Chapter 2
Network on Chip Aspects

2.1 Introduction

NoC infrastructure is composed of routers, NIC, and interconnects as shown in Fig. 2.1. Routers are connected with their neighbors using multiple number of interconnects. Each PE is connected to a local port of a router through a NIC. The network interface controller adapts the messages from the PEs to NoC routers and vice versa. Depending on the structure of the PEs, NoC can be classified into two categories, homogeneous NoCs and heterogeneous NoCs. In homogeneous NoC, the internal structure of PEs is the same and the chip is symmetric while the PEs in a heterogeneous NoC may use different structures.

Data transfer in NoC could be done using different schemes of clocking. In NoC design, the most popular clocking schemes are synchronous, mesochronous, plesiochronous, and pseudochronous. In order to synchronize the nodes in the NoC, clock distribution network is required in synchronous system. On the other hand, asynchronous circuits are clockless circuits and all communication is based on the use of handshaking to organize the data transfer. The synchronization of data in both NoC designs is presented in Sect. 2.2. The chosen routing algorithm is considered the most important aspect that discriminates the NoC architectures. The classification of routing algorithms is demonstrated in Sect. 2.3. Conclusions of this chapter are presented in Sect. 2.4.

2.2 Synchronization in NoC

The synchronous clocking schemes and the synchronization issues are summarized in Sect. 2.2.1. On the other hand, the building blocks of asynchronous designs, handshake protocols, communication channel, data encoding, and protocols converters are provided in Sect. 2.2.2.

© Springer International Publishing Switzerland 2016

R. Ezz-Eldin et al., *Analysis and Design of Networks-on-Chip Under High Process Variation*, DOI 10.1007/978-3-319-25766-2_2

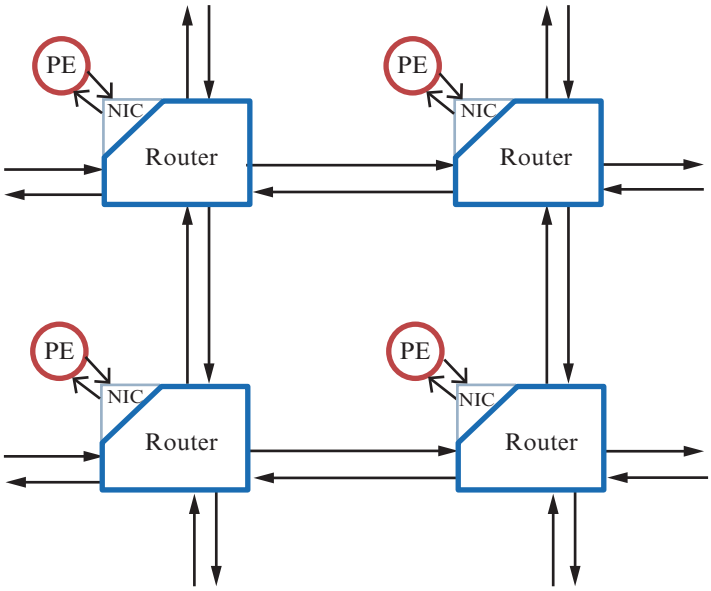


Fig. 2.1 NoC mesh topology

2.2.1 Synchronous Designs

In synchronous system, the clock signal serves as a global timing reference for communicating data among the different modules to synchronize the data processing elements. In addition, the clock signal is used as an enable signal for all latches to ensure correct timing in synchronous design. The single-clocking schemes are the available solution to distribute one clock signal and tie all data signals to that clock signal. The single-clocking schemes are categorized into synchronous, mesochronous, plesiochronous, and pseudochronous as described in Sect. 2.2.1.1. Different clock distribution strategies for NoC design are described in Sect. 2.2.1.2. The synchronization failure and metastability are demonstrated in Sect. 2.2.1.3.

2.2.1.1 Single-Clocking Schemes

The single-clocking schemes depend on adjusting the data sampling, by delaying either the local clock or the arriving signal, to keep signal transitions from unsafe regions [1]. The different categories of the clocking schemes are described in the following sections.

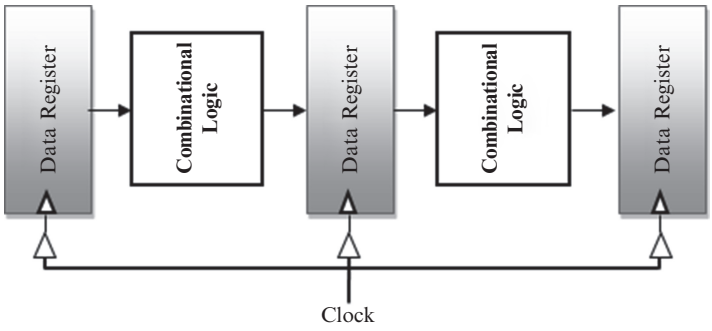


Fig. 2.2 Synchronous scheme

A. Synchronous Clocking Scheme

One clock signal is required to synchronize all parts of the entire system by distributing a global clock signal over a chip area. For synchronous NoC designs, complex function blocks are subdivided into smaller combinational logic blocks and then the registers are inserted to separate them as shown in Fig. 2.2 [2]. The global clock is distributed to clock all registers at the same time. At each active clock edge, the data signal leaves the register, propagates through the combinational network, moves to the next register, and is fully latched into that register before the next clock signal appears.

To satisfy critical worst case timing constraints, pipeline registers are inserted into equally spaced timing windows to achieve the global performance and local timing requirements. The clock period is limited by the critical path delay in the circuit. Hence, the critical path delay must be less than the fixed clock period, otherwise the system malfunctions. Synchronous clocking scheme is designed based on the size of system and the clock frequency.

B. Mesochronous Clocking Scheme

Mesochronous clocking scheme is used to solve the problems of traditional synchronous scheme such as the limitation of maximum clock frequency by the delay. Mesochronous scheme uses clocks with same frequencies but different phases [3–5]. Mesochronous design is marked by multiple clock domains which run at the same frequency but have unknown phase relations between them as shown in Fig. 2.3. Due to delays in the clock path, the phase relation between the clocks is uncertainly known. Thus, the output data D_1 is not synchronized with the clock $\varnothing_B$ of the receiver. Mesochronous clocking is well suited for NoC-based systems, which are intrinsically modular [1].

Potentially, a mesochronous NoC can have lower latency than a synchronous NoC because the clocks are not necessarily aligned in phase [6]. However, a detection of the correct phase for the received data is considered a big issue in mesochronous

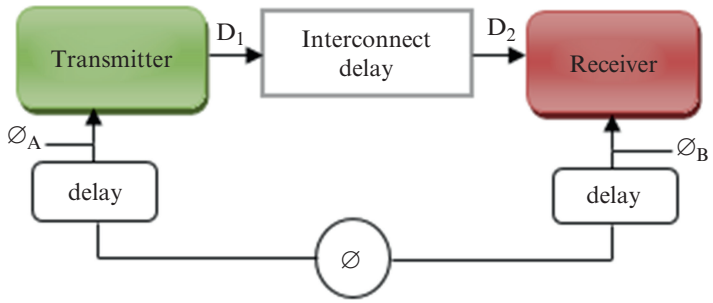


Fig. 2.3 Mesochronous scheme

design. Several solutions are proposed to solve synchronization problems in mesochronous design. In [4], Self-Tested Self-Synchronization (STSS) is presented as a mesochronous clocking in each PE. STSS automatically selects the proper clock edge for sampling the data by using a test pattern that alternates between zero and one. Two series D Flip-Flops should have different values when the test pattern is received correctly. Other clock edge is used, when the test pattern is not received correctly. Globally Updated Mesochronous design (GUM) is proposed in [5] as another solution for this problem. In GUM, the mesochronous design is split into several modules and each one is implemented using a synchronous clocking scheme with separate clock delay circuitry for calibration. Bidirectional data links are used to communicate between all modules to adjust the time delays. A calibration phase is used to establish synchronization among all modules.

C. Plesiochronous Clocking Scheme

Plesiochronous clocking scheme is used in large-scale networks. Hence, it is not practical to share a common clock source between the transmitter and the receiver. In plesiochronous clocking scheme, the clock distributions have nominally identical frequencies with slight phase mismatch. Each clock is generated separately but are closely matched in frequency. The variable phase margin must periodically be adjusted at the receiver for synchronization, taking into consideration the avoidance for duplicating or missing data. As shown in Fig. 2.4, the transmitter module is clocked by CLK_1, which is plesiochronous with respect to CLK_2. The variable phase margin unit is exploited to derive CLK_3 from the data sequence for buffering the data in a FIFO. Therefore, CLK_3 is synchronous with the data at the input of the FIFO and is mesochronous with CLK_1. A mismatch between the clock frequencies of the transmitter and the receiver modules exists. Hence, data might have to be dropped if the transmit frequency is higher, or it can be duplicated if the transmit frequency is lower than the receive frequency. In [7, 8], a plesiochronous clocking scheme is used between routers to reduce the global clock routing problem. Therefore, every router across the chip is operating with one clock signal.

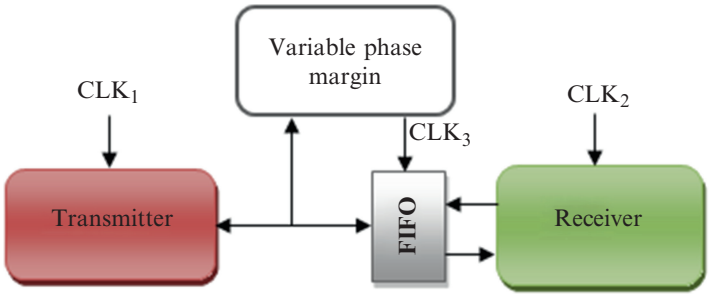


Fig. 2.4 Plesiochronous scheme using FIFO

D. Pseudochronous Clocking Scheme

Pseudochronous (pseudo-synchronous) clocking scheme uses clock distribution with the same frequency and constant phase difference [6, 9]. A digital programmable delay generator is used to adjust the phase difference among the clock regions. Pseudochronous can be used in NoC if the phase constant is selected carefully to guarantee the arrival of data before the clock.

Various techniques have been developed but each one has its own limitations and drawbacks. One of the most challenging aspects of NoC design is the distribution of the clock signal to every gate on the chip and ensures that the critical timing requirements are satisfied. Different clock distribution strategies are introduced in the following sections.

2.2.1.2 Clock Distribution Network

The timing requirement is considered as one of the most critical constrains in the design of clock distribution network. In fact, system level trade-off between system speed, physical die area, and power dissipation is significantly affected by the design of the clock distribution network [2]. Different clock distribution strategies such as buffered clock distribution trees and symmetric trees are discussed in the following sections.

A. Buffering Clock Distribution Trees

To build a tree structure, the buffers are inserted at the clock source and/or along a clock path. Buffers are inserted to amplify the clock signals which is degraded by the distributed interconnect impedance and isolate the local clock nets from the upstream load impedance [10]. The clock source is inserted at the root of the tree as shown in Fig. 2.5 [2]. The initial portion of the tree is described as trunk while the individual paths of the tree is known as branches. Finally, registers being driven are the leaves as shown in Fig. 2.5.

