A Reconfigurable Platform for Real-Time Embedded Video Image Processing

N.P. Sedcole, P.Y.K. Cheung, G.A. Constantinides, and W. Luk

Imperial College, London SW7 2BT, UK.
pete.sedcole@imperial.ac.uk, p.cheung@imperial.ac.uk,
george.constantinides@ieee.org, w.luk@doc.ic.ac.uk

Abstract. The increasing ubiquity of embedded digital video capture creates demand for high-throughput, low-power, flexible and adaptable integrated image processing systems. An architecture for a system-on-a-chip solution is proposed, based on reconfigurable computing. The inherent system modularity and the communication infrastructure are targeted at enhancing design productivity and reuse. Power consumption is addressed by a combination of efficient streaming data transfer and reuse mechanisms. It is estimated that the proposed system would be capable of performing up to ten complex image manipulations simultaneously and in real-time on video resolutions up to XVGA.

1 Introduction

As advances are made in digital video technology, digital video capture sensors are becoming more prevalent, particularly in embedded systems. Although in scientific and industrial applications it is often acceptable to store the raw captured video data for later post-processing, this is not the case in embedded applications, where the storage or transmission medium may be limited in capacity (such as a remote sensor sending data over a wireless link) or where the data are used in real-time (in an intelligent, decision-making sensor for example). Real-time video processing is computationally demanding, making microprocessor-based processing infeasible. Moreover, microprocessor DSPs are energy inefficient, which can be a problem in power-limited embedded systems. On the other hand, ASIC-based solutions are not only expensive to develop, but inflexible, which limits the range of applicability of any single ASIC device.

Reconfigurable computing offers the potential to achieve high computational performance, at the same time remaining inexpensive to develop and adaptable to a wide range of applications within a domain. For this potential to become of practical use, integrated systems will need to be developed that have better power-performance ratios than currently available FPGAs, most likely by curtailing the general applicability of these devices such that optimisations for the particular application domain can be made. Such systems may be termed 'domain specific integrated circuits'.

This paper outlines a proposed architecture for an integrated reconfigurable system-on-a-chip, targeted at embedded real-time video image processing. The

P.Y.K. Cheung et al. (Eds.): FPL 2003, LNCS 2778, pp. 606–615, 2003.

system is based on the Sonic architecture [1], a PCI-card based system capable of real-time video processing. Our objective is to integrate this system into a single device for embedded video processing applications.

In this paper we identify and discuss the unique issues arising from large scale integration of reconfigurable systems, including:

- How the complexities of designing such systems can be managed. Hardware modules and abstraction levels are proposed to simplify the design process.
- The implications of modularisation on the allocation and management of resources at a physical level.
- Effective connectivity and communication between modules.
- Minimising power consumed in data transmission and data reuse, the two most important factors in low-power design of custom computational platforms [2].

2 Related Work

Reconfigurable custom computing machines, implemented as arrays of FPGAs, have been successfully used to accelerate applications executing on PCs or workstations [3, 4]. Image processing algorithms are particularly suitable for implementation on such machines, due to the parallelisms that may be exploited [1, 5]. As mentioned in section 1, the system described in this paper is based on the Sonic architecture [1], a video image processing system comprising an array of FPGAs mounted on a PCI card.

Research has been conducted into embedded reconfigurable systems, and integrated arrays of processing elements [6–9]. Often these are conceived as accelerators of software-based tasks, and as such are closely coupled with a microprocessor, with the microprocessor forming an integral part of the data-path. As a consequence, these systems are usually adept at exploiting instruction-level parallelism; task-level parallelism is often ignored.

