

Presentation Title: SHARC 2146x Processor Overview

Presenter Name: Ramdas Chary

Chapter 1: Introduction

Hi everyone my name is Ramdas Chary and I am a DSP Applications Engineer with Analog Devices. I'd like to welcome you today and thank you for joining me as we talk about the SHARC® 2146X processor.

The outline on this presentation will be organized as follows; we'll be talking about the SHARC Processor roadmap and the markets that it's going to be targeting. Next we're going to talk a little bit about the SHARC 2146X block diagram and the memory structure and the memory map. Once we've done this we'll move on to describing some of the new features that have been introduced on the SHARC 2146X. This includes support for the variable instruction size execution, the DDR2 controller, and the interface. The dedicated hardware accelerators, the thermal diode and the media local bus which will be offered for automotive products in this processor family.

We'll be talking a little bit about the high speed point to point network communication in the form of the link ports. And finally we'll talk about some power management and clocking options on the 2146X.

Once we've talked about the new features on the 2146X we'll spend a couple of slides talking about some of the enhancements. These include a couple of instructions that have been enhanced. We'll be talking about some changes and enhancements to the peripherals, and the direct memory access engine.

Finally we'll conclude by pointing out some valuable information and resources that you may be able to access online.

Chapter 2: 2146x in the SHARC Roadmap and Markets

Moving on let's start with the 2146X and the SHARC Processor Roadmap and markets. Over the last several years our highly successful 32-bit SHARC processors have targeted many different markets ranging all the way from industrial instrumentation and automotive segments to audio including pro and consumer audio as well as automotive markets. We've done this by providing a combination and many flavors of processors. We've included high end performance and system level solutions that have combined a lot of MIPS and computational power with a lot of on-chip memory and product specific and market specific features.

We've also tried to include or address cost sensitive and power sensitive applications by providing variants and flavors of our SHARCs which have lower power consumption by a combination of lesser amount of on-chip memory and a reduced requirement of MIPS.

The SHARC 2146X is offered as a system level solution, it has a large amount of on-chip memory and it has the highest amount of gigaflops to date. We also have introduced dedicated hardware accelerators to offset some of the most popular and commonly used algorithms such as the , FFT the FIR, and the IIR algorithms.

We've also tried to include a diverse set and a collection of modules and peripherals which we think will be very helpful in these market segments that we're going after.

Now let's look at some of the SHARC 2146X features that are targeted for some of the specific markets that we're going after. If you look at digital home we have introduced link ports which provide for high speed point to point communication. We have dedicated hardware accelerators, we also have a large amount of on-chip memory, which we believe will help a user in achieving higher channel density and higher performance.

In terms of the pro audio market in addition to the on-chip memory that we talked about, we also have the S/PDIF interface which we think we will very helpful for customers in achieving higher channel densities and more software processing in terms of special effects and pre-and post processing.

In terms of the industrial and instrumentation markets again we have newly introduced the DDR2 DRAM controller and interface and a thermal diode and we have continued on with the

precision clock generators or PCG's which we feel will give us an advantage in this market segment.

Last but not least in the automotive markets we have introduced a media local bus which we'll be talking about on an upcoming slide, as well as a thermal diode which allows a customer to accurately monitor and act on the die temperature of the part.

We also have continued the S/PDIF interface and we also have dedicated on-chip decoders in ROM which will help automotive customers with their software processing.

Now so far we've talked about the SHARC Processor roadmaps and some of the features that we think will help us in these market segments. Now over the next couple of slides I just want to describe these features in detail and just go over them and list them at one place.

Chapter 3: SHARC 2146x Features

On this slide let's look at the SHARC 2146X at a glance. It's the newest member in the SHARC 32-bit floating-point processor family and it has the largest amount of on-chip SRAM of all SHARC processors to date in the form of 5 megabits of on-chip memory.

In addition the SHARC 2146X is also the first processor in the SHARC family in many ways. We have newly added support for what we call variable instruction size execution or VISA which has an advantage of reducing user code sizes by approximately 20 to 30%. It's also the first processor in the SHARC family to be manufactured in the 65 nanometer process and has the highest core clock frequency of all SHARC processors to date.

We've also introduced as we talked about before, a 16-Bit wide DDR2 DRAM controller and interface which runs at a clock rate of up to half the core processor frequency. And consequently at data rates of up to the core clock frequency.

We've introduced the media local bus or the mediaLB support for products in this family dedicated for automotive markets. We've introduced 2 byte wide link ports which provide a high speed point to point communication between two SHARCS or between a SHARC and a host FPGA processor.

And last but not least we've included dedicated hardware accelerators for some of the most popular and commonly used algorithms such as the [FFT, FIR, and the IIR] with an intention of freeing up the core MIPS for customer specific algorithms, or proprietary functions or control functions.

