4

A Rotated Array Clustered Extended Hypercube Processor: The RACE-H™ Processor

Gerald G. Pechanek
Lightning Hawk Consulting Inc. and Priest & Goldstein, PLLC

Mihailo Stojancic
ViCore Technologies Inc.

Frank Barry
Onward Communications Inc. and Appalachian State University

Nikos Pitsianis
Duke University

CONTENTS

4.1 Introduction

Many products require efficient high-performance processing to meet the growing computational requirements of numerous media applications. Tailoring a processor's architecture and system interfaces to minimize processing overhead and inefficiencies in data transfers is required to provide efficient processing for these media applications. In addition, efficiency and low power require the use of techniques that remove the dependency on the processor clock speed to obtain adequate performance. Applications such as high-definition (HD) multistandard video processing require almost continuous processing at the highest performance level. Because power use is highly dependent on frequency, very little power savings can be achieved in these high-compute applications by varying clock frequencies to minimize power use during less-demanding program segments. Use of a flexible parallel architecture is an important means to provide high performance without having a high dependence on the clock rate.

As an example, the high-profile H.264/AVC video encoding standard for picture sizes of 1920×1080 and larger represents one of the most computationally intensive algorithms to be implemented in future commercial and consumer products. The performance requirements greatly exceed the capabilities of current generation multigigahertz general-purpose processors. In addition, the video encoding standards require a great deal of flexibility to support the numerous coding tools, such as the discrete cosine transform, support for adaptive block sizes, intraspatial prediction, intertemporal prediction, support for interlaced coding and lossless representation, and deblocking filtering, to name only a few [1]. This flexibility requirement imposes programmable capability in the encoding hardware. To address these video requirements and provide an alternative to obtaining performance from a high-speed clock, a processor requires a highly flexible approach to parallel processing. The RACE-Hypercube™ processor, running at relatively low clock frequencies, provides multiple forms of selectable parallelism and an architecture that is moldable to an application.

To meet the performance and flexibility requirements in a cost-effective manner the RACE-H™ processor provides a hybrid architecture consisting of a scalable array processor enhanced with a scalable array of application-specific hardware-assist coprocessing units. To make such a hybrid processor widely available at low cost requires the use of standard design practices that allow the design to be fabricated at multiple semiconductor suppliers. This means that custom-designed SOCs, optimized to a particular manufacturing process, cannot be easily used. Consequently, the standard approach of increasing clock speed on an existing design in an attempt to meet higher performance requirements is not feasible.

The need to support multiple standards, such as MPEG-2, MPEG-4, and VC-1 standards as SMPTE 421M, and to quickly adapt to changing standards, has become a product requirement [2]. To satisfy this need, programmable DSPs and control processors are being increasingly used as the central SOC design component. These processors form the basis of the SOC product platform and permeate the overall system design including the on-chip memory, DMA, internal buses, and the like. Consequently, choosing a flexible and efficient processor, which can be manufactured by multiple semiconductor suppliers, is arguably the most important intellectual property (IP) decision to be made in the creation of an SOC product.

In recent years, a class of high-performance programmable processor IP has emerged that is appropriate for use in high-volume embedded applications such as digital cellular, networking, communications, video, and console gaming [3,4,5]. This chapter briefly describes the RACE-H™ processor as an example of the architectural features needed to meet the highest demands of the H.264.AVC high-profile video encoding requirements. The next section provides a brief description of the RACE-H™ architecture. Section 4.2, titled "The RACE-H™ Processor Platform," describes how the RACE-H™ architecture fulfills system requirements, with a focus on the DMA subsystem and development tools. The "Video Encoding Hardware Assists" section briefly discusses examples for processor element hardware assists. The "Performance Evaluation" section presents performance results and projections and Section 4.6 concludes the chapter.

4.2 The RACE-H™ Architecture

In numerous application environments, there is a need to significantly augment the signal-processing capabilities of a MIPS, ARM, or other host processor. The RACE-H™ processors provide streamlined coprocessor attachment to MIPS, ARM, or other hosts for this purpose. The RACE-H™ architecture offers multiple forms of parallelism that are selectable at each stage of development, from SOC definition through software programming. Through selectable parallelism, the RACE-H™ processors achieve high performance at relatively low clock rates, thereby minimizing power use.

