Discourse on Extending Embedded Medical Image Processing Systems Using the High Speed Serial RapidIO Interconnect

Moritz Schmid[1], Frank Hannig[1], Jürgen Teich[1], Ralf Diefenbach[2], Hartmut Pettendorf[2], and Heinz Hornegger[2]

[1] Hardware/Software Co-Design, Department of Computer Science, University of Erlangen-Nuremberg, Germany.
{moritz.schmid, hannig, teich}@cs.fau.de
[2] Siemens Healthcare Sector, H IM AX, P.O.Box 1266, Forchheim, Germany.
{ralf.diefenbach, hartmut.pettendorf, heinz.hornegger}@siemens.com

Abstract. In this paper we study the possibilities offered by relatively new embedded systems interconnect technologies to extend and improve existing processing frameworks. We have chosen medical image processing applications for our research due to their rigid requirements, such as very high fault-tolerance, stringent real-time and demanding computational power. We present an approach of how to design Serial RapidIO endpoint controllers on FPGAs that are very small, provide low latency and very high data rates. Using this lightweight controller design, we can efficiently distribute high throughput processing applications across multiple FPGAs in a framework.

1 Introduction

Coronary artery disease is a leading cause of death and morbidity in industrialised countries [13]. Therapeutic options are decided upon the basis of the current state of the coronary arteries, which can be assessed by coronary artery lumen evaluation [10]. Although catheter angiography is the standard for coronary lumen evaluation, it is an invasive and potentially harmful procedure, that involves admission to a hospital and discomfort for the patient. More importantly, it offers only little information on coronary artery wall changes, associated with early, or even chronic stages of coronary artery atherosclerosis [4]. Coronary computed tomography angiography (CTA) was proposed as a non-invasive approach to analyse the condition of the coronary arteries. In this procedure, a patient's cardiovascular system is injected with a contrast medium, which can be made visible by X-radiation. Magnetic resonance imaging and electron-beam CT have been proposed. However, both show significant limitations in reliable visualisation of the coronary arteries. Lately introduced multi-slice and dual-source CT scanners have the potential to not only enable non-invasive coronary lumen evaluation, but also imaging of the artery wall and therefore, atherosclerotic plaque visualisation [6].

Revealing results can be obtained by taking high resolution scans of the affected area over a longer period of time, preferably at a rate of about 25 pictures per second to create the impression of a moving picture. A major concern here is, that exposure to high levels of radiation is extremely harmful to the patient's health. The risks may be mitigated by reducing the dosage of radiation, which in turn dilates relative noise in the picture. For the resulting images to be viable for diagnosis, sophisticated digital signal processing must be applied to separate relevant content from the disturbance in the picture. The image processing requires substantial resources for the calculations and is subject to very strict real time conditions. Even at the lowest image resolution suitable for angiography, the computational load would still overwhelm the most recent desktop processors. For these reasons, it has become a common practice to implement the algorithms on customised hardware using FPGAs and dedicated signal processors. However, current developments in medical imaging requirements strive to demand even more processing power than contemporary chips can provide. To overcome this bottleneck the algorithms may distributed across multiple hardware resources.

2 Related Work

Apart from the official standard specification for RapidIO, ranging from version 1.4 to the most current version 2.1 ([9]), numerous technical articles can be found introducing the reader to the standard (e. g., [5]), comparing it to other popular interconnect technologies for embedded systems (e. g., [14]) and demonstrating its applicability for heterogeneous multiprocessing (e. g., [7]). Several works target the implementation of routers (e. g., [11]) and switches (e. g., [12]) for RapidIO. However, our work differs from these approaches as we attempt to design very lightweight endpoint controllers. A series of research on the use of RapidIO for space-based radar applications was conducted by Bueno and others. The authors' work ranges from a first approach in [1] on the feasibility of RapidIO for real-time, high network-traffic to the consideration of the implementation of concrete industry-standard algorithms in [2]. Especially the second work, as well as Bueno's doctoral thesis [3] focus on partitioning real-time algorithms for processing in RapidIO networks. In [15], Zhang and others direct their research on the abilities of Serial RapidIO as an interconnect for heterogeneous multiprocessor architectures for signal processing. However, the authors focus on the kind of signal processing as occurring in wireless transmission systems and, further, instead of only using FPGAs, they consider the combination of FPGAs and DSPs.

