Impact of an IMA software architecture on legacy avionic software

Thomas Brixel

Impact of an IMA software architecture on legacy avionic software

Thomas Brixel

EADS Deutschland GmbH – Defence and Security, Rechliner Strasse, 85077 Manching, Germany

Abstract: The paper discusses the performance and timing issues when migrating legacy avionics software to an Integrated Modular Avionic (IMA).

The software in question is running on a mission computer equipped with several Motorola 68020 processors and two dual redundant databusses. A new hardware was introduced due to obsolescence problems. To reduce risk the legacy software should be migrated to the new hardware with possibly no changes. Therefore a software stack with standardised software interfaces according to IMA concepts was introduced that provides additionally the software interfaces required by the legacy software on top of IMA.

On the original mission computer the software accesses the databusses directly via memory mapped I/O. This is no longer possible with a layered software architecture. With the implemented IMA software stack I/O is transmitted to a dedicated module via VME backplane. Calls to the hardware specific I/O drivers are handled on that module and responses replied back to the application software.

The paper presents the results of timing and performance measurements with both the legacy and new software architectures on the respective target hardware.

The points that need special interest when specifying the hardware, supplier provided software and when implementing the IMA software stack are discussed.

Keywords: avionic software, migration, integrated modular avionic, performance measurement

1. Introduction

The avionics software that is analysed is a mission software running of a proprietary avionic computer in a military fighter aircraft. The avionics system is a federated architecture where the avionic computers are connected via busses. The avionic computers have a high functional integration and complete subsystem functionality may be allocated to a single computer. With this kind of architecture changes and extensions are expensive and time consuming. An enormous effort is spent on integration.

For avionics the rapid development of computer technology is both a blessing and a curse. On one side it provides the computing power necessary to utilize advanced algorithms. On the other side the lifetime of computer components is much shorter than that of an aircraft, especially a military fighter aircraft. The lifetime of aircrafts is measured in decades whereas the components may become obsolete during development.

To improve reusability of software and ease migration to new platforms a software stack according to IMA concepts is introduced that allows the mission software to become hardware independent. This is realised on a new hardware platform that is free of obsolescence and provides the required processing capabilities for future enhancements. To reduce risk the legacy software should be migrated with possible no changes.

2. Hardware Architecture

The mission computer interfaces comprise two dual redundant databusses, direct digital links (discretes) and power supply. The databusses are realised according to STANAG 3838/3910. The discretes specification is STANAG 3909. The new and legacy mission computers have the same shape and connectors.

Figure 1: Mission Computer

Legacy Hardware: The legacy computer is equipped with two different kind of modules on a VME backplane. The processing modules have a Motorola 68020 processor. The memory mapped I/O registers of the passive bus interface module is accessible by each processor.

New Hardware: The new computer is equipped with processing modules and an active bus interface unit on a VME backplane. The bus interface module is operated with a PowerPC 405 processor. The processing modules have a PowerPC 750 processor.

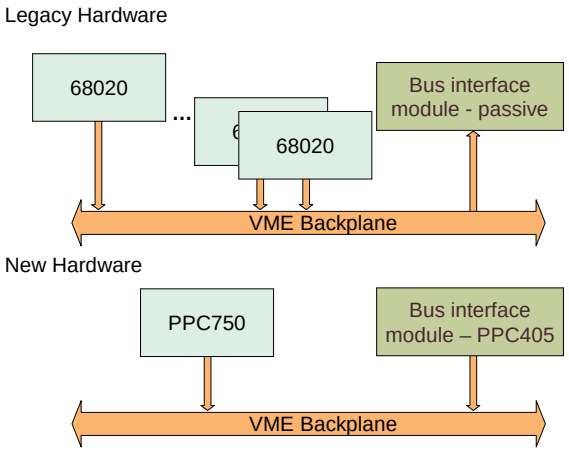

Figure 2: Hardware Architecture

3. Mission Software Architecture

The mission software is designed for optimal resource utilisation, flexibility and hardware independence. It is implemented in Ada.

The software is decomposed into modules that can be allocated to any processor. Each module is controlled by one or more Ada tasks and communicates with other modules via message queues. The complete software is input triggered.

So called Target Specific Ada Packages (TSAP) allow full control of the STANAG 3838/3910 bus.