These forms of parallelism include 14 degrees of parallelism that may be selected. The first degree concerns the number of very long instructional word (VLIW) slots that are executed. The total number of slots available for execution is determined at implementation time where up to eight instruction slots may be implemented in the RACE-H™ architecture. The number of slots that may be executed up to the maximum number implemented can then be selected and vary instruction by instruction during program execution. The second degree includes the determination of supported application-specific instructions and the selection of the appropriate instructions for algorithmic operations. The third degree concerns the number and type of selectable application-specific hardware assists implemented in each processing element (PE), allowing program control over the enabling and operation of each hardware assist. By attaching hardware assists to each PE, the hardware assist capability of the core scales with the RACE-Hypercube dimensions. The fourth degree is determining the number of PEs to implement and for specific algorithmic use to mask the PE array as needed for optimum power and performance efficiency. The fifth degree concerns operating the core as a single-issue uniprocessor for control single-thread operation. The sixth degree concerns operating the core as a variable-length indirect VLIW (iVLIW) uniprocessor for increased performance in uniprocessor tasks. The seventh degree supports operating each PE as a single-issue PE for all enabled PEs. The eighth degree supports operating each PE as a variable-length iVLIW issue PE for all enabled PEs. The ninth degree allows the selection of 32-bit packed data operations, including single 32-bit, dual 16-bit, and quad 8-bit operations which can be mixed on each instruction slot in a VLIW. The tenth degree allows the selection of 64-bit packed data operations, including single 64-bit, dual 32-bit, quad 16-bit, and octal 8-bit operations that can be mixed with each other and with packed 32-bit data operations on each instruction slot in a VLIW. The eleventh degree concerns conditionally executing instructions independently within a VLIW and independently within each PE. The twelfth degree supports the independent selection of mesh, torus, hypercube, and hypercube-complement PE-to-PE communications concurrently on the array with DMA, load, store, and the other execution unit operations. The thirteenth degree allows for independent threaded array operations across the number of implemented and enabled PEs to be controlled by a single controller and the fourteenth degree supports background DMA operations that may be scaled across the number of DMA lanes implemented to match the dimensions of the RACE-Hypercube.

To provide for these multiple degrees of freedom, the RACE-H™ processor is organized as an array processor using a sequence processor (SP) array controller and an array of distributed indirect VLIW PEs. The SP and each PE is provided with a small VLIW memory (VIM) that stores program-loaded VLIWs, where the VLIWs may be indirectly selected for execution. By varying the number of PEs on a core, an embedded scalable design is achieved with each core using a single architecture. This embedded scalability makes it possible to develop multiple products that provide a linear increase in performance and maintain the same programming model by merely adding array processor elements as needed by the application. As the processing capability is increased, the PE memory interface bandwidth is increased, and the system DMA bandwidth may be increased accordingly. Embedded scalability drastically reduces development costs for future products because it allows for a single software development kit (SDK) to support a wide range of products.

In addition to the embedded scalability, RACE-H™ processors are configurable in hardware organization and in software use of on-chip resources. For example, the processor hardware may be configured in the number and type of processor cores included on a chip, number of VLIW slots, supported instructions, the sizes of the SP's instruction memory, the distributed iVLIW memories, the PE/SP data memories, and the I/O buffers, selectable clock speed, choice of on-chip peripherals, and DMA bus bandwidth. The processor software may configure the use of the hardware dynamically instruction by instruction. Multiple RACE-H™ processors may also be included on a chip and organized for data pipeline multiprocessing between the multiple SP/PE-array processors. Within a RACE-H™ processor, the parallelization of subapplication tasks may also use forms of thread parallelism with a centralized host-based control, described later in this chapter.

Figure 4.1 shows the major architectural elements that make up a typical RACE-H™ processor. The RACE-H™ processor combines PEs in clusters that

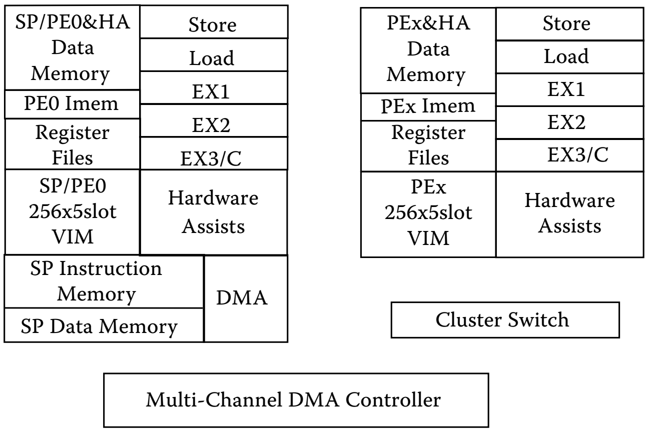


FIGURE 4.1
RACE-H processor architecture elements.

also contain a sequence processor (SP), uniquely merged into the PE array, and a cluster-switch. The SP provides program control, contains instruction and data address generation units, and dispatches instructions fetched from the SP instruction memory to the PEs in the array. Both the SP and PEs each include a common set of execution units using the same indirect VLIW architecture for all processing elements. Figure 4.1 illustrates a five-issue variable length VLIW organization where the instruction set is partitioned into store, load, execute 1 (EX1), execute 2 (EX2), and execute 3 and communicate (EX3/C) instruction slots. The architecture is easily expandable to support the addition of another load unit and additional execution units.

