

Rapid Application Development Of A Demonstrator Radar Signal Processor Using GEDAE.

B.A. Purdie

B.Sc., M.Eng., CIM. Associate IEE
AMS, Radar Systems Division,
Eastwood House, Glebe Road, Chelmsford,
Essex, CM1 1QW, UK
Tel. +44 (0)1245 702702 Ext. 2456
bruce.purdie@amsjv.com

C.J. Wardell

B.Sc., M.IEE
AMS, Radar Systems Division,
Eastwood House, Glebe Road, Chelmsford,
Essex, CM1 1QW, UK
Tel. +44 (0)1245 702702 Ext. 2445
colin.wardell@amsjv.com

ABSTRACT

Software development for large signal processing systems typically employs manual coding methods in a waterfall design lifecycle, imposing inefficiencies which are becoming more widely recognised by the industry.

This paper describes work carried out by AMS during development of a demonstrator radar signal processor using Blue Horizon Development Software's GEDAE™ toolset. To realise productivity benefits offered by GEDAE appropriate development processes must be used. A method is presented which seeks to integrate algorithm development and implementation phases, as part of an iterative lifecycle.

Technical aspects of GEDAE used in development of the radar signal processing application are also discussed.

1. INTRODUCTION

AMS is very aware of the expense incurred by projects undertaking signal processor application development using 'traditional' algorithm development methods and hand coding as part of a waterfall design lifecycle. The work products of one development phase must be translated/reinterpreted to drive the next phase, with little or no reuse. Rigorous peer review effort is necessary to prevent defect propagation. Early architecture definition and late integration of the application and hardware target result in late identification of any significant design flaws. Redesign/re-coding is necessary through the system life as technology advances and hardware platforms are superseded.

The use of Blue Horizon's GEDAE toolset for Rapid Application Development, which enables autogeneration of embedded applications for specific hardware targets from a graphical application specification, promises many productivity benefits. GEDAE's graphical method of representing the application should enable easily communicated modular designs, facilitating reuse. Those who specify should be able to rapidly embed and test their designs on real hardware, breaking down engineering discipline boundaries. Development phases should significantly overlap and merge.

AMS has a Private Venture project to develop a small Multi Function Phased Array Radar (MFR). A major aspect of this project was that rapid application development methods were to be explored.

The development of the demonstrator signal processing unit (SPU) application is a vehicle for AMS' continued evaluation of GEDAE and methods for its use, while developing a real multi-mode radar signal processing application, typical of those used in AMS products.

It is a stated aim of Blue Horizon that specific methodologies are not imposed on the GEDAE user. This should enable significant changes to be made to formal development methods, and the use of development lifecycles that obtain the maximum benefit from the advantages promised by such tools. AMS is evaluating GEDAE with candidate methods for its use, with a view to controlled insertion of this technology and associated techniques if the promised benefits are realised.

From the project start-up, methods of working were sought whereby algorithm definition, performance analysis and signal processor implementation tasks were aligned to maximise efficient use of the work products created, and to reduce the cost and time scale of the development. During initial development iterations, GEDAE algorithm specifications for major algorithm components in the signal processing chain have been implemented. A harness was devised to conduct performance analysis of the algorithm components. A MATLAB[3] environment simulation produces environment data test vectors, which are used to drive the GEDAE algorithm representation, with MATLAB performance analysis being carried out on the resulting output data. Iterative integration towards development of a complete signal processing application has started, using the initial characterised GEDAE algorithm components.

This paper describes development work undertaken to date, the key features of the methodology developed and the conclusions that have resulted. These include an assessment of the technical capabilities of the GEDAE tool which have been explored to date, with respect to the multi-mode radar signal processing application developed, and an assessment of the effectiveness of the proposed technical methods and the rapid application development process used (including configuration control methods).

An outline of further work required to complete the signal processing application is given.

2. CONTEXT OF THE DEVELOPMENT

2.1 Application Requirement

A diagram showing the demonstrator radar subsystems, is given in figure 1

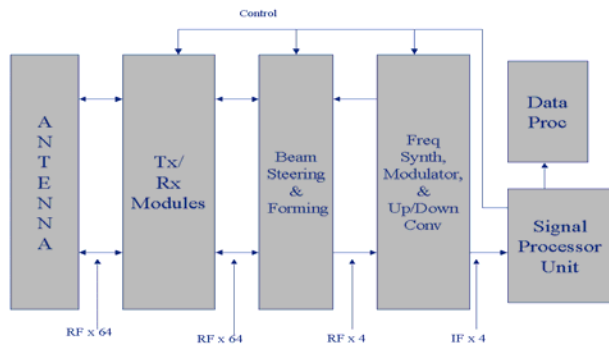


Figure 1. Signal Processor Context.

The essential signal processor capabilities required are:

- The SPU shall accept radar signals via several Intermediate Frequency (IF) channels
- The SPU shall provide radar plot output to a Data Processing Subsystem
- The SPU shall control other radar subsystems, required for radar transmission and reception.

