COVER FEATURE

Video Image Processing with the Sonic Architecture

Professional video image processing requires more computational power and data throughput than most general-purpose computers can provide. Sonic, a configurable computing system that performs real-time video image processing, meets these needs by using plug-ins to accelerate software applications.

Simon D. Haynes
John Stone
Sony Broadcast & Professional Europe

Peter Y.K. Cheung
Wayne Luk
Imperial College, University of London

Current industrial videoprocessing systems use a mixture of high-performance workstations and application-specific integrated circuits. However, video image processing in the professional broadcast environment requires more computational power and data throughput than most of today's general-purpose computers can provide. In addition, using ASICs for video image processing is both inflexible and expensive. For example, adding a new special effect can require designing one or more new ASICs, a time-consuming process that increases development costs.

Configurable computing offers an appropriate alternative for broadcast video image editing and manipulation. It combines the flexibility, programmability, and economy of general-purpose processors with the performance of dedicated ASICs.[1] An added advantage is that designers don't need to acquire major new skills because ASICs and configurable computing require nearly identical design knowledge and techniques. In fact, designers often use configurable logic to prototype ASICs before implementation.

Sonic is a commercially viable configurable computing system that performs real-time video image processing. This system can implement algorithms for two-dimensional linear transforms, fractal image generation, filters, and other video effects. Sonic's flexible and scalable architecture contains configurable processing elements that accelerate software applications and support the use of plug-in software.

CONFIGURABLE VIDEOPROCESSING PLATFORMS

The successful development of a configurable computer faces several challenges, including appropriate partitioning of algorithms between hardware and software, exploiting spatial and temporal parallelism, integrating the configurable computer into the software framework, and selecting a suitable configuration strategy. The architecture can significantly affect the performance of configurable computing systems in videoprocessing applications because of the high data throughput and processing requirements, as the "Videoprocessing System Requirements" sidebar indicates.

Hardware and software partitioning

In general, partitioning between hardware and software can be either automatic or manual. While researchers continue to explore automated partitioning and compilation of high-level languages,[1,2] developers are using a manual approach for many videoprocessing applications. Such an approach is attractive because the partitioning for video processing is usually well defined. In addition, hardware implementations are already available for many videoprocessing algorithms, and designers can easily transfer them directly to the configurable computing platform.

The frequency of hardware reconfiguration has a significant impact on the architecture. At one extreme lies frequent or dynamic reconfiguration, which exploits reconfigurability and allows reuse of hardware resources at different times for different operations. For videoprocessing, this could mean that device configuration occurs more than once per video frame. However, using frequent reconfiguration can be difficult in practice because a frequently configured device requires a high configuration bandwidth, and trans-

0018-9162/00/$10.00 © 2000 IEEE

Videoprocessing System Requirements

A typical videoprocessing system performs many functions, including sequence editing, format conversion, chroma-keying (which uses color to separate objects from their background), linear effects such as image rotation or resizing, and nonlinear effects such as mapping video to three-dimensional objects. Many underlying low-level operations such as antialias filtering, interpolation, and color space conversion are common to these types of functions.

The system needs to perform real-time videoprocessing for live editing or as fast as possible for offline editing. In some cases, editing material at faster than real-time speed is desirable. Table A demonstrates the enormous throughput that real-time processing of video in different formats requires.

With these throughput rates, even a relatively simple algorithm involves a large number of operations per second. For example, an HDTV video rotation operation, which consists of a high-quality antialias filter followed by an address remap and an interpolation filter, can require processing rates of around 10^{10} multiply-accumulate operations per second.

Image processing algorithms are highly suitable for hardware acceleration because they have spatial parallelism that makes it possible to process different parts of the image simultaneously. In addition, many of the operations that these algorithms use are simple. In many cases, exploiting temporal parallelism is easier because it isn't necessary to handle problems associated with segmenting an image. For example, kernel filters require information at the edges about parts of the image outside their segment. In addition to parallel processing, hardware pipelining is possible with many algorithms, thus increasing throughput.

Videoprocessing also requires an external video interface for the hardware acceleration platform. The platform can use this interface either for real-time video-streaming or for capturing and playing video sequences to the video server. In general, a Serial Digital Interface is preferred as most professional broadcast equipment supports it.

Table A. Throughput rates for real-time processing of video in different formats.