We've also introduced on die thermal [diode] which allows a user to accurately measure and monitor the die temperature. This is very important in industrial instrumentation applications as well as for automotive customers.

For our current SHARC users we've stayed backward source compatible with the previous SHARC processor families including the 2136X and the 2137X. We've also enhanced a couple of new instructions which are primarily intended to help our our compiler and code generation tools to optimize and generate better code.

We've added a nifty little feature which we think will be really popular with customers which allows you to directly DMA data to and from any of our serial ports to the external memory without having to move the data into internal memory. This is true with both the DDR2 as well as asynchronous memory, and we believe that this will be a useful feature for our customers.

Finally with the intention of keeping power consumption at optimum levels we give you the ability to turn off clocks to modules and peripherals that are unused

Chapter 4: SHARC 2146x Block Diagram

Now let's talk about the SHARC architecture and the 2146X block diagram in a little more detail. The SHARC 2146X can be thought of as being laid out as four main blocks. Here on the top left hand corner is the SHARC core processor which has been carried forward from our earlier generations with a few enhancements to the program sequencer and the instruction cache as well as a few changes to achieve higher clock speeds. But from a user standpoint it has basically stayed unchanged.

Next we have the internal memory block which is laid out in the form of four blocks which we'll talk about on the upcoming slides.

Here to the right of the memory block is the external port block, which comprises the asynchronous memory interface or the AMI and the DDR2 DRAM controller which is new to the 2146X. The AMI is 8-bits wide in data and has 24 address lines associated with it. The address and data lines on the AMI can be used as flags and PWM outputs if they're not used as part of the AMI.

The DDR2 DRAM controller is new on the 2146X and is a 16-bit wide controller, here on the data bus its 16-bit wide and it has 19 dedicated address lines. Now the pins and the pads of the DDR2 controller are high speed I/Os and dedicated solely for the DDR2 controller and not multiplexed with any other pins.

Here laid out all across the bottom is the I/O processor, which is a very important and key part of the 2146X. It comprises mainly the digital applications interface, or the DAI and the digital peripheral interface, or the DPI. And between the two of them, the DAI and the DPI encompass a lot of the peripherals and modules that have existed and continue to exist in the 2146X.

I'm not going to be talking about the DAI and DPI in detail in this presentation, but if you'd like further information on these units I encourage you to refer to the SHARC architecture overview presentation which is also available on line.

Also available as part of the I/O block is the DMA arbiter and the DMA engines associated with the peripherals. And here we have the FIR, FFT and the IIR accelerator blocks along with the mediaLB port and the Byte wide link ports.

There are actually three sets of buses that run all across the processor. These are the program memory or PM address and data buses, the DM or the data memory address and data buses and the I/O address and data buses which feed the DMA engine.

Chapter 5 SHARC 2146x Internal Memory Organization

Now let's talk a little bit about the internal memory organization of the SHARC 2146X. On the previous block diagram we observed that it has the largest amount of on-chip memory in the form of five megabits of SRAM. The five megabits of internal SRAM are organized in the form of four internal blocks. Two of these blocks, blocks zero and block one are one and a half

megabits in size. And blocks two and three are one megabits in size. The main advantage of organizing this memory in the form of these blocks instead of having them in one single block is that it allows for increased band width and higher performance.

What this means is that the processor core could be accessing data or instructions in one of the memory blocks, and the DMA engine or the I/O processor could be simultaneously accessing any of the different blocks without any overhead or any penalties.

In addition to the five megabits of internal SRAM we also have four megabits of metal masked programmed ROM. This ROM will be used to store audio decoder libraries and other commonly used algorithms for automotive and consumer product offerings. But there is also the option for users to have their custom libraries be metal masked programmed on to the ROM during the manufacturing process. Certain minimum requirements have to be met in terms of quantities and NRE but if you'd like further information I encourage you to contact your local ADI sales rep.

The internal memory organization of the 2146X also now provides support for variable instruction size code execution. This is in addition to the fixed instruction size execution that existed before. And the primary advantage of this is that it allows user functions or user code to be 20 to 30% smaller in terms of the size, and an increased efficiency in the usage of the internal memory.

The performance of code whether it's by variable instructions or by the traditional instructions sizes will be unchanged as we'll talk about on the upcoming slides.

Now let's take a look at the four internal blocks within the SHARC 2146X.

As we talked about before we have block zero and block one which are 1.5 megabits in size, and blocks two and three which are one megabits in size. You can think of each of these memory blocks as connecting to a cross bar switch. And the other side of this cross bar switch is fed by all the three internal buses that span the processor. This includes the program memory or the PM address and data buses, the data memory, or the DM address and data buses, and the I/O memory or the I/O address and data buses.

At any cycle or any point in time all three of these buses can actually access three different blocks in the same cycle. So for example in same instruction, or in the same cycle, I could be performing an access over the program memory or the PM bus to block zero, and at the same time I could be performing an access over the DM bus to blocks one. And I could have a DMA happening in the background which is moving data either to or from either of these two blocks, blocks two and block three.

