

Building DSP Applications via Graphical Design – “Does a picture ‘cost’ a thousand words?”



Using component-based DSP programming to simplify development

Discover how graphical component-based DSP programming can minimize development cycles, reduce risk, and simplify complex DSP projects.

Table of Contents

| | |
|---|-----------|
| Section 1. Overview of Graphical Design for DSP | 7 |
| 1.1. What is 'Graphical DSP' or 'Block Diagram' Programming?..... | 7 |
| 1.2. Component Oriented Nature of Design – “Black Box” Design | 8 |
| 1.3. Graphical Compiler Analogy..... | 8 |
| 1.4. The Integrated Circuit – Possible Case Study? | 8 |
| Section 2. Several Methods of Graphical DSP Programming..... | 9 |
| 2.1. Simulation/Modeling on PC | 9 |
| 2.2. Limited Real-time Development on PC | 10 |
| 2.3. C Source Code Generation | 11 |
| 2.4. Direct DSP Chip Support - Object Code Generation | 12 |
| 2.4.1. Does not require C or Assembly Language Programming..... | 12 |
| 2.4.2. Fine Grained - Compiler Analogy | 12 |
| 2.4.3. Large Grained – Component Delivery/Usage Vehicle..... | 13 |
| 2.4.4. Building DSP Applications (Generating DSP Object Code Files) | 13 |
| 2.5. Direct DSP Board Support - Live Real-time Interactive Development with Actual DSP Hardware | 14 |
| Section 3. Important Considerations for a Graphical DSP Design & Development Tool..... | 15 |
| 3.1. Open Software Architecture | 15 |
| 3.1.1. Creating New Components | 15 |
| 3.2. Efficiency | 16 |
| 3.3. True Hierarchical Design..... | 16 |
| 3.4. Recursive Support..... | 16 |
| 3.5. Interoperability with Existing Development Tools | 16 |
| 3.5.1. DSP Development Software | 16 |
| 3.5.2. DSP Development Standards..... | 16 |
| 3.5.3. DSP Development Hardware | 17 |
| DSP-Centric Focus..... | 17 |
| 3.5.4. Fixed-Point / Floating-Point | 17 |
| 3.5.5. Single Sample Processing / Block-Oriented Processing..... | 17 |
| 3.5.6. Direct Support for Real-world Interface (A/D, D/A)..... | 18 |
| Section 4. Targeting DSP Hardware – Graphical Tool for “Rapid Production” | 20 |
| 4.1. Ability to target Embedded DSP Applications | 20 |
| 4.2. Ability to target plug-in Industry Standard DSP Development Boards..... | 20 |
| Section 5. Applications of Graphical DSP Design..... | 21 |
| 5.1. Communications..... | 21 |
| 5.2. Control | 22 |
| 5.3. Pro Audio | 22 |
| 5.4. Telecom..... | 23 |
| 5.5. Research | 24 |
| 5.6. Teaching/Education..... | 25 |
| 5.7. Virtually Anything else... .. | 25 |
| Section 6. Benefit vs. Cost Tradeoff..... | 26 |
| 6.1. Application Development Concerns | 26 |
| 6.1.1. Time-to-market | 26 |
| 6.1.2. Learning Curve Requirements..... | 26 |
| 6.1.3. Maintainability | 26 |
| 6.1.4. Interoperability with Existing Development Tools..... | 26 |
| 6.1.5. Cost..... | 27 |
| 6.2. Application Design Concerns | 27 |

| | |
|--|-----------|
| 6.2.1. Application Speed..... | 27 |
| 6.2.2. Application Size | 27 |
| 6.2.3. Embedded Development Issues..... | 27 |
| 6.2.4. Device Independence | 28 |
| Section 7. Hyperception's Product Offerings in Graphical Design Tools | 29 |
| 7.1. Hypersignal [®] Block Diagram | 29 |
| 7.2. RIDE | 29 |
| 7.3. VAB (Visual Application Builder) | 29 |
| 7.4. ImageDSP Image Processing Software..... | 29 |
| Section 8. Summary | 30 |
| 8.1. Future DSP Development | 30 |
| 8.2. Contact Information | 30 |

List of Figures

| | |
|---|----|
| Using component-based DSP programming to simplify development | 1 |
| PC-based Simulation from graphical block diagram..... | 9 |
| PC-based limited real-time development from graphical block diagram | 10 |
| Simulation with subsequent C source code generation from graphical block diagram | 11 |
| Direct DSP Object Code Generation from graphical block diagram..... | 12 |
| Live real-time interactive development on DSP hardware from graphical block diagram | 14 |
| Frame-based Architecture for Real-time DSP – Telecom and other Applications | 18 |
| Interrupt-based Architecture for Real-time DSP – Control Applications | 18 |
| Simple PSK modem application graphically designed with RIDE/VAB for C5000 | 21 |
| Design example for a real-time ACI motor control system using a C2000 DSP and RIDE/VAB .. | 22 |
| Example with RIDE allowing for fine-pitched voice analysis in music | 23 |
| Simple G.723 Encode/Decode application graphically designed with RIDE/VAB for C6000..... | 24 |

Section 1. Overview of Graphical Design for DSP

Just draw a block diagram of your design, produce real-time working code for a DSP – sounds good, but how well does it work? What about the typical engineering tradeoff – better level of abstraction for my design, but at what cost? And for DSP applications, if you're not running real-time at the end of the day, what good was it – just an exercise in novel approaches that look good, but don't produce?

Hyperception has been studying this problem for well over a decade (quite a long time in DSP years) and has produced a unique approach to graphical design for DSP – our Real-time Integrated Development Environment or RIDE[®], and the subset product Visual Application Builder or VAB[™]. In particular, RIDE/VAB can directly produce DSP object code from a graphically constructed, user-created algorithm to achieve an overall software design that works on a programmable DSP. Many application areas exist and include, but certainly are not limited to, those found in telecom, image processing, speech processing, audio processing, control [such as motor control, and other feedback systems], robotics, and wireless applications.

Historically, textual-based representations for algorithms (such as Assembly and C) along with corresponding assemblers and compilers were 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 labor 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.

Development of algorithms with RIDE/VAB is similar to development in C in that variables, operators, expressions, and functions are used to create other functions, and/or the overall program. The difference is that each of these constructs is expressed graphically through component-based design as opposed to textually. Although initially quite different than textual-based software coding, the advantage of developing an algorithm similar in nature to how an engineer typically constructs and conceptualizes (i.e., a block diagram) has definite advantages during the course of a project's development.

