The Use of Timing Simulation in Air Force Integrated Avionics

Donald A. Bertke, Member IEEE
Ball Systems Engineering Division, Fairborn, OH

Mark E. Minges
Wright Research Development Center, WPAFB, OH

Abstract

Today's tactical aircraft employ multiple processor architectures which host multiple real-time avionics functions. The real-time processing requirements dictate high system performance and efficient resource utilization. The software and hardware interacting to meet these performance requirements are critical to the success of the integrated avionics system. Thus, the resolution of the real-time interaction of multiple functions sharing processing resources deserves special analysis and consideration. Real-time processing interaction in these complex systems is hard to predict and very difficult to isolate. The software timing problems have erratic symptoms which can cause lengthy integration and test efforts. Each system function needs different processing requirements, input data rates, output data rates, queue and buffer sizes, task interaction and communication, and a host of other variables. It is almost impossible to calculate and determine concurrent function interaction.

The use of a timing simulation can help ease the burden of finding and resolving function interaction problems. A timing simulation can examine the systems sensitivity to change. System performance can be assessed for a wide range of mission scenarios. Design, implementation and upgrade tradeoffs can be assessed for the variable parameters by simulating the system, recording system performance and analyzing the effect of change. Technical decisions are easier to make when the implication of the alternatives are fully understood.

The Integrated Communications Navigation Identification Avionics (ICNIA) program developed a timing simulation to explore the timing issues of its complex architecture. ICNIA shares processing resources to accomplish the various Communication, Navigation and Identification (CNI) functions needed for specific mission scenarios. This paper describes the complex ICNIA architecture, the timing simulation, and the benefits of using the simulation for dynamic timing analysis of complex architectures.

Introduction

The Integrated Communications Navigation Identification Avionics (ICNIA) program is an example of an advanced state-of-the-art embedded avionics system. Its key objectives are to demonstrate that multiple CNI functions can be integrated into a modular airborne radio system. The ICNIA concept, with the Advanced Development Model (ADM) as the present demonstration device, uses advanced Radio Frequency (RF) and Digital technologies. The ICNIA shows that significant increases in reliability and CNI availability can be achieved with substantial reduc-
tions in weight and volume in comparison with equivalent current discrete CNI equipment. The ICNIA can maintain mission critical CNI operation through multiple hardware failures or battle damage. The ICNIA is a tri-service program with Air Force, Army and Navy participation.

Prior to the ICNIA, individual avionics systems were developed for each new CNI requirement, resulting in a proliferation of unique discrete systems. The ICNIA consolidates up to sixteen separate avionics systems operating in the 2 MHz to 2 GHz frequency range. The ICNIA is composed of several dynamically reconfigurable modular building blocks. These basic building blocks are assembled to support any aircraft CNI configuration requirements. The dynamic reconfiguration capability greatly enhances CNI availability and eliminates mission abort due to single system equipment failure. The common module approach simplifies maintenance and supply requirements which result in significant life cycle cost savings.

The modular ICNIA architecture is flexible enough to support the tri-service CNI requirements. These CNI functions share the common system resources and operate concurrently during the aircraft mission. The CNI requirements change for different mission phases. The dynamic resource and CNI function allocation results in a wide range of performance needs for each system module. Timing problems are extremely difficult to isolate in a system that is constantly changing.

A system simulation is one method used to resolve timing problems. The simulation becomes a focusing point to establish specific steady states within the dynamic environment. These defined steady state conditions are used to determine specific tests for the real system. The measurements from these characteristic tests help validate the simulation and verify real system operation. This information reveals most of the initial sources of timing problems.

The BALL Architecture Simulation (BARSIM) was developed under the ICNIA Independent Software Verification (ISV) contract. The BARSIM was created to resolve timing and performance questions about the ICNIA architecture. The BARSIM is very useful in verifying system design performance and in the identification of subtle processing interaction problems encountered during the CNI mission scenario evaluations. The BARSIM outputs include a performance record for the system resources and a detailed event file of system operation.

