3.1 IMPLEMENTATION OF NOC ROUTER

3.1.1 Router Architecture

The research study has proposed a NoC router with lower area utilization. The general NoC architecture is illustrated in Figure 3.1 as shown below.

Figure 3.1 General block diagram of router architecture for a network-on-chip (Source: Swati Malviya & Anurag Jaiswal 2010)

The motivation to reduce the area is based on the fact that the less is area; the lesser is the power consumption. The study has chosen one of the
popular methods of buffering techniques of storing and then forwarding. This has proven to be the simplest possible decoding logic. Obviously reducing both area and power have been achieved. The connections are established automatically with simpler decoding logic.

3.1.2 Router Design

The router consists of five ports namely east, west, north, south and local port and a central cross point matrix. Each of the ports has an input channel and an output channel to receive and transmit data packets respectively. Data packets move through the input channel of one port of the router and then forwarded to the output channel of other port. Each input and an output channel have decoding logic which helps to perform the functions of the router. Buffers are incorporated at all ports to store the data temporarily. The store and forward type buffering method used. Control logic makes arbitration decisions and communication is established between input and output ports. The connection of configuration is made between ports and central cross point matrix. According to the destination path of data packet, control bit lines of cross point matrix are set. Directing the movement of data from source to destination is called switching mechanism. The packet size varies between 8 to 120 bits length. The method of data transfer of an on chip routing interconnect structure is known as Network on Chip (NoC) architecture (Kale & Gaikwad 2011).

3.1.2.1 Input Channel

The architecture of input channel at each port has its own control logic as shown in the Figure.3.2. Each input channel has a First Input First Output (FIFO) of depth 16, data width of 8 bits and a control logic which is implemented based on the model of Finite State Machine (FSM).
When the input channel receives request from neighboring routers, if it is free immediately it sends the acknowledge of the request. In the packet format, first flit is header and following flits are data bits, as long as the request signal is held high it accepts the data for transmission. Input channel flow chart is shown in Figure 3.3.

The output channel of requesting router ensures that the request line is held high until it empties the data packet, which is being accepted by the input channel. The input channel keeps the acknowledgement line at high, as long as the transfer of data is continued which is indicated by the request line.

Once the transfer data is accomplished, the request and the acknowledgement lines go low in a sequence. The data packets received from
the requesting router are stored locally in the FIFO for store and forward dataflow style. Next the control logic reads the header of the packet and decides which output channel is to be requested for sending out data packets from the router..

On decision, it sends the request to identified output channel. It is to be noted that each of the input channel is running an independent FSM and hence can initiate five possible parallel connections simultaneously. Once the

![Flow Chart](image.png)

**Figure 3.3 Input channel flow chart**
input channel gets a grant from the requested output channel, the control bits of cross point matrix are set accordingly to establish data flow process.

### 3.1.2.1.1 Functioning of input channel

The input channel starts functioning as soon as the input is given to it at _GNTL_. Each and every input from the user is given to the MUX, the operation starts if there is a data strings, such as “00001001” at input and the acknowledgement line is held high corresponding to the request (_req_) made.

### 3.1.3 XY Routing

At the input channel, once the FIFO is filled, the _X_-coordinate of the destination router (say for example _Hx_) is compared with locally stored _X_-coordinate of the router which first decides on the horizontal displacement as shown in the Figure 3.4. If _Hx_ > _X_ then the packet is forwarded to the east port of the router and if _Hx_ < _X_ then the packets goes out through the west port of the router. If _Hx_ is equal to _X_ then the _Y_-coordinate of the router is to be decided on the vertical displacement.

![Figure 3.4. Block diagram showing XY routing algorithm](image-url)
If \( H_y > Y \) the packet is forwarded to the north port and if \( H_y < Y \) the packet is forwarded to the south port. If \( H_y \) equals \( Y \), it indicates that the packet is at the destination router and thus the packet is forwarded to the local port. A packet is forwarded horizontally till the target column is reached and is then forwarded vertically to the destination router in a \( XY \) routing.

This means that there is no request for the east or west output ports by the north or south ports. For this reason the FSMs of the mentioned output channels are simplified, as they need not to wait for the said input ports. This results in a significant reduction in both the area and number of clock cycles in serving requests. This helps the implementation of light weight router, having area overheads at the minimum with acceptable level of reasonable performance. The functioning of \( XY \)-routing may be summarized as follows:

- \( XY \) routing algorithm routes packets first in \( x \)-direction (horizontal) to the correct column and then in \( y \)-direction (vertical) to the receiver.
- Then the routing operation is done on the basis of the conditions of \( X > Y \), \( X = Y \) and \( X < Y \).
- One of the advantages of \( XY \) routing is that it hardly enters into deadlock.
- The \( XY \) routing has addresses of the routers in terms of \( XY \) coordinates and is more suitable for networks based on mesh topology.

### 3.1.4 Crossbar

Crossbar switch consist of a set of multiplexers and demultiplexers. These are interconnected in such a way that all possible connections between the five input and output channels are established if required. The crossbar switch flow chart is shown in Figure 3.5.
The output channel while granting the request to an input channel configures the multiplexers and de-multiplexers of available input and output channels. Subsequently, the connections are established between channels for the transfer of the data packets. A cross point switch also called as matrix switch or crossbar switch connecting numerous input and outputs in a matrix scheme.

![Cross bar flow chart](image)

Figure 3.5 Cross bar flow chart
The crossbar switch design has five inputs and five outputs, a multiplexer crossbar switch is shown in Figure 3.6. All the multiplexers are connected to all five inputs with select lines of 3 bit long, out of five select lines available one will be selected based on the arbiter logic at a time. Outputs of multiplexers are the output ports of the 5×5 router (Zhizhou & Xiang 2010).

