Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

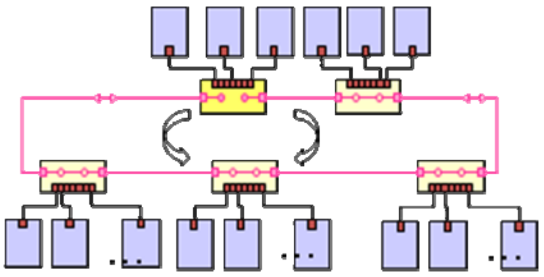
Institut für
Technische Informatik und
Kommunikationsnetze

High Availability Seamless Ring Protocol Implementation in FPGA

Xiaozhuo Jiang

Master Thesis

2009, February 13



Advisor ABB: Prof. Dr. Hubert Kirrmann

Co-Advisor ABB: Dr. Jean-Charles Tournier

Supervisor ETH: Prof. Dr. Lothar Thiele

Computer Engineering and Networks Laboratory, ETH Zurich

Abstract

The IEC committee SC65 WG15 “Highly Available Automation Networks” published the IEC standard 62439 in 2008 to provide several redundancy methods to overcome the lack of a commonly accepted redundancy solution in Industrial Ethernet. As one of these methods, the Parallel Redundancy Protocol (PRP) IEC62439 Clause 6 relies on the parallel operation of two local area networks, and provides completely seamless switchover in case of failure of links or switches, thus fulfilling all the hard real-time requirements of substation automation. The High Availability Seamless Ring (HSR) which is proposed in IEC 62439-3 applies the PRP principle to build a simple seamless ring by treating each direction as a separate network.

One main application of the HSR is the process bus in the substation automation network as specified in the IEC 61850 standard. This application is characterized by heavy traffic, real-time communication and requirement of bumpless switchover in case of fault.

One challenging issue in HSR is to reject the duplicated frames coming from the both ports of a node and the frames circulating in the ring. The algorithm applied in PRP cannot be wholly transplanted into HSR because of HSR’s ring topology. In this thesis three algorithms are proposed to solve the duplicated and circular frame discarding. The proposed algorithms are compared with the help of the software network model. One algorithm is chosen. By designing a switch structure with the integration of the chosen algorithm, the HSR protocol is implemented in FPGA.

CONTENTS

1 Introduction

1.1 Context

In April 2008 the IEC committee SC65 WG15 “Highly Available Automation Networks” published six industrial communication network protocols dedicated to provide the redundancy of the network. Among these 6 protocols, the Parallel Redundancy Protocol (PRP) has been selected for the IEC 61850 communication network within substations.

PRP applied the “redundancy in the nodes” method to offer the redundancy. Each PRP node (called a DANP or Doubly Attached Node with PRP) is attached to two independent LANs which may exhibit different topologies. The networks are completely separated and are assumed to be fail-independent. Figure 1.1 shows the topology of PRP.

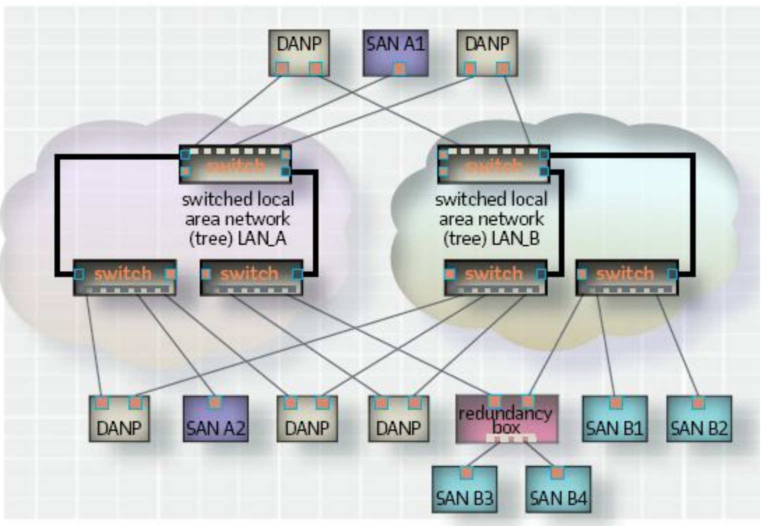


Figure 1.1 An illustration of the PRP topology [1]

Compared to other protocols released in standard IEC62439, PRP provides completely seamless switchover in case of failure of links or switches, thus fulfilling all the hard real-time requirements of substation automation. Whereas other protocols like RSTP and MPR (Media Redundancy Protocol) need a recovery time [2].

