Channel Slicing: a Way to Build Fast Routers for Asynchronous NoCs

Wei Song and Doug Edwards
School of Computer Science, University of Manchester, Manchester, M13 9PL UK
{songw,doug}@cs.man.ac.uk

***Abstract*—Asynchronous on-chip networks are power efficient and tolerant to process variation but they are slower than synchronous on-chip networks. One reason for the low speed is the way that asynchronous routers use to build wide channels. To meet the bandwidth requirement, current routers broaden their channels by synchronizing multiple sub-channels. The C-element tree in the completion detection circuit increases the cycle period. A low latency asynchronous wormhole router is proposed using sliced sub-channels. Channel slicing removes the C-element tree in the completion detection circuit and convert a channel into multiple independent sub-channels reducing the cycle period. The router is implemented by a 0.13 μm technology. The cycle period of the router at the typical corner is 2.2 ns, providing 1.82 GByte/sec throughput per port.**

I. Introduction

Network-on-chip (NoC) [1] provides a scalable on-chip communication architecture for current multi-processor system-on-chip (MPSoC) systems. The on-chip network could be a synchronous network where routers are driven by a global clock, or an asynchronous network where routers are self-timed circuits connected by asynchronous pipelines. Thanks to mature EDA tools and the timing assumptions allowed by the global clock, synchronous networks are fast and area efficient but the clock tree is power consuming [2]. By contrast, the clockless asynchronous networks are comparatively slow but power efficient. In addition, they are tolerant to process variation and could divide the whole chip into several isolated clock domains, which unifies the network interface and shortens the overall design time.

In this paper, a low latency asynchronous router is designed using sliced channels. The state-of-the-art quasi delay-insensitive (QDI) pipelines in routers are built by synchronizing multiple bit-level pipelines (sub-channels) [3], [4], [5], [6], [7]. The C-element tree in the completion detection circuit (synchronization circuit) increases the cycle period and reduces throughput. Instead of synchronizing sub-channels, we propose to use sub-channels in parallel. Since the C-element tree is removed, sub-channels run faster than the synchronized channel. Extra controllers are added to resynchronize sub-channels during special intervals, such as the route decision procedure.

The proposed router is implemented using a 0.13 μm standard cell library and the cycle period is 2.2 ns, providing 1.82 GByte/sec throughput per port.

The remainder of this paper is organized as follows: section II describes the general architecture of the on-chip network. Section III explains how channel slicing can improve speed. Section IV demonstrates the detailed implementation of the router. Section V shows the simulation results of the implementation and analyzes the impact of pipeline data width on area and speed. Finally the paper is concluded in section VI.

II. Network Architecture

A network node in a globally asynchronous and locally synchronous (GALS) network comprises a processor element, a network interface and a router. The processor element could be a local system controlled by a processor or a hardware IP running a specific function. Serving as a slave device to the processor element, the network interface provides a duplex channel for the processor element to communicate with the chip level asynchronous network. To ease the network communication, the network interface splits the frames generated by the local processor element into flits of fixed length before sending them to routers. It also regroups received flits into frames before delivering them to the local processor element. In a GALS network, the network interface also serves as a synchronous/asynchronous adaptor to ensure the faultless cross timing domain data transmission. Similar to the routers used in macro networks, routers in on-chip networks are distributed route deciders and message delivers but with tighter area budget and higher throughput requirement. They are fully asynchronous circuits in the proposed GALS network.

This paper concentrates on the wormhole flow control method and the hardware implementation of asynchronous routers; therefore, all other design aspects are set to broadly accepted configurations. A mesh topology is used due to its easy mapping on a 2-D layout. Frames are routed by the XY dimension order routing algorithm. Nodes in the network are identified by a (x, y) address. Network interfaces have enough buffer space to guarantee that a flit is consumed by a network interface in finite time. The network is assumed to be error-free so that no deadlock or livelock occurs. The data width of all ports is set to 32-bit to meet the throughput requirement for a normal multi-processor SoC application. A flit is also 32 bits and is transmitted in one cycle. A frame comprises a head flit, several data flits and a tail flit. The head flit contains a 1 byte address, denoting the target node, and 3 bytes data. The maximal size of a network is 16x16.