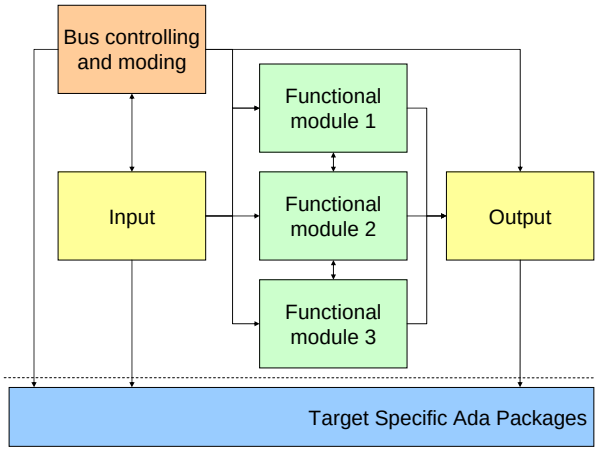

Figure 3: Example of Software Modules

The hardware independence of functional modules is reached by the software design. The modules Input and Output that access the external bus have a low dependency to the underlying hardware. Due to the mode of operation of a STANAG 3838/3910 bus the Bus controlling and moding is highly dependent not even to the hardware but also to the other computers that are connected to the bus.

4. Runtime Environment

4.1. Legacy Ada Runtime and TSAP

Ada Runtime: On the legacy hardware the mission software is compiled and linked with a bare board Ada runtime that provides hardware initialisation and task scheduling.

TSAP: The TSAP operations directly access the bus interfaces via memory mapped I/O. Interrupts can be generated by the bus interface module and are routed via backplane to the respective processing module.

4.2. Message Queues

On the legacy hardware the mission software is evenly distributed to several 68020 processors. The message queues are implemented using global memory that is accessible for each processor.

On the new hardware the complete mission software is running in one program on a single processor. This allows to use an analogue message queue implementation with local memory. The communication between software modules is not affected by the introduction of the IMA architecture.

4.3. IMA Architecture

An open standardised software architecture is introduced with the new hardware based on a commercially available real time operating system (RTOS) that provides time and space segregation.

The IMA software follows Allied Standard Avionics Architecture Council (ASAAC) standards that are under consideration as NATO Standard Avionics Architecture (STANAG 4626, reference [1]).

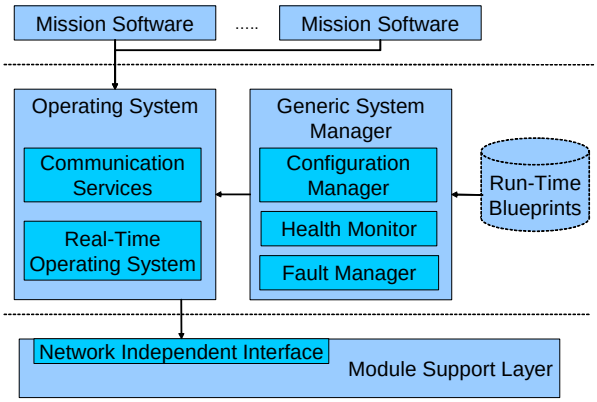

Figure 4: ASAAC Software Architecture

The RTOS is integral part of the IMA software implementation as it provides vital services such as

hardware initialisation, memory management and scheduling.

The communication services provide virtual channels that are independent from the location and number of receivers. A protocol between instances of the Operating System Layer ensures that this is fulfilled. The instances of the Operating System Layer that reside on different processors communicate via transfer connections that are accessed via the Network Independent Interface.

A virtual channel can be configured in a way that every message is acknowledged to the sender when it was successfully written to the receivers message queue. An instance of the IMA software stack runs on each processing module and on the bus interface module.

4.4. IMA Implementation of Target Specific Ada Packages

The Target Specific Ada Packages required by the mission software are implemented utilising virtual channels. Two different implementations exist with respect to the virtual channel usage. In either case the actual bus access is performed on the bus interface module.

Remote Procedure Call Implementation: The first implementation provides the TSAP operations as a remote procedure call. In and out parameters of the TSAP operations are encapsulated into messages that are transmitted via virtual channel. There is a single virtual channel to send commands to the bus interface module. A dedicated virtual channel to receive responses for a command is assigned to each task that calls TSAP operations. The complete error handling is performed by the mission software.