ICNIA System Description

The ICNIA System Description includes the functional requirements, the architecture design, the hardware components and the software structure. Each will be briefly discussed.

ICNIA Functional Requirements

The ICNIA ADM’s will support up to sixteen CNI systems currently implemented as discrete systems. The ICNIA ADM’s use common hardware components to support these CNI functions. The ICNIA ADM’s use a flexible architecture to promote future growth and maximize fault tolerance. The ICNIA ADM’s use software control to promote dynamic reallocation and simplify system enhancement. The CNI functional requirements are shown in Figure 1.

ICNIA Functional Architecture

The ICNIA functional architecture is shown in Figure 2. It is partitioned into three groups: the Radio Frequency Group (RFG), the Signal Processing Group (SPG), and the Data Processing Group (DPG). Each group encompasses an area of similar technology, which simplifies the logistics control of parts and repair. The functional grouping enhances the use of common hardware modules to build the system components.
The requirements for receive, transmit, and control of each CNI function are divided among the three functional groups. Each of these actions is called a thread. The implementation of each thread can be traced from group to group and from component to component within each group. Figure 3 shows an example of how a receive thread is broken into discrete steps of activity. These steps move the RF signals from the antenna to the digitizing components. These signals are then sent over the Bus network through the architecture to the Signal and Data Processors, which transform the signals into useable data. This step by step decomposition of every requirement identifies the system resources needed by each CNI function. The shared resources are identified when the CNI threads are combined for a mission scenario. These shared resources require special consideration for performance, allocation and utilization. The BARSIM uses a similar step by step approach to represent each CNI function in the simulation environment.
Hardware Description

The integration aspect of ICNIA is based upon a set of common hardware modules housed in 3/4 ATR format. The CNI functions use the same modules to accomplish their unique signal processing requirements. The hardware is programmed to support the processing needs of the CNI function to which it is assigned at run time. If a module fails during the mission, a spare module is programmed to support the function with a minimum of signal loss. If a spare module is not available, then a less critical CNI function is turned off so that its resources can be used to support a higher priority CNI function. This dynamic reconfiguration capability insures that the most mission critical CNI functions will always be available.

Radio Frequency Group

The RFG performs the transmission and reception of the CNI signals. This group is divided into L-Band and UHF/VHF frequency ranges. The RF group contains the Antenna Interface Unit (AIU), the Receiver Switch, the Receivers, and the Power Amplifiers (PA). The RF group provides selectable connections between the receivers and the Universal Matched Filters (UMF) in the SPG. This capability provides flexible resource assignment and fault tolerance.

Signal Processing Group

The SPG converts the CNI baseband signals into digital data. Each Line Replaceable Unit (LRU) consists of 4 UMF’s, a Multi-Function Modulator (MFM), a Signal Processor (SP), a Time Frequency Reference Unit (TFR), and a Red to Black Transfer Unit (RBX). The UMF performs baseband phase conversion, demodulation, detection, and signal acquisition. It sends the digitized data to the SP for signal processing. The MFM performs carrier modulation for all transmitted CNI signals. The TFR provides a stable time base to maintain time synchronization. The RBX interfaces the RFG with the SPG.

The SP uses microcode and macrocode algorithms to implement the CNI function tasks. The SP uses a VHSIC parallel pipelined architecture that does not support context switching between software tasks. Once activated, a task will process to completion before relinquishing the resource for the next task.
Data Processing Group

The DPG performs all CNI application processing, ICNIA system management, integrated test and maintenance, and aircraft interfacing over the 1553B bus. The Data Processors (DP) use a first generation VHSIC MIL-STD-1750A architecture. The software is written in JOVIAL and Ada.

Software Architecture

The ICNIA ADMs use extensive software to control the system and coordinate the concurrent CNI function processing. Much of the Data Processing resources are consumed with scheduling, time sequencing, and Built-In-Test (BIT). The CNI function software is split between the DP and SP Computer Program Configuration Items (CPCI).

