Chinese Journal of Electronics
vol. 33, no. 2, pp. 362–370, March 2024
https://doi.org/10.23919/cje.2022.00.094

RESEARCH ARTICLE

Architecture Design of Protocol Controller Based on Traffic-Driven Software Defined Interconnection

Peijie LI, Jianliang SHEN, Ping LYU, Chunlei DONG, and Ting CHEN

Information Engineering University, Zhengzhou 450002, China

Corresponding author: Peijie LI, Email: lpj@ndsc.com.cn
Manuscript Received April 19, 2022; Accepted March 21, 2023

Abstract — To solve the problems of redundant logic resources and poor scalability in protocol controller circuits among communication networks, we propose a traffic-driven software defined interconnection (TSDI) mechanism. The unified software defined interconnection interface standards and the normalized interconnection topology are designed to implement the architecture of TSDI-based protocol controller. The key indicators of power, performance and area (PPA) can be realized while resolving the flexible interconnection of the controller. We designed a TSDI-based RapidIO controller as an example. Compared to traditional designs, the design could achieve more protocol scalability, and RapidIO protocol standards of Gen4 could be supported directly. The key PPA indicators, such as a lower delay of 56.1 ns and more than twice throughput of 98.1 Gbps, were achieved at the cost of a 23.4% area increase.
Keywords — Software defined interconnection, Traffic-driven, Protocol controller, Embedded interconnects, Interface standards, Interconnection topology.
Citation — Peijie LI, Jianliang SHEN, Ping LYU, *et al.*, "Architecture Design of Protocol Controller Based on Traffic-Driven Software Defined Interconnection," *Chinese Journal of Electronics*, vol. 33, no. 2, pp. 362–370, 2024. doi: 10.23919/cje.2022.00.094.

I. Introduction

To meet the increasing demand for shorter application development cycles and higher network scalability brought by protocol upgrades, the architecture of protocol controllers must be full-dimensional defined [1], [2]. Many researches [3]–[5], trying to solve protocol controller performance and scalability issues, mainly focus on mature IP cores [6]–[8]. These designs were based on dedicated architecture, a hierarchical hardware processing architecture in which corresponding parallel hardware circuits must be designed when requirements expand and protocol upgrades. The dedicated architecture resulted in resource redundancy and lower scalability. It is vital for the architecture that the protocol controller can provide new functionality without redesign. The overarching goal of the architectural revolution is to deliver high-performance, low-power, and efficient area solutions.

Many reconfigurable architectures have been presented based on FPGA (field-programmable gate arrays), state machine, microcontroller, or eFPGA (i.e., embedded FPGA). In [9], a vendor-independent structure based on software defined network (SDN) was proposed for disaggregating the control and data planes. A Programmable logic controller (PLC) [10] architecture was dedicated to implementing a programmable controller based on FPGA. However, the SDN and FPGA architectures were significantly worse than application specific integrated circuit (ASIC) design in power, performance, and area (PPA). In order to meet these factors, a dynamically reconfigurable packet distribution unit [11] is used to solve the traffic distribution in the embedded system. Moreover, an approach to developing a dynamically reconfigurable transport layer controller architecture in ASIC was presented by Suvorova [12]. These designs were based on dynamically reconfigurable automata and finite state machine [13], [14] with DataPath [15], [16]. For more efficiency, Suvorova [17] proposed an approach to developing a dynamically reconfigurable transport layer controller based on a processor core with reduced instruction set computer V (RISC-V) architecture. Based on an

Associate Editor: Prof. Shaojun WEI, Tsinghua University.

eFPGA-augmented RISC-V SoC, Arnold in [18] was designed to achieve more flexibility and lower power. However, existing work should pay more attention to fully defined protocol controllers, which have stream processing characteristics. They lack versatility, require more time to reconfigure, and are not enough to implement the more flexible hardware structure with better PPA. In [1], [19], a new software-defined architecture combines the software defined interconnection (SDI) with the reconfigurable coarse-grained units. SDI can provide a new technology for the design of protocol controllers.

According to the stream processing characteristics of protocol controllers, this paper introduces a traffic-driven software defined interconnection (TSDI) structure to break traditional controllers' hierarchical hardware processing architecture. The main contributions of this work is as follows:

1) Flexible architecture. A new architecture and a general network structure for realizing the flexible interconnection and reconfiguration among software-defined elements designed in protocol controllers: a) independent input address, b) independent output addressing, and c) a software-defined network.

2) Unified hardware structure. A new template for supporting the combination and recombining of the complex topologies based on protocol traffic stream processing: a) normalized software-defined elements model and topologies, and b) software-defined interconnection interface based on stream addressing.

3) Use case. We demonstrate the TSDI architecture on a RapidIO controller, achieving excellent application flexibility and protocol scalability performance with the cost of a certain increase in resources compared to traditional controllers.

The remainder of this paper is organized as follows. Section II introduces the characters of the TSDI network. In Section III, we describe the architecture of the protocol controller based on TSDI. The software-defined interface standard, software-defined interconnection topology, and an implementation example are presented in Section III. Experimental results are reported in Section IV, and Section V summarizes the main points of this paper.