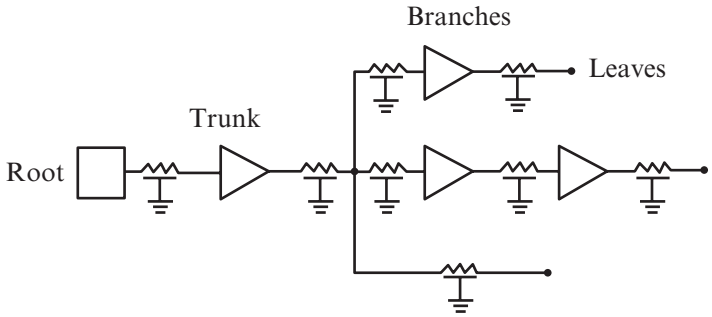


Fig. 2.5 Tree structure of clock distribution network [2]

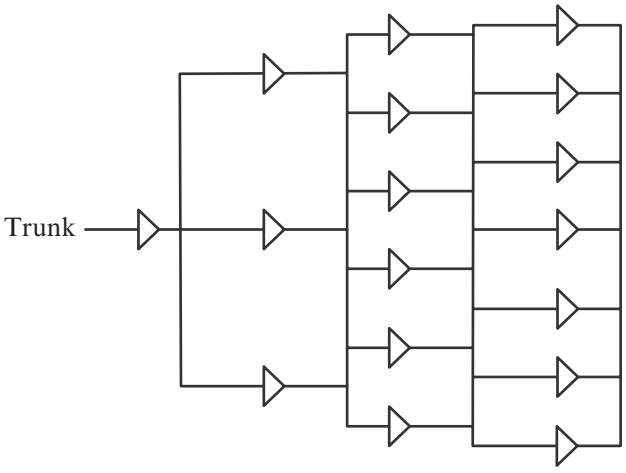


Fig. 2.6 Mesh clock distribution structure [2]

Occasionally, the mesh structure of the clock trees, as shown in Fig. 2.6, is considered as an extended structure of the standard structure [2]. Mesh network is used to minimize the interconnect resistance. Furthermore, it decreases clock skew and improves the reliability of the circuit [2].

B. Symmetry in Clock Distribution Network

Skew minimization is the target for almost all symmetric clock distribution networks such as H-tree structure and X-tree structure as shown in Fig. 2.7a, b, respectively. The clock skew depends upon the physical size, the semiconductor process, the number of active buffers, and the clocked latches distributed within the clock distribution network [2]. H-tree is widely exploited to eliminate the clock skew since the basic building block at each level of the distribution network is a symmetric H-structure. Identical clock signals are provided at the end of each leaf in

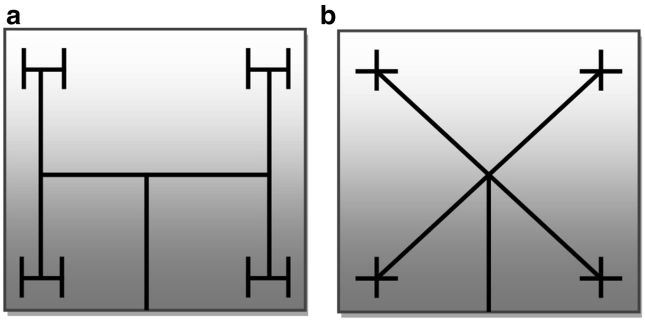


Fig. 2.7 Symmetric (**a**) H-tree and (**b**) X-tree

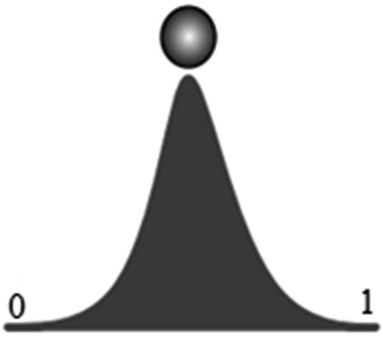


Fig. 2.8 Metastable state

H-structure to the next level of smaller H-structure routes. H-tree structure allows the clock signal to arrive at the leaves of the tree at the same time because the same delay exists at each path from the clock source to a clocked register. The delay difference between the clock signal paths (clock skew) in H-tree structure is the result of increasing the process variation parameters with the advanced technology as demonstrated in Chap. 5.

2.2.1.3 Synchronization Failure and Metastability

Successful clock domain translation is necessary to avoid metastability which causes faulty communication in the logic circuitry. Flip-Flops (FF) are susceptible to metastability. FFs have two well-defined stable states, logic 0 and logic 1. However, a metastable state is identified between these two stable states as shown in Fig. 2.8. If the input data is not valid when it is sampled, a FF may be set on a metastable state. FFs stay on the metastable state until one of the two stable states is entered randomly [11].

The delay of FF is divided into two main regions, deterministic and metastable [11]. The delay of FF in deterministic region is determined by the setup time while the delay of the flip-flop in the metastable region is not resolved [12]. At synchronous

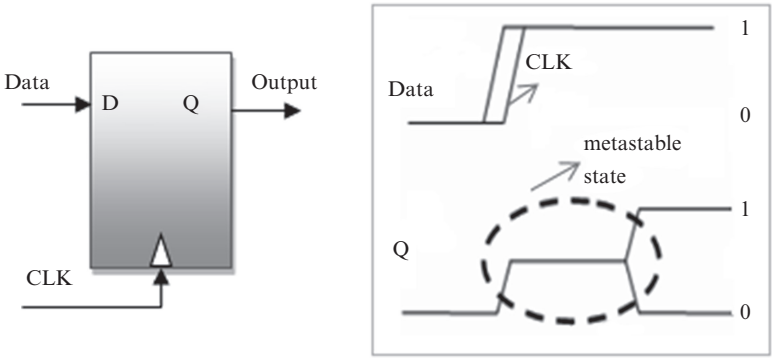


Fig. 2.9 Example of metastability in FF

system, when the metastability occurs in a FF, a limited amount of time is required to settle on a single stable state. Therefore, a synchronization failure occurs in a synchronous system when FF does not resolve the metastability in the required time.

For example, when the data arrives very close to the rising edge of the local clock and violates the setup condition, metastability may occur at the output of the flip-flop as shown in Fig. 2.9.

The most critical issue of a synchronous clocking scheme is that all blocks are timed with one common clock signal. Whereas, asynchronous schemes transfer the information without the necessity for using a clock signal. The details of asynchronous designs are presented in the next section.

2.2.2 Asynchronous Designs

Asynchronous circuits represent a class of circuits which are not controlled by a global distribution clock network but by the control signals. Therefore, no timing information are used to coordinate between all the participating components. Communication among the channels is governed by handshaking protocols for synchronization. In Sect. 2.2.2.1, different classes of asynchronous circuits are introduced. Asynchronous handshake protocols are described in Sect. 2.2.2.2. The communication channels are categorized in Sect. 2.2.2.3. To guarantee the timing robustness in asynchronous designs, data encoding is used and presented in Sect. 2.2.2.4. Finally, converters for asynchronous protocols are presented in Sect. 2.2.2.5.

2.2.2.1 Building Blocks of Asynchronous Design

This section provides an explanation of the important building blocks of asynchronous circuits such as Muller C-element, MUX, DMUX, and capture-pass Latch (CP Latch). In addition, different types of arbiters are demonstrated at the end of this section.

A. Muller C-element

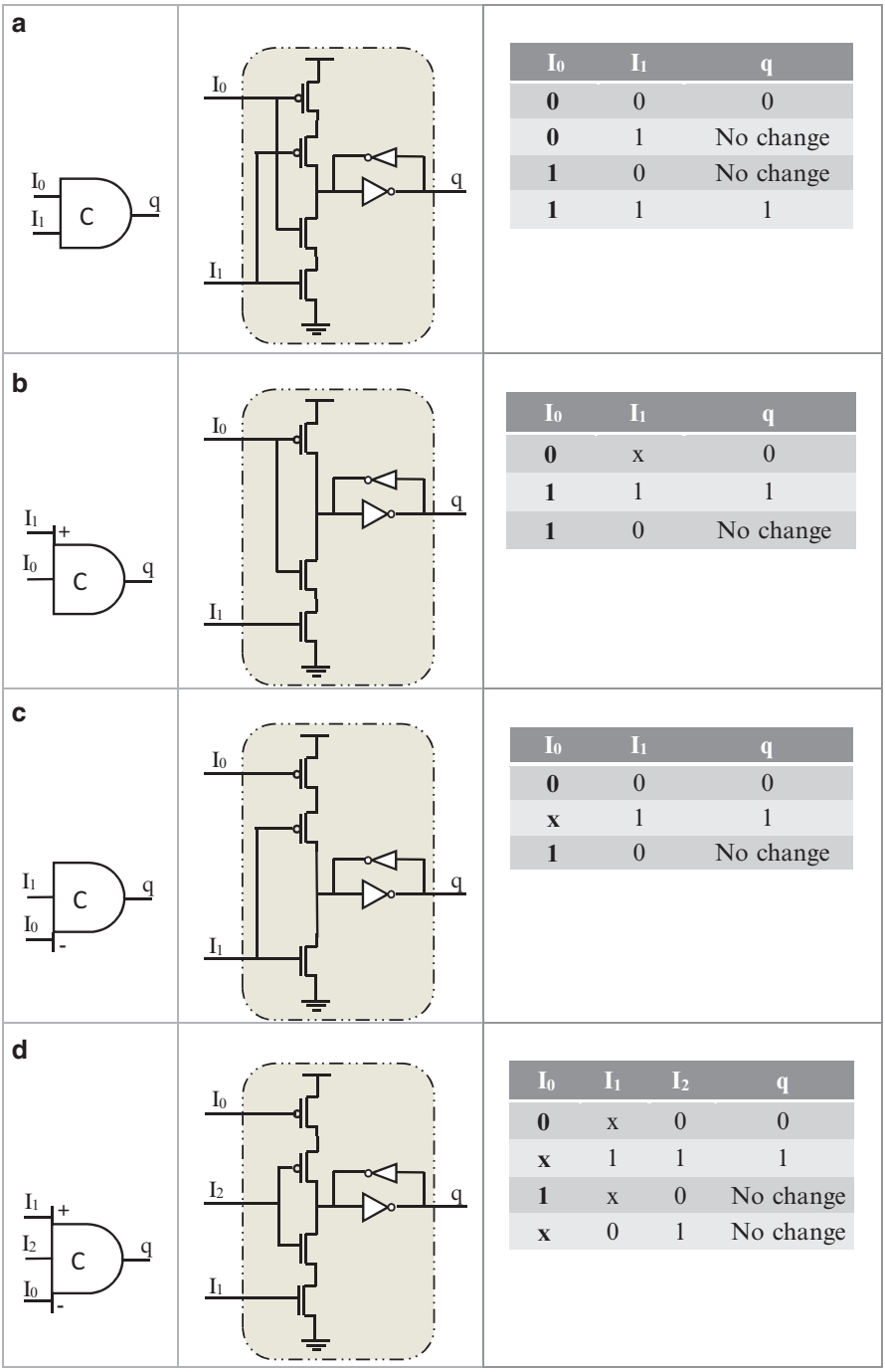
The Muller C-element is commonly encountered in asynchronous designs. It is used as a state holding element in addition to exploiting it for synchronizing multiple independent processes using synchronizing events. Muller C-element provides the AND function for events [13]. The symbol of symmetric 2-inputs C-element, circuit implementation, and truth table are shown in Fig. 2.10a. When the inputs levels are matched, the same logic level is assumed at the output. Otherwise, the latched output stays unchanged as listed in truth table at Fig. 2.10a. Other different designs for the asymmetric C-elements and its truth tables are shown in Fig. 2.10b, c. Asymmetric C-element has different sets of inputs based on the structure of N and P stacks which lead to rising or falling for the output transition. Each asymmetric C-element appended with either the minus (−) or plus (+) strips on the symbol. As shown in Fig. 2.10b, the C-element with plus strip only contributes for the pull down stack. Therefore, the output of C-element plus strip is driven to logic 0 if and only if the input i_0 is low as reported in truth table at Fig. 2.10b. C-element with minus strip only contributes for the pull up stack as shown in Fig. 2.10c. Then, the output of C-element minus strip is driven to logic 1 when the input i_1 is high as reported in truth table at Fig. 2.10c. The Muller C-element shown in Fig. 2.10d is a symmetric behavior with three inputs and it is described by truth table. The C-element is a fundamental part of the asynchronous building blocks. To build a complete asynchronous design, a storage element with two event signaling control wires is designed and presented in the next section. MUTEX is a basic arbitration element for asynchronous NoC designs as described in Sect. B. MUX, DEMUX, and arbiters are required MUTEX in their implementations to make an arbitrary decision as presented in the following sections.