Format	Image size (pixels)	Frame rate (Hz)	Throughput (Mpixels/sec)
PAL	720 × 576	25.0	10.4
NTSC	720 × 480	29.97	10.4
HDTV (SMPTE 260M)	1,920 × 1,080	30.0	62.2

ferring the configuration data can incur an unacceptable time penalty. At the other extreme, a statically configured device simply performs an ASIC's function, wasting the flexibility that configurable devices offer. A practical compromise will likely lie somewhere between these two extremes: The configurable device performs an entire task with each different configuration, similar to using an ASIC, but reconfiguring the device changes the task and functions much like switching to a different ASIC.

Configurable versus custom designs

Designers must also decide whether to use an off-the-shelf configurable device or design a custom configurable one. Custom configurable devices have some of the same design disadvantages as ASICs, including long design times and high development costs. In contrast, the advantages of off-the-shelf devices include their high density and low cost because of mass production, the availability of a large set of mature design tools, and the potential for using coreware.

Other research[1] suggests that the gate capacity of current configurable devices can be a limitation. Although this constraint is becoming less important with the introduction of larger devices, using extra gates for replicating processing units normally improves videoprocessing algorithm performance. Therefore, the architecture should maximize the available logic by supporting multiple configurable devices.

The architecture should provide more memory than the inadequate amount that typical configurable devices have; for example, the latest Xilinx part, the XCV3200E, has only 106 Kbytes of BlockRAM. Adequate memory is particularly important for videoprocessing, which usually requires large frame stores. For example, a two-frame buffer for HDTV images requires 17 Mbytes of memory, assuming 4 bytes per pixel.

Other design elements

The architecture should include external video buses to overcome host bus limitations, which many researchers have found to be the system bottleneck. With a scalable architecture, different architecture implementations can match different users' requirements, and future expansion is possible. Finally, the architecture should allow easy exploitation of the spatial and temporal parallelism that is a common characteristic of videoprocessing algorithms.

SONIC ARCHITECTURE

The Sonic architecture demonstrates the practicality of using configurable computing to accelerate offline and real-time video image processing. As Figure 1 shows, the design consists of plug-in processing elements (PIPEs) connected by the PIPE bus and PIPEflow buses. Sonic's architecture exploits the spatial and temporal parallelism in video image processing algorithms. It also enables design reuse and supports the software plug-in methodology.

Bus architecture

Sonic's bus architecture consists of a shared global bus combined with a flexible pipeline bus. The architecture uses these buses to implement several different computational schemes.

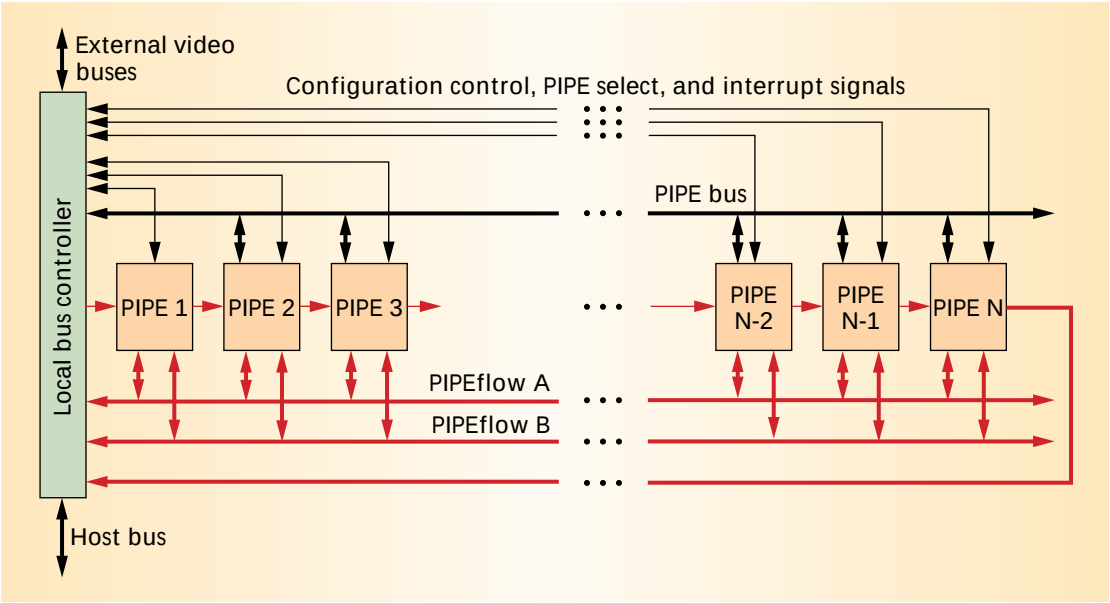

Figure 1. Sonic architecture. A shared global bus (PIPE bus) and flexible pipelined buses (collectively called PIPEflow buses) connect the plug-in processing elements (PIPEs). PIPEflow buses are shaded in red.