The proposed system has similarities to the DISC [10], which allows relocatable reconfigurable modules to vary in size in one dimension, and also includes the concept of position-independent connections to global signals. The modules in the DISC are very simple however. Our proposed system also incorporates ideas similar to the 'dynamic hardware plugins' proposed by Horta *et al.* [11] and virtual sockets described by Dyer *et al.* [12], although in both of these cases communication and interconnect structures are designed ad hoc on an application by application basis. Kalte *et al.* describe a system-on-a-programmable chip which does include a structured interconnection architecture for connecting dynamically reconfigurable modules [13]. Their system connects multiple masters to multiple slaves using either a multiplexed or crossbar-switch scheme. The interconnection scheme described in our paper allows any module to be a bus master, and is more suitable for streaming data transfers.

3 Managing Design Complexity

As device densities increase, circuit designers are presented with huge amounts of uncommitted logic, and large numbers of heterogeneous resources such as memories, multipliers and embedded hard microprocessor cores. Creating manageable designs for such devices is difficult; attempting to exploit dynamic reconfiguration as well only increases this complexity.

In order to make the design process tractable, we propose a modularised, hierarchical system framework, based on Sonic [1]. Modularisation partitions the system design into conceptually manageable pieces. In addition, high-level algorithmic parallelism can be exploited by operating two or more modules concurrently. To simplify the design, use and reuse of these modules, module interfaces need to be standardised, and abstractions are required in the transfer of data between modules.

3.1 System Architecture

As depicted in Figure 1, modularity is achieved by separating the data path into a variable number of processing elements connected via a global bus. Each processing element (PE) also has unidirectional *chain bus* connections to the adjacent PEs, for fast local data transfer. The left-most and right-most PEs are specialised input and output streaming elements respectively; data stream through the system in a general left-to-right direction.

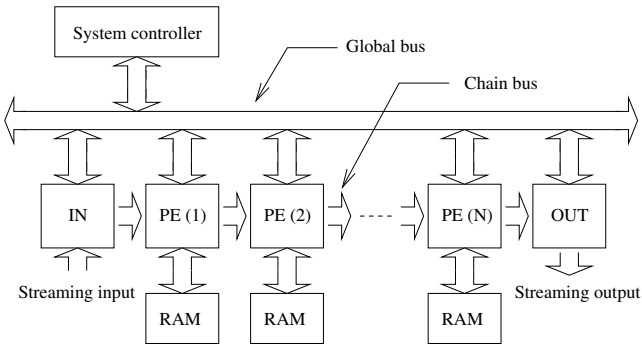

Fig. 1. The proposed system architecture.

Each element in the data path is designed to realise a complex task, such as a filter (2D convolution) or a rotation, implemented physically as a partial configuration instanced within a reconfigurable fabric. The processing applied to the data stream is determined by the type, number and logical sequence of the processing elements. The system designer controls this by programming a microprocessor within the system control module. PE module configurations (stored

as bitstreams) are loaded into the fabric and initialised by the control software, which also directs data-flow between PEs. As will be discussed below, the actual transfer of data is handled by a router within each PE. This scheme allows the implementation of a range of resource allocation and reconfiguration scheduling techniques, while realising the advantages of a data-path driven design.

Video processing algorithms can require a large amount of storage memory, depending on the frame size of the video stream. While available on-chip memory is constantly increasing, the size of storage space necessary, coupled with the bit-density (and therefore cost-per-bit) advantage of memory ICs, will ensure that external RAM will remain a necessary feature of video processing systems. The traditional memory bottleneck associated with external memory is avoided in our case by distributing the memory between the processing elements. The connection mechanisms between each PE and memory is discussed further in section 4.

3.2 Processing Elements

The logical composition of a processing element, as shown in Figure 2, comprises an engine, a router and input and output stream buffers. All of these are constructed from resources within the reconfigurable fabric. The PE has connections to the global bus (for communication between any two PEs), to the directly adjacent PEs to the left and right, and (optionally) to two external RAM banks. These interfaces are all standardised, which enables fully developed processing element configurations to be used and reused unchanged within a larger system design.