3 Serial RapidIO

3.1 History

The RapidIO (RIO) Interconnect Specification, released by the RapidIO Trade Association [8], is an open standard, developed to achieve high-performance, low-cost, reliable and scalable system connectivity in embedded systems, networking applications and communication devices. Its capability for advanced traffic management, built-in error-correction mechanisms, and provisioning for high performance and throughput have aided to position RIO as a dominant interconnect, for example, especially in the area of mobile communication base-station infrastructure. RIO has the strongest embedded ecosystem, because of it is spread across all of the major embedded markets and has achieved deployment by major original equipment manufacturers (OEMs) and defence contractors.
The RIO standard dates back until 1997, when Motorola began working on a new front-side bus for its Power- and PowerPC-based processors. By 1999, Motorola and Mercury Computer Systems had joined together to complete the first complete RIO specification. In 2000, these companies drove the formation of the RIO Trade Association, which made the specification an independent standard. The RIO specification became an international standard as ISO/IEC 18372:2004 in the year 2004. In 2005, the RIO Trade Association announced version 1.3 of the specification. The most recent version 2.1 of the specification was released in late 2009, but is not widely supported yet. The work in this document is based on version 1.3 of the RIO specification. The RIO Trade Association hosts all of the available specifications at [9].

3.2 Technological Overview

The RIO standard was developed to meet the unique requirements found in embedded systems. RIO focuses on on-board, inter-board, and back-plane connectivity. It can be implemented completely in hardware, which includes error handling and traffic management. This limits impact on software and provides a transparent means of intra- and inter-systems communication. The protocol is partitioned into a layered architecture to reduce complexity, and provide flexibility and extendability. It is shown in Figure 1.
At the top level of the specification, the *logical layer* is in control of the transaction model between the endpoints of a RIO network. The specification defines several types of transactions. *ATOMIC* transactions represent a specific class of transactions for cache-coherence schemes. DMA-type data models are supported through *MESSAGE* transactions. RIO also implements *simple write* (NWRITE), *read* (NREAD), *acknowledged write* (NWRITE_R) and *streaming write* (SWRITE) transactions. The SWRITE transactions are especially suitable for the transfer of large quantities of data, since they use a shortened header and thus, provide the most efficiency. *MAINTENANCE* is another important class of transactions. They are used to configure endpoint devices and intermediate switches. Finally, *DOORBELL* transactions provide a very lightweight means to

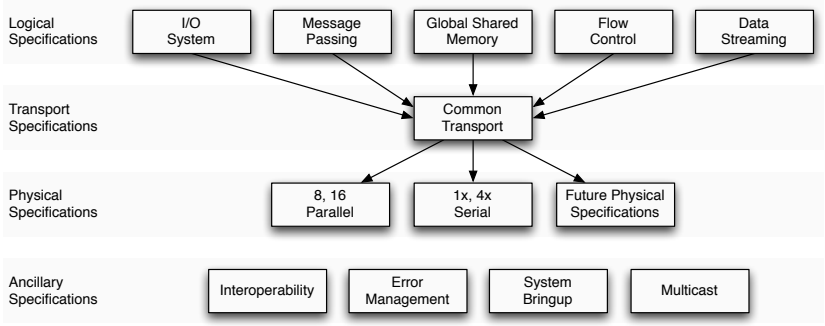

Fig. 1: Overview of the RIO specification hierarchy.

transport very short information of up to 16 bits.
The *transport layer* directly below the logical layer is in charge of routing packets through the fabric from one endpoint to another by traversing a number of switches. Endpoints can be addressed using their unique device identifier (ID), similar to IP addresses. In contrast, the switches of the fabric do not possess individual addresses. They can be targeted indirectly using a combination of endpoint device ID and a *hopcount*-value, which specifies how many hops the packet should traverse on a specific route before it has reached its destination.
There exist separate parallel and serial physical specifications. RIO provides support for two different physical standards. The *parallel physical layer* is known as the 8- or 16-bit link protocol end point specification with low-voltage differential signalling (8/16 LP-LVDS). It specifies a parallel, bidirectional communication of either 8 or 16 bits, including clock and frame signals. In contrast, the serial specification does not define a source synchronous interface, but rather has to be synchronised between link-partners by special frame signals. Clock information is encoded within the transmitted data, using the 8B/10B encoding scheme. The serial specification uses a physical coding sublayer (PCS) and a physical media attachment (PMA) sublayer to organise the packets into a serial bit stream at the transmitter, and de-serialise the bit stream at the receiver. In this work, however, we concentrate on the serial physical specification. The *serial physical layer* is responsible for exchanging packets on a link-by-link basis, also known al the data link protocol. Data flow between RIO devices is managed by control symbols. *Start of Frame* (SOF) and *End of Frame* (EOF) control symbols are used to delimit packets. There also exist control systems for packet acknowledgement, flow control and maintenance functionality. Further, the PHY specification defines error management, deadlock avoidance and the electrical interface.
Unlike legacy bus technologies, RIO provides link maintenance and error management mechanisms, which enable initial system discovery and configuration, as well as error detection and recovery. Each RIO device has a set of capability and command status registers (CAR/CSR), which specify the device's abilities and current state. By the use of maintenance operations, a RIO device may

