

SONIC - A Plug-In Architecture for Video Processing

Simon D. Haynes*, Peter Y. K. Cheung*, Wayne Luk*, John Stone**

s.d.haynes@ic.ac.uk, p.cheung@ic.ac.uk, wl@doc.ic.ac.uk, john.stone@adv.sonybpe.com

Abstract

This paper presents the SONIC reconfigurable computing architecture and the first implementation, SONIC-1. SONIC is designed to support the software plug-in methodology to accelerate video image processing applications. SONIC differs from other architectures through the use of Plug-In Processing Elements (PIPEs) and the Application Programmer's Interface (API). Each PIPE contains a reconfigurable processor, a scaleable router that also formats video data, and a frame-buffer memory. The SONIC architecture integrates multiple PIPEs together using a specialised bus structure which enables flexible and optimal pipelined processing. SONIC-1 communicates with the host PC through the PCI bus and has 8 PIPEs. We have developed an easy to use API which allows SONIC-1 to be used by multiple applications simultaneously. Preliminary results show that a 19 tap separable 2-D FIR filter implemented on a single PIPE achieves processing rates of more than 15 frames per second operating on 512 x 512 video transferred over the PCI bus. We estimate that using all 8 PIPEs, we could obtain real-time processing rates for complex operations such as image warping.

1 Introduction

Reconfigurable platforms are often designed with little consideration for *how* the board will be used and for *what* purpose. The resulting platforms can be very general [1,2], but performance in specific domains can often be compromised in favour of overall modest acceleration. General platforms are also inherently more difficult to integrate into the software environment efficiently, requiring very general APIs. We recognise that perhaps the biggest issue in custom computing machines is one of software integration: developers will not use a platform, no matter how good it is, if the software interface is poor.

The other stumbling block for reconfigurable platforms is the process of mapping algorithms into hardware. Although there have been numerous attempts to compile the hardware automatically from a software description [3], the reality is that many people have found this to be difficult in practice [4]. We therefore believe that reconfigurable platforms should be designed to simplify the accelerating hardware design process as much as possible.

Our architecture, SONIC, is specifically targeted for video image processing tasks. Focusing the application domain in this way also means that greater acceleration can be achieved than would be the case for a more general architecture. This also simplifies the Application Programmer's Interface (API).

SONIC is more than just another reconfigurable platform. The SONIC architecture encompasses the complete software reconfigurable hardware environment. When designing SONIC our starting point was the software model. The SONIC architecture has also been developed to simplify the interface between the software and hardware. Firstly we give the software designer a simple, easy to understand software model. Secondly, the designer of the accelerating hardware is given as much abstraction from the detail of the implementation as possible.

The SONIC architecture uses the software plug-in architecture. The use of plug-in architectures for reconfigurable processing is not new[5], but the novelty of SONIC lies with the fact that the SONIC architecture was designed specifically for this programming methodology.

* Imperial College of Science, Technology and Medicine, London, England.

** Sony Broadcast & Professional Europe, Basingstoke, England.

2 Requirements of Video Image Processing

In order to develop a reconfigurable architecture suited to video image processing, it is first necessary to have an understanding of the requirements of typical video image processing tasks. Video image processing in this context means tasks such as image warping, chroma-keying of images, and effects such as image shattering, in addition to more typical examples of image processing, such as filtering, and edge detection.

It is well known that image processing, particularly video image processing, are suitable candidates for hardware acceleration. This is largely due to two reasons; i) large amounts of parallelism, and ii) relatively simple nature of the operations required.

Video image processing is typified by high data rates (187.5 Mbytes/sec for real time HDTV), making an efficient method of data transferral between host and platform important. The memory system must also be able to cope with the high data rates. Projects, such as the P³I [6], have emphasised the need for clean efficient memory system when handling video images.

The structure of many video image processing tasks can often be decomposed into pipelined sub-operations. An example is shown in Figure 1, where a separable 2-D FIR Filter has been implemented using two 1-D FIR Filters plus a 'Corner Turner', which transposes the image through 90°. [7] has also shown that pipelined processing and specialised datapaths are an important architecture feature for image processing. To give good performance, the SONIC architecture should be able to exploit this kind of pipelined, stream-based processing. Existing reconfigurable platforms such as Splash-2 [1] use this.

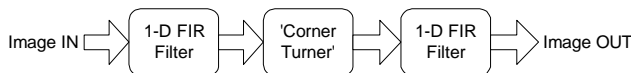


Figure 1 - A 2-D FIR Filter decomposed into sub-units

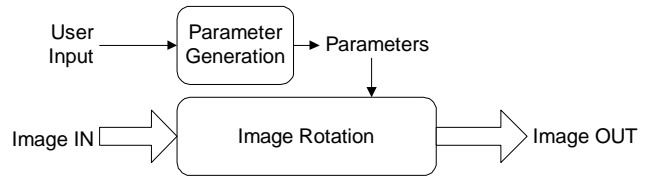


Figure 2 - Image data and parameter data flow

Video image processing tasks can also often be separated into two distinct information paths as shown in Figure 2: One is a high bandwidth datapath, the other a low bandwidth path. The high bandwidth path usually performs simple operations on the stream of image data, such as interpolation, the low bandwidth path providing the parameters for the operations. These usually originate from user input. For operations such as rotation or filtering the parameters can simply be one or two numbers. For more complex operations, such as image warping, they can be a number of vectors. The parameters may change from frame to frame, or over a single image. The generation of the parameters often requires floating point and other complex operations best suited to general purpose microprocessors.

