Heterogeneous DSP: Easier than you think

Ray Hardison
Software Engineering Manager
Ixthos Inc.

INTRODUCTION

There's a lot of neat technology out there! No sooner than you buy what you need to put together one design does another manufacturer come out with a new whiz-bang-chip that blows away the competition and sets the bar higher. The problem is, you have a significant amount of time invested with one product line and the thought of having to spin up on a different architecture and development tool set makes things appear more difficult than they are worth. It would really be nice to get one or two of those new features, especially since your current solution doesn't do that part of the job well. But in some ways it still solves certain parts of the problem better than that emerging star.

Wouldn't it be great to reap the best each technology has to offer? To mix and match solutions without have to start all over again with each piece. To apply a fixed-point processor to a fixed-point section of a problem and feed the results to the floating point machine where that type of computation is needed. Problem is; how do you make all these dissimilar pieces play together, without spending all of your time and money on getting them to talk to each other?

This paper presents the concept and implementation strategies for developing heterogeneous Digital Signal Processing, hDSP, systems. Rather than focus on one specific vendor's solution, we will look at general industry trends and evolutions in hardware and software that further promote the combining of dissimilar technologies.

WHAT IS A hDSP SYSTEM?

hDSP is not a new concept, as a matter of fact some of the earliest system implementations involved connecting a DSP to a general purpose processor, GPP. Figure 1 depicts such an architecture where a DSP is connected to the GPP via a bus. The GPP has data storage and display devices to offer to the application, and the DSP has I/O interfaces. Where, in these type of system configurations, the GPP will typically load the program from disk to the DSP and start it executing, what makes it a hDSP configuration is the actual involvement of the GPP in the processing of the data stream.

This is a qualifier of a hDSP configuration. The dissimilar elements, in this case GPP and DSP, must participate in processing the data in unison. If the GPP just loaded,

started, and performed some basic command and control, it would be acting as a server to a client. In hDSP the GPP would be part of the data stream, processing the data.

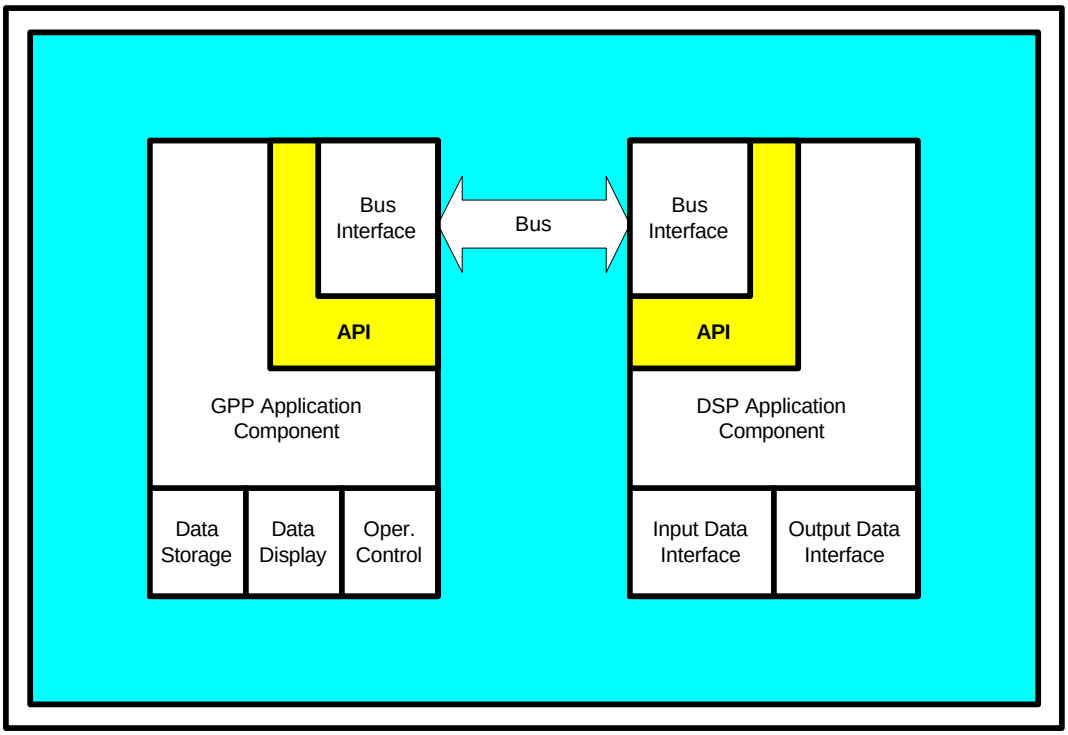

Figure 1. Simple hDSP configuration. One GPP, one DSP, and a common API.