traverse the network and access these registers to discover and configure other devices. To avoid error situations, more than one host is allowed to configure the network. Mean time between failure (MTBF) and mean time to repair (MTTR) are critical system considerations. Because of RIO operates at very high frequencies, errors are likely to occur and a strong error detection and recovery mechanism is required. Besides using LVDS, RIO relies on a CCITT16 cyclic redundancy check to detect errors in packets and control symbols. Since failure detection and recovery is part of the physical specification, it will be discussed in the appropriate section in more detail.
RIO offers several features to improve system performance. Source routing and packet priority tagging were implemented to reduce blocking of packets. Switch-based interconnect systems can either employ multicasting or source-routing to forward packets across the network. Although multicasting has the advantage of the initiator not having to know where to find the destination of the transaction, moderate traffic situations might already cause the network to become unresponsive. Source-directed routing uses the target address to route the packet on a specific path through the fabric, avoiding to burden other paths than those between the initiator and the target of a transaction with traffic. RIO headers were carefully designed to not supply any more information than is absolutely necessary to complete the transaction. For most RIO transactions, the overhead will be around 28 bytes, including headers for request and response packets as well as the control symbols for acknowledgement and CRC codes.

4 Elevating an SRIO IP Core for Large Data Transfer

In this section, we will illustrate the implementation of a very lightweight Serial RapidIO endpoint controller, which is specifically crafted to provide efficient transfer of large quantities of data. The endpoint controller servers as an application layer, on top of an implementation of the Serial RapidIO specification in form of the Xilinx Serial RapidIO IP core.
Xilinx gives the developer the choice to only implement a subset of all the possible SRIO transactions. Choosing only to include maintenance functionality and the SWRITE transaction produces a very small implementation of the company's IP core. However, including the resulting hardware module in an application with stringent real-time and latency requirements is not a job for the faint-of-heart. The implementation of a custom controller to be able to efficiently transport image data is an absolute necessity.
The transaction concept in implemented in such way that actions are initiated by issuing requests and confirmed by responses from the target of the request. In our case, although the data transfer via the SWRITE transaction only involves requests, we need to retain the configurability of the endpoint by maintenance transactions, which also require to send responses. Accordingly, the user interface of the Xilinx SRIO IP Core, which is shown in Figure 2, is divided into separate interaction ports. Despite the existence of a maintenance interface, the

user is expected to communicate with the IP Core solely through the user interface. We have chosen to craft two individual applications for transmitting and

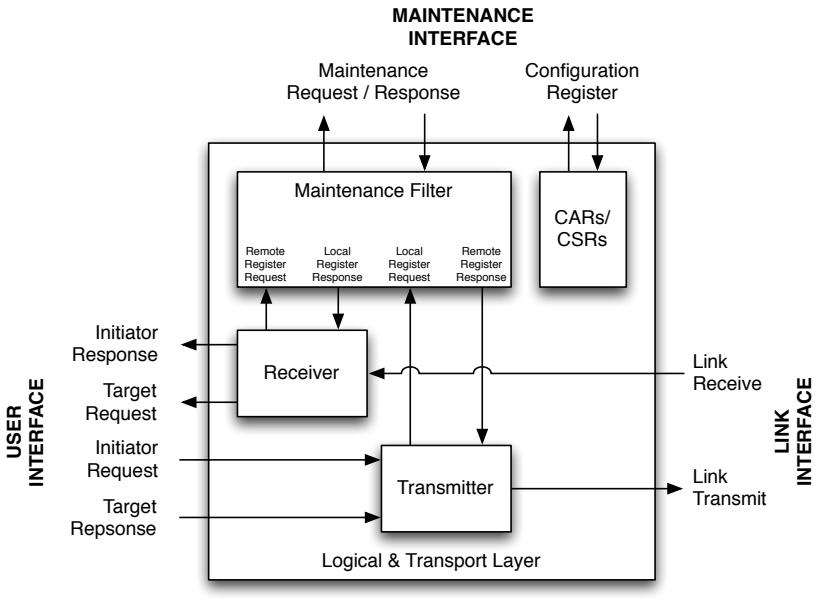

Fig. 2: Overview of the Xilinx SRIO IP Core user interface.