The RACE-H™ processor is designed for scalability with a single architecture definition and a common toolset. The processor and supporting tools are designed to optimize the needs of a SOC platform by allowing a designer to balance an application's sequential control requirements with the application's inherent data parallelism. This is accomplished by having a scalable architecture that begins with a simple uniprocessor model, as used on the SP, and continues through multi-array processor implementations. The RACE-H™ architecture supports a reasonably large array processor, as well as a simple stand-alone uniprocessor, the SP. In more detail, a RACE-$H_{2\times2}$™, RACE-$H_{4\times4}$™, and RACE-$H_{4\times4\times4}$™, are shown in Figure 4.2. The RACE-H™ architecture allows organizations with multiple SPs where each SP controls a subcluster of PEs. For example, a 4×4 may be made up of four 2×2 clusters each with their own SP or a 4×4 may be configured with only a single SP controlling the 16 PEs. In a similar manner, a $4 \times 4 \times 4$ 3D-cube may have multiple configurations of SPs and PEs depending upon a product's requirement.

The RACE-H™ architecture uses a distributed register file model where the SP and each PE contain their own independent compute register file (CRF),

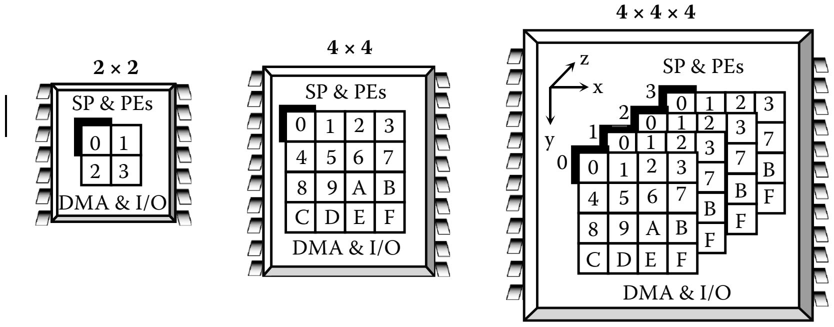


FIGURE 4.2
RACE-H_{2x2}, RACE-H_{4x4}, and RACE-H_{4x4x4} scalable products.

up to eight execution units (five shown), a distributed very long instruction word memory (VIM), local SP instruction memory, local PE instruction memory, local data memories, and an application-optimized DMA and bus I/O control unit. The CRF is reconfigurable dynamically to act as a 32 × 32-bit or 16 × 64-bit register file instruction by instruction and can vary within a VLIW and is totally integrated into the instruction set architecture. An 8 × 32-bit address register file (ARF) and a 24 × 32-bit miscellaneous register file (MRF) are also defined in the instruction set architecture. Extending to support 128-bit wide and larger width register files is also architecturally supported. The balanced architectural approach taken for the CRF provides the high- performance features needed by many applications. It supports octal byte and quad halfword operations in a logical 16 × 64-bit register file space without sacrificing the 32-bit data-type support in the logical 32 × 32-bit register file. Providing both forms of packed data independently on each execution unit allows optimum usage of the register file space and minimum overhead in manipulating packed data items.

In the RACE-H™ architecture, the address registers are separated from the compute register file. This approach maximizes the number of registers for compute operations and guarantees a minimum number of dedicated address registers. It does not require any additional ports from the compute register file to support the load and store address generation functions and it still allows independent PE memory addressing for such functions as local data dependent table lookups.

The RACE-H™ instruction set is partitioned into four groups using the high two bits of the instruction format: a control group, an arithmetic group, a load–store group, and a reserved proprietary instruction group. Figure 4.3 shows 32-bit simplex instructions in groupings that represent an example of five execution unit slots of a VLIW plus a control group (01). Up to eight execution slots are architecturally defined allowing for additional load and execution units. Group (00) is reserved for future use. The execution units include store and load units and a set of execution units. For example, a first

SP	Five Slot VLIW Instruction Types				
Control	**Store**	**Load**	**EX1**	**EX2**	**EX3/C**
Group 01	Group 10	Group 10	Group 11	Group 11	Group 11
Call	Base+Disp.	Base+Disp.	ADD/SUB	ADD/SUB	Copy
Jump	Direct/Indirect	Direct/Indirect	Butterfly	Butterfly	Shift/Rotate
Loops	Table	Table	Compare	MPY/MPYA	Permute
Return	ARF group	ARF group	AbsoluteDiff	MPYCmplx	Bit Operations
Load VLIW		Immediate	Min/Max	MPYCmplxA	SP Send/Rcv
Execute VLIW		Broadcast	Logicals	SUM2P/SUM2PA	PE to PE Exchange
		TCM			TCM

FIGURE 4.3
RACE-H instructions.

execution unit is EX1, supporting arithmetic instructions. A second execution unit is EX2, supporting, for example, multiply, multiply accumulate, and other arithmetic instructions. A third execution unit is EX3/C, supporting data manipulation instructions such as shift, rotate, permute, bit operations, various other arithmetic instructions, PE-to-PE communication, and hardware-assist interfacing instructions.