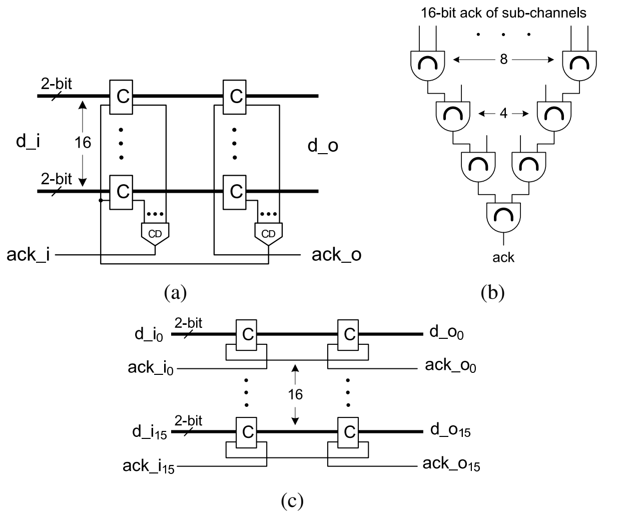


Fig. 1. (a) A 32-bit 1-of-4 pipeline, (b) the completion detection circuit and (c) the channel sliced pipeline

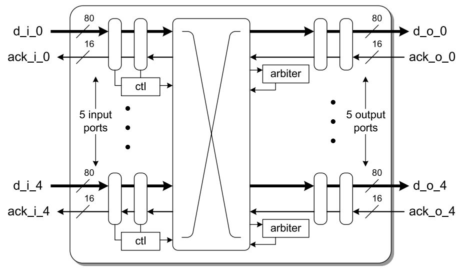


Fig. 2. Router structure

III. Channel Slicing

Many handshake protocols could be used to build asynchronous circuits but only some of them are suitable for asynchronous router designs. The 4-phase bundled-data protocol has been used in MANGO [8], QNoC [9] and ASPIN [7]. The 4-phase dual-rail protocol has been used in ASPIN [7] and the 4-phase 1-of-4 protocol has been used in CHAIN [3], QoS [4] and ANoC [5]. Finally, m-of-n protocols have been used in SpiNNaker [6].

The 4-phase 1-of-4 protocol is preferred. Bundled-data protocols work under cautious timing constraints and the matched delay lines are vulnerable to process variation [8]. M-of-n protocols transmit more data bits in one cycle than the 1-of-4 protocol but they need extra decoders and encoders [10]. Because the address in the head flit is analyzed by every router on the path, a decoder is added on each input port to translate the head flit, which introduces area overhead. The 4-phase 1-of-4 protocol is QDI, comparably area efficient than m-of-n protocols and more power efficient than the dual-rail protocol.

In all QDI routers [3], [4], [5], [6], [7], a wide channel is built by synchronizing multiple bit-level sub-channels, such as the 32-bit 1-of-4 channel shown in Figure 1(a). The synchronized channel behaves similar to the pipeline formed by flip-flops in synchronous circuits. Techniques used in synchronous routers, such as the virtual channel, could be easily adopted. However, the completion detection (CD) circuit is a 16-input C-element tree, shown in Figure 1(b). Assuming that all 2-input gates have the same latency and the C-element is a two level combinational logic, this completion detection circuit has 8 levels of logic. As the forward path of a basic 1-of-4 pipeline only has 4 levels of logic, the completion detection circuit accounts for 66% of the cycle period.

Synchronization is necessary for timing division multiple access (TDMA) technologies, such as the virtual channel flow control, but not for wormhole routers. According to the wormhole flow control method, a route is decided and reserved by the head flit and data flits simply follow the head flit. Since no frames could prevent data flits from following the head flit, no synchronization is needed. We propose to slice the synchronized channel into sub-channels, illustrated in Figure 1(c), to allow independent data transmission on sub-channels. Extra controllers are added to ensure that the head flit is successfully analyzed.