The core component of each PE is the processing engine; it is in this component that computation is performed on the image pixels. The engine is also the only part of the PE that is defined by the module designer. Serialised (raster-scan) data flow into and out of the engine by way of the buffers, the relevance of

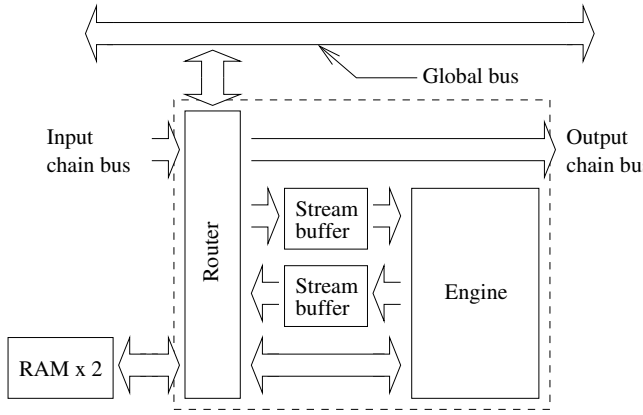

Fig. 2. The structure of a processing element.

which will be discussed in section 5. The key component for data movement abstraction is the router; this is directed by the system controller to transfer data between the buffers and the global bus, the chain bus or external memory. Since the data format is fixed, and all transfer protocols are handled by the router, the module designer is only concerned with the design of the computational logic within the engine.

4 Physical Structure

The mechanisms described in the previous section for controlling design complexity have ramifications for the physical design of the system. As mentioned above, the processing element modules are instanced within a reconfigurable FPGA fabric. The structure of this fabric needs to be able to support variable numbers and combinations of various sized modules. Since modules are fully placed and routed internally at design-time, the completed configurations must be relocatable within the fabric. The provision of mechanisms for connecting PE configurations to the buses and external RAM is required.

The physical system structure is illustrated in Figure 3. The processing elements are implemented as partial configurations within the FPGA fabric. The PEs occupy the full height of the fabric, but may vary in width by discrete steps. The structure and nature of the reconfigurable fabric is based on the Virtex-II Pro FPGA family from Xilinx, Inc. It is heterogeneous, incorporating not only CLBs but RAM block elements and other dedicated hardware elements such as multipliers. However, it exhibits translational symmetry in the horizontal dimension, such that the PEs are relocatable along the length of the fabric. The

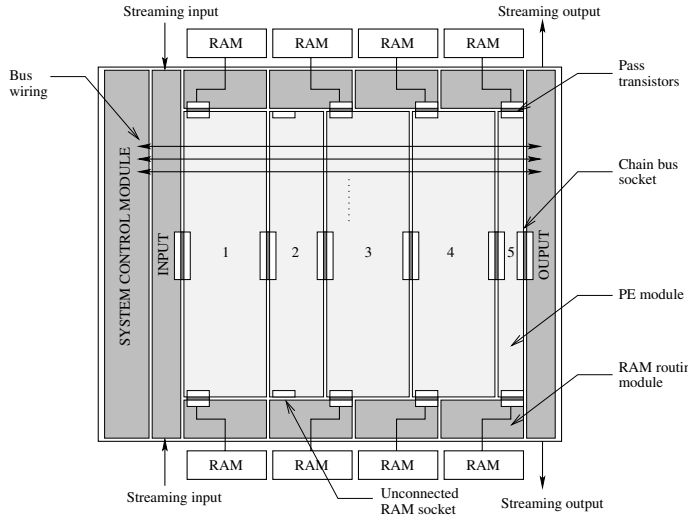

Fig. 3. A diagram representing the physical structure of the proposed system, with the reconfigurable fabric (shaded light grey) configured into five processing elements.

choice of a one-dimensional fabric simplifies module design, resource allocation and connectivity.

The global bus is not constructed from FPGA primitives; it has a dedicated wiring structure, with discrete connection points to the FPGA fabric. The advantage of this strategy is that the electrical characteristics of the global bus wiring can be optimised, leading to high speeds and low power [14, 15]. In addition, the wiring can be more dense than could otherwise be achieved.

