1_TYPE;

```

```

--
--
-- additional task declarations
--
--
--
--
-- task body TASK0_TYPE is
--
--
-- NEXT_ITERATION_TIME : CALENDAR.TIME;
-- CYCLE_DURATION      : constant := 0.004;
--
begin
  accept START do
    -- MAIN PROGRAM IS NOW WAITING FOR
    -- RENDEZVOUS COMPLETION
    null;
  end START;
  NEXT_ITERATION_TIME := CLOCK;
  loop
    --BISF      XFLG0;
    --Perform avionics activity
    --
    --
    --
    NEXT_ITERATION_TIME := NEXT_ITERATION_TIME +
      CYCLE_DURATION;
    delay (NEXT_ITERATION_TIME - CLOCK);
  end loop;
end TASK0_TYPE;

task body TASK1_TYPE is
  NEXT_ITERATION_TIME : CALENDAR.TIME;
  CYCLE_DURATION      : constant := 0.008;
  -- Activity performed once every 8 msec.
begin
  accept START do
    -- MAIN PROGRAM IS NOW WAITING FOR
    -- RENDEZVOUS COMPLETION
    null;
  end START;
  NEXT_ITERATION_TIME := CLOCK;
  loop
    --BISF      XFLG1;
    --Perform avionics activity
    --
    --
    --
    NEXT_ITERATION_TIME := NEXT_ITERATION_TIME +
      CYCLE_DURATION;
    delay (NEXT_ITERATION_TIME - CLOCK);
  end loop;
end TASK1_TYPE;

--
--
-- Additional Task bodies
--
--
--
--
begin
  TASK0.START;
  TASK1.START;
  --
  --
  -- Additional task starts
  --
  --
end AVIONICS_TASKS;

```