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. And with a small amount of initial learning, well written, easy-to-follow, modular, algorithms can be designed very quickly by the user. The RIDE/VAB products were created by virtue of leveraging over 15 years of experience in DSP development software for the PC.

1.1. 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.

1.2. Component Oriented Nature of Design – “Black Box” Design

Component-based design methodology affords the designer with the luxury of not having to code each individual algorithm function 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.

1.3. Graphical Compiler Analogy

RIDE/VAB 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 (i.e., C source code) into DSP object code, RIDE/VAB turns a graphical language (i.e. block diagram) directly into DSP object code. Similar to a C-compiler or linker, RIDE/VAB 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 optimize 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 self-documenting nature of graphical design are considerable.

Similar to a C compiler, 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 simulators are used in conjunction with textual-based code development, they also may be used along with code developed graphically.

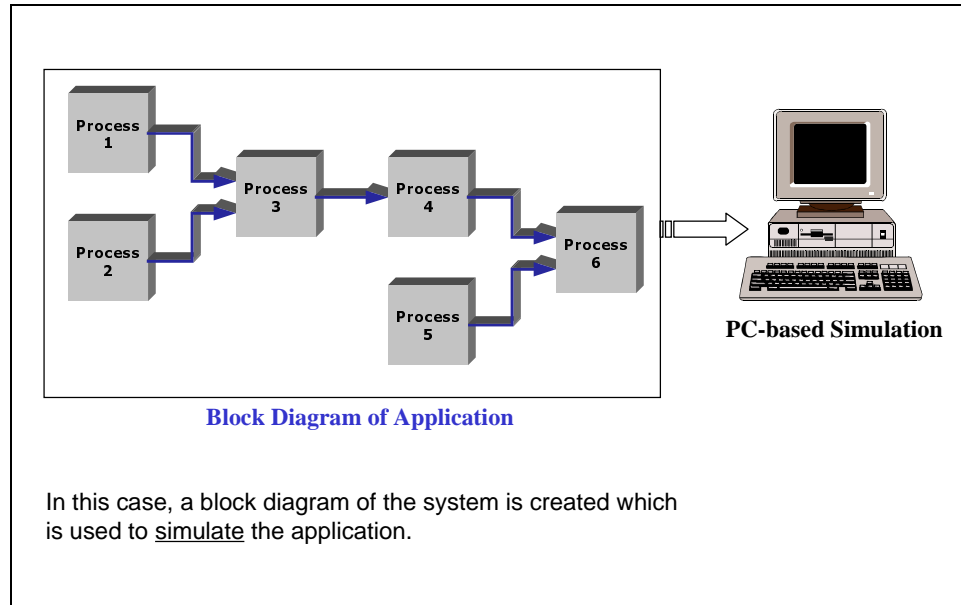
1.4. The Integrated Circuit – Possible Case Study?

In the hardware design world there may exist a possible case study: in the earlier years when products were designed using resistors, capacitors, and transistors (or vacuum tubes), sometimes hand-soldered from lead to lead, design cycles were much longer, maintainability and reproducibility of design was inferior to that of today. Could it be that the introduction of component based design in the hardware world – the introduction of the Integrated Circuit (IC) – improved overall hardware design in a fashion similar to what might be expected from component software design? When looking at the back of old radios and TV's, does the hand soldering of resistor to capacitor remind us of hand crafting assembly or C textual source code? We believe that it does, and with graphical component-oriented design, the software design world may reap the benefits that have already been seen in the hardware design world.

Section 2. Several 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 modeling, 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. Although Hyperception has products that address all these facets well, it is the latter that represents a new, inventive, and powerful means of graphically programming a DSP.

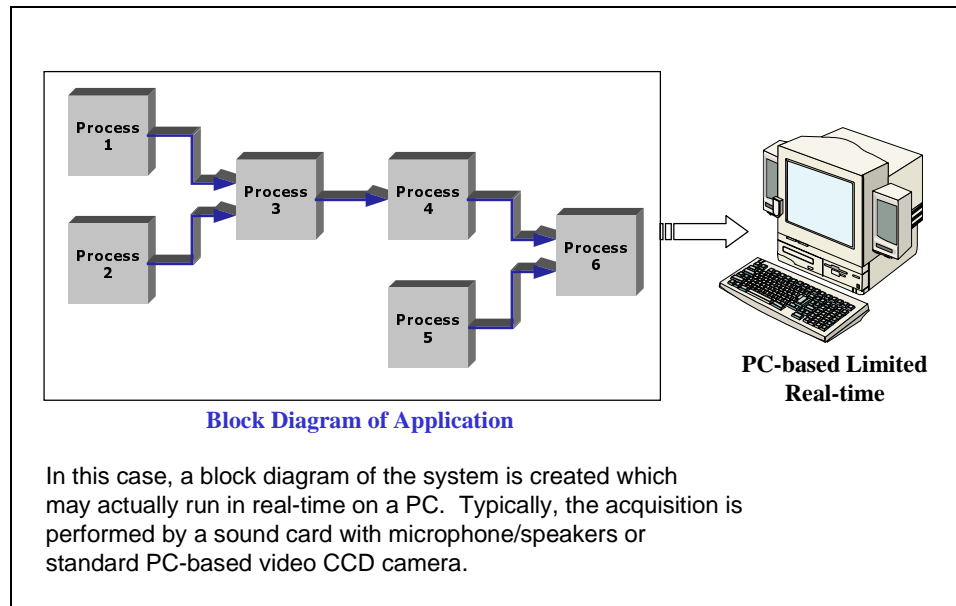
2.1. Simulation/Modeling on PC



PC-based Simulation from graphical block diagram

One method where a block diagram approach might be used in the development of a DSP application lies in the simulation and modeling 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 is 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. An example of this would be found in Hyperception’s Block Diagram product. Although for many applications this use of graphical block diagrams is all that is required, for others the fact that the design cannot be executed directly on a DSP is a drawback.