![Crossbar Switch Diagram](image)

**Figure 3.6 Architecture of crossbar switch for network-on-chip router**
(Source: Swati Malviya & Anurag Jaiswal 2010)

### 3.1.5 Output Channel

Based on the selection bits, the multiplexer works according to the request from the input channel. Further based on the input, the corresponding output will receive data from the crossbar. Each output channel of all the ports has an 8 bit FIFO at depth 16 and a control logic to make arbitration decisions. The output channel gets request from different input channels and
grants permission to anyone. Further it sets the control bit lines of matrix cross points as shown in the Figure 3.7. With the help of a single decoding logic into its FIFO the output channel accepts the packet as long as the sending inputs FIFO is not empty.

If the data transfer is completed, then the matrix cross point controls are reset. FSM then initiates the process forward the data into the nearby router using handshake algorithm, empty status of its FIFO initiates the next inter-channel transfer.

Figure 3.7 General architecture of output channel of a router for network-on-chip
3.1.5.1 Functioning of output channel

The output from the arbiter is sent to the de-multiplexer block of output channel and correspondingly it enables the particular output, i.e., “00001001” data packet is received at output channel from the crossbar switch.

3.1.6 Arbiter

A main element of router is arbiter; it is used to control the mechanism of packet switching. Arbiter can able to apply switch based algorithm in the router, there are two major methods of arbitrations such as matrix arbitration and round robin arbitration. System level arbiter architecture with IOs is shown in Figure 3.8.

![System level architecture of arbiter](Source: Suyog et al. 2012)

3.2 NOC ARBITRATION

3.2.1 Matrix Arbitration

In the arbitration of matrix algorithm, if all the input packets of all the ports (a, b, c, d and e) allotted the same priority request for same output port e.g. port b as shown in the contention table Table 3.3, based on the status of input and output ports the arbiter is checking the priority, and the matrix pattern
is generated. Then the matrix arbiter checks the priorities, based on the priority orders the matrix arbiter assigns the priorities to the input ports. Matrix arbiter generates a control signal to select the particular line and forward the packet from source to destination port (Suyog et al. 2012).

### 3.2.2 Round Robin Arbitration

Round-Robin (RR) is one of the simplest scheduling algorithms for interconnect communication. The RRA algorithm is self explanatory as shown in the Tables 3.1 and Table 3.2, based on the data address and selection line values the output is selected with or without contention as shown in Table 3.3.

![Diagram of proposed round robin arbitration](image-url)

**Figure 3.9 Architecture of proposed round robin arbitration**
### Table 3.1 Round robin arbitration without contention and with contention table

<table>
<thead>
<tr>
<th>INPUT PORT</th>
<th>SA</th>
<th>DA</th>
<th>Select line</th>
<th>O/P port without</th>
<th>O/P port with contention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port a</td>
<td>000</td>
<td>001</td>
<td>Selb (000)</td>
<td>Port b</td>
<td>Port b</td>
</tr>
<tr>
<td>Port b</td>
<td>001</td>
<td>010</td>
<td>Selc (001)</td>
<td>Port c</td>
<td>Port b</td>
</tr>
<tr>
<td>Port c</td>
<td>010</td>
<td>011</td>
<td>Seld (010)</td>
<td>Port d</td>
<td>Port b</td>
</tr>
<tr>
<td>Port d</td>
<td>011</td>
<td>100</td>
<td>Sele (011)</td>
<td>Port e</td>
<td>Port b</td>
</tr>
<tr>
<td>Port e</td>
<td>100</td>
<td>000</td>
<td>Sela (100)</td>
<td>Port a</td>
<td>Port b</td>
</tr>
</tbody>
</table>

I/P: Input; SA: Selection address; DA: Destination address; O/P: Output; Sel: Select; a, b, c, d and e: Ports

### Table 3.2 Matrix arbitration without contention table

<table>
<thead>
<tr>
<th>I/P port</th>
<th>SA</th>
<th>DA</th>
<th>Grant (4:0)</th>
<th>Select line</th>
<th>O/P port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port a</td>
<td>000</td>
<td>001</td>
<td>Ma (0100)</td>
<td>Selb (000)</td>
<td>Port b</td>
</tr>
<tr>
<td>Port b</td>
<td>001</td>
<td>010</td>
<td>Mb (00100)</td>
<td>Selc (001)</td>
<td>Port c</td>
</tr>
<tr>
<td>Port c</td>
<td>010</td>
<td>011</td>
<td>Mc (00010)</td>
<td>Seld (010)</td>
<td>Port d</td>
</tr>
<tr>
<td>Port d</td>
<td>011</td>
<td>100</td>
<td>Md (00001)</td>
<td>Sele (011)</td>
<td>Port e</td>
</tr>
<tr>
<td>Port e</td>
<td>100</td>
<td>000</td>
<td>Me (10000)</td>
<td>Sela (100)</td>
<td>Port a</td>
</tr>
</tbody>
</table>

I/P: Input; SA: Selection address; DA: Destination address; O/P: Output; M: Matrix; Sel: Select; a, b, c, d and e: Ports

### Table 3.3 Matrix arbitration with contention table

<table>
<thead>
<tr>
<th>I/P port</th>
<th>SA</th>
<th>DA</th>
<th>Grant (4:0)</th>
<th>Select line</th>
<th>O/P port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port a</td>
<td>000</td>
<td>001</td>
<td>Ma (01000)</td>
<td>Selb (000)</td>
<td>Port b</td>
</tr>
<tr>
<td>Port b</td>
<td>001</td>
<td>001</td>
<td>Mb (01000)</td>
<td>Selb (001)</td>
<td>Port b</td>
</tr>
<tr>
<td>Port c</td>
<td>010</td>
<td>001</td>
<td>Mc (01000)</td>
<td>Selb (010)</td>
<td>Port b</td>
</tr>
<tr>
<td>Port d</td>
<td>011</td>
<td>001</td>
<td>Md (01000)</td>
<td>Selb (011)</td>
<td>Port b</td>
</tr>
<tr>
<td>Port e</td>
<td>100</td>
<td>001</td>
<td>Me (01000)</td>
<td>Selb (100)</td>
<td>Port b</td>
</tr>
</tbody>
</table>

I/P: Input; SA: Selection address; DA: Destination address; O/P: Output; M: Matrix; Sel: Select; a, b, c, d, and e: Ports
It is used to operate the time slices and assigned to each process in the order of circular pattern without any priority. Architecture of RRA is shown in Figure 3.9. It is easy to implement and can be applied to other scheduling applications such as data packet scheduling in computer networks etc., (Si and Mei, 2007).

