

Building DSP Applications via Graphical Design

Richard White
Kane Computing Ltd
7 Theatre Court, London Road
Northwich, Cheshire CW9 5HA UK
+44(0)1606 351006
richard@kanecomputing.com

ABSTRACT

This paper explains what Graphical DSP programming is, and looks at different methods of graphical DSP Programming. The paper will look at important considerations for evaluating the merits of using a graphical design approach to implementing DSP applications together with an examination of the benefits versus cost trade off of this approach to software design.

Keywords

Graphical DSP programming, frame based architecture, graphical DSP compiler, object code generation, C source code generation, open software architecture, hierarchical design, recursive support.

1 INTRODUCTION

Historically, textual-based representations for algorithms (such as Assembly and C) along with corresponding assemblers and compilers are used to produce DSP object code. Development of applications by use of textual representations as the basis for algorithms involves considerable expertise by the person developing the algorithm, and generally involves a significant amount of labour and time to produce the final working code for a system. In short, the process is laborious, time consuming, error-prone, and often requires the learning of relatively cryptic syntax. Another disadvantage is that the documentation and description of the system is generally a somewhat distinct and independent task. Accordingly, there has been a need to provide a method for efficiently generating object code from a graphical, as opposed to textual, construction.

In addition many functions could be used many times but many are lost in one persons sequential program, to be re-written by the next designer. This is overcome to some extent by the availability of conventional C code or in some cases hand optimised object code libraries. These libraries still have to be integrated and compiled, debugged etc. How much nicer to have these functions available as directly produced object code from a graphically constructed, user created block diagram of the design.

This approach is especially productive for those with limited experience and expertise in assembly, C and other textual algorithm development tools, and thereby allows the creation of a working real-time DSP system where it would have been considered impossible before. In addition, the self-documenting nature of the algorithm/design is an important consideration, and allows for a quick learning curve by other engineers, or by the same engineer at a later date. The ability to support a number of different processors allows for relatively painless migration from one processor family to another. With its open software architecture, extensibility and creation of new components is very easily accomplished. With a small amount of initial learning, well-written, easy-to-follow, modular, algorithms can be designed very quickly by the user.

2 WHAT IS 'GRAPHICAL DSP' or 'BLOCK DIAGRAM PROGRAMMING'?

DSP algorithms are created from a graphical design, or block diagram approach by using a methodology of developing DSP algorithms and systems graphically by simply connecting functional components together; this graphical design is then used to produce DSP object code directly within the environment – no C compiler or assembler required. This method of component-based design is quite similar to drawing a 'block diagram' of the system being designed.

2.1 Component based design methodology

Component-based design methodology afford the designer the luxury of not having to code each individual algorithm from scratch. Instead the designer can break the entire design down into a grouping of separate "black boxes", each of which are responsible for implementing a specific task. This black box approach lends itself to making the "big picture" easier to see, and removes the designer from ever having to know how the specific function works, so long as he/she understands what utility the algorithm provides. In this fashion an entire real-time DSP application can be constructed from individual block components of pre-defined purpose. Building DSP applications in this way will allow individuals to more quickly and easily construct valid DSP designs.

2.2 Graphical Compiler Analogy

A graphical design tool can be thought of as a “Graphical DSP Compiler” – as opposed to a C compiler for example – and allows you to create a DSP algorithm from a graphical design, or block diagram approach. Using component-based functions, algorithms are efficiently crafted and directly compiled into DSP Object Code. This technology provides dramatic reductions in development time while yielding increased maintainability of DSP projects.

Just as C compilers turn a textual language (ie. C source code) into DSP object code, RIDE/VAB turns a graphical language directly into DSP object code. Similar to a C compiler or linker, a graphical design tool allows for the use of DSP libraries to resolve block component function calls. This allows the designer to make choices as to using third-party link libraries to optimise performance. With this design methodology users will write much or all of their algorithm, or software, graphically as opposed to textually (C/assembly). The savings in design/development time as well as the advantages of maintainability and the self-documenting nature of graphical design are considerable.

Similar to C programs, algorithms may be developed for a number of targets, including industry standard DSP boards that may already be in the marketplace or custom target hardware developed by the end-user. In fact, just as DSP chip emulators are used in conjunction with textual-based code development, they also may be used along with code developed graphically.

3 DIFFERENT METHODS OF GRAPHICAL DSP PROGRAMMING

There are several different methods that can be used to graphically create design algorithms for use on a DSP. These include simulation and systems modelling, limited real-time development on a PC, simulation with subsequent C source code generation & final cross-compilation to a DSP, and direct DSP object code generation.