To support initial system integration activities in 2002/2003, the following SPU capabilities are to be developed:

- Radar Control.
 - Control of interface peripherals to generate external signals to drive other subsystems
 - Control of the Analogue to Digital (ADC) peripherals, and input the data to the SPU.
 - Control of the SPU processing application.
- Apply a limited set of SPU algorithms to the input data:
 - Digital Pulse Compression (DPC); a matched filter which gives peak Signal to Noise response when the received signal matches the transmitted pulse [7].
 - Moving Target Detection (MTD); a Fast Fourier Transform filter. Used to separate signal frequency components of returns due to potential targets from those due to slow or stationary clutter. [7].
- Provide an Engineers' human computer interface (HCI), to enable local control of the SPU.
- Perform monitoring and recording of selected internal data flows.

The tasks a MFR is required to implement mean that the signal processor application must be able to support a very large number

of modes of operation and be able to change mode with minimum latency. (See Sections 3.9 and 3.10).

2.2 Project Team Resources

The development team consisted of members from two separate sites, with differing technical backgrounds (new graduates, experienced software and signal processing engineers, systems engineers, and a GEDAE technical expert).

Performance analysis is being conducted using Networked SUN SPARC[8] workstations (Solaris 2.8[8]/GEDAE 3.4.1) and desktop PCs (WindowsNT 4.0[5]/MATLAB 12.1[3])

Implementation prototyping environment is a Mercury[4] PCI development workstation; WindowsNT 4.0[5] Host, Mercury VantageRT[4] cards with 4 AltiVec[6] processors, third party ADC cards and bespoke interface cards (still to be developed)

3. METHODOLOGY

3.1 The Rapid Prototyping Process Implemented

An iterative risk-driven or spiral lifecycle, based on that proposed by the ESPADON programme [1], has been chosen to manage the development. This methodology focuses on rapid prototyping of signal processing applications for embedded systems. System requirements are used to identify the highest risk items, and development tasks are devised to address the identified risks. As solutions to each risk are developed through one or more iteration, the application requirement is refined and system components mature through the stages of specification, functional design, and architectural design, to final implementation.

By using appropriate rapid application development tools and development environment, major gains in productivity should be realised by re-use of work products throughout the project lifecycle. Application of this method to the development tasks completed to date is described in Section 5.

3.2 Algorithm Lifecycle

An algorithm evolves from initial concept to final implementation through a number of stages. The stages are traditionally carried out by different teams using different representations of the algorithm in different environments. A typical mapping is shown in Figure 2.

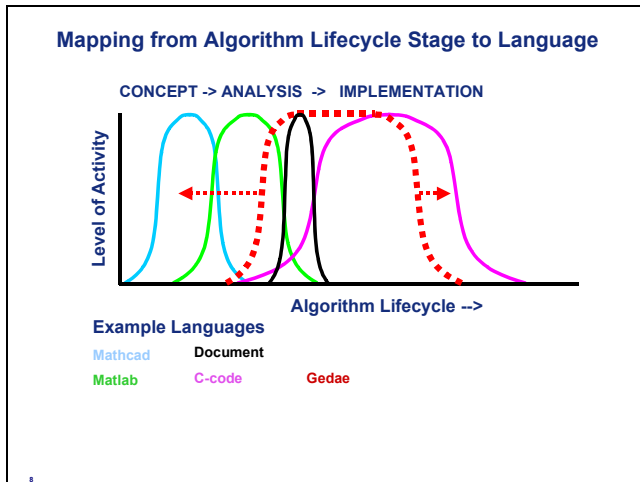


Figure 2. - Algorithm lifecycle stages

The translation from one language and environment to another, as illustrated in Figure 2, results in :-

1. High cost of translation
2. Risk of change in algorithm properties between representations
3. Requirement for separate documentation and testing etc. for each representation
4. Loss of insight and cross-pollination within project team

The advantages of a single representation, maintained and developed over the algorithm lifecycle are: -

1. Reduced translation cost
2. Reduced probability in change in algorithm properties
3. Commonality of documentation and test harness over stages
4. Reduced cycle time between algorithm definition for analysis and implementation for testing
5. More efficient application of wide domain expertise at any stage

The potential disadvantages are: -

1. The representation may be non-optimum for any given stage
2. The overhead in administering a global representation may be high

Having identified the potential for a GEDAE algorithm representation to span most of the algorithm lifecycle, our task was to develop a methodology for using GEDAE to maximise the advantages and minimise the disadvantages.

3.3 Stimulus – Device Under Test - Performance Analysis Framework

The standard structure for developing a component of a system is to represent that component within a system, here called the simulation environment, in which it can be stimulated, can operate, and its response detected and analysed.

The structure of any analysis system is therefore composed of the three elements – stimulus, device under test (DUT) and response analysis.

The simulation environment comprises all three elements and each element may be represented by real, simulated or hybrid representations of that element.

