

Software for DSP Comes of Age

The use of programmable Digital Signal Processors (DSPs) is central to many of today's hot application areas. The market forces operating in sectors such as telecommunications mean that the window of opportunity may only be open for a short time, and delays in software development can threaten viability of an entire project. To compound the problem, DSP's are becoming more sophisticated and powerful with each new device announcement, a factor that fuels increasing complexity in the application software and thus increases risk. In circumstances like this it is crucial that the software developer has the best possible tools to help in the application development task. This article discusses the range of development software available to the DSP developer and covers some of the factors influencing its evolution.

As a supplier of DSP based hardware for nearly 20 years Blue Wave Systems (formerly Loughborough Sound Images – producers of the first ever DSP plug in board for PC's at the start of the 1980's) has been in the ideal position to monitor and drive the development of both hardware and software for the DSP developer. Until recently software to aid in the development of DSP applications was the poor relation when compared to that available for various other processor types. The desktop developer has had a range of more or less sophisticated Integrated Development Environment tools available for many years and GNU, XRay and more recent commercial offerings like Tornado from Wind River Systems have ensured the 'mainstream' (CISC/RISC) embedded developer has come to fare equally well. However, until recently the DSP developer was stuck in the world of simple command line driven development tools which had little or no integration with the somewhat clunky debug tools available to support the devices. A rash of new product releases, company take-overs and new entries to the DSP software tools market have combined to address this situation, and in a short space of time development tools for DSP have leapt from the standards of the early 1980's to tools fit for the new millennium.

In the early days of programmable DSP the focus was on the chips. Interest was all about MIPS and MFLOPS, architectures and peripheral interfaces. While the capabilities of the hardware still matter a great deal, increasingly it is the software that is proving to be the key to the success of a DSP device. Ease of development is crucial. The device may be what the application will run upon, but the application itself is where the customer will expend most effort, and it is the tools which the engineer actually works with which will form his opinion of the chip.

This has been recognised right up to the top levels of the DSP industry as evidenced by the often quoted remarks about the crucial role to be played by development software made by Texas Instruments CEO Tom Engibous. This high level interest has manifested itself in the flurry of takeover activity in the area over the last year or two. Texas Instruments (TI) have begun to bring much more of the software development capability under the corporate umbrella. Their pur-

chase of Tartan (C & C++ tools), Spectron (DSP RTOS & tools) and GO DSP (DSP IDE) to add to their already well respected in house software team indicates the importance the company attaches to software in the future. Similarly Analog Devices have started to enhance their own internal capabilities through the purchase of White Mountain DSP (emulators and debug technology) and Edinburgh Portable Compilers (C & C++ tools), companies with enviable reputations in their respective areas.

While it should not be doubted that some changes in the DSP software arena would have occurred as the natural evolution of the industry, the extent of the change would have been so dramatic without the increasing uptake of DSP as a mainstream technology. The growing acceptance of DSP has been the single greatest factor in the enhancement of the tools available to support the devices. As DSP becomes more widely used the type of developer now being asked to work on DSP processors is less likely to be an electrical engineer with a Electrical Engineering degree than a software engineer with a computer science based education. This type of user expects to work in a high level language (usually C or C++). More importantly many of them will have experience of the type of tools which are available for desktop and embedded RISC devices and will expect the same level of functionality to be available for DSP's. This is a strong force pushing the development of all types of support software for DSP, not just compilers.

There is quite a diverse selection of software which exists to make the life of the DSP application developer easier. Tools to produce an application range from block diagram based code generation applications, which take a graphical representation of the desired application and produce executable DSP code, to the more traditional compiler and assembler tool chain used to turn source code into an executable object file. The debug of applications may be approached using software simulators or debuggers working on real DSP devices (either via emulator hardware or on DSP development boards). Both of these approaches allow the developer to begin work on the application before the deployment hardware has been developed, an important point in time critical situations.

Outside the development and debug arena there are a number of other software products which should be of major interest to the DSP developer. Libraries of functions (DSP 'intrinsics' such as FFT's, filters and windowing as well as more application specific functions) are an obvious aid to the developer. Similarly, the availability of Real Time Operating System (RTOS) to run on DSP's provide a software resource which can represent a major software productivity gain in many application developments.

No matter whether an application is developed using a block diagram tool or from source code written by the engineer, in all cases the code tool-chain for the target DSP will be involved in the process. All development for code to run on DSP processors is undertaken using cross development tools which may be available for PCs, workstations or both. The cross development model means that the code generation tools which are used to create the DSP code are run on a separate host machine not on the DSP itself. Obviously the resulting DSP object code must then be loaded to the DSP processor by some means before it can be executed. The development process for DSP code is depicted in figure 1.