Since PRP requires doubling the network infrastructure, it overkills in some relative simple application scenarios. The High Availability Seamless Ring is proposed based on the PRP protocol (Figure 1.2). It allows a significant reduction of the hardware costs, since no switch is used, instead each end node must be equipped with a switch element to implement the ring protocol. Although the network layer and the algorithms in PRP and HSR are nearly the same, HSR cannot apply all the algorithms in PRP because of its ring topology and the application scenario of the HSR, which is characterized by high network traffic. Therefore additional features and modifications must be made to guarantee the performance of HSR. This becomes the motivation of this Thesis, which will be explained further in the next section.

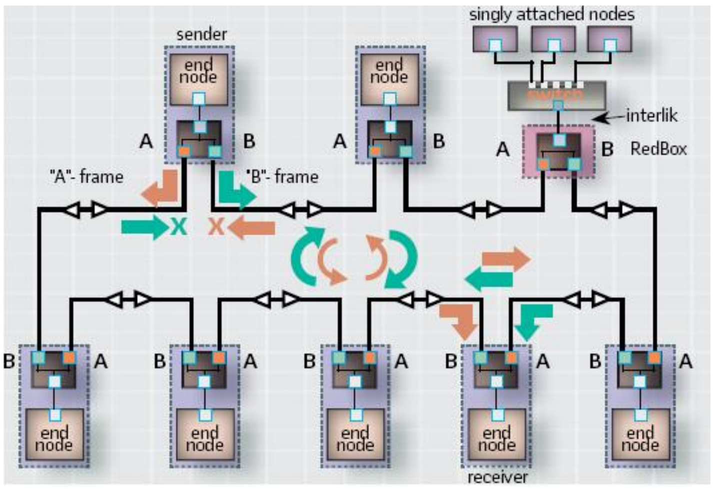


Figure 1.2 An illustration of the HSR topology

1.2 Motivation

In PRP, an additional layer, the “Link Redundancy Entity” (LRE) (Figure 1.3) is placed under the link layer.

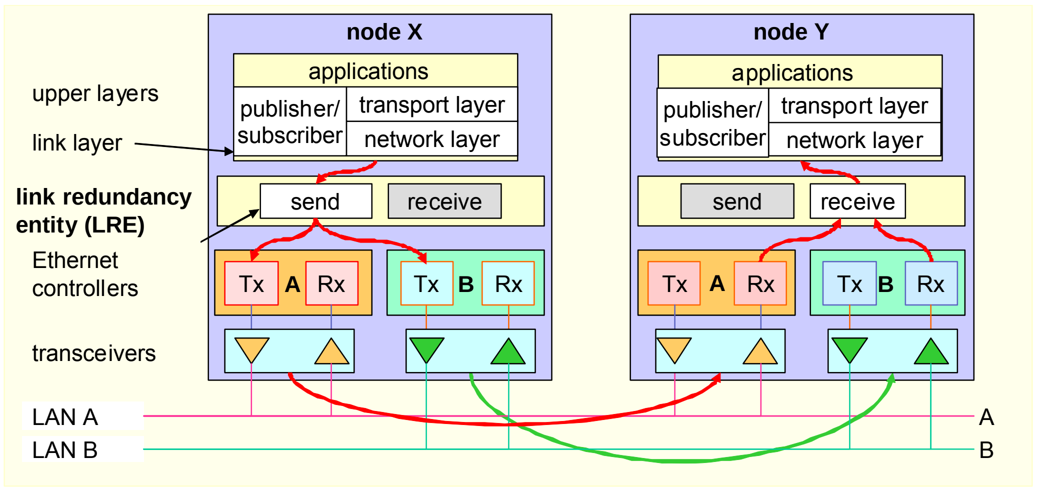


Figure 1.3 Location of the LRE in the network layer stack of PRP [3]