IV. Router Design

A. Router Structure and Data Flow

Figure 2 shows the internal structure of the proposed router. A router has five input and five output ports for four adjacent routers and the local network interface. A buffer with two pipeline stages is added on each input port and output port. Input buffers and output buffers are connected by a crossbar configured by the arbiter on each output port. Route decisions are made on each input buffer and a route is reserved by obtaining a grant from the corresponding arbiter on the output port. Since sub-channels run independently, they have their own ack wires. An end-of-frame (EOF) wire is also added to each sub-channel to identify the tail flit. As a result, one sub-channel has five data wires and one ack wire, the same as Chain [3]. A 32-bit channel has 16 sub-channels. Every port contains 80 data wires and 16 ack wires.

The basic wormhole data flow is slightly changed due to the removed synchronization. Figure 3 shows the modified data flow. A flit is sliced into 16 parts and each of them is transmitted on a sub-channel. The head flit is firstly blocked in the first stage of the input buffer. Then the control logic analyzes the address in the head flit and makes a request to one of the arbiters. After the request is granted, a path is reserved in the crossbar and the frame is delivered by independent sub-channels. The crossbar is reset by the input buffer once all parts of the tail flit are delivered.

The head flit is blocked in the first stage of the input buffer instead of the last stage as in ASPIN [7] for two reasons: firstly, it reduces the fan-out of the second stage which is on the critical cycle; secondly the route decision procedure and the crossbar reset proceed in parallel.

B. The Data Path of a Sub-channel

Figure 4 illustrates the data path of a single sub-channel. rt_err and acken are two signals driven by the extra

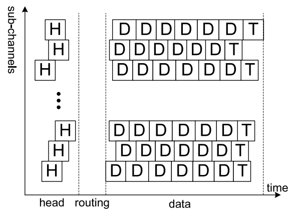


Fig. 3. The modified wormhole data flow

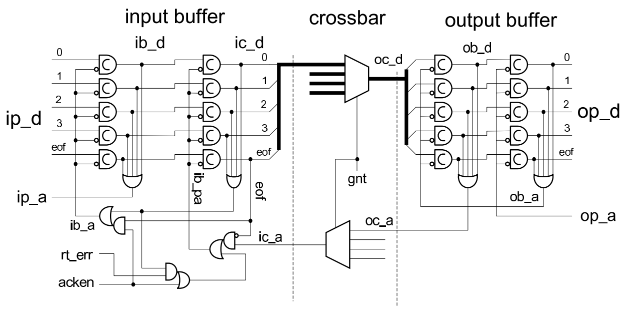
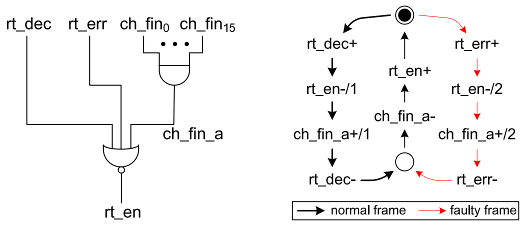
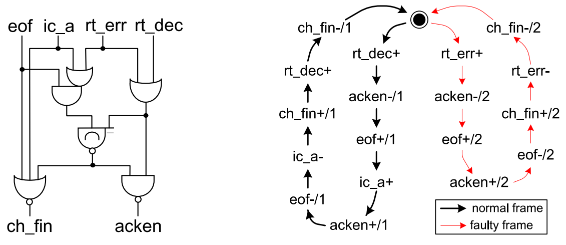


Fig. 4. The data path of a sub-channel



(a) (b)

Fig. 5. (a) A route decision controller and (b) its STG



(a) (b)

Fig. 6. (a) A sub-channel controller and (b) its STG

controller added on each sub-channel. acken, the active low signal enabling the data path, is set low after a route request is initiated and it is driven to high to stall the data path when the tail flit is detected on ic_d. rt_err indicates incorrect route requests and is set high when a faulty frame is going to be dropped. gnt is the grant result from arbiters (section IV-C), which enables the MUXes and DEMUXes in the crossbar.

