

ASIP SYNTHESIS

Arnold James D'Souza (2001 VLS 008)
Praveen Prakash (2001 VLS 019)

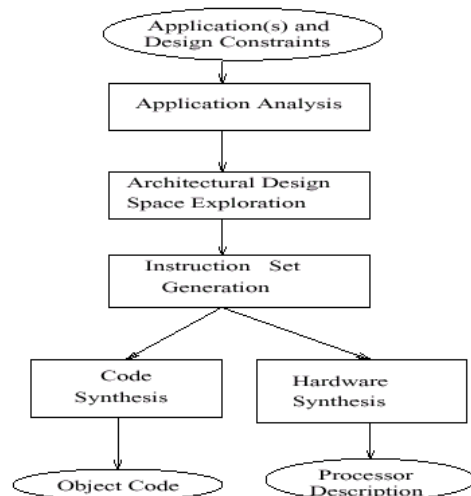
Introduction

An **Application Specific Instruction Set Processor (ASIP)** is a processor designed for a particular application or for a set of specific applications. An ASIP exploits special characteristics of application(s) to meet the desired performance, cost and power requirements. ASIPs can be very efficient when applied to a specific application such as digital signal processing, servo motor control, automatic control systems, avionics, cellular phones etc. ASIPs offer a better cost/performance ratio by exploiting the characteristic of the application they are dedicated to. They respond to the demand of an improvement in the computing power through the introduction of special hardware units and the exploitation of parallelism available in the application. On the other hand, they can be cost effective by saving hardware resources with the implementation of only those ones actually used in the application. ASIPs are balanced between two extremes; ASICs and General Programmable Processors (GPP). Since an ASIC is designed for one behavior, it is difficult to make any changes at a later stage. ASIPs offer the required flexibility in such case at a lower cost than GPPs.

ASIP Synthesis Steps ¹

The following are the important steps in any ASIP design methodology:

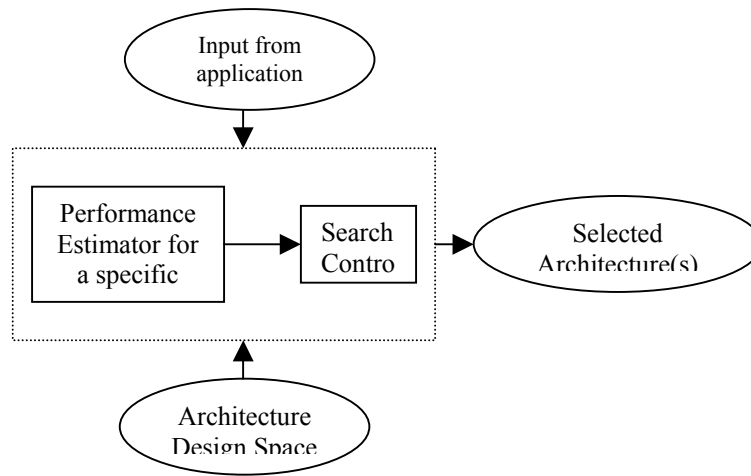
- Design starts with application behavior.
- Evaluate several architectural options.
- Identify hardware functionalities to speed up the application.
- Introduce hardware resources for frequently used operations.



ASIP Design Methodology

Application Analysis: The ASIP design process starts with the analysis of the application. It is essential to analyze the application to get the desired characteristics/requirements, which can guide the hardware synthesis as well as instruction set generation. This analysis includes determining various parameters like the data type and their access methods, execution counts of operators and functions etc. used in the application program, the frequency of individual instructions and sequence of contiguous instructions. Other parameters include, the average basic block size, number of Multiply-Accumulate (MAC) operations, ratio of address computation to data computation instructions etc.

Architectural Exploration: The figure below shows a typical architecture explorer block diagram. Inputs from the application analysis step are used along with the range of architecture design space to select a suitable architecture. The selection process is a search technique over the design space driven by a performance estimator.



Architecture Explorer

Almost all the techniques suggested by various researchers emphasize the need for a good parameterized model for the architecture. Different values can be assigned to the parameters, which constitute the design space. So the size of the design space will depend on the number of parameters and the range of values, which can be assigned to these parameters. The various parameters, which can be varied are:

- Number and type of FUs, Storage Units, and Interconnect Resources.
- Issue width, pipeline depth, number of branch units, number of memory units, the size of instruction and data-caches.
- Number of registers, addressing support
- Complex arithmetic patterns, like complex multiplication etc.