The LRE connects the upper layer and the two Ethernet controllers and behaves towards the upper layer like a single Ethernet interface. When transmitting, it appends the Redundancy Check Tag (RCT) in the outgoing frame, duplicates the frame and sends both frames over the two sending ports of the node. These two frames travel through the network and arrive at the receiving node with certain time difference. If the receiving node works in the duplicates accept mode, the LRE receives both frames of a pair and forwards them to the upper layer. The rejection of one duplicate should occur at the higher layer in principle. In this case, the designed application should be able to handle duplicates. For example, the TCP is designed to handle duplicate. Applications using UDP must be able to treat duplicates since UDP is a connection-less protocol. If the receiving node works in the duplicate discard mode, the LRE should pass the first frame of a pair to the upper layer and reject the second.

The duplicates rejection function of the LRE can be realized in software in PRP. But In the application scenario of HSR, the communication is often very heavy. If the function of rejecting the duplicates of the LRE is executed in software, the processor will be heavily loaded.

Besides, unlike in PRP that nodes only send and receive frames, nodes in HSR must also be able to forward frames based on the forwarding rules. The routing of the frame is executed by the switch logic in HSR nodes. Again, if the switch logic is implemented in software, the processor is heavily loaded due to the heavy traffic in HSR.

To offload the processor, the duplicates rejection function of the LRE and the switch logic are preferably implemented in hardware. The performance of the implemented duplicates rejecting function should apply the “best effort principle”.

One challenging issue in the implementation is to find an algorithm to reject the duplicated frames and circular frames (the generating of the circular frame will be explained in the next chapter), because the algorithm used in PPR is not resource efficient and not designed for handling the circular frames. Therefore, the motivation of this Thesis is to find an effective algorithm to reject the duplicated and circular frames and implement the HSR protocol (the switch logic and the LRE) together with the selected algorithm in hardware.

1.3 Tasks

The Tasks of this thesis are divided into 4 parts:

First, different algorithms are proposed to reject the duplicated and circular frames.

Second, a software simulation environment is set up to test the proposed algorithms.

Third, the hardware structure of the switch is designed and programmed in VHDL language, the functionality is simulated with the ModelSim of Menthor Graphics.

Fourth, the VHDL code is synthesized with Quartus of Altera and the function of the design is evaluated by the pos-Synthesis simulation.

1.4 Contributions

In respect to the work of rejecting the duplicated frames in HSR, the University Zurich of Applied Science has done an architecture study of the Redundancy Box in year 2007 [8]. In 2008 they implemented the RedBox in FPGA [7]. The algorithm they applied for rejecting the duplicated frames is “circular buffer”. In this method, a frame is declared as not in the table only after the entire table is searched, which introduces a long delay. The work by Christoph Klarenbach introduced a method of integrating the real-time Ethernet in the FPGA[26], which gives an example for the FPGA implementation in this Thesis.

In this Thesis, 3 algorithms are proposed to reject the duplicated and circular frames in the ring. A software test environment has been set up to simulate the communication in the ring. The performance of the 3 algorithms is simulated and compared with the help software simulation environment. The advantages and disadvantages of the algorithms and their different configurations are also explained in respect of the performance and the hardware complexities. The structure of the switch has been designed to implement the HSR protocol including the selected algorithm.

The selected algorithm for rejecting the duplicated and circular frames has reached almost 100% rejection ratio under the network configuration given in IEC61850-9-2 standard and in a network scale less than 64 nodes in a ring. The searching time is significantly reduced compared with the algorithm proposed by the University Zurich of Applied Science. The limitation of the proposed algorithm in respect to the scale of the network is further discussed. Finally, the designed switch executes all the functionalities successfully according to the pos-Synthesis simulation.