3 The SONIC Software Model

Software plug-ins are becoming widely used in applications such as Adobe[®] Photoshop[®] and Adobe[®] Premiere[®]. This style of software architecture uses the main application code to implement the GUI interface, file handling, and other 'house keeping' tasks, whilst much of the application's functionality is accomplished by the use of plug-ins. The plug-ins perform such tasks as image rotation, or filtering. Each plug-in is invoked by the application as it is required.

A software plug-in architecture is particularly well suited to image handling software, since the interface is simple: The application gives the plug-in an image (usually in a well defined format), and then retrieves the resultant image after the processing has taken place. Software plug-ins are good for software design, since they allow for future extension of applications, encourage a more structured style of code development, and also allow third-party vendors to provide value added extensions. Plug-ins can also reduce the size of the main executable.

In order to make software acceleration a practical proposition for application developers, who are often only skilled in software engineering, it is necessary to disguise the fact a reconfigurable platform is being used. Hardware acceleration can be embedded within software plug-ins, without the application designer ever knowing. Indeed, hardware acceleration can be used in an application *after* the application has been written.

Figure 3 gives an overview of the structure of a SONIC plug-in. The plug-in consists of two parts; a) A software implementation of the task, and b) A hardware implementation of the task. Since both software and hardware implementations of the task are available, this would allow for a decision to be made at runtime as to which to use. The software implementation can also be used to validate the hardware design. The plug-in may contain multiple hardware descriptions for use in different implementations.

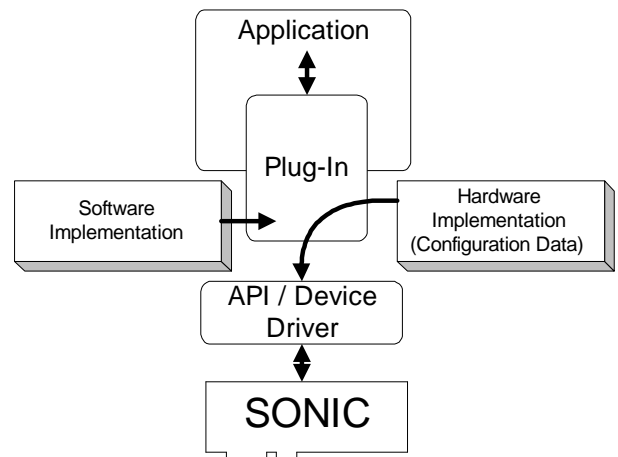


Figure 3 - A SONIC application plug-in

Figure 4 shows the SONIC architecture's software model. The Plug-In contains an hardware description file. This file encapsulates the hardware description of the plug-in. It typically contains the configuration data for an FPGA, although in a more complicated plug-in it could also contain (or point to) configuration data for multiple FPGAs, programs for DSPs etc. In addition it also contains information about how the devices are to be connect together to process an image.

The API includes functions which handle PIPE resource allocation and scheduling in a transparent way. For example PIPE caching implements the concept of virtual hardware, whilst minimising the overhead required for reconfiguration. The API also allows the plug-in to automatically use the software implementation should there be no free PIPE resources available.

We believe that the actual hardware design for the plug-in will usually have to be done by a hardware designer. This is because the hardware generated from a software description tends to be inefficient at present, and in order to gain