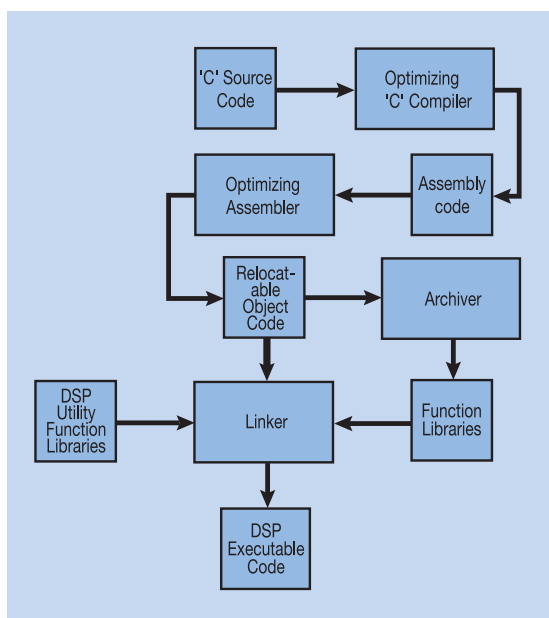


Figure 1.

The process in figure 1 shows the entry point to the development process as a C source file this has not always been the case, indeed, in the early years of programmable DSP assembler was king. High level language compilers simply did not exist. Indeed, given the type of use and users of DSP it was unlikely that a C compiler would have found a ready market. The mark of a skilful DSP programmer was the ability to squeeze every ounce of processing power out of the device through programming which made use of every obscure trick and feature the somewhat idiosyncratic DSP architecture made available.

When DSP C Compilers first became available (Loughborough Sound Images released the industry's

first C compiler for a TI DSP device back in 1983) the initial uptake was slow. Many developers saw compiled code as a recipe for application bloat, and believed the register and memory structure of early generation DSP's was not suited to compiled code. However, over time (and with the increasing power and generalisation of the DSP architectures) the idea has become more accepted to the point where today C is the development language of choice for the majority of our customers.

This move to high level language should not be surprising, DSP developers recognise all the standard arguments made for the use of high level language development, and worries about implementation inefficiencies have now been laid to rest. The chip companies suggest that almost as much work is put into the development of the compilation tool-chains which target the latest range of DSP devices as goes into the design of the device itself. Benchmarking of code tends to support the position. The latest generation of TI compilers (for the 'C6000 family) can produce code on a par with hand crafted assembler. This is in some part due to the existence of the 'assembly optimiser' a tool which takes the assembler instructions generated by the compiler and schedules them to make best use of the eight execution registers and multi-stage pipeline of the device. TI's competitor Analog Devices have also recognised the importance of the compiler to DSP developers. Their recent purchase of Edinburgh Portable Compilers provides them with a development team with a strong compiler track record.

In the 'mainstream' embedded world C is not the only choice of language open to developers, and this is also true of DSP, but to a limited extent. Ada compilers are available for some DSP's and C++ has been available in the past and could well make a re-emergence in the near future. Java has also been discussed as a strong possibility for the future but for the moment its development is mainly in C with the odd pocket of assembler.

Once code is developed for the DSP it has to be debugged and maybe profiled. While this may be done using a DSP chip simulator (a software program which executes the application code within a model of the target device) in many cases this is unacceptable. For situations where it is necessary to run the code on a real DSP and the final system hardware is not available the use of a plug in development system board becomes an attractive option. These boards will have a DSP and often add external memory and IO interfaces (a good example is the PCI/C6202-EVM from Blue Wave Systems). Through the use of such a board code may be downloaded to the board and tested using a debugger.

The debugger offers functionality to examine the internal status of the device (such as the register set and peripheral devices). In addition it will provide the ability to view memory resources in a range of formats (including as disassembled opcodes where the memory is holding code). A key feature of a debugger is the ability to offer the user controlled execution of the DSP application code through single step and breakpoints.

Of course its essential that the code being debugged is available to be viewed in its source level (eg C) format. There are a range of debuggers available for use with DSP devices from simple character based monitors which work via a serial port to full GUI based extravaganzas which pull in other elements of the development process (more on this below).