1.5 Overview

In chapter 2, the High Availability Seamless Ring is introduced. The introduction includes its topology, the communication rules, and the challenging issues by now.

In chapter 3, the three proposed algorithms are explained. The advantages and disadvantages are discussed in respect to the searching time and the complexity of the hardware implementation.

In chapter 4, the software simulation model is introduced. The performance of the three proposed algorithms with different configurations is compared. A short insight into the limitation of the proposed algorithms is given.

In chapter 5, the designed structure of the switch is explained; the way each component works is described in detail, several issues in the FPGA design is introduced. The programmed switch is synthesized in Quartus, a brief synthesis report is presented. After the synthesis, the pos-Synthesis simulation is performed to verify the correctness of the design

In chapter 6, a conclusion of this thesis and a view of possible future work are given.

2 The High Availability Seamless Ring

This chapter gives an overview of the High Availability Seamless Ring (HSR) protocol. The topology and the communication rules of the HSR are first described, then the network layer model and the structure of the switch unit is illustrated. The typical application scenario is introduced and the problems existing in the communication by now are explained.

2.1 The Topology of HSR

One topology of HSR is shown in Figure 2.1 and Figure 2.2 with unicast traffic and multicast traffic respectively. Each end node has two ports connected to the ring. For each frame to send, the node sends it duplicated over both ports (A-frame and B-frame). One frame of the pair travels in the ring in the clockwise direction, the other frame travels in counter-clockwise direction.

Nodes within the ring are restricted to be HSR-capable switching end nodes. General purpose nodes (e.g. the singly attached node in Figure 2.1) cannot be attached directly to the ring, but require a Redundancy Box (RedBox). A pair of such RedBox can be used to connect hierarchically a HSR to a PRP network. All the frames in the ring must be a HSR frame. A non-HSR frame must be appended with a HSR tag when it entries the ring (blue arrows in Figure 2.2).

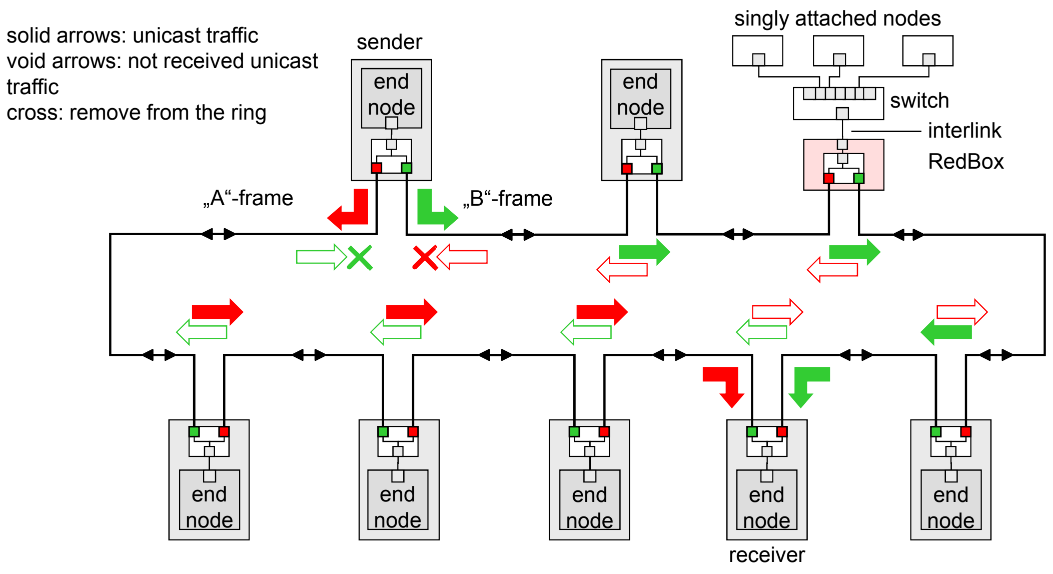

solid arrows: unicast traffic
void arrows: not received unicast traffic
cross: remove from the ring
sender
end node
„A"-frame
„B"-frame
singly attached nodes
switch
interlink
RedBox
receiver

Figure 2.1 HSR with unicast traffic: the solid arrows stand for the unicast traffic, the void arrows stand for not received unicast traffic, the cross stands for the traffic removed from the ring [3].

