# Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems

**Santanu Dutta, Rune Jensen, and Alf Rieckmann**
Philips Semiconductors

Viper is a complete system-on-a-chip solution from Philips Semiconductors for next-generation digital television and digital video applications. It incorporates Philips' intellectual-property reuse methodology in addition to standard hardware and software building blocks.

**THE DIGITAL SET-TOP BOX** receiver is a key element in the worldwide deployment of digital television (DTV) and related interactive services. First-generation set-top boxes were quite simple with support only for channel-tuning and audio-video data decoding. Increasingly, however, they are used as a gateway to the Internet and as a hub for home networking. Those domains feature a wide range of consumer broadband multimedia services, such as digital-quality broadcast television, personal video recording, and high-speed Internet access. Such services are paving the way for the set-top box to become not only a residential entertainment center but also a central piece of home computing equipment capable of handling various advanced applications.

This still-evolving market needs a highly programmable silicon-and-software solution to enable a quick convergence in user products. Philips Semiconductors has provided such a solution based on the MIPS/TriMedia dual-processor Nexperia-DVP (Digital Video Platform) architecture for implementing new subscriber services on top of middleware such as OpenTV, Microsoft TV, and Canal+ Technologies' MediaHighway. Viper, the PNX8500 chip, is one of the first system-on-a-chip (SOC) solutions to emerge from the Nexperia-DVP platform.

Viper is a highly integrated, multimedia-processing core targeted at the advanced set-top box (ASTB) and DTV markets. Featuring two on-chip CPUs and a host of autonomous functional units with direct memory access (DMA), Viper is a true example of a multiprocessor SOC. The CPU cores, aided by an array of peripheral devices (multimedia coprocessors, bus interfaces, and I/O units) and high-performance hierarchical buses, facilitate concurrent processing of audio, video, graphics, and communication data.

## Chip architecture

Viper receives, optionally decrypts, decodes, converts, and displays multiple media streams having different data formats. Besides MPEG-2 transport streams, the chip typically handles live video, audio, and various other stream types, in compressed or uncompressed formats. Processing these media streams requires not only tremendous computational power but also real-time system response in scheduling various decode-and-display functions while simultaneously running a standard operating system. Figure 1 shows the Viper architecture.