receiving information. The transmit module of the endpoint controller is used to send data by issuing SWRITE and DOORBELL transactions to communication partners. In addition, it offers maintenance operability by MAINTENANCE requests. Ingress data and management information is obtained from the IP core by the receiver module. RapidIO has a very sophisticated data flow mechanism, which can only slightly be interfered with at compile-time. To ensure maximum interoperability between devices of different manufacturers, these options are reduced to switch between source or target oriented flow control. Using FIFO buffers between the endpoint controller and the SRIO IP Core enables us to let RapidIO control flow of information. We have evaluated this design and obtained very good results. However, due to restrictions by Xilinx, we cannot publish these.

The user interface of the IP Core is rather complicated and needs a sophisticated timing design to be operated correctly and efficiently. To mitigate these requirements, we understand our endpoint controller as a frontend to the IP core, which renders the task of using the SRIO module in an actual design rather simple. Instead of having to drive many different signals on the user interface of the IP Core to control the transmission, we have reduced the required signals to a minimum and have included a simple FIFO interface to reduce stringent timing

and synchronisation.
The custom endpoint controller design is depicted in Figure 3. Both modules,

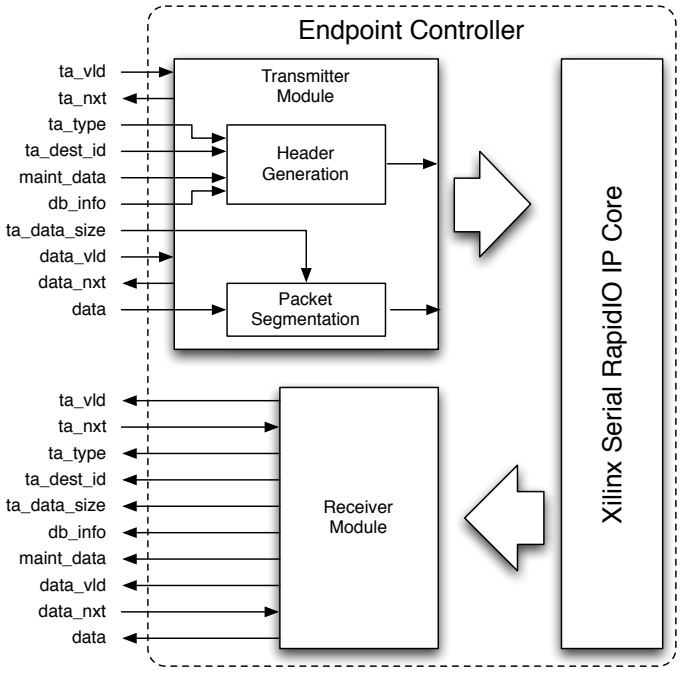

Fig. 3: The SRIO endpoint controller, consisting of a transmitter and a receiver module. Both custom modules are connected to the appropriate ports of the Xilinx SRIO IP Core user interface.

the transmitter and the receiver, including signals to be driven by the user are shown. We implement a very simple FIFO-type interface for configuration and data communication. As soon as the user has driven the configuration signals, the transaction can be started by assertion of the `ta_vld` signal. The data interface can be connected to a FIFO buffer so that the consumption of data by the controller can be automated. If the `ta_nxt` signal is asserted, the controller has finished transmitting all remaining data and is ready to accept new tasks. The receiver module is crafted in exactly the same manner. If the controller has received new data to pass on to the user, the `ta_vld` signal is asserted and the data can be accepted by assertion of the `ta_nxt` signal. The internals of the both modules are more complex, however, the receiver module can be designed smaller than the transmitter. As we restrict the functionality to only a subset of the RapidIO functionality, we can greatly decrease the logic, required for transaction control.