Although sub-channels run in parallel during the data session, they stall after the tail flit to keep the next head flit in the first pipeline stage in the input buffer. An input buffer has one route decision controller and several sub-channel controllers, one for each sub-channel. For an incoming frame, the route decision controller enables the route decision procedure. Once a route request is initiated, sub-channel controllers enable their data paths.

Figure 5 demonstrates the internal structure of the route decision controller and its STG. The route decision procedure is always enabled through rt_en+ after a frame is transmitted. A route decision could be a possible route request (rt_dec+) or a faulty request (rt_err+). The frame generating a faulty request will be dropped. After the route request is made, the route decision procedure is disabled until the frame is transmitted, denoted by ch_fin+ on all sub-channels.

Figure 6 shows a sub-channel controller and its STG. A data session begins after a route request is made. A faulty frame is dropped by connecting the ack line generated from ic_d directly to itself, enabled by rt_err in Figure 4. Note that the ack line connected back is generated from data bits but not the EOF bit to guarantee that the EOF bit is always detected by the sub-channel controller. When the tail flit arrives, it is dropped by acken+ and then the sub-channel stalls until the next data session. For normal frames, the ack line ic_a from output buffers is used and data are forwarded to the requested output port.

C. Routing and Arbitration

As an example, Figure 7 shows the route decision circuit in the south input buffer and the connected arbiter on the east port. Enabled by rt_en, the 8-bit address (16-bit in 1-of-4 code) blocked in the first pipeline stage enters comparators after the second pipeline stage is cleared (ib_a is low). The route request is captured by the C2P elements enabled by ch_fin_a-. One-hot coded, the route request drives rt_dec or rt_err to ‘1’, which then disables the route decision procedure and starts the data session. C2P elements hold the value during the whole data session. The south input buffer could not be connected with the south output buffer, therefore, the corresponding route request is connected to rt_err.

Valid route requests are sent to arbiters. Since only four input buffer could request to one output port concurrently, the multi-way MUTEX arbiter [11], shown in Figure 7, is faster and smaller than other arbiter styles [12], [13], [11]. The successful request is granted by one of the four gnt outputs.

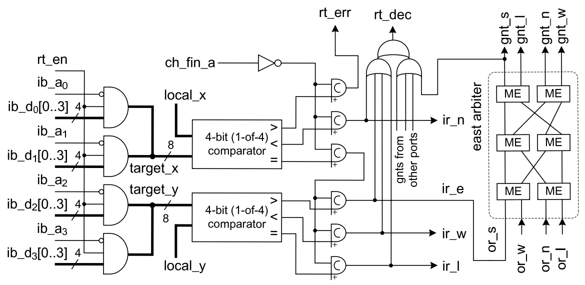


Fig. 7. The routing decision circuit and the arbiter

V. Performance

A. Physical Implementation

The router has been implemented using the Faraday 0.13 μm standard cell library based on the UMC 0.13 μm technology. Route decision controllers and sub-channel controllers are speed independent circuits generated from their STGs using Petrify [14] and other parts are manually written in Verilog HDL.

The area after synthesis is around 12.6K gates (0.050 mm^2). The final router is placed and routed on a 0.3x0.3 mm block using 5 metal layers. The speed simulation is back-annotated with the RC extraction from the layout and run under the typical corner (25 °C, 1.2 V). The cycle period for data flits is 2.2 ns, providing maximal 1.82 GByte/s throughput on a single port. The average latency of a data flit is 2.1 ns. For the head flit, the routing decision and the arbitration procedures consume about 0.8 ns without contention.

B. Effect of Data Width

An extra and important advantage of channel slicing is that the cycle period does not increase with the width of data paths. For normal QDI pipelines, increasing the data width means more sub-channels are synchronized by the C-element tree leading to a larger speed penalty. The bundled-data router in QNoC also reported similar effect (0.2% per bit degradation [9]). However, for the routers using channel slicing, sub-channels run independently during the data session. Increasing the number of sub-channels has little impact on the cycle period of a single sub-channel.