### Multiprocessor system

Early in the design, we decided to use a


0740-7475/01/$10.00 © 2001 IEEE

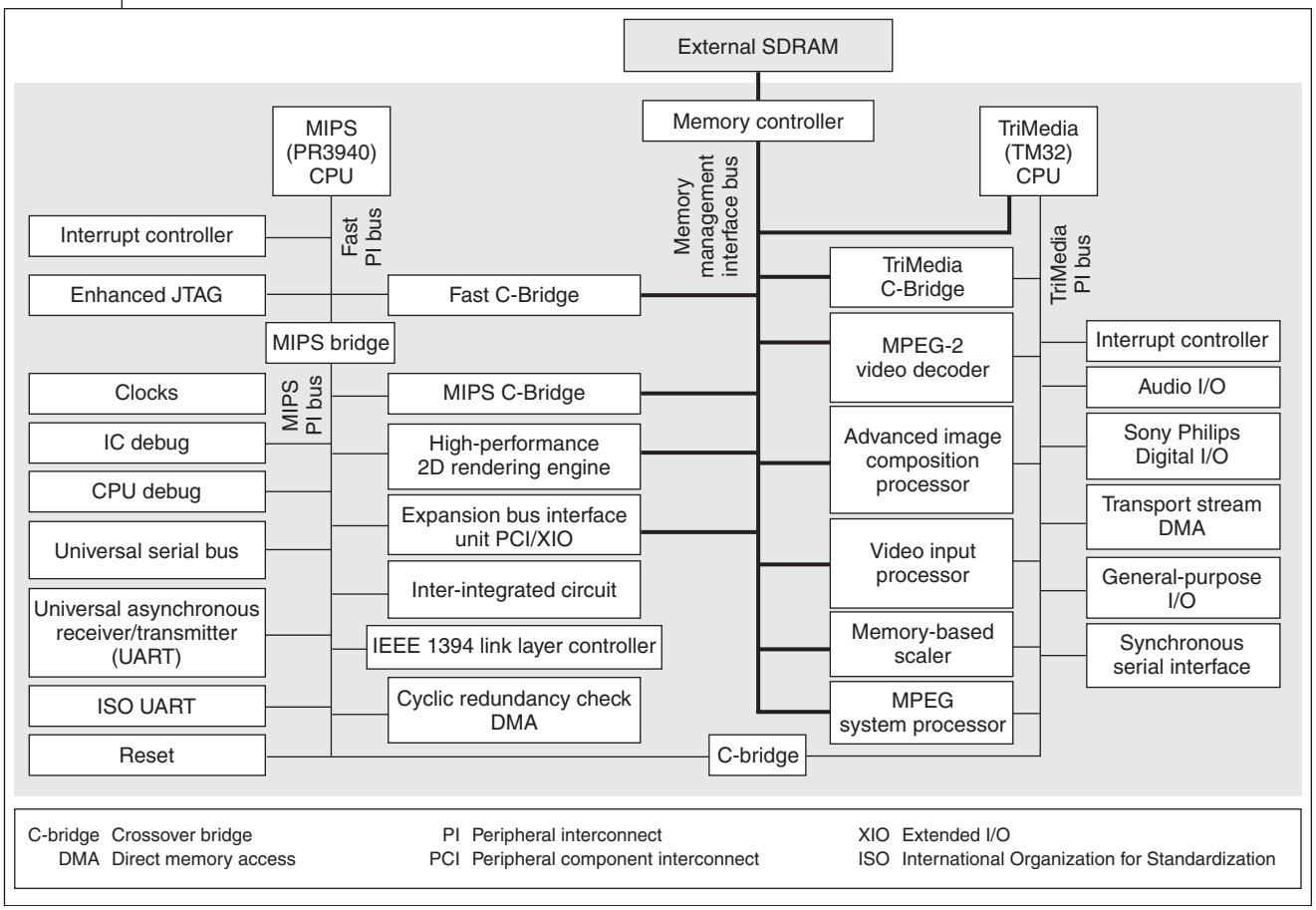


**Figure 1. Simplified block diagram of Viper.**

standard 32-bit MIPS RISC core (PR3940) and a 32-bit very long instruction word (VLIW) TriMedia core (TM32).[1] This choice was guided by Philips' portfolio of processor cores with existing applications and standard compilers. The RISC processor's three requirements were high performance; ability to run popular embedded operating systems; and efficient control of infrastructure peripherals, such as a universal asynchronous receiver transmitter (UART), universal serial bus (USB), and inter-integrated-circuit bus ($I^2C$).[2] The VLIW processor's requirements, on the other hand, were very high performance and a multimedia-enhanced instruction set suitable for audio and video processing. Such partitioning balances the system and distributes tasks and task domains across the two CPUs.

**TriMedia CPU core.** The TM32 CPU is a proven VLIW core, built and used at Philips since 1996. It has served as the computational heart for a series of mediaprocessors and supports a small real-time operating system kernel. The VLIW core handles Viper's real-time multimedia-processing tasks. The TM32's instruction set architecture (ISA) is optimized for SOC solutions requiring real-time processing of video, audio, graphics, and communications data streams. Besides traditional microprocessor operators, the ISA includes powerful multimedia-specific instructions and features five issue slots; up to five operations can be packed into a single VLIW instruction and scheduled in parallel. These operations are optionally guarded and can simultaneously target any five of 27 pipelined functional units. Other TM32 features are

- 32-Kbyte instruction cache (I-cache),
- 16-Kbyte dual-ported data cache (D-cache), and
- up to 200 MHz of operating speed.

**MIPS RISC CPU core.** We chose the 150-MHz PR3940 core as a second on-chip CPU to run the operating system and handle various control-processing tasks. Compatible with the MIPS II and MIPS 16 ISAs, this low-power, high-performance processor supports a wide range of operating systems such as WindRiver, VxWork, and Linux. It is particularly well suited for applications based on Windows CE. Key features include a six-stage pipeline, a 16-Kbyte I-cache, a 32-Kbyte D-cache, a multiply-accumulate/divide unit, a memory management unit, various timers, and a sophisticated debug support unit.

### General peripherals

To facilitate building a baseline ASTB or DTV system, Viper supports the following general-purpose, on-chip peripherals: a USB host controller, three UART interfaces, two multimaster $I^2C$ interfaces, one synchronous serial interface for implementing soft modems, and a general-purpose I/O module with infrared remote-receive capability (via fast software-read of time-stamped signal-event sequences). The design incorporates multiple instances of special peripherals, like the ISO-standard UARTs, to interact with smart cards providing *conditional access*, which lets DTV or ASTB service subscribers descramble and watch selected programs.

### Enhanced IEEE 1394 link layer

Viper includes an IEEE 1394 link layer controller that provides a physical-link-layer interface to the external world. However, to meet the requirements of set-top-box applications, which can send a transport stream to or receive a time-shifted version from an external digital VCR, we have enhanced the link to support the IEC-61883 Digital Interface Standard and 5C-based copy protection, a well-known digital copy-protection scheme.