B. Capture-Pass Latch

The CP Latch is an event-controlled storage element, and provide equivalent responses to the rising and falling events [13]. The symbol and the implementation of the event-controlled storage elements are shown in Fig. 2.11. The CP Latch uses two control signals (capture “C” and pass “P”) to drive a latch and determine the output mode as listed in Table 2.1. When inputs C and P are in the same state, the latch is in the Pass mode and the latch becomes transparent and the output follows any change in the input value. When both control signals are different, the latch turns to Capture mode and the latch becomes insensitive to changes in the input data. Consequently, the output value being held is stable.

C. MUTEX

The mutual exclusion circuit (MUTEX) has two inputs and two outputs as shown in Fig. 2.12. MUTEX makes a non-deterministic decision between two asynchronous requests. When one of two inputs is activated, the corresponding output should be activated. If both inputs are activated but one of them is activated before the other,



I_0	I_1	q
0	0	0
0	1	No change
1	0	No change
1	1	1

I_0	I_1	q
0	x	0
1	1	1
1	0	No change

I_0	I_1	q
0	0	0
x	1	1
1	0	No change

I_0	I_1	I_2	q
0	x	0	0
x	1	1	1
1	x	0	No change
x	0	1	No change

Fig. 2.10 Symbol, circuit implementation and truth table of Muller C-element. (**a**) Symmetric 2-inputs C-element. (**b**) Asymmetric C-element with plus strip. (**c**) Asymmetric C-element with minus strip. (**d**) Symmetric 3-inputs C-element

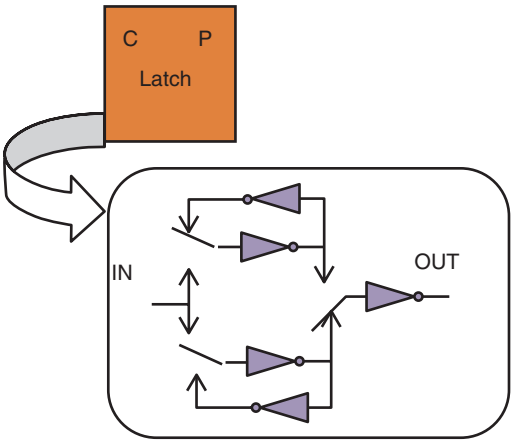


Fig. 2.11 Capture-pass latch implementation

Table 2.1 Truth table of CP latch

C	P	Mode
0	0	Pass
0	1	Capture
1	0	Capture
1	1	Pass

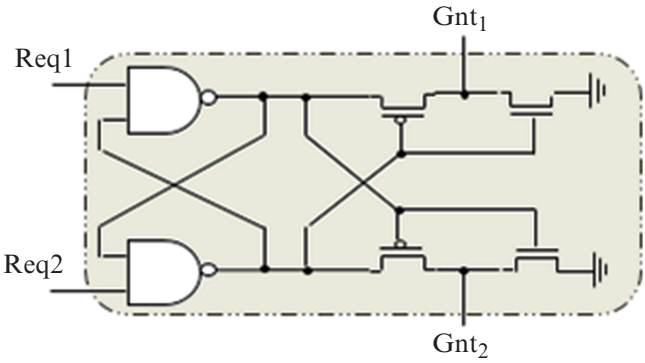


Fig. 2.12 Mutual exclusion component

the late request should be held back until the first request has been taken down and then allowed it to propagate. A metastability results when both inputs are activated at the same time or within a small time [14]. MUTEX consists of two cross-coupled NAND gates and analog filter [14]. Cross-coupled NAND structure is exploited to enable one input and block the other. The metastability can be produced from cross-coupled NAND structure when two requests arrives simultaneously. Hence, during the period where the decision is made, the state of the output may be neither a logic

0 nor a logic 1, but somewhere in between. This situation is known as metastability [15]. To resolve the metastability problem, analog filter is used to process the metastability and maintain valid logic levels. MUTEX is the intrinsic part in the handshake arbiters. The implementation of arbiters is demonstrated in the next section.

D. MUX and DEMUX

MUX and DEMUX are implemented to apply the bundled-data protocol on input and output data. MUX and DEMUX are designed as a function block to pass the input data using control signals as the selector. The control signals ($Cntrl_0$ and $Cntrl_1$) are used as two mutually exclusive requests that select between the two alternative input to transfer data to the output [13]. For MUX, C-elements produce an output request signal based on the control signals (at point A, B). The output request signals (at point A, B) are mutually exclusive and ORed together to produce the request signal at the output channel as shown in Fig. 2.13. The output acknowledge signals (ACK_{O0} and ACK_{O1}) response to the input acknowledge signals (ACK_I) and the output request signals (at point A, B). On the other hand, at DEMUX, an OR gate is used to produce the output acknowledge (ACK_O) from the input acknowledge signals (ACK_{I0} and ACK_{I1}) as shown in Fig. 2.14. The request signals on the two output ports (Req_{O0} and Req_{O1}) are mutually exclusive using the control signals.

E. Arbiter

Arbiter is considered an important component in NoC designs. Arbiters are classified into four types as described in the next sections.

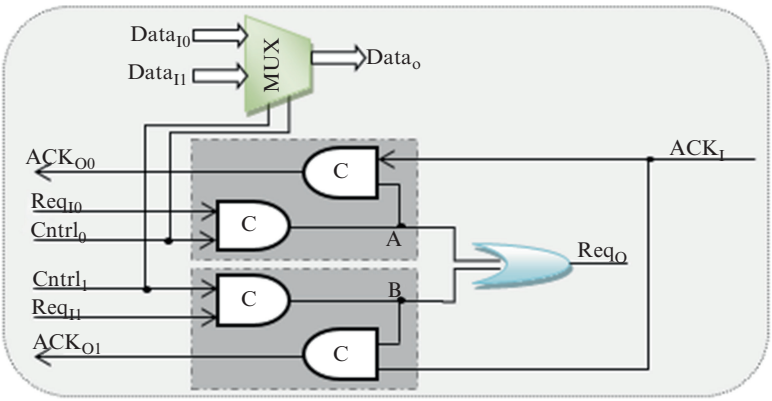


Fig. 2.13 Four-phase bundled data of MUX

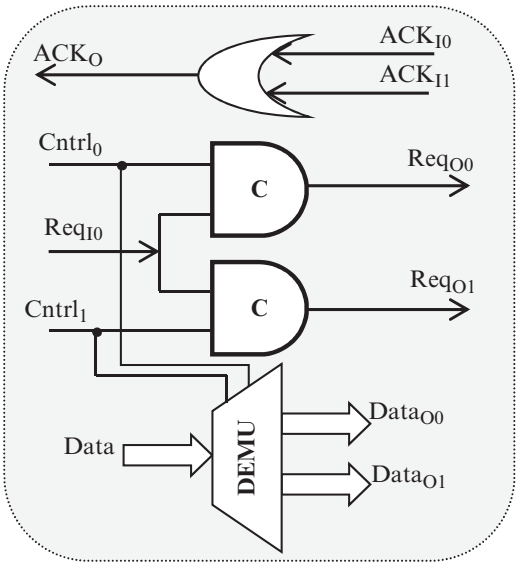


Fig. 2.14 Four-phase bundled data of DEMUX

(a) *Handshake arbiter*

The handshake arbiter is used to allow only one input port to access an output port. When the packet arrives to the output port successfully, ACK signal is generated to complete the handshaking sequence and permit the next packet to access input port. A handshake arbiter consists of MUTEX, cross-coupled NAND, two C-elements, and one OR gate as shown in Fig. 2.15 [13]. MUTEX is exploited to ensure that the grant signals (G_0 and G_1) are mutually exclusive even if both request signals (R_0 and R_1) are sent simultaneously. C-elements and cross-coupled NAND gates are used to ensure that the acknowledgment signals (ACK_0 and ACK_1) are mutually exclusive. The active Req signal is allowed through OR gate.

(b) *MUTEX cascade arbiter*

A larger MUTEX arbiter can be constructed from a cascaded two input MUTEXs. The multi-way MUTEX arbiter is used in asynchronous NoC designs [16, 17]. When the number of requests is small, the multi-way MUTEX arbiter is becoming effective, fast, and realizing low latency. However, the hardware requirements grow quadratically with the number of contending input requests. The *N*-way MUTEX arbiter is presented in [18]. As shown in Fig. 2.16a, the three-way MUTEX arbiter contains three input requests and three output grant signals. The arbiter grants one communication channel at the output to one of the three requesters and the others are blocked. The MUTEX arbiter is strictly fair and all requests have the same minimum arbitration delay since the number of MUTEXs on every request is equal. The implementation of four way MUTEX arbiter is shown in Fig. 2.16b.