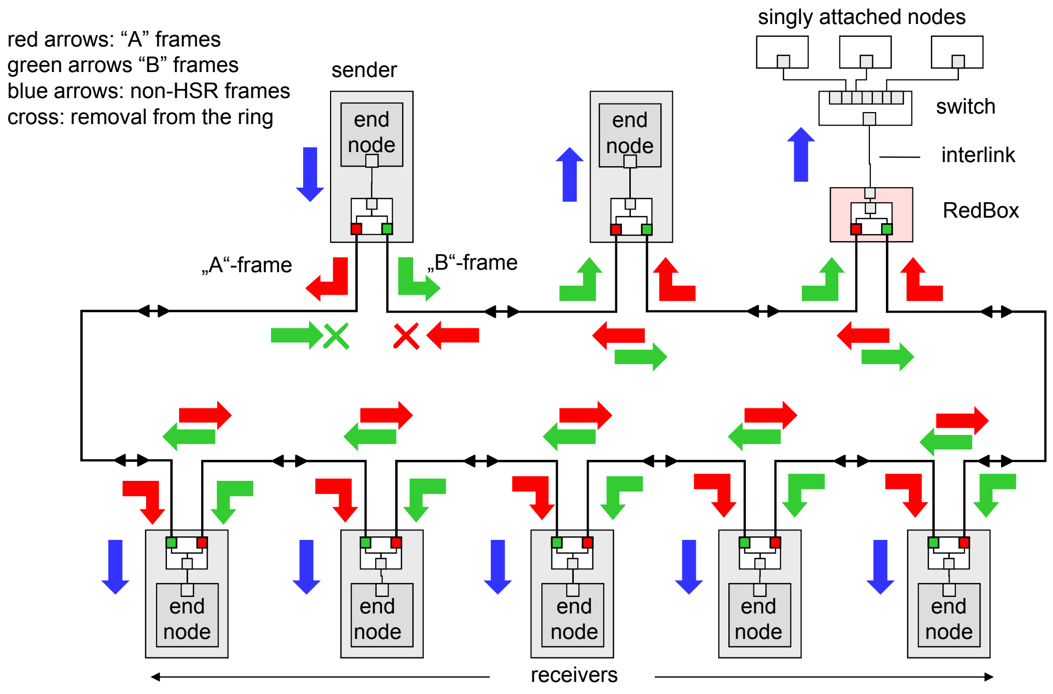


Figure 2.2 HSR with multicast traffic: the solid arrows stand for the multicast traffic, the void arrows stand for not received multicast traffic, the cross stands for the traffic removed from the ring [3].

2.2 The Network Management

A node has the same MAC address on both ports, and only one set of IP addresses assigned to that address. This makes redundancy transparent to the upper layers and therefore it is a layer 2 redundancy. This configuration allows the Address Resolution Protocol (ARP) to work the same as with a Singly Attached Node (SAN). TCP/IP traffic is not aware of the layer2 redundancy, but it is designed to deal with duplicates.

2.3 The Communication Rules of the Switch End Node in HSR

The communication rules are defined in the IEC 62439-3 standard [4].

2.3.1 Sending

For each frame to send on behalf of the higher protocol layers, a sending node (e.g. “sender” in Figure 2.1) detects which kind of traffic (HSR or non-HSR) it generates. This decision is application-dependent; it can for instance be based on the protocol type or a priority field. By default, all traffic is HSR.

Based on that decision, the node shall:

1) for the HSR traffic (if the node is attached to the ring): send two frames tagged as HSR, one over each ring port, called “pair”, otherwise

2) for a non-HSR traffic (if the node is not attached to the ring): send the frame unmodified to the switching element, which will treat it according to its bridging protocol (e.g. send only over the non-blocking port).