2.2. Limited Real-time Development on PC

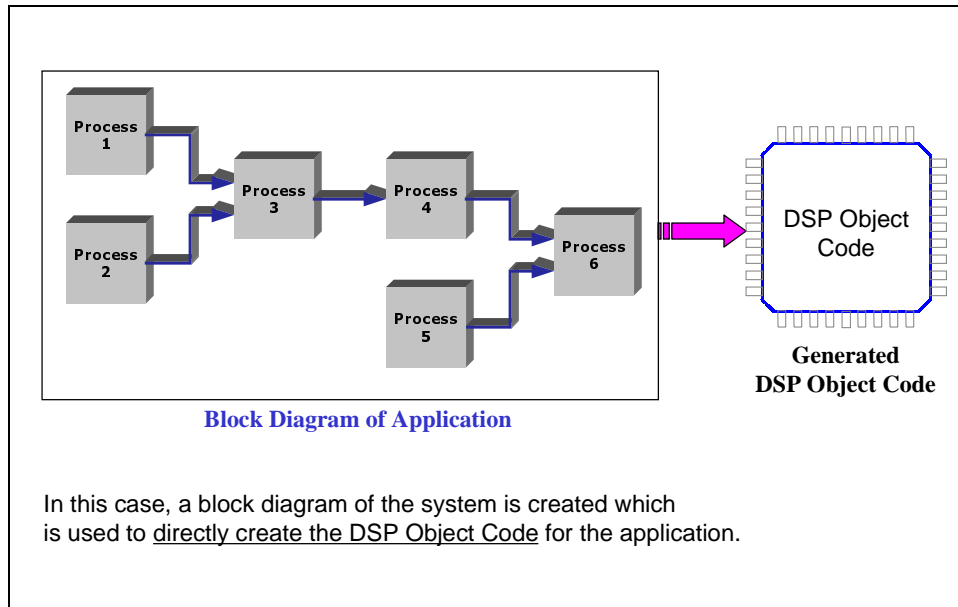


PC-based limited real-time development from graphical block diagram

Another interesting method of developing a DSP application from a graphical approach is the fact that with today's faster PCs and the use of low-cost **sound cards** and **video cameras** allows some limited real-time DSP applications to be constructed and implemented on a PC. 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. Again, an example of this is Hyperception's Block Diagram product in which the individual block components run at executable speed, as opposed to other block diagram type of products that run at interpretive (or script-driven) speed. Advantages of using a sound card or video camera with a PC to allow limited real-time development include the relative inexpensive nature of PC-based sound cards and video cameras (as opposed to dedicated DSP hardware). Some drawbacks include limitations of the rate at which an algorithm can sustain real-time operation, as well as some delay issues associated with perhaps using a larger framesize to allow the processing of the algorithm in real-time. Additionally, direct support for networked PCs such as TCP/IP over standard Ethernet allows for scalable systems to be constructed from a number of low-cost PCs to address higher processing bandwidth applications, perhaps for a data intense real-time system.

Could there possibly be a better way in which to develop real-time DSP algorithms from a graphical block diagram approach?

2.4. Direct DSP Chip Support - Object Code Generation



Direct DSP Object Code Generation from graphical block diagram

RIDE/VAB DSP development software from Hyperception offers a significant improvement in the way in which a DSP can be programmed from a graphical block diagram approach. These software tools have benefited from over 15 years of experience in DSP development software for the PC. Unique to these tools is the capability of generating DSP machine code directly from the block diagram algorithm itself. 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.

2.4.1. Does not require C or Assembly Language Programming

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.

2.4.2. Fine Grained - Compiler Analogy

One advantage of RIDE/VAB component-based graphical block diagram design to produce direct DSP object code is the inherent ability to support a fine-grained approach to signal processing. Individual block components each represent algorithms that are packaged as small units of machine code that implement the intended function. This fine-grained approach allows for such low code overhead that they could be considered to be "inline assembly" code. This implies a powerful capability: the designer can very quickly create his own unique functions by incorporating lower-level block components in a hierarchical fashion to construct new block components – *all the while maintaining very low code overhead*. The capability of being able to

construct a DSP design algorithm from a fine-grain approach without losing performance or increasing overhead is quite extraordinary.

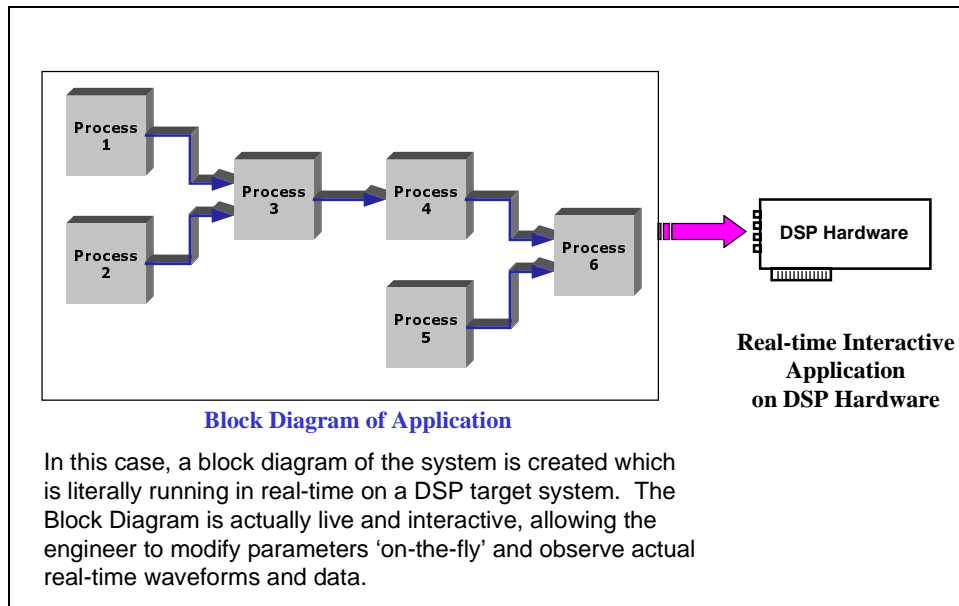
2.4.3. Large Grained – Component Delivery/Usage Vehicle

In addition to providing a fine-grained approach, RIDE/VAB DSP software allow for a large-grained approach to programming a DSP through direct object code generation. Instead of thinking in terms of constructing a DSP algorithm from low-level block components, the large-grained approach to DSP design allows for the capability of using just a few block components to implement the entire algorithm. For example, place a Modem block component into an empty worksheet, and voila, you've just implemented your modem design on a DSP. The ability to take a very large grain approach in a graphical design is an important factor to consider. This aspect lends itself to the notion that a graphical representation of an algorithm can be represented by high-level block components that implement the entire desired system. This ability enables the development tool to be used as a delivery vehicle to rapidly produce sophisticated DSP designs.