This is not to say that a user cannot perform multiple accesses over multiple buses to a single block. There are hardware interlocks in place which allow for an arbitration scheme and the accesses will happen it'll just take a few cycles. But laying out the system and the instructions and the data and the DMA in this way in different blocks actually allows for the highest possible performance and the highest possible bandwidth.

Now let's look at the internal memory map of the SHARC 2146X. Without going into too much detail on the slide which is actually available in the data sheet, I'd just like to point out that as we saw in the previous slide, any of these internal memory blocks, whether they be blocks zero, block one, block two, or block three can be configured to store any combination of long words, instructions, normal 32-bit words or short words. For example block zero which we said contains one and a half megabits of actual memory can be configured as either 24K of long words, or it could store 32K of 48-bit instructions, or 48K of normal 32-bit word data, or 96K of 16-bit short words. But this does not mean that they have to be dedicated or each block has to be dedicated for just one type of word. Any block could contain any combination of long words, instructions, normal words, and short words. For example if I so wanted I could choose to have 16K of instructions in block zero RAM followed by 24K of 32-bit normal words. And if I had a lot of code or if my application was larger than 32K I could actually store the functions in block zero RAM and in block one RAM and I could use the remainder of block one RAM to store data. And this is true of any of the four memory blocks.

And the flexibility that this gives us in terms of accessing and addressing will actually be very useful for us in supporting variable instruction sizes which we'll talk about on the next slide.

The way we supported variable instruction size execution is by two main techniques, the first is by the addition of new 16 and 32-bit - what we called compressed opcodes - for the most commonly used and most popular SHARC instructions.

We've also tried to remove redundant or unused bits from the original 48-bit opcodes and the 48-bit instructions that used to exist on SHARC.

I'd like to stress also the fact that this is a completely optional feature and that if a customer feels that their code size fits comfortably within the internal memory of the 2146X and that they don't really need to use code compression they can chose to not use it. All the traditional instructions and op codes that used to exist on the previous SHARC processors continue to be functional and will work exactly as they did before.

This can be thought of as I said as optimization for size rather than any kind of trade off between performance and size or having any kind of degradation in terms of performance. So what exactly does a user have to do in order to enable these variable instruction size support? In a nutshell, not a whole lot because the code generation tools actually do the work for you. When a user invokes the code generation tools and using an optional switch the C-compiler and the assembler along with our enhanced linker automatically replace the 48-bit opcodes for the most commonly used instructions with the 16-bit and 32-bit equivalents. Thirty-two bit opcodes are stored as two 16-bit values.

The code generation tools also eliminate unused and/or redundant bits within the initial 48-bit instruction. So for example if there was a part of an instruction that was not being used then those bits that correspond to the unused functionality would be removed by the code-generation tools. But it's completely transparent from the user standpoint.

I'd also like to stress the fact that the final DXE file or the executable generated by the tools could be a combination of instructions which are uncompressed as well as compressed. So a customer does not need to either choose compression or uncompressed code for their entire project. They could have some modules or some functions which they choose to be uncompressed, and then they could have some functions or modules that are compressed. And the DXE can work seamlessly and can be a combination of both of these instruction types.

The code generation tools do their part and in terms of silicon, the program sequencer and the instruction decoder also have been enhanced to be able to fetch instructions either from short word space, which would be your compressed opcodes, or from the legacy 48-bit space for the 48-bit legacy instructions. And they can build them and execute them and the customer or the user has to do absolutely nothing. So no user modes, or no bits, nothing really needs to be done from a user standpoint in order to determine whether the sequencer is executing uncompressed or compressed opcodes.

Just the address that is being generated by the sequencer alone determines whether the sequencer will fetch compressed opcodes or uncompressed opcodes.

I'd also like to stress the fact as I pointed out on the previous slide, that there is no overhead in terms of performance, or no degradation with executing these instructions or compressed opcodes. So in terms of performance it should be the same, so this can be thought of as an optimization for size.

We actually ran the code-generation tools on some of the sample library functions that we had including some basic math functions, the jpeg algorithm, and what we call the Susan image processing algorithm. And even though user functions or user modules and user projects may achieve different compression depending on the individual instructions and the options or the possibilities of compression, we actually notice between a 20 and 30% improvement or reduction in the code size. Remember that the performance stays the same. So the number of MIPS or the number of cycles needed to execute these functions will be the same regardless of whether they are compressed or uncompressed, it's just a reduction in the code size.