The synchronous, global PIPE bus's bandwidth matches the host bus's bandwidth. The system uses this bus for fast image transfer to the memory on the PIPEs, PIPE parameter access (runtime data required by the PIPEs), control of the PIPEflow bus routing through the PIPEs, and configuration of the PIPEs. Each PIPE has several unique signals that control configuration, interrupt signaling, and device selection.

The Sonic architecture uses the PIPEflow buses to implement pipelined operation. Data passes along the pipeline using the PIPEflow buses that connect adjacent PIPEs. PIPEflow buses A and B can transfer images from any PIPE or the local bus controller to any number of other PIPEs or the local bus controller; for example, PIPE 1 can use the PIPEflow A bus to send image data to PIPEs 2, 3, 4, and 6. The system uses a predefined raster-scan protocol to send data over these buses. The PIPEflow bus bandwidth matches the external video bus bandwidth.

PIPEs

The PIPEs are the most important part of Sonic's architecture because they perform the processing. Figure 2 shows the three PIPE parts: the PIPE engine (PE), PIPE memory (PM), and PIPE router (PR).

In Sonic, the PIPE engine handles computation and the PIPE router handles image data movement and formatting. The user application controls the PIPE engine; Sonic's Application Programming Interface (API) library, a set of well-defined functions, controls the PIPE router.

It is the PIPE router, and the way it is used, that make Sonic unique. The PIPE router provides a flexible and scalable solution to routing and data formatting. It is responsible for

- generating the PIPEflow-in data for the PIPE engine,
- allowing PIPE bus access to the PIPE memory, and
- handling the PIPEflow-out PIPE engine data.

When the system uses the PIPE router to generate PIPEflow-in data, the PIPE router performs three tasks to ensure that the PIPE engine receives the data in a format that it expects:

- *Data formatting.* The PIPE router ensures that the data is in the correct format for PIPE engine computation—packaged as part of a plug-in. For example, if one computation involves operations—with YCrCb components, an imaging format—and another operates using RGB components, the PIPE router pipelines the computations together, performing the automatic conversion between these two formats. The PIPE router also supports conversions between other formats.
- *Data routing.* The PIPE router routes data between the various buses, the PIPE engine, and the PIPE memory. This means that the PIPE engine can perform the same computation with different data flow. For example, an identical computation can be used as a single entity or as part of a larger chain of PIPEs, with the data coming from either the host computer (via the PIPE memory) or the external video buses.
- *Data accessing.* The PIPE router supplies the data to the PIPE engine in a variety of ways. It can use the normal horizontal raster-scan mode to carry out simple operations, such as gamma correction. Designers can use a combination of horizontal

and vertical raster-scan modes to implement two-pass algorithms. A more complicated "stripped" accessing greatly simplifies designing 2D filters and block processing algorithms. All of these programmable options simplify the task of developing Sonic's PIPE applications.

The PIPE engine processes the image. This is the only part of the PIPE that a particular user application directly configures. The user application contains the PIPE engine configuration data, which defines the configurable device's function. Although the PIPE engine typically uses the PIPEflow buses to get the data via the PIPE router, the PIPE engine has direct access to the PIPE memory. This is useful when the system needs to access the image randomly, as in the implementation of explosion effects, where the image appears to "shatter" as if blown apart.

The PIPE memory stores images. The design the PIPE engine implements can access the PIPE memory explicitly or the Sonic API can access it implicitly. The actual PIPE implementation can take many forms. Although it conceptually consists of three parts—the PIPE router, engine, and memory—the PIPE implementation could consist of just one device or even many devices. The original intention clearly is to use configurable logic, but a DSP processor or a customized ASIC could implement the PIPE engine or PIPE router.