3.1 Simulation/Modelling on PC

One method where a block diagram approach might be used in the development of a DSP application lies in the simulation and modelling of a DSP algorithm on a PC. In this case the block diagram design is implemented on the host PC and allows the designer to develop his DSP application without generating any DSP executable program. The process in one of algorithm simulation as opposed to targeting a DSP for real-time program execution. This method is often useful for trying “what-if” scenarios and prototyping a design algorithm. The main benefit from this approach is the reduction of technical risk as the concept is proven in simulation prior to being applied to actual real-time DSP hardware using conventional development methods.

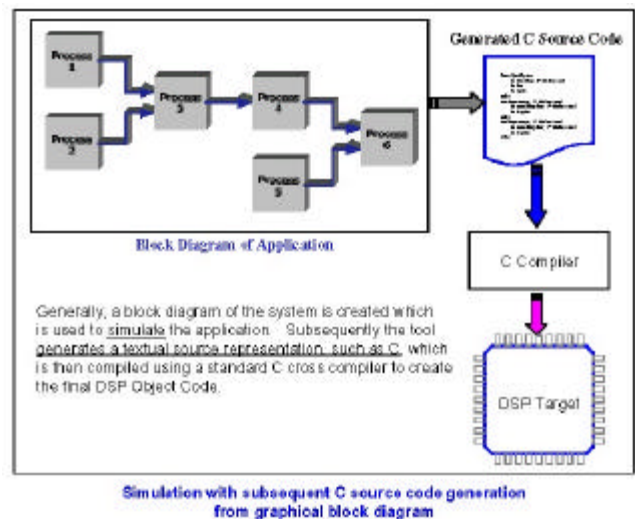
3.2 Limited real-time Development on PC

In this case, a block diagram of the system is created which may actually run in real-time on a PC. Typically, the acquisition is performed by a sound card with microphone/speakers or standard PC-based video CCD camera.

Of course this approach requires that the software used for the development of the block diagram must provide sufficient speed so as to allow the design algorithm to run in real-time on the host PC.

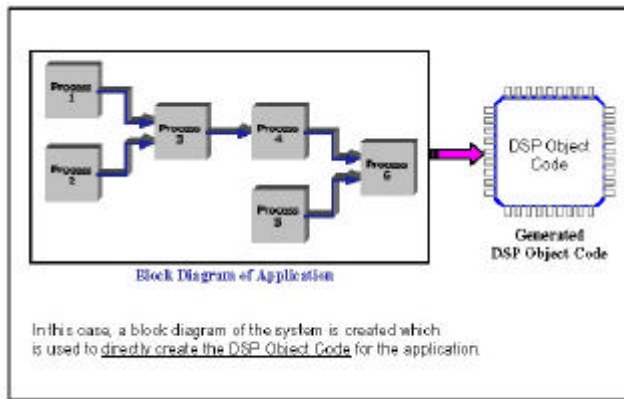
3.3 C Source Code Generation

Another means of designing an algorithm via graphical means is to employ C source code generation techniques in conjunction with a PC-based block diagram simulation. In this scenario the DSP algorithm is first constructed as a block diagram simulation that executes on the host PC. After the desired algorithm has been constructed and the simulation yields the desired results, then the entire block diagram design can be generated as its C source code counterpart. Typically this is done through the use of a source code generation tool that is capable of producing the C source code that implements the simulated design.



3.4 Direct DSP Chip Support – Object Code Generation

Direct DSP chip support through object code generation from a component-based block diagram allows for the graphical programming of a DSP without ever requiring that source code be produced for the design. With this approach the ‘source code’ for the design algorithm is the block diagram representation itself. This design environment allows for both a fine-grained and large-grained approach while offering design flexibility, efficient object code generation and algorithm maintainability.



Direct DSP Object Code Generation from graphical block diagram

The direct DSP object code generation approach ensures that no C or assembly language source code need ever be required in order to program a DSP. Programming is done at an intuitive block diagram level, and the result is an executable DSP program – a program of the same type that is produced by a traditional C-compiler/assembler/linker. Data flow and the program execution are governed by graphical constructs as opposed to those used in C or assembly.

3.5 Live Real-time Development with DSP

In this case, a block diagram of the system is created which is literally running in real-time on a DSP target system. The Block Diagram is actually live and interactive, allowing the engineer to modify parameters ‘on-the-fly’ and observe actual real-time waveforms and data.