II. The TSDI Network

TSDI is mainly composed of three types of software-defined grains (SDGs), the calculating element C, the memory element M and the interconnecting element I, in the reconfigurable resource pool [2], [19]. SDGs include software-defined processing grains (SDPGs) and interconnection grains (SDIGs). SDPGs, implementing the calculate and memory function, can be extracted into private and public reconfigurable elements by using the hardware reconfigurable technology and the mathematical model of software-defined protocol controllers. SDIGs, special SDPGs, can connect different SDPGs according to AI, the input address of the last hop, and AO, the addressing output of the next hop. A TSDI model can be abstracted as below:

$$TSDI_{\mathrm{out}} = Flow_n(SDPG_{n_m}, SDIG_{n_m}) \tag{1}$$

where $TSDI_{\mathrm{out}}$ is the final stream output of the traffic, $Flow_n(SDPG_{n_m}, SDIG_{n_m})$ is the n-th level stream output and $Flow_n(SDPG_{n_m}, SDIG_{n_m})$ can be defined as the stream processing output of the $n-1$ level output:

$$\begin{aligned} &Flow_n(SDPG_{n_m}, SDIG_{n_m}) \\ &\quad = Flow_n(Flow_{n-1}(SDPG_{n-1_{m1}}, SDIG_{n-1_{m1}})) \end{aligned} \tag{2}$$

where $SDPG_{n_m}$ are m SDPGs in n-th level stream processing elements selected from the reconfigurable resource pool:

$$SDPG_{n_m} = F_{\mathrm{C,M,I}}(0), F_{\mathrm{C,M,I}}(1), \ldots, F_{\mathrm{C,M,I}}(m) \tag{3}$$

And $SDIG_{n_m}$ are all m SDIGs' functions in n-th level stream processing elements, which mainly describe the interconnection relationship among SDGs:

$$\begin{aligned} SDIG_{n_m} = {} & F_{\mathrm{C,M,I}}(0)(AI, AO), F_{\mathrm{C,M,I}}(1)(AI, AO), \\ & \ldots, F_{\mathrm{C,M,I}}(m)(AI, AO) \end{aligned} \tag{4}$$

When processing protocol traffics in the TSDI network, related SDPGs are interconnected through corresponding topologies according to different traffic types (such as idle sequences, control symbols (CS), messages, and protocol messages).

Figure 1 shows a typical service processing schematic diagram of TSDI, which can optimize the topology structure according to being processed traffics. When dealing with the idle sequence analyzing traffic, where the streaming characteristic is presented from decoding and descrambling to error detection, TSDI optimizes the topology into a pipe processing model. When message parsing traffic is handled, which shows the branches of parallel processing characteristics, TSDI optimizes the topology into a mixed processing form of serial and parallel.

III. The Architecture of Protocol Controller Based on TSDI

The traffic processing flow of the communication protocol complies with the protocol stack's multi-layer network architecture and has the characteristics of stream processing. Figure 2 shows the architecture of the traffic processing in the traditional protocol controller.

In Figure 2, the received data from a channel are pipeline processed by serial-to-parallel conversion and bit synchronization in the transceiver module, then handed over to the physical layer for symbol decoding (descrambling, decoding, error correction and error detection, etc.) and data identifying. The data reformed by physical layer are de-stripped into packets, which will be analyzed in transport layer and then stored to or read from

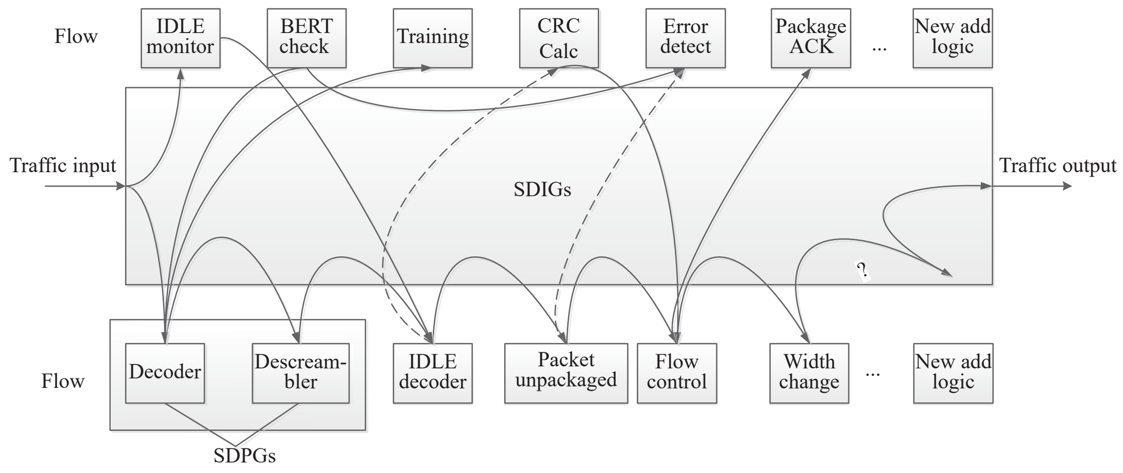


Figure 1 Schematic diagram of TSDI typical service processing.

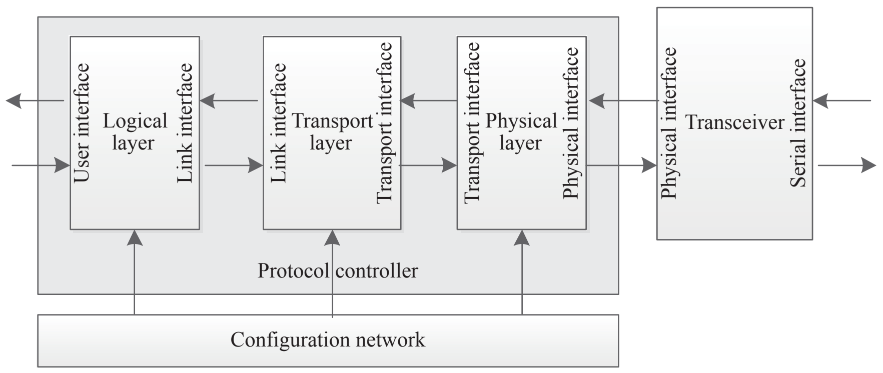


Figure 2 The architecture of traffic processing in the traditional protocol controller.

the inside memory according to the traffic types. The unpackaged load data will be operated at logical layer, and transport layer will respond to the patterner and send it to physical layer.