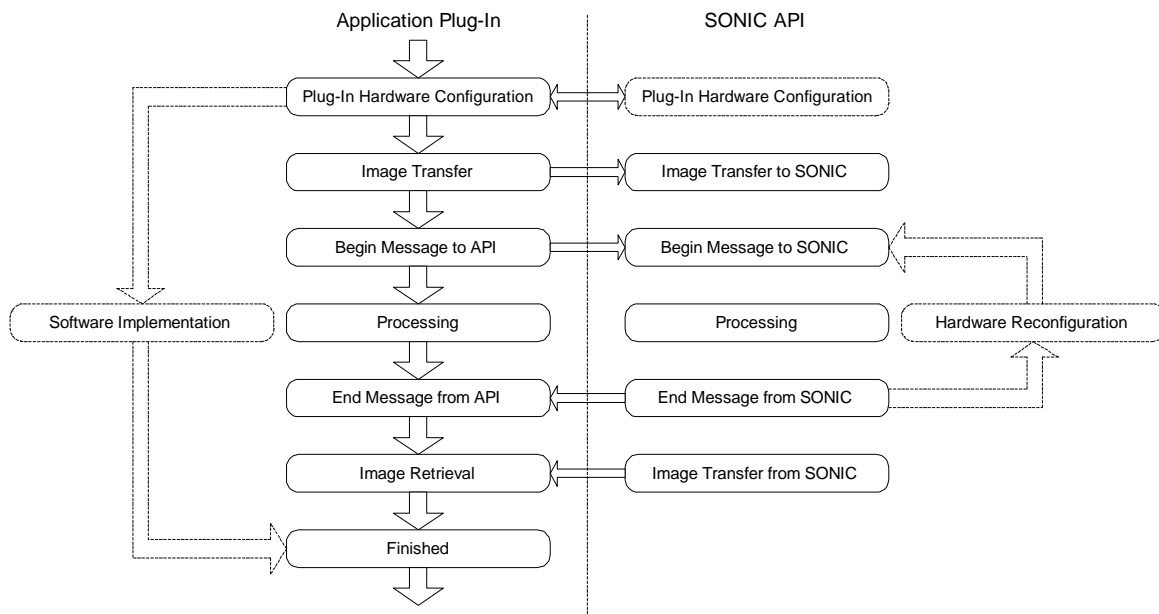


Figure 4 - Software Model for SONIC

the best from hardware, it is sometimes necessary to recast the problem into a different form - making it difficult to accomplish automatically. In addition we envisage different types of FPGA devices being used on SONIC, and most of the methods for hardware generation from a software description are targeted to a specific devices.

We certainly do not preclude automatic hardware generation from a software description, it is simply that we recognise the need for making SONIC simple to understand for anyone designing plug-in hardware.

4 Architecture of the SONIC Platform

SONIC was designed with the following characteristics, which follow the plug-in methodology, and allow for efficient processing of images:

- Support for the SONIC architecture software model.
- Give the plug-in hardware designer abstraction from the detail of the platform.
- Support for pipelined processing.
- Allowing multiple plug-ins to be implemented simultaneously.
- Scalable processing, memory, and interconnect.
- Efficient and simple memory model.

The overall SONIC architecture consists of a number of Plug-In Processing Elements (PIPEs), connected by the PIPE bus, and PIPEFlow buses. Figure 5 gives an overview of the SONIC architecture.

4.1 The SONIC Bus Architecture

SONIC's bus architecture consists of a shared global bus combined with a flexible pipeline bus. This allows the SONIC architecture to implement a number of different computational schemes.

The PIPE Bus

The PIPE bus is a synchronous, global bus which should be matched to the bandwidth of the host bus. The purpose of the PIPE Bus is:

Fast Image Transfer - The PIPE Bus can be used for fast image transfer to or from the memory on the PIPEs, using the Host bus.

PIPE Parameter Access - Any run-time information required by the PIPEs can be transferred using the PIPE bus. Used in this way, the PIPE bus implements the parameter path in shown Figure 2.

Control of the Routing - The PIPE Bus is used to instruct each PIPE where to route the PIPEFlow buses (SONIC's flexible pipelined routing).

Configuration of the PIPEs - In some implementations, where a high bandwidth is required for configuration, the PIPE Bus can be used to carry configuration data for the PIPEs.

PIPE Control Signals

Each PIPE has a number of unique control signals, these are used for configuration control, interrupt signals, as well as part of the protocol for the PIPE bus.

PIPEFlow Buses

Since pipelined operation has been found to be important in video image processing, the SONIC architecture uses PIPEFlow buses. They are designed to allow pipelined operation. Data passes along the pipeline using the PIPEFlow buses connecting adjacent PIPEs. The PIPEFlow Start bus can be used to get data to the start of the pipeline, and the PIPEFlow End bus to retrieve the data from the end. Data is sent over these buses using a pre-defined 'raster-scan' protocol. Depending on the implementation, the PIPEFlow buses may use one or more pre-defined protocols.

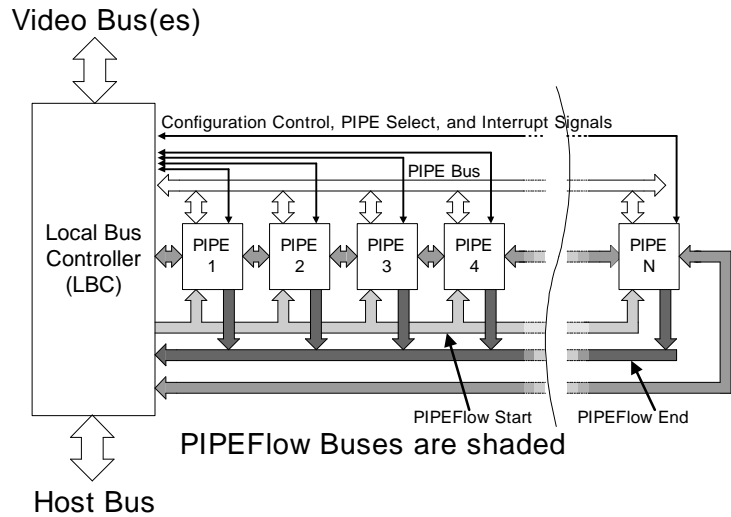


Figure 5 - The SONIC architecture

4.2 The PIPE (Plug-in Processing Element)

The PIPEs are the most important part of the SONIC platform architecture. They are the elements which perform the processing. Each PIPEs consists of the three conceptual parts shown in Figure 6: the PIPE Router (PR), PIPE Engine (PE), and PIPE Memory (PM).