Performance Estimation: Performance of possible architectures is estimated and a suitable architecture satisfying performance, and power constraints and having minimum hardware cost is selected. Two major techniques are used for Performance Estimation:

- *Scheduler Based:* In this type of approach the problem is formulated as a resource constrained scheduling problem with the selected architectural components as the resources and the application is scheduled to generate an estimate of the cycle

count. Profile data is used to obtain the frequency of each operation. This approach captures the differentiating features of the instruction set and the special functional resources, rather than the complete specification required for code synthesis or simulation. The estimator also suggests optimizations and ranked bottlenecks. The optimizations considered include optimized multi-operation pattern matching (e.g. MAC), address mode optimization, loop optimizations etc.

- *Simulator Based:* A simulation model of the architecture based on the selected features is generated and the application is simulated on this model to compute the performance. A specific simulator for each architecture template is derived in three steps: the architecture instance is constructed, an execution model is added, and the executable is instrumented with metric collectors to obtain performance numbers.

Search Control: Algorithms for search range from branch-and-bound to exhaustive over a restricted domain. The search space is first reduced by eliminating the machine configurations, which do not satisfy given area constraints. A Branch-and-Bound algorithm is then used for searching an optimum solution. The domain of search can also be reduced by restricting the various architectural parameter ranges based on certain gross characteristics of the processors, such as branch delay, presence/absence of MAC, depth of pipeline and memory bandwidths.

Instruction Set Generation: The various approaches to Instruction Set generation can be classified as:

- *Instruction Set Synthesis:* Here the instruction set is synthesized for a particular application, based on the application requirements, quantified in terms of the required micro-operations and their frequencies. In one of the approaches, the problem of instruction formation is integrated with the scheduling problem. Simulated Annealing technique is used to solve the scheduling problem. Instructions are generated from times steps in the schedule and each time step corresponds to an instruction. The drawback of this approach is that it can't handle multi-cycle instructions.

Another approach to instruction synthesis is to first match the list of micro-operations to the primary instructions and then estimate the execution time if only these instructions are employed. If the estimated performance is unacceptable, then complex instructions are considered. If the performance is still unacceptable, special instructions, which need special hardware units, are included.

- *Instruction Selection:* Here, a superset of instructions is available and a subset of them is selected to satisfy the performance requirements within the architectural constraints. For example, the functionalities may be classified as *primary*, *basic*, and *extended* and the intermediate instructions are described as primary RTL, basic RTL and extended RTL respectively. The selection problem is formulated as an ILP problem with the objective to maximize the performance of the CPU under the constraints of the chip area and power consumption. Other approaches use pattern matching for instruction selection. The set of template patterns is extracted from the instruction set and the graph representing the intermediate code of the application is covered by these patterns. By arranging instruction patterns in a tree structure, the matching process becomes significantly fast.

Code Synthesis: Two approaches are normally adopted for code synthesis:

- *Retargetable Code Generator:* Here, an object code for the application is generated taking the architecture template, the Instruction Set Architecture (ISA) and the application as the input. This approach tries to address following sub-problems:
 - Instruction Mapping: DFG of the given application is mapped to the instruction pattern using tree/graph pattern matching technique in an attempt to get a good cover.
 - Resource Allocation and Binding: Resources like registers, memory, FUs, busses, are allocated and corresponding bindings are performed.
 - Scheduling: Instructions are scheduled for execution.
- *Compiler Generator:* Taking architecture template and instruction set architecture as the inputs, a customized compiler is generated. This is used to generate object code for the given application written in a high level language. A compiler generated in this manner generally has phases similar to general compilers, namely, program analysis, intermediate code optimization and code generation. The difference is that the optimizations are tailored to the specific architecture-application combination.

CASE STUDIES

In the remaining part of this report, we consider two specific examples of processors, which are customized to meet certain requirements, imposed on them by the applications they are intended for. We consider two processors:

- Intel IXP1200 Network Processor
- Philips TM-1300 Media Processor

The Intel IXP1200 Network Processor

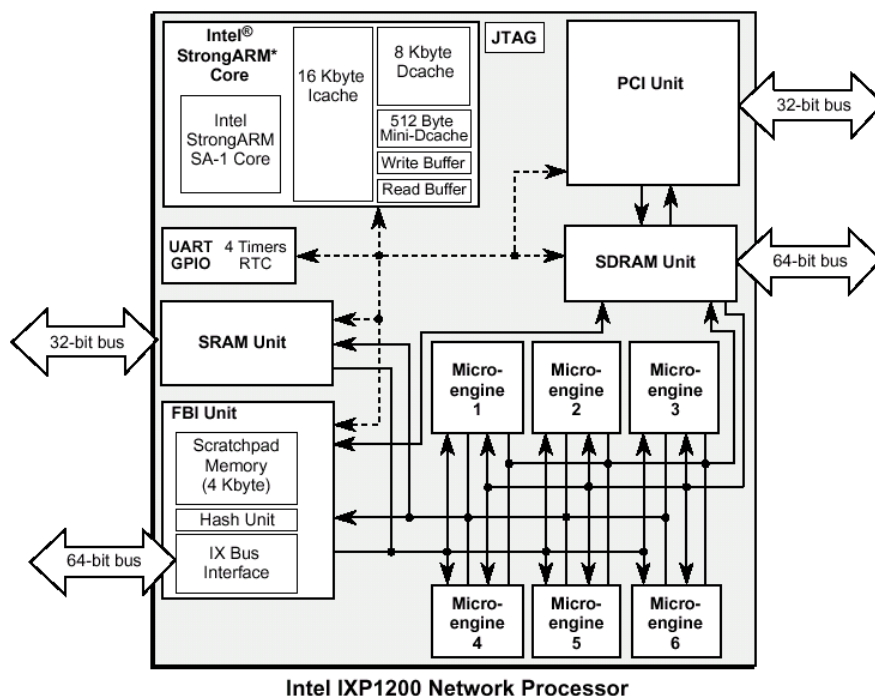
The IXP1200 is a true Network Processor, which combines the best attributes of a custom network ASIC with the capabilities of a general-purpose microprocessor. Applications of this processor include remote access concentrators, ISP/WAN switches, enterprise multi-service switches and routers, storage area networks, LAN switches, and intelligent PCI adapters.

The IXP1200 is designed specifically as a data control element for applications that require access to a fast memory subsystem, a fast interface to I/O devices such as network MAC devices, and processing power to perform efficient manipulation on bits, bytes, words, and long-word data.

It combines the popular StrongARM* processor with six independent 32-bit RISC data engines with hardware multithread support that combined, provide over 1 giga-operations per second. The Microengines contain the processing power to perform tasks typically reserved for high speed ASICs. In LAN switching applications, the six Microengines are capable of packet forwarding of over 3 million Ethernet packets per second at Layer 3. The StrongARM* processor can then be used for more complex tasks such as address learning, building and maintaining forwarding tables, and network management.

ARCHITECTURE^{2,3}

The network processor is a fully programmable device, which has been specifically designed to handle the high-speed data manipulation requirements of networking equipment. It implements a symmetric array of six RISC data processors and a StrongARM* processor, two memory interfaces, a PCI Interface, and an IX Bus Interface on a single chip. The 64-bit IX bus provides high bandwidth connectivity to the micro-engines, StrongARM*, memory, and off-chip devices like a MAC device or another IXP1200. A PCI-bus interface allows integration with an external control processor.



StrongArm* Core

This is a standard 32-bit RISC processor. The StrongARM* core processor and six RISC Microengines provide the processing power required to forward greater than 3 million Ethernet packets per second through the IXP1200. A multi-IXP1200 system scales linearly, so that a system comprised of eight IXP1200s can process over 24 million packets per second.

Microengines

Six 32-bit, multithreaded RISC Microengines perform data movement and processing without assistance from the StrongARM* core. Each Microengine has four independent program counters, zero overhead context switching and hardware semaphores from other hardware units to ensure that each Microengine can be fully utilized. A Microengine's powerful ALU and shifter perform both ALU and shift operations in a single cycle. The instruction set is specifically designed for networking and communications applications that require bit, byte, word and long-word operations to forward data quickly and efficiently. Each Microengine contains a large amount of local memory and registers. It contains 4 Kbytes of high-speed RAM Control Store organized as 1024 by 32 bits, used for program execution, 128 32-bit General Purpose Registers, and 128 32-bit transfer registers to service the SRAM and SDRAM Units.

The micro-engines perform all the packet processing tasks. They have hardware support (i.e. zero-overhead swapping) for four threads each, for a total of 24 threads on the chip. While the four threads on a micro-engine share a common register file, there is a software policy that splits the register file into four parts (one for each thread). This makes it possible for the device to swap contexts in a single cycle. The micro-engines also have special instructions for packet processing, like find first bit set, barrel shift, and extract byte/word.