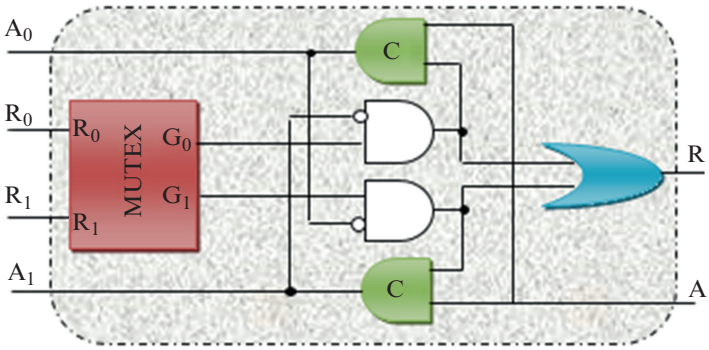


Fig. 2.15 Handshake arbiter

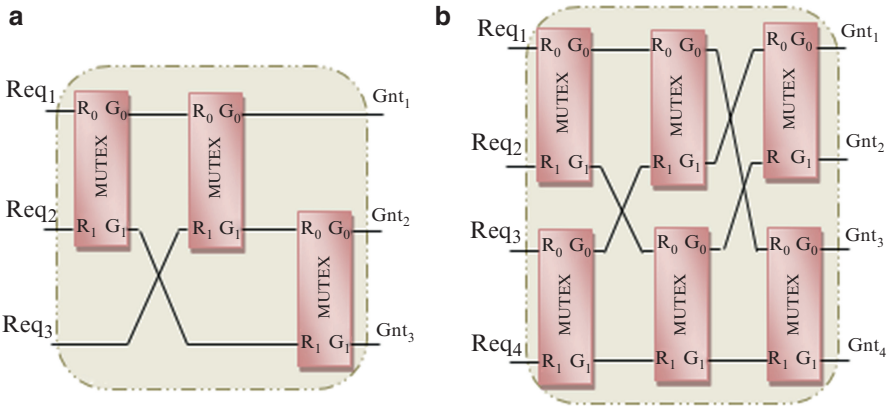


Fig. 2.16 Multi-way MUTEX cascade arbiter, (**a**) three-way MUTEX and (**b**) four-way MUTEX

(c) *Tree arbiter*

The tree arbiter is utilized extensively with a large number of requests [13]. Tree arbiter consists of MUX and multiple number of the handshake arbiter. As shown in Fig. 2.17, the tree arbiter has six input requests and six output grant signals. The number of input requests is divided into power-of-two sets of configurable input requests and similarly for the grant signal for each block [19]. Consequently, each set consists of two input requests and two grant signals. Each set is connected to handshake arbiter located at the first level. Every two blocks in a low level of the tree are connected to one block in the upper level. The root consists of one MUTEX as shown in Fig. 2.17. The tree is developed as a binary tree to optimize circuit implementation. If the binary tree is balanced, the tree arbiter is considered a strictly fair since the depth of all requests (from leaf to root) is $\lceil \log 2R \rceil$ where R is the number of requests. Otherwise, the leaf requests near to the root MUTEX gate are more likely to be granted than others in addition to having minimum arbitration delay since it is proportional to the depth of each request.

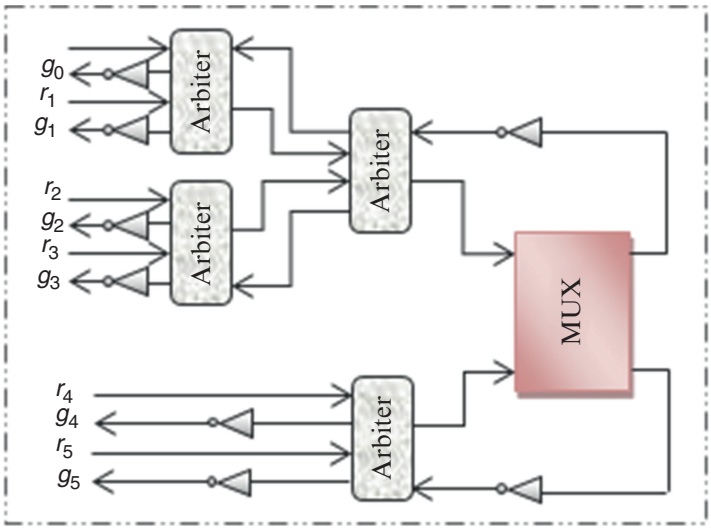


Fig. 2.17 Tree arbiter

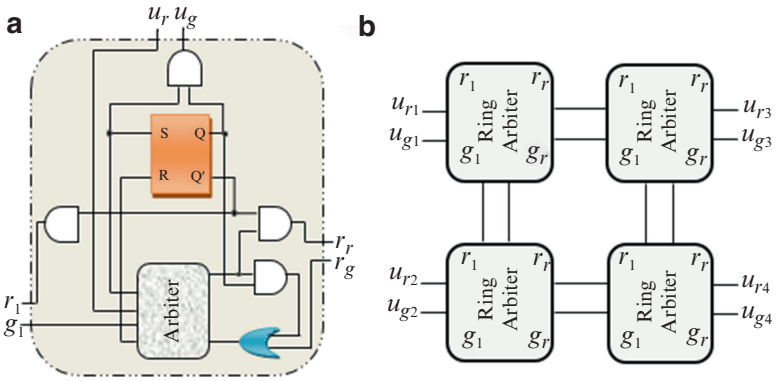


Fig. 2.18 Ring arbiter, (**a**) arbiter block and (**b**) arbiter diagram

(d)*Ring arbiter*

The ring arbiter implementation is presented in [20, 21]. The circuit for a ring arbiter element and a four way ring arbiter diagram are shown in Fig. 2.18a, b, respectively. Each block in four way ring arbiter diagram has two request inputs and one combined request output, which is the forward request to the neighbor block. Two input requests are one from the local user and the other one from its anticlockwise neighbor. Each block has a similar internal structure of the handshake arbiter in addition to SR latch to keep the token. All blocks of ring arbiter contain only one token. When an element's input request (Req) is activated, the state of its latch is checked. If the token is held, the grant signal (grnt) is asserted. If the request is activated in the block which does not have the token, the request

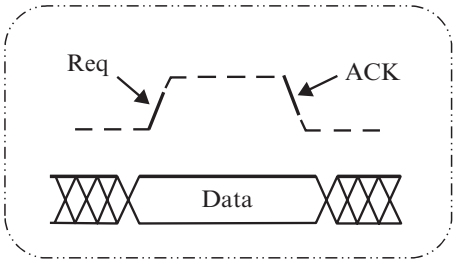


Fig. 2.19 The single-track handshaking protocol

is transferred to the neighbor block in clockwise direction. The minimum arbitration delay for a request depends on the location of the token. The arbiter is designed as a ring implementation, since the worst case arbitration latency grows linearly with the number of possible contending devices.

The main building blocks of asynchronous NoC design are described in the previous section. To synchronize the communication between all modules of asynchronous NoC design, the handshake protocols are used instead of the global clock distribution network as presented in the following section.

2.2.2.2 Handshake Protocols

Request (Req) and Acknowledge (ACK) signals are used as handshake signals. Handshake protocol signaling can be classified into four-phase (return-to-zero) and two-phase (non-return-to-zero). Additionally, other handshaking protocols are previously studied such as single track and one phase.

A. Single-Track Handshaking Protocol

As shown in Fig. 2.19, this protocol requires just one single handshake signal and two transitions to transfer data between the sender and the receiver. The transmitter or the receiver only alter the signal from "0" to "1" or "1" to "0," respectively. The single-track handshake protocol is considered a dynamic protocol [22]. Single-track protocol relies momentarily on high impedance states on interconnects. Hence, the asynchronous circuit runs correctly when it is not exposed to ambient noise [23, 24].

B. One-Phase Handshaking Protocol

One-phase handshaking protocol requires only one transition which makes it faster as compared to both two- and four-phase handshaking. One-phase handshaking protocol uses data coloring scheme to indicate the data validity and acceptance. The transmitted symbol consists of bit value and color information [25, 26]. The transmitter and receiver consist of color detector to extract the color information. If the transmit data is received correctly, the receiver changes the color. As a result of changing the color between transmitter and receiver, the data validity or ACK signal

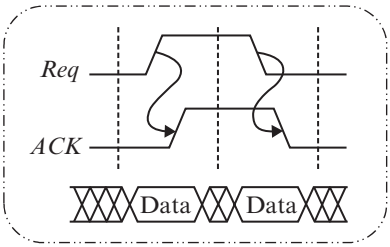


Fig. 2.20 Two-phase handshake protocol

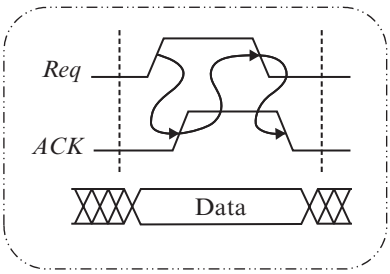


Fig. 2.21 Four-phase handshake protocol

is not required to be transmitted. The disadvantage of this protocol is that it requires complex circuits and consumes large power.

C. Two-Phase Handshaking Protocol

Two-phase handshaking protocol (non-return-to-zero) has only two transitions as shown in Fig. 2.20 [27]. It starts with a transition of the Req signal and finalized with one transition of the ACK signal as shown in Fig. 2.20. Two-phase signaling protocol is preferred for long on-chip communication since it has the advantage in terms of power consumption and performance. The disadvantages of two-phase signaling protocol is that it requires control logic circuit which leads to delay overhead.

D. Four-Phase Handshaking Protocol

The four-phase handshaking (return-to-zero) requires four transitions as shown in Fig. 2.21. ACK signal is asserted when the incoming data are captured. Before starting new handshaking, it is necessary that the signals (Req and ACK) return to zero level, respectively. As a consequence, four-phase handshaking protocol has four transitions as shown in Fig. 2.21. The disadvantage of the four-phase handshaking protocol is that the delay increases with long interconnect which leads to significant reduction in throughput.

2.2.2.3 Communication Channel

Req and ACK signals are used to indicate the data validity and data acceptance, respectively. The handshaking signals are transferred between the sender and receiver onto the communication channel. The communication channels are classified into push

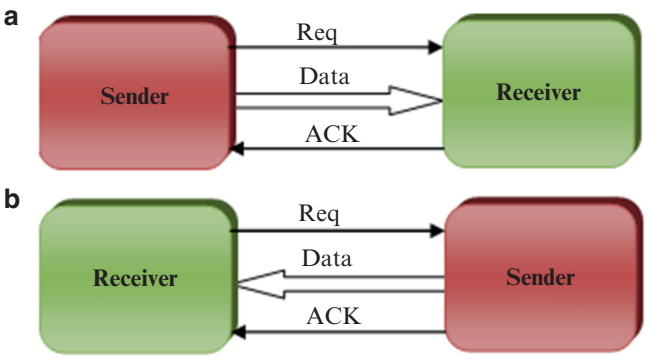


Fig. 2.22 Communication channels, (**a**) push channel and (**b**) pull channel

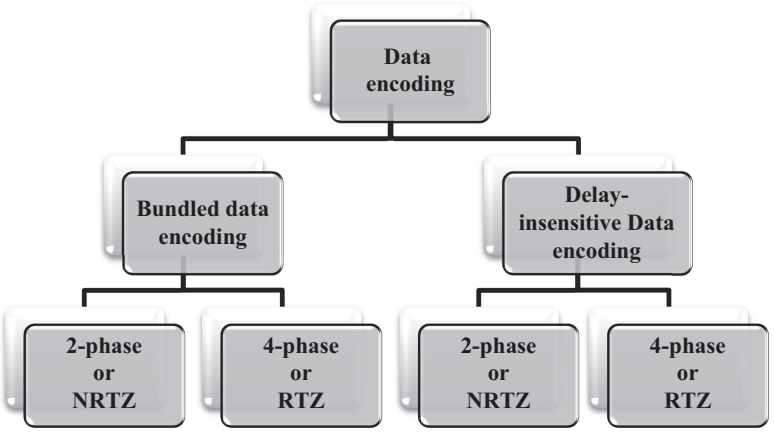


Fig. 2.23 The categories of data encoding

channel and pull channel. A push channel data flow with the Req direction as shown in Fig. 2.22a. Whereas, the pull channel data is transmitted in the same direction as ACK signal as shown in Fig. 2.22b. Req and ACK signals are transmitted in signaling interconnects using data encoding technique as described in the next section.