3.2.3 **Hamming Code**

Hamming code is a linear error correcting code, it is used to detect up to two bits errors and correct one bit error, hamming code is suitable to detect and correct the frequently occurring errors in on board electronic circuits of satellite communication, where as parity code does not correct the bit flips errors included by the radiation. The methodology applies hamming code in order to ensure the error free arbiter used in the router.

### 3.2.3.1 Calculation of the hamming code

The parity check bits of each byte of the S-Box LUTs are pre-calculated. The Hamming code bits can be mathematically represented as follows:

\[
\begin{align*}
    h(S_{\text{RD}}[a]) &\rightarrow h_{\text{RD}}[a] \\
    h((S_{\text{RD}}[a] \otimes 2) &\rightarrow h_{2\text{RD}}[a] \\
    h((S_{\text{RD}}[a] \otimes 3) &\rightarrow h_{3\text{RD}}[a]
\end{align*}
\]

Where, \(a\) is the state byte and \(h\) is the calculation of the Hamming code. Hamming codes algorithm steps are listed in Table 3.4 as shown.

**Table 3.4 Simplified simulation flow for hamming codes algorithm**

| Inputs: X = 1, 2, 3, 4, 5, etc Write the bit numbers in binary: 1, 10, 11, 100, 101, etc. All bit positions that are powers of two are parity bits determine: Values of parity bit as 1, 2, 4, 8... Change the position for parity bit 1 bit 1, 3, 5, 7, 9, etc. Change the position for parity bit 2 bit 3, 6, 7, 10, 11, etc. Repeat: Until change of position for the last parity End |
3.3 BIDIRECTIONAL ROUTER

3.3.1 Conventional Unidirectional Router

Unidirectional router performs the routing operation only in one direction. Main drawbacks of this routing logic include occurrence of path failures, dead lock and live lock problem. The 5×5 Round Robin Arbiter consists of number of OR gates, AND gates and D flip-flops.

Only one request is accepted at a given time. Similarly other requests are processed based on the priority of the data. Mask-able and Unmask-able priority arbiters are designed and implemented by using Round Robin Arbitration technique (Manjushree et al. 2014). An unmask able arbiter is granted, if the previous grant output is zero. This gives an input for a two input AND gate. If the request is one, the OR gate output is one and the output is inverted as shown in the figure 3.9.

Next, it gives an input to AND gate. Likewise the priority is generated for different input data. Once the unmasked priority is enabled, the next masked priority is processed continuously in routing algorithms. It defines the path followed by a packet from source to target switch.

The deadlock, live lock, and starvation situation must be avoided at all the times. Deadlock is defined as a condition of nodes requiring access to sources but cyclically keeps rotating without any forward development irrespective of any series of occurrences. Live lock is another condition in which the packets circulate in a network without moving towards intended destination. Starvation occurs when a packet requests an output channel in a buffer is always allocated to another packet and it never becomes available.

The routing mechanisms are classified into three different categories, the first refers to point where the routing decisions are taken, the
second category refers to how a path is defined and the third is a path length. According to what routing choices are chosen, it is possible to categorize the routing as a source and distributed routing. The whole path is decided at the source switch in source routing, where as each switch obtains a packet and describes the packet forwarding direction in distributed routing.

![Block diagram of conventional unidirectional router](suyog_et_al_2012)

**Figure 3.10 Block diagram of conventional unidirectional router (Source: Suyog et al. 2012)**

The header of the packet carries all the routing information in the source routing that increase the packet size whereas in the distributed routing the path is chosen as a function of the network traffic condition. The direction of deterministic routing is completely assigned from the relative position of address of the source and target. The destination is a function of instantaneous traffic of the network in adaptive routing and also increases the more possible paths usable by a packet to reach to its destination. But in fully adaptive algorithms live lock and deadlock situations can occur, that limits its usage.
Regarding criteria of the path length, routing can be minimal or non-minimal. Minimal routing mechanism ensures the shortest path between source and target. In non minimal routing, the packet can follow any available path between source and target. Non minimal routing offers great flexibility in terms of possible paths, but can result in lock situations and increase the latency to deliver the packet.

Conventional unidirectional router structure is shown in Figure 3.10. It consists of Round Robin Arbiter module, First in First out (FIFO) buffers and crossbar switches. Arbiter is used to grant the data based on the priority. Higher priority data are routed first. A FIFO buffer is used to store the data few times. It functions as a temporary storage device. These FIFO buffers are used in both input and output channel side. Crossbar switches is used to transfer the data, which comes from the arbiter. Control logic in the channel is incorporated in this router to send the control signal to crossbar switches. For example, input channel A can route the data through corresponding output channel A only, and cannot route the data via output channel B. Hence it is called unidirectional router (Minakshi et al. 2014).

3.3.2 Proposed Bidirectional NoC Router

In this methodology, the design of bidirectional Network on Chip router is presented, it avoids the contention situation. The Proposed router architecture of NoC is shown in Figure 3.11. This consists of In-Out port, Static RAM, Round Robin Arbiter, Routing Logic and Channel Control module. Arbiter is a type of device that chooses one output from a number of inputs based on the logic used. The number of input and output in crossbar switches are the same as in the arbiter. Normally, there are three parameters from the crossbar inputs which are to be identified.
Figure 3.11 Architecture of proposed bidirectional NoC Router
These are data, request and destination. Data is the information or the actual message to be routed. The information is to be forwarded to the output port for further processing. The address of the output port is included in destination information. The same address is used to find the path to send the information. The next parameter is requested, if the request is activated then the data is routed from the corresponding input port (Turhan & Karadeniz 2012). If two or more packets send the request simultaneously from the crossbar input, the round robin arbiter can enable only one request at a time. Thus one packet can only access the crossbar switch at a moment. This may cause loss of a few data packets. To address this issue in buffers of FIFO or memory (SRAM), an arbiter is incorporated. To obtain the lost packets, the output port of crossbar switch is stored in FIFO or SRAM. The packet from the FIFO or SRAM can be transferred in next clock cycle. It is a contention free crossbar (Rajalingam & Bechtel 2014; Sahar et al. 2013).