Figure 3 shows the mapping of the three elements onto the current example of a radar processing algorithm set.

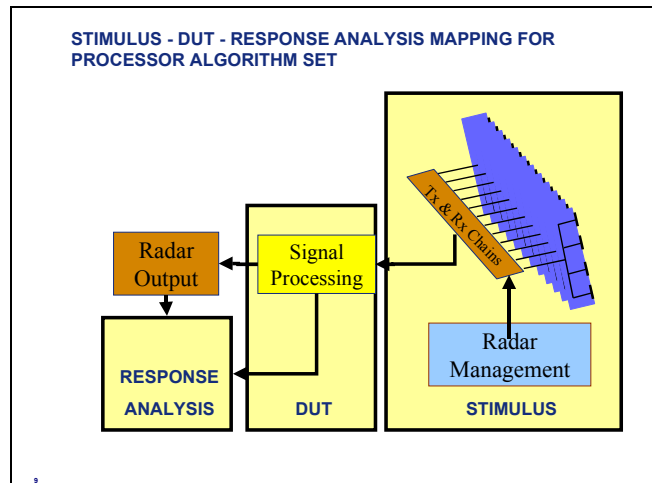


Figure 3 - Stimulus - DUT - Response analysis Mapping for Algorithm Set

In the case of the radar algorithm set, the stimulus is the algorithm environment, composed of the radar environment and radar equipment up to the provision of the data at the input to the algorithm set, together with the radar management function.

The DUT is the algorithm set.

The response analysis is a separate element, which exists externally to the radar system.

With the clear separation of the simulation environment into these three components, it is possible to consider separately the best representation of each element and at each lifecycle stage.

The representation by a mathematical model will include all three elements – stimulus, DUT and response analysis, but the final implementation is likely to be limited to the DUT only.

3.4 Elimination of Translations between Representations

Traditionally the algorithm set is translated from the representation in which it was developed and validated into a non-executable representation, and then again into the implementable representation, suffering the identified costs, long time scales to first implementation and potential for mistakes at each translation.

The stages are shown for each element in Figure 4.

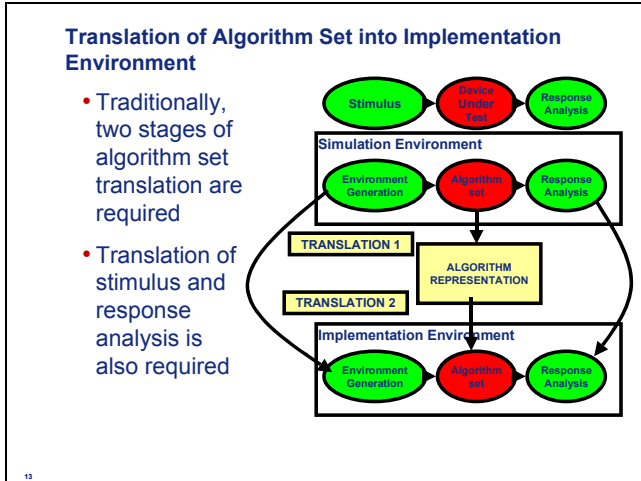


Figure 4 - Translation operations for elements of simulation environment

The first translation elimination is achieved by using a common algorithm representation for analysis modelling and algorithm documentation for coding. This is referred to as an executable algorithm representation.

The second translation elimination is achieved by implementing the executable representation on the target hardware, leading to the situation depicted in Figure 5.

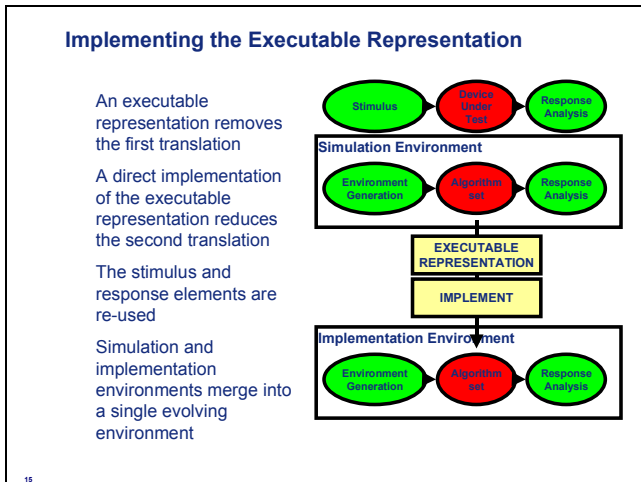


Figure 5 - Implementing the Executable Representation

While eliminating translations of the algorithm set should lead to significant reductions in cost and time scale, further benefits can be gained from consideration of the stimulus and response analysis elements, as shown below.

3.5 Choice of Language for Element Representation

The representation of the three elements need not be the same.

In our example the stimulus was written in MATLAB, the DUT in GEDAE and the analysis in MATLAB.

It may seem strange that, after taking considerable effort to use a single representation of an element through the lifecycle, different languages are considered for the different elements.