Providing a common set of execution units is also supported by the architecture. The load and store instructions support base plus displacement, direct, indirect, and table addressing modes. In addition, many application-specific instructions are used for improved signal-processing efficiency. An example of these instructions is the set of multiply complex instructions for improved FFT performance described in reference [6]. In addition, the load unit and EX3/C unit have tightly coupled machine (TCM) instructions which are used to control hardware-assist coprocessors attached to each PE. Any of the five slots of instructions can be selected on a cycle-by-cycle basis, for single-, two-, three-, four-, or five-issue VLIWs, in this particular five-issue VLIW example. The single RACE-H™ instruction set architecture supports the entire RACE-H™ family of cores from the single merged SP/PE0 1×1 to any of the highly parallel multiprocessor arrays (1×2, 2×2, 2×4, 4×4, $4 \times 4 \times 4$, …); for more details see reference [7].

The control and branch instructions are executed by the SP and in the PEs, when the PEs are operating independently. The SP and PEs are also capable of indirectly executing VLIWs that are local to the SP and in each PE. Note that VLIWs are stored locally in VLIW memories (VIMs) in each PE and in the SP and are fetched by a 32-bit execute VLIW (XV) instruction. To minimize the effects of branch latencies, a short variable pipeline is used consisting of Fetch, Decode, Execute, and ConditionReturn for non-iVLIWs and Fetch, PreDecode, Decode, Execute, and ConditionReturn for iVLIWs. The PreDecode pipeline stage is used to indirectly fetch VLIWs from their local VIMs. It is noted that an XV instruction supplies an offset address to a local VIM control unit, which computes a VIM address based on a local VIM base address register. In addition, the VLIWs located at the same VIM address in different PEs do not have to be the same VLIW. These architectural features allow for independent program action at each PE, referred to as synchronous multiple instruction multiple data (SMIMD) operation. In addition, an extensive scalable conditional execution approach is used locally in each PE and the SP to minimize the use of branches.

All loads–stores and arithmetic instructions execute in one or two cycles with no hardware interlocks. The TCM instructions initiate multicycle operations that execute independently of other PE instructions. Furthermore, all arithmetic, load–store, and TCM instructions can be combined into VLIWs, stored locally in the SP and in each PE, and can be indirectly selected for execution from the small distributed VLIW memories. In one of the supported architectural approaches, a Load iVLIW (LV) instruction is used by the programmer or compiler to load individual instruction slots making up a VLIW with the 32-bit simplex instructions optimized for the algorithm being programmed. These VLIWs are used for algorithm performance optimization, are reloadable, and require only the use of 32-bit execute VLIW (XV) instructions in the program stored in the SP instruction memory.

Many algorithms require an additional level of independent operations at each PE. The RACE-H™ architecture supports independent and scalable program thread operations on each PE. Figure 4.4 illustrates a scalable thread flowchart of independent and scalable thread operations for the RACE-$H_{4\times4}$™. The SP controls the thread operation by issuing a thread start (Tstart) instruction. The Tstart instruction is fetched from SP instruction memory and dispatched to the SP and all PEs. Based on the Tstart, each PE

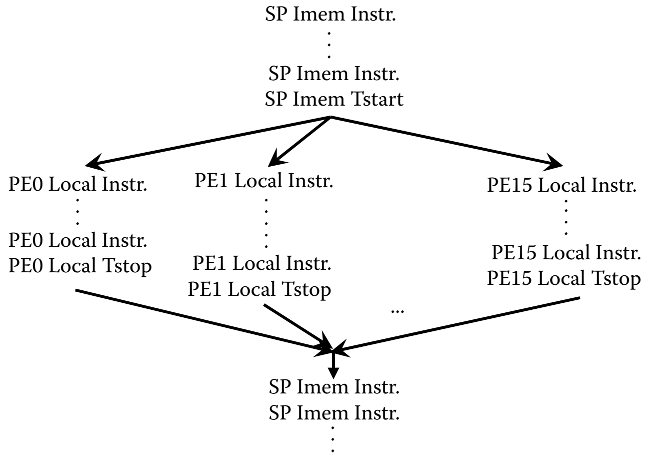


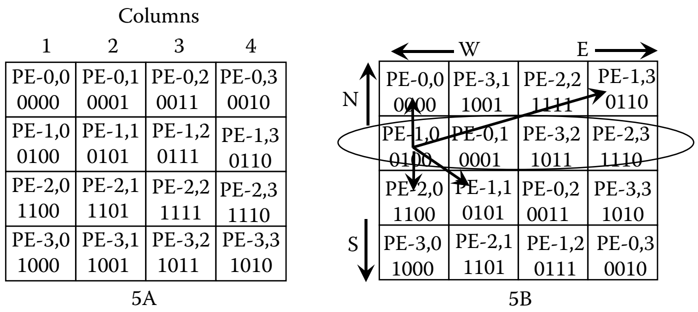
FIGURE 4.4
4 × 4 independent PE threads.

switches to local independent PE operations fetching PE local instructions from PE instruction memory. The PE instruction memory may store all types of PE instructions including PE branch instructions and a thread stop (Tstop) instruction. As shown in Figure 4.4, each PE operates independently until its operations are complete, at which point each PE fetches a Tstop instruction. A Tstop instruction causes the PE to stop fetching local PE instructions and then to wait for SP dispatched instructions. Once all PEs have completed their local independent operations, the SP continues with its fetching operation from the SP instruction memory and dispatches PE instructions to the array. In SIMD operations, a dedicated bit in all instruction formats controls whether an instruction is executed in parallel across the array of PEs or sequentially in the SP.