What the GPP would bring to the application would primarily be a function of the direction of data flow. If the DSP is collecting data from some I/O, such as an analog to digital converter, filtering, transforming, integrating, and shipping it off to the GPP. The GPP could be collecting the processed results, performing graphical display, reducing, or providing storage. In the reverse situation the GPP could be providing information to the DSP that would be converted and sent to a digital to analog converter for output. Potentially, the DSP could be used as a coprocessor to the GPP where the data would be exchanged bi-directionally.

This example also demonstrates the benefit of hDSP; each dissimilar element contributes to the process in a unique way, bringing its strengths to bear on the problem. The GPP contributes its data display, storage, and operator interface attributes while the DSP contributes its deterministic, real-time I/O handling, fast computational capabilities. This concept can be expanded to using fixed-point DSPs, floating-point DSPs, CISC and RISC processors and even mixing like products from different manufactures. We will look more at the potential combinations as we proceed, but first we need to look at the basics of how these dissimilar pieces communicate.

Figure 1 also depicts an Application Program Interface, API, as a layer between the DSP and GPP application components and the bus interfaces. This software's purpose is to facilitate the communications between the two applications removing the application from the responsibility to directly manipulate the bus interface. A good API provides software constructs like queues, mailboxes, the capability for the applications to check for and wait on data, plus the capability to multi-buffer and be notified of completed writes. The GPP's API and the DSP's API must be compatible and use the same constructs and conventions in a like manner. Differences in the raw data formats exchanged between the two should be concealed from the two applications with the sending application's intended quantities correctly received by the receiving application.

In early hDSP configurations these APIs were minimal to non-existent causing the system developer to implement their own. As DSP products evolved DSP board manufactures started to provide APIs targeted at using their boards with specific GPPs on specific buses. If you wanted a off the shelf solution, i.e. you didn't want to design an embedded hDSP solution, the approach was as follows:

1. Figure out the DSP performance requirements for your problem.
2. Select a DSP.
3. Select a board vendor.
4. Find out what GPP environments the vendor provides support for.
5. Select the GPP

This selection sequence results in the DSP/board vendor decision determining the GPP. Since DSP selection criteria can be so much more performance critical than GPPs, this strategy has been a natural trend. The brass ring to be obtained is of course good connectivity between the two. Otherwise you just as well develop your own API and use the GPP you want. But what if you want to use dissimilar DSPs and a GPP together in a system? How does the basic configuration of Figure 1 expand? What issues now become prominent in the decision process?

MULTI-BOARD hDSP

A DSP board plugged into a GPP's system bus may be all you need, especially since many DSP boards contain a number of DSPs. The vendors of these multi-DSP boards normally provide the on board hardware connections along with inter-processor communication software libraries. If the vendor has a good product, the API for inter-processor communication would extend to the GPP API. This would permit the GPP to exchange data with any of the DSPs and the DSPs to use the same protocol to inter-exchange data on the board.

If your application needs more DSPs than you can obtain on one board, you can add more boards to the system bus as depicted in Figure 2. The GPP's API will need to

support multi-board, multi-processor, addressing and the DSPs' API will likely need to have some type of awareness as to how they rank in the system configuration with respect to other DSP boards. An easy way to handle this is to have the GPP tell each DSP what system processor ID it is upon program load into the DSP. System processor IDs will help the DSP distinguish on-board inter-processor communication with off-board inter-processor communication.

In multi-board configurations it is preferable to remove the inter-processor communication across boards off of the system bus. This keeps the system bus available for the GPP to interact with the DSPs without getting into an arbitration battle for the bus. Likewise it provides parallel data movement paths so that DSPs can inter-communicate while system bus transfers are occurring. Figure 2 depicts two inter-board connections. One occupies the I/O space of the board and the other is a designed-in intercommunication channel. Both have common APIs but the I/O space connections may not be part of the standard API used for the designed-in and GPP communications.