The architecture of the PIPEs means that computation, handled by the PE, is separated from the movement and formatting of the image data, which is handled by the PR. The PE is controlled by the plug-in, and the PR by the API. It is the PR, and the way that it is used, which makes SONIC unique.

The PIPE Router (PR)

The PR provides a flexible, scalable solution to routing and data formatting by the SONIC architecture. The PR is responsible for three tasks:

- Accessing the PM by PIPE Bus.
- Generating the PIPEFlow In data for the PE.
- Handling the PIPEFlow Out data from the PE.

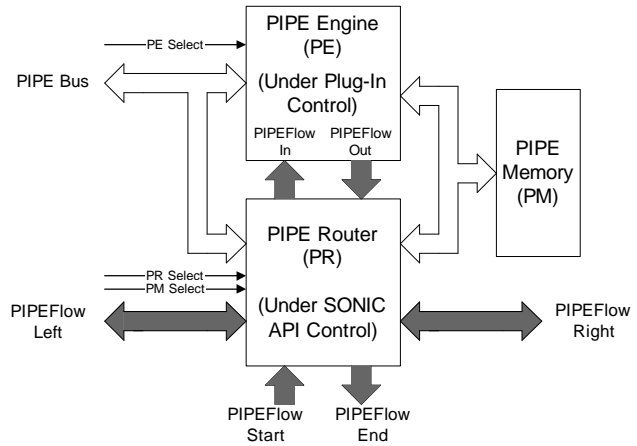


Figure 6 - Architecture of the PIPE

The PR is much more than a simple router and memory handler. The SONIC architecture uses the PR to present the image data to the PE in the format in which the plug-in hardware expects it. There are three elements to this:

Data Locations - The PR must route the data from the correct place. Not only can the PR route the data from one of the PIPEFlow buses, but PIPEFlow data could also be routed to or from the PM. This means that *precisely the same* plug-in can be used either as a single entity, with it's data coming from the PM, or as part of a larger chain of processing with the data coming from the previous PIPE.

Data Format - The PR is responsible for ensuring that the data are in the correct format for the plug-in in the PE. For example, if the plug-in in the PE is designed to operate with Hue, Saturation and Volume (HSV) components and is pipelined to the previous PIPE which outputs RGB components, the PR must perform the HSV to RGB conversion. The PR could also support conversions from formats such as 4:2:2 or 4:1:1 sampled YCrCb, or even de-interlaced interlaced frames.

Data Access - The PR is capable of supplying the data to the PE in a variety of ways, as shown in Figure 7. Simple operations, such as gamma correction, can be carried out using the normal horizontal raster scan mode. The normal vertical raster scan mode, allows for designers to easily implement two-pass algorithms. The more complicated 'stripped' accessing greatly eases the design of 2-D Filters, and block processing algorithms.

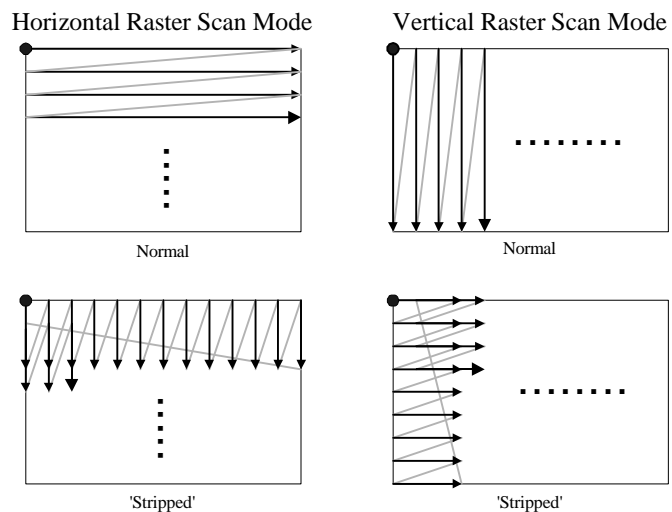


Figure 7 - Different Raster-Scan Modes of the PR

The PIPE Engine (PE)

The PE processes the image. This is the only part of the PIPE directly configured by the plug-in. The plug-in hardware description contains the configuration data for the PE. Although the PE typically gets the data via the PIPEFlow bus, the PE has direct access to the PM. This allows the plug-in hardware designer to have complete control over how the PM is accessed, if required. This is useful for situations where the image must be accessed randomly (explosion effects, for example).