2.3.2 Receiving

A receiving node (e.g. "receiver" in Figure 2.1) detects the type of traffic of the received frame based on the HSR tagging and shall:

1) for a non-HSR frame(if the node is not attached to the ring): pass it unchanged to its higher protocol layers, otherwise
2) for an HSR frame(if the node is attached to the ring): remove the HSR tagging and pass the modified frame to its higher protocol layer, if this is the first frame of a pair, otherwise
3) discard the duplicate if this is the second frame of a pair.

2.3.3 Forwarding

A node that receives a valid frame over one ring port shall:

1) If it identifies this frame as a non-HSR frame, handle it according to the rules of its bridging protocol, otherwise:
2) If it identifies the frame as HSR frame, it shall forward it without modification over its associated ring port, except that it shall discard it:
    a) if it identifies the frame as an HSR frame that it already sent in the same direction, which is usually the case for multicast frames (solid arrows in Figure 2.2) but also for unicast HSR-frames without a receiver (void arrows in Figure 2.1);
    b) if the node is the sender of this node, as shown in Figure 2.1; this condition is enabled by default and can be disabled for debugging purpose.
    c) if the associated ring port of the node is not operating or its link not active. If a previously connected port is not connected to the network for a time longer than 1 s, a node shall purge the port's buffer so that it cannot send an obsolete frame, and only allow buffering when the port is reconnected.

These rules remove circulating HSR frames and open the ring, in the same way as an RSTP or similar protocol. It applies to frames originally sourced by the node and to frames circulating in case a device is removed after having sent a frame, and the ring is closed again, for instance by a mechanical bridging device or when a Singly Attached Node (SAN) is removed.

The arriving time difference between two frames of a pair depends on the relative position of the receiving node and the sending node. Assuming a worst case in which each node in the ring is transmitting at the same time its own frame with the largest size of 1536 octets, each node could introduce 125 us of delay at 100 Mbit/s. With 50 nodes, in case of uncast traffic, the time skew may exceed 6 ms, so there is possibility that the situation described above exists.

2.3.4 Cut-through

Nodes in HSR should work in the cut-through mode to reduce the forwarding delay. After the destination address, source address and sequence number have been received and the frame is confirmed as not received or not sent before, the node begins forwarding the frame over the other line. The cut-through operation is not applied to the receiving port to the host. The frame passed to the host is always completely received first. Only good frames are passed to the host.

2.3.5 Bad Frame Handling

Special care is needed to handle the bad frames when nodes in the ring are operating in the cut-through mode. If a frame is asserted as bad frame before the cut-though operation is performed, it is simply dropped. If a frame is detected as a bad frame after the cut-through operation is performed, a garbling sequence is appended at the end of the frame and the source address of this frame will be registered by the node. If a frame with the same source address is received again on the same line, no cut-through is performed, the frame is sent only after it is completely received and verified to be a good frame. If the frame is a good frame, the entry of this source address is cleared. Next time a frame with the same source address is received, the frame is sent in cut-through mode again.

A node in HSR should be able to detect the garbling sequence appended at the end of the frame. A garbling sequence tells a node that the frame is already been registered by another node as a bad frame, this node does not need to register this bad frame again. By doing so, only the first node which received the bad frame performs store-forward on the frames with same source address, other nodes still operate in cut-through when a frame with the same source address is received. This can reduce the transmission time in the ring of a good frame with the same source address as the bad fame. A bad frame appended with a garbling sequence will be passed around the ring until it is discarded by the node which sends it or is rejected as a circular frame.

2.4 Frame Format for HSR

A HSR frame is identified uniquely by their source MAC address, destination MAC address and the HSR Tag. The frame format is shown in Figure 2.3.

The HSR tag is placed at the beginning of the frame to allow early identification of frames for cut-through operation. After the destination address, the source address and the sequence number are received, the frame is uniquely identified.

The HSR tag is announced by the dedicated Ethertype = 0x88FB, which is the same as IEC 62439-3's Ethertype. If the frame carries a tagging according to 802.1Q, it shall be inserted before the HSR tagging.