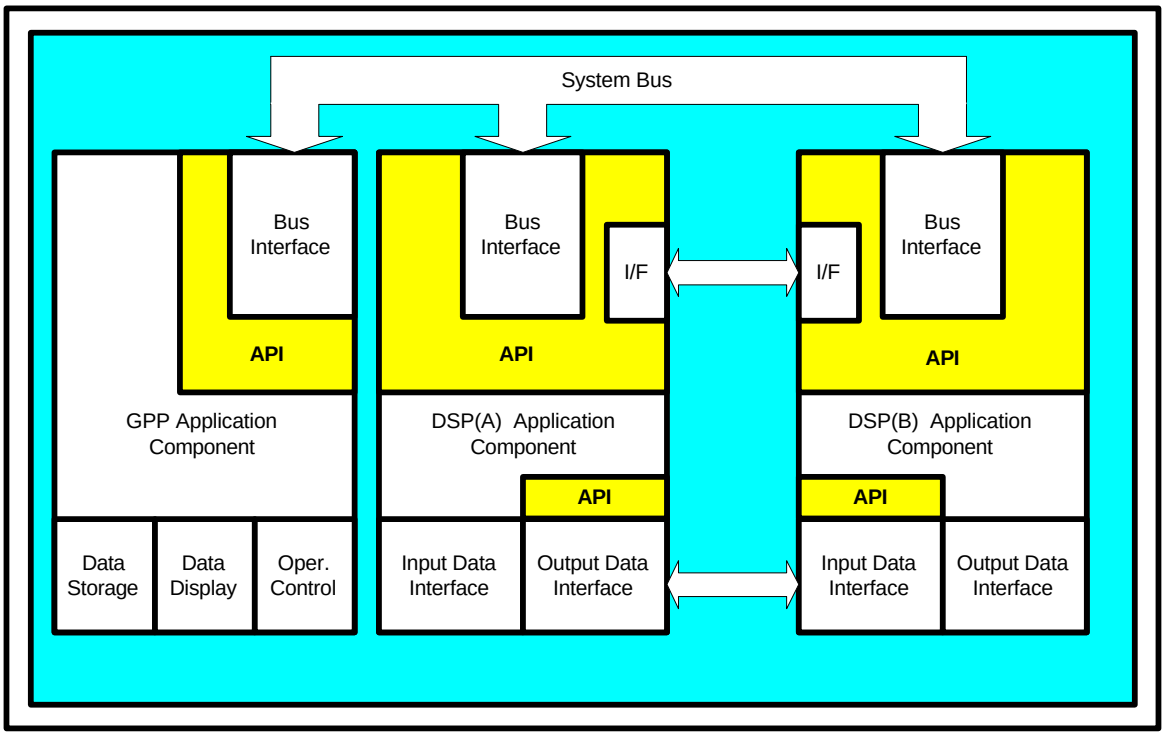

Figure 2. Multi-board hDSP system. DSP(A) and DSP(B) can be similar or different.

Off board interconnection schemes vary with board manufactures and DSPs. DSPs that have high speed ports and internal I/O Processors are normally connected using these ports in a point-to-point connection. Sophisticated API's may provide many modes of operations for these types of links such as through-routing, broadcasting, bi-directional and

uni-directional transfer modes. If the DSP's don't provide any sort of intercommunication links, then it's up to the board manufacture to come up with some sort of solution. Some of these solutions have evolved into multi-processing interconnects that vendors center all their product offerings around. When these vendors expanded their DSP offerings to several different types, they extended their interconnection solution into a heterogeneous configuration. This would be the case of Figure 2 where one DSP board is of type A and the other is of type B. It becomes the responsibility of the hardware and software to make the interconnection seamless.

This type of multi-board hDSP solution has been available from some of the larger DSP vendors for some time. These vendors each promote the relative advantages of their product offerings. The selection process for configuring this type of hDSP solution becomes as follows:

1. Figure out the DSP performance requirements for your problem.
2. Select the DSPs.
3. Find a board vendor that supports the DSPs.
4. Verify interconnects and software can satisfy requirements.
5. Find out what GPP environments the vendor provides support for.
6. Select the GPP

Of course if you were very diverse in your DSP selection, you would find it very difficult to locate a vendor. In the past there were only a few companies that had any kind of heterogeneous offerings and the DSPs they offered were only a few well-known types. However, the current market place is exhibiting more diversification in product offerings. DSP board manufactures are expanding their product offerings from what was originally a speciality house approach to a broad market approach. This is resulting in some interesting new architectural developments that put hDSP capabilities at the board level.