Both the routers using synchronized channels (the traditional way) and the routers using sliced channels have been implemented with different data widths. Figure 8 shows the results after synthesis. For both routers, the area increases linearly with the data width. The channel sliced routers are slightly larger than the traditional routers due to the extra sub-channel controllers and the increased wire count. However, the channel sliced routers are significantly faster. The cycle period of the channel sliced routers remains around 2.2 ns but the cycle period of the traditional routers increases along with the data width (around 2.8 ns for the 32-bit case).

VI. Conclusion

In this paper, a low latency asynchronous router has been implemented using channel slicing. Channel slicing removes the C-element tree in the completion detection circuit of QDI pipelines. This removal reduces the cycle period and make sub-channels run in parallel during the data session. The final router has been implemented on a 0.3x0.3 mm block using the Faraday 0.13 μm standard cell technology. The synthesis result is around 12.6K gates. Simulations are back-annotated with RC extraction and run at the typical corner. The cycle period is around 2.2 ns providing 1.82 GByte/sec throughput on each port.

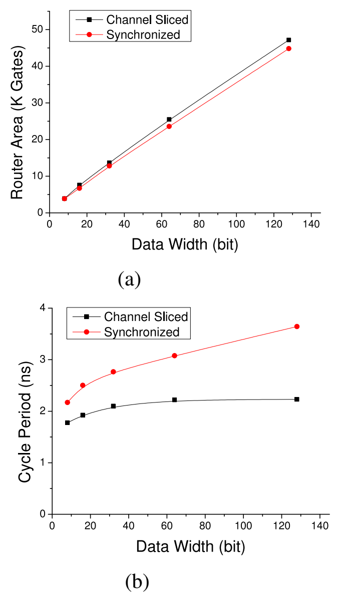


Fig. 8. (a) Area and (b) cycle period under different data width

References

[1] W. J. Dally and B. Towles, “Route packets, not wires: on-chip interconnection networks,” in *Proc. of DAC*, 2001.

[2] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Oberg, P. Ellervee, and D. Lundqvist, “Lowering power consumption in clock by using globally asynchronouslocally synchronous design style,” in *Proc. of DAC*, 1999, pp. 873–878.

[3] J. Bainbridge and S. Furber, “Chain: a delay-insensitive chip area interconnect,” *IEEE Micro*, vol. 22, pp. 16–23, 2002.

[4] T. Felicijan and S. B. Furber, “An asynchronous on-chip network router with quality-of-service (qos) support,” in *Proc. of SOCC*, Sept. 2004, pp. 274–277.

[5] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, “An asynchronous NOC architecture providing low latency service and its multi-level design framework,” in *Proc. of ASYNC*, March 2005, pp. 54–63.

[6] L. A. Plana, S. B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, “A globally asynchronous, locally synchronous infrastructure for a massively-parallel multiprocessor,” *IEEE Design and Test of Computers*, vol. 24, no. 5, pp. 454–463, 2007.

[7] A. Sheibanyrad, “Asynchronous implementation of a distributed network-on-chip,” Ph.D. dissertation, University of Pierre et Marie Curie, 2008.

[8] T. Bjerregaard and J. Sparsø, “A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip,” in *Proc. of DATE*, 2005, pp. 1226–1231.

[9] R. Dobkin, R. Ginosar, and A. Kolodny, “QNoC asynchronous router,” *Integration, the VLSI Journal*, vol. 42, no. 2, pp. 103–115, 2009.

[10] J. Bainbridge, W. Toms, D. Edwards, and S. Furber, “Delay-insensitive, point-to-point interconnect using m-of-n codes,” in *Proc. of ASYNC*, May 2003, pp. 132–140.

[11] D. J. Kinniment, *Synchronization and Arbitration in Digital Systems*. John Wiley & Sons Inc., 2007.

[12] K. S. Low and A. Yakovlev, “Token ring arbiters: An exercise in asynchronous logic design with Petri nets,” Newcastle University, Tech. Rep., 1995.

[13] M. B. Josephs and J. T. Yantchev, “CMOS design of the tree arbiter element,” *IEEE Transactions on VLSI*, vol. 4, no. 4, pp. 472–476, Dec 1996.

[14] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, “Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers,” *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, 1997.