2.4.4. Building DSP Applications (Generating DSP Object Code Files)

Implementing DSP applications for real-time operation by directly generating efficient DSP object code can be as simple as selecting the desired function components, establishing data flow and conditional logic amongst the components through line connections, and selecting individual component setup parameters. The resulting DSP executable is ready for use in the intended DSP application.

2.5. Direct DSP Board Support - Live Real-time Interactive Development with Actual DSP Hardware



Live real-time interactive development on DSP hardware from graphical block diagram

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 RIDE/VAB environments offer support for a wide variety of DSP hardware and enable the designer to get immediate feedback as to how the algorithm is running on the 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 at any (or many) point(s) in the block diagram implementation. DSP profiling information can easily be obtained so that you can see just how much – or little – processing power is being consumed by the design. The capability of getting immediate feedback (e.g. waveform displays, D/A output, etc.) in the design is an important consideration when using a graphical design environment.

Section 3. Important Considerations for a Graphical DSP Design & Development Tool

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.

3.1. Open Software Architecture

The RIDE/VAB DSP development environments are based upon an Open Software Architecture that employs object-oriented programming concepts. These tools were designed with the idea in mind that DSP developers must be able to expand upon the base development tool and easily customize it to meet their unique requirements. 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.

3.1.1. Creating New Components

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. With Hyperception graphical products, there are two methods of creating new (user-developed) block components – graphically within the environment, or conventionally external to the environment.

Graphically Creating New Components

The ability to graphically create new block functions by encapsulating existing block components is an important feature of a graphical DSP design environment. By leveraging lower-level block components a designer can create a single more powerful block component without ever having to write any C or assembly code. For example, a series of Delay, Gain, & Add optimized block components can be used together to implement an optimized FIR filter, and then the entire grouping be condensed (via hierarchy) into a new, single block component. This new block component could then be used in subsequent DSP designs. Creating customized block functions via a graphical method is a powerful feature of RIDE/VAB DSP software development tools.

Using Conventional Compilers/Assemblers to Create New Components

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. To address this issue both a Block Wizard and an eXpressDSP Component Wizard exist that will allow for the quick creation of new block components. These robust wizards, which have been developed for over a decade, will walk the user through a series of screens in which certain aspects of the intended function are discerned. The resulting source code templates that are generated by the wizard will allow the user to add their specific algorithm to the template. The completed templates are then compiled by traditional methods such as Microsoft Visual C/C++ and an appropriate DSP vendor C-compiler/assembler/linker. By using this approach the designer is able to create entire libraries of block components to augment those supplied standard with RIDE/VAB.

3.2. Efficiency

The graphical design/development tool and the actual DSP program produced by the tool must both be efficient. The designer should be able to move quickly when constructing their algorithm and should not be constrained by the development environment itself. An efficient development environment should take care to make judicious use of available DSP memory, provide immediate feedback capabilities (such as waveform displays and memory read/write) for “on-the-fly” DSP development, and reduce any overhead to bare minimums. 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.

3.3. True Hierarchical Design

Hierarchical design is important when creating block diagram representations of a DSP application. From an aesthetic point-of-view, hierarchy makes the entire design more readable and easier to follow. It also allows you to take better advantage of the block diagram “real-estate”. Of course, hierarchy is also a convenient method employed when creating custom block components from existing lower-level blocks. 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. The graphical design environment must also be constrained by the fact that the resulting hierarchical components be treated in the same manner as are individual block components. For example, the designer should have the capability of using conditional connection logic to govern under which conditions, and how many times the hierarchical block should be executed. True hierarchical design is often “easier said than done” and serves as an important part of the RIDE/VAB graphical design environments.

3.4. 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. It is important for the tool to be able to provide a proper answer to the “chicken and egg” question of recursive algorithms.

3.5. Interoperability with Existing Development Tools

3.5.1. DSP Development Software

Interoperability with conventional algorithm development tools was also an important consideration for RIDE/VAB development software. 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. In the latter case, the user may feel comfortable in leveraging existing code generation tools (C compiler, assembler) to create these new components that may then be used by RIDE/VAB. Both products support standard DSP linkable libraries in addition to any newly created DSP object files that have been created via ‘conventional’ means, and can typically work alongside the DSP debuggers used in textual-based DSP development.

3.5.2. DSP Development Standards

One nice example of interoperability with an existing DSP development standards is the eXpressDSP Component Wizard. This wizard works with RIDE/VAB to help the design engineer create algorithms that adhere to the Texas Instruments TMS320™ DSP Algorithm Standard. These eXpressDSP-compliant algorithms can be used directly in the RIDE/VAB graphical design environments. The Component Wizard automatically generates the textual C source code base code that is required for adherence to the standard, and offers direct support of TI’s Code Composer Studio™ (CCS) product. This level of interoperability provides a saving of considerable man-time that is normally associated with developing an algorithm from scratch. In

addition to the eXpressDSP™ and CCS support (and upcoming support for DSP/BIOS™) RIDE/VAB can be thought of as a simple method in which to harness components effectively.

3.5.3. DSP Development Hardware

Lastly, support for many industry-standard DSP boards, hardware is an important capability. RIDE/VAB support a number of common DSP boards, including low-cost “DSK”, “EVM” and “EZ-Kit/Lab” types of hardware from the major DSP semiconductor companies.

DSP-Centric Focus

If a graphical design environment is used to develop DSP applications, shouldn't that environment be one that has been built to provide a DSP-centric focus?

Just because a tool provides a block diagram interface doesn't guarantee that the algorithm produced actually employs a DSP-centric design approach. A block diagram development tool may simply be making use of some “add-on” toolbox to address a DSP function library. Often times this add-on library is driven from a sales-point-of-view to satisfy a market demand. The fact that a toolbox of DSP functions may exist does not imply that the development tool itself will produce optimal DSP block diagram implementations. However, RIDE/VAB block diagram environments have been designed to employ DSP-centric concepts with DSP algorithm development in mind: *they were built for speed.*

A DSP-centric focus requires that issues such as fixed-point and floating-point arithmetic are addressed, and that the DSP design can support both block-oriented, frame-based processing and single-point sample-by-sample processing. Additionally, direct support for a real-world interface must be available.