The PIPE Memory (PM)

Each PIPE contains memory (PM), which can be used for image storage and manipulation. If the plug-in hardware designer does not use the PM, the SONIC architecture allows the PR to use the PM for image storage, through the API.

4.3 Different implementations of the PIPE

The actual implementation of the PIPE could take many forms. Firstly, despite conceptually consisting of three parts (the PR, PE & PM), the implementation of the PIPE could consist of just one device or even many devices. Secondly, although the original intention is clearly to use reconfigurable logic, the PE and/or PR could be implemented with a DSP processor, or customised ASIC, with programming code replacing hardware configuration data.

5 Integration of the SONIC Platform Architecture with the software model

Figure 8 shows how the SONIC platform architecture complements our software model. In this example the SONIC platform has two PIPEs configured, and locked by their plug-ins. In this instance plug-In 1 uses the PR to generate the data from the PM, whilst Plug-In 2 is accessing the PM directly. The remaining unused PIPEs are free to be used to implement more plug-ins as required.

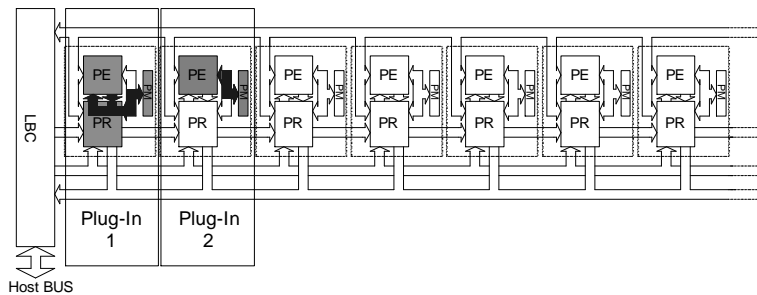


Figure 8 - SONIC Platform with 2 Plug-Ins

Figure 9 shows what happens after several events:

1. Plug-In 1 has finished, and unlocked the PIPE.
2. Plug-In 3 has started, locking 3 PIPEs

Although Plug-In 1 has unlocked its PIPE the PIPE still remains configured. This means that if Plug-In 1 should restart then there is no need for the API to reconfigure the PIPE. Because of this, the API attempts to use the least recently used PIPE, when a plug-in tries to start. Plug-In 3 shows how a larger plug-in can be implemented using multiple PIPEs., in this instance passing data via the PIPEFlow buses. More complex plug-ins can easily be designed by cascading smaller plug-ins which use the PIPEFlow buses.

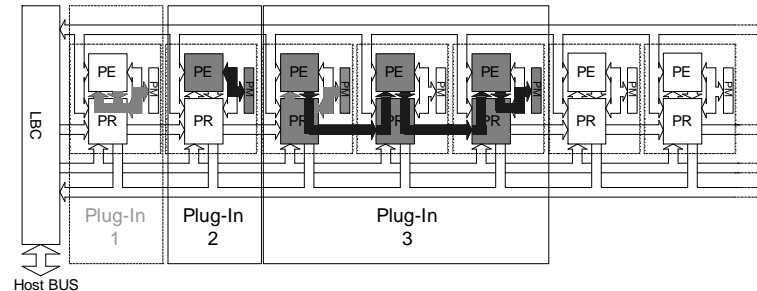


Figure 9 - Plug-In 1 Unlocks and Plug-In 3 Starts

6 Implementation of the SONIC architecture

A photograph of our implementation of the SONIC architecture, SONIC-1, can be seen in Figure 10. We implement the PIPEs using daughter board modules, which can be inserted into the 200 pin DIMM sockets on the main board. The modularity of the design is beneficial for several reasons:

- Easier development.
- Improved device density of the board.
- Easier testing (a board with headers for a logic analyser was made could be inserted in place of a PIPE).
- Allowed for future expansion, by allowing for different devices to be used in the PIPEs

An LCD display has been included on the board to improve the testability of the board.