The whole traffic processing flow of the protocol controller shows the stream characteristics such as branching, reversible processing, storage, and computing. The downstream logic and upstream logic can be abstracted as a simple model which is set as the input of the previous stage and output of the subsequent stage. As shown in Figure 3, the traffic processing flow is abstracted as a stream process of multiple Functions, in which SDPGs and SDIGs are interconnected through stream processing mode.

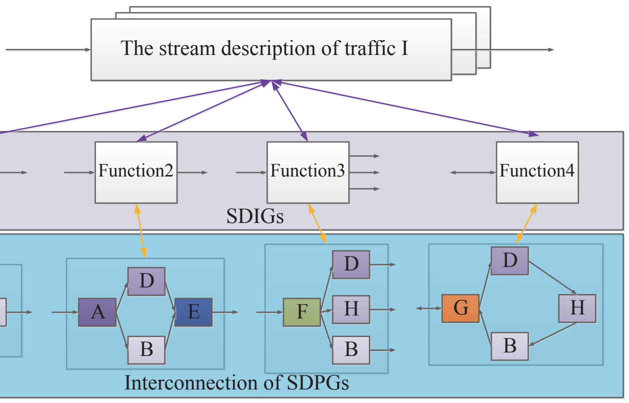


Figure 3 The diagram of traffic process flow abstracted as SDIGs and SDPGs.

The basic structure of SDPGs based on C or M and SDIGs based on Function can be summarized in the form of both input and output. In order to achieve flexible interconnection among SDPGs and SDIGs, TSDI uses a software-defined interconnection interface, which restricts not only the interfaces of the controller and each protocol layer but also the interfaces of SDPGs and SDIGs. The interface is based on stream address, in which the next hop can be calculated by input address mapping result. The output of previous SDPGs acts as the input of the next SDPGs. Therefore, the whole business will be interconnected by SDIGs and SDPGs.

Definition 1 The stream processing of traffic is defined as $T(T_i, n, T_o)$.

In Definition 1, T_i is the function of traffic input, named as source traffic interconnection grain (STIG), T_o is the function of traffic output, named as destination traffic interconnection grain (DTIG), and n is the hop counts from T_i to T_o.

Definition 2 The stream processing of function is defined as $F(F_i, m, F_o)$.

In Definition 2, F_i is the input of the first SDPGs in interconnect function, named as source function input of process grain (SFPG), F_o is the output of the last SDGs in interconnect function, named as destination function output of process grain (DFPG), and m is the hop counts from F_i to F_o.

Definition 3 The function of process grain is defined as $P_{i,o}$.

In Definition 3, i is the input of the SDPG, and o is the output of the SDPG.

The stream processing model of the protocol controller based on TSDI can be described as $T(P_i(i,o), n, P_{m_{i+n-1}})$.

1. Software-defined interconnection interface based on stream address

The SDIGs and SDPGs used in TSDI can be normalized for the abstracted model described as the stream interconnection among SDGs. The interface standard must define the input and output feature of the stream address since the grain is interconnected according to the mapping result of the stream address. And a reconfigurable data width feature of the input and output interface should also be supported because of the grains' stream aggregation and redistribution characteristics. The interface should be bidirectional simultaneously to balance the flexibility of the interconnection and the feasibility of the physical realization of each grain in TSDI. The interface should include the clock signal to realize the redefinable bandwidth. The entire interface standard can be defined in Deinition 4.

Definition 4 The interface of SDGs is defined as $I(S_{\mathrm{clk}}, S_{\mathrm{rst}}, D_{\mathrm{clk}}, D_{\mathrm{rst}}, D_{\mathrm{i}}, D_{\mathrm{o}}, P_{\mathrm{dir}}, S_{\mathrm{a}}, D_{\mathrm{a}})$.

In Definition 4, S_{clk} is the input clock of grains, S_{rst} is the input reset signal of grains, D_{clk} is the output clock of grains, and D_{rst} is the output reset signal of grains. D_{i} and D_{o} are the input and output data signals, which can be software-defined to match the bandwidth, allocate the meaning of each bit, and change the direction on demand, P_{dir} is the description primitives for the location of the data signal describing the D_{i} or D_{o} as horizontal or vertical when physical implementation. S_{a} is the input address of the current grain, and also the output of the previous. D_{a} is the addressing output of the current grain and also the input of the next.

To simplify the interconnection network and reduce the difficulty of physical implementation, the TSDI can be implemented as an array of SDGs. The interface standard of SDGs in TSDI is a software-defined one based on stream addresses, in which the bandwidth and direction can be redefined. The interconnection between two SDGs is based on the mapping result of flow addresses, determined by the input of the last hop and the output of the next hop. So, when matching a type of traffic, the interconnection topology of the controller will be generated, which makes the critical features of the traffic be predicted in advance.