### Expansion bus interface

Viper connects through an expansion bus interface unit to various board-level memory components and peripherals not located on the chip. The bus interface allows simultaneous connection of 32-bit peripheral component interconnect (PCI) master/slave devices as well as extended-I/O devices such as 8-bit-wide microprocessor slave peripherals, standard or NAND-type flash memories, and data-over-cable-service interface specification devices.

### Drawing engines

A high-performance 2D-rendering engine accelerates graphics functions such as bitblt, line drawings, text fills, window clipping, and monochrome data expansion. Drawing is supported at any naturally aligned memory location and any naturally aligned image stride.

### Transport stream processor

Capable of parsing MPEG-2 transport streams, the MPEG system processor (MSP) effectively performs filtering, time stamping, descrambling, demultiplexing, and section filtering of transport streams, all based on the packet identifier. Instantiated three times in the Viper SOC, each MSP features a dedicated 16-bit RISC engine to filter the MPEG-2 transport stream packets and control the decrypting process for different conditional-access providers.

### Audio interfaces

The Viper chip contains three audio input and three audio output modules (grouped as audio I/O) that provide two Philips inter-IC Sound ($I^2S$) stereo input ports, two $I^2S$ stereo output ports, and a third port that can be configured as either an $I^2S$ input or an 8-channel output. The Sony Philips Digital I/O module provides a Sony Philips Digital Interface (SPDIF) output with IEC-1937 capabilities and an SPDIF input to connect to external sources such as a DVD player.

### Video-processing blocks

One of the main design goals of Viper was to create high-quality audio and video with a resolution of $1{,}920 \times 1{,}080$ interlaced pixels. We designed Viper to simultaneously receive multiple streams and render, composite, and display two output streams on separate output channels (for video and audio). The Viper's video-processing units are as follows:

- *Advanced image composition processor.* Instantiated twice in the Viper SOC, it combines images from the main memory and composes the displayed picture.

- *Memory-based scaler.* This unit scales and converts video and filters graphics.
- *MPEG-2 video decoder.* This slice-level decoder decodes up to 133 megapixels/second and is suitable for high-definition decoding.
- *Video input processor.* Instantiated twice in the SOC, it receives standard definition video in National Television System Committee (NTSC) and phase-alternating line (PAL) formats.

### Miscellaneous modules

Modules that support the Viper SOC but lack user-level functionality (and, therefore, are not necessarily made explicit in Figure 1) include

- one programmable interconnect (PI) bus controller per PI segment;
- one programmable interrupt controller per CPU;
- an enhanced Joint Test Action Group (EJTAG) module providing MIPS debug ports;
- a JTAG module that implements boundary scan logic and provides TM32 debug ports;
- a global-register-file module containing software-readable and -writable registers not associated with any particular peripheral device;
- global clock, reset, and power-down logic; and
- boot logic supporting autonomous or host-assisted bootstrapping.

## Design considerations

One of Viper's most challenging tasks is controlling all media streams to decode and produce the right data at the right time. One stringent requirement on the output (display) side, for example, is for the audio to match the timing of the video. Furthermore, depending on the application and user mode, the different video-processing blocks might need to be shared among multiple streams, thereby calling for a challenging real-time scheduling of operations. Such complicated processing demands that the highly interactive Viper respond to events quickly to prevent buffer under- or overflows.

### CPU tasks and trade-offs

Scheduling in the Viper SOC usually falls to the CPU with the faster response time: the TM32. Despite running all the audio decoding and processing functions, the TriMedia core has enough extra bandwidth to implement other nontrivial multimedia algorithms that the hardware functional units do not directly support. The MIPS core, on the other hand, runs the operating system and, on top of it, the service provider's software application. The application software handles service accessibility (conditional access) and general control functions. The MIPS processor also handles all graphics-related functions.

### Sharing system resources

With the two CPUs splitting the multimedia processing and control processing, each CPU becomes responsible for the peripherals belonging to its task domain. This leads to the concept of bus separation (assuming separate processor buses), in which each CPU generally "owns" all the devices on its local bus. Not all peripheral devices, however, can be owned by one CPU in all user applications, so we kept every peripheral accessible from both CPUs but with a preference. If the CPUs share a peripheral at runtime, they must negotiate that peripheral's availability through the use of semaphores (signals that help govern access to common system resources by only one CPU at any time).