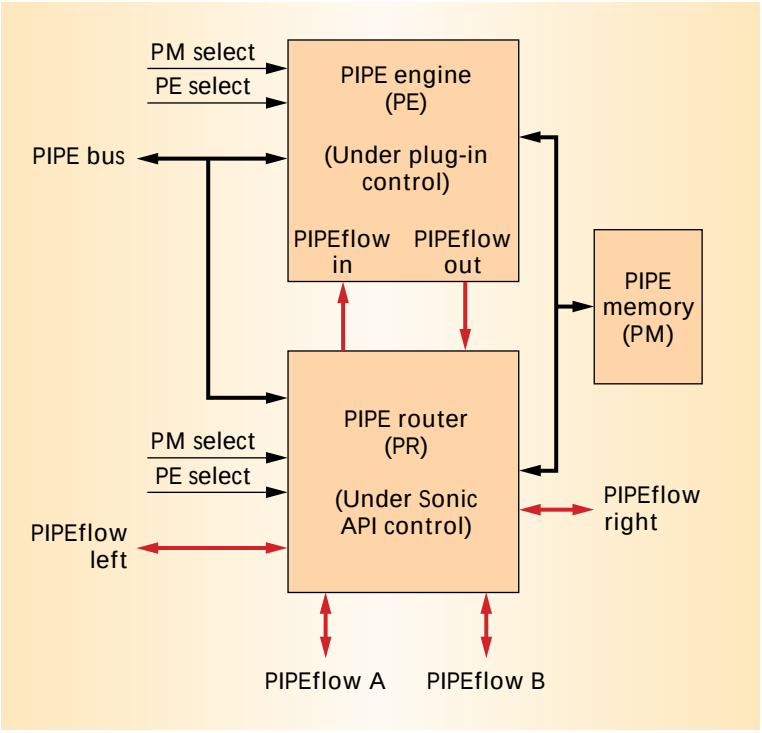

Figure 2. PIPE architecture. The PIPE engine implements the algorithm, the PIPE memory acts as an image store, and the PIPE router formats and moves data.

Exploiting parallelism

The Sonic architecture exploits the spatial and temporal parallelism present in video image processing algorithms. Sonic exploits spatial parallelism either within a single hardware design or by configuring multiple PIPEs with the same design and distributing parts of each frame across multiple PIPEs. Of course, this can require some subtle alterations to the individual designs, depending on the precise function. Sonic exploits temporal parallelism by configuring multiple PIPEs with an identical design, and then distributing successive frames across the PIPEs. This arrangement maximizes bus usage to overcome potential I/O bandwidth limitations. Sonic uses these techniques to trade off resources for speed: The more PIPEs available, the faster the operation.

Software plug-ins

Sonic's design provides software acceleration for a wide range of video applications, including those not written specifically for a configurable computer. Software plug-ins provide the necessary framework to accomplish this acceleration.

A software plug-in is a piece of code that application developers write to extend an application without recompiling the original application. Plug-ins have a well-defined interface that describes the function the application performs and specifies how data passes between the application and the plug-in. The application can invoke plug-ins such as filters and transforms as required. The application further benefits from the plug-in architecture's more structured code development style.

For configurable computing, software plug-ins allow applications to use hardware acceleration without requiring the main application to support the hardware. Indeed, developers can retrofit hardware acceleration after they write the application. For example, designers used configurable hardware and software plug-ins to accelerate Adobe Photoshop.[3] Software plug-ins also solve the problem of exploiting the hardware's configurable nature: The system can reconfigure the same configurable hardware to implement different plug-ins at different times.

Figure 3 shows the steps required for processing an image, what happens at each stage in Sonic, and where the API sits. The software plug-in contains a pure software implementation (used if Sonic is busy or not present) and a file that encapsulates the plug-in's hardware description. A hardware description file contains the configuration data for one or more field-programmable gate arrays (FPGAs). The microprocessor uses this software model to perform processing while the hardware platform processes an image. The software process goes through each step,