2. Software-defined interconnection topologies based on TSDI

Each SDG has a two-input and two-output interconnection topology. However, the topology must be simpler to achieve normalization. In the interconnection structure of SDGs as shown in Figure 3, the input and output interfaces are the main features. The abstract model can be described in Figure 4, including single-in-single-out (SISO), multiple-in-single-out (MISO), single-in-multiple-out (SIMO), and self-in-self-out (SIO). It is necessary to analyze the structure of the forms to normalize the topologies.

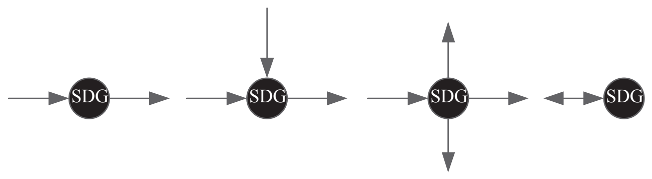


Figure 4 The abstract model of SDGs.

For MISO, the S_{a} of the SDG is the same as the D_{a} of all the last SDGs. It is the aggregation of multiple traffic with the same destination stream address. When the aggregation exceeds the maximum bit width defined by the interface standard, the MISO topology will be transformed into multi parallel channels until the bandwidth meets the sum of SDGs' multi-inputs or until the processing frequency of SDGs is increased to match the required bandwidth. Each of the channels has a serial processing topology. The bandwidth of D_{i} will be changed, and the width of the interface will be redefined. Therefore, MISO can be simplified as SISO topology, as shown in Figure 5.

For SIMO, it is easy to understand that the S_{a} is simultaneously mapped to multiple SDGs of the next hop, which means that multiple copies of the SDG will be generated, and the same input will drive all the copies while each output drives the next corresponding SDG. SIMO can be simplified, as shown in Figure 6.

SIO is a self-loop with the same S_{a} and D_{a}. The structure still follows the SISO and can be easily replaced by a SDG and a SDIG, shown in Figure 7.

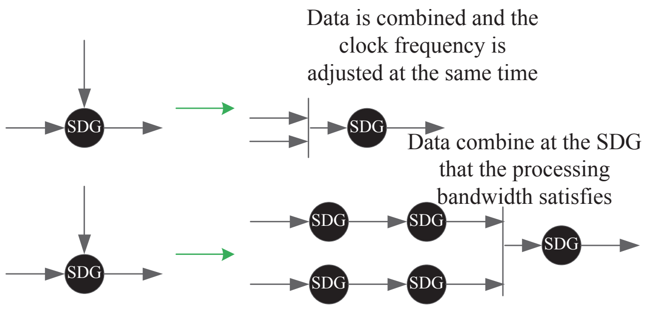

Data is combined and the clock frequency is adjusted at the same time
SDG
Data combine at the SDG that the processing bandwidth satisfies

Figure 5 Equivalent structure of MISO SDG.

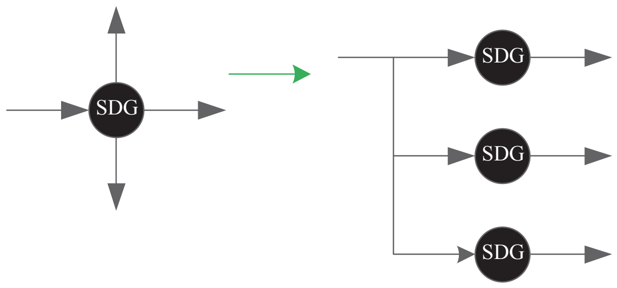


Figure 6 Equivalent structure of SIMO SDG.

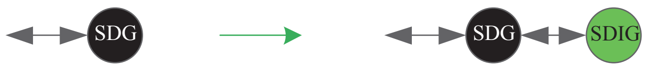


Figure 7 Equivalent structure of SIO SDG.

All the topologies can be normalized to SISO structure because of the interface's redefinable and bidirectional characteristics. The traffic processing flow shown in Figure 3 can be represented by the combination of stream topology, tree topology, ring topology, star topology, etc., and the topology's internal function can be defined by software. The SDGs in Function, no matter the SDPGs or SDIGs, are designed based on the SISO structure, whose interface meets the SDI interface based on the stream address, the physical direction (horizontal/vertical), and the defined effective bit widths. The structure is shown in Figure 8. A traffic described in TSDI can be stripped to several functions. Every function can be realized by the effective combination and connection of SDGs. Furthermore, the SDGs, comply with the SDI interface and interconnection topology, can be decomposed into the simplest topology using the interconnection of SISO structure.

3. A use case of RapidIO controller based on TSDI

As a protocol controller, the RapidIO controller has the characteristics of stream processing. As shown in Figure 9, the maintenance traffic processing flow can be simplified to five stream processing functions. Function1 mainly realizes data synchronization and decoding. The package parsing and error detection are implemented in Function2. Function3 mainly processes message forwarding. Function4 implements the routing table lookup and detection. Function5 is abstracted to parse the message.