A powerful capability lies in the ability to construct a DSP algorithm graphically and then immediately see the results running on the target DSP. The block diagram algorithm is not being run as a simulation, it is instead executed directly on the target DSP. This capability allows for efficient design of the algorithm. Data can be viewed graphically and DSP profiling information can easily be obtained.

4 IMPORTANT CONSIDERATIONS FOR GRAPHICAL DSP DESIGN AND DEVELOPMENT

There are several important considerations involved when evaluating the merits of using a graphical design approach to implement DSP applications. These include whether or not the design tool provides for an Open Software Architecture, whether or not the tool produces efficient DSP implementations, whether the tool offers true, n-level hierarchical support, and the ability to properly handle recursion in component-based block diagram designs.

4.1 Open Software Architecture

An Open Software Architecture must allow for the ability to quickly create new block components, provide design efficiency, support true hierarchical design, support recursive

data paths, provide an interoperability with existing DSP development tools, and most importantly, provide a DSP-centric focus to algorithm design & development.

The designer who uses a graphical development tool to produce DSP applications must not be constrained to using a fixed set of block components when implementing his/her design. There must be a convenient method for the designer to augment the existing set of block components with unique, custom functions.

The ability to graphically create new block functions by encapsulating existing block components is an important feature of a graphical DSP design environment.

If a graphical approach to DSP design is to be accepted it must supply a means of adding unique block components that implement a designer’s custom algorithms. In many companies the engineer will have existing bodies of C and Assembly source code that reflect years of hard-earned intellectual property; a good graphical development tool must be able to leverage this textual base of code efficiently and practically.

4.2 Efficiency

The graphical design/development tool and the actual DSP program produced by the tool must both be efficient.

The resulting DSP program that is produced should be one that is free of unnecessary overhead and implements the desired design algorithm in real-time on the target DSP.

4.3 True Hierarchical Design

Hierarchical design is important when creating block diagram representations of a DSP application. However, true n-level hierarchy must ensure that the designer can create hierarchy blocks that consist of other hierarchy blocks, that in turn use other hierarchy blocks, and so on.

4.4 Recursive Support

Many DSP algorithms require the use of data feedback paths and rely on the use of recursion in a design application; IIR filtering and control-oriented feedback systems are typical examples which inherently require recursive structures. The ability of a graphical block diagram development tool to meet this need is critical to producing DSP applications.

4.5 Interoperability with Existing Development Tools

Interoperability with conventional algorithm development tools is an important consideration. As users desire to add their own ‘routines’ or IP to their design by creating custom block components they can either create them graphically through a hierarchical approach or by generating a block component source code template using existing code generation tools.

5 BENEFITS VERSUS COST TRADE OFF

The engineer involved in creating a DSP application has to consider many factors when deciding on how best to approach the task at hand. The benefit vs. cost tradeoff over application development concerns and application design concerns need to be examined.

5.1 Time to Market

Time-to-market is one of the most critical attributes of today's DSP designs; getting to market late can be a death sentence to an otherwise good product concept. With graphical DSP design, the design schedules can shrink considerably, getting products out the door much more quickly than was done previously. In addition, the typical iterative concept/code/debug cycle where much engineering time is spent, is reduced dramatically, as designers can quickly try things in real-time and observe the effects of the DSP dynamically.

5.2 Learning Curve Requirements

The learning curve for this technology is relatively shallow, allowing its use on even first-time (perhaps only-time) DSP applications.

If not entirely eliminated, the time spent using these tools may be reduced substantially reducing expertise requirements for many DSP applications. This reduction of learning curve will allow DSP to be used in new products and market areas that were traditionally non-DSP.

5.3 Maintainability

One of the benefits of designing graphically is that the components used are true "Black Box" components, and connecting them graphically precludes the possibility of the internal workings of one component intermixing with another.

By ensuring the use of true "Black Box" design, maintainability of design increases.

5.4 Cost

In many case studies it has been shown that the capital investment in a good graphical DSP design tool is easily recouped in reduced man hours and in some cases lower hourly rates due to not requiring highly experienced and expensive programmers.

5.5 Application Design Concerns

Another important consideration relates to how good the actual produced code is when the algorithm has been designed graphically.

Where DSP object code has been generated graphically, it is usually faster and more efficient than code produced using conventional textual languages such as C.

6 PARALLEL PROCESSING

With more and more compute intensive application, it is important that processing power is available to meet these demands. One way of achieving this is to use multiple DSPs. It is necessary therefore when considering design and development tools to ensure parallel processing is supported.