Combining both CPUs in one system, from a SOC standpoint, lowers the overall computing cost by sharing system resources such as main memory, disk, and network interfaces. All the on-chip functional units, or peripherals, are programmed via CPU writes to their control registers. Because these control registers are memory mapped, programmable reads from or writes to the registers are commonly referred to as memory-mapped I/O or programmable I/O transactions. Even though both CPUs can address each peripheral, the peripheral is usually read or written by the CPU whose local bus it connects to.

### Bus system hierarchy and design

We explored various bus architecture options before deciding on the bus topology and implementation and began by carefully studying the system requirements.

Table 1. Comparison of tristate and point-to-point buses.

| Item | Tristate bus | Point-to-point bus |
|---|---|---|
| Wiring | Few, but potentially very long, wires resulting in crosstalk sensitivity and antenna problems; comparable routing resources for the two buses | More wires; depending on the multiplexer tree, this may be a big issue even for a 0.18-micron technology; wires are generally shorter and not expected to have antenna problems |
| Testability | Complicated—requires tristate control, at-speed test due to potentially highly resistive/capacitive paths, potentially long runtimes for ATPG tools, and potentially many patterns | Simple—same as ordinary modules |
| Layout | Complicated—requires careful decoupling and regrouping of drivers for large buses | Simple—assuming few agents; may be unroutable if number of agents is large |
| Post-layout timing fixes | Cannot be buffered and will in general result in poor transition times | Simple—same as ordinary modules |
| Observability for debugging | Simple—few wires | Difficult |
| Multiple-master access to peripherals | Fairly simple—all modules connect to each other via tristate bus | Potentially more complicated—depending on number of masters with access to all peripherals |
| Performance | Fairly low—no simultaneous access | High—simultaneous access possible |
| Speed | Low—limited by highly capacitive bus | High—depends on partitioning of the multiplexer tree |
| Layout area | Solutions comparable | Solutions comparable |
| Modularity | Best | Not as good as tristate—highly depends on the number of masters requiring access to multiple peripherals |

**Bus system requirements.** The following architectural requirements guided the Viper's design and bus topology selection:

- The PR3940 I-cache and D-cache traffic must be separate from the DMA paths and TM32-based programmable I/O transactions. The PR3940 core must have a high-performance and low-latency path to memory.
- The TM32 I-cache and D-cache traffic must be separate from the DMA paths and PR3940-based programmable I/O transactions. The TriMedia core must also have a high-performance and low-latency path to memory.
- The MIPS core must have low-latency access on its local bus to the peripherals that the PR3940 CPU typically accesses.
- The TriMedia core must have low-latency access on its local bus to the peripherals that the TM32 CPU typically accesses.
- All registers in the peripherals must be accessible from the PR3940 CPU, the TM32 CPU, the PCI block, the boot block, and the EJTAG block.
- The PR3940 programmable-I/O traffic should be separate from that of the TM32 to provide low-latency interrupt handling.
- Peripherals typically used by the PR3940 core must be able to perform direct memory accesses on the PCI module. (This requirement supports DMA access of an external processor's system memory.)
- System security must support access to kernel area in memory by only trusted peripherals.

With the bus requirements nailed down, our next step was to choose the bus topology. However, before exploring different topologies, we studied both tristate and point-to-point bus implementations to find out which better suited our needs.

**Tristate versus point-to-point bus design.** A comparative study of tristate versus point-to-point bus implementations, as Table 1 outlines, shows that a point-to-point bus architecture is desirable for designs requiring high performance, simple testability, and reduced layout. One main prob-

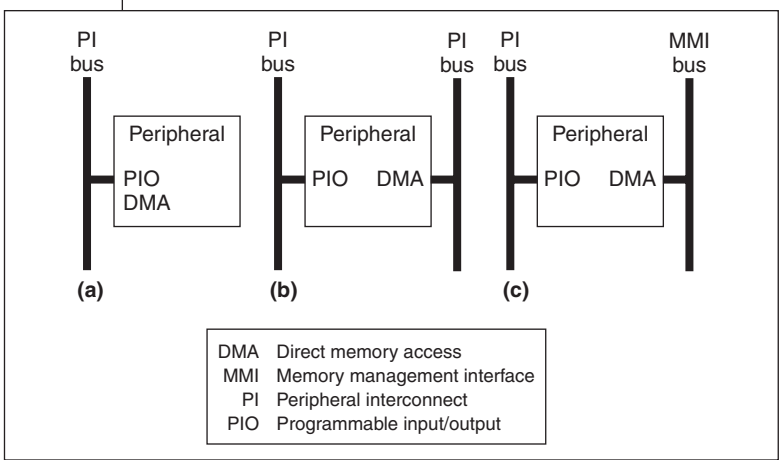


**Figure 2. Options for bus topologies: shared PIO and DMA (a), split PIO and PI DMA (b), and split PIO and MMI DMA (c).**

lem with this architecture, however, is that it does not easily allow for multiple-master access to peripherals. For example, if a peripheral requires access by four masters, the peripheral must have four slave interfaces; adding an additional master will require changes to the peripheral to support five masters. The point-to-point bus is not sufficiently modular or scalable.