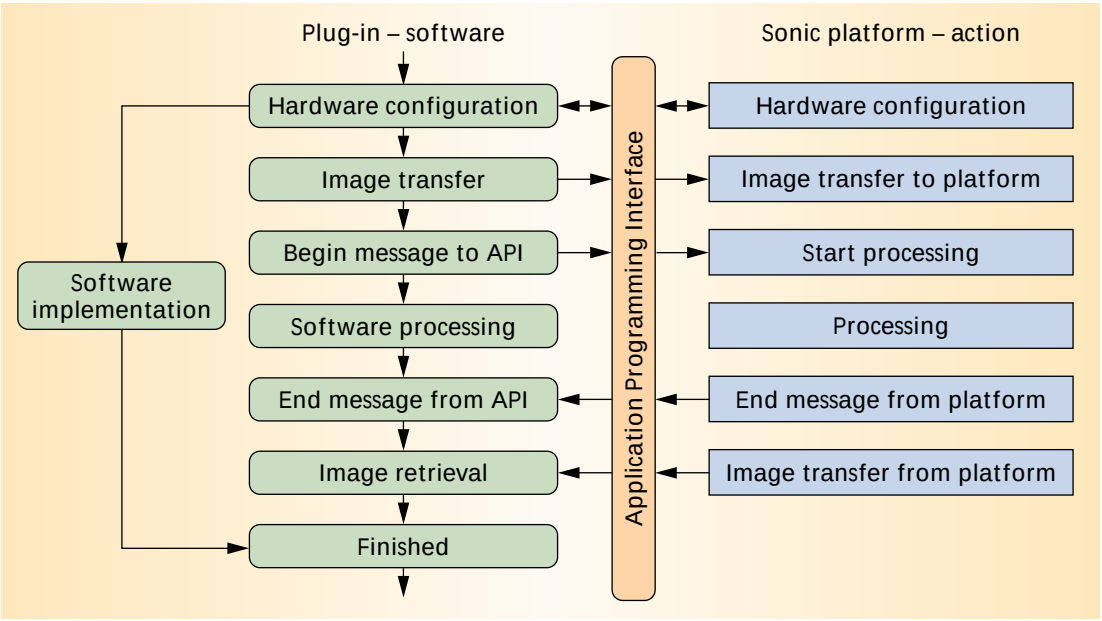

Figure 3. Software plug-in for video image processing. The plug-in contains a pure software implementation and a file that encapsulates the hardware description. Software carries out the steps in the rounded boxes on the left, and Sonic carries out the rectangular blocks in hardware.

calling the API where necessary. The API then causes a corresponding action in Sonic. The software always controls Sonic, but sometimes it must wait for Sonic to finish processing an image. This well-defined partitioning of the task between software and hardware enables the development of a highly efficient system architecture combined with a simple API.

Figure 4 shows how Sonic supports our software model. In this example, the Sonic platform contains three plug-ins. PIPE 1 contains a filter plug-in, which is currently unused (although PIPE 1 isn't used, it still remains configured; so if the API requires a filter plug-in, it doesn't need to reconfigure the PIPE). PIPE 2 implements a rotate plug-in for a paint application. The rotate plug-in's PE accesses the PIPE memory directly. PIPE 3 implements another filter plug-in, but this time it's for a video application. Sonic can adopt the same design for PIPE 1 and PIPE 3, potentially using the design for different applications. PIPEs 4, 5, and 6 show how multiple PIPEs can implement a larger plug-in, in this instance using the PIPEflow buses to pass the data. Designers can cascade smaller plug-ins that use the PIPEflow buses to create more complex plug-ins—an example of the Sonic architecture's design reuse capability. The remaining PIPEs are free to implement other plug-ins.

Design ease and reuse

Sonic's architecture emphasizes easy plug-in design and design reuse. Having a clear plug-in definition simplifies the process of designing new plug-ins. The PIPE router and the associated Sonic API ensure that the software plug-in and hardware correctly translate the image format, and they provide a mechanism for easily transferring images. This arrangement abstracts much of the Sonic architecture's detail from application developers, improving both their productivity and the portability of their designs.

The Sonic architecture accomplishes design reuse in three ways:

- Because the hardware has a rigid interface, plug-ins can use the same hardware designs for different applications. For example, the hardware for rotating images can accelerate rotation for any application.
- The PIPEflow processing model means that the system can use a hardware design on its own or as part of a larger design.
- Sonic can exploit temporal and spatial parallelism by using the same design to configure more than one PIPE.

ARCHITECTURE IMPLEMENTATION

Figure 5 shows Sonic-1, our current Sonic architecture implementation. We used daughterboard modules to implement PIPEs inserted into the 200-pin DIMM sockets on the main board. Sonic-1 supports eight PIPEs, the maximum we could physically fit on the board. Sonic-1 has a PCI interface to the host PC and an SDI interface for video equipment. The design's modularity offers several benefits:

- ease of development,
- improved device density on the board,
- ease of testing using a module with headers for a logic analyzer that we can insert in place of a PIPE, and
- potential for future expansion using different configurable devices in the PIPEs.

PIPE implementation

Because we currently use Altera parts that cannot be partially reconfigured, we placed the PIPE engine and PIPE router in separate devices: a FLEX10K100 for the PIPE engine and a FLEX10K50 for the PIPE router. We can clock the PIPE engine at 33 MHz or 66 MHz.