The reasons are that: -

1. there is no translation of any element representation
2. the benefit of GEDAE implementation is only relevant to the DUT (in the short term)
3. the greater availability of, company expertise in and existing elements of MATLAB make it a better choice for stimulus representation and response analysis

3.6 The MATLAB – GEDAE – MATLAB Interface

The need to interface MATLAB and GEDAE elements is an added overhead, but this has certain perceived advantages: -

1. The interface between elements is made very clear
2. Code testing within a single representation carries a significant risk of common-mode errors. By representing algorithm and test routine in different languages, the risk of a common error is reduced.

The interface between elements may either be implemented by direct transfer of data between concurrently running elements or by generation and use of stored data sets by typically non-concurrently running elements.

The latter was chosen because: -

1. It is simpler to get going
2. It provides the potential for real-time DUT operation running on non-real-time stimulus data, with non-real-time analysis, or other permutations.

3.7 Analysis Layering

It is recognised that the simulation of a process in which a series of algorithms are chained together, and a data-reduction of many orders of magnitude achieved through the chain, may not be a realistic environment in which to develop individual algorithms. However, separate modelling of subsets of the algorithm set reduces the potential for evaluating group and emergent properties.

The solution adopted was to form a matrix in which the major algorithm modules may each be stimulated either by the preceding algorithm module, or directly by a dedicated stimulus module, and may feed either the next algorithm module of a dedicated response analysis module.

This allows arbitrary contiguous subsets of the algorithm set to be analysed.

The combination of algorithm modularity and stimulus-DUT-response analysis modularity results in the powerful matrix structure shown in Figure 6, where the algorithm set is divided into five representative modules and each module has a corresponding stimulus and response analysis module.

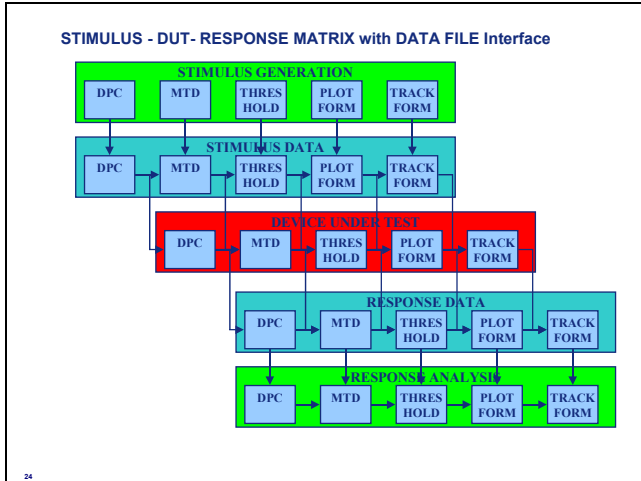


Figure 6 - Stimulus - DUT - Response Analysis Matrix

3.8 Parallel between Development and Implementation stages

A major benefit of the integration of analysis and implementation stages may now be developed. By mapping the I/O structure of the analysis matrix to the implementation source and test points, the stimulus and analysis elements developed in the analysis stage may be re-used throughout the implementation stages. As well as reducing the analysis code generation, the associated test and analysis definition work is reduced, and the process of comparing test results with expected results from the analysis stage is streamlined.

This is shown in Figure 7 as a development of the element translation diagrams above.

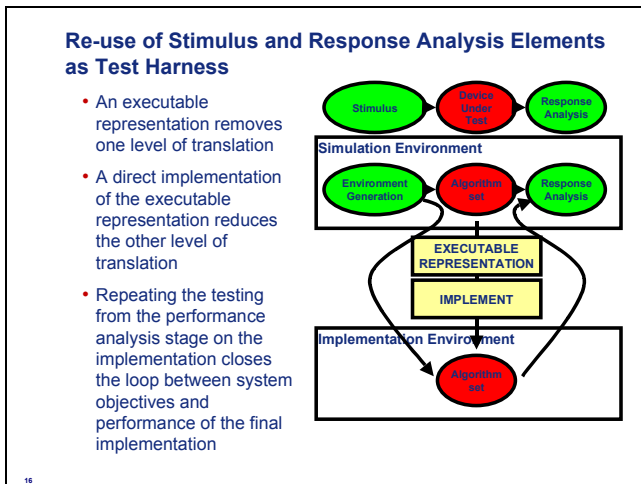


Figure 7 - Re-use of Stimulus and Response Analysis Elements as Test Harness

Note that the testing of the implementation may now be made against the system objectives rather than the analysis results.

3.9 Task Scheduling - Concepts

The operation of an MFR radar requires a function which determines the allocation of radar resource to radar objective. This may take the form of a task scheduler, which determines the

direction of radar search and other parameters as a function of time. The complexity of the task scheduling may range from the implementation of a simple repetitive search pattern of fixed waveform to the dynamic scheduling of a multitude of different tasks selected to optimise the radar's effectiveness in a changing environment.