To more optimally support the PE array containing the distributed register files, the RACE-H™ network is integrated into the architecture providing single-cycle data transfers within PE clusters and between orthogonal clusters of PEs. The EX3/C communication instructions can also be included into VLIWs, thereby overlapping communications with computation operations, which in effect reduces the communication latency to zero. The RACE-H™ network operation is independent of background DMA operations which provide a data streaming path to peripherals and global memory.

The inherent scalability of the RACE-H™ processor is obtained in part through the advanced RACE-Hypercube™ network which interconnects the PEs. Consider, by way of example, a two-dimension (2D) 4 × 4 torus and the corresponding embedded 4D hypercube, written as a 4 × 4 table with row, column, and hypercube node labels. (See Figure 4.5A.)

In Figure 4.5A, the $PE_{i,j}$ cluster nodes are labeled in Gray-code as follows: $PE_{G(i),G(j)}$ where G(x) is the Gray code of x. The array of Figure 4.5A is transformed by a series of operations that rotate columns of PEs. First, columns 2, 3, and 4 are rotated one position down. Next, the same rotation is repeated with columns 3 and 4 and then, in a third rotation, only column 4 is rotated. The resulting 4 × 4 table is shown in Figure 4.5B.



1	2	3	4
PE-0,0 0000	PE-0,1 0001	PE-0,2 0011	PE-0,3 0010
PE-1,0 0100	PE-1,1 0101	PE-1,2 0111	PE-1,3 0110
PE-2,0 1100	PE-2,1 1101	PE-2,2 1111	PE-2,3 1110
PE-3,0 1000	PE-3,1 1001	PE-3,2 1011	PE-3,3 1010

PE-0,0 0000	PE-3,1 1001	PE-2,2 1111	PE-1,3 0110
PE-1,0 0100	PE-0,1 0001	PE-3,2 1011	PE-2,3 1110
PE-2,0 1100	PE-1,1 0101	PE-0,2 0011	PE-3,3 1010
PE-3,0 1000	PE-2,1 1101	PE-1,2 0111	PE-0,3 0010

FIGURE 4.5
Hypercube RACE-H network.

Notice that the row elements in Figure 4.5B, for example {(1,0), (0,1), (3,2), (2,3)}, contain the transpose PE elements. By grouping the row elements in clusters of four PEs each, and completely interconnecting the four PEs in a cluster, connectivity among the transpose elements is obtained and extends the connectivity beyond the connectivity provided by a hypercube. In the new matrix of PEs, the east and south wires, as well as the north and west wires, are connected between adjacent clusters. For example, using Figure 4.5A note that node (2,3) connects to the east node (2,0) with wraparound wires in a torus arrangement. Node (2,3) also connects to the south node (3,3). Now, using Figure 4.5B, note that nodes (2,0) and (3,3) are both in the same row cluster adjacent to the row cluster containing node (2,3). This means that the east and south wires can be shared and, in a similar manner, the west and north wires can be shared between all clusters. This same pattern occurs for all nodes in the transformed matrix. This effectively cuts the wiring in half as compared to a standard torus, and without affecting the performance of any SIMD array algorithm.

Because the rotating algorithm maintains the connectivity between the PEs, the normal hypercube connections remain. For example, in Figure 4.5B, PE (1,0/0100) can communicate to its nearest hypercube nodes {(0000), (0101), (0110), (1100)} in a single step. With the additional connectivity in the clusters of PEs, the longest paths in a hypercube, where each bit in the node address changes between two nodes, are all contained in the completely connected clusters of processor nodes. For example, the circled cluster contains node pairs [(0100), (1011)] and [(0001), (1110)] which would take four steps to communicate between each pair in previous hypercube processors, but takes only one step to communicate in the RACE-H™ network. These properties are maintained in higher-dimensional RACE-H™ networks containing higher-dimensional tori, and thus hypercubes, as subsets of the RACE-H™ connectivity matrix. The complexity of the RACE-H™ network is small and the diameter, the largest distance between any pair of nodes, is two for all d where d is the dimension of the subset hypercube [8]. The distance between mesh, torus, hypercube, and hypercube-complement nodes is one.

4.3 The RACE-H™ Processor Platform

The RACE-H™ processors are designed to act as independent processors that may attach to ARM, MIPS, or other hosts. The programmer's view of a RACE-H™ multiprocessor is a shared memory sequentially coherent model where multiple processors operate on independent processes. With this model, an SOC developer can quickly utilize the signal-processing capabilities of the RACE-H™ core subsystems because the operating system already runs on the host processors. In its role as an attached coprocessor, the RACE-H™ core is subservient to the host processor. A core driver running on the host operating system manages all the RACE-Hypercube processor resources on the core. A RACE-H™ system interface allows multiple RACE-H™ cores to be attached to a single-host processor.