Chapter 6 SHARC 2146x DDR2 DRAM Interface

Now let's talk about the 2146X DDR2 DRAM controller and interface. The SHARC 2146X has a 16-bit wide DDR2 controller and a DDR2 memory interface. It runs at clock rates of up to half the processor core clock frequency and consequently supports data rates of up to the core clock frequency. The presence of the DDR2 controller gives us a high speed and a high performance external memory interface in addition to the on-chip memory that's already available. So it actually nicely offsets and complements the on-chip memory that we already have. In terms of

DDR2 devices that are out there we support all the standard vendors. We support the most popular sizes out there, all the way up to two gigabits in size. So we support devices of two fifty-six megabits, five-twelve megabits, one gigabits, and two gigabit devices out there.

In addition to running at half the core clock frequency the DDR2 controller also gives the flexibility of running at multiple core clock to DDR2 clock ratios. We also support eight bit as well as 16-bit wide devices. Note that the data bus width of the DDR2 controller is 16-bits wide so we can support this as either two 8-bit devices or one by-16 device on the bus.

We also support all the popular page sizes so we support 512 words, 1K, 2K, and 4K pages, and we also support multi page operation, which is important from a performance standpoint.

In order to also increase bandwidth we support burst lengths of up to four words.

And last but not least there is a lot of DDR2 memory out there, so there is a total of up to 254 mega words of DDR2 memory space that is available and the devices actually have to be laid out in the form of four separate memory banks.

All of these features that we talked about can actually be programmed very easily by means of memory map registers that can be programmed by the core.

The DDR2 DRAM controller also supports automatic packing of data between these external 16-bit bus and either 48-bit instructions or 32-bit instructions. For those of you familiar with the SHARC architecture and the SIMD mode, or the Single Instruction Multiple Data mode we now support dual accesses in the same instruction from the external DDR2 memory. This was something that did not exist before, but has been enhanced to include on the 2146X.

We also support direct code execution from the DDR2 memory for both uncompressed as well as compressed instructions.

Chapter 7 SHARC 2146x Hardware Filter Accelerators

Now let's talk a little bit about the 2146X dedicated hardware accelerators. The purpose behind having these accelerators is not to insinuate that the core incapable of performing these common algorithms, but rather to offload some of these functions to the hardware units so that the more

valuable core MIPS can be utilized for proprietary algorithms or customer specific functions or control code.

There are a total of three dedicated hardware accelerators on the 2146X. Each of these is dedicated for performing the FFT, the FIR, and the IIR algorithms.

Each accelerator has its own dedicated compute unit and its own dedicated localized memory which is distinct and separate from the processor's internal memory that we talked about before. Each accelerator can also be programmed by its own memory mapped configuration registers which can be written to and accessed by the core as well as status registers which can also be monitored by the core to check and record on the status of the accelerator.

The accelerators connect to the internal memory via two dedicated DMA channels. These DMA channels are used to move data and coefficients into and out of the accelerators and between the internal memory.

The DMA channels can be programmed to move data and coefficients from any of the internal memory blocks that we discussed before. So there is no restriction in terms of which memory blocks can be used for data or coefficients.

The way this is set up is by the user configuring a transfer control block, or a TCB as we call it which tells the DMA engine to move the data and the coefficients into the accelerator and start the process of filtering.

There's also an option to generate an interrupt to the core upon the completion of an accelerator work unit.

While these accelerators - the FFT, the FIR, or the IIR can be enabled for operation at any point and time, however they could be primed and set up ahead of time. And they could just be enabled one after the other.

Now let's look a little bit at the block diagram of the accelerators. Each accelerator - as we talked about - contains its own dedicated compute unit which is unique for the algorithm that it's processing. In addition to the compute unit, each accelerator also has its own localized memory.

There are two blocks and they can be used to store coefficients as well as data. The contents of this localized memory actually depend on the accelerator itself.

So for example in the case of the FIR this localized memory could store the coefficients and the delay-line memory data. In case of the IIR accelerators it could store the input as well as the state machine and the state memory and in case of the FFT accelerator it could store the twiddle factor data as well as the input and the intermediate data results.

As we talked about on the previous slide, there are two dedicated DMA channels which allow for moving of data and coefficients between the internal memory and the localized memory of the accelerators.

In addition to the DMA engine there is also a control block here where specific filter parameters such as filter length, number of channels, and so on can be programmed by the core and the accelerators can also be enabled or disabled.

There are also status registers where the core could actually monitor the state machine of these specific accelerators.

Now let's look in a little more detail at the FIR accelerator. There are a total of four 32-bit floating-point MACS on the FIR accelerator, each of which can generate a 32-bit result. There are also four 32-bit fixed-point MACS, each of which generates an 80-bit result.

These multiple accumulate units of the MACS are used for actually performing the FIR algorithm. In addition to the compute unit there are 1024 words of localized coefficient memory and 1024 locations of memory for storing the delay line data. As we mentioned on the previous slide, several parameters of a typical FIR filter can be programmed and configured by the user by means of the configuration register, which is a memory mapped register which can be written and read by the core.