Each processing element must have chain bus connections to the neighbouring PEs; this is accomplished through the use of 'virtual sockets', implemented as hard macros. The chain bus signals are routed as 'antenna' wires to specified locations along the left and right edges of the module. When two configurations are loaded adjacently into the array, these wires are aligned, and the signal paths may be completed by configuring the pass transistors (programmable interconnect points) separating the wires. Thus, each module provides 'sockets' into which other modules can plug into. Similar ideas has been proposed previously [11, 12], although the connection point chosen in previous work is a CLB programmed as a buffer.

A similar concept is used in connecting processing modules to the external RAM banks. Since the processing elements are variable-sized and relocatable, it is not possible to have direct-wired connections to the external RAM. The solution to this is to wire the external RAM to 'routing modules' which can then be configured to route the RAM signals to several possible socket points. This allows the registration between the RAM routing module and the processing element to be varied by discrete steps, within limits. If external RAM is not required by a particular processing element, such as PE 2 in Figure 3, the RAM resources may be assigned to an adjacent PE, depending on the relative placements.

5 Data Transfer and Storage

The transfer and storage of data are significant sources of power consumption in custom computations [2], so warrant specific attention. In the preceding system, Sonic, data transfer is systolic; each clock cycle one pixel value is clocked into the engine, and one is clocked out. This limits the pixel-level parallelism possible within the engine, and constrains algorithm design. In particular, data reuse must be explicitly handled within the engine itself, by storing pixel values in local registers. This becomes a significant issue for the engine design when several lines of image data must be stored, which can total tens of kilobytes.

In the proposed architecture the input stream buffer efficiently deals with data reuse. Being constructed from embedded RAM block elements (rather than from CLBs) a high bit density can be achieved. Image data is streamed into the buffer in a serial, FIFO-like manner, filling it with several lines of a frame. The engine may access any valid pixel entry in the buffer; addressing is relative to the pixel at the front of the queue. Buffer space is freed when the engine indicates it has finished with the data at the front of the queue. This system enables

greater design flexibility than a purely systolic data movement scheme while constraining the data access pattern sufficiently to achieve the full speed and power benefits of serial streaming data transfer. This is particularly beneficial when data are sourced from external RAM, where a sequential access pattern can take advantage of the burst mode transfer capability of standard RAM devices.

The input and output stream buffers are physically constructed from a number of smaller RAM elements for two reasons. Firstly, a wide data-path bit-width between the buffers and the engine can be achieved by connecting the RAM elements in parallel, enabling several pixels to be processed in parallel within the PE. The second important benefit is the ability to rearrange the input buffer RAM elements into two (or more) parallel stream buffers, when the engine requires more than one input data stream, such as in a merge operation. Likewise, the output buffer may be subdivided into several output streams, if the engine produces more than one output. We label each stream buffer input or output from an engine a 'port'.

In addition to allowing efficient data reuse and fine-grained parallelism, the stream buffers create flexibility in the transfer of data over the global bus. Instead of a systolic, constant rate data-flow, data can be transfered from an output port buffer of one PE to the input port buffer of another PE in bursts, which allows the global bus to be shared between several logical communication channels. The arbitration between the various logically concurrent channels is handled by a reconfigurable arbitration unit within the system controller. This enables a range of arbitration strategies to be employed depending on the application, with the objective of preventing processing stalls from an input buffer under-run or output buffer overrun.

6 Performance Analysis

In the previous sections the proposed system was described in qualitative terms. We will now present a brief quantitative analysis to demonstrate how the system is expected to meet the desired performance requirements. To do this, it is necessary to make some assumptions about the sizes of various system elements; these assumptions will be based on the resources available in the Xilinx Virtex II Pro XC2VP125, and comparison with the latest incarnation of the Sonic architecture: UltraSONIC [16]. The numbers given here are speculative and do not represent any attempt at optimised sizing. Nevertheless, they are sufficient for the purpose of this analysis.