To complement the configurable hardware, there is a RACE-H™ library of both DSP and control software routines. In addition, existing host-optimized compilers may be used for the sequential code that remains resident on the host allowing the parallel code to be optimized for the RACE-H™ cores.

Data and control communication between thread coprocessors and system components (such as a host control processor, memories, and I/O) is carried out using a DMA subsystem, one or more system data buses (SDBs), and a system control bus (SCB). The SDB provides the high bandwidth data interface between the cores and other system peripherals including system memory. The SDB consists of multiple identical lanes, and is scalable by increasing the number of lanes or increasing the width of the lanes. The SCB is a low-latency coprocessor-to-coprocessor/peripheral messaging bus that runs independently and in parallel with the SDB. This system of multiple independent application task-optimized cores is designed to have each core run an independent system-level thread supported by the programmable DMA engines.

The DMA subsystem consists of two or more transfer controllers, the DMA bus, SDB, and SCB interfaces. Each transfer controller manages data movement between the SDB and coprocessor memories, one direction at a time, across a single data "lane" of the DMA bus. Transfer controllers operate independently of each other, fetching their own transfer instructions from core memories once initiated by either the SP or the host control processor. Transfer signaling instructions and hardware semaphores may be used to synchronize data transfer with array processing and host processor activities. Figure 4.6 shows

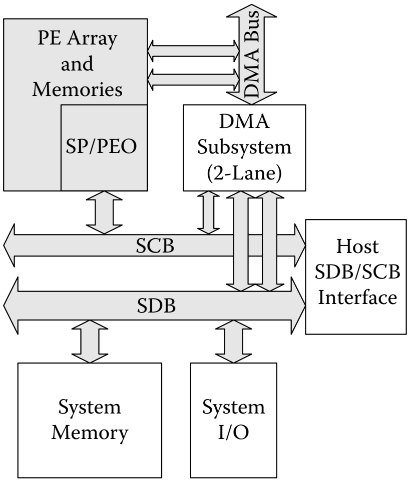


FIGURE 4.6
Two-lane DMA subsystem.

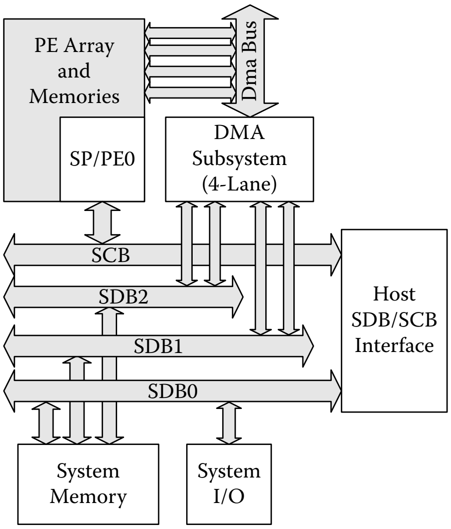


FIGURE 4.7
Four-lane DMA subsystem with two SDB-memory buses.

a configuration with a two-lane (two transfer controllers) DMA subsystem, and Figure 4.7 shows a four-lane DMA subsystem. The transfer controllers support instructions that allow independent addressing modes for the SDB side and DMA bus side. For example, in an image encoding application a group of picture macroblocks may be read from system memory using a "stride" addressing mode and distributed to array memories in various patterns using only two transfer instructions.

Each SDB is a high-bandwidth bus based on the AMBA 2.0 standard [9] and scalable in width and maximum burst length. It is used primarily for data flow between array coprocessor memories and system memories or I/O. In a multi-SDB configuration (used for very high data bandwidth), the DMA transfer controller and host interface connections will be allocated between buses to match performance requirements.

The SCB is a low-latency control bus used for communication between the host control processor, the SP, and the DMA subsystem. It is used for enabling and configuring runtime system options and also allows the SP to configure and control the DMA transfer controllers.

To streamline development, verification, and debugging, a range of modeling and prototyping platforms support system modeling, and software and hardware system development, including:

- A cycle-accurate C-simulator, which can be used to develop RACE-H™ processor software. This can be used directly with other C simulations, or with control processor tools and bus models to provide a software simulation model of an entire system.

- A software development toolkit (SDK) consisting of the GNU GCC compiler and binary tools, the RACE-H™ ANSI-C Parallel C compiler, a back-end compiler to enable high-level assembly programming with register allocation, instruction scheduling, and packing of instructions into VLIWs, and VIMA [10], a whole-program scheduler of iVLIW instructions.
- An emulation board, can be used to model RTL of an entire SOC system.
- A debugger.
- An assortment of software libraries including a real-time framework, user application program interface, debug I/O, C program runtime, mathematical libraries, and specialized DSP libraries.

The RACE-H™ debugging GUI is shared by the C-simulator and emulation board. With this development environment the software can be integrated and tested. Verification tools and supporting scripts and guidelines for the physical design are available.