Some of the parameters that can actually be controlled are the number of channels, up to a max of 32, the number of taps within the filter, up to a total of 4096, and also popular FIR configurations such as up-sampling, down-sampling and the control ratios for these filters.

In addition configuration of the accelerators there is also dedicated bits within the FIR configuration register for enabling, disabling the accelerator unit itself. For disabling and enabling the DMA engine that's associated with the accelerator, as well as for optionally generating an interrupt to the core upon the completion of the accelerator work unit.

As we mentioned on the previous slide, the FIR accelerator supports all standard and popularly used operating modes of a typical FIR filter. So for example we support single rate operating modes, we support decimation, or down-sampling, as well as interpolation or up-sampling.

What we mean by single rate operating mode is when the number of taps within a filter is less than the maximum size of the memory, which in this case is 1024. In this case the entire filter can be performed in a single iteration and the results are written back to memory.

But in case of what is known as single-rate-multi-iteration mode where the number of taps is larger than the maximum amount of memory available, then the ~~IAR~~ accelerator implements such a filter in multiple iterations. So for example if you had 2048 taps and in a 1024 point filter in this case the accelerator would do the assessed two iterations of 1024 taps each. Something to keep in mind in the iteration mode is that fixed-point data formats are not supported and the data must be floating-point.

In addition to single rate operating modes we also support decimation and interpolation and decimation is basically where you generate a single output result for some amount of input samples. So maybe one output for N input samples. There is flexibility within the FIR configuration registers for actually setting up the parameters for decimation.

In case of interpolation we generate multiple outputs for each input sample that goes into the accelerator. Again as is the case with decimation and single rate operating modes, many of the popular parameters for this particular operating mode are programmable through the FIR configuration register.

It's important to point out here that decimation and interpolation do not support iterative modes. So for example decimation and interpolation are single iteration operations and hence the

filtering is limited to 1024 taps. Another point to remember is that up sampling and down sampling ratios have to be integers.

Now let's talk a little bit about the IIR hardware accelerator unit. The IIR hardware accelerator is implemented using the popular transposed direct form II filter. There's one multiply accumulate unit which supports both 32-bit as well as 40-bit floating-point data. Just as was the case with the FIR accelerator the IIR accelerator operates on data and coefficients that have been fetched from internal memory and already placed in its own localized memory.

The IIR accelerator supports up to a maximum of 24 channels and up to 12 cascaded biquads per channel. And there is sufficient amount of localized memory, 80 kilobits in this case to locally store all the biquad coefficients for all the 24 channels.

As was the case with the FIR accelerator, in the case of the IIR unit as well we support all popular and standard implementations of an IIR filter. So anything that you would possibly do in software if you were using the core to perform this algorithm can be done using this hardware accelerator.

So we support sample-based operations where a single input sample from a particular channel is processed through all the biquads at a time, and then the next channel is processed and so on. And we also support Window-based filters where a certain number of inputs are taken in at a time, and multiple output samples are generated.

In case of the Window-based IIR filters the window size is configurable for any number between one and 1024. As was the case with the FIR accelerator, there is a memory mapped configuration register that is unique to the IIR accelerator unit which allows a user to program almost all of the parameters that might be useful for an IIR filter. This includes the number of channels which can be programmed up to a max of 24. You can set operating modes, floating-point and rounding modes - whether you round up to zero or the nearest value.

We also can program a number of biquads per filter up to a max of 12. And as we just talked about in case of Window-based filters, we can configure the number of samples or the window size up to 1024. In addition to setting the parameters for the IIR filter the config register also

allows the core to enable and disable the filter itself, for enabling, disabling the DMA engine associated with the filter, and for optionally generating an interrupt to the core.

Now let's look at the FFT hardware accelerator unit. Along the same lines as the FIR and the IIR accelerator the FFT unit also contains four 32-bit floating-point MACS, each of which generates a 32-bit result, and a total of six 32-bit floating-point adders, each of which generates a 32-bit output. The combination of this multiple and accumulates is the implementation of the basic radix 2 FFT butterfly that is actually implemented on this accelerator.

In addition to the compute unit there is a 512 location localized memory to store the twiddle factors and 1024-location memory to store the input data and intermediate results.

There is a memory mapped configuration register that is unique to the FFT accelerator along the same lines as there were for the FIR and the IIR, where a user can program the number of points for the FFT and also enable and disable the accelerator and the DMA engines and for optionally generating an interrupt to the core upon the completion of the FFT work unit.

The FFT accelerator also supports most popularly used and most common user operating modes. So for example it supports both complex and well as real inputs. In case of real input mode the FFT accelerator automatically zeros out the imaginary inputs and presents the data as a complex input to the accelerator itself.

For FFT sizes of up to 256 points, the localized memory that we just talked about on the previous slide is sufficient to perform the entire FFT and write the data back without having to store any intermediate results.