To pair the input ports and output ports, an allocation or a matching process is performed in a router. One input port is paired with only one output port to avoid conflicts. Initially, the router architecture is designed to use for routing operation in the wired network, then it is implemented for the method of matching maximization for present and next allocations. Finally, it provides the exhaustive operations of a matching with the future requests. In First in First out (FIFO), the data read/write occur in an order (Thomaset et al. 2010).

If an address pointer is incremented by one, FIFO full and empty flags point to the status of FIFO. The FIFO clears the data if the data is read from it. Random Access Memory (RAM) is a type of volatile memory, for which a read/write operation can be done randomly. This, however, is not possible in FIFO. It always starts from first address and the address pointer keeps on incrementing by one step at a time. If FIFO reads data from memory
at a given address, immediately the data are cleared off memory whereas in case of RAM, the data are available even after the read operation has been accomplished (Saravanakumar et al. 2013).

Dynamic RAM (DRAM) consists of transistors and capacitors. The drawback of DRAM is that it needs periodical refreshment of charge to compensate leakage power through discharging of capacitors. Moreover, the operating rules are different in DRAM. Because of these factors, the performance of DRAM is poorer than Static RAM (SRAM). However, chip size of DRAM requirement is smaller than SRAM due to the fact that DRAM erases the data immediately after the data has been read off. The major advantage of SRAM is higher speed of it than the DRAM (Young Tin Yoon et al. 2013). Static RAM consists of thousands of transistors which occupy more area, but it has less leakage power due to performance enhanced transistors. It also improves the readability. In this research work, the proposed router is designed using SRAM to improve the speed of the router and minimize the power leakage. Round Robin Arbitration is incorporated to minimize the area and dynamic power consumption. The source allocator and virtual channel allocator are used to control the channel. Virtual channel allocator is used to virtually change the corresponding channel direction and switch allocator is used to remove the path failures (Si & Mei 2007; (Subhananthan & Joy Merline 2014).

The proposed Bidirectional Router of NoC is used to transfer the data by three different methods. First, all input and output channels act as either slaves or masters. For example, if the first output channel acts as a slave, then the first input channel acts as a master. All other channels perform the routing operation in a similar way. Second, all the data from all input channels are routed through same output channel, this eliminates the path failures. Third, all input packets are transferred through all the output ports.
These avoid the live lock and dead lock problems. Also the proposed bidirectional NoC router uses less on chip area and performs at higher frequency than the traditional unidirectional router (Subhasakthe et al. 2014). The area and delay are reduced due to incorporation of high performance routing logic in the proposed bidirectional router (Suyog et al. 2012; Thomas et al. 2010; Turhan & Karadeniz 2012; Young Jin Yoon et al. 2013; Zhiliang Qian et al. 2012)

3.4 ADDRESS BASED ROUTER

3.4.1 Existing Priority Based Router

Similar to any other network designs, the network router is the most important component for the communication network with NoC incorporated in it. In a packet switched network, the main function of a router is to forward an incoming packet to the destination and in other cases it forwards the packet to another router for onward transmission. Our efforts are focused to keep NoC router design simpler which in turn may reduce cost as the complexity of a router design is directly responsible for increased cost. The design of router mainly consists of three parts: (i) FIFO, (ii) Arbiter and (iii) Crossbar Mechanism. The designs are explained in detail in later sections to follow.

Priority based Round Robin Arbitration (RRA) techniques are used to design the arbiter for NoC router. Robin arbiter is used to determine the priority of the request channel. It routes immediately the data with highest priority. Lower priority data are routed in subsequent steps. Bit wise XOR operation is performed to find errors if any occur in the data packet during channel transmission. The level of priority whether minimum or maximum is identified based on the request contained in the data packets (Si & Mei 2007; Bobda et al. 2005). The study has done comparative analyses of the
existing priority based router for the estimation of performance and subsequently proposed another algorithm for improved performance. The next sections explain proposed router.

### 3.4.2 Address Based Routing

The routing algorithms play a vital role in network path selection. A better routing algorithm is able to find an optimal path and it can be simpler with low overhead. It is desirable to strong, stable and same time it is required to converge quickly. Routing algorithms are implemented on software and hardware logic using routing table (Pionteck et al. 2006). There exists a number of routing algorithms developed for the both, specific and general networks. More efficient and robust implementation involves different challenges and novel approach. The routing algorithms can be trained and programmed as intelligent tools where these can learn the new conditions easily and adapt a variable environment. The statistical routing techniques cannot be efficient in modern age high quantum and high speed data transmissions. The algorithms can also have memory effects of past experiences which can be recalled and applied in current situations (Jovanovic et al. 2008). The characteristics of such algorithms require learning, dynamic adaptation and reasoning capacity for an improvement (Jovanovic et al. 2009). Based on these analyses, it has proposed a NoC router called Optimal Address based Router (OAR).

#### 3.4.2.1 The packet format

Packet format consists of mainly 3 parts of header data, information data packets and parity bits. The packet format describes in Figure 3.12 as shown. Normally a packet is 8 bit wide and 63 byte long. Packet header contains two fields Destination Address (DA) and length.
The packet destination address is 2 bits long, based on the packet destination address the router directs the packet to corresponding ports. Each output port has 2bit unique port address. The packet specifies the number of data bytes. A packet can be of a minimum data size of 1 byte and a maximum size of 63 bytes (Hosseinabady 2011).

If the length is 1 then data length is 1 byte and if length is 2 then data length is 2 bytes and so on. In case of length is 63 then the data length is 63 bytes. The payload data should be in terms of bytes and can assume any value. Parity field contains the information bits along with safety check bits in a packet. It is chosen as an even number of bytes for bitwise parity calculated over the header and data bytes of packets.