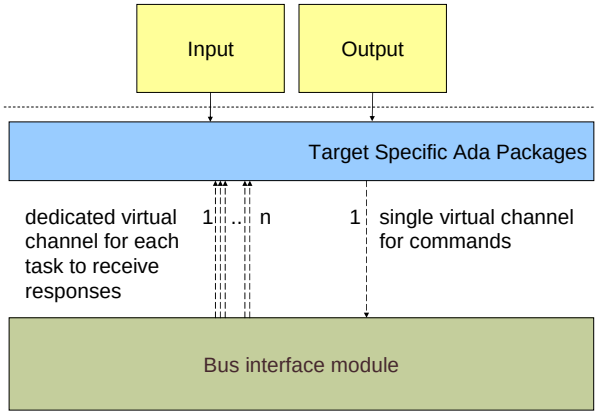

Figure 5: TSAP Implementation as RPC

Mapping of Bus Messages to Virtual Channels: The second implementation directly maps input and output messages to dedicated virtual channels. The mapping of virtual channel identifiers to STANAG 3838/3910 bus messages is part of the IMA runtime configuration. Input messages are sent from the bus interface module to the respective virtual channel without acknowledgement. The virtual channels operate overwriting. Any message that arrives overwrites the previous one as it is also the case in the receive buffer on the bus interface module. There is no interaction with the mission software, so error handling needs to be performed by the IMA software stack.

Output messages are sent by the called TSAP operation to the respective virtual channel. The mission software gets a feed back from the bus interface module by acknowledgement. The error handling of output messages can still be performed by the mission software.

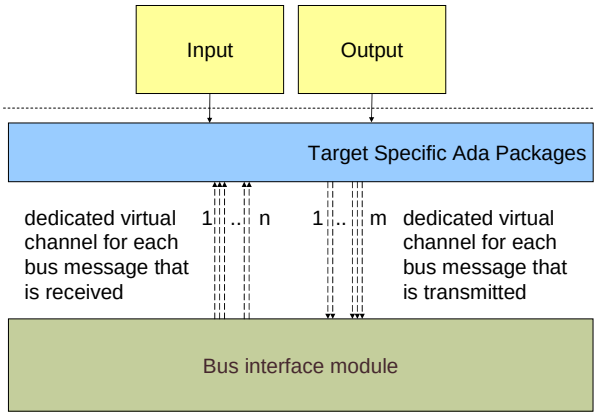

Figure 6: Direct Mapping of Bus Messages to Virtual Channels

5. Measurement Method

5.1. Statistics Calculation

Time measurement and calculation of statistics is implemented in a generic Ada package, which is instantiated for each task to avoid parallel access. It can be used to measure an arbitrary number of individual timings that are distinguished by an element index. The element index can be an integer or enumeration (suitable as array index), the type is specified at package instantiation. The package provides the following operations.

- START_TIME (ELEMENT_INDEX):
- END_TIME (ELEMENT_INDEX):
- CONTROL_TASK.START (START_DELAY, PROFILING_DURATION)

The CONTROL_TASK waits for the time of START_DELAY before time measurement is enabled. This assures that the precision of results is not polluted by start up and initialisation of tasks in

the mission software. When PROFILING_DURATION is elapsed the CONTROL_TASK disables time measurement for all instances running on the same CPU before the results are printed.

Start and end times of calls are obtained from operation CALENDAR.CLOCK that is provided by the Ada runtime. Operation START_TIME stores the time obtained from CALENDAR.CLOCK as start time for the given ELEMENT_INDEX. Operation END_TIME again takes the current time from CALENDAR.CLOCK and updates the statistics.

The calculated statistics values for each individual ELEMENT_INDEX are

n : number of calls

$\overline{dt}$: average (arithmetic mean) duration of calls

$\langle dt \rangle$: mean deviation of call duration to average

$dt_{\max}$: maximum duration of a call

$dt_{\min}$: minimum duration of a call

Average and deviation are calculated incrementally.

$$dt = current\_time - start\_time \quad [1]$$

For the first call of END_TIME:

$$n = 1 \quad [2]$$

$$\overline{dt} = dt \quad [3]$$

$$\langle dt \rangle = 0 \quad [4]$$

$$dt_{\min} = dt \quad [5]$$

$$dt_{\max} = dt \quad [6]$$

For succeeding calls of END_TIME:

$$n = n + 1 \quad [7]$$

$$\overline{dt}_n = \frac{\overline{dt}_{n-1} * (n-1) + dt}{n} \quad [8]$$

$$\langle dt \rangle_n = \frac{\langle dt \rangle_{n-1} * (n-1) + \left| dt - \overline{dt}_n \right|}{n} \quad [9]$$

$$dt_{\min} = \min(dt_{\min}, dt) \quad [10]$$

$$dt_{\max} = \max(dt_{\max}, dt) \quad [11]$$

The incremental calculation achieves the best results when values are near the average from the beginning, which is likely for this measurement.

5.2. Measured Operations

During the migration of the mission software to the new IMA software architecture access to the STANAG 3838/3910 bus has been identified as a possible performance bottleneck. Therefore the measurement focuses on reading and writing data on the bus.