2.2.2.4 Data Encoding

Data encoding is used to guarantee the timing robustness in data transmission. Bundled-data encoding and delay-insensitive encoding are widely used types of data encoding. The classification of data encoding is shown in Fig. 2.23. The bundled-data encoding and delay-insensitive encoding are summarized in Sects. A and B, respectively.

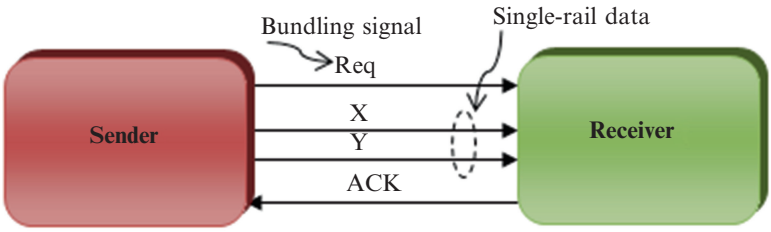


Fig. 2.24 Bundled-data encoding

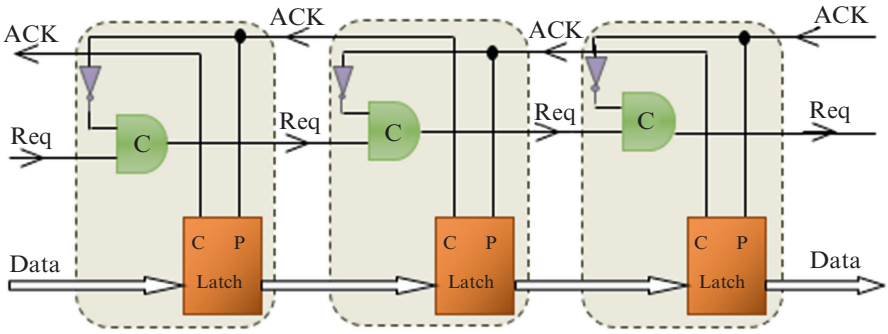


Fig. 2.25 A 2-phase bundled-data pipeline

A. Bundled-Data Encoding

To guarantee that all data bits are stable and valid at the receiver, a bundling signal (Req) is used as an extra interconnect. Req and ACK signals can be transmitted separately [28] as shown in Fig. 2.24. Bundled-data encoding can be divided into two-phase and four-phase as described in the following sections.

(a) *Two-phase bundled-data encoding*

In the two-phase bundled-data protocol, Req or ACK event is encoded as a single transition. Consequently, the transition from 0 to 1 or from 1 to 0 represents a single event. Two-phase protocols of single-rail bundled data are also commonly used. The circuit of two-phase bundled-data encoding consists of Muller C-elements and CP latches as shown in Fig. 2.25 [13, 29]. The transition of the control signals (ACK and Req) are created by CP latch. CP latch is alternated between capture mode and pass mode since the transition of the control signals are altered, respectively, on the C and P inputs. When the "Capture" input makes a transition, the register capture the data and when the "Pass" input makes a transition, the register passes the data through the register.

(b) *Four-phase bundled-data encoding*

Four-phase bundled-data protocol needs four communication actions to initiate the next communication cycle. Data typically must remain stable from

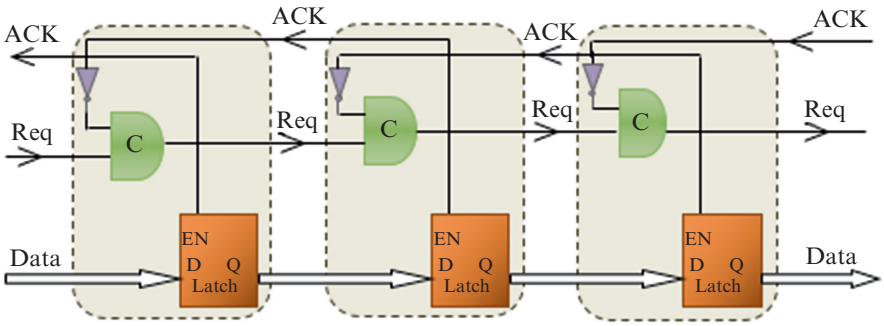


Fig. 2.26 A 4-phase bundled-data pipeline

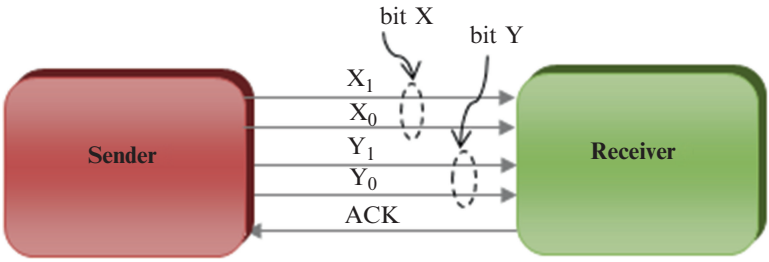


Fig. 2.27 Dual-rail data encoding

Table 2.2 Dual-rail encoding

Bit	Dual-rail encoding	
X	X1	X0
0	0	1
1	1	0
No data	0	0 (NULL)

a setup time (before the next Req is asserted high) until a hold time (after ACK is de-asserted low) to avoid glitches arbitrarily between transactions [30]. The four-phase bundled-data structure is composed of two types of handshake components, Muller C-elements and latches, as shown in Fig. 2.26.

B. Delay-Insensitive Encoding

Dual-rail (1-of-2) and quad-rail (1-of-4) encodings are most common types of delay-insensitive encoding used in on-chip implementation. In delay-insensitive encoding, the Req signal is encoded in the data lines by using two interconnects per one data bit as shown in Fig. 2.27.

Dual-rail data encoding consists of two interconnects per data bit from sender to receiver and one interconnect for ACK signal from receiver to sender [14]. As reported in Table 2.2, every bit in the message carries the data validity to enable the

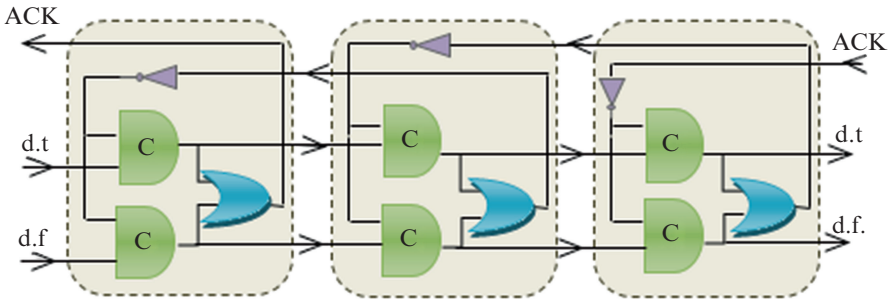


Fig. 2.28 A 4-phase dual-rail pipeline for 1-bit

receiver to receive data correctly [31]. By detecting a level for 4-phase protocol, or an edge for 2-phase protocol (on one of the two rails), the dual-rail encoding is determined when a new data is valid. Consequently, it is not necessary to use separate handshake signal. The receiver typically uses a completion detector (CD) to identify that a valid code word has been received.

As shown in Fig. 2.28, the circuit implementation of four-phase dual-rail data encoding (for one-bit) is composed of two parallel Muller C-elements, inverters and OR gates [14]. Muller C-elements are used to achieve the synchronization by using a common ACK signal. The ACK signal which is generated by the OR gate safely indicates the state of the pipeline stage as being "valid" or "empty."

For dual-rail data encoding (N bits), $2N$ interconnects per data bit from sender to receiver and one interconnect for ACK signal from receiver to sender are used as shown in Fig. 2.29. Only one transition is used to transmit valid data in parallel by individual ACK signal to synchronize between function blocks. OR gates and C-element are used to generate the output ACK signal. As shown in Fig. 2.29, the data symbol is captured by C-elements at the input d_i and is delivered to the next pipeline stage immediately through d_o. All C-elements are triggered by ack_o signals which are exploited as latches. The C-element tree generates a common acknowledge signal ACK_{in} for synchronizing data as shown in Fig. 2.29. All data bits should be captured to send ACK_{in} signal. On the other hand, the ACK_{in} signal is reset when all data bits are released.

2.2.2.5 Converters for Asynchronous Protocols

To avoid the delay between the control and data interconnects in asynchronous NoC designs, the delay-insensitive data encoding is required for long interconnects. Consequently, robust global timing is obtainable in NoC designs. The conversion between the single-rail data encoding and delay-insensitive encoding is necessary for global communication. Bundled-data encoding interface is used in some of asynchronous communication wrappers [32–34]. Bundled-data encoding is

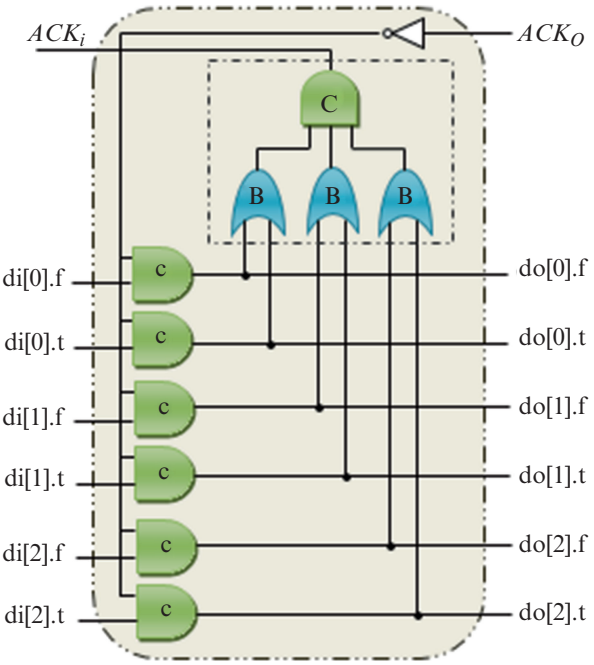


Fig. 2.29 A 4-phase dual-rail pipeline for *N*-bit

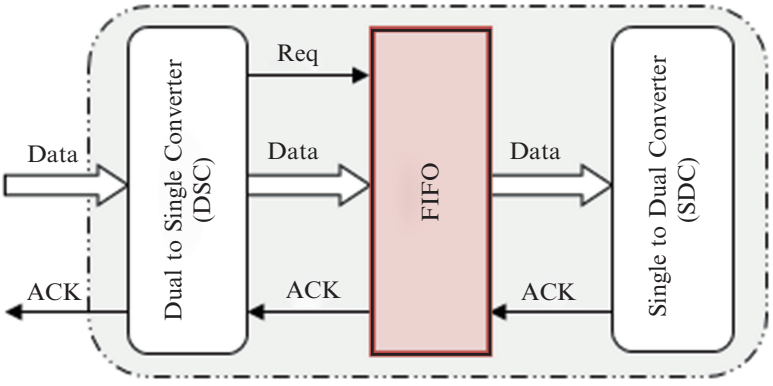


Fig. 2.30 Protocols converters

considered as the intrinsic solution for both the complexity and area overhead of FIFOs which are inserted at the input/output ports of NoC router. It is necessary to insert the Dual-rail (delay-insensitive) to Single-rail (bundled encoding) Converter (DSC) before the FIFO and Single-rail to Dual-rail Converter (SDC) after FIFO as shown in Fig. 2.30.