![Figure 3.12 A general data packet format of a typical NoC router](image)

The NoC design implemented here, consists of 6 main blocks as shown in Figure 3.13. These blocks are named as `fsm_router`, `router_reg`,...
ff_sync, and 3 FIFOs. The terms are chosen based on the variables used in programming codes for simulation test. The control signals to the router-reg module and FIFO are provided by the fsm-router block. The router_reg module contains the status bits, data and parity registers for the router_1×3. These registers are latched to input data or new status through the control signals generated by the fsm-router block. There are 3 FIFOs, one for each output port, which stores the data coming from input port based on the control signals generated by fsm_router module. The ff_sync module provides synchronization between fsm_router module and 3 FIFOs, So that single input port can communicate freely with 3 output ports (Ejlali et al. 2010). All labels are also renamed to keep an easy to understand terminology.

Figure 3.13 The simplified schematic diagram of NoC router implemented
3.4.2.2 The main feature of implemented router

The functionality of router can be explained as Packet valid signal is made high until the last information data byte has occurred. Then it should be made low at the time of sending the parity byte. Information data size is controlled by length field. Packet parity comes from the source network, then it does the operation of XOR (bitwise) with header byte. It is called computed parity. Again, it does XOR operation with information data byte. The read enable signal is made high if the valid channel is high i.e. FIFO should not be empty during read operation. Suspend data signal goes high, whenever the header byte is received. After this operation suspend data signal is made low else in otherwise case the next data cannot be loaded. After parity signal is exceeded, suspend data signal is made high so that another packet can enter. Error signal is generated if there is a mismatch between packet parity and calculated parity. When loading the first information data, write signal should be made high and decode address is made low. Once the decode address is low, it holds the last value of write signal. Decode address helps to point the same FIFO till parity byte comes.

3.4.2.3 The FSM and FIFO synchronization block

This block actually detects the address of channel and latches it till packet valid is asserted. The FSM model is described in Figure 3.14 as shown below. Address and write_enb_sel are used for latching the incoming data into the FIFO of that particular channel. A FIFO full output signal is generated, if the present FIFO is full, and FIFO empty output signal is generated by the present FIFO, if it is empty. Conditions of data are explained as follows.

- If \( data = 00 \) then \( FIFO_{empty} = empty_0 \) and \( FIFO_{full} = full_0 \)
- If \( data = 01 \) then \( FIFO_{empty} = empty_1 \) and \( FIFO_{full} = full_1 \)
- If \( data = 10 \) then \( FIFO_{empty} = empty_2 \) and \( FIFO_{full} = full_2 \)
The output signal $vld\_out$ is generated if $empty$ of present $FIFO$ goes low, which means that present $FIFO$ is ready to read further as explained below
\[ vld\_out\_0 = \sim empty\_0, vld\_out\_1 = \sim empty\_1, vld\_out\_2 = \sim empty\_2. \]

The ‘fsm\_router’ module is the controller circuit for the router. This module generates all the control signals whenever a new packet is sent to router. These control signals are also used by other modules to send data at output and write data into the $FIFO$. The $write\_enb\_reg$ signal from the $FSM$ module is used to generate $write\_enb$ signal for the present $FIFO$ which is selected by present address.

$State\_decode\_address$ is a default state. It waits for the $packet\_valid$ assertion if the address is valid and packet valid signal goes high, and FIFO for that address is empty ($FIFO\_empty$ signal is high for this condition) then data can be loaded. Then it goes to the next state $LOAD\_FIRST\_DATA$. If $FIFO$ is not empty it goes to $WAIT\_TILL\_EMPTY$. This implies that new data can’t be accepted till $FIFO$ is ready. If the output signal $detect\_add$ is made high, then $ff\_sync$ module can detect the address of $FIFO$ which can be further used. $Detect\_add$ signal is also used by $router\_reg$ module to latch the first byte in internal register.

3.4.2.4 Different states of router

The first state is $STATE\_LOAD\_FIRST\_DATA$ where the state $lfd\_state$ signal is generated, which gives an indication to the $router\_reg$ module that first data byte can now be latched. At the same time $suspend\_data$ signal goes high so that first data byte can be latched to the output data register in $router\_reg$ module. During the next clock edge, this state is changed to $LOAD\_DATA$, unconditionally.
The second state \textit{STATE-LOAD\_DATA} during which the data are latched to the data registers of \textit{router\_reg} module, if \textit{ld\_state} signal is generated for \textit{router\_reg} module. \textit{Suspend\_data} signal is made low, so that router can accept the new data from input simultaneously and subsequently latched data are sent to the \textit{FIFO} and \textit{write\_enb\_reg} signal is generated for writing into present \textit{FIFO}. If \textit{FIFO\_full} signal turns high then no more data can be accepted by router and it changes to indicate \textit{FIFO\_FULL\_STATE}. Data are latched until a \textit{packet\_valid} signal is asserted. Again, if it is de-asserted during \textit{LOAD\_DATA} state, it goes to \textit{LOAD\_PARITY} state and the last \textit{parity} byte is latched.

The third state is \textit{STATE – LOAD\_PARITY} where state last byte is latched. It is called as a parity byte. If \textit{FIFO\_full} signal is high, data cannot be latched. It, therefore, goes to state of \textit{FIFO\_FULL\_STATE} and else if \textit{FIFO\_full} is low, it goes to state of \textit{CHECK\_PARITY\_ERROR}. Signal \textit{lp\_state} is generated for \textit{router\_reg} module. If \textit{Suspend\_data} signal goes to high state then router doesn’t accept any further data. \textit{Write\_enb\_reg} signal is made high for latching the last byte. The \textit{lp\_state} signal is generated for the \textit{router\_reg} module to latch the last byte and now the parity bytes can further be compared.