The PIPE memory consists of 4 Mbytes of SRAM arranged as 1M × 32 bits. The PIPE bandwidth memory is 132 Mbytes/sec, which matches that of the PIPE

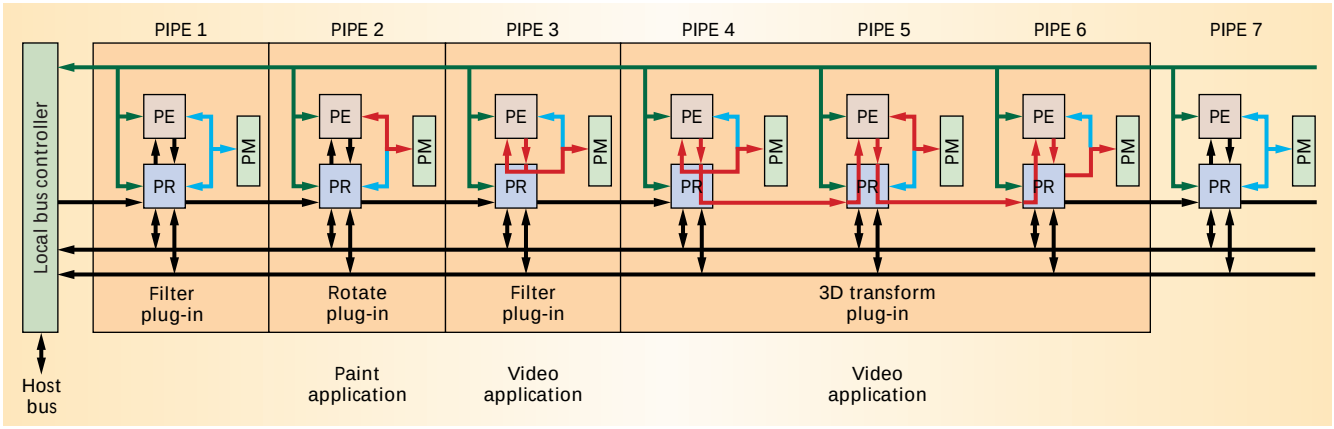

Figure 4. Sonic platform with three plug-ins. PIPEs 2 through 6 contain active plug-ins. Although PIPE 1 is not currently used, it is still configured so that the API doesn't have to reconfigure the PIPE if it needs a filter plug-in.

bus and PCI bus. We used SRAM as opposed to SDRAM or DRAM to simplify the hardware design process, especially for random memory accesses. We chose the memory size to fit two PAL/NTSC frames.

Bus implementation

Sonic implements the PIPE bus as a 32-bit multiplexed address and data bus with four control signals. The PIPE bus can match the maximum PCI bus bandwidth of 132 Mbytes/sec. The PIPEflow buses are 19 bits in width—16-bit data plus three control bits—and they operate at 66 Mbytes/sec. This bandwidth is half the PIPE memory bandwidth, so Sonic can read *and* store PIPEflow data to the same PIPE memory. Pin availability on the PIPE and the PR limits the bus size. Because it typically uses 8-bit RGBα data, this bus is time-multiplexed between RG and Bα components. The PIPEflow bus's data rate is above the rate that either PAL or NTSC real-time video data requires.

Sonic-1 contains hardware dedicated to smoothly interfacing the PIPEs to the Sonic API through the host PCI bus. The main Sonic board has two elements:

- *Local bus controller.* Two Altera 10K50s and a PLX 9080 implement the local bus controller interface with the PCI bus. The PCI bus transfers data between the host PC and Sonic-1. The PLX 9080 PCI interface chip gives a peak transfer speed of 132 Mbytes/sec from the host PC to the PIPE memory.
- *Serial digital interface.* The SDI input and output allow real-time processing on Sonic-1. SDI is widely used throughout the professional broadcasting industry to transfer real-time video data, bypassing the PCI bus.

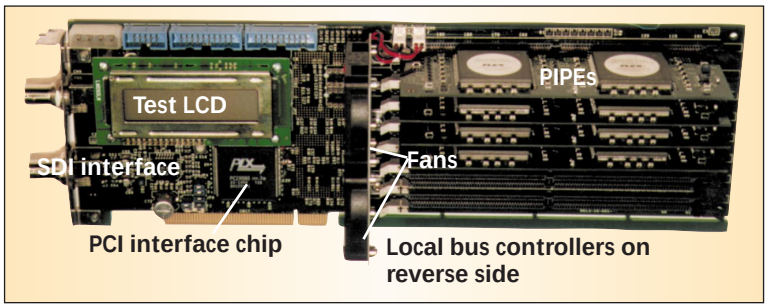

Figure 5. Sonic-1 platform. Four PIPES populate the eight PIPE slots.

COMPARISON WITH OTHER ARCHITECTURES

Our approach provides a competitive alternative to general-purpose computing platforms.[4] We designed the Sonic architecture to improve video image processing performance and to simplify the hardware design process. Table 1 provides a comparison with two other configurable systems—Wildfire[5] (Annapolis Micro Systems) and Riley-2[6] (developed jointly by Imperial College and Hewlett-Packard Laboratories)—that helps demonstrate some of Sonic's unique features. Wildfire is a commercially available configurable computing architecture based on the earlier Splash-2 image-processing platform.[7] In Wildfire, the host processor configures FPGAs independently of one another. The interconnection between FPGAs can only have one of 16 selected configurations, which

Table 1. Comparison of the Sonic architecture with other configurable architectures.