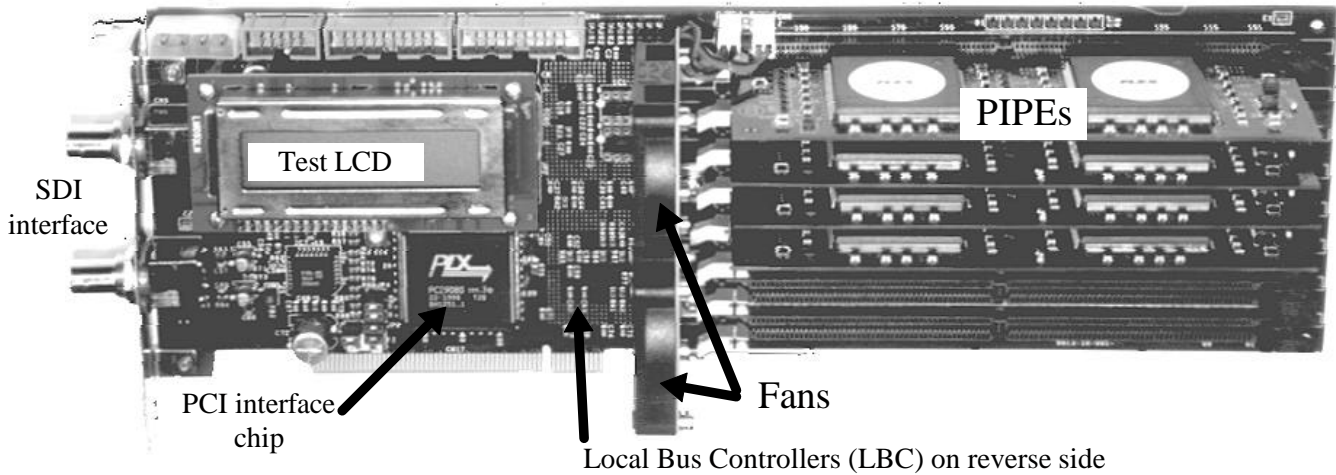


Figure 10 - SONIC-1 platform, with 4 PIPEs

6.1 Implementation of the PIPE

A block diagram of our implementation of the PIPE is shown in Figure 11. Since we use Altera parts, which cannot be partially reconfigured, it was necessary to place the PR and PE in separate devices (A FLEX10K70 for the PE, and FLEX10K20 for the PR). The 10K70 can be clocked at 33 or 66Mhz. Figure 12 shows a photograph of our implementation.

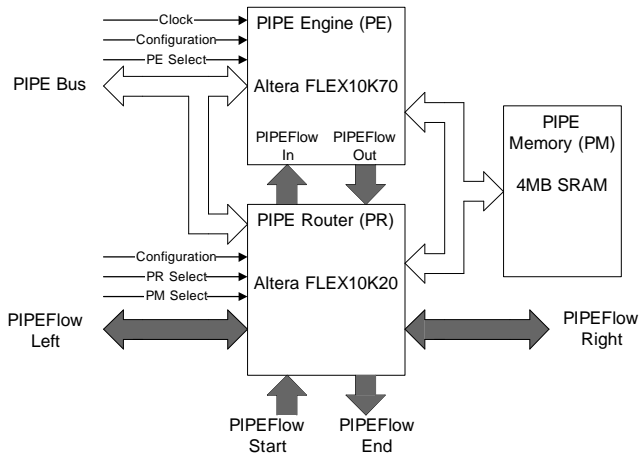


Figure 11 - Our implementation of the PIPE

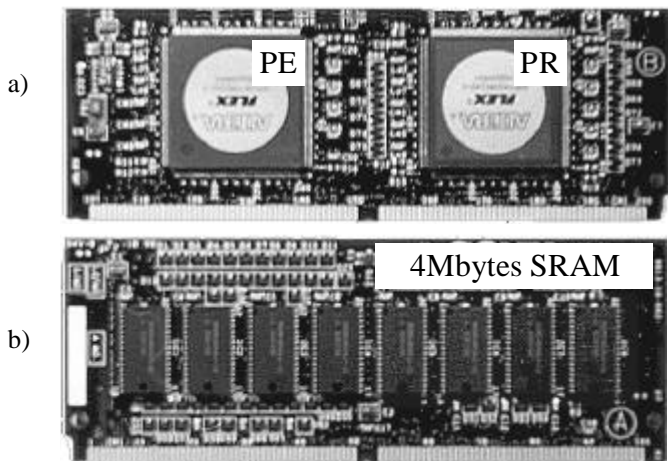


Figure 12 - detail of a) front, and b) reverse of PIPE

The PM consists of 4Mbytes of SRAM arranged as 1M x 32 bits. The bandwidth of the PM is 132MB/s which matches that of the PIPE Bus, and is twice that of the PIPEFlow bus.

There are also 22 bit bi-directional connections between the PEs of adjacent PIPEs (utilising the remaining device pins), which can be used when multiple PIPEs are combined as MEGA PIPEs.

6.2 Implementation of the buses

The PIPE Bus

The PIPE Bus is implemented as a 32 bit multiplexed address/data bus (plus 4 control signals). It is capable of matching the maximum bandwidth of the PCI bus (132MB/s).

The PIPEFlow bus

The PIPEFlow bus are 19 bits in width (16 bit data + 3 control bits) and operate at (66MB/s). This bandwidth is half that of the PM, so it is possible to read *and* store PIPEFlow data to the same PM. Pin availability on the PIPE and the PR placed the limitation on the size of this bus. Because 8 bit RGB α data is typically used, this bus is time multiplexed between RG & B α components.

6.3 Support Architecture

SONIC-1 contains hardware dedicated to smoothly interfacing the PIPEs to the SONIC API through the host PCI bus. It also contains an SDI interface which can be used as an image data stream interface independently from the PCI bus. The elements of the main SONIC board are:

Local Bus Controller (LBC)

The LBC was implemented using 2 Altera 10K50s, and a PLX 9080 to interface with the PCI bus. The PCI bus transfers data between the host PC and SONIC-1. The PLX 9080 PCI interface chip can support burst mode transfers, giving a maximum theatrical transfer speed of 132MB/s from the Host PC to the PIPE PM.