The fourth state is \textit{STATE – FIFO\_FULL\_STATE} where neither new data are accepted nor any more data are latched further. For this condition, \textit{suspend\_data} signal goes high and \textit{write\_enb\_reg} signal goes low. A signal \textit{full\_state} is generated for \textit{router\_reg} module. This state changes to \textit{LOAD\_AFTER\_FULL} state if \textit{FIFO\_full} goes low.

The fifth state is \textit{STATE – LOAD\_AFTER\_FULL} where a \textit{laf\_state} signal is asserted for \textit{router\_reg} to latch the data. Once the \textit{FIFO\_FULL\_STATE} is asserted no new data is accepted. For this, \textit{suspend\_data} signal goes high.
Figure 3.14 Finite State Machine Model showing state flow for optimal address based NoC Router
The last data are latched in router_reg module if the write_enb_reg goes high. It checks for parity_done register and if it is found to be high then the LOAD_PARITY state has passed. If parity_done is low, it goes to the last state CHECK_PARITY_ERROR and it checks for low_packet_valid register. If this register is high then packet_valid for present packet is de-asserted. If low_packet_valid is high, then it goes to LOAD_PARITY state otherwise it goes back to the LOAD_DATA state.

The sixth important state is STATE – WAIT_TILL_EMPTY where neither new data are accepted nor data are latched by router_reg module. For this the suspend_data signal goes high and write_enb_reg signal goes low. It waits for the FIFO_empty signal which if goes high then the state goes to the LOAD_FIRST_DATA.

The seventh important state is STATE – CHECK_PARITY_ERROR wherein the state reset_int_reg signal is generated to reset the status and parity registers of the router_reg module. Neither any data are latched, nor is any input data are accepted. The router_reg compares the data parity of packet with that one of calculated parity during this state. The state changes to default state DECODE_ADDRESS on the arrival of next clock edge

### 3.5 ADAPTIVE LINK BUFFERS

The high speed VLSI architectures require repeaters and the data path operators to perform efficiently at much higher data transfer speeds. The existing repeaters are replaced by tri-state repeaters capable of sampling and holding the data on an inter router link. It gives the flexibility to function in a reconfigurable adaptive routing mode during the flow of network traffic. The controlled circuit mainly consists of a pass transistor, capacitor and clock. The capacitor switches the global clock. Pass transistors are driven by the clock signals. The Global Clock Signal (GCLK) is subsequently divided into two
signals, GCLK1, and GCLK2. The frequencies at sub clocks should be half of global clock. The gate control input signal at the pass transistor takes care of congestion control operation. The congestion control signal controls the repeaters which are synchronized with global clock. The capacitor is charged and discharged via pass transistor path. Due to the capacitor charging and discharging, the congestion control signal is much delayed for each clock cycle (Mizuno et al. 2001). Shared buffer is shown in Figure 3.15.

![Diagram of Shared Buffer Router](image)

**Figure 3.15 Internal structure of shared buffer router.**

For the subsequent clock cycles, the corresponding repeaters operate tri-state buffer to hold the data in a position. Meanwhile the congestion signal travels to next control block. The main objective of the control block is to reconfigure the adaptive link buffer is shown in Figure
3.16. The field programmable gate array circuit is included in order to reduce the area, power and latency of the NOC circuit.

![Diagram of Adaptive Buffer Architecture](image)

**Figure 3.16 Adaptive buffer architecture.**

3.5.1 **Reconfigurable Computing**

The reconfigurable adaptive link buffer is simulated in Xilinx platform using Verilog. It is illustrated in Figure 3.17 with four stages of link buffer. If congestion signal goes high, the output is tri-stated and delayed by one clock cycle ranging 1 to 10 ns. Data are released if a congestion signal goes low at output ranging from 10 to 22 ns. The proposed control circuit acts as an optimum router and congestion controller. The power efficiency and control circuit can be turned OFF if there is no congestion signal. In addition to it, a single control block can drive the control input of the entire adaptive buffer at a given stage. The simple switched capacitor maintains clock accurately at variable frequency. It also provides signal stability at high clock speeds (Ho et al. 2001).
3.5.2 Field Programmable Gate Array

Reconfigurable computing consists of an array of elements that can be interconnected easily by using Field Programmable Gate Array (FPGA) principles. The interconnection element is user programmable, block level FPGA representation consists of a two-dimensional array of logic blocks that can be connected by programming. These logic blocks are connected through wire segments of different lengths. Interconnects are programmable switches which connect wire segments to the logic blocks. Logic circuits in FPGAs are partitioned into logic blocks and subsequently the logic blocks are interconnected by using switches.

FPGAs are capable of implementing digital circuits of various sizes. The size of circuit is also called as the density of the circuit. There can
be a trade-off between complexity and flexibility of both the logic blocks and the interconnection of an FPGA (Kamal & Yadav 2012).

### 3.5.3 Reconfigurable Design using FPGA

FPGA based digital circuit design is based on CAD principles. The digital circuit design can be optimized by using FPGA implementation tools. For this process, the initial logic entry may be syntax based or it may involve the drawing of a schematic using a schematic capture tool. For syntax based programming, entering an HDL description is essential. It can be done by specifying Boolean expression or a state machine entry.

The Boolean functions are optimized by using logical tools for on chip area or data transfer speed or similar other applications. For simulations, first the Boolean expressions are implemented as digital circuits on FPGA logic blocks. This is called technology mapping. After mapping the digital circuits on logic blocks, placement is optimized by deciding a location for each logic block to be configured in the FPGA (Blake et al. 2005). The last step for implementation process is the routing of logic blocks. For this, a number of algorithms are available for different FPGA architectures. After completion of all the steps of simulation tests, the architecture is implemented on FPGA by using a programming module which configures the hardware design on FPGA chip.

### 3.5.4 Routing Topology

Routing topology is based on the flow control techniques and a number of available routing algorithms for optimization. The flow control module focuses on buffer size and channel bandwidth requirements. The adaptive reconfigurable NOC can be used to eliminate the buffer unit of architecture. This reduces on chip area and thus less power consumption.
Wire delays are normally impacted by channel bandwidth allocation schemes. Routing is accomplished in two steps. The first step is called global or loose routing, it focuses on wiring channels.