For Viper, we decided to use a high-performance point-to-point, memory management interface (MMI) bus for bandwidth- and latency-critical access to the external SDRAM. For access to modules' slave (memory-mapped I/O) registers and for lower-bandwidth DMA peripherals, the tristate PI bus seemed more appropriate. Thus, we designed a point-to-point 64-bit MMI bus for the DMA connection between the high-bandwidth peripherals and the main memory. For the DMA connection between the low-bandwidth peripherals and the main memory, however, we used a tristated 32-bit PI bus. We designed the PI bus to also allow for the memory-mapped I/O register programming of both high- and low-bandwidth peripheral devices.

Even though there are different ways to convert a tristate bus to a multiplexed or wired-OR structure, we didn't explore these options for Viper. The reason is that Philips has extensively used the PI tristate bus and so had a large portfolio of various intellectual property (IP) modules already available with the required PI interface built in.

**Comparison of bus topologies.** High-bandwidth peripherals and peripherals requiring low-latency access to the external memory clearly require a direct interface to the point-to-point memory bus. For those peripherals not requiring high-bandwidth DMA, we evaluated the following architectural options, as Figure 2 illustrates:

- shared programmable I/O (PIO) and DMA on a common PI bus;
- split PIO and DMA on separate PI buses; and
- split PIO and DMA on the PI and the MMI bus, respectively.

For Viper modules that are not bandwidth hungry, we chose the first option: shared PIO and DMA on a common PI bus. We did this primarily because most of the existing portfolio of IP modules already supported this topology and, therefore, this option had the least risk.

Philips, however, is designing its future IP modules to support all three bus structures, letting the system architect decide which bus topology to use for a requirement. Separating PIO and DMA traffic clearly enables simpler and higher-performance system designs. Table 2 (next page) compares the three different bus topologies.

**Putting it all together.** We implemented Viper with two different bus types: a 64-bit point-to-point MMI bus (also called the memory or the DMA bus) and a 32-bit PI bus (specifically, a PI bus designed in accordance with Peripheral Interconnect Open Processor Initiative Standard 324).

The high-performance memory bus provides high-speed memory access to those on-chip units requiring high bandwidth and low latency. The bus connects to the external memory (SDRAM) via a 64-bit, 143-MHz main-memory interface that generates the required SDRAM protocol, but isolates the on-chip resources by using the proprietary DVP protocol. The memory interface also controls on-chip component access to the memory highway via a round-robin arbitration algorithm with programmable bandwidths.

Unlike the memory bus, the PI bus provides not only memory-mapped I/O (MMIO) reads

Table 2. Comparison of bus topologies.

| Parameter | Ease of implementation in three bus topologies: Shared PIO DMA | Split PIO and PI DMA | Split PIO and MMI DMA |
|---|---|---|---|
| Expandability (ease of adding/ subtracting peripherals) | Very good | Good | Average |
| Low-latency PIO | Average | Very good | Very good |
| Simplicity | | | |
| Read/write ordering and coherency | Good | Good | Good |
| Performance prediction | Average | Very good | Very good |
| Ease of understanding | Average | Very good | Very good |
| Implementation of system security | Good | Very good | Very good |
| DMA | | | |
| Latency | Good | Good | Very good |
| Throughput | Good | Good | Very good |
| Number of buses | Very good | Average | Bad |
| Layout | | | |
| Area | Very good | Good | Bad |
| Floorplanning | Good | Good | Bad |
| Clock system | Good | Average | Bad |
| Power routing | Average | Average | Good |
| Portfolio fit with existing designs | Good | Bad | Bad |

and writes to memory-mapped control registers of the various peripherals but also a medium-bandwidth DMA path via a bridged gateway connection to the MMI bus. The PI bus is divided into three different segments, so the chip features four bus segments or bus domains:

- MMI bus. The bus system's essential backbone, this bus connects the external SDRAM with all the on-chip agents for low-latency, high-throughput DMA access. There is no programmable I/O traffic on this bus.
- F-PI (fast PI) bus. The fast PI bus provides low-latency access to the memory and selected peripherals by the PR3940 CPU.
- M-PI (MIPS-PI) bus. This bus provides access to peripherals typically controlled by the PR3940 CPU.
- T-PI (TriMedia-PI) bus. This bus provides access to the peripherals typically controlled by the TM32 CPU.