There are two main methods for the implementation of a debugger. One is based upon a monitor program running on the DSP which communicates with the debugger client and does its debug bidding. The second makes use of whatever boundary scan path the chip supports to allow the debug client to control and monitor the operation of the device. Debuggers based upon an on device monitor program have the disadvantage that they take resources (memory in particular but also interrupt vectors) from the user application. This is not a problem for scan path based debuggers which use the manufacturing test path built into the device by the chip vendor (such as JTAG) to master the device for debug purposes. This means of debug offers the minimum of intrusion while allowing extensive access to the internal state of the device.

No matter what the model adopted to implement the debugger, the task is complicated by the cross development nature of code generation for DSP devices. It must be possible for the debugger to 'attach' to the target processor through one mechanism or another. An expensive approach is through the use of some type of emulation hardware, which will plug into the host machine and connect to the device via some serial or JTAG based header. However, an emulator may not always be necessary. In some cases (as on the Blue Wave Systems range of DSP boards) the logic to enable the connection of the debugger to the device is incorporated into the DSP board hardware. This means the debug interface registers to access the JTAG scan chain of the device is available through the expansion bus of the host machine directly, and is an approach to be much preferred over external emulation – it saves both cash and expansion slots.

Up to now we have discussed the tools used to develop and debug code, however, a second type of software important to the developer is that which is 'incorporated into' the application. One of the most obvious time saving devices for the DSP code developer is the humble library function. The linker shipped along with the code generation tool-chain enables functions from such libraries to be linked in (and the archiver enables them to be created).

A library of fully tested and optimised DSP primitives callable from C can save weeks of development time (who wants to reinvent the wheel). With the right (i.e. bug free and reasonably optimised) library the use of a C compiler for code development becomes much more attractive. As the majority of time in a DSP application is spent within the function performing the DSP task e.g. an FFT the fact that the 'glue code' is compiled C is not so much of an issue. A large number of companies (e.g. Wideband) produce function libraries for various applications and DSPs. The utility of such libraries should not be underestimated.

Real Time Operating Systems made their appearance in the DSP market just after C compilers. If users were not ready for compilers on grounds of code inefficiency their horror at the idea of including an OS kernel into their applications can easily be imagined. This is not to say the OS's on offer at the time were inefficient – but no matter how efficient they were the time was not right.

This situation has now changed to a very great degree. Just as RTOS have become the norm when programming RISC based embedded systems the same is becoming true of DSP. The need to meet aggressive development deadlines and the desire to concentrate on the application not the infrastructure means the time is right for operating systems on DSP and this is reflected in increasing levels of customer interest.

A DSP RTOS is similar in most ways to RTOS's available for RISC devices. Typically they offer multi tasking based on various scheduling algorithms, inter task communication primitives such as semaphores and mailboxes, interrupt support etc. In addition to their common features various RTOSs offer features specific to one range of DSP's or another. Some provide the ability to program a network of interconnected DSP devices as a single notional device (e.g. Virtuoso from Eonic Systems) whereas some make the availability of protocol stacks such as TCP/IP a differentiating factor (MQX from Precise). However, the one unifying benefit promoted by all OS vendors is the time to market advantages provided by the use of a tested, established microkernel or scheduler. The argument for RTOS's is similar to the argument for libraries (and most RTOS's are a sophisticated library linked in with the users application code) and it is a very compelling one.

By the mid 1990's there was a healthy range of software available for DSP processors from a wide range of sources. However, while all the various debuggers, operating systems, code generation tools etc worked together to one degree or another, they were still very obviously 'separate products' and it was the responsibility of the user to fit them together. The emergence of Code Composer from GO DSP (now a wholly owned subsidiary of TI) was the first step towards the Integrated Development Environment for DSP systems. This tool (in common with others such as the Visual DSP framework from Analog Devices) brings the development and debug environment for DSP onto a par with that for many non DSP embedded devices. Such IDEs provide a single application running on a users PC or workstation where the creation of code, its compilation and debug can be managed as a unified project. Missing at the moment is the addition of RTOS awareness (i.e. the ability to monitor and debug an application in terms of the tasks and threads implemented with the OS) however, this is just around the corner, TI plan to release Code Composer Studio. This adds RTOS awareness to the IDE through an kernel instrumentation functions and real time data transfer protocol to get the resulting event data back to the host for display.

All the above mentioned software has as its focus the chip, not the system the user was to put the chip into. This 'chip-centric' approach should not be a surprise,

as by definition the chip is the one element that is common in all applications. However, the recent introduction of open interfaces to the various DSP IDEs has allowed board manufacturers such as Blue Wave to extend the tools. Adding functionality which makes the IDEs able to monitor and manipulate more than just the DSP makes the tool 'system-centric' and thus further eases the task of the developer. Initialisation of on board resources, automatic generation and configuration of peripheral access code and utility functionality such as flash memory programming are all functions which are being added to the IDE for DSP.