BOARD LEVEL hDSP

Mixing processors types on a board basis, where each type resides on one board, can result in big and costly system solutions. Thanks to some new innovations this may not be the case anymore. Board manufactures are starting to embrace common mezzanines, standard local buses, and heterogeneous on-board architectures. Figure 3 shows a generic example of one such board implementation. It is based on the Common Heterogeneous Architecture for Multi-Processing, CHAMP, developed by Ixthos. This board architecture has a PCI local bus, two PMC mezzanines, DSPs, and a GPP. The GPP in this case is a RISC processor that performs a number of board functions and manages the system bus, which is a VMEbus. Bridge interfaces are used to isolate sections of the local bus so that local traffic stays contained, yet the whole bus is accessible from any processor.

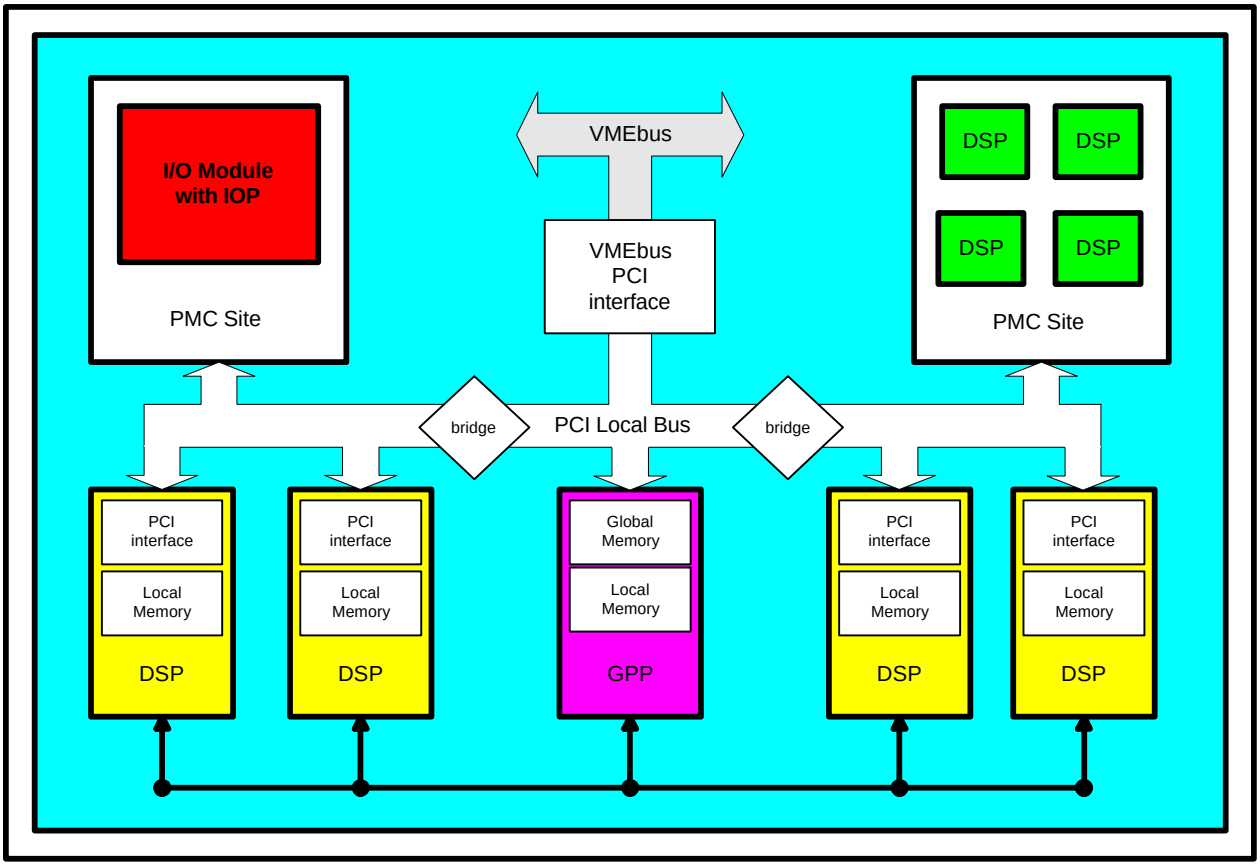

Figure 3. Board level hDSP configuration using a generic architecture VMEbus board.