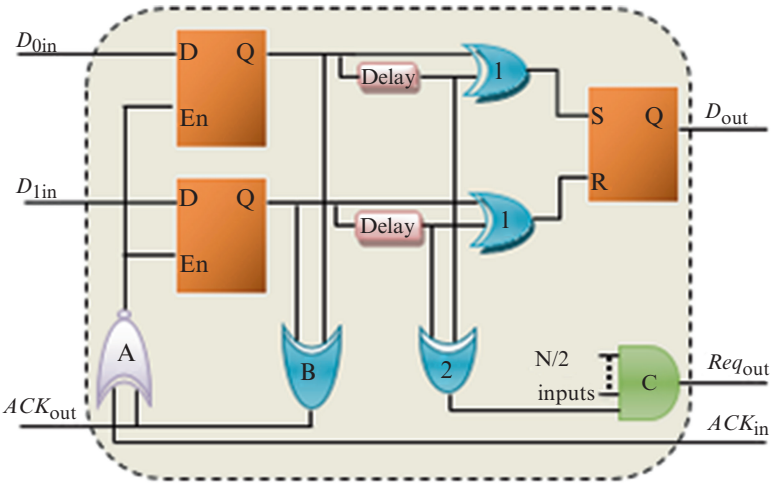


Fig. 2.31 DSC architecture

A. Delay-Insensitive to Bundled Encoding

DSC architecture consists of D Flip-Flops, SR latch, C-element, XOR, and XNOR gates as shown in Fig. 2.31. XOR_1 gates detect the transitions on the interconnects. SR latch is exploited to decode the data into the single-rail form. XOR_2 gate together with an *N*/2-input C-element are used to detect the Req signal at the output [31]. D Flip-Flops, $XNOR_A$, and XOR_B are used to get input ACK signal and regenerate the output ACK signal.

B. Bundled Encoding to Delay-Insensitive Encoding

SDC architecture consists of Double-Edge Triggered D Flip-Flops (DETDFF), multiplexers, D Flip-Flops, XOR, and XNOR as shown in Fig. 2.32. DETDFF is sensitive to the edge of Req signal which captures the value of the input data after either rising or falling edge of Req transitions [31]. Multiplexers are used to allow transition of the corresponding output of flip-flop, if and only if the appropriate input data is presented. Consequently, both DETDFF and multiplexers are used to decode the input data into the dual-rail form. D Flip-Flops, $XNOR_A$, and XOR_B are exploited to get input ACK signal and regenerate the output ACK signal. On the other hand, the routing algorithm is one of the key factor in NoC architecture as presented in the following section.

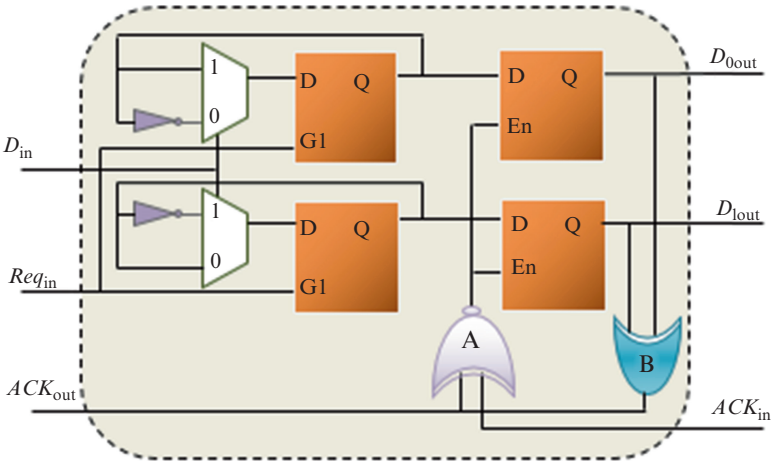


Fig. 2.32 SDC architecture

2.3 Routing Algorithms in NoC

Routing algorithm is responsible for the selection of the appropriate path to transfer the messages between any two communicating nodes (source-destination pair). Selection of the routing algorithm is crucial to achieve good network performance. Achieving minimum communication average message delay, high throughput, avoiding deadlocks, and livelocks must be considered to design an effective routing algorithm. Routing algorithms can be classified into different categories depending on how the routing algorithm chooses between the available set of paths from source to destination. All the previous issues are presented in this section.

2.3.1 Classification of Routing Algorithms

Source vs. distributed routing, deterministic vs. adaptive, minimal vs. non-minimal, and congestion-aware vs. congestion-oblivious are the main categories of routing algorithms. All categories are presented in the following sections.

2.3.1.1 Source vs. Distributed Routing Algorithms

In source routing algorithms, the sequence of hops for the whole path between the source and destination are encoded in the header of the packet at source node. At each router during the trip between source and destination nodes, the header of the packet is decoded to navigate the packet towards the destination. The primary

disadvantages of source routing for NoC are the limited path adaptivity and large overhead to store the path information in the header of the packet. On the other hand, at distributed routing algorithm, the destination address is only inserted in the header of the packet [35, 36]. Every router gets the routing decision independently based on the knowledge of the destination address. The drawback of the distributed routing algorithm is the complexity of the router design to handle a distributed scheme.

2.3.1.2 Deterministic vs. Adaptive Routing Algorithms

Deterministic routing algorithms route the packets in a certain path for communication between source-destination pair. Dimension-order XY routing algorithm (DOR-XY) is typical example of deterministic routing algorithms. The disadvantage of deterministic routing algorithm is that the path from the source to destination is determined irrespective to the congestion in the network. Consequently, under realistic traffic loads, the load balances in the network are not maintained [37]. On the other hand, multiple output channels are available for the packet to traverse between the source and destination in adaptive routing algorithms. Each router should choose one of the admissible output channels depending on the network status (such as congestion or faults in the network). Thereby, the probability of routing packets through congested regions is decreased. Adaptive routing algorithms is classified into partially adaptive and fully adaptive. Partially adaptive routing algorithms allow routing packets to route through the shortest paths only. On the contrary, in fully adaptive routing algorithm, every packet can be routed along any minimal path available between the source and destination pair [38, 39]. Generally, the adaptive routing algorithm can decrease the latency and increase the throughput.

Routing algorithm is divided into two main blocks, the routing function [40, 41] and the selection function [42]. In deterministic routing algorithm, the routing function returns only a single output port. Hence, the selection function block does not exist as shown in Fig. 2.33a. At adaptive routing algorithm, the routing function generates a set of admissible output channels to forward the packet toward the destination node. The selection function block is used to select one output channel from the set of admissible output channels returned by the routing function based on the network status information as shown in Fig. 2.33b.

2.3.1.3 Minimal vs. Non-minimal Routing

Minimal routing algorithms guarantee finding the shortest path between sender and receiver. The deadlock can be avoided by using the minimal paths. Minimal routing algorithms may cause an excessive average message delay, since it is not always able to avoid traffic congestion. Non-minimal routing algorithms allow messages to route through longer paths for congestion avoidance or fault tolerance purposes as shown in Fig. 2.34.

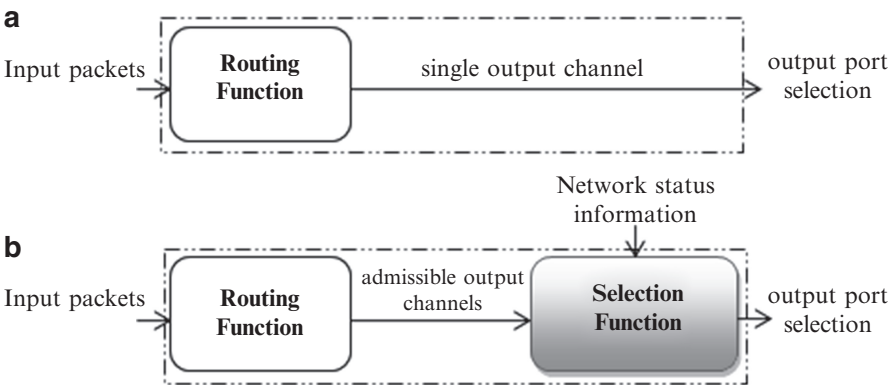


Fig. 2.33 Routing and selection functions for (**a**) deterministic routing algorithms and (**b**) adaptive routing algorithms

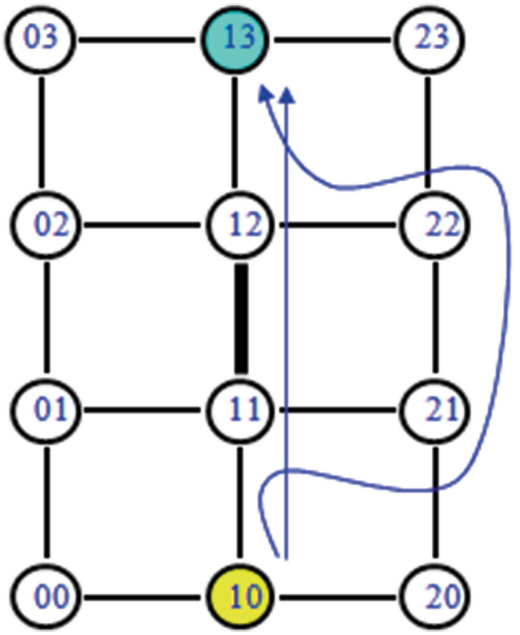


Fig. 2.34 Minimal and non-minimal routing path

2.3.1.4 Congestion-Oblivious vs. Congestion-Awareness Routing

In oblivious routing algorithms, multiple paths from the source to the destination are available and the selection of the routing path does not depend on the current status of the network. Hence, the congestion-oblivious routing algorithms (such as XY) neglect the congestion conditions on its routing decisions which lead to

disruption in the load balance. On the other hand, the selection schemes in the congestion-awareness routing algorithms (such as MAXY [43]) are based on a locally available congestion estimates. The number of free virtual channels at an output port is proposed as a contention metric for congestion-aware routing algorithms [44]. On the other studies [45, 46], congestion is defined as a free buffer slots in the input ports of the adjacent neighbors. Congestion-awareness routing algorithms outperform congestion-oblivious ones since congestion-awareness algorithms aim to select the less congested paths to produce load balance in the network, especially under realistic traffic loads.

2.3.2 Routing Issues

Deadlock, livelock, and starvation must be avoided to ensure correct functionality of the NoC routing algorithms. These problems are discussed in the following sections.