The 4 most significant bits of the16-bits HSR tag distinguish a PRP management frame from a HSR management frame or a HSR payload.

a) 4-bit path identifier which can be a ring identifier or indicate a PRP supervision frame

b) 12 bit frame size (LSDU_size)

c) 16-bit sequence number (SequenceNr)

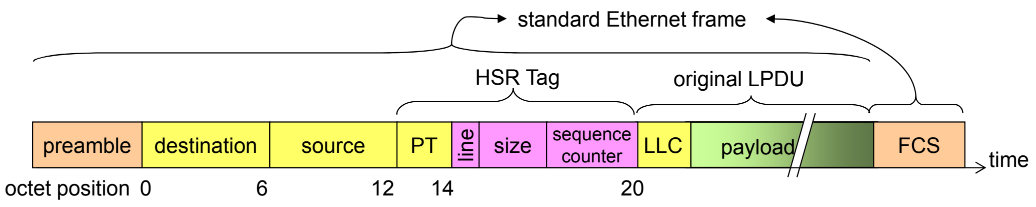


Figure 2.3 The frame format of HSR [4]

The sequence number is inherited from PRP, where they are used to discard the duplicated frames when receiving. The concept of "Duplicate Discard" in PRP will be explained later in section 3.1, and the reason why it cannot be wholly applied in HSR is also given there.

Because of the insertion of the HSR tag, the length of the frame may exceed the maximum length of 1522 octets allowed by the IEEE 802.1 D standard. But since the traffic in the ring is private, the modification can be done in the switch element to adapt the exceeded frame length and this will have no influence on the Ethernet traffic outside the ring.

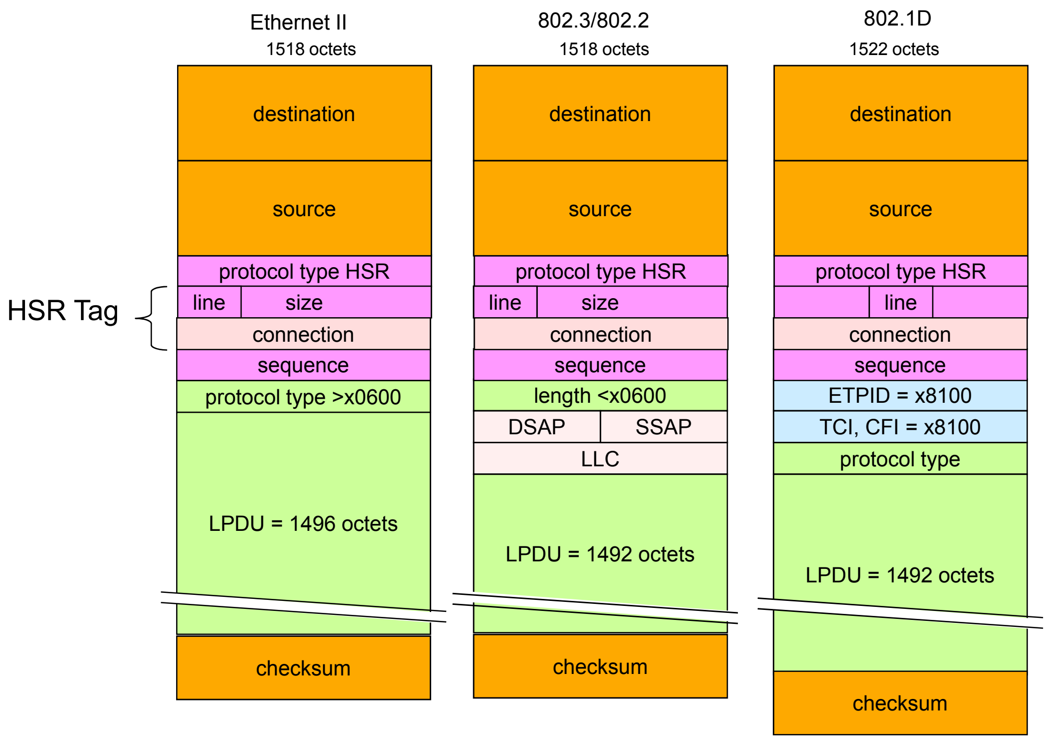


Figure 2.4 Frame format in different Ethernet standard after insertion of the HSR tag. The additional six bytes could generate oversize frames of more than 1522 octets [7]

2.5 The Node Structure and Operation in HSR

The structure of the node in HSR is shown in Figure 2.5.