SDI Port

SONIC-1 also has a Serial Digital Interface (SDI) plus supporting logic, which can be used simultaneously as an input and output for video independently of the PCI bus. This interface is widely used throughout the professional broadcasting industry. This allows for pipelined processing of video, with the video using the SDI interface, and the PCI bus being used for control data. Transferral of images to and from the host PC is also possible, with the SDI port.

7 Example - A Separable 2-D FIR Filter for Adobe[®] Premiere[®]

In order to demonstrate how easy the SONIC architecture is to use, we give an example of implementing a separable 19 Tap 2-D FIR Filter for Adobe[®] Premiere[®]. A 2-D separable filter can be implemented using 2 1-D FIR Filters, processing once in the horizontal direction, and once in the vertical direction. Rather than use 2 filters we use a single 1-D FIR Filter twice. The basic design can be seen in Figure 13.



Figure 13 - Hardware Design for the Separable 2-D FIR Filter

The important point here is how little the hardware designer needs to know about the SONIC platform. All the information they require is that they will receive a stream of data through the PIPEFlow IN port, and must send the processed data out using the PIPEFlow OUT port.

The fragment of the 'C' code which handles the SONIC platform can be seen in Figure 14. This code configures a PIPE with the hardware shown in Figure 13. Runs the data through it horizontally once, and then vertically. *Precisely* the same hardware design for the PE can be used for both, since the PR accesses the data.

```

UINT hPIPE;          //Handle to the PIPE
DWORD Done;         //Bit 1 is high when finished processing

Sonic_Conf(&hPIPE, "SEP_2D_FIR_FILTER.RBF"); //Allocates a PIPE, configures it if necessary, and locks it.
Sonic_PR_ImageSize_Write(hPIPE, Width,Height); //Set the width and height of the image.
Sonic_PR_Route_Write(hPIPE, PR_TO_AND_FROM_PM); //Get the PIPEFlow data from the PM and put it back there when done.
Sonic_PM_Write(hPIPE, pSrcImage); //pSrcImage Points to the source image, write it to the PM.

Sonic_PR_ImageMode_Write(hPIPE,PR_HORIZONTAL_RASTER); //Make the PR read the image using horizontal raster scan.
Sonic_PR_Pipeflow_Write(hPIPE,PR_PROCESS); //Start the PR generating the rasterscan.
do {
  Sonic_PR_Pipeflow_Read(hPIPE,Done); //Wait for the PE to finish processing.
} until (Done & 1);

Sonic_PR_ImageMode_Write(hPIPE,PR_VERTICAL_RASTER); //Make the PR read the image using vertical raster scan.
Sonic_PR_Pipeflow_Write(hPIPE,PR_PROCESS); //Start the PR generating the rasterscan.
do {
  Sonic_PR_Pipeflow_Read(hPIPE,Done); //Wait for the PE to finish processing.
} until (Done & 1);
Sonic_PM_Read(hPIPE,pDstImage); //Read the resultant image back to pDstImage.
Sonic_Unlock_PIPE(hPIPE); //Unlock the PIPE, so other plug-ins can use it.

```

Figure 14 - Software Code Fragment required to handle SONIC

Figure 15 shows the plug-in within Premiere®. The usefulness of having the API only configure the PE when strictly necessary is highlighted by this plug-in. Adobe® Premiere® loads the plug-in in for each *frame*. However, because the API leaves the PIPE configured when the plug-in finishes, the PIPE is only configured once.

We ran Adobe® Premiere® on a 300MHz Pentium II machine using a sequence of 50 576x461 frames. The results can be seen in Table 1.

The time taken to process the sequence can be split into two times: *Processing Time* - the time actually spent in the plug-in, and *Framework Time* - the time which Adobe requires to prepare each frame (since the frames are stored in a compressed format). The processing speed up is 30 times, although the adobe framework overhead has reduced the total speedup to 5.5 times. When using the SONIC platform in other applications where no such compression takes place, we would expect to see ≈30 times speedup overall. To improve the speedup for Adobe® Premiere®, we intend to explore the possibility of using SONIC to accelerate the frame compression/decompression.