4.4 Video Encoding Hardware Assists

A number of algorithmic capabilities are generally common between multiple video encoding standards, such as MPEG-2, H.264/AVC, and SMPTE-VC-1. Motion estimation/compensation and deblocking filtering are two examples of compute-intensive algorithms that are required for video encoding.

Motion estimation is computationally the most expensive part of a video encoding process. On average it can take about 60–80% of the total available computational time, thus having the highest impact on the speed of the overall encoding process. It also has a major impact on the visual quality of encoded video sequences.

The most common motion estimation algorithms are block-matching algorithms operating in the time domain. Here motion vectors are used to describe the best temporal prediction for a current block of pixels to be encoded. A time domain prediction error between the current block of pixels and the reference block of pixels is formed, and a search is performed to minimize this value. The best motion vector minimizes a cost function based on the prediction block distance and the block pixel difference.

A block of pixels of the current video frame, which is in the search range of a passed frame (in temporal order), is compared with all possible spatial positions within the search range, looking for the smallest possible difference. For the best matching reference block, a motion vector is derived which describes the relative position of the two blocks.

Multiple different criteria have been proposed for the best match evaluation. They are of different complexity and efficiency in terms of finding the global optimum over a given search area. The sum of absolute differences (SAD) is the most commonly used criterion for the best match evaluation.

A hardware assist (HA) for block-matching search may be attached to each PE and is capable of performing full search (within a search window of programmable size) for integer pixel motion vectors calculation. It is capable of simultaneous extraction of results for 16×16, 16×8, 8×16, 8×8, and 4×4 motion search based on the SAD criterion for each particular block size and given search range. The search range window may vary. For example, a search range such as 64×64 or 128×96 may be used. The hardware assist is also capable of setting up a coarse hierarchical search (through use of a special TCM instruction) by automatically decimating pixels of a larger search range (64×64, for example) and bringing the decimated 32×32 search area into the pipelined compute engines of the hardware assist. Partial search results (SAD for each current block position within the search range) may be stored locally in each PE for further processing, or stored in the local HA/PE memory.

A similar hardware assist for deblocking filtering may be attached to each PE of the RACE-H™ processor, providing for a major offloading of the most compute-intensive operations, and allowing for real-time full-feature HD video encoding.

4.5 Performance Evaluation

To illustrate the power of the highly parallel RACE-H™ architecture, a simple example is presented: Two vectors are to be added and the result stored in a third vector.

```
For ($I = 0$; $I < 256$; $I$ ++)
$A[i] = B[i] + C[i]$.
```

In a sequential-only implementation, there would be required a loop of four instructions, two load instructions to move a B and a C element to registers, an add of the elements, and a store of the result to register A. The sequential implementation takes (4*256) iterations = 1024 cycles, assuming single-cycle load, add, and store instructions.

Assuming the data type is 16-bits and quad 16-bit packed data instructions are available in the processor, the vector sum could require (4*64) iterations = 256 cycles.

Furthermore, assuming an array processor of four PEs where each PE is capable of the packed data operations, then the function can be partitioned between the four PEs and run in parallel requiring (4*16) iterations = 64 cycles.

Finally, assuming a VLIW processor such as the 2×2 RACE-Hypercube processor, a software pipeline technique can be used with the VLIWs to minimize the instructions issued per iteration such that (2*16) iterations = 32 cycles are required. This represents a 32× improvement over the sequential implementation.

With use of the 14 different levels of parallelism available on each core, the benchmarks shown in Figure 4.8 can be obtained on a RACE-Hypercube processor, with the array size shown in parentheses. (The 4 × 4 numbers are extrapolated from coded 2 × 2 numbers.)

The RACE-H™ architecture allows a programmer or compiler to select the level of parallelism appropriate for the task at hand. This selectable parallelism includes packed 32-bit and 64-bit data operations. With each additional PE, the packed data support on a processor core grows such that a 2 × 2 effectively provides four PEs each with five 64-bit execution units providing 4*5*64 bits = 1280-bits/cycle of packed data support. A 4 × 4 provides four times this for 4*1280 bits = 5120 bits/cycle (640 bytes/cycle) of packed data support which at 250 MHz provides a performance of 160 gigabytes/sec. A 4 × 4 × 4 3D cube effectively provides four times this for 4*5120 bits = 20,480 bits/cycle (2560 bytes/cycle) of packed data support which at 250 MHz provides a performance of 640 gigabytes/sec. With at least three 64-bit hardware-assist functions operating independently and in parallel in each PE, an additional 3*(8 bytes/cycle)*64 PEs*250 MHz = 384 gigabytes/sec of performance. The 4 × 4 × 4 3D cube provides 1.024 trillion bytes/sec at a relatively low clock