System	Generality	Scalable architecture	Supports software plug-ins in a multitasking environment	Data formatting and access mechanism
Sonic	Mainly image processing	Yes: Memory, routing, and configurable hardware	Yes	Yes
Wildfire[5]	High	No: Crossbar limits scalability	No: Requires API modification	No: Must be "designed in"
Riley-2[6]	High	No	No	No

Table 2. Results of Sonic-1 applications.

Task	PIPEs used	Speedup*	PAL frame rate (Hz)
19-tap symmetrical, separable 2D filter plug-in for Adobe Premiere (PCI)	1	5.5	0.5
3 × 3 FIR filter (PCI)	1	5.0	12.0
3 × 3 FIR filter (PCI)	2	8.8	21.1
2D image transform (PCI)	3	34.2	21.1
Color corrector (PCI)	2	2.6	21.1
3 × 3 FIR filter + color corrector (PCI)	6 (3 × 2)	13.9	21.1
Julia set generator (PCI)	8	9.9	36.5
Ripple effect generator** (PCI)	3	—	21.1
2D image transform, including antialias filtering (SDI)	5	—	25.0
Object tracker (SDI)	2	—	25.0

*Speedup data is based on a comparison of Sonic-1 and a pure software implementation on a PC containing dual 400-MHz Pentium II processors with MMX support.

**No optimized software version of the ripple effect has been written.

are set on power-up. Riley-2 is a configurable computing system for research that consists of four dynamically reconfigurable devices (Xilinx's XC6216) connected in a ring; each device has 512 Kbytes of SRAM. An i960 that interfaces to the host processor through a PCI bus controls the system.

Another approach to accelerating videoprocessing is using a digital signal processor such as the TriMedia from Philips or Texas Instruments' TMS320. DSPs have many attractive properties, such as software-based algorithm development and simplified debugging. Because DSPs and general-purpose processors have a similar underlying architecture that executes a sequence of instructions, algorithms that map well onto one will probably map well onto the other. In contrast, configurable logic allows a total shift in the computational paradigm that performs significantly better than a DSP or a general-purpose processor for dataflow-dominant algorithms. For example, although it requires a significant number of processor cycles, configurable hardware can achieve a simple byte-shuffle operation at almost no cost.

APPLICATION EXAMPLES

Table 2 shows the results using Sonic-1 to implement the sample algorithms. For each task, we used a machine with dual 400-MHz Pentium II processors with MMX support to compare the Sonic-1 implementation against an equivalent software implementation, except for the real-time processing applications and the ripple effect. We used a commercially available software implementation for the 3 × 3 FIR filter and the 2D transform; otherwise, we coded the examples using C++ with no optimization: For example, we didn't use MMX instructions. We used PAL-size images at 720 × 576 resolution.

We configured all the examples when we initiated the plug-in. The figures don't include the configuration time of about 150 milliseconds because it is negligible for long frame sequences. Sonic-1's processing time includes the time it takes to transfer the images to and from the platform over the PCI bus.

The results show that Sonic-1 provides marked speedup compared with pure software implementations for a variety of examples. The examples also demonstrate how easy it is to use more PIPEs to exploit temporal parallelism and increase the speedup. The currently available PCI bus bandwidth limits the speedup. The 3 × 3 FIR filter and color corrector examples show how the system can pipeline different designs together to give compound functions. The Adobe Premiere plug-in's poor performance is due to the time its software implementation takes to compress and uncompress each frame.

In addition to using Sonic-1 for software acceleration, we have successfully used the SDI to perform real-time processing. When we use Sonic-1 for real-time operation, it processes PAL video at 25 frames per second. We implement a 2D image transform for operations such as rotation, scaling, and shearing. The transform includes a nine-tap linear-phase antialias filter. We also implemented an object tracker that uses an 8 × 8-pixel search block area over a 63 × 23-pixel search area.

SONIC-1 LIMITATIONS

Sonic-1's ultimate performance as a software accelerator is currently limited by the PCI bus's performance. For example, with transfer rates of 132 Mbytes/sec—the maximum for a 32-bit 33-MHz PCI bus—the 2D image transform performs at 21.1 frames per second using three PIPEs. Without this limitation,

eight PIPEs would perform the same operation at 127 frames per second.