Each DSP contains a PCI interface that can automatically convert between big-endian and little-endian formats to keep the data in a bus-native format. This interface can also provide smart functions such as DMA transfers with strides into and out of DSP memory space. Performing these manipulations "on-the-fly" and in the background can speed up many data manipulation algorithms commonly performed in DSP applications.

The PMC sites can host I/O cards or DSP cards, so you can very easily plug a DSP card with type (A) processors onto a baseboard having type (B) processors, with all having a local bus connection to a RISC processor. The figure also depicts a smart I/O PMC card with its own processor (IOP), giving the configuration shown 4 distinct processor types, connected on one, in this case, 6u VME board.

PMC sites are showing up on many RISC and CISC based VME cards. With the introduction of PMC cards that contain DSPs and dual PMC sites on these cards, it is possible to configured a hDSP system on one of these boards that has two distinct types. With a DSP PMC that has serial ports as an external interface, a parallel data connection could be established to other cards on the VMEbus that have the same ports. The

combinations of interconnections and components can get quite interesting, but can they really communicate with each other?

SO WHAT ARE THE PROBLEMS?

It is an age-old story, you can hook together all the hardware, but getting it to work together is totally dependent on the software. If you have the time, and good documentation, you can do it all yourself, but few of us have the interest, motivation, or budget to put together a software solution for a hardware puzzle. What we want is the capability to focus on our application and have a consistent and reliable method for data to be exchanged among processors. This is where a common high-level API can serve us well. But is there such a thing as a common API? Does it extend beyond one vendor's product offerings?

An API that has the potential to position itself to be a common standard for hDSP has been slowly emerging. This API is referred to as the Message Passing Interface, or MPI [1]. A consortium of industry, academia, and government agencies defined MPI, and several vendors have announced product support. Interest in it appears to be growing in the industry, and numerous extensions to the specification are starting to surface to address various implementation issues that have arose from the first attempts to apply the standard in the market place.

MPI is a definition of high-level function calls that provide inter-process/processor communication abstraction. Its definition supports:

- Point-to-point communication
- Collective operations
- Process groups
- Communication domains
- Process topologies
- Environmental management and inquiry
- Performance profiling support

MPI's success or failure will be determined by how useful it is found to be. The hope is that it can be a "silver bullet" that will permit an application to be easily moved from architecture to architecture, keeping software cost down as hardware evolves. Its ability to ease hDSP implementations is really only a side effect of its intended purpose. Its ability to serve in that role will rely totally on the desire, and demand, for manufacturers to provide the lower level software connectivity required permitting the MPI calls to function across architectures.

One thing that may work in the favor of MPI and hDSP is the continuing trend in the industry of product diversification and merging. Very few board manufactures make

just one type of board anymore. Since a company can have several types of product offerings, it behoves them to provide inter-operability among their products. Finding vendors that support MPI and hDSP connectivity in their product line is becoming more common. Yet it really all comes down to demand, connectivity will only be put in place if people want it, and a big factor will be the capability and availability of software development tools for such an environment.

One of the biggest headaches in a hDSP implementation can be dealing with several different code development environments at once. Maybe you have the luck to have group of developers that can be partitioned to the effort in much the same way as the components are partitioned to the design. But even given such a case, there are normally as many issues as there are tools. Given what you are trying to accomplish with a hDSP solution, a high-level solution for the development software makes sense. Let the development environment manage the different compilers, libraries, and syntaxes.

Several vendors now offer DSP development tools that support partitioning and redistribution amongst processor types. The types supported are somewhat limited at present but appear to be expanding. These tools typically generate and construct the processor code therefore automating the usage of the processors development tools. They also provide performance-monitoring capabilities that are essential to large-scale multi-processing development. A common API like MPI can bring hDSP and these tools into a harmonious mix. After all, developers of these tools are just building hDSP applications on which our applications get layered.