For FFT of sizes larger than 256 points, the data and the twiddle factors have to be stored; have to be moved in and out through the DMA in portions. And this is automatically done by an accelerator without the need for any user intervention.

In addition to these modes there's also the option to have the FFT process one chunk of data after another continuously which we call the multi-iteration mode. Or there is an option after performing a single FFT operation for the accelerator to actually go into idle and then can be restarted at some future point by user or the core writing to the configuration register.

Chapter 9: SHARC 2146x Thermal Diode

Now let's talk about the thermal diode on the SHARC 2146X. The purpose of the thermal diode is to provide an accurate measurement of the die temperature to a user. And it actually takes the guess work out of having to measure the temperature of a part on the package, and having to extrapolate the results to get an estimate of what the accurate die temperature is.

Having the thermal diode actually takes the guess work out of having to extrapolate this information. And customers can use this information to accurately implement control mechanisms and ensure that the processor is always operating within the acceptable temperature ranges. This is extremely critical for industrial & instrumentation applications as well as in the automotive market where you want to ensure that the processor stays within the operating range.

The thermal diode is also convenient in the sense that it provides the temperature as a voltage value and actually makes it very easy for any control mechanism implemented by user to actually act upon the results.

Chapter 10: Media Local Bus Interface

Let's talk about the media local bus interface on the SHARC 2146X. Before we discuss the media local bus port let's spend a couple of seconds understanding the MOST - or the media oriented systems transport network. The MOST as it's called is a widely accepted standard and has a lot of vendors and a lot of automobile manufacturers who conform to this technology. And basically what it does is it enables multiple devices within an automobile such as DVD players, mp3 players, GPS systems, Blue-Tooth devices and car phones to seamlessly work as if they were a single system instead of having to be operated as multiple independent separate devices.

The media local bus, or the MLB is an intra chip communication bus which allows a processor to connect seamlessly to the MOST network.

So what exactly does that mean? What this means is that we can have multiple devices such as the ones I'm pointing to here, this could be a SHARC 2146X and this could be another SHARC or any other device that implements this bus standard. Sitting on this common bus which could either be three or five bits wide, and they would all be connected to a dedicated chip called the

media local bus or the mediaLB controller. The back end of this controller connects to the MOST network and the purpose of the controller is to actually translate network data and standards between the MOST and the MLB, or the media local bus.

As for our implementation of the media LB interface it is an industry standard provided by SMSC which is considered to be a leader in this technology, and is based on their offering of the OS 62400 device macro.

We support all popular and commonly used operating modes of the mediaLB port. So for example we support both the three pin as well as the five pin interfaces. We support selectable clock ratios as well as multiple options for controlling streaming and packet data. We have also implemented a loop-back mode in our hardware which allows for better testability.

In addition we also support the most popularly used formats, so for example we support Big Endian and Little Endian formats. We support streaming of channel data and RX and TX status information.

We also support moving the data between the media LB port and the memory either by DMA or [direct memory] access, which does not require code intervention, or we also have an optional core driven transport mode.

All of these features can be programmed by the user via memory mapped control registers.

Chapter 11 SHARC 2146x Link Ports

Now let's talk a little bit about the 2146X link ports. There are two link ports on a SHARC 2146X, each of which can transmit data eight bits at a time, and is a high speed point to point communication protocol. Having this protocol that can move data between two SHARCS or between a SHARC 2146X and host FPGA processor allows us to actually free up the external port for other applications or other interfaces.

The basic amount of data that is moved at a time is eight bits and the basic data unit of a link port transfer is four bytes, which is 32-bits of data.

In addition the link ports also implement the link port acknowledge or LACK signal which allows for a link port receiver to throttle and control the data from a transmitter. So for example if the receive buffer on the link port is full it actually inserts its link port acknowledge and the transmitter is forced to wait before it can transmit more data to the receiver until the receiver FIFO has emptied.

Now this hand shaking protocol allows for a high band width and optimized data transfer. Each link port is bi-directional which means it can transmit or receive data, but it has to be programmed to transmit or receive again by way of memory mapped control registers which are programmable by the core.

The implementation of the link ports on the 2146X is similar to the implementation of the link ports that used to exist on previous SHARCs such as the 2116X but with some enhancements. The link ports can be programmed via software through the use of memory mapped registers to run at a variety of clock speeds relative to the core.

The way the protocol is implemented is that the link port transmitter always generates and provides the clock and a link port receiver always runs on an external clock. So for example if you were communicating between two 2146X processors via the link port the transmitter would be generating and transmitting the clock along with the data, and the receiver would just be operating off of the link port clock and asynchronous to the link port receiver's core clock frequency.

There is a dedicated DMA channel per link port so there is a total of two DMA channels that are associated with the link port to support moving blocks of data over each link port.

In addition to DMA mode of operation there is also support for a core driven mode where each data word generates an interrupt to the core which then has to be processed.