The FBI Unit and the IX Bus

The FBI Unit is responsible for servicing fast peripherals, such as MAC-layer devices, on the IX Bus. This includes moving data to and from the IXP1200 Receive and Transmit FIFOs. The FBI Unit contains the Transmit and Receive FIFO elements, control and status registers (CSRs), a 4 Kbyte Scratchpad RAM, and a Hash Unit for generating 48- and 64-bit hash keys. It also contains the drivers and receivers for the IX Bus.

SRAM and SDRAM Units

The IXP1200 supports two high performance memory units. The SRAM Unit provides fast memory that can be used to store look-up tables. The SDRAM Unit provides lower cost memory for forwarding information and transmit queues.

The SDRAM Unit provides access low cost, high bandwidth memory for mass data storage. The SDRAM is operated by commands that are loaded into command queues within the unit. The SDRAM Unit decodes the command, reads or writes the data, then deletes the command from the head of the queue. The read and write sources may be SDRAM memory locations, transfer registers, or the Transmit and Receive FIFOs in the FBI Unit. The PCI, Microengines, and StrongARM* core require single byte, word, and longword write capabilities. The SDRAM Unit supports this using a read-modify-write technique. As data is written from the PCI or StrongARM* core to SDRAM, a quadword is read from SDRAM. The IXP1200 then updates only the bytes that were enabled and

writes the entire quadword of data back to SDRAM memory. These three steps are performed automatically.

The IXP1200 provides an SRAM Unit for very high bandwidth memory for storage of lookup tables and other data for the packet processing Microengines. The SRAM Unit controls the SRAM (up to 8 Mbytes), BootROM (up to 8 Mbytes) for booting, and 2 Mbytes of SlowPort address space for peripheral device access.

In addition to the micro-engines, the IXP has some special hardware units that aid in packet processing. There is a programmable **HASH ENGINE** and specialized queues that are shared by all the micro-engines and the StrongARM. Receive (Transmit) FIFOs provide an interface to MAC-layer devices by reading (writing) packets into (out of) on-chip queues that can be accessed via the IX bus. There is an on-chip data cache of 8KB, 16KB of instruction cache for the StrongARM, and 4kbyte of on-chip Scratchpad SRAM.

Some hardware units provided in the IXP1200 to cater to the application requirements are:

- Hardware hash unit
- Serial UART
- Real-time clock
- 24-bit timers with CPU
- 4-Kbyte Scratchpad memory

Instruction Set

As might be expected for such a specialized architecture, the IXP1200 has some unusual characteristics. Of particular note are the provisions for multithreading and data throughput. The processor hides the long latencies of off-chip memory references by rapidly switching contexts with other threads. Using this feature on the IXP1200 requires minimal coding. For example, the microengine instruction set has an instruction for reading and writing to SRAM (where routers often store their IP-address lookup tables). An optional token appended to the instruction invokes multithreading:

SRAM [read, xfer2, addr_sram, 4],ctx_swap

This instruction makes one of the IXP1200's microengines to read four 32-bit words from external SRAM (starting at symbolic address *addr_sram*) and store the values in four internal registers (starting at register *xfer2*). The final token, *ctx_swap*, gives the processor permission to swap out this thread in favor of another thread running on the same microengine until the memory operation is complete. The next line of code in the first thread won't execute until the processor signals the microengine that all four words of data have arrived in the destination registers. The IXP1200 can suspend execution of any thread until a specified event (in this case, reading four words of data from external SRAM) allows execution to resume. This allows the processor to hide memory latencies beneath the useful work performed by other threads.

The IXP1200 has similar instructions for accessing external SDRAM, the internal scratchpad memory, and a pair of on-chip FIFO buffers for packet I/O. (Each FIFO buffer can hold 16 elements of 64 bytes, the most common size of Internet packets.) Context swapping is very useful in packet processing, because routers often hold packets in off-chip SDRAM while analyzing the headers. A common task is to read several bytes of

data in sequence— for instance, when a router prepares to examine the nested headers of a packet. A programmer can stack several read (or write) operations like the one shown above in a row, then append the *ctx_swap* token to only the last instruction in the series. These operations are entered in a queue, and context swapping is done among other threads while accessing memory. Execution is resumed at the next line of code only after the last memory transfer is complete.

Other tokens allow additional efficiencies. For instance, the SDRAM instruction has an *optimize_mem* token to hold pending memory operations in a pair of special queues for the odd and even banks of SDRAM. While the SDRAM is performing row-and-column operations on one bank, it precharges the opposite bank, saving a few clock cycles as it switches back and forth. Other than appending the *optimize_mem* token to the SDRAM instruction, the process is transparent to the programmer.