The memory size that we used limits us to processing PAL or NTSC size images on a single PIPE. It is possible to process larger image sizes if the algorithm allows splitting across multiple PIPEs.

The PIPEflow bus bandwidth of 66 Mbytes/sec also limits Sonic's real-time performance. Although this bandwidth is adequate for PAL/NTSC video, it falls short of the bandwidth that HDTV video requires for real-time processing.

Current Sonic work in progress includes a graphical debugging application, additional real-time video applications, and PIPE designs using larger configurable parts. We are also developing a Sonic Simulink compiler that assembles signal flow graphs into configurable hardware.

Possible future work includes developing a new Sonic architecture implementation to take advantage of the 66-MHz 64-bit PCI bus and extending advanced compilation techniques to rapidly generate efficient designs.[2,8] Sonic would be useful for applications involving both synthetic and real images.[9] As system-on-a-chip technology matures, it can use Sonic as a scalable architecture to integrate appropriate parameterized cores for reconfigurable applications. ❖

Acknowledgments

We thank Sony Broadcast & Professional Europe and the UK Engineering and Physical Sciences Research Council for their support.

References

1. W.H. Mangione-Smith et al., "Seeking Solutions in Configurable Computing," *Computer,* Dec. 1997, pp. 38-43.
2. M. Weinhardt and W. Luk, "Pipeline Vectorization for Reconfigurable Systems," *Proc. 7th IEEE Symp. Field-Programmable Custom Computing Machines* (FCCM 99), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 52-62.
3. S. Singh and R. Slous, "Accelerating Adobe Photoshop with Reconfigurable Logic," *Proc. 6th IEEE Symp. FPGAs for Custom Computing Machines* (FCCM 98), IEEE CS Press, Los Alamitos, Calif., 1998, pp. 236-244.
4. S. Guccione, "List of FPGA-Based Computing Machines," http://www.io.com/~guccione/HW_list.html.
5. B.K. Fross, D.M. Hawver, and J.B. Peterson, "Wildfire Heterogeneous Adaptive Parallel Processing Systems," *Proc. 12th Int'l Parallel Processing Symp./9th Symp. Parallel and Distributed Processing* (IPPS 99/SPDP 99), IEEE CS Press, Los Alamitos, Calif., 1998, pp. 611-615.
6. P.M. Mackinlay et al., "Riley-2: A Flexible Platform for Codesign and Dynamic Reconfigurable Computing Research," LNCS 1304, Springer-Verlag, Berlin, 1997, pp. 91-100.
7. P.M. Athanas and A.L. Abbott, "Real-Time Image Processing on a Custom Computing Platform," *Computer,* Feb. 1995, pp. 16-24.
8. M.B. Gokhale and J.M. Stone, "NAPA C: Compiling for a Hybrid RISC/FPGA Architecture," *Proc. 6th IEEE Symp. FPGAs for Custom Computing Machines* (FCCM 98), IEEE CS Press, Los Alamitos, Calif., 1998, pp. 126-135.
9. W. Luk et al., "Reconfigurable Computing for Augmented Reality," *Proc. 7th IEEE Symp. Field-Programmable Custom Computing Machines* (FCCM 99), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 136-145.

Simon D. Haynes is a research and development engineer at Sony Broadcast & Professional Europe. His research interests include algorithms and architectures for image processing. He received an MEng and a PhD in electronic and electrical engineering from Imperial College. Contact him at simon.haynes@adv.sonybpe.com.

John Stone is a research and development project manager at Sony Broadcast & Professional Europe. His research interests include algorithms and architectures for image processing and media creation. He received an MSc in communications and signal processing from Imperial College and a BSc in electrical engineering from Wits University in South Africa. Contact him at john.stone@adv.sonybpe.com.

Peter Y.K. Cheung is the deputy head of the Electrical and Electronic Engineering Department at Imperial College, University of London, where he is a reader in Digital Systems. His research interests include VLSI architectures for DSP and videoprocessing, reconfigurable computing, embedded systems, and high-level synthesis and optimization of digital systems, particularly those containing field-programmable gate arrays. He received a BSc in electrical engineering from Imperial College. Contact him at p.cheung@ ic.ac.uk.

Wayne Luk is a member of the academic staff in the Department of Computing, Imperial College, University of London. His research interests include theory and practice of customizing hardware and software for specific application domains, such as graphics and image processing, multimedia, and communications. His current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays. He received an MA, an MSc, and a DPhil in engineering and computing science from the University of Oxford. Contact him at wl@doc. ic.ac.uk.