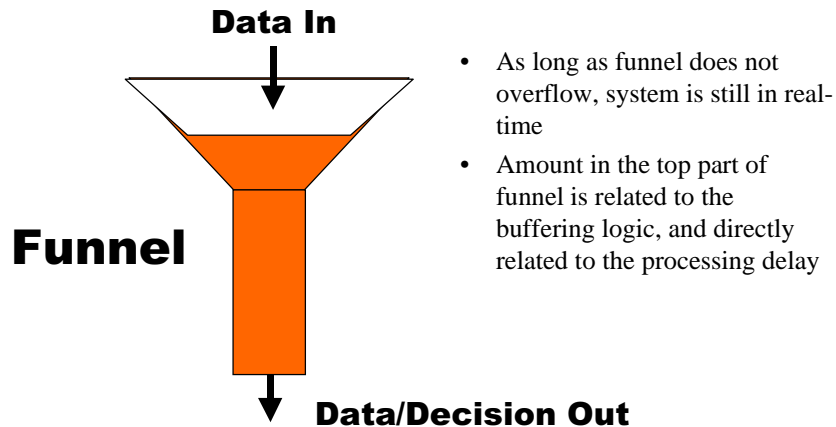
3.5.4. Fixed-Point / Floating-Point

An algorithm that is to be executed on a DSP must consider whether the DSP is a fixed-point or floating-point processor. A graphical DSP development tool should provide for this capability so that the designer is able to select precision settings for any block components used in the block diagram design.

3.5.5. Single Sample Processing / Block-Oriented Processing

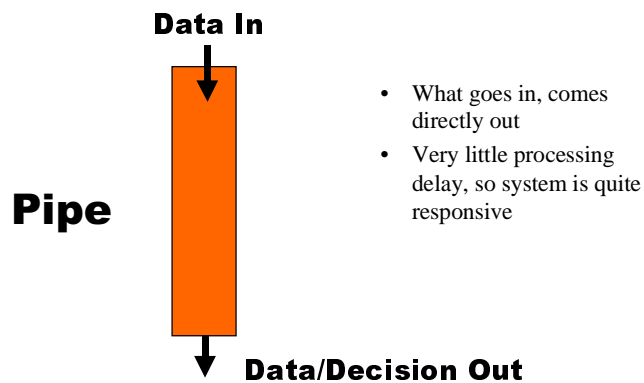
The graphical design environment should be flexible enough to provide for both single-sample processing and block-oriented, frame-based processing in order to address DSP algorithm design concerns. Many DSP applications (e.g. control systems) require that single-sample processing be applied, while others (e.g. speech processing) make use of a frame-based approach. Each of these processing types both have their own unique requirements and a DSP-centric graphical design environment such as those offered by Hyperception will properly address them. The following two figures use analogies wherein the data to be processed by a real-time DSP system is shown as water; the first figure demonstrates the model for a frame-based algorithm (funnel), and the second figure demonstrates the model for an interrupt-based, or single-sample based system (pipe).

Typical Frame-based model for Real-time



Frame-based Architecture for Real-time DSP – Telecom and other Applications

Interrupt-based model for Real-time



Interrupt-based Architecture for Real-time DSP – Control Applications

3.5.6. Direct Support for Real-world Interface (A/D, D/A)

Often implied in the construction of a DSP application is the fact that the DSP will be operating on some real-world data in an effort to effect some desired result. There is often a need for a designer to be able to access real-world data via an analog-to-digital (A/D) conversion process and/or generate data via a digital-to-analog (D/A) conversion process.

A DSP-centric approach to graphical DSP development requires that there be some means of directly supporting these A/D and D/A processes so that the designer is able to use actual real-world data during the construction of their application. This ability to receive **immediate** feedback while working on their DSP application is a powerful development consideration. Graphical programming via direct DSP object code generation is the only way to make this type of development possible.

Section 4. Targeting DSP Hardware – Graphical Tool for “Rapid Production”

The RIDE product directly supports a variety of important DSPs and includes the Analog Devices SHARC™, as well as the Texas Instruments C2000™, C5000™, and C6000™ ISA families (C3x™ is also supported). VAB is a subset of the RIDE product and directly supports only Texas Instruments DSPs (and does not support PC-based simulation as RIDE does).

RIDE/VAB graphical design products allow you to quickly produce a completed DSP application. The finalized DSP program can be implemented on a target DSP from directly within the RIDE/VAB environment such that the designer can change algorithm parameters “on-the-fly” or view data results through a GUI. Additionally, the finalized DSP program can be built to address an embedded DSP application.

4.1. Ability to target Embedded DSP Applications

RIDE/VAB software can be used to develop complete DSP programs and then target embedded DSP applications. The Application Build command is powerful feature that is used to produce the current real-time design to an executable DSP application that can be used in an embedded system or other platform. The output DSP executable file can be used to program Flash memory, loaded into a simulator or loaded into target DSP hardware with an emulator/debugger.

The DSP executable file that is produced is effectively the same as if you had written conventional textual source code yourself, and then used a DSP assembler/compiler/linker to create it.

Because the RIDE/VAB memory map is user-configurable, and the design environment supports a “virtual” DSP mode without requiring a DSP board in the system, applications can be created and produced for any target hardware. In this usage, RIDE/VAB may best be thought of as an advanced DSP ‘Graphical Compiler’.

4.2. Ability to target plug-in Industry Standard DSP Development Boards

Using RIDE or VAB in conjunction with a DSP development board makes for an extremely effective development environment. Algorithms can be quickly constructed and immediately executed on the development hardware. In this way the designer is able to quickly gauge the performance of their design, and make immediate modifications to his algorithm if necessary. The ability to build a DSP application directly in DSP memory provides many benefits. The DSP algorithm can easily be evaluated, with the evaluation yielding real-world results as opposed to providing simply an algorithm “simulation”. Finally, RIDE/VAB supports direct programming of the FLASH on Industry Standard DSP Hardware, making the term ‘**Rapid Production**’ a reality.