2.3.2.1 Deadlock

The freedom from deadlock is the main issue for any routing algorithm. Deadlocks occur as a result of circular dependencies on network resources by messages or packets [47]. Deadlocks have two categories, routing-dependent deadlocks and message-dependent deadlocks [47–49].

Routing-dependent deadlocks occur when there is a cyclic dependency of resources created by the packets on the various paths in the network [50, 51]. Deadlock happens when one or more packets in the network become blocked and stay blocked for an indefinite time, waiting for an event that cannot happen. For example, four packets P_1; P_2; P_3; and P_4 wait for buffer space occupied by each other in a loop and all buffers are full, therefore no packet can advance as shown in Fig. 2.35 [52].

Message-dependent deadlocks occur when there is a dependency between the request and reply packets on the target's input and output channels [53, 54]. As shown in Fig. 2.36, when node *B* wants to send out new pending request *reqB* to node *A* it must first receive reply *repA* from node *A* about some previous requests. *repA* helps node *B* to free its internal buffers and hence node *B* is able to prepare new request. However, node *A* cannot send out reply *repA* to node *B* because the network buffers are full that requires node *B* to consume some messages to free the network buffer. However, node *B* also cannot consume any messages since its internal memory is full and it cannot send out a message to node *A* because the buffers in the network are full. Both node *A* and *B* wait for each other to consume packets but none of them can then the deadlock happens [52].

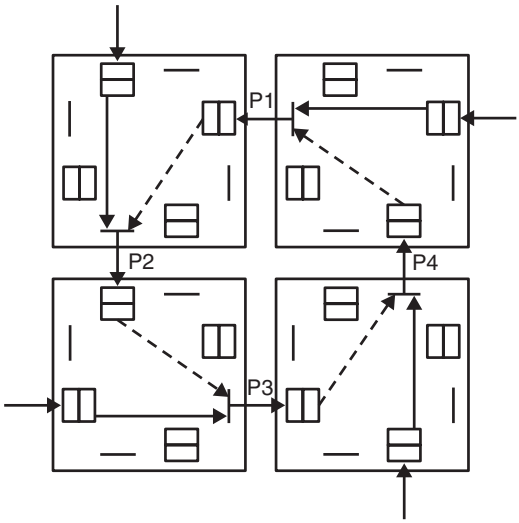


Fig. 2.35 Routing-dependent deadlock [52]

2.3.2.2 Livelock

Livelock is a situation in the network where when some packets are not able to reach their destinations. Livelock may arise only when non-minimal routing is employed. The routing algorithm selects the longest path whether the shortest path exists or not by observing the network congestion. However, it can be avoided by limiting the number of misrouting allowed to each packet traversing the network [39].

2.3.3 Routing Algorithm Schemes

Many deadlock-free routing algorithms were proposed such as DOR-XY, Randomized, Oblivious, Multi-phase, Minimal (ROMM) [55], Turn models, Odd-Even (OE) [56], DyAD [57], and MAXY. The different routing algorithms are summarized in the following sections.

2.3.3.1 ROMM Routing Algorithm Scheme

ROMM routing algorithm reduces the latency by confining the intermediate node within the range of a minimal path between source and destination pairs [55]. In 2-phase ROMM, the XY routing algorithm is used to route the packet from the source to a random node in the first phase before routing it to the destination in the second phase. Generally, in p-phase ROMM algorithm, $p-1$ random nodes $(Z_0, Z_1, \ldots Z_{p-1})$ are chosen on the minimal path between source and destination

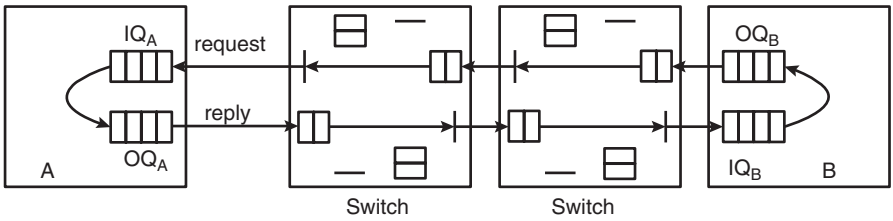


Fig. 2.36 Message-dependent deadlocks [52]

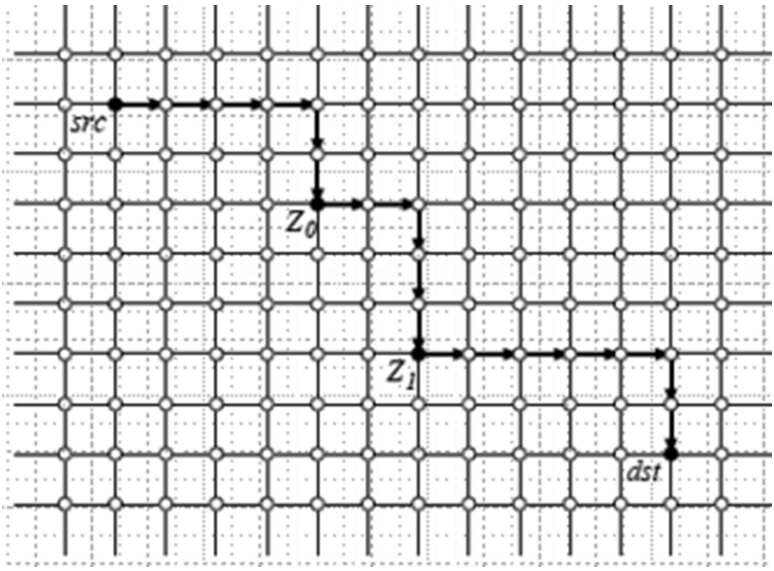


Fig. 2.37 Possible path from source to destination using 3-phase ROMM [55]

nodes as shown in Fig. 2.37. To rout the packets, ROMM routing algorithm achieves the minimum number of network hops, increases the routing flexibility, and avoids the congestion by using randomization. ROMM guarantees the deadlock free by using virtual channels.

2.3.3.2 Turn Model Schemes

Turn Model routing scheme provides deadlock freedom in the two-dimensional mesh topology [56]. The deadlock can be avoided by prohibiting just enough turns to break all the cycles. Generally, eight turns are the intrinsic reason to the occurrence of deadlock as shown in Fig. 2.38.

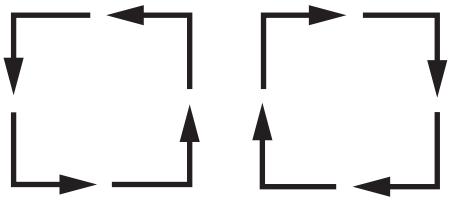


Fig. 2.38 The all turns to occur the deadlock in two-dimensional mesh [56]

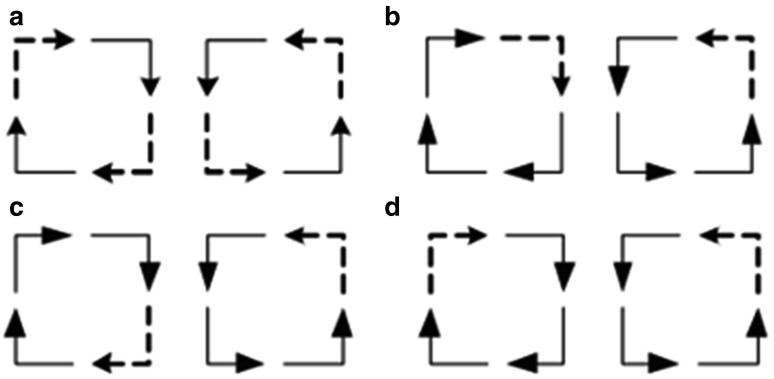


Fig. 2.39 All possible turn models: (**a**) XY routing, (**b**) Negative-First, (**c**) West-First, and (**d**) North-Last [56]

XY, Negative-First, West-First, and North-Last are the well-known turn models [58] as shown in Fig. 2.39a–d, respectively. In XY routing algorithm, a packet routes along the *X*-axis then *Y*-axis toward the destination. XY routing algorithm is free from deadlock since it prohibits four of the turns as shown in Fig. 2.39a. Therefore, the remaining four turns cannot cause the deadlock in XY routing algorithm. Negative-First routing algorithm prohibits two 90° turns in two-dimensional mesh and break every cycle. The allowed turns are the turn from West to East and the turn from South to North as shown in Fig. 2.39b. West-First routing algorithm prohibits two 90° turns in a two-dimensional mesh and breaks every cycle as shown in Fig. 2.39c. The turn from West to East and either the turn from South to North or the turn from North to South are the allowed turns in West-First routing algorithm. On the other hand, two 90° turns are prohibited in North-Last routing algorithms. The turn from South to North and either the turn from West to East or the turn from East to West are the incorporated turns in this routing algorithm as shown in Fig. 2.39d.

2.3.3.3 OE Routing Algorithm Scheme

OE turn model is one of the most popular partially adaptive routing algorithms for NoC [56]. The restrictions of OE turn model prevent the routing decision from deadlock and livelock without using any virtual channels by prohibiting certain turns in odd and even columns. OE turn model prohibits turning from East to North

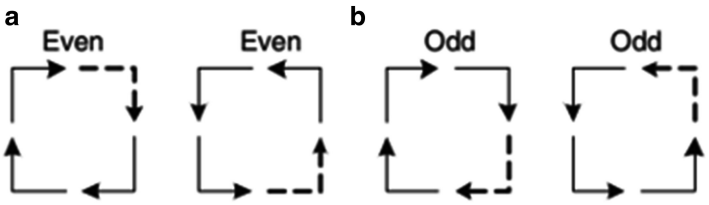


Fig. 2.40 The rules of OE model [56]. (**a**) Prohibited turns in even columns and (**b**) prohibited turns in odd columns

and from North to West (from East to South and from South to West) at routers located in an even column (odd column) as shown in Fig. 2.40. Therefore, the degree of adaptiveness provided by OE turn model is higher than the other turn models.

2.3.3.4 DyAD Routing Algorithm Scheme

DyAD, dubbed from Dynamic Adaptive Deterministic switching, is a partial adaptive routing scheme [57]. DyAD is combination of a deterministic and adaptive routing algorithms. It switches between XY and OE routing algorithms based on the congestion status of the network. In DyAD routing algorithm, each router in NoC always monitors the network condition. Congestion values are calculated for all neighboring routers and decisions are made according to this real time information. When the network is not congested, deterministic routing is used. Once a congestion occurs in the network, it dynamically switches to an adaptive routing scheme. DyAD routing algorithm guarantees the freedom from deadlock. It contains the deterministic and adaptive routing modes into the same NoC routing algorithm.

2.3.3.5 MAXY Routing Algorithm Scheme

Minimally Adaptive XY (MAXY) routing algorithm is an adaptive routing for NoC. MAXY routing algorithm targets to equalize the absolute difference between X and Y coordinates of the current nodes [43]. The congestion status for both feasible directions is taken into consideration when the absolute difference is equal for both directions. Congestion is defined as a free number of output buffer channels. When the congestion levels are equal, a random selection is chosen to route the packet. This routing algorithm avoids the deadlock scenario by using virtual channels.