Based on the utilised logic cell count of modules in UltraSONIC and the cell count of the XC2VP125, between four and ten processing elements are possible in the current technology, depending on the complexity of each PE. Assuming the same I/O numbers as the Xilinx device (1200) and given the physical connectivity constraints of the system topology (see Figure 3) it is estimated that external RAM would be limited to eight banks, implying that not every PE would have access to external RAM. It is theoretically possible to implement 272 bus lines spanning the full width of the XC2VP125, however we will assume a bus width

of only 128 bits. The Xilinx device includes 18 columns of Block RAMs, each of which can be arranged as 1024 bits wide by 512 deep memory. Therefore assigning each processing element two 32KB stream buffers is not unreasonable, perhaps configured as 512 wide by 512 deep. A wide buffer facilitates pixel-level parallel computations, although for algorithms that do not exhibit much low-level parallelism a wide buffer would be a disadvantage.

External RAM sizes and system throughput is determined by the video format the system is applied to. Table 1 gives some figures for some representative video formats. In previous work, external RAM was sized such that one video frame would fit one RAM bank [16], which would imply between 330KB to over 3MB per bank in this case.

Using these figures, some calculations on the processing characteristics of the system can be made. With a 512-bit wide buffer, 16 pixels (at 32-bits/pixel) can be operated on in parallel. As Table 2 indicates, several complete lines of image data can be stored in the stream buffers for data reuse. An example engine clock rate is given, assuming each block of 16 pixels is accessed 10 times during the computation, a realistic value for a 2D convolution with a 5x5 kernel. This clock rate is at least an order of magnitude below the state-of-the-art for this technology, which is highly desirable as low clock frequencies correspond with lower operating voltages and very little pipelining, which all translate into lower power consumption. Table 2 also gives the required clock rate of the global bus, for five concurrent channels, all operating at the full throughput rates given in Table 1. Again, these speeds should be easily achievable, especially given that it is a custom structure and not constructed from FPGA primitives.

These calculations demonstrate that the proposed system is expected to be able to perform between four and ten complex image computations simultaneously and in real-time on video data of resolutions up to XVGA.

Table 1. A sample of video formats. Frame sizes are based on 32 bits per pixel, while throughput is calculated at 25 frames per second.

Format	Lines	Columns	Frame size	Throughput
PAL	288	352	330 KB	9.67 MB/s
DVD PAL	576	720	1620 KB	39.55 MB/s
XVGA	768	1024	3072 KB	75.00 MB/s

Table 2. System characteristics. The cycle time is the average time allowed to process 16 pixels in parallel. The transfer capacity of the bus is calculated assuming a 10% overhead for arbitration and control.

	Processing element			Global bus	
Format	Lines/buffer	Cycle time	Clock rate	Transfer capacity	Bus speed
PAL	23	6.3 μs	1.6 MHz	53 MB/s	3.5 MHz
DVD PAL	11	1.5 μs	6.5 MHz	218 MB/s	14.3 MHz
XVGA	8	0.8 μs	12.3 MHz	413 MB/s	27.0 MHz

7 Conclusion and Future Work

A reconfigurable platform suitable for embedded video processing in real-time has been presented. The platform comprises a number of configurable complex processing elements operating within a structured communication framework, all controlled by a central system controller. The design aims to meet the demanding requirements of real-time video processing by exploiting fine-grained (pixel-level) parallelisms within the processing elements, as well as task-level algorithmic parallelisms by operating processing elements concurrently.

The customisation of processing elements enables the system to be adapted to a wide range of applications, and since these elements are dynamically reconfigurable, run-time adaptation is also possible. Moreover, the inherent modularity of the PEs, coupled with the communication infrastructure, facilitates design and reuse.

Finally, power efficiency is addressed by targeting data movement. Dedicated bus wiring can be optimised for low-power transmission, while data movement is minimised by reusing data stored in stream buffers and constraining processing element designs. Parallel processing results in lower average operating frequencies, which can be translated into lower power consumption by reducing the operating voltage.