The DP CPCI contains the System Management (SMS), Integrated Test and Maintenance (ITM) and the CNI application software. The SMS controls all system operation, including what software is running in which processor. When a CNI application is requested by the operator, the SMS determines if the system can support the new function. Fault detection and isolation is handled by ITM. ITM constantly checks to see if all system components are operating correctly. When a failure is determined, ITM notifies SMS. The SMS then reallocates resources as needed to maintain the most critical CNI functions.

The SP CPCI contains the signal processing algorithms. These algorithms are data driven to perform receive decoding and transmission encoding of data. All SP activity is controlled by the DP.

The ICNIA is a complex multi-function multi-processor system. There is significant interaction between the hardware and the software. Dynamic reconfiguration increases the amount of coordination needed to ensure that the system is operating correctly. A detailed discrete simulation was needed to verify proper design and operation. The BARSIM was developed to help engineers to understand the complex interactions occurring within the ICNIA architecture. The next section describes BARSIM and how it is used.

![Figure 4 Software structure.](image)

BALL Architecture Simulation

The Ball Architecture Simulation (BARSIM) shown in Figure 5 has four major parts. The event processor is the heart of the simulation. Every activity that occurs is treated as an event that is synchronized in a time ordered queue. The event processor executes each event as dictated by the environment and architecture data base files. The architecture data base defines the system components and their connectivity to represent the ICNIA architecture. The environment data base defines the CNI signals that ICNIA processes. The timing required to perform this processing is recorded and summarized in a statistics file. The statistics file provides the data needed to assess the performance of the architecture in the scenario environment. If more detail about the scenario is needed, the BARSIM provides a detailed event list file that records all events processed.

Each CNI function and system service is reduced to a series of actions and processes. These actions can be either periodic actions or triggered by events. A series of these actions is referred to as a signal thread.
The ICNIA Simulation

The BARSIM implementation of the ICNIA architecture includes each bus, processor, and preprocessor. This basic architecture was replicated to implement the CNI specific events needed to evaluate the various mission scenarios. The simulation efforts concentrate upon the digital messages and processing.

Each CNI function was reduced to its basic step by step action/events. These actions were analyzed to derive the approximate timing required by each architecture component. This basic function data is referred to as a signal thread. There can be multiple threads for each CNI function. The following is an example thread for a simple plain text narrow band voice channel as seen in Figures 6 and 7.

1. The UMF sends message 1010 (69 words, priority 4) over the Local Signal Processing (LSP) bus to the SP every 4 milliseconds. The data is attached to the BIU of the UMF. The BIU signals the LSP bus arbiter that it has data to send. The arbiter decides when the BIU may send one Packet (15-16 bit words) of data on the bus. The SP BIU receives each packet of data and transfers the data to the SP Input Output Processor (IOP). The IOP checks the incoming data and determines which SP buffer to load. The IOP sets up a Direct Memory Access (DMA) channel to transfer the data into the SP memory. When all 69 words of data have arrived, the IOP triggers an SP interrupt. The interrupt routine finds which SP software task is waiting for the data and adds the task to the process ready queue.

2. The SP software task uses 573 microseconds to process the 1010 message and send the 3601 (36 words, priority 4) over the Spread Spectrum (SS) Bus to the Voice Input Output (VIO) module. This is accomplished by calling an Input Output routine that formats the data, sets up a DMA channel on the IOP, and triggers the IOP to start the transfer. The IOP transfers the data to the SP BIU attached to the SS Bus. The BIU signals the SS bus arbiter that it has data to send. The arbiter decides when the BIU may send one Packet of data on the bus to the VIO. Since the VIO is not on this bus, the data goes to a Bus Coupler. The Bus Coupler receives each packet and transfers it to another BIU. In the ICNIA, it is a BIU on the Terminal Control (TC) bus. This BIU signals the TC bus arbiter that it now has data to send. The arbiter decides when the BIU may send one Packet of data on the bus to the VIO. The VIO BIU receives the Packet of data and transfers it to the VIO processor, which converts the digital data into voice over the intercom.