An important feature on the 2146X processor is now the ability that we can boot via the link port.

Link port booting is implemented over link port 0 and whether the processor is going to boot through the link port or not is determined by sampling of the boot config pins. The boot config

pins are described on the processor data sheet and I encourage you to refer to that for more information.

When a processor, when the 2146X is configured for link port booting it behaves as a link slave, in other words it expects the data and the clock to be supplied by the transmitter. The link port transmitter at the other end could either be another SHARC 2146X or it could be an FPGA or a host processor which implements the link port transmitter protocol at its end.

Chapter 12 SHARC 2146x Power Savings and Clock Management

Now let's look at some of the features that have been implemented on the 2146X with the view of optimizing and maintaining power savings and clock management.

There are a couple of memory mapped registers which can be programmed by the core at any time which allow a user to optionally turn off the clock to any of the modules or the peripherals that are unused. The purpose of having this is to reduce the overall power consumption of the processor.

There are also options by way of memory mapped registers to program the ratios of the DDR2 controller, the media LB port, and the link port operating frequencies as we talked about.

I'm not going to get into the details of these registers themselves but I encourage you to refer to the 2146X programmers reference and the hardware reference manuals for a detailed breakdown of the bits and the functionality of these registers.

Chapter 13 SHARC 2146x Direct Memory Access (DMA)

Thus far we've talked about some of the new features and the new modules that have been implemented on the 2146X. In this section we'll talk a little bit about some of the enhancements that we've accomplished on the existing DMA (or the direct memory access engine) and some of the existing peripherals on the SHARCs.

In terms of the DMA engines, there are a total of 67 zero-overhead DMA channels that are available on the processor which allow for transmitting and receiving data between multiple peripherals and any of the internal blocks of memory.

There's also DMA channel support for the dedicated hardware accelerators as we talked about as well as for moving data over link ports.

And as I pointed out earlier in the presentation we have added a nifty little feature on the DMA engine where a user can directly transmit or receive data between any of the serial ports and external memory without having to have it moved into internal memory.

In terms of the capabilities of the DMA engines we support all of the functionalities and the operating modes that used to exist before. So for example we have conventional DMA where you're just moving a block of data from one peripheral or one location to another. We also have the option for chain pointer DMA where upon the completion of a DMA block of data, the DMA automatically kicks off another transfer. We have the option for circular buffer DMAs where the pointer of the DMA actually wraps around to the top of upon the completion of the work unit.

We have the option for what is called tap list DMA or scatter/gather DMA which is particularly useful in audio applications. And we also have the powerful delay line DMA functionality which allows for a user to move a block of data and also optionally select and move samples within that data block to a new location. For a detailed description of the different DMA flavors that are supported, again I encourage you to refer to the processor hardware reference manual and the user documentation.

Now let's look at the breakdown of the 67 DMA channels on the 2146X that we talked about on the previous slide. There is a total of 16 DMA channels dedicated to the serial ports. There are eight DMA channels that are dedicated to the input data ports (or the IDPs). There are a total of two DMA channels associated with the UARTS, and two DMA channels associated with the serial peripheral interface (or the SPI). There are two DMA engines associated with the external port which includes the DDR2 controller and the asynchronous memory interface.

And there are also two DMA engines for moving data internally from one internal memory block to another.

So these are the DMA channels that existed on previous SHARC processors and continue to stay implemented on the 2146X. In addition to these channels we also now have two dedicated DMA

channels for the link port, and we have two dedicated DMA channels as we talked about for the accelerators. And we have a total of 31 DMA channels available and dedicated for the media local bus port.

So far we actually looked at the 2146X, the new features on the 2146X and some of the enhancements in terms of DMA engine. Now let's take a couple of slides and look at some of the enhancements to the instructions and to the peripherals.

Chapter 14 SHARC 2146x Enhanced Instructions

On the 2146X we have stayed backward source code compatible with our earlier SHARCs but we've also enhanced a couple of instructions, mainly to help our code-generation tools to generate more optimized code.

There are two main instructions that I'd like to go over here. The first is what we call the enhanced modify instruction. So for those of you existing SHARC users, you're probably familiar with the modified I,M instruction which basically modifies a data address unit or DAG I register with a modified value and writes the result back to the I register.

On the 2146X while maintaining the old instruction we have also made it a little more flexible to allow the destination of this modify to be to a different register.

Another instruction that has been modified has to do with the shifter. Now again existing SHARC customer would be familiar with two of the instructions, the FEXT and FDEP which operated on the 32-bit registers at a time.

Now we have actually enhanced the ability for a user to go in and manipulate the internal 64-bit shift register known as the BFF or the Bit FIFO. The details behind the instruction usage are available in the programmer's reference and the user's documentation. I'm not going to spend a lot of time going into it right here, but I'd just like to point out that the enhanced Bit FIFO manipulation instruction allows a user to operate on the entire 64-bit register and not just 32-bit chunks at a time.