A simulation used for development also requires control in order to allow the developer to exercise different elements of the DUT. An efficient development simulation control system allows

1. Maximum variation in the test routine with minimum overhead in setting up
2. Efficient configuration control of the tests conditions and results

By developing the task scheduling at an early stage, the combined objectives of MFR task scheduling and efficient simulation control can be met.

The following method of task scheduling has resulted in a test configuration control method which allows

1. individual DUT functions to be exercised very simply
2. complex DUT functions to be exercised by simple extension to the task schedule
3. Simple test configuration, repeatable at all stages of development and implementation
4. Almost limitless extension to the test complexity, including exercise of task scheduling as a DUT function

An additional benefit is that the experience gained in task scheduling for algorithm development is directly relevant to the development of the MFR task scheduling function.

3.10 Task Scheduling - Method

The task schedule is composed of a sequence of tasks, each of which results in the capture by the radar of a block of data representing the sampled received signal from a coherent burst of transmitted pulses. In this case the dimensions of the data block are (PRI, RANGE, BEAM).

The information in the task schedule is used: -

1. In the stimulus element to generate the sequence of data blocks
2. In the DUT element to allow interpretation of the data block and to control the algorithm set operation on the data block
3. In the Response analysis element to evaluate the operation of the DUT

The data flow is shown in Figure 8

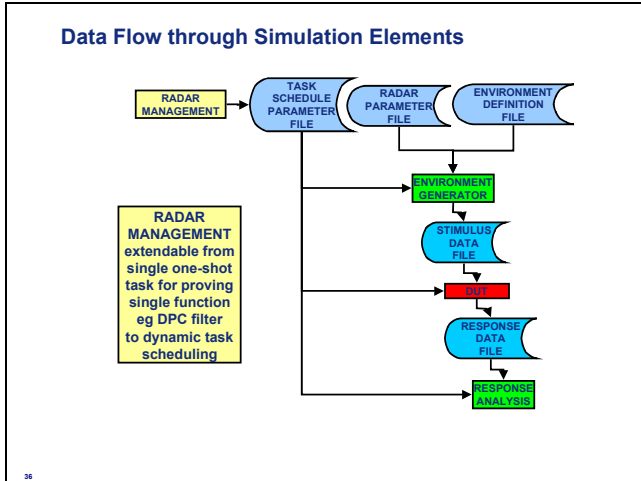


Figure 8 - Data Flow through Simulation Elements

The control of the DUT is achieved by implementing a dataflow in which each task is represented by two tokens, one being the task data block and the other containing the task definition data (which includes the data block dimensions).

The basic flowgraph to implement this is shown in Figure 9.

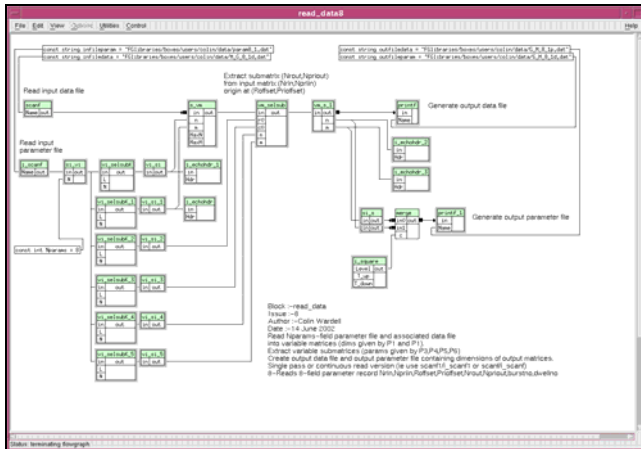


Figure 9 - Dynamic data flow using Variable Matrices

The data block size is controlled dynamically using GEDAE variable matrices, where the matrix size is controlled by fields in the parameter block. The control of data block size by this method, rather than by mode control, results in a great simplification of the flowgraph.

If this representation is to survive into the final implementation, the variability in the data block dimensions from task to task must be efficiently handled. Specifically, the variable matrix approach must be shown to have an acceptably small overhead in both processing load and memory requirement. This is addressed in Section 4.1 below.

4. GEDAE TECHNICAL CAPABILITIES EXPLORED (TO DATE).

Initial development iterations have been performed (See Section 5), addressing initial application requirements and perceived risks, producing some characterised algorithms, and prototypes of components required to implement multi-mode operation, control of the SPU application, the engineers HCI, and recording/monitoring. The technical capabilities of GEDAE explored, during these activities, are now described.

4.1 GEDAE Variable Vector Primitives

The demonstrator signal processor is required to process bursts of data, where each burst can be of different size to the previous burst, and the number of possible burst sizes is very large. A ‘branch-merge’ data flow configuration, with static data flow in a number of parallel processing paths, cannot easily cater for all possibilities. GEDAE’s variable vector primitives provide a simple way of producing a suitable flowgraph representation of the required functionality. The variable vector primitives of GEDAE version 3.4.1 incur a processing overhead. Benchmarking has allowed this overhead to be quantified. The overhead measured for both a simple mathematical operation (add constant K to a vector) and a complex mathematical operation (real vector FFT) is summarised in Table 1.