There is another thread associated with this example. As the data changes, the SP also adjusts the Automatic Gain Control (AGC) of the receiver. The AGC processing is already included in the SP processing time of the first thread. However, there is additional processing needed to send an actual AGC update as follows.
Figure 6 Narrow Bond Voice Abbreviated Architecture.

Figure 7 Narrow Bond Voice Thread Time Line.

UMF GENERATES 5 PACKETS TO SP NB TASK
SP RECEIVES THE 5TH PACKET FROM THE UMF AND SCHEDULES NB TASK
NB TASK BEGINS EXECUTION
NB TASK COMPLETES AND GENERATES 3 PACKETS TO VIO
AGC COMPLETES AND GENERATES 1 PACKET TO RBX
RBX RECEIVES AGC PACKET FROM SP
VIO RECEIVES 3RD PACKET FROM THE SP
3. The SP software task uses 15 microseconds to format and send message 505 (9 words priority 6) over the SS bus to the Red Black Transfer (RBX) module. The task calls the SP Input Output routine that formats the data, sets up a DMA channel on the IOP and triggers the IOP to transfer the data. The IOP transfers the data to the SS bus BIU. The BIU signals the arbiter that it has data to send. The arbiter decides when the BIU may send one Packet of data to the RBX BIU. The RBX BIU receives the Packet and transfers the data to the RF processor.

Each of these steps take time and coordination. The UMF transfer of data to the BIU takes 300 nanoseconds per word. The bus arbiter checks the BIUs' need to send data every 2 microseconds. The transfer of one Packet of data on the bus takes 2 microseconds. The BIU that last transmitted data must wait one packet cycle (2 microseconds) before it can send another. The BIU to IOP transfer takes 300 nanoseconds per word. The IOP transfer to the SP takes from 9 to 48 microseconds, depending upon the total message length in words. The SP interrupt routine takes 22.5 microseconds to set the task ready. When the task reaches the top of the priority queue, the scheduler takes 25.7 microseconds to set up the task to run. The IOP transfer from the SP to the BIU takes from 4 to 34 microseconds, depending upon the total message length in words. From a timing standpoint, there is a lot of things going on for this simple thread.

It is this complexity that drove the need for the BARSIM. Most of these time delays are the same for like components. Each BIU transfer takes 300 nanoseconds. Each bus transfer takes 2 microseconds. The interrupt and scheduling time for each SP and DP is the same. BARSIM provides all of this detail in its architecture database as seen in Figure 8. Once the threads are defined and each component identified that the thread will use, the database is built linking the actions.

Starting at the beginning of the thread, the ICNIA system component called UMF is attached to the Local Signal Processing (LSP) bus. There are a total of 4 UMF BIUs, 1 SP BIU, and 1 Bus Coupler. To create the UMF to SP data message, the BARSIM data base provides a data generator record that is attached to the UMF BIU. This generator can create the data message periodically to satisfy the signal thread requirement. The generator record needs to know the SP BIU identification number and the SP software task identification number. At the BARSIM initialization, this record will be set up to create this periodic data for duration of the simulation run.

The SP BIU record is attached to the SP IOP record. This tells the BIU to transfer all data it receives to the IOP. The IOP record is attached to the SP record. This tells the IOP to transfer all data it receives to the right SP. The SP software tasks are also defined as records. The BIU and IOP transfers preserve the destination data to properly identify the right SP software task record. The software task record identifies how much data it must receive before it becomes activated, how long it runs when started, and what data it generates when it is done.

The SP record keeps track of incoming data, which software task is executing, and which software tasks are ready to run. The BARSIM processes the SP record cyclically. The time used for each occurrence depends upon which operation it performs. The data interrupts are processed first. The time used for each interrupt is a data base field. In the ICNIA simulation the value is adjusted each scenario to account for the amount of data needed to trigger each interrupt event. Next, the currently executing software task is updated. The task record processing time is decremented each update. In the ICNIA, each
update us 40.2 microseconds. This time is also an adjustable data base value.