Our next main objective is to translate the proposal as described in this paper into a prototype, so that we may assess its feasibility. Since the physical structure of the system is based heavily on the Virtex II Pro, it would be logical to implement the prototype in a device from this family. In order to investigate the operational performance of the system, it will be necessary to map one or more actual applications from Sonic to the new platform, which may require the development of tools and processes.

Acknowledgements

The authors would like to thank Simon Haynes, Henry Epsom and John Stone of Sony Broadcast and Professional Europe for their helpful comments. The support from Xilinx Inc., and Patrick Lysaght in particular, is appreciated. N.P. Sedcole gratefully acknowledges the financial assistance from the Commonwealth Scholarship Commission in the United Kingdom.

References

1. Haynes, S.D., Stone, J., Cheung, P.Y.K., Luk, W.: Video image processing with the Sonic architecture. IEEE Computer **33** (2000) 50–57
2. Soudris, D., Zervas, N.D., Argyriou, A., Dasygenis, M., Tatas, K., Goutis, C., Thanailakis, A.: Data-reuse and parallel embedded architectures for low-power, real-time multimedia applications. In: International Workshop - Power and Timing Modeling, Optimization and Simulation. (2000)
3. Arnold, J.M., Buell, D.A., Hoang, D.T., Pryor, D.V., Shirazi, N., Thistle, M.R.: The Splash 2 processor and applications. In: IEEE International Conference on Computer Design: VLSI in Computers and Processors. (1993)

4. Vuillemin, J.E., Bertin, P., Roncin, D., Shand, M., Touati, H.H., Boucard, P.: Programmable active memories: Reconfigurable systems come of age. IEEE Transactions on VLSI Systems **4** (1996) 56–69
5. Athanas, P.M., Abbott, A.L.: Real-time image processing on a custom computing platform. IEEE Computer **28** (1995) 16–24
6. Callahan, T.J., Hauser, J.R., Wawrzynek, J.: The Garp architecture and C compiler. IEEE Computer **33** (2000) 62–69
7. Ebeling, C., Cronquist, D.C., Franklin, P.: RaPiD – Reconfigurable Pipelined Datapath. In: Field–Programmable Logic and Applications. (1996)
8. Goldstein, S.C., Schmit, H., Budiu, M., Cadambi, S., Moe, M., Taylor, R.R.: PipeRench: A reconfigurable architecture and compiler. IEEE Computer **33** (2000) 70–77
9. Waingold, E., Taylor, M., Srikrishna, D., Sarkar, V., Lee, W., Lee, V., Kim, J., Frank, M., Finch, P., Barua, R., Babb, J., Amarasinghe, S., Agarwal, A.: Baring it all to software: RAW machines. IEEE Computer **30** (1997) 86–93
10. Wirthlin, M.J., Hutchings, B.L.: A dynamic instruction set computer. In: IEEE Symposium on FPGAs for Custom Computing Machines. (1995)
11. Horta, E.L., Lockwood, J.W., Taylor, D.E., Parlour, D.: Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. In: Design Automation Conference. (2002)
12. Dyer, M., Plessl, C., Platzner, M.: Partially reconfigurable cores for Xilinx Virtex. In: Field–Programmable Logic and Applications. (2002)
13. Kalte, H., Langen, D., Vonnahme, E., Brinkmann, A., Rückert, U.: Dynamically reconfigurable system-on-programmable-chip. In: Euromicro Workshop on Parallel, Distributed and Network-based Processing. (2002)
14. Benini, L., De Micheli, G.: Networks on chips: A new SoC paradigm. IEEE Computer **35** (2002) 70–78
15. Dally, W.J., Towles, B.: Route packets, not wires: On-chip interconnection networks. In: Design Automation Conference. (2001)
16. Haynes, S.D., Epsom, H.G., Cooper, R.J., McAlpine, P.L.: UltraSONIC: a reconfigurable architecture for video image processing. In: Field–Programmable Logic and Applications. (2002)