The TSAP provides four operations to read and write.

- GET_MIL
- PUT_MIL
- GET_EFABUS
- PUT_EFABUS

The operations can be used in synchronous and asynchronous mode. In synchronous mode the operation blocks until the data is actually transmitted by the bus. Therefore only asynchronous mode is suitable for timing and performance measurements.

The majority of calls to the operations is performed by modules Input and Output. Calls by module Bus controlling and moding are not significant for this analysis. In module Input two tasks are reading different bus messages. In module Output there is one task that updates the bus messages after it collected the data from the functional modules.

The duration and frequency of asynchronous calls in dependency to the data size were measured. The message size in 64 Byte blocks is used as element index in calls to the profiling package.

The duration is measured by taking the start time, performing the call, taking the end time and updating the statistics.

- Take start time
- Perform asynchronous call
- Take end time
- Update statistics

The frequency is the duration between two calls. It is measured by taking the end time, updating the statistics, taking the start time and performing the call.

- Take end time
- Update statistics
- Take start time
- Perform asynchronous call

In this way for the first call the interval measured for the frequency is the time since initialisation of the time measurement. This introduces a deviation that is negligible for the large number of calls.

The measurement starts 30 seconds after start up of the mission software and is performed over 300 seconds.

6. Measurement Results

At a first view on the measurement results the new hardware with the IMA software architecture looks worse than the legacy hardware. Why this is not the case is discussed in detail in the next section.

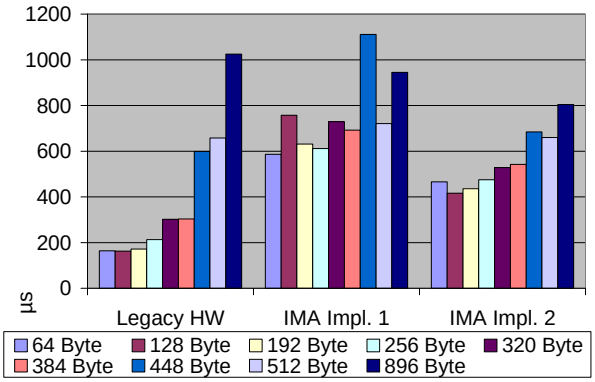

Figure 7: Average Times of Output Task

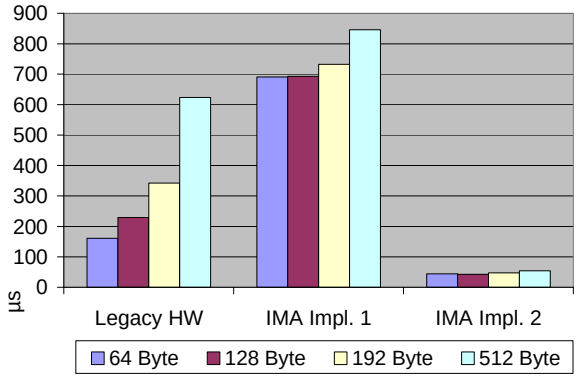

Figure 8: Average Times of Input Task 1

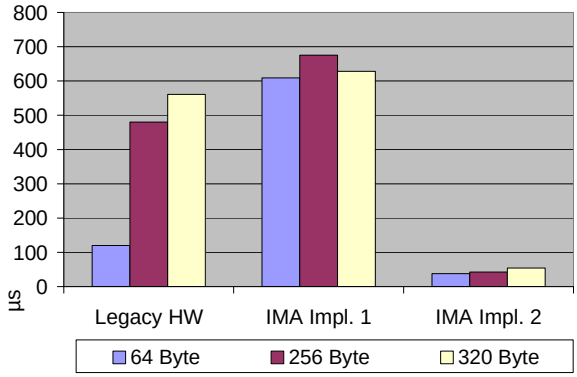

Figure 9: Average Times of Input Task 2

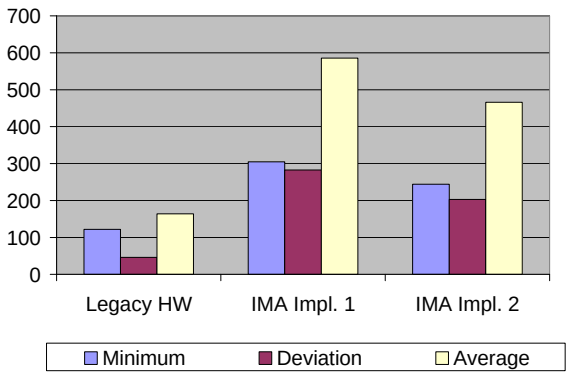

Figure 10: Times for 64 Byte Output Message

7. Analysis of Measurement Results

The frequency of calls for the various message sizes is similar in the three result sets. Approximately the same number of messages and amount of data is transmitted in each case.