Chapter 15 SHARC 2146x Peripheral Enhancements

In addition to keeping the peripheral set that existed on the earlier SHARCs we made a few tweaks and enhancements to some of the peripherals which we feel will help make it even more desirable from a user standpoint.

One of the features on the SPORTs, when operating in I²S master DMA mode is now the ability for the SPORT to generate a trailing serial clock edge after transmitting the last word of a DMA transfer. This used to be done via software and the ability to do this in hardware actually reduces the number of MIPS from a I²S receiver standpoint. We've added a new feature when operating the SPORT in TDM mode where once it's enabled and waiting for an external frame sync, the serial port does not immediately begin transmitting or receiving a word. It actually waits for a frame sync edge and upon seeing the edge it starts to receive or transmit data. This actually helps us in synchronization in a TDM mode and prevents us from either receiving a portion of a word or incorrectly transmitting or receiving words when the SPORT is first enabled.

A useful feature with respect to SPORT transmit DMA is that we have the option to generate a SPORT DMA interrupt upon either a DMA word count reducing to zero as was the case before, or we also have the option to generate an interrupt after the last bit of the last word has been transmitted out. This is very helpful because a user can just go into the interrupt service routine and disable the SPORT without having to worry about the last word being correctly shifted out.

As I alluded to before when it comes to SPORT DMA we have added a nifty little feature which allows you to move data back and forth between any of the serial ports and directly to external memory, or from external memory.

And last but not least an important enhancement to the SPORT DMA mechanism is the ability to place the DMA TCBS or the transfer control blocks now in either internal or external memory. On previous SHARCs they used to be limited to having to be placed within internal memory, we've ~~kind of~~ (please remove) removed that restriction on the 2146X.

Now what are some of the other enhancements on some of the other peripherals like the IDP, SPI and the S/PDIF?

In terms of the IDP or the input data port we've also provided some bits in terms of control registers for synchronization which actually makes it easier for the IDP to actually transmit or receive data on a positive or a negative edge of the clock, which is programmable and again prevents the IDP from transmitting or receiving data in the middle of a word.

We have the ability to program the delay between the SPI clock cycles when operating in SPI master mode. Again this is a feature that we believe is useful especially when it comes to SPI slaves that are on the other side.

As was the case with the SPORT, on the SPI interface as well we provide the option to generate an interrupt to the core upon either completion of the DMA data transmit or receive *or* upon when the last bit of the last word has been transmitted out. Again this also helps from a user standpoint because it takes out the guess work and a user could go into the interrupt service routine and immediately turn off the SPI without having to worry about the last word being transmitted out correctly.

Also on the SPI we have the option to control the slave chip select in both C-phase =0 as well as C-phase =1 modes. We've also support for non audio frames of 2048 and 4096 samples.

Chapter 16: Conclusion

So to conclude in this presentation we've looked at some of the key features of the 2146X processor. We have discussed the on-chip memory that's present on the processor along with a brief description of the way the internal blocks are laid out. We've talked about the new feature in terms of the variable instruction size execution which allows for a user code reduction of between 20 and 30% while maintaining the same performance as regular code execution.

We've also touched upon some of the new modules, specifically the DDR2 DRAM controller which now provides us with a high speed and a dedicated high performance external memory interface to supplement the large amount of on-chip memory that we have.

We've talked a little bit about a thermal diode which allows a user to accurately monitor and act upon any temperature variation of the actual die of the processor.

Analog Devices
SHARC 2146X
ADEV032

We've spent a couple of slides talking the media local bus interface which is of particular importance to products that are offered to the automotive markets.

We've also talked about the two dedicated byte-wide link ports which are now present on the 2146X which provide for high speed point-to-point communication channel between the 2146X and a host FPGA processor.

We've also spent a little time understanding the dedicated FFT, FIR, and IIR hardware accelerators which now allow for the most popular user algorithms to be off-loaded to these units thereby freeing up valuable core MIPS for user and system specific applications and proprietary algorithms.

I'd like to thank you today for joining me as we discuss the SHARC 2146X. For additional information I encourage you to contact our website at; www.analog.com/SHARC which has very valuable information in addition to what was presented today. We have user information in terms of data sheets, hardware reference and program's reference manuals. We have applications and EE-notes and we also have code examples.

I also encourage you to register for the 90-day free trial of our VisualDSP++ tools which actually provides an excellent opportunity to understand our processors.

The test drive comes with the entire suite of code generation tools such as the compiler, the assembler, and the linker and a software simulator.

We also offer a variety of multi-day workshops which provide you with a detailed information on our SHARC processors including the architecture, the features, the peripherals and the programming using our tools.

If you have any further questions on today's presentation I encourage you to click on the "Ask a question" button and we will definitely get back to you. Thank you very much.