The SP operation closely mimics the real system operation. Initially, the SP processes a null task at 40.2 microsecond intervals. As data arrives, the software tasks are enabled and placed on the ready queue in priority order. The first record in the queue is set to execution. Data interrupts suspend execution for the duration of the interrupt.

At this point basic timing data can be collected about the thread. When the generator is executed, the data packets are attached to the BIU and their start time is recorded. As each packet is transferred on the bus, time passes and is recorded for each event. Eventually the packets are consumed by hardware or software. The statistics of their delays are recorded for analysis. If the event file is enabled, each step of their journey is recorded for later analysis. When each SP operation occurs, its event and time are recorded. When a software task becomes ready a timer is started. When it begins executing, its queue wait is recorded. As it is executing, the amount of time it spends in the SP is recorded. When it completes, any data it generates is recorded and timed. This information tells us a great deal about the capabilities of the design to support the thread.

Analyzing the packet delay data identifies if the bus network is fast enough. In the ICNIA, there are 4 levels of message prioritization. Each level has maximum, minimum and average transit times recorded as shown in Figure 9. The time constraint for each thread compared to these recorded times is the first indication of success or failure.
PACKET STATISTICS

<table>
<thead>
<tr>
<th>Priority</th>
<th>Max</th>
<th>Min</th>
<th>Average</th>
<th>No. of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7196.80</td>
<td>8.00</td>
<td>776.32</td>
<td>35297</td>
</tr>
<tr>
<td>5</td>
<td>1350.50</td>
<td>32.00</td>
<td>245.10</td>
<td>2690</td>
</tr>
<tr>
<td>6</td>
<td>170.26</td>
<td>22.00</td>
<td>57.57</td>
<td>2877</td>
</tr>
<tr>
<td>7</td>
<td>381.34</td>
<td>18.00</td>
<td>45.93</td>
<td>507</td>
</tr>
<tr>
<td>Total</td>
<td>7196.80</td>
<td>8.00</td>
<td>682.85</td>
<td></td>
</tr>
</tbody>
</table>

ALL TIMES IN MICROSECONDS

Figure 9 Packet Transit Time Summary.

The SP statistics identifies problems in the software design. The time spent in each SP operation is revealed in Figure 10. The time for scheduling, data interrupts, null task, and CNI task processing is recorded. The scheduling (SC %) and interrupt times (IPP %) indicate processor overhead. The null task (NT) identifies free time available in the processor. The SP usage shows how much time is spent in processing the application software.

SP STATISTICS

<table>
<thead>
<tr>
<th>ID</th>
<th>NT %</th>
<th>SP RUN-TIME</th>
<th>APQ</th>
<th>CPP</th>
<th>SC</th>
<th>SC %</th>
<th>IPP %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>46.256</td>
<td>1000000.0</td>
<td>0</td>
<td>0</td>
<td>738</td>
<td>1.845</td>
<td>1.210</td>
</tr>
<tr>
<td>2</td>
<td>42.912</td>
<td>1000000.0</td>
<td>8</td>
<td>5</td>
<td>296</td>
<td>0.740</td>
<td>0.980</td>
</tr>
</tbody>
</table>

SP # 1 usage in microseconds : 512788.1 % 51.279
SP # 2 usage in microseconds : 554875.9 % 55.488

Figure 10 SP Usage Summary.

Each SP software task record has a wealth of information as seen in Figure 11. The statistics file indicates the number of times the task ran (NTR), the maximum number of data packets it received (MPC), the percentage of SP use, the average queue delay after it was ready (AVG Q), and the maximum queue delay after it was ready (MAX Q). The NTR field indicates if the task ran the correct number of times. This is a good field to check to see if there are any scheduling problems. The MPC field indicates if the input buffer area is large enough. This is a good field to check to see if scheduling delays could cause the thread a problem. The SP use field confirms the scenario loading caused by this thread. If this percentage varies greatly with predicted values, then there is a problem. The AVG Q field indicates the normal delay for the task. The MAX Q field indicates the most delay experienced by the task. The MAX Q and MPC fields are complimentary. If both are large, then problems exist.