The processing flow of maintenance package traffic can be set as the stream processing queue of Function1,

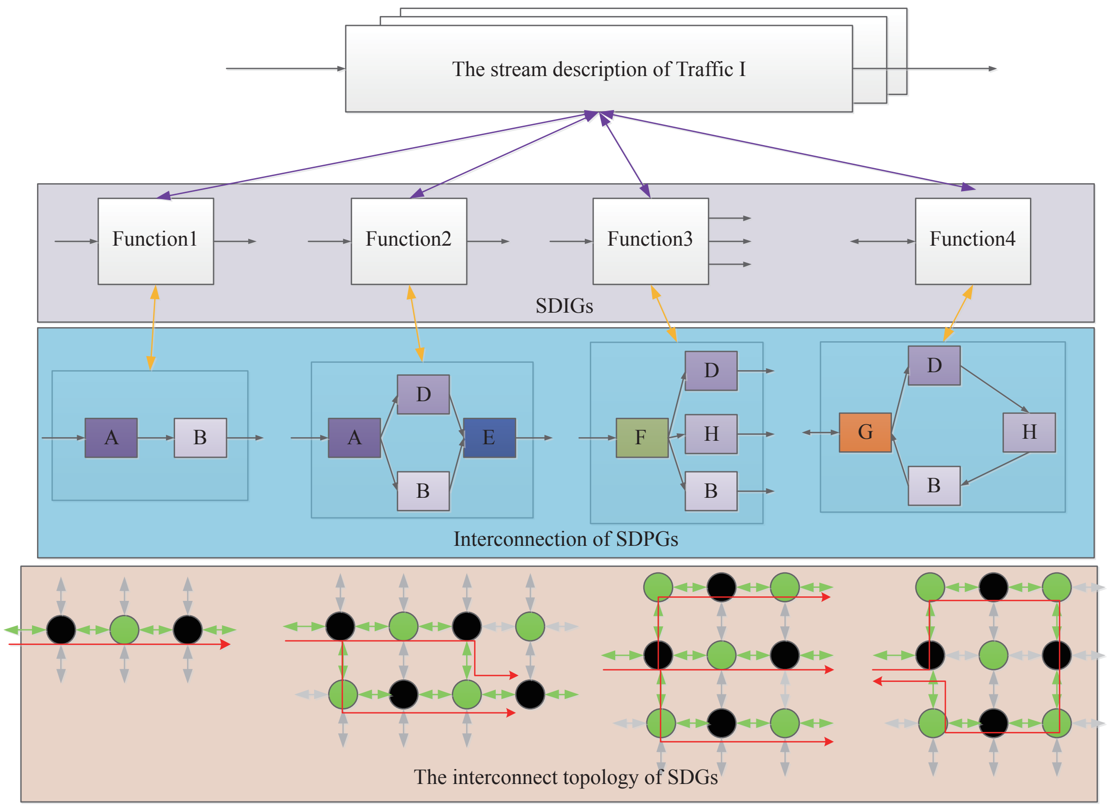


Figure 8 The interconnect topology based on SISO SDGs.

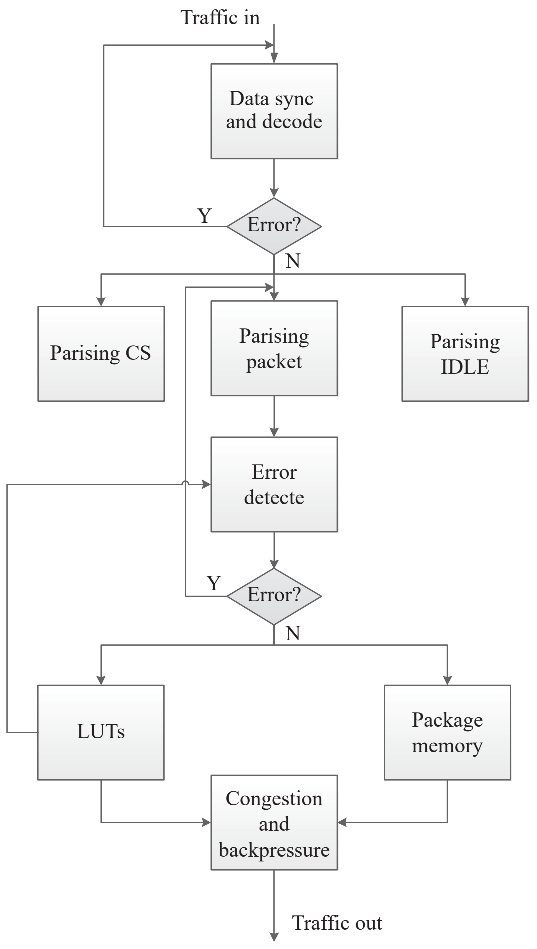


Figure 9 Processing flow of maintenance package traffic.

Function5, Function2, Function3, and Function4, shown in Figure 10. The main SDGs of Function1 include bit synchronization, decoding, and descrambling. Function2 has the SDGs of message parsing and error detection. The SDGs of error detection, routing table lookup, package storage and congestion control are implemented if Function3. The SDGs of error detection and routing lookup table are realized in Function4. And the SDGs such as descrambling, package parsing, control symbol parsing and IDLE sequence parsing are designed in Function5.

Based on the normalized SDGs and interconnection topology, the flow of maintenance packet processing can be implemented by TSDI architecture, as shown in Figure 11.

The RapidIO controller can be implemented by normalized SDGs and the interconnection topology. TSDI-based RapidIO controller first generates the optimized configuration execution file at the software level according to the traffic type and flow, then defines or redefines the interconnection topology of SDGs by static configuration files or dynamically sensing real-time traffics.

IV. Evaluation and Comparison of Achievable Parameters

A RapidIO controller compatible with Gen4/3/2/1 protocol is implemented based on TSDI as a use case. Table 1 gives the main features of this design compared with the traditional controllers [7]–[9]. The TSDI-based RapidIO controller has excellent application flexibility and protocol scalability performance. It can support more protocol versions than the dedicated architecture and achieve more than twice the throughput than IDT's design.