Benchmark	Data Type	Performance
256 pt. Complex FFT (4×4)	16-bit real & imaginary	189 cycles
256 pt. Complex FFT (2×2)	16-bit real & imaginary	383 cycles
256 pt. Complex FFT (1×1)	16-bit real & imaginary	1115 cycles
1024 pt. Complex FFT (4×4)	16-bit real & imaginary	580 cycles
1024 pt. Complex FFT (2×2)	16-bit real & imaginary	1513 cycles
1024 pt. Complex FFT (1×1)	16-bit real & imaginary	5221 cycles
2048 pt. Complex FFT (4×4)	16-bit real & imaginary	1350 cycles
2048 pt. Complex FFT (2×2)	16-bit real & imaginary	3182 cycles
2D 8×8 IEEE IDCT (4×4)	8-bit	18 cycles
2D 8×8 IEEE IDCT [11] (2×2)	8-bit	34 cycles
2D 8×8 IEEE IDCT (1×1)	8-bit	176 cycles
256 tap Real FIR filter, M samples (4×4)	16-bit	4*M + 86 cycles
256 tap Real FIR filter, M samples (2×2)	16-bit	16*M + 81 cycles
256 tap Real FIR filter, M samples (1×1)	16-bit	64*M + 78 cycles
4×4 Matrix * 4x1 vector (4×4)	IEEE 754 Floating Point	2-cycles/4-output vector
4×4 Matrix * 4x1 vector (2×2)	IEEE 754 Floating Point	2-cycles/output vector
3×3 Correlation (720col) (4×4)	8-bit	145 cycles
3×3 Correlation (720col) (2×2)	8-bit	271 cycles
3×3 Median Filter (720col) (4×4)	8-bit	360 cycles
3×3 Median Filter (720col) (2×2)	8-bit	926 cycles
Horizontal Wavelet (N Rows = 512) (4×4)	16-bit	370 cycles
Horizontal Wavelet (N Rows = 512) (2×2)	16-bit	1029 cycles

FIGURE 4.8
1 × 1, 2 × 2, and 4 × 4 RACE-Hypercube processor benchmarks.

frequency of 250 MHz with short execution unit pipelines based on 64-bit data types and an architecture that is programmer friendly.

4.6 Conclusions and Future Extensions

The pervasive use of processor IP in embedded SOC products for consumer applications requires a stable design point based on a scalable processor architecture to support future needs with a complete set of hardware and software development tools. The RACE-H™ cores are highly scalable, using a single architecture definition that provides low power and high performance. Target SOC designs can be optimized to a product by choice of core type, 1×1, 1×2, 2×2, 4×4, $4 \times 4 \times 4$, and by the number of cores. The tools and SOC development process provides a fast path to delivering verified SOC products. Future plans include architectural extensions, such as improved VIM loading techniques, extensions to 128-bit datapaths effectively doubling performance, programmable hardware-assist engines, and other extensions representing supersets of the present design that would further improve performance in intended applications.

Trademark Information

RACE-H™, RACE-H_{ixj}™, RACE-H_{ixjxk}™, and the RACE-Hypercube™ are trademarks of Lighting Hawk Consulting, Inc. All other brands or product names are the property of their respective holders.

References

[1] G. J. Sullivan, P. Topiwala, and A. Luthra, "The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions," SPIE Conference on Applications of Digital Image Processing XXVII, Special Session on Advances in the New Emerging Standard: H.264/AVC, August, 2004.

[2] Alan Gatherer et al., "DSP-Based Architectures for Mobile Communications: Past, Present, and Future," *IEEE Communications Magazine*, pp. 84–90, January, 2000.

[3] Krishna Yarlagadda, "The Expanding World of DSPs," *Computer Design*, pp. 77–89, March, 1998.

[4] Ichiro Kuroda, and Takao Nishitani, "Multimedia Processors," *Proceedings of the IEEE*, Vol. 86, No. 6, pp. 1203–1227, June, 1998.

[5] Jan-Willem van de Waerdt et al., "The TM3270 Media-Processor," *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture (Micro'05)*, pp. 331–342, 2005.

[6] Nikos P. Pitsianis and Gerald G. Pechanek, "High-Performance FFT Implementation on the BOPS ManArray Parallel DSP," *Advanced Signal Processing Algorithms, Architectures and Implementations IX*, Volume 3807, pp. 164–171, SPIE International Symposium, Denver, CO, July 1999.

[7] Gerald G. Pechanek and Stamatis Vassiliadis, "The ManArray Embedded Processor Architecture," *Proceedings of the 26th Euromicro Conference: "Informatics: Inventing the Future,"* Maastricht, The Netherlands, September 5–7, 2000, Vol. I, pp. 348–355.

[8] Gerald G. Pechanek, Stamatis Vassiliadis, and Nikos P. Pitsianis, "ManArray Interconnection Network: An Introduction," *Proceedings of EuroPar '99 Parallel Processing*, LNCS 1685, pp. 761–765, Toulouse, France, Aug. 31–Sept. 3, 1999. www.arm.com/products/solutions/AMBA_Spec.html.

[9] Nikos P. Pitsianis and Gerald G. Pechanek, "Indirect VLIW Memory Allocation for the ManArray Multiprocessor DSP," *ACM SIGARCH Computer Architecture News*, Vol. 31, Issue 1, March 2003, pp. 69–74.