Some of the application specific instructions are listed below:

Instruction	Meaning
R_FIFO_RD	Read the receive FIFO buffer
T_FIFO_WR	Write to the transmit FIFO buffer
SCRATCH	Read/Write Scratchpad RAM
SDRAM	Read/Write SDRAM
SRAM	Read/Write SRAM
FIND_BSET	Find first bit set in any 16-bit register field
CTX_ARB	Swap context and wake on event
HASHx_64	Perform 48-bit hash (x = 1, 2, 3)
HASHx_48	Perform 64-bit hash (x = 1, 2, 3)

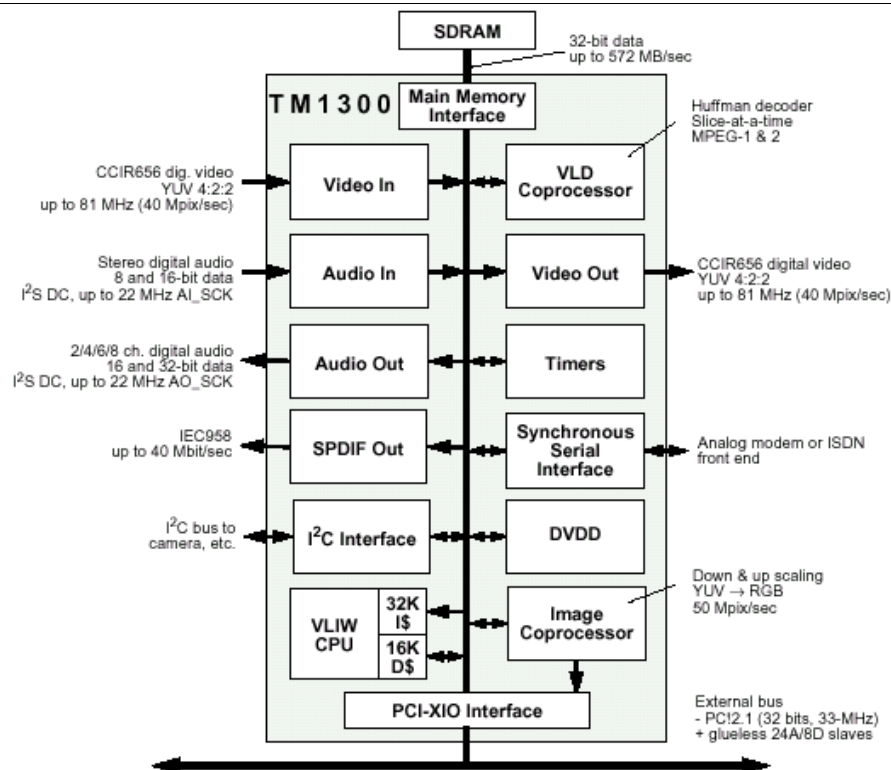
Philips TM-1300 Media Processor

TriMedia TM1300 processor is a VLIW-ASIP designed for multimedia class of applications. The processor can be the heart of an independent embedded system or it may be used as a hardware accelerator in a PC environment.

The VLIW processor core may be running a Real-Time OS kernel (such as pSOS). TM1300 has independent DMA-driven multimedia I/O units and co-processors. There is a local DRAM, which feeds the processor, co-processors and I/O units. The processor coordinates the activities of the I/O units and the coprocessors.

TM1300 has I/O units and an external bus interface, which provide for a glue-less interface to the A/V equipment and the host system. The external bus can interface to 8-bit peripherals as well as PCI2.1 master/slave devices without need of extra support chips/ bus-bridges. Following is a brief introduction to the components of the TM1300:

Architecture ⁴

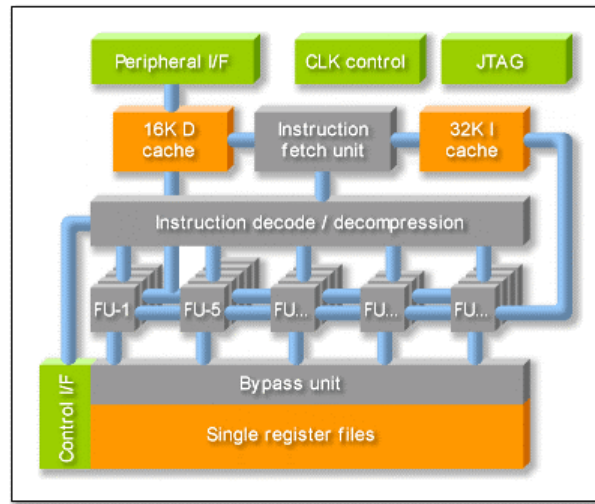


Internal Data Highway Bus

The internal bus (or data highway) connects all internal blocks together and provides access to internal control/ status registers of each block, external SDRAM, and the external bus peripheral chips. The internal bus consists of separate 32-bit data and address buses. Transactions on the bus use a block-transfer protocol. On-chip peripheral units and coprocessors can be masters or slaves on the bus. Access to the internal bus is controlled by a central arbiter, which has a request line from each potential bus master. The arbiter is programmable so that the arbitration algorithm can be tailored for different applications. Peripheral units make requests to the arbiter for bus access and, depending

on the arbitration mode, bus bandwidth is allocated to the units in different amounts. Each mode allocates bandwidth differently, but each mode guarantees each unit a minimum bandwidth and maximum service latency. All unused bandwidth is allocated to the DSPCPU (the VLIW core). The bus allocation mechanism is one of the features of TM1300 that makes it a true real-time system instead of just a highly integrated microprocessor with unusual peripherals.

VLIW Processor Core (DSPCPU)



The heart of TM1300 is a powerful 32-bit DSPCPU core. The DSPCPU implements a 32-bit linear address space and 128, fully general-purpose 32-bit registers. The registers are not separated into banks; any operation can use any register for any operand. The TM1300 core uses a VLIW instruction-set architecture and is fully general-purpose. The VLIW instruction length allows five simultaneous operations to be issued every clock cycle. These operations can target any five of the 27 functional units in the DSPCPU, including integer and floating-point arithmetic units and data-parallel multimedia operation units.

Video-in Unit

The Video In (VI) unit interfaces directly to any CCIR 601/656-compliant device that outputs 8-bit parallel, 4:2:2 YUV time-multiplexed data. Such devices include direct digital camera systems, which can connect gluelessly to TM1300 or through the standard CCIR 656 connector with only the addition of ECL level converters. The VI unit demultiplexes the captured YUV data before writing it into local TM1300 SDRAM. Separate planar data structures are maintained for Y, U, and V. The VI unit can be programmed to perform on-the-fly horizontal resolution subsampling by a factor of two if needed. Many camera systems capture a 640-pixel/line or 720-pixel/line image. With subsampling, direct conversion to a 320-pixel/line or a 360-pixel/line image can be performed with no DSPCPU intervention. Performing this function during video input reduces initial storage and bus bandwidth requirements for applications requiring reduced resolution.

Video-out Unit

The Enhanced Video Out (EVO) unit essentially performs the inverse function of the VI unit. EVO generates an 8-bit, CCIR656 digital video data stream that contains a composited video and graphics overlay image. The video image is taken from separate Y, U, and V planar data structures in SDRAM. The graphics overlay is taken from a pixel-packed YUV data structure in SDRAM. Compositing allows both alpha-blending and chroma keying. The EVO unit can also upscale the video image horizontally by a factor of two to convert from CIF/SIF to CCIR 601 resolution.

The sample rate for EVO unit pixels is independently and dynamically programmable. The high-quality, on-chip sample clock generator circuit allows the programmer subtle control over the sampling frequency so that audio and video synchronization can be achieved in any system configuration. When changing the sample frequency, the instantaneous phase does not change, which allows sample frequency manipulation without introducing audio or video distortion.

Image Coprocessor

The ICP off-loads common image scaling or filtering tasks from the DSPCPU (the VLIW core). Although these tasks can be easily performed by the DSPCPU, they are a poor use of the relatively expensive CPU resource. When performed in parallel by the ICP, these tasks are performed efficiently by simple hardware, which allows the DSPCPU to continue with more complex tasks. The ICP can perform either horizontal or vertical image filtering and resizing. In another mode, the ICP can perform horizontal resizing followed by color-space conversion. While sending the final, resampled and converted pixels over the PCI bus to the video frame buffer, the ICP uses a full, per-pixel occlusion bit mask accessed in destination coordinates to determine which pixels are actually written to the graphics card frame buffer for display. Conditioning the transfer with the bit mask allows TM1300 to accommodate an arbitrary arrangement of overlapping windows on the PC video screen.