PPA is the best criterion for verifying the quality of the proposed architecture. This design takes the compatibility, delay, and logic resources as the key parameters of PPA to evaluate the TSDI-based RapidIO controller. An environment connecting with the standard commercial Candence VIP is set up to test the protocol compatibility and latency performance. And the Synopsys Design Compiler is used to synthesize the design for resource calculation.

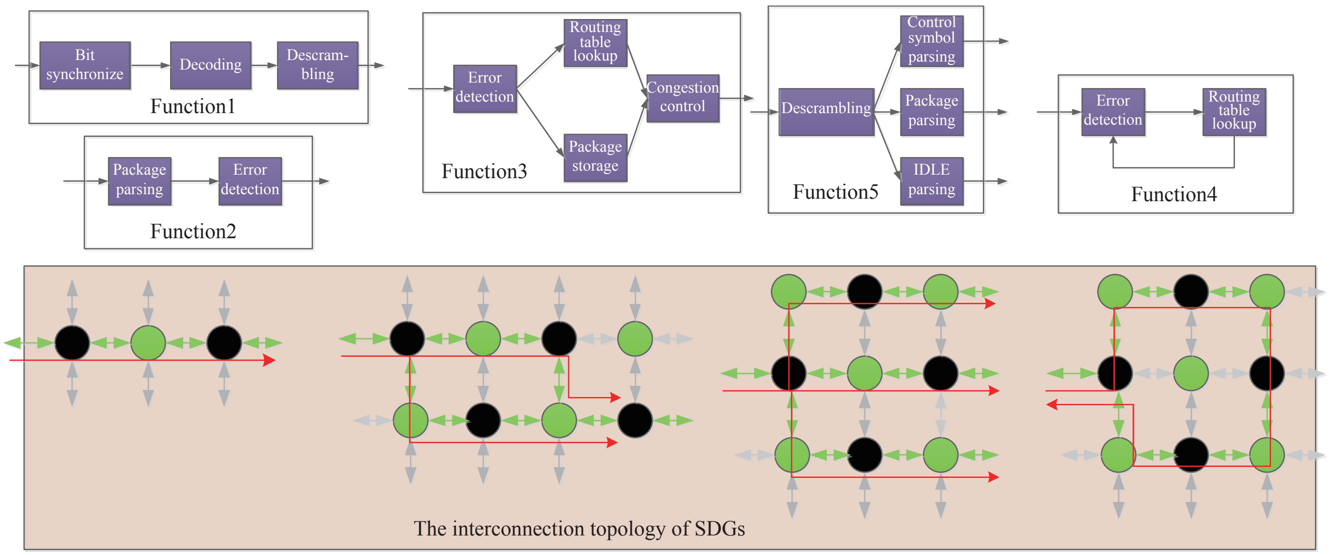


Figure 10 Function topology of maintenance package processing.

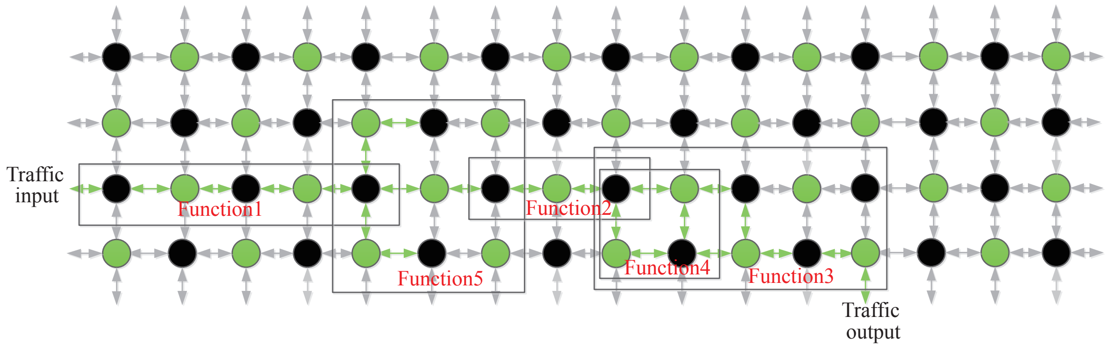


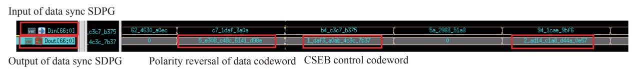
Figure 11 TSDI-based RapidIO controller topology driven by maintenance package traffic.

Table 1 Comparison of main features

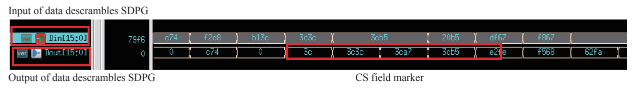
Main feature	Xilinx	Mobiveil	IDT	TSDI
Supported protocol	V2.2/1.3	V3.1/2.2/1.3	V3.2/2.2/1.3	V4.0/3.2/2.2/1.3
Path mode	1x/2x/4x	1x/2x/4x	1x/2x/4x 2x+2x 2x+1x 1x+1x	1x/2x/4x 2x+2x 2x+1x+1x 1x+1x+1x+1x
IDLE sequence	IDLE2/1	IDLE3/2/1	IDLE3/2/1	IDLE3/2/1
Parallel data width (bits)	20	10/20/40/64/67	10/16/20/32/40	10/16/20/32/40
User data width (bits)	128	256	256	64/128/256
DevID (bits)	16	8/16	8/16/32	8/16/32
Address width (bits)	34	34/50	34/50/66	34/50/66
Link training	IDLE2	IDLE2/CW/DME	IDLE2/CW/DME	IDLE2/CW/DME
Throughput (Gbps)	20	39.4	47.1	98.1
Maximum unacknowledge packets	32	4096	128	1024 configurable
V4.0 scalability	Not	Not	Not	Support