But does not all this layering of software cause a performance hit? Wouldn't hDSP be faster and leaner if all the interfaces were implemented directly? Absolutely, but it comes down to what's cheaper and has a shorter development cycle. Man-hours tend to be a scarce commodity where hardware constantly gets faster and cheaper and development tools continue to improve. Attention to the performance impacts is also a major consideration for the API and tools developers, they may have impacts, but they are also most aware of those impacts and striving to reduce them.

So what is the down side of hDSP? It's basically that the pieces are coming together but they are not quite there yet. You cannot select a DSP board from one manufacturer, and another type of DSP board from a different manufacturer and be confident they will work together in a seamless manner. You may be able to accomplish a similar configuration if you stick with one manufacturer's products, but you will be compromising your selection criteria to the product line. Unified hDSP development environments are still very new and have limited product support. And last, but certainly not least, the whole concept of hDSP may not be worth the effort, the compromises of a one-processor type solution may be less painful than the compromises a hDSP configuration would impose.

WHAT ARE THE BENEFITS?

So if hDSP is becoming more obtainable, why would you want to use it? Are there benefits to offset the additional aggravation? Or would it be a better bet to look for a processor that can do it all?

The variations available alone can overwhelm an engineer. Not only are there a large number of processor types available; there are also different operating systems each which emphasize certain aspects of a processor's performance. A single DSP product can be ordered as a low-cost reduced feature version of the high cost full product. Some products can have many different sub-types. All these variants exist because of their ability to solve a certain application problem better than the rest. If there were a one-chip solution, variants wouldn't exist.

To simplify and summarize the benefits, a few key feature groups tend to stand out. They are:

1.) Floating point DSPs. High performance 32 bit precision designs that have high benchmarks for multiply and accumulate computations. They typically have very deterministic performance, minimal overhead operating systems, and sufficient cache and bus speed to permit data flows commensurate with the processors computational bandwidth.

2.) Fixed point DSPs. Can be 32 bit or 16 bit. 16 bit are used in many low cost embedded cases. 16 bit fixed point is very well matched to digital/analog and analog/digital converters. 32 bit versions are normally related to a floating point product offering.

3.) MP DSPs. Features are added to the DSP to facilitate inter-processor communication. Typical features would be high-speed ports, shared memory support, and integrated I/O processor with DMA.

4.) RISC/CISC. Supports more elaborate cache and memory management along with more extensive operating systems. Some provide 64 bit precision floating point computation. Can support peripheral devices like SCSI controllers, network chips, etc. through existing drivers. Some of the higher performance RISC architectures have impressive DSP performance capabilities.

DSPs can bridge several of the above. The SHARC™ from Analog Devices, Inc. performs 32 bit floating point and 32 bit fixed point operations at full performance. In addition the processor has some of the best MP support available in DSP [2]. Newer DSP products contain features normally associated with RISC/CISC processors such as very long instruction words, deep pipelines, and multi-level caches. It's apparent from recent

product announcements that a merging of processor topologies is occurring. So is a super-chip around the corner? Is all this dissimilar technology merging together into one?

hDSP ON A CHIP?

Any discussion on hDSP should address the new developments of processor manufacturers merging DSP type cores into RISC processors resulting in a tightly coupled co-processor arrangement. These processors are interesting as to what they add to the processor in capability and how they get programmed. There are many variants in the implementations, some sharing the external bus, some having independent ones. Obviously, the manufacturers' feel that hDSP benefits warrant an integrated solution making a stronger processor option. The goal in some cases appears to be reaching for a super-processor that can satisfy all requirements becoming a one-processor solution for all. However all these manufacturers have done is create another type to consider in the heterogeneous mixing bowl. No one design does it all as of today.

CONCLUSIONS

hDSP as a system design strategy is an approach of trade-offs. Benefits can only be realized if the connectivity can be purchased or developed. The evolution of board designs and diversification of product lines is helping to make hardware interconnection easily obtainable while software interconnection still remains in the domain of the individual manufactures. More manufactures are offering hDSP designs and declaring support for MPI. High level DSP programming tools are positioned to utilize these developments to obtained true hDSP development potential across different manufacturers' products. Whether hDSP becomes a more exploited design strategy in the future, or disappears will mostly depend on the demand for it in the marketplace. Customers will have to want it and request it.

REFERENCES

[1] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra, MPI: The Complete Reference, The MIT Press, 1996

[2] ADSP-2106x SHARC User's Manual, Analog Devices, Inc. 1995