7.1. Legacy Hardware

On the legacy hardware the TSAP operations directly access the bus interface. There is no latency due to context switches or signalling to another processor. The processor is occupied during the complete TSAP call transferring data to the bus interface. The deviation of the call duration is small. The existing deviation and the maximum call duration results from tasks with higher priority competing for processing time. The maximum call duration is a multiple of the average.

7.2. IMA Implementation 1

With the IMA implementation 1 the bus messages are transferred to / from the bus interface module. The processor is not fully occupied during the call. The request to read / write a message is transmitted via transfer connection to the bus interface module. The processor is free for other tasks while the calling task is waiting for the response. So the call duration is not pure processing time but includes the communication with the bus interface module. The minimum call duration that is approximately half of the average denotes that requests are queued. This queuing adds to the deviation that is generated from multi tasking already observed on the legacy hardware. As a result the deviation is higher compared to the legacy hardware.

The call duration for input messages with the IMA implementation 2 gives an impression of the processing time that is required on the processing module.

7.3. IMA Implementation 2

Input and output have to be distinguished for IMA implementation 2 as they are different in operation.

Input Messages: To process the TSAP call the latest message received from the bus interface module is read from the local virtual channel buffer. The transmission from the bus interface module to the processing module happens in the background and is not measured. This implementation makes best use of the new hardware and the IMA software architecture. The lowest minimum and average call duration is measured here. Maximum call durations that are a multiple of the average can still be observed due to tasks competing for processing time.

Output Messages: The implementation and measurement results are similar to the first IMA implementation. Even when using a dedicated virtual channel for each output message the requests are queued for transmission through the transfer connection and processed sequentially on the bus interface unit. The small improvements in the call duration may result from the reduced load of the bus interface module due to the different handling of input messages. Another reason may be that the adaptation of TSAP calls to the hardware device driver of the bus interface module is performed on the processing module, which has a higher performance.

8. Conclusion

The mission software runs successfully on the legacy hardware and on the new hardware with both of the IMA implementations. The bus messages are read and written with the required rates. There is sufficient processing time for functional modules. The measurement provides indications for the optimisation of both the mission software and IMA software stack.

The limiting factor on the legacy hardware is processing power and bandwidth. The call duration increases significantly with growing message sizes. Therefore the software is optimised for small message sizes to a certain extent. On the new hardware the size of messages is no longer a problem. Modern hardware has the ability to handle higher amounts of data. Despite the higher processing power of new hardware a layered IMA software architecture introduces latencies. As a consequence the number of messages becomes more relevant, which conflicts with the optimisation of the mission software for small but many messages.

The latencies can be minimised by the design of the IMA software stack. The duration for reading input messages with the second IMA implementation displays that, bearing in mind that the transmission of messages from the bus interface unit to the receiver of the virtual channel is no longer measured. Only a small improvement in access times can be observed for output messages in the second IMA implementation because writing the output messages still implies waiting for the acknowledgement from the bus interface module. To get an acknowledgement enables the mission software to detect errors and handle them adequately with a situation awareness that an generic software like the IMA software stack can not achieve. The error handling implemented in the legacy mission software can be reused and provides a safety that can not be designed into the IMA software stack without modifications to the mission software as well.

The hardware specification needs to address the requirements of the IMA software architecture. The hardware must be capable to handle a high number of interrupts and task switches with low latency.

The IMA implementation must be designed that it requires as few as possible context switches. On the other hand it must ensure safety and reliability, which may mean to accept the latencies this implies.

9. Acknowledgement

Special thank goes to the colleagues at EADS Deutschland GmbH:

Markus Moser and Mark Beerling for the active support of the time measurement

Gert Wiegert for his helpful annotations.

10. References

[1] "STANAG 4626 - Modular And Open Avionics Architectures , Part II – Software, Draft 1", North Atlantic Treaty Organisation (NATO), 2004

11. Glossary

ASAAC:	Allied Standard Avionics Architecture Council
IMA:	Integrated Modular Avionic
I/O:	Input / Output
NATO:	North Atlantic Treaty Organization
RPC:	Remote Procedure Call
RTOS:	Real Time Operating System
STANAG:	Standardization Agreement
TSAP:	Target Specific Ada Package
VME:	Versa Module Europe