1) Protocol compatibility test

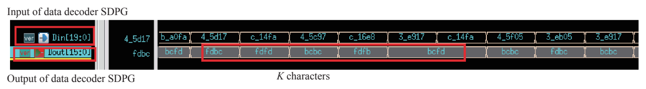
Take IDLE3 traffic processing as an example. The input of the data sync SDPG in IDLE3 traffic processing flow is the received parallel data, while the output is the synchronized data. The synchronized data can be addressed to the input of the data decoder SDPG, and the decoded data will be routed to the input of the data descramble SDPG based on the stream address. The simulation waveforms of different IDLE sequence processing are shown in Figure 12. The stream processing flow among SDPGs in the TSDI-based RapidIO controller is designed to meet RapidIO Gen4/3/2/1 protocol compatibility.



(a) RapidIO Gen4/3 IDLE3 traffic


Input of data descrambles SDPG
Din[15:0]
Dout[15:0]
Output of data descrambles SDPG
CS field marker

(b) RapidIO Gen2 IDLE2 traffic



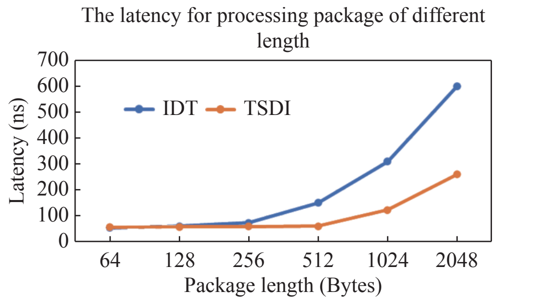
(c) RapidIO Gen1 IDLE1 traffic

Figure 12 Different traffic in TSDI-based RapidIO controller.

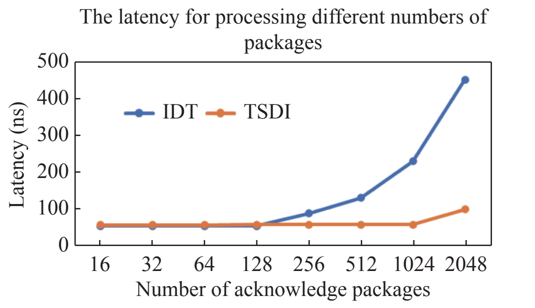
2) Latency performance test

The latency performance tests the respective transmission and reception delays with different packet lengths. The test environment connecting with the standard commercial VIP collects the test results. This article mainly compares IDT's RapidIO controller design. The path modes of both controllers are configured as 1x mode at 12.5 Gbps. And two sets of latency tests are performed. One is the latency test with a processing package of different lengths, where the final value is the average value of the measured latency repeated for 5 times. The other tests the latency curve when processing different loads with small packages (64-byte length), where the final value is the average value of measured latency repeated for 5 times. The result is plotted in Figure 13.

Because the bit width of SDGs in TSDI can be software-defined automatically, the TSDI-based controller has a stable latency value when processing packages with different lengths. When the package length changes, the bandwidth can be optimized by redefining the interface to reduce the processing cycles. The TSDI-based RapidIO controller performs better at the latency characteristics when processing different loads because the TSDI-based RapidIO controller supports acknowledging a maximum



(a) The latency for processing different length of package



(b) The latency for processing different number of package

Figure 13 Transmission delay comparison.

of 1024 unacceptable packages. Moreover, based on the SDI protocol, the controller can dynamically perceive the traffic load to adapt to the number of unacceptable packages. But when it exceeds the limit, the latency of the TSDI-based RapidIO controller is close to the IDT, showing a linear growth curve.

3) Controller resource comparison

This design is synthesized on the 28 nm process, and the target frequency is 400 MHz. The comparison results between this design and IDT's 10xN RapidIO controller are shown in Table 2. The number of Std Cell Instances has increased by 8.33%, which is mainly determined by the resource increase for the designing of SDGs and the resource decrease for the extraction of common SDGs. The storage size has been doubled to match the maximum bit width of the SDGs' interface. Overall, compared to the IDT controller, this design has a 23.4% increment in area.

Table 2 Resource area comparison

Items	IDT	TSDI	Comp. (%)
Std cell instances	600K	610K	1.67%
Std cell area (mm^2)	0.56	0.569	1.61%
Memory (bits)	214144	428288	100%
Memory area (mm^2)	0.16	0.32	100%
Total area (mm^2)	0.72	0.889	23.4%

Furthermore, the advantage of a 23.4% increment in area cost should be verified. We compared the area growth rate to the dynamic reconfigurable architecture and the switching on/off scheme in [12]. The comparison results are shown in Table 3.

Table 3 Area growth comparison

Items	The dynamic reconfigurable architechure	The switching on/off scheme	TSDI
Area growth (%)	33.3	55.8	23.4

V. Conclusions

In this paper, we propose a protocol controller architecture based on the traffic-driven software-defined interconnection, define an SDI interface of SDGs and normalize the interconnection topology. Given that the stream processing characteristic, the architecture of the protocol controller can be implemented as the interconnection of a series of SDGs. A RapidIO controller is designed based on TSDI. According to the results of evaluation, it can be concluded that the TSDI-based controller can exchange the limited resources for flexibility and scalability. In the future, the low-power design of SDGs and the software design for more flexible interconnection control will be further considered to get more efficient.