Variable Length Decoder (VLD)

The variable-length decoder (VLD) relieves the DSPCPU of decoding Huffman-encoded video data streams. It can be used to help decode high bitrate MPEG-1 and MPEG-2 video streams. The lower bitrate of videoconferencing can be adequately handled by DSPCPU software without coprocessor. The VLD is a memory-to-memory coprocessor. The DSPCPU hands the VLD a pointer to a Huffman-encoded bit stream, and the VLD produces a tokenized bit stream that is very convenient for the TM1300 image decompression software to use. The format of the output token stream is optimized for the MPEG-2 decompression software so that communication between the DSPCPU and VLD is minimized.

Audio In/Out Units

The Audio In (AI) and Audio Out (AO) units are similar to the video units. They connect to most serial ADC and DAC chips, and are programmable enough to handle most serial bit protocols. These units can transfer MSB or LSB first and left or right channel first. The audio sampling clock is driven by TM1300 and is software programmable within a wide range. Like the VO unit, AI and AO sample rates are separately and dynamically programmable. The high-quality on-chip sample clock generator circuits allows the

programmer subtle control over the sampling frequency so that audio and video synchronization can be achieved in any system configuration. When changing the sample frequency, the instantaneous phase does not change, which allows sample frequency manipulation without introducing audio or video distortion.

As with the video units, the audio-in and audio-out units buffer incoming and outgoing audio data in SDRAM. The audio-in unit buffers samples in either 8- or 16-bit format, mono or stereo. The audio-out unit transfers 16- or 32-bit sample data for mono, stereo or up to 8 audio channels from memory to the external DACs. Any manipulation or mixing of sound data is performed by the DSPCPU since this processing will require only a small fraction of its processing capacity.

S/PDIF Out Unit

The Sony/Philips Digital Interface Out (SPDO) unit allows output of a 1-bit high-speed serial data stream. The SPDO unit can also be used as a general purpose high-speed data stream output device such as a UART. The SPDO unit supports 2-channel PCM audio, one or more Dolby Digital six-channel data streams, or one or more MPEG-1 or MPEG-2 audio streams. It supports arbitrary programmable sample rates independent of and asynchronous to the AO unit sample rate.

Synchronous Serial Interface

The on-chip synchronous serial interface (SSI) is specially designed to interface to high integration analog modem frontends or ISDN frontend devices. In the analog modem case, all of the modem signal processing is performed in the TM1300 DSPCPU.

I2C Interface

TM1300 implements an I2C master for use in single master environments only. This interface allows TM1300 to configure and inspect the status of I2C peripheral devices, such as video decoders, video encoders and some camera types.

Instruction Set ⁴

TM1300 has a standard 32-bit instruction set. It also supports single precision floating point arithmetic instructions. For media applications, the TM1300 features a host of “custom” operations. The basic motivation behind these specialized instructions is following: The media applications work on 8-bit video streams and 8/16-bit audio streams. The data-path, however, is 32-bit wide. It makes a lot of sense to use the 32-bit data-path for concurrent operations on four 8-bit or two 16-bit operands. Hence these custom operations. These instructions not only permit efficient utilization of resources but also help save in terms of code-size and power-consumption (due to lesser bus-activity). These instructions have corresponding macro functions for use in C/C++ source code. The assembly tools generate the appropriate opcodes. Following is a summary of these instructions:

Function	Custom Op	Description
DSP absolute value	dspiabs	Clipped signed 32-bit absolute value
	dspidualabs	Dual clipped absolute values of signed 16-bit halfwords
Shift	dualasr	dual-16 arithmetic shift right
Clip	dualiclipi	dual-16 clip signed to signed
	dualuclipi	dual-16 clip signed to unsigned
Min,max	quadumax	Unsigned bitwise quad max
	quadumin	Unsigned bitwise quad min
DSP add	dspiadd	Clipped signed 32-bit add
	dspuadd	Clipped unsigned 32-bit add
	dspidualadd	Dual clipped add of signed 16-bit halfwords
	dspuquadaddui	Quad clipped add of unsigned/signed bytes
DSP multiply	dspimul	Clipped signed 32-bit multiply
	dspumul	Clipped unsigned 32-bit multiply
	dspidualmul	Dual clipped multiply of signed 16-bit halfwords
DSP subtract	dsphisub	Clipped signed 32-bit subtract
	dsposub	Clipped unsigned 32-bit subtract
	dspidualsub	Dual clipped subtract of signed 16-bit halfwords