This will usually require additional functions such as tasks splitting and the addition of threads and code needed for inter-task communications. It should also be able to create code and build files as needed to use multi-processor features of the target operating system plus create make files with all task and processor specific dependencies established.

7 APPLICATIONS

All areas of DSP design may benefit from Graphical Design tools and a few examples are given below

7.1 Communications

An example of a communication design might be a modern transmitter application using the single Phase Shift Keying (PSK) method of modulation. Here 8 blocks could be used with the use of hierarchy where the Pseudo Random Noise Generator block is in itself a complete worksheet.

7.2 Control

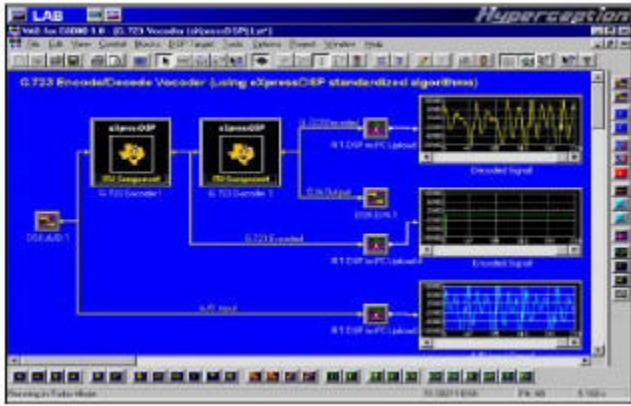
It would be very quick and easy to design a motor control application graphically. Motor control is a good example for the need for recursive structures (feedback).

7.3 Professional Audio

Music synthesisers, voice modification units, basic instruments can be designed, as well as real time analysis of pro-audio related applications. An example of the latter might be voice pitch analysers with envelope displays.

7.4 Telecommunications

An example of a block diagram design of a simple G.723 Encode/Decode application is shown below.



The application takes an audio input such as a microphone, or other audio source and performs some DSP processing. The RIDE/VAB worksheet applies several XDAIS-compliant algorithms to the audio stream and then sends it to an audio output so that the results can be heard over a speaker. The data is also uploaded from the DSP for subsequent graphical display so that a visual comparison can be made between the original input signal and the decoded signal.

The audio output will depend on whether or not the G.723 algorithm is being used. The worksheet can be user-modified so that the audio input can be connected directly to the audio output (bypassing the vocoder encode and decode block components). The results can then be compared to those achieved when using the eXpressDSP compliant vocoder block components. In this example, the most noticeable differences become apparent when music that has a wide dynamic range is used as an input to the DSP hardware.

7.5 Research

The ability to quickly modify DSP designs and change parameters 'on-the-fly' lends itself well to research and development efforts. Simply by using a live DSP target, the user may quickly observe waveforms, data etc., while the DSP is running in real-time. The quick iteration cycle of modifying the system and observing the effects saves much time during typical research endeavours. In addition, those performing the research need not expend time on becoming C and Assembly language experts, but can begin to graphically program their DSP system immediately.

7.6 Teaching/Education

Obviously a graphical method of programming DSP's translates into the academic setting rather easily. The potential for teaching concepts and seeing them run in real-time, without the prior requirement for the students to be experts in writing C or Assembly source code has real benefits. This graphical method of programming DSP's is being used in a number of school systems for allowing real-

time DSP using actual real-world hardware to be taught/used at much earlier levels.

8 CONCLUSIONS

A graphical oriented design methodology, represents a significant milestone with respect to programmable DSP design; heretofore, only textual based representations (C, Assembly) were the basis for creating algorithms for a DSP. With this powerful technology engineers will be able to design much or all of their DSP algorithm, or software, graphically as opposed to textually. The savings in design/development time as well as the advantages of maintainability and the self-documenting nature of a graphical design is of considerable importance in widening the use and popularity of DSPs in a variety of applications, many of which have yet to be conceived.

Graphical component-based design of algorithms will play an important part of the future for DSP and related engineering areas, as well as software architecture design in general. Graphical DSP Design represents a modern approach to minimising the learning curve, improving the maintainability, and to solving some of the problems presented by classical textual-based algorithm development tool approaches. The benefits of this design approach are many and should be worth considering for many new and existing DSP applications.

REFERENCES

- [1] J Zachman, Hyperception "Building DSP Applications via Graphical Design – "Does a picture 'cost' a thousand words?"
- [2] B LaMaccia, Momentum Data Systems "Pegasus Software Development Environment"