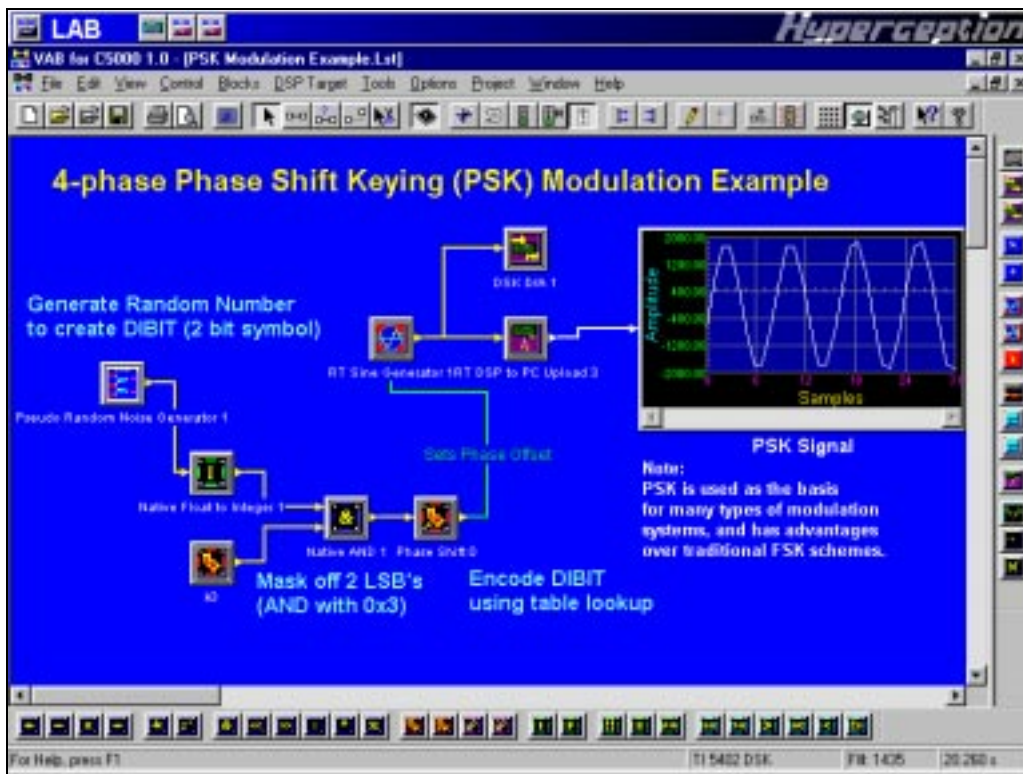
Section 5. Applications of Graphical DSP Design

Several examples of using RIDE/VAB to target specific DSP hardware are presented below. These design examples contain information on how the algorithm was actually created in VAB or RIDE. These examples will illustrate several important DSP application areas: *Communications, Control, Pro-Audio, Telecom, Image Processing, Research*, as well as *Teaching/Education*.

5.1. Communications

The screen capture shown below represents a graphically designed, basic modem application that is running on a C5000 DSP; the target in this case happens to be a C5402 DSK. The design process is the same as the C2000 and C6000 examples below, and of course the user may run directly in real-time (with live interchange of real-time data) within the environment, or use RIDE/VAB as an object 'Code Generation' tool to create the actual DSP object code file for use.

A modem transmitter application is designed graphically using simple Phase Shift Keying (PSK) method of modulation. A pseudo random generator is used to create the bits used to key the modulator. Note the use of hierarchy in the design of that component (Pseudo Random Noise Generator); this component was itself designed as a worksheet, or block diagram, and is actually made up of a number of other components. Right-clicking on this component would 'blow-up' the component, showing the complete block diagram, which could also include components which were designed hierarchically (to N levels). The random number is converted to integer, masked appropriately (to 2 bits), and then used to drive the actual phase value used by the carrier signal (RT Sine Generator) which is creating the output signal. The resulting signal is sent to the D/A as well as uploaded to allow interactive viewing of the modulation. If the D/A is driving a speaker, the signal may be listened to in real-time as the display is being updated live.



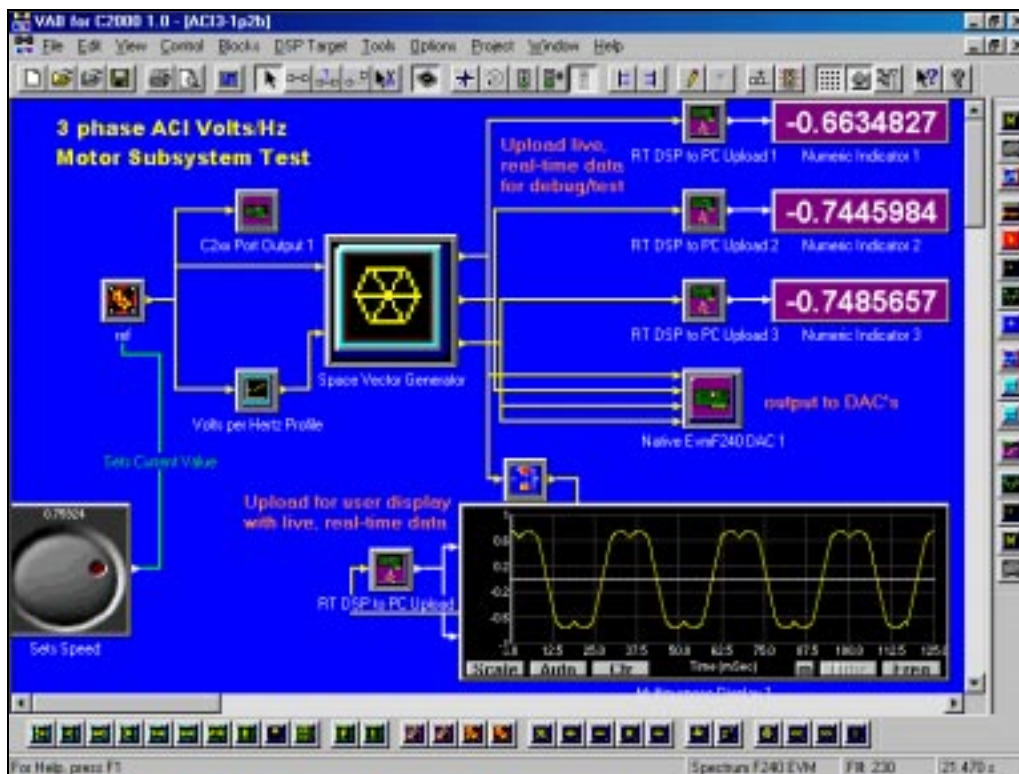
Simple PSK modem application graphically designed with RIDE/VAB for C5000

5.2. Control

The following screen capture illustrates an ACI motor control application designed graphically for C2000; the development hardware in this case is the F243 EVM, designed by Spectrum Digital. The design process is identical to the C5000 example above, with the user selecting those software components desired for this real-time application and connecting them together with a mouse.