Boiled down to its essentials the story of DSP support software becomes the provision of tools to reduce customer time to market. Some of the most recent and interesting developments which address this task remove the need for traditional programming to one degree or another.

Tools which abstract the user from the generation of DSP application code are not a recent development, Hyperception had a block diagram based code generation tool back in the 1980's and there have been a number since then. However, the entry of companies like Lockheed Martin, Honeywell and most interestingly The MathWorks into this market sector indicates it may be about to burst into life again. These tools allow the user to connect together functional blocks using a block diagram based interface. The tool will then generate the application described by the diagram by either generating source code which is then compiled or by dynamically linking together pre compiled blocks of code. While typically used for the development of prototype systems this is not always the case. The efficiency of the code generated by this type of tool is increasing. This coupled with the increasing efficiency of the underlying compilation tools and the power of the DSP's opens the possibility of the use of code generation systems for deployable applications.

An approach to application creation which is wholly focused on the production of deployable code with minimum programming and maximum speed is that of the application framework. The framework approach offers a 'bolt together' solution based upon fully tested and validated code for various vertical markets. It absorbs a lot of the risk from DSP code development, and allows end system developers to concentrate on their application functionality rather than the implementation of 'background' code and well known algorithms.

In overview terms a framework is a software environment which runs on the DSP device providing all the infrastructure code needed to allow the execution and management of many (and often varied) algorithms in parallel. The framework provides a rigorous environment to ensure the efficient and controlled management of resources takes place. An excellent example of such a framework is FACT, the Blue Wave Systems framework for telecom applications (see figure 2).

This software allows multiple DSP devices to process multiple channels of telecom data without the need for the user to get their hands dirty with code at all. The configuration and management of the users applica-

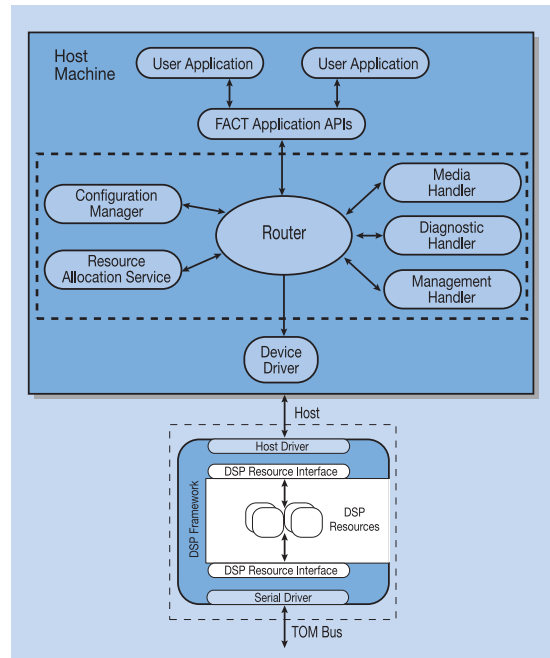


Figure 2.

tion is done from a user interface on the host machine. Algorithms can be bought from a number of providers (as long as they are written to the framework interface specification) or alternatively the user may choose to create their own compliant algorithm blocks to plug into the framework. As the framework and algorithms can only be put together in a certain number of pre-determined ways it is possible for them to be written in a fully optimised way. This ensures applications built using the framework are ready for deployment almost as soon as they are put together.

DSP support software has come a long way and its development continues apace. Clearly a major factor ensuring continued development is the increasing competition between some DSP and RISC technologies. As DSP has become more mainstream RISC has moved into areas once the preserve of DSP. The two technologies compete for a large common market, a market which is worth many millions of dollars to the company which wins the sockets. In the past RISC enjoyed a support software advantage over DSP, but as I hope the above article has indicated, this advantage has been eroded. Rest assured that a large number of engineers spread across many DSP chip vendors and associated 3rd party companies are dedicated to ensuring the gap doesn't reopen. ■

After four years as a software engineer producing real time control software for private digital exchanges, Nick joined Loughborough Sound Images (now Blue wave Systems). Over the last 11 years he has worked in various software engineering roles and during this time has seen DSP development software evolve from assembler only support to the current crop of DSP focused Integrated Development Environments. Nick is now working as a Product Manager within the Blue Wave Systems marketing group with particular responsibility for software based products.