Table 1. GEDAE 3.4.1 Variable Vector Processing Overhead

Complexity	Functions Compared (vector: variable vector)	Measured variable vector processing Overhead
Low	v_addK:vv_addK	26%
High	v_rfft:vv_rfft	12%

The risk of using vv primitives in an application will thus be approximately a 26% increase in the required number of embedded processors, where 26% is the measured overhead for a simple mathematical operation where the variable vector overhead will be most apparent. Blue Horizon is endeavouring to make processing performance of variable vector primitives comparable with fixed vector primitives. These benchmarks will be repeated with later issues of GEDAE, to monitor the risk.

As a result of the above it was decided to develop the main algorithmic components of the SPU using variable vector primitives. It is worth noting that the potential benefit of the variable vector implementation was identified during algorithm development, and fed forward as a candidate implementation solution.

4.2 GEDAE DataBase Manipulation

An efficient and reusable method of SPU application control, using the textual task schedule files (as used to drive the stimulus - DUT - analysis framework) and pass the parameters for each burst to the embedded application, was required.

A prototype using a linked list was developed. Custom primitives (GEDAE box types ‘eval’ and ‘trigger’) were produced to read a user identified task schedule text file, and populate/control access to a linked list of burst parameter records. A further function was created to enable the oldest burst parameters to be passed to the embedded application independently of the file read operation.

This SPU control design means that the input of burst parameter records is decoupled from the downstream embedded processing.

Thus in a later development iteration, the burst parameter input file read can be replaced by another source of burst parameters (e.g. an Ethernet read) in a later system implementation, with no change to downstream components.

4.3 GEDAE Finite State Machine Implementation

It had become obvious that a finite state machine was required to provide local control of the application operation via a HCI.

Standard and custom trigger boxes were used to produce a set of hierarchical flowgraphs forming state machine components, designed to be as generic as possible, with system events forming trigger inputs, and the required responses appearing as trigger outputs. Required state machine functionality can be simply implemented by constructing flowgraphs which are almost identical in appearance to a typical graphical representation of the required finite state machine design. As state machine functionality of the resulting flowgraphs can be readily be understood, simple verification of the implementation can be achieved by visual inspection. See Figure 10.

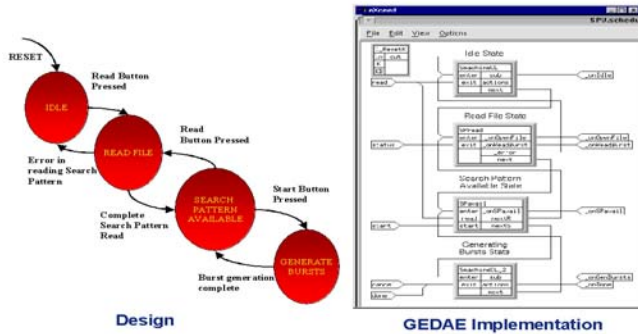


Figure 10. Finite State Machine Design and Implementation.

4.4 GUI Construction

A simple GUI is needed to provide the required Engineer control of the SPU application, to enable user selection of the text file containing the list of burst parameter sets, and provide the required execution control. A prototype GUI was easily constructed using 'out-of-the-box' GEDAE primitives. See Figure 11.

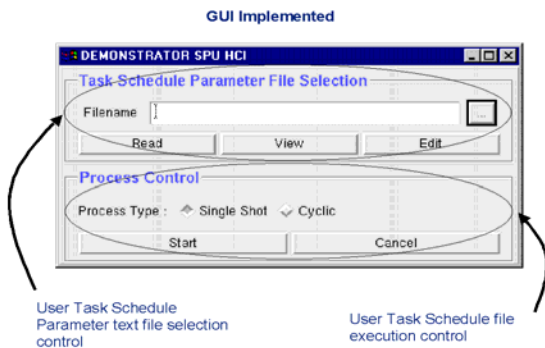


Figure 11. HCI GUI to enable user event input.

4.5 GEDAE Monitoring/Recording

In the current prototype SPU, monitoring and recording functionality is again implemented using 'out-of-the-box' GEDAE primitives. Satisfactory monitoring graphics were easily constructed. See Figure 12

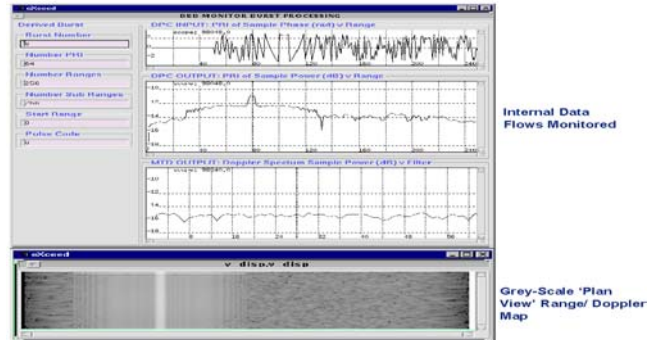


Figure 12. Monitoring Graphics.