In this example, which would likely be used during the development of a larger, more complex system, the interactive capability of RIDE/VAB is used to allow the engineer to vary the reference level (ref) corresponding to the desired set speed of an ACI motor application in a live, hands-on fashion. This speed value is echoed to some LEDs located on the target hardware, as well as being used to drive other software components (Volts per Hertz Profile, and Space Vector Generator) which then create the actual 3-phase real-time waveforms. These waveforms are sent to the D/A converter that could then drive the actual motor. The generated waveforms also happen to be uploaded (interactively, again) to allow the engineer to observe them as they are created, both as instantaneous values (numeric indicators) and as a complete waveform display.

Although not shown in this example, most motor control applications would make use of recursive structures (feedback) as well as drive the onboard PWM signals – both of which are supported by RIDE/VAB software. Once again, the actual DSP object code may be obtained directly from this block diagram or RIDE/VAB 'source' as an object file that looks as if it had come from an assembler/linker, ready for programming into ROM, Flash memory, etc.



Design example for a real-time ACI motor control system using a C2000 DSP and RIDE/VAB

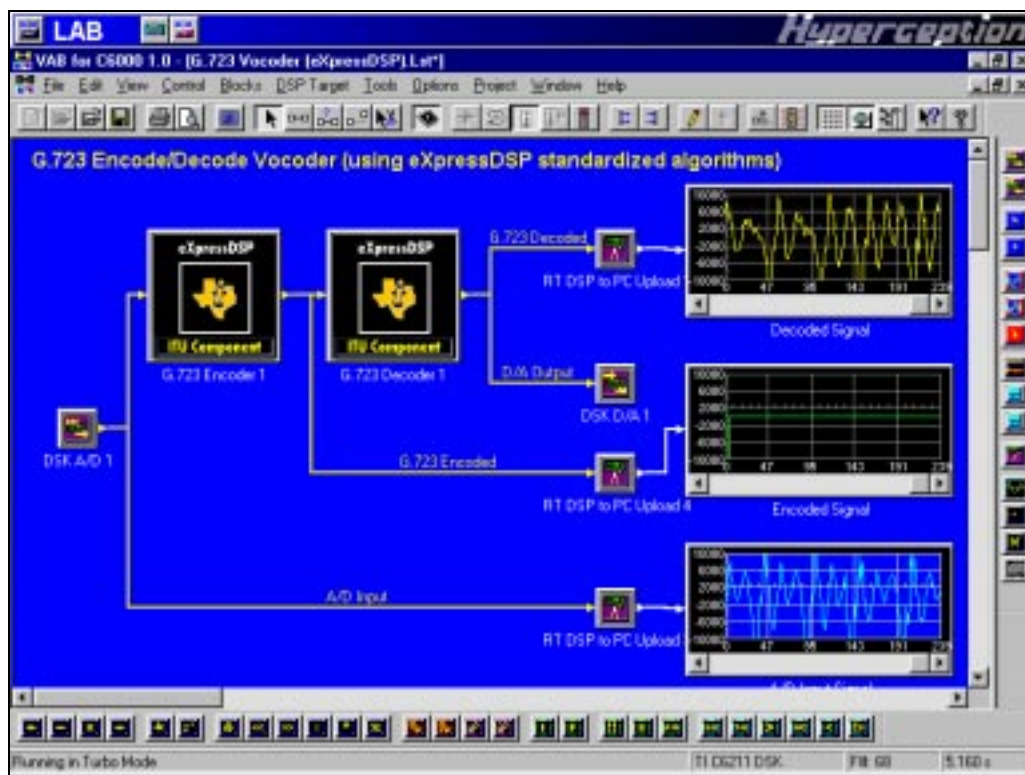
5.3. Pro Audio

In the figure below, an example application which could be used by those working in the audio market is shown. A graphical block diagram has been constructed to allow analysis of music,

be run directly in real-time on the target within the RIDE/VAB environment, with **no** textual source code (C, Assembly) generation steps required. If desired, the graphically designed algorithm may then be 'built' to a standard DSP object file, for programming into ROM, FLASH, etc., or downloaded to other targets (including a simulator) via Code Composer Studio.

As can be seen from the figure shown below, the audio input is represented in a RIDE/VAB worksheet through a single A/D block component (located near the left-hand side of the worksheet). The block diagram depicted in the worksheet graphically shows the data flow for the algorithm. The eXpressDSP compliant algorithms used in this worksheet example are two vocoder algorithms that are represented by the G.723 Encoder and G.723 Decoder block components. In fact, these components are actually the same vocoder algorithms that Texas Instruments include as part of its eXpressDSP Developer's Kit, with the same processing time requirements (so RIDE/VAB is not adding a layer of 'fat' to the design). The D/A block component in the worksheet represents the audio output data sent to the D/A converter.

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.



Simple G.723 Encode/Decode application graphically designed with RIDE/VAB for C6000

5.5. 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

endeavors. 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 relatively immediately.

5.6. Teaching/Education

Obviously a graphical method of programming DSP's translates into the academic setting rather easily. ¹The 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.

5.7. Virtually Anything else...

The graphical methodology employed by RIDE/VAB software makes it suitable for a wide variety of signal processing applications including those in image processing, medical-related technology, automotive applications, military applications, and many more. ²Application notes exist for a variety of product areas, with more market areas being addressed using DSP technology than ever before.

¹ For those interested, Hyperception has a white paper specific to using this graphical methodology for teaching real-time DSP. This paper may be requested by e-mailing info@hyperception.com .

² Application notes may also be requested by e-mailing info@hyperception.com .

Section 6. Benefit vs. Cost Tradeoff

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.

6.1. Application Development Concerns

Concerns throughout the development cycle for new DSP designs have been addressed well with RIDE/VAB. Items such as the time-to-market, learning curve, maintainability, interoperability, and cost are important factors that were used to drive the overall RIDE/VAB architecture. The following are considerations about developing DSP code graphically:

6.1.1. 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 RIDE/VAB, the design schedules can shrink considerably, getting products out the door much more quickly than done previously. Since the design is effectively the same or similar to the initial design block diagram, the implementation can be accomplished in much less time. 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 on the DSP dynamically, reducing hours into minutes. Since the resulting graphical design speeds project development by generating a conventional COFF file ready for production, the term 'Rapid Production' fits this type of design methodology particularly well.

6.1.2. Learning Curve Requirements

The learning curve for this technology is relatively shallow, allowing its use on even first-time (perhaps only-time) DSP applications. Since the construction of DSP algorithms is very similar to the design as drawn in block diagram fashion, and since designs may be implemented without need for C or assembly, it may be possible to eliminate the learning curve for these tools. 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.