Five bridges connect the MMI bus and the various PI bus segments, as Figure 1 shows. The FC-Bridge, the MC-Bridge, and the TC-Bridge act as gateways providing memory access to the corresponding PI segments, whereas the C-Bridge acts as a crossover PI-to-PI MMIO bridge that lets memory-mapped I/O access from each processor control or observe the status of all peripheral modules. The M-Bridge basically bridges transactions between the fast PI bus and the slower peripheral PI bus segments on the MIPS side.

Following an analysis of bus access patterns and bandwidth requirements, we appropriately sized the data buffers (memories) in the peripherals to handle the bus latencies. Large buffers in the peripherals have two advantages:

- Peripherals will not frequently access the bus, resulting in lower bus utilization and, therefore, a higher-performance system.
- Peripherals can handle fairly long latencies from the time they request the bus until the transaction is completed.

The Viper's M-PI and T-PI buses have average utilization rates below 5%, resulting in a high-performance system with low-latency PIO and DMA transactions.

## VLSI implementation

The Viper design complies with Digital Video Platform, Philips' scalable silicon-system architecture, which supports a wide range of digital video applications.

### RTL design

Not only did we reuse several of Philips' existing IP components in Viper, we also designed, quite early on, various standardized blocks and templates that would be needed to implement the peripherals. These blocks included interfaces for the MMI bus, PI bus, interrupt, debug, and test. They were designed such that they could be easily deployed and would enable layout and timing closure. Reusing predesigned standard blocks significantly reduced not only the number of bugs that we found during verification but also the top-level integration effort. We enforced a small set of coding guidelines, along with strict signal-naming conventions. Throughout the design, we also followed design styles recommended by experience—for example, no latches or asynchronous resets.

Per our reuse guidelines, we implemented

- high-bandwidth peripherals with a DMA interface according to the Philips DVP standard;
- low-bandwidth peripherals with a DMA interface according to the Philips PI-bus standard;
- MMIO interfaces according to the PI-bus standard;
- miscellaneous interface signals, such as reset, endianness, and interrupts, according to the DVP interface standard; and
- interface signals, wherever possible, with standard modules, such as the MMIO bus interface module, peripheral reset module, and peripheral interrupt module.

### Verification and emulation

We used both register-transfer- and gate-level simulations to achieve design verification. We also developed a regression suite, which we executed weekly. The runtime for a full regression was about 72 hours (with roughly 60 CPUs balancing the load in a load-sharing-facility-based queue).

We developed a small, block-level verification environment for fast test execution and debugging. Eventually, we transferred all tests to the system level and executed them in a full-chip environment. Standardizing the verification language let us transfer tests quickly from the block level to the system level.

To check design integrity over an unbounded number of simulation cycles, we also resorted to emulation: two emulation systems from IKOS Systems and two from QuickTurn. This enabled software-hardware codevelopment and was extremely useful for system-level verification. To ensure confidence in the design, we successfully ran all drivers and the required operating systems on the emulated design before tapeout.

### Design for testability

Viper is fully scan-testable, and its stuck-at-fault coverage exceeds 99%. We tested some of the larger CPU memories and caches with built-in self-test. In addition to the structural tests, we also planned system-level tests to ensure a high-quality design and to minimize the number of customer returns.

We achieved very high test coverage using test-shell isolation of each peripheral. This guaranteed that every peripheral in the design was completely testable in a stand-alone mode; in most cases, the designer was responsible for achieving fault coverage above 99%. We obtained test isolation by ensuring that each IP core input and output was both controllable and observable. Controllable inputs and observable outputs facilitate stand-alone as well as parallel testing of a core. Observable inputs and controllable outputs, on the other hand, allowed development of interconnect tests that verified bus connections between different IP cores. We tested buses and all interconnectivity between the on-chip peripherals and the CPUs with interconnect tests. These tests were relatively straightforward to implement because every peripheral contained a specially designed test shell.

## Physical design

We approached Viper's design synthesis from the bottom up, synthesizing the entire design in approximately eight hours, using multiple Synopsys licenses to speed up the process. We conducted chip-level synthesis whenever design modifications were required. In our design flow, scan test insertion followed the synthesis pass; thereafter, the netlist was partitioned and laid out. The design's high quality required multiple iterations of the synthesis-partitioning-layout loop.