Sum of products	ifir16	Signed sum of products of signed 16-bit halfwords
	ifir8ii	Signed sum of products of signed bytes
	ifir8iu	Signed sum of products of signed/unsigned bytes
	ufir16	Unsigned sum of products of unsigned 16-bit halfwords
	ufir8uu	Unsigned sum of products of unsigned bytes
Merge, pack	mergedual16lsb	Merge dual-16 least-significant bytes
	mergelsb	Merge least-significant bytes
	mergemsb	Merge most-significant bytes
	pack16lsb	Pack least-significant 16-bit halfwords
	pack16msb	Pack most-significant 16-bit halfwords
	packbytes	Pack least-significant bytes
Byte averages	quadavg	Unsigned byte-wise quad average
Byte multiplies	quadumulmsb	Unsigned quad 8-bit multiply most significant
Motion estimation	ume8ii	Unsigned sum of absolute values of signed 8-bit differences
	ume8uu	Unsigned sum of absolute values of unsigned 8-bit differences

TriMedia SDE (Software Development Environment) ⁵

The SDE includes a set of tools, which facilitate development of multimedia applications in C/C++. It has an ANSI C/C++ compiler, linker/loader, source-level debugger, cycle accurate machine-level simulator, runtime system and an execution environment for hosted/standalone system.

The compiler is optimized for VLIW processor core. Speculative execution is supported. The debugger provides access to memory and all on-chip peripherals. There are special hardware assists for instruction and data watchpoints.

pSOS Real-Time OS ⁶

pSOS is a multitasking OS which sees all the running processes as tasks, I/O drivers or ISRs. It also has support for asymmetric multiprocessing environment with one master node. The task execution is fully pre-emptive and scheduling is priority based. The OS supports dynamic storage allocation. It also features a variety of time-out, wakeup and alarm services.

A Look from the ASIP point of view

VLIW-based ASIPs have four customization dimensions:

- Functional-Unit Specialization
- Interconnect Specialization
- Register File Specialization
- Instruction-Set Specialization

In this case study we discussed some of the architectural features of TM1300 processor. Now we highlight the "application specific" specialization with respect to the four dimensions mentioned above.

- *Functional Unit Specialization*

The TM1300 blocks such as the Image Coprocessor (ICP) and specialized I/O units providing glueless interface is a clear example of specialization. Within the VLIW core, there are FU's, which perform special functions, for example:

- *alu*: 32-bit pack/unpack
- *dspalu*: dual 16-bit, quad 8-bit multimedia arithmetic
- *dspmul*: dual 16-bit and quad 8-bit multimedia multiplies
- *shifter*: multi-bit shift
- *fcomp*: single cycle floating point compares
- *ftough*: iterative floating point square root and division

- *Interconnect Specialization*

The Data Highway, as highlighted above, uses block transfer protocol, which suits the data-operation intensive media applications. The guarantees on bandwidth and max service latency facilitate real-time processing of streaming media.

- *Register File Specialization*

As mentioned above, the DSPCPU has just one register file which feeds to five sets of FU's (it being a 5-issue VLIW core). In effect, there is no "specialization" as such.

- *Instruction-set Specialization*

TM1300 has a standard 32-bit instruction set. It also supports single precision floating point arithmetic instructions. The "custom" operation set, discussed briefly above, enables efficient data processing while maximally utilizing the 32-bit data-path in the core.

Specialized Instruction-Set vs General Purpose Instruction-Set

Following are two examples

1. Consider the following code fragment, which finds the first bit set in a byte	
x86	IXP1200
<pre> ... MOV AX, byte (non-zero) MOV CX, 00 BACK: RCL INC CX JNC BACK ... </pre>	<pre> find_bset reg </pre>
2. Consider another code fragment, which computes the unsigned sum of two operands and clips the result into unsigned range.	
x86	TM-1300
<pre> ... ADD AX, BX JNO DONE MOV AX, 0xFFFF DONE: </pre>	<pre> dspuadd rsrc1, rsrc2 </pre>

References

1. Manoj Kumar Jain, M. Balakrishnan, Anshul Kumar, “ASIP Design Methodologies”.
2. “Understanding Network Processors”, A report on survey and analysis of Network Processors, by Niraj Shah, EECS Berkeley.
3. “Intel IXP1200 Network Processor: Datasheet”,
<http://developer.intel.com/design/network/datashts/278298.htm>
4. “TM-1300 Media Processor: Product Specification”, Philips Semiconductors.
<http://www.semiconductors.philips.com/pip/ptm1300fbea>
5. TriMedia Technologies – Software Development Environment
<http://www.trimedia.com/products/briefs/sde.html>
6. TriMedia Technologies – Real-Time Operating Systems
<http://www.trimedia.com/products/briefs/rtos.html>