The DP records provide similar data on the DP operation and DP task execution. The main difference between the DP and SP models is that the DPs permit context switching and the SPs do not.

Once alerted to a problem by the statistics file, the detailed event file is analyzed to resolve the problem. The event file, as seen in Figure 12, is a text file that contains all of the scenario events in time order. The file is searched using a text editor to verify the conditions and sequence of event leading up to the problem. At this point, the system and software designs are analyzed to isolate if the problem really exists, or is a phantom event caused by the fidelity of the simulation. If it is a real problem. The proposed design changes can be added to the BARSIM and the same scenario rerun to verify that the changes fix the observed problem.

The narrow band voice CNI function is a simple example, but it contains some very subtle timing constraints. The UMF sends data every 4 milliseconds. The SP implementation uses two input data buffers. This establishes a requirement for the function to run every 4 milliseconds with a maximum delay tolerance of 7.4 milliseconds. If the function does not run within 7.4 milliseconds, then the next arriving data message will overwrite one of the two data buffers, causing a loss of data.

Losing 4 milliseconds of narrow band voice data is not a mission critical consideration. However, other CNI functions are very sensitive to
the loss of even one buffer of data. Among these are Encrypted Voice and GPS. The loss of just one buffer of data can result in the loss or disruption of the CNI function for several seconds to several minutes. These are mission critical considerations.

Each ICNIA mission scenario was analyzed to verify that CNI execution time constraints were not violated by the implemented processing interaction. The results of these analyses identified numerous CNI functional disruptions caused by processing delays.
Timing Simulation Benefits

The BARSIM was initially used to verify the capacity of the high speed bus network. This analysis found that the architecture had 2 to 3 times more data flowing through the network than predicted. An earlier analysis had not included packet promulgation from one bus to another. This resulted in larger than expected packet delay times. As a result of this analysis, some messages were moved to a higher priority level and others were lowered as dictated by thread constraint times.

Additional analysis found another architecture design flaw. The Signal Processors were initially grouped together on the global SP bus, separate from the UMFs. This forced all UMF data from both SPG's to be focused on the shared SP bus. This resulted in severe data delay which would have exceeded some function execution time constraints. As a result to these findings, the SPs were moved to the Local SP busses.

The analysis of the SP processing interaction statistics identified a significant software design flaw. The software had been coded to satisfy the functional requirements of the CNI waveform, without consideration for the processing constraints of other CNI functions. This resulted in software that was functionally correct and executed fine by itself, but failed to execute properly when set up to run concurrently with other CNI functions in the same SP.

The symptoms of this timing interaction in the SP was very subtle when observed on the test bench. Software that had passed unit test as a stand alone thread now reacted erratically in a simultaneous signal environment.

The BARSIM statistics and event file data quickly identified these and other problems. The simulation data recorded the processing interaction, delays and disruptions so that the offending software could be identified. Analysis of the software verified that the recorded problems were a result of the code implementation. In subsequent analysis, the BARSIM was used to explore optional coding techniques to verify that the interaction faults would be reduced or eliminated. It was much easier to check coding modifications in BARSIM than it was to code and retest.

Some of the timing problems found by using BARSIM were fixed with simple code modifications. Other problems required more extensive redesign to correct the timing problems. The BARSIM saved many testing hours by identifying these problems early in the integration cycle.

Conclusion

Using the BALL Architecture Simulation greatly helped the ICNIA program to better understand integrated avionics architectures. The BARSIM verified the capabilities of the architecture and identified which components were most likely to cause the performance problems. The BARSIM data permitted dynamic timing analysis of the software design and implementation to identify which processing was sensitive to timing delays.

System level functional simulations have long proven their usefulness in the life cycle of system development. Timing simulations such as the BARSIM are equally useful, indeed essential for real time embedded computer systems. Timing analysis must be performed at all stages of a systems life cycle to prevent significant performance reduction caused by simple implementation decisions and mistakes. System upgrades should be assessed for their effect on the systems operation before time and money are committed. No system design or upgrade should occur without performing a detailed timing analysis to verify the benefits and capabilities of the system.