During partitioning, we changed the logical hierarchy and created a new physical hierarchy. Partitioning divided the design into manageable-sized blocks, called chiplets. Closely related to top-level floorplanning, partitioning followed these guidelines:

- A chiplet's size should be no more than 200,000 layout instances (of standard cells, memories, and hard macros).
- There should be as few synchronous signal crossings between chiplets as possible; every attempt must be made to contain each clock domain in one chiplet.
- Tristate drivers and corresponding logic for the PI buses must be repartitioned and grouped to minimize bus loading.
- The clock module must be partitioned into a separate chiplet because of its complexity and sensitivity to crosstalk.

Viper was partitioned into nine chiplets. In addition, there were three hard macros: the TM32 CPU; the PR3940 CPU; and a custom analog block containing the phase-locked loops, direct digital synthesizers, and clock dividers.

Signals connected between the chiplets via abutment. There was no top-level routing, except for the clocks that had to be matched. We routed interchiplet signals between nonneighboring chiplets by inserting feed-through buffers in the chiplets that the signal had to route through. Following partitioning, the netlist was formally verified to be identical to the netlist after design-for-test insertion.

Figure 3 shows a block diagram of the floor plan. Arrows indicate locations of synchronous clock-domain crossings. Chiplets M-PI and T-PI contain the peripherals that sit on and are owned by the M-PI and T-PI buses. Floorplanning ensured the following:

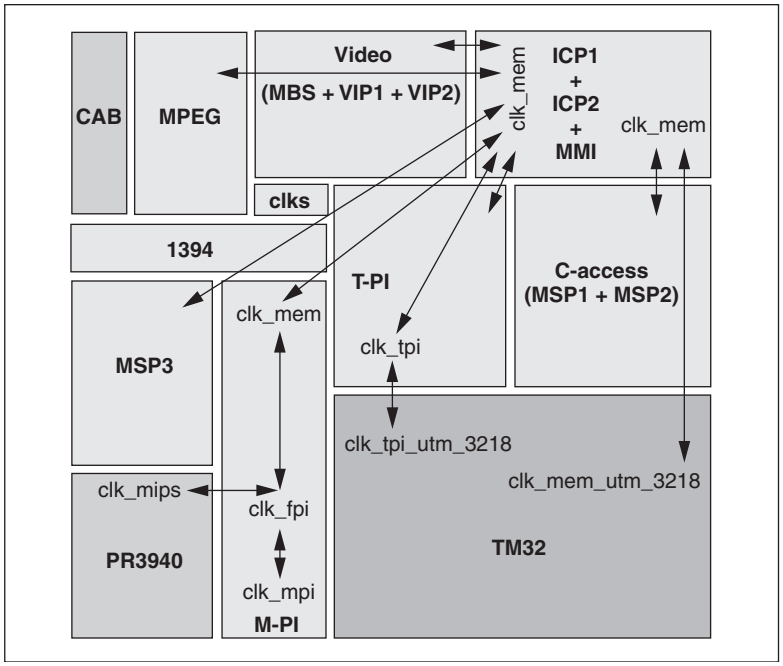


**Figure 3. Viper (PNX8500) floorplan, showing the nine chiplets. The plan also shows three hard macros: the TM32 CPU, the PR3940 CPU, and a custom analog block (CAB).**

- Each chiplet was routable and had the desired density.
- Top-level connections between chiplets were possible by abutment (following feed-through buffer insertion, Viper had approximately 30,000 top-level connections).
- Top-level and inter-chiplet timing closure could be achieved.

Viper's timing had to encompass both best- and worst-case operating conditions. Best-case conditions included a fast process, a 1.9-V supply at the package pins, and a 0°C junction temperature. Worst-case conditions included a slow process, a 1.7-V supply at the package pins, and a 125°C junction temperature. We achieved timing closure in two stages:

- Chiplet-level timing analysis included intra-chiplet timing data and constraints analysis, and I/O budget setup.
- Top-level timing analysis included inter-