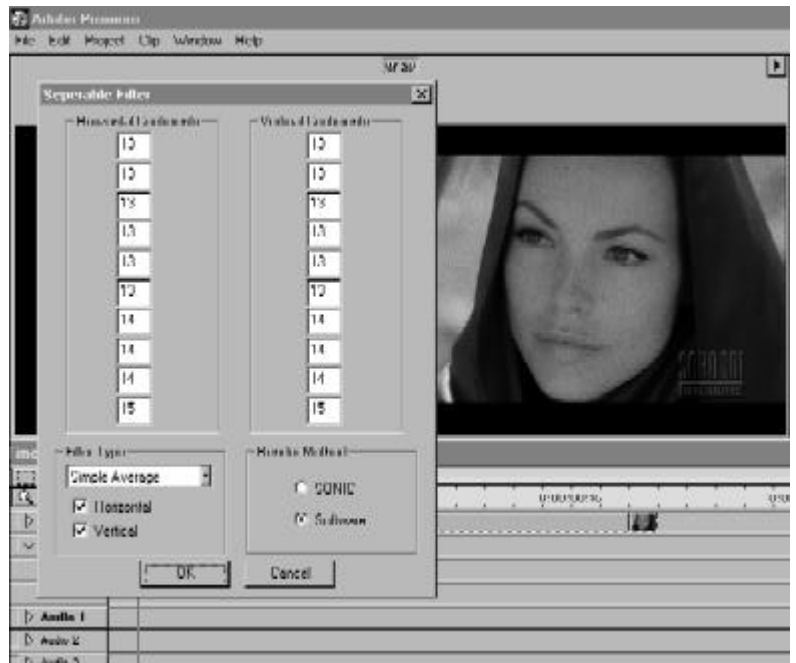


Figure 15 - The Separable 2-D FIR Filter in Adobe® Premiere®

	Processing Time (PT)	Adobe Framework Time (AT)	Total Time = PT+AT
SONIC-1	3.8s	21.6s	35.4s
Software	117.4s	21.6s	139.0s
Processing Speed-Up	30.9x		
Total Speed-Up			5.5x

Table 1 - Performance of the Separable 2-D Filter

Another important fact is that this plug-in only used a single PIPE. Assuming a linear speed up as more PIPE are used (which would be the case if the PIPEFlow buses were used for image movement), a more complicated Plug-In using all 8 available PIPES would expect to achieve speedups of around 250 times.

8 Conclusions

The uniqueness of the SONIC architecture is due to the PIPE Router (PR) and the API. The API enables resource allocation and scheduling which is invisible to the API user, whilst conforming to a simple software model. The PR simplifies the reconfigurable hardware design process, by carrying out all the image transferral and conversion necessary to give the PIPE Engine (PE) the correct data in the correct format. The SONIC architecture also demonstrates:

- the advantages of designing a reconfigurable platform with a well defined software model;
- that a Plug-In software methodology is particularly suited to reconfigurable platforms;
- simple simultaneous use of reconfigurable hardware by multiple applications;
- that PIPEs can be pipelined together to create complex plug-ins;
- good flexibility and expandability.

We have demonstrated that our implementation (SONIC-1) gives impressive performance, and have used the software plug-in methodology to write SONIC-1 plug-ins for Adobe® Premiere®. The development of plug-ins for other software packages is underway.

Other current and future work which we would like to carry out includes; building a library of hardware components which can be used by the designers for the basis of new designs; developing more benchmark plug-ins for various applications; refining of the design flow, and improving the API to give enable more sophisticated scheduling of the reconfiguration of the PIPEs.

Acknowledgements

We gratefully acknowledge the support provided by the UK Engineering and Physical Sciences Research Council, and Sony Broadcast & Professional Europe.

References

- 1 P.M. Athanas and A.L. Abbott, "Real-Time Image Processing on a Custom Computing Platform", IEEE Computer, Vol. 28, Issue 2, pp 16-24, Feb1995.
- 2 P.I. Mackinlay, P.Y.K. Cheung, W. Luk and R.D. Sandiford, "Riley-2: A flexible platform for codesign and dynamic reconfigurable computing research", Field-Programmable Logic and Applications, W. Luk, P.Y.K. Cheung and M. Glesner (editors), LNCS 1304, Springer 1997, pp. 91-100.
- 3 D. Galloway, "The Transmogripher C Hardware Description Language and Compiler for FPGAs", IEEE Symposium on FPGAs for Custom Computing Machines, 1995, pp. 136 - 144.
- 4 R.D. Hudson, D.I. Lehn, & P.M. Athanas, "A Run-Time Reconfigurable Engine for Image Interpolation", IEEE Symposium on FPGAs for Custom Computing Machines, April 15th - 17th, 1998, pp. 88 - 95,
- 5 S. Singh and R. Slous, "Accelerating Adobe Photoshop with Reconfigurable Logic", IEEE Symposium on FPGAs for Custom Computing Machines, April 15th - 17th, 1998, pp. 236 - 244.
- 6 M.J. Colaitis, J.L. Jumpertz, B. Guérin, B. Chéron, F. Battini, B. Lescure, E. Gautier, & J-P. Geffroy: "The Implementation of P³I, a Parallel Architecture for Video Real-Time Processing: A Case Study", Proceedings of the IEEE, Vol. 84, No. 7, pp 1019-1037, July 1996.
- 7 H.T. Kung, "Computational Models For Parallel Computers", Scientific applications of Microprocessors, R.J. Elliot & C.A.R. Hoare, Prentice Hall, pp 1-15, 1989.