Figure 3.18 Reconfiguration design flow.

Figure 3.19 Wormhole routing architecture and data flow
The second step is called detailed routing, it ensures precise wire routes for different channel layers. Reconfiguration design flow and warm hole routing are shown in Figure 3.18 and Figure 3.19

3.5.5 Area and Channel Routing

The reconfigurable routing mechanisms like Maize routings are largely used for optimization of channel and area allocation. The efficient paths between source and destination are identified by executing algorithm. The total on chip area is estimated in order to implement the architecture with reduced chip area, global delays and local delays. Channel routing is used to find the shortest path between the interconnected sub modules of architectures. Both area and channel routing algorithm establishes proved and improved efficiency network data traffic of the NoC. The brief steps of Lee’s Maze mechanism are listed for illustration of execution flow of routing (Karam et al. 2009).

Struct grid_point

{Int value;
 /*zero for unused, positive for label, -1 for obstacles*/} Lee (struct grid_point S,T)

{Set of structgrid_point wave front, new_wave_front:
Structgrid_pointneighbor, element, path_element:

Int label;

/*step1:wave propagation*/ New_wave_front {S}; Label ≤0;

While(T€ new_wave _front)

{label ≤label+1; wave_front ≤new_wave_front; new_wave_front ≤Ø;

for each element € wave_front for each “neighbor of ” element

/*a neighbor is located above, below, at the left or at the right.*/}
If (neighbor.value = 0)
{Neighbor.value = label; New_wave_front = new_wave_front(neighbor);}

/*step 2: back tracking*/
Path_element = T;
For (i = label; L1; i-1)
{Path_element = the neighbor of path_element such that neighbor.value = i;}
/* in case of multiple possibilities use a heuristic to make a choice. */
Path_element.value = -1

/*step 3: clean up*/
For each point “on the grid” If (point.value > 0) Point.value = 0 ;}

3.5.6 Channel Routing

robust_robust_router(struct netlist N)
{ Set of int row; Struct solution S;

Int total[channel_width+1], selected_net [channel_width+1];
Int top, height, c, r, i; top = 1; height = density(N); for (r = 1; r < height; r++)
{for all “nets I in netlist N” wi = compute_weight; (N, top);

total[0] = 0;
for (c = 1; c < channel width; c++)
{selected_net[c] = 0;

total[c] = total[c-1];

if (“some net n has top terminal at position c”)

if (wn + total[xmin-1]) > total[c])

{Total[c] = wn + total[xmin-1];

}}
selected_net[c] = n;

If (“net n has a bottom terminal at position c”)
if(wn+total[xnmin-1])>total[c])
{total[c] = wn+total[xnmin-1]);
 selected_net[c] = n; } row = Ø;
c = channel_width; While (c>0) if(selected net[c])
{ n = selected _net[c]; row = rowU{n}; c = xnmin-1;} else
F = solutionU{row}; top = !top;
1 = “N without the nets selected in row”}

“Apply maze routing to eliminate possible vertical constraint violations”}

3.5.7 Performance Metrics

3.5.7.1 Latency

The main components responsible for the most of the delay are the cascaded drivers at the input and the bus trunks. These contribute for the delay due to inherent nature of architecture. The bus delay can be mathematically modeled as shown in equation 1

\[ \tilde{b} = \frac{k_1CL_1}{k N_1/K} + k_2N + k_3N^2 \]  \hspace{1cm} (1)

where \( \tilde{b} \) is bus delay, \( N \) is number of switches, \( K \) is number of nodes and \( C \) is control input. There are a few types of switches, suitable for NOC, such as crossbar switch. A standard crossbar has an \( N \) input and \( N \) output as well as a control input signal. Any input may be connected to any of the outputs or to any combination of outputs as required with the help of control signal. One way to implement a crossbar switch is using MOS transistors as shown in
A data path from each input connects to N switches and in the same way each switch is connected to one of the N output data paths. An alternative design using multiplexer is shown in Figure 3.21. A 2-to-1 multiplexer is usually used to select the desired input for each output. The multiplexer based crossbar switch has many advantages such as reduced time delay for selecting the optimum data path operators.

Multiplexer tree hierarchy is shown in Figure 3.22. The average delay times of a switching circuit normally depends upon the topology of the NOC in this research work, the reconfigurable adaptive routing is used for delay measurement and complexity analysis for balanced tree technique. The balanced tree technique is illustrated in Figure 3.23. Table 3.5 listed the comparison of network delays for different parameters.

The proposed NOC routing uses 2×2 MUX based crossbar switch. This switch is normally used for effective delay minimization for bus and crossbar switch.

Table 3.5  Comparison of network delays for different parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>O (N²)</td>
</tr>
<tr>
<td>Switch-based crossbar</td>
<td>O (N)</td>
</tr>
<tr>
<td>MUX -based crossbar</td>
<td>O (log N)</td>
</tr>
</tbody>
</table>
Figure 3.20 Wormhole switch model architecture.

Figure 3.21 Architecture of a 2-to-1 multiplexer cell.

Figure 3.22 Multiplexer tree hierarchical block diagram.
3.5.7.2 Power and energy consumption

There are many ways to put in strategies for building low-power gates. These may include multiple threshold inverter, variable threshold CMOS (VTCMOS) gate etc. All these methods are more aggressive method, in which variable threshold CMOS (VTCMOS) gate is more popular and frequently used for various applications (Sarathy et al. 2008). VTCMOS logic can wake up much faster compared to other logic structures.

![Balance tree hierarchical block diagram](image)

Figure 3.23 Balance tree hierarchical block diagram.

3.6 ROBUST RECONFIGURABLE ROUTING ALGORITHM

3.6.1 R3 Algorithm for NoC Router

Lee and Sachen proposed a very significant research on global routings. Similarly Griffith introduced rectilinear Steiner trees algorithm. The new algorithm is based on the concept of large number of logic gates that are to be optimized for layout using standard cells. It is different in the sense that the vertical routing channels can be used instead of feed through wires. The number
of routing tracks directly represents the number of channels. The proposed algorithm is a good choice for the implementation of NoC router for the various applications.