Presently recording is provided by simple writing of the data to file for off-line analysis, using the MATLAB analysis components of the Stimulus – Device Under Test - Performance Analysis Framework.

4.6 GEDAE Custom Primitives

The current integrated prototype SPU implementation uses 120 primitives. 12 of these (i.e. 10%) are custom primitives produced to provide functionality not available 'out of the box' from the current GEDAE primitive libraries. The generation of custom primitives was necessary to implement the control functionality in the functional state machine, the data base manipulation, and to enable custom control records to be derived and passed around the application. Similarly custom primitives will be required for application control of peripherals (still to be developed).

Therefore, to use GEDAE to develop an application of any complexity, one or more development team members are required to have in-depth knowledge of the GEDAE methods and language, to develop and test the custom primitives necessary to implement non-standard behaviour, provide optimised performance, and interact with the application's environment. Such primitives will be available for reuse once qualified.

5. APPLICATION DEVELOPMENT TASKS (TO DATE).

The following table describes the development iterations undertaken to date, based on the ESPADON[1] lifecycle.

Table 2. Application Development Tasks Undertaken

	ESPADON Lifecycle Stage	Development Task Addressed, work product generated	Paper Section Describing Requirement/ Risks Addressed
1.	Specification	Prototype GEDAE performance Analysis Framework	3.10
2.	Specification	GEDAE Executable Algorithm Specifications for DPC & MTD	2.1
3.	Specification	MATLAB Environment Generator Response Analysis Components	3.6

4.	Specification	Integration of 1, 2, & 3, to form initial performance analysis framework	3.3/3.2
5.	Specification	SPU Environmental Analysis. Produce GEDAE 'context' model (NB. not executable) & MSWord External Interface description	2.1
6.	Specification	SPU Behavioural Analysis. Produce GEDAE model of capabilities (NB. not executable) & MSWord description of SPU States and Modes	2.1
7.	Functional Design	Multi-mode application variable vector primitive benchmarking	4.1
8.	Functional Design	Prototype SPU Control. Manage Burst Parameters by GEDAE database	4.2
9.	Functional Design	Prototype HCI. GEDAE Finite State Machine to enable user events to control SPU	4.3
10.	Functional Design	Prototype GUI. GEDAE GUI to enable user input of events	4.4
11.	Functional Design	Integrate GEDAE prototypes from 8, 9, & 10	2.1
12.	Functional Design	GEDAE prototype of monitoring displays & integrate with 11	4.5/2.1
13.	Architectural Design	Embedding of GEDAE components and initial benchmarking	2.1

Iterations 1 to 4 (Performance Analysis Stream) have been performed in parallel with iterations 5 to 13 (SPU Implementation Development Stream), with considerable transfer of ideas between the two parallel streams, and the obvious transfer of work products. Work on several of the tasks was conducted at the two separate sites, with integration and final prototype testing being conducted at AMS.

5.1 Development stages remaining

The tasks to be implemented to further iterate the initial prototypes we have at present are as follows:

- Conduct more extensive embedding and benchmarking of all areas of the application, to determine the achievable performance of the current prototypes, and determine the work required to restructure the application to produce a more optimised implementation.
- Develop those areas of the application that will implement the real time peripheral control, and interface with the high bandwidth radar data input from the ADC.
- Development of more sophisticated methods of data monitoring and data recording which are more consistent with operation of the application in real-time, or near real-time.

AMS is continuing to gather metrics to enable more quantitative productivity improvement figures to be produced. Areas of specific focus will be:

- Continued investigation of the processes used to look for areas of improvement
- Establishment of methods to promote reuse of work products - particularly design for reuse

5.2 Summary of Development Productivity Metrics to date

The project has not yet progressed to a stage where anything other than a qualitative assessment of the productivity benefits can be given.

Table 3. Development Effort Summary

Tasks	Effort (man weeks)
SPU Implementation Development Stream: Environmental And Behavioural Analysis (Iterations 5&6)	15.6
Performance Analysis Stream (Iterations 1 to 4)	13.2
SPU Implementation Development Stream (Iterations 7 to 13)	12.8
TOTAL (to date)	41.6

These figures are clouded by the GEDAE and domain knowledge learning curves for several team members.

It would appear that the project has dwelled somewhat on the analysis to develop Environmental and Behavioural model descriptions of the application. I.e. even though we were supposed to be piloting a new process it was easy to slip back into a traditional sequential development where the requirement is fully defined before progressing. Closer monitoring of progress should prevent this in future.

However, once prototyping and benchmarking work began productivity increased – 12.8 man weeks from nil to working (admittedly still rudimentary) signal processing implementation.