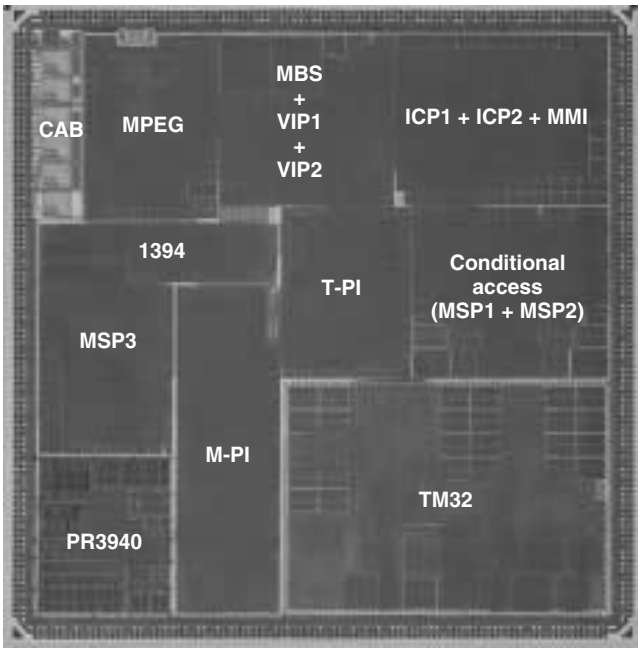


**Figure 4. Layout of Viper (PNX8500).**

**Table 3. Design statistics.**

| Parameter | Value |
|---|---|
| Process technology | TSMC 0.18 μm, six metal layers |
| Transistors | About 35 million |
| Instances | 1.2 million instances, or 8 million gates |
| Memories | 243 instances, 750-Kbit memory |
| CPUs | 2 (TriMedia TM32 and MIPS PR3940) |
| Peripherals | 50 |
| Clock domains | 82 |
| Clock speed | TM32: 200 MHz; PR3940: 150 MHz; SDRAM: 143 MHz |
| Power | 4.5 W |
| Supply voltage | 1.8-V core and 3.3-V I/O |
| Package | BGA456 |

chiplet timing, clock matching, and I/O timing analysis.

To achieve timing closure, we made engineering change orders to the netlist after routing. Following each manipulation step, formal verification ensured that the modified netlist was functionally equivalent to the one after test insertion.

We aligned all clock domains having synchronous chiplet crossings. For example, if the memory interface clock in one chiplet was synchronously connected to the same clock in another chiplet, we phase-aligned these clocks and analyzed the signal paths to meet timing constraints. We achieved clock alignment by tweaking the clock insertion delays, using aligners in the clock module. Similarly, we made the clock trees as structurally identical as possible.

As part of the physical design process, we met design completion and manufacturability goals by implementing techniques such as design rule checks, antenna fixes, track filling, and doubling of vias wherever possible. Figure 4 shows the layout plot for the Viper design's initial version.

Table 3 summarizes the major design parameters.

**WE HAVE LEARNED** much from the Viper design experience and trust it will guide us in the future, particularly since the next-generation SOC designs are significantly more complex, calling for still higher levels of integration. Some of our current activities, in addition to regular chip-development tasks, are investigating more efficient on-chip bus architectures and better design-reuse methodologies. ■

## Acknowledgments

We thank the Viper management and design teams for their hard work, particularly chief architects Gert Slavenburg and Lane Albanese, without whose foresight and leadership the project never would have been successful.

## ■ References

1. S. Rathnam and G. Slavenburg, "An Architectural Overview of the Programmable Multimedia Processor, TM-1," *Proc. 41st IEEE Computer Society Int'l Conf.* (COMPCON 96), IEEE CS Press, Los Alamitos, Calif., 1996, pp. 319-326.
2. D. Paret and C. Fenger, *The I2C Bus*, John Wiley & Sons, New York, 1997.

**Santanu Dutta** is a design engineering manager at Philips Semiconductors in Sunnyvale, California. His research interests include design of high-performance

video signal processing architectures, circuit simulation and analysis, and design and synthesis of low-power and testable digital systems. Dutta has a BTech in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, an MA in engineering from Princeton University, an MS in electrical and computer engineering from the University of Texas at Austin, and a PhD in electrical and computer engineering from Princeton University. He is a senior member of the IEEE.

**Rune Jensen** is a senior design engineer and project leader at Philips Semiconductors in Sunnyvale. His research interests include the design and implementation of multiprocessor SOC architectures for set-top box applications, design reuse methodologies, and bus subsystem design. Jensen has an MSc in electrical engineering from Aalborg University in Denmark.

**Alf Rieckmann** leads the IC architecture group in the IC Engineering Department of Broadband Home Servers at Philips Semiconductors in Sunnyvale. His research interests include system performance evaluation methodologies, SOC infrastructures, and video signal processing. Rieckmann has a diploma in electrical engineering from Universität Hannover, Germany. He is a member of the IEEE.

■ Direct questions or comments about this article to Santanu Dutta, Philips Semiconductors, 440 N. Wolfe Rd., M/S 80, Sunnyvale, CA 94088-3409; santanu.dutta@philips.com.

**For further information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.**