3.6.2 Problem Definition

The layout of the grids is accomplished in horizontal and vertical directions separately for specific purposes. The horizontal lines are laid out across the cell similar to rows appearance and the vertical lines are laid out in the vertical direction. The vertical lines are chosen equal to the horizontal lines. There is no fixed rule for the distance between horizontal lines but the channel routing decides the distance. The grid has routing area divided into rectangular elementary blocks.

**Figure 3.24 A Simplified Diagram of a Mixed CMOS/nano System**

The design layout depends upon the grid usage which implies that terminals are placed in the same rectangular area. The rectilinear Steiner trees interconnect all points that lie at the center of the unit rectangle as shown in Figure 3.24. The main function of local routing is to refine a global solution by allotting exact position of the wires in the grid layout
3.6.3 CMOL-FPGA Architecture Implementation

CMOL technology is based on a semiconductor nano wire molecular architecture (Strukov & Likharev 2005). CMOL architecture is a versatile technique which can be used for any of the CMOS modules and arbitrary electronics circuits. Contrary, a nano module is always restricted to a regular circuit only. The CMOL technology is most suitable for FPGA architectures. The molecular nano scale circuits with two level nano wire crossbar form a junction. This junction has ability to act as nano wire cross point where connections are reconfigurable. The reconfiguration property is very important for CMOS and molecular structures.

The self assembling nano molecules cannot be very reliable for good performance. It has also alignment difficulties. This results in a reduced performance in very large production. The CMOL modules may have high molecular defects. It, however offers better efficiency in terms of power dissipation and area occupation compare to CMOS circuits. The CMOL has nano junctions of two cross wires which operate nano device with two terminals. The cross points electrically activated or deactivated at the cross points of the mesh topology structure, depending upon the circuit configuration requirements (Masoumi 2006).

The fabrication process in CMOL is easier compared to the three-terminal nano modules. The transistors integrated in the CMOS subsystem supplement the operation of the two terminal modules since two terminal devices are less efficient. Two terminal nano modules offer a few advantages while CMOS subsystems provide the operations such as address decoding, voltage gain and output signal sensing. This results in better performance with low power consumption and high speed. It is assumed that the molecular level interconnection will become common in nano crossbar junction points in near future. The nano scale wire crossbar point implementation of two
terminal modules will be a breakthrough for mixed CMOS circuits with nano molecular design structure. Such a design is illustrated in Figure 3.25.

The CMOS hybrid structure consists of a molecular structure that functions as a latch circuit switch activated by two input signals. The general architecture of a FPGA CMOL and a CMOS cell is depicted in Figure 3.25. It presents nano wires cross bar networking. Two terminal nano molecular devices are assembled between the wires at each cross points. The nano molecules works as latching switches or rectifying characteristics, just similar to that of the PN junction diodes (Masoumi 2006). The nano molecular wires are rotated with respect to CMOS underlying cells to make contacts. The configurable property is achieved by CMOS cells accessible through the row and column lines. The circuit cell configuration is made of CMOS NOT gate as shown in Figure 3.26 (a) with its row and column lines as shown there.

![Figure 3.25 The hybrid CMOS Molecular Circuit with Nano Wire Cross junctions and Nano Devices with Superior Performance](image-url)
The connection wires are placed in each cell to make connections between the bottom and top wire meshes, where the inputs are connected to the bottom and outputs are connected to the top of nano molecular wires. Similarly, many molecules are connected to different cells. There are small gaps may be observed in between nano molecular wires of CMOL FPGA structure. A square shaped connectivity domain is defined for each cell. Connectivity domain is a region around a given cell, which connects inputs or outputs of original cell to any cell in the region. If all the molecules conduct, then all the inputs and outputs of the cells are connected to act as latching switches. It can be controlled for appropriate cells to conduct on the respective address lines. The each output cell can be ON or OFF relative to the input cell. The molecules connect the respective output to its domain connectivity. Each cell of its corresponding molecular modules is considered as a NOR gate as shown in Figure 3.26 (b). Any of the Boolean functions can be implemented by using NOT and NOR gates alone. The functioning of the same may be summarized as follows.

Figure 3.26 A Typical CMOS Cell used in the Topology of CMOL
(a) Each rectangular cell is made up of a NOT gate and its input-output make the contact to separate layers of nano wires in the structure.

(b) Each of a CMOS transistor makes an inverter cell with its corresponding address lines as ‘pass’ transistors.

(c) At the cross point of the wire meshes, the molecules can turn ON the junction in the nano wire meshes.

(d) The three inputs NOR gate circuit representation is equivalent to a NOR gate with diode-resistor connections as shown in the Figure 3.26.

The nano devices are highly non reliable and there can be high defects due to misalignment of nano wires and nano devices. If the mapping of CMOS configuration are not properly achieved then another adjacent cells can be used by virtue of its connectivity domain to operate circuit properly. The swapping is provided by using a different routing algorithm instead of physical relocation of the gates. The reconfiguration process is illustrated in Figure 3.27. The CMOL tile is analogous to the island model of CMOS FPGA architectures (Strukov & Likharev 2005).

Normally the original logic circuits are mapped onto a network of NOR gates. The original circuits are not optimized but have well defined fan-in and fan-out. The circuits are then latched to generate a net list. Further it assigns a particular number to CMOS cell easy identification for performing certain logic functions. The other CMOS cells are dedicated for routing operations. The Figure 3.27 illustrates the following summary.

(a) A combination of a NOR and a NOT gate functions as a logic XOR gate.
(b) Assume that in a CMOL rectangular block circuit mapping, at least one of the molecules of gate is defective and hence this gate is to be relocated.

(c) The intersections of a faulty region of I/O cells of affected gate are again determined to relocate the region of the gates.

The connection can be swapped so as to remove the defective regions and relocate the new gate for functional continuity. Similarly other defective operations are repeated for the detected cases.

Figure 3.27 A Typical Illustration of a CMOL FPGA Circuit