2.4 Conclusions

Data transfer in NoC could be done synchronously or asynchronously. In order to synchronize the nodes in the NoC, clock distribution network is required for synchronous system. Buffered clock distribution trees and symmetric trees are different

forms of clock distribution networks. On the other hand, asynchronous NoC circuits are based on the use of handshaking protocols to organize the data transfer. The handshake protocols at NoC designs are the bundled-data encoding for single-rail protocol and the delay-insensitive encoding for dual-rail protocol. The conversion of protocols is exploited to reduce the delay between the control and data lines which exists in bundled-data encoding.

Each router basically consists of the routing algorithm to determine the most appropriate path. Deterministic routing algorithms, on the contrary to the most adaptive routing algorithms, define the path from the source to destination irrespective to the congestion in the network. Adaptive algorithms aim to select the less congested paths to produce load balance in the network, especially under realistic traffic loads. Adaptive routing algorithm should be manipulated with the process variation. A novel adaptive routing algorithm based on delay with process variation and congestion for asynchronous NoC design is presented in Chap. 6.

References

1. Öberg J (2003) Clocking strategies for networks-on-chip. Kluwer Academic Publishers, New York, pp 153–172
2. Friedman EG (2001) Clock distribution networks in synchronous digital integrated circuits. IEEE J 89(5):665–692
3. Mu F, Svensson C (1999) self-tested self-synchronization for parallel systems. IEEE Trans Parallel Distrib Syst J 10(8):769–780
4. Mu F, Svensson C (2001) Self-tested self-synchronization circuit for mesochronous clocking. IEEE Trans Circuits Syst II 48(2):129–140
5. Söderquist I (2003) Globally updated mesochronous design style. IEEE J Solid-State Circuits 38(7):1242–1249
6. Nilsson E, Oeberg J (2004) Reducing power and latency in 2-D mesh NoCs using globally pseudochronous locally synchronous clocking. In: Proceeding of the international hardware/software codesign and system synthesis, Sept 2004, pp 176–181
7. Dally W, Dennison LR, Harris D et al (1994) Architecture and implementation of the reliable router. In: Proceeding of the symposium record hot interconnects II, pp 197–208
8. Dennison LR, Dally W, Xanthopoulos T (1995) Low-latency plesiochronous data retiming. In: Proceedings of the conference on advanced research in VLSI, Mar 1995, pp 304–315
9. Sheibanyrad A, Greiner A (2007) Hybrid-timing FIFOs to use on networks-on-chip in GALS architectures. In: Proceedings of the ESA, pp 27–33
10. Friedman EG, Powell S (1986) Design and analysis of a hierarchical clock distribution system for synchronous standard cell/macrocell VLSI. IEEE J Solid-State Circuits 21(2):240–246
11. Dike C, Burton E (1999) Miller and noise effects in a synchronizing flip-flop. IEEE J Solid-State Circuits 34(6):849–855
12. Jex JG, Dike C (1995) A fast resolving binmos synchronizer for parallel processor interconnect. IEEE J Solid-state Circuits 30(2):133–139
13. Sparsø J, Furber S (2002) Principles of asynchronous circuit design: a systems perspective. Springer, Heidelberg
14. Mead CA, Conway LA (1980) Introduction to VLSI systems. Addison-Wesley, London
15. Chaney TJ, Molnar CE (1973) Anomolous behavior of synchronizer and arbiter circuits. IEEE Trans Comput J 22(4):421–422

16. Dobkin R, Ginosar R, Kolodny A (2009) QNoC asynchronous router. Integr VLSI J 42(2): 103–115
17. Song W, Edwards D (2010) A low latency wormhole router for asynchronous on-chip networks. In: Proceedings of the Asia and South Pacific design automation conference, pp 437–443
18. Kinniment DJ (2007) Synchronization and arbitration in digital systems. Wiley, New York
19. Josephs MB, Yantchev J (1996) CMOS design of the tree arbiter element. IEEE Trans VLSI Syst J 4(4):472–476
20. Martin AJ (1991) Synthesis of asynchronous VLSI circuits. Department of Computer Science, California Institute of Technology, Pasadena
21. Martin AJ (1985) The design of a self-timed circuit for distributed mutual exclusion. In: Proceedings of the Chapel Hill conference on VLSI, pp 245–260
22. Berkel K, Bink A (1996) Single-track signaling with application to micropipelines and handshake circuits. In: Proceedings of the second international symposium on advanced research in asynchronous circuits and systems, Mar 1996, pp 122–133
23. Upadhyay A, Hasan SR, Nekili M (2004) A novel asynchronous wrapper using 1-of-4 data encoding and single-track handshaking. In: Proceedings of the IEEE Northeast workshop on circuits and systems, June 2004, pp 205–208
24. Ferretti M, Beerel PA (2002) Single-track asynchronous pipeline tem-plates using 1-of-N encoding. In: Proceedings of the design, automation and test in Europe conference and exhibition, Aug 2002, pp 1008–1015
25. Hanyu T, Takahashi T, Kameyama M (2003) Bidirectional data transfer based asynchronous VLSI system using multiple-valued current mode logic. In: Proceeding of the international symposium on multiple-valued logic, May 2003, pp 99–104
26. Nigussie E, Plosila J, Isoaho J (2005) On asynchronous full-duplex dual-rail link with multiple-valued current-mode signaling. In: Proceeding of NORCHIP conference, Nov 2005, pp 222–225
27. Reese RB, Thornton M, Traver C (2004) Two-phase micropipeline control wrapper with early evaluation. IEEE Electron Lett 40(6):365–366
28. Peeters AMG (1996) Single-rail handshake circuits. PhD thesis, Eindhoven University of Technology
29. Sutherland IE (1989) Micropipelines. Commun ACM J 32(6):720–738
30. Nowick SM, Singh M (2011) High-performance asynchronous pipelines: an overview. IEEE Des Test Comput J 28(5):8–22
31. Nigussie E, Lehtonen T, Tuuna S et al (2007) High-performance long NoC link using delay-insensitive current-mode signaling. In: Proceedings of the Hindawi Publishing Corporation, VLSI design, Mar 2007
32. Dobkin R, Ginosar R, Sotiriou CP (2006) High rate data synchronization in GALS SoCs. J IEEE Trans VLSI Syst 14(10):1063–1074
33. Ning W, Fen G, Fei W (2009) Design of a GALS wrapper for network on chip. Proc World Congr Comput Sci Inf Eng 3:592–595
34. Liljeberg P, Plosila J, Isoaho J (2004) Self-timed communication platform for implementing high-performance systems-on-chip. Integr VLSI J 38(1):43–67
35. Flich J, lopez P, Malumbers MP, Duato J (2000) Improving the performance of regular networks with source routing. In: Proceeding of the IEEE international conference on parallel processing, Aug 2000, pp 353–361
36. Aydogan Y, Stunkel CB, Aykanat C, Abali B (1996) Adaptive source routing in multistage interconnection networks. In: Proceedings of the international parallel processing symposium, Apr 1996, pp 258–267
37. Bertsekas DP, Gallager RG (1992) Data networks. Prentice Hall, Englewood Cliffs
38. Ni LM, McKinley PK (1993) A survey of wormhole routing techniques in direct networks. J Comput 26(2):62–76
39. Duato J, Yalamanchili S, Lionel N (2003) Interconnection networks—an engineering approach. Morgan Kaufmann, San Francisco

40. Chien AA, Kim JH (1995) Planar-adaptive routing: low-cost adaptive networks for multiprocessors. J ACM 42(1):91–123
41. Upadhyay J, Varavithya V, Mohapatra P (1997) A traffic-balanced adaptive wormhole routing scheme for two-dimensional meshes. J IEEE Trans Comput 46(2):190–197
42. Ye TT, Benini L, Micheli GD (2004) Packetization and routing analysis of on-chip multiprocessor networks. J Syst Arch 50(2–3):81–104
43. Rameshan N, Ahmed M, Gaur MS et al (2009) QoS aware minimally adaptive XY routing for NoC. In: Proceedings of the international conference on advanced computing and communications
44. Ma S, Jerger NE, Wang Z (2011) DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip. In: Proceedings of the international symposium on computer architecture, June 2011, pp 413–424
45. Lotfi-Kamran P, Rahmani AM, Daneshtalab M et al (2010) EDXY—a low cost congestion-aware routing algorithm for network-on-chips. J Syst Arch 56(7):256–264
46. Lotfi-Kamran P, Daneshtalab M, Lucas C et al (2008) BARP-A Dynamic routing protocol for balanced distribution of traffic in NoCs. In: Proceedings of the design, automation and test in Europe, pp 1408–1413
47. Song YH, Pinkston TM (2003) A progressive approach to handling message-dependent deadlock in parallel computer systems. IEEE Trans Parallel Distrib Syst J 14(1):259–275
48. Starobinksi D, Karpovsky M et al (2003) Application of network calculus to general topologies using turn-prohibition. J IEEE/ACM Trans Netw 11(3):411–421
49. Duato J (1993) A new theory of deadlock-free adaptive routing in wormhole networks. Proc IEEE Trans Parallel Distrib Syst 4(12):1320–1331
50. Mural S, Meloniz P et al (2006) Designing message-dependent deadlock free networks on chips for application-specific systems-on-chips. In: Proceeding of the international conference on very large scale integration, Oct 2006, pp 158–163
51. Mural S, Meloniz P et al (2008) Designing routing and message-dependent deadlock free networks on chips. In: De Micheli G, Mir S, Reis R (eds) VLSI-SoC: research trends in VLSI and systems on chip. Springer, New York, pp 337–356, Chapter 19
52. Bui DN, Patel HD, Lee EA (2010) Deploying hard real-time control software on chip-multiprocessors. In: Proceedings of the IEEE international conference on embedded and real-time computing systems and applications, Aug 2010, pp 283–292
53. Bononi L, Concer N, Grammatikakis M (2007) NoC topologies exploration based on mapping and simulation models. In: Proceedings of the Euromicro conference on digital system design architectures, methods and tools, Aug 2007, pp 543–546
54. Concer N, Bononi L, Soulie M et al (2009) CTC: an end-to-end flow control protocol for multi-core systems-on-chip. In: Proceedings of the ACM/IEEE international symposium on networks-on-chip, May 2009, pp 193–202
55. Nesson T, Johnsson SL (1995) ROMM routing on mesh and torus networks. In: Proceedings of the ACM symposium on parallel algorithms and architectures, pp 275–287
56. Chiu G (2002) The odd-even turn model for adaptive routing. J IEEE Trans Parallel Distrib Syst 11(7):729–738
57. Hu J, Marculescu R (2004) DyAD: smart routing for networks-on-chip. In: Proceedings of the annual design automation conference, pp 260–263
58. Glass CJ, Ni LM (1992) The turn model for adaptive routing. J ACM Comput Arch News 20(2):278–287

http://www.springer.com/978-3-319-25764-8

Analysis and Design of Networks-on-Chip Under High Process Variation

Ezz-Eldin, R.; El-Moursy, M.A.; Hamed, H.F.A.

2015, XXI, 141 p. 84 illus., 34 illus. in color., Hardcover
ISBN: 978-3-319-25764-8