A significant factor in enabling this productivity has been the fact that GEDAE allows the application to be modelled graphically, and all team members were using the same toolset. This has:

- Enabled (controlled) rapid development iterations
- Assisted efficient multi-site/discipline team working
- Promoted efficient re-use of items, particularly enabling seamless transfer of the executable algorithm specifications to the prototype implementation.

Previous evaluation studies conducted both by AMS and outside companies have reported a 2 to 3 times productivity improvement through the use of such methods (reported by ESPADON[1]).

5.3 Approach to Configuration Control

AMS' Software Tools Group have devised a method for Parallel Working/Configuration Management for use with the GEDAE toolset. A Company wide workshare environment has been devised using Wind River's SNIFF+[9] and The Free Software Foundation's RCS[2]. SNIFF+ is a complete source code analysis and development environment. However in this instance SNIFF+ is simply used as a GUI to drive RCS and manage the configuration of GEDAE tool versions, and work products developed by projects using GEDAE. All development work is carried out in the GEDAE environment. This system has been successfully rolled out to 3 projects currently using GEDAE in AMS. The main advantages of the method developed are:

- The control of GEDAE tool updates from Blue Horizon is simple.
- Similarly, control of access to reusable items is simple.
- Separate configuration control of individual project work products is achieved easily, and different projects can potentially use different versions of the GEDAE toolset simultaneously.
- Changes to a project's work products are shared through the configuration management tool.
- Users work in controlled project areas (not their home worlds), and disk usage is managed efficiently through the configuration control tool.
- A context sensitive editor, which highlights C code constructs, can be used in SNIFF+ for editing custom primitives.

6. CONCLUSIONS

The development lifecycle proposed by ESPADON has been applied with some success to a demonstrator application development. However, care must be taken to avoid slipping into a traditional sequential development (with which a team may be more comfortable).

The consideration of GEDAE as an example of a rapid prototyping tool has acted as a catalyst to the development of a methodology for integrating algorithm lifecycle stages which promises a set of benefits far wider than those anticipated from the application of the tool in a traditional waterfall lifecycle.

The increased efficiency in communication in a multi-discipline multi-site team engendered by this methodology, and the ease of re-use of developed components, are among the most significant benefits.

However, real application development does still require the team to have one or more members with in-depth GEDAE knowledge.

A method of configuration control, which enables efficient controlled tool updates, and transfer of reusable items across projects, has been implemented and is now successfully being used by three projects in AMS.

The approaches taken to implement control of the radar application, using functional state machine and data base manipulation, together with simple GUI and application monitoring tools, has led to an efficient and re-useable working prototype which is expected to feed forward to a final implementation.

It has been shown that the use of variable vector primitives greatly simplifies the flowgraph representation of this multi-mode radar application. The risk associated with this approach (compared with a 'fixed vector' GEDAE implementation) has been quantified as a possible increase in the number of processors required in the final embedded implementation. The degree of improvement in variable vector primitive performance, achieved in GEDAE releases after version 3.4.1, will determine whether this risk is realised.

7. REFERENCES

- [1] The ESPADON Consortium, Madahar, B. et al. 'Technical Seminar. Embedded Systems Show 17th May 2001, Rapid Prototyping and Virtual Prototyping.' ExCel ESS 2001 16-17 May, 01. London UK.
- [2] Free Software Foundation, RCS
<http://www.gnu.org/software/rcs/rcs.html>
- [3] The Mathworks, Inc. MATLAB[®]
<http://www.mathworks.com/products/prodoverview.shtml>
- [4] Mercury Computer Systems, Inc. VantageRT[®]
<http://www.mc.com/search/productslevel2.cfm?id=8&type=product>
- [5] Microsoft Corporation. WindowsNT[®] 4.0
<http://www.microsoft.com/ntworkstation/default.asp>
- [6] Motorola, Inc. Altivec[™] <http://e-www.motorola.com/webapp/sps/site/overview.jsp?nodeId=03M943030450467M0ymK5Nf2>
- [7] Skolnik M.I., 'Radar Handbook', Second Edition, McGraw-Hill, 1990.
- [8] SUN Microsystems, Inc. SPARC[®], Solaris[™]
<http://www.sun.com/software/solaris/>
- [9] Wind River Systems, Inc. SNIFF+[™]
http://www.windriver.com/products/sniff_plus/

8. ACKNOWLEDGEMENTS

The authors would like to thank:

- Henry Booth, Nicholas Stringer, Michael Germuska, Nicholas Brandon, Clive Jackson of Radar Systems Division, AMS, and Ian Alston, David Sternbach of BAE Systems – ATC, for their technical contributions during the development of the demonstrator signal processing application.
- Terry Jackson and Nigel Waites of the Software Tools Group of Radar Systems Division, AMS, for their contributions in devising the configuration control methods used.
- David Heath, Robert Johnston, David Money, and Jane Norton of Radar Systems Division, AMS, for providing comment on this paper.