6.1.3. 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. Another way of stating this is that it is virtually impossible to create an algorithm using 'spaghetti code', since the design is completely graphical, not textual. By ensuring the use of true 'Black Box' design, maintainability of design increases. Additionally, the fact that the RIDE/VAB design allows for hierarchical components (sub-system design) and that it corresponds to a typical engineering block diagram, allows for better maintainability and transfer to others should the initial DSP designer no longer be available to maintain the project. Reusability of software components is, of course, an integral part of this design paradigm.

6.1.4. Interoperability with Existing Development Tools

Interoperability with conventional algorithm development tools was also an important consideration for RIDE/VAB development software. 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. In the latter case, the user may feel comfortable in leveraging existing code generation tools (C compiler, assembler) to create these new components that may then be used by RIDE or VAB. Both products support standard DSP linkable libraries in addition to any newly created DSP object files that have been created via 'conventional' means. This level of interoperability

provides a saving of considerable man-time that is normally associated with developing an algorithm from scratch. In addition to the eXpressDSP and CCS support (and upcoming support for DSP/BIOS) RIDE/VAB can be thought of as a simple method in which to harness components effectively. Although VAB is specific to Texas Instruments, RIDE is generalized and includes direct support for Analog Devices SHARC processors, with additional DSP's being supported in an on-going fashion. Of course, the Enterprise Edition of Hyperception products, which includes the ANSI C source code generator, may be used with DSP's from virtually every vendor using a conventional textual based (C source) code approach.

6.1.5. Cost

Retail price points for Hyperception Graphical DSP Design products range from \$1.5k to \$10k; for most engineering companies this should not represent a significant barrier for their DSP application designs. When compared with the potential savings in both calendar time and Non Recurring Engineering (NRE) labor, the cost of these tools is very reasonable.

6.2. Application Design Concerns

Another important consideration relates to how good the actual produced code is when the algorithm has been designed graphically. The following considerations are related to **DSP object code which has been developed graphically:**

6.2.1. Application Speed

Concerns over the speed of the resulting graphically-designed DSP applications are probably the most common; ***what good is a DSP algorithm if it doesn't run in real-time at the end of the day?*** Although good common engineering sense and savvy would predict that the very nice level of abstraction afforded by the RIDE/VAB's graphical design must cost something – and that something must be efficiency – they would likely be wrong. We believe that the larger 'cost' was in the decade of development of the RIDE/VAB technology, not the resulting efficiency of code. If one only considers this graphical design as a component delivery vehicle, able to sew together components to create an overall algorithm, the 'thread' used to stitch them together is quite small³. Another thought relating to efficiency, if one considers this graphical design as perhaps a new-generation graphical language, then thoughts related to the conventional textual language (C compiler) are in order; C was never designed as a language for DSP's! Perhaps a language that was designed for DSP's, albeit graphical, may have certain inherent advantages for DSP applications (such as support for fixed-point arithmetic, data flow mentality, etc.).

6.2.2. Application Size

In a similar fashion, the approach used with RIDE/VAB is such that very little additional code is produced, so code bloat is typically not noticeable in the resulting application.

6.2.3. Embedded Development Issues

Another question that comes up quickly is related to how an embedded DSP target hardware is supported. The answer is to view RIDE/VAB similar to a C compiler/Linker, which produces DSP object code mapped to a specific piece of hardware. The linker typically uses information in a CMD file to associate physical memory, etc. to the algorithm code segments; RIDE/VAB does the same basic thing allowing the designer to specify memory maps, etc. that fit the particular DSP target of interest. The graphical design is then used to produce the COFF file (DSP object file), ready for downloading with a debugger, programming into a FLASH memory, etc.

³ Profiling information for a number of algorithms/applications may be obtained from Hyperception, Inc. to address speed/efficiency questions

6.2.4. Device Independence

RIDE/VAB supports all TI ISA DSP platforms, including the C2000, C5000, and C6000. Support for C3x and C4x architectures is also available.⁴ In addition, RIDE currently also supports the Analog Devices SHARC DSPs directly. The ability to move from one DSP platform to another allows for convenient migration as DSP applications are moved to newer, more powerful, or less expensive DSPs. For TI DSP support, VAB is packaged for each ISA DSP family, similar to CCS – VAB for C2000, VAB for C5000, and VAB for C6000. RIDE software (a superset of VAB) contains drivers for all Hyperception supported types of DSPs.

⁴ Although C4x support is available in RIDE[®], current planning does not call for continued support of C4x in RIDE and VAB.

Section 7. Hyperception's Product Offerings in Graphical Design Tools

Hyperception offers quality products for many types of DSP, mathematical analysis, and systems simulation and modeling projects. These leading edge tools have been designed to provide you with a component-based graphical design environment that is ideal for accelerating DSP project schedules. Our products provide a distinct competitive advantage, and can serve as valuable development tools that can ease DSP designs -- whether an application calls for proof-of-concept work, or for rapid-production capability.

7.1. Hypersignal® Block Diagram

The Block Diagram product has been created to provide a PC-based simulation and systems modeling design environment. Support for standard sound cards allow for limited real-time applications to be implemented. Block Diagram is available in Standard, Professional, and Enterprise editions.

7.2. RIDE

RIDE component-based DSP development software provides all of the simulation and PC-based capabilities found in Block Diagram, but additionally allows for the direct graphical development of real-time DSP applications. In addition, RIDE contains drivers for all supported DSP chips and DSP boards. RIDE is available in Standard, Professional, and Enterprise editions.

7.3. VAB (Visual Application Builder)

The VAB software is a subset of the RIDE product and provides direct DSP support for Texas Instruments C2000, C3x, C5000, and C6000 DSPs only. VAB does not support PC-based simulation directly.

7.4. ImageDSP Image Processing Software

The ImageDSP component-based image processing software provides a development environment ideally suited for PC-based image processing applications.

Section 8. Summary

A graphical oriented design methodology, such as RIDE/VAB, 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 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.

8.1. Future DSP Development

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 minimizing 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.

8.2. Contact Information

For more information, including product datasheets, technical application notes, or information related to Real-time DSP for Education, contact Hyperception, Inc. at:

E-mail: sales@kanecomputing.com

Web: www.kanecomputing.com

Tel: +44(0)1606 351006

Fax: +44(0)1606 351007/8