5 Extending a Medical Image Processing Framework Using Serial RapidIO

The previously introduced controllers were designed specifically to take up as little space on an FPGA as possible and to leave as much resources as possible for the implementation of processing elements. In this way, we can use SRIO as an interconnect between FPGAs to distribute computation data, but also to connect already existing frameworks to new components or, for example, provide an accelerated FPGA-based interconnect for legacy systems. The requirements for medical imaging hardware have always been very strict, so that it is no surprise that these systems are custom handcrafted to yield the demanded efficiency. As the industry progresses, systems might become outdated or unable to cope with new requirements. Being very lightweight, our RapidIO endpoint controller is a very suitable addition to those systems, since it will make it possible to extend the system with new hardware but also fit onto older chips. Moreover, as we consider imaging systems, most of the data to be transported will be medical images. Because of the size of these may be changed or simply not be known at the link-partner, it is necessary to pass on this information to the target. RapidIO offers several possibilities for this task. The most obvious is to come up with a certain header format, by which the target can identify data semantics, such image delimitations. The drawback of such a solution is that having to send and extra header introduces additional latency. Another possibility is to use the DOORBELL transactions 16 bit data-field to notify the link-partner of start and end of an image or other information. DOORBELLs are very small and can be transferred within a single clock cycle. Hence, these are the ideal solution to pass any control information between link partners.

6 Conclusion

In this paper, we have shown an approach of how to implement a lightweight, high throughput and low latency endpoint controller for Serial RapidIO on FPGAs. We can greatly simply the user interface to the Xilinx Serial RapidIO IP Core and the amount of resources to implement the controller by restricting the functionality to only the necessary components of the RapidIO specification. The resulting controller can efficiently aid in distributing data to other SRIO devices. We have used the controller to study the implementation of distributed signal processing algorithms, which can be relieved from accuracy restrictions imposed by limited resources. However, licence agreements with the vendor of the SRIO IP Core prohibit us from publishing detailed performance measurements.

References

1. Bueno, D., Leko, A., Conger, C., Troxel, I., George, A.: Simulative analysis of the RapidIO embedded interconnect architecture for real-time, network-intensive

applications. In: Local Computer Networks, 2004. 29th Annual IEEE International Conference on. pp. 710–717 (Nov 2004)
2. Bueno, D., Conger, C., George, A.D., Troxel, I., Leko, A.: RapidIO for radar processing in advanced space systems. ACM Trans. Embed. Comput. Syst. 7(1), 1–38 (2007)
3. Bueno, D.R.: Performance and dependability of RapidIO-based systems for real-time space applications. Ph.D. thesis, Gainesville, FL, USA (2006), adviser-George, Alan D.
4. Chartrand-Lefebvre, C., Cadrin-Chnevert, A., Bordeleau, E., Ugolini, P., Ouellet, R., J.-L.Sablayrolles, Prenovault, J.: Coronary computed tomography angiography: Overview of technical aspects, current concepts, and perspectives. Canadian Association of Radiologists journal 58(2), 92–108 (2007)
5. Fuller, S., Cox, T.: Anatomy of a forward-looking open standard [RapidIO]. Computer 35(1), 140–141 (Jan 2002)
6. Nieman, K., Oudkerk, M., Rensing, B., van Ooijen, P., Munne, A., van Geuns, R., de Feyter, P.: Coronary angiography with multi-slice computed tomography. Lancet 357(9256), 599–603 (2001)
7. Rao, N.: Using serial RapidIO for FPGA co-processing. Online (2007), *http://www.dspdesignline.com/howto/202803152*, accessed March 7, 2009.
8. RapidIO Trade Association: Rapidio. *http://www.rapidio.org*
9. RapidIO Trade Association: RapidIO specification (2009), *http://www.rapidio.org/spec/current*, accessed January 7, 2010.
10. Scanlon, P., Faxon, D., Audet, A., Carabello, B., Dehmer, G., Eagle, K., Legako, R., Leon, D., Murray, J., Nissen, S., Pepine, C., Watson, R.: ACC/AHA guidelines for coronary angiography. Journal of the American College of Cardiology 33(1756), 824 (1999)
11. Schelle, G., Grunwald, D.: Cusp: a modular framework for high speed network applications on fpgas. In: FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays. pp. 246–257. ACM, New York, NY, USA (2005)
12. Sukhtankar, S., Hecht, D., Rosen, W.: A novel switch architecture for high-performance computing and signal processing networks. In: NCA '04: Proceedings of the Network Computing and Applications, Third IEEE International Symposium. pp. 215–222. IEEE Computer Society, Washington, DC, USA (2004)
13. WHO: Cardiovascular Diseases Fact Sheet. WHO Fact Sheets (137) (2007)
14. Wood, B.: Backplane Tutorial: RapidIO, PCIe and Ethernet. Online (2009), *http://www.dspdesignline.com/howto/212701200*, accessed March 7, 2009.
15. Zhang, X., Gao, M., Liu, G.: A scalable heterogeneous multi-processor signal processing system based on the RapidIO interconnect. In: Intelligent Information Technology Application Workshops, 2008. IITAW '08. International Symposium on. pp. 761–764 (Dec 2008)