Acknowledgement

This work was supported by the National Science and Technology Major Project (Grant No. 2016ZX010 12101).

References

[1] P. Lv, Q. R. Liu, J. X. Wu, *et al.*, "New generation software-defined architecture," *Scientia Sinica Informationis*, vol. 48, no. 3, pp. 315–328, 2018. (in Chinese)

[2] J. X. Wu, "Thoughts on the development of novel network technology," *Science China Information Sciences*, vol. 61, no. 10, article no. 101301, 2018.

[3] X. T. Guo, Y. W. Lei, and Y. Guo, "Design and implementation of dual-channel serial RapidIO for multiple transmission modes," *Computer Engineering & Science*, vol. 41, no. 2, pp. 233–239, 2019. (in Chinese)

[4] C. F. Dan, J. Li, S. L. Jing, *et al.*, "Design of RapidIO bus system based on software configuration," *Microcontrollers & Embedded Systems*, vol. 20, no. 7, pp. 11–14, 2020. (in Chinese)

[5] J. C. Shen, "Design of DMA high-speed transmission scheme based on general RapidIO controller," *Microcontrollers & Embedded Systems*, vol. 20, no. 7, pp. 20–24, 2020. (in Chinese)

[6] Xilinx, "Serial RapidIO endpoint LogiCORE IP product guide 7," Xilinx, 2017.

[7] Mobiveil, "Mobiveil RapidIO controller (GRIO)," Mobiveil, 2016.

[8] IDT, "RapidIO-IP-10xN user manual," IDT, 2017.

[9] M. Cicioğlu and A. Çalhan, "A multiprotocol controller deployment in SDN-based IoMT architecture," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 20833–20840, 2022.

[10] E. Hrynkiewicz and M. Chmiel, "Programmable logic controller-basic structure and idea of programming," *Electrical Review*, vol. 88, no. 11b, pp. 98–101, 2012.

[11] E. A. Suvorova and V. V. Rozanov, "Dynamic reconfigurable packet distribution unit for embedded systems," in

2019 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF), St. Petersburg, Russia, pp.1–9, 2019.

[12] E. Suvorova, "An approach to dynamic reconfigurable transport protocol controller unit development," in *2020 26th Conference of Open Innovations Association (FRUCT)*, Yaroslavl, Russia, pp.429–437, 2020.

[13] A. Karatkevich, A. Bukowiec, M. Doligalski, *et al.*, *Design of Reconfigurable Logic Controllers*. Springer, Cham, Switzerland, 2016, doi: 10.1007/978-3-319-26725-8.

[14] M. Tsavos, N. Sklavos, and G. P. Alexiou, "Lightweight security data streaming, based on reconfigurable logic, for FPGA platform," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, Kranj, Slovenia, pp.277–280, 2020.

[15] S. Xydis, G. Economakos, D. Soudris, *et al.*, "High performance and area efficient flexible DSP datapath synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, pp. 429–442, 2011.

[16] S. Xydis, G. Palermo, and C. Silvano, "Thermal-aware datapath merging for coarse-grained reconfigurable processors," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, pp.1649–1654, 2013.

[17] E. A. Suvorova, "An approach for development of RISC- V based transport layer controller," in *2021 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*, St. Petersburg, Russia, pp.1–9, 2021.

[18] P. D. Schiavone, D. Rossi, A. Di Mauro, *et al.*, "Arnold: An eFPGA-augmented RISC-V SoC for flexible and low-power IoT end nodes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 677–690, 2021.

[19] P. Lv, Q. R. Liu, H. C. Chen, *et al.*, "Domain-oriented software defined computing architecture," *China Communications*, vol. 16, no. 6, pp. 162–172, 2019.

Peijie LI received the M.S. degree in communication engineering from Information Engineering University in 2014. Since September 2014, he has been working as a Research Assistant at the Institute of Information Technology in Information Engineering University. He is currently working towards the Ph.D. degree in Information Engineering University. His research interests include software defined hardware, SerDes, and System on Wafer. (Email: lpj@ndsc.com.cn)

Jianliang SHEN received the Ph.D degree in National University of Defense Technology. He is currently an Associate Professor of Information Engineering University. His research interests include software defined interconnection, system architecture design and SoC technology. (Email: sjl@ndsc.com.cn)

Ping LYU received the Ph.D degree in communication engineering from Information Engineering University in 2019. Currently, she is a Professor and M.S. supervisor. Her research interests include new generation network information system architecture, and she is engaged in Large scale integrated circuit design. (Email: lp@ndsc.com.cn)

Chunlei DONG received the M.S. degree in microelectronics from University of Chinese Academy of Sciences in 2014. Since September 2014, he has been working as a Research Assistant at the Institute of Information Technology in Information Engineering University. His research interests include software defined Interconnection, switching fabric, and system on wafer. (Email: dcl@ndsc.com.cn)

Ting CHEN received the B.S. degree in microelectronics from University of Electronic and Scientific Technology of China in 2008, M.S. and Ph.D degrees in electrical science and technology from the National University of Defense Technology, China, in 2010 and 2014, respectively. He is currently working as a Research Assistant at the Institute of Information Technology in Information Engineering University. His research interests include high performance interconnection structure, parallel processing architectures, and circuits. (Email: ct@ndsc.com.cn)