When sending, the LRE duplicates each frame and send the pair of the frame over port A and port B (1, 2).

When forwarding, the switching logic relay frames from one port over the other port (3, 4), except it is the frame it already forwarded or it is the sender of this frame.

When receiving, the LRE receives both frames, keeps the first frame and discards the duplicate (7).

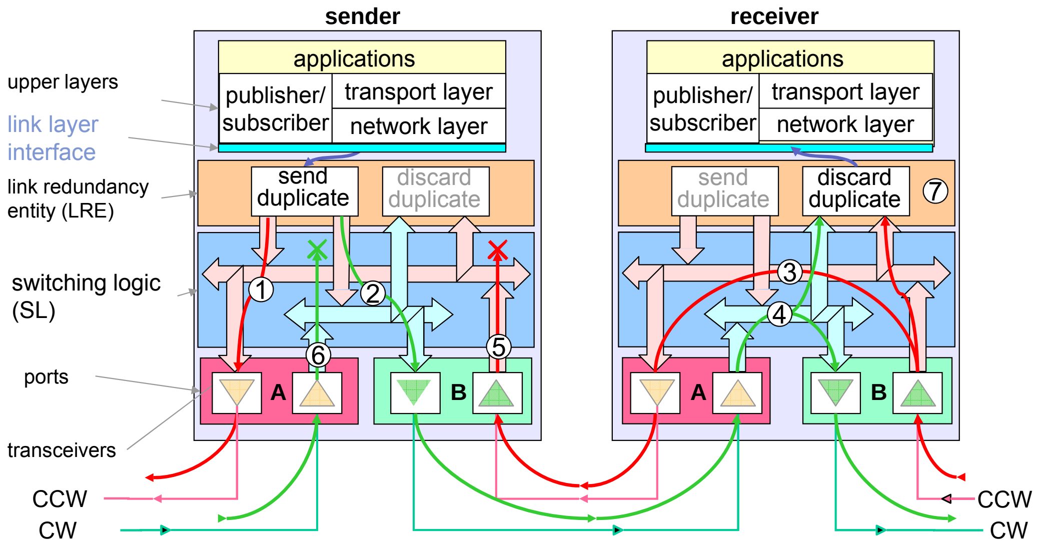


Figure 2.5 The node structure in HSR [3]

2.6 Duplicate Handling

Duplicate Handling is an important issue in HSR. The duplicate handling can work in two modes:

a) Duplicate Accept, in which the sender LRE uses the original frames and the receiver LRE forwards both frames it receives to its upper protocol layers.

b) Duplicate Discard, in which the sender LRE appends a Redundancy Control Trailer to both frames it sends and the receiver LRE uses that Redundancy Control Trailer to send only the first frame of a pair to its upper layers and filter out duplicates.

It is advantageous to discard duplicates already at the link layer. It is because not all the protocols in the upper layers can deal with duplicates. From the view of costs, the processor has twice as many interrupt requests as when only one ring exists. To offload the application processor, the LRE can perform “Duplicate Discard”, which should be realized in hardware.

In PRP, the Sequence Number, which is located in the HSR frame format shown in Figure 2.4.1, is used to drop the duplicates (recall that HSR is a modified application of PRP). Each time an LRE sends a frame to a particular destination it increases the sequence number corresponding to that destination and sends both frames over both LANs.

The algorithm used for rejecting the duplicates is the “Drop Window” algorithm. Briefly speaking, it builds a window at each line A, B. If the received frame at one line falls into the window of the other line, the frame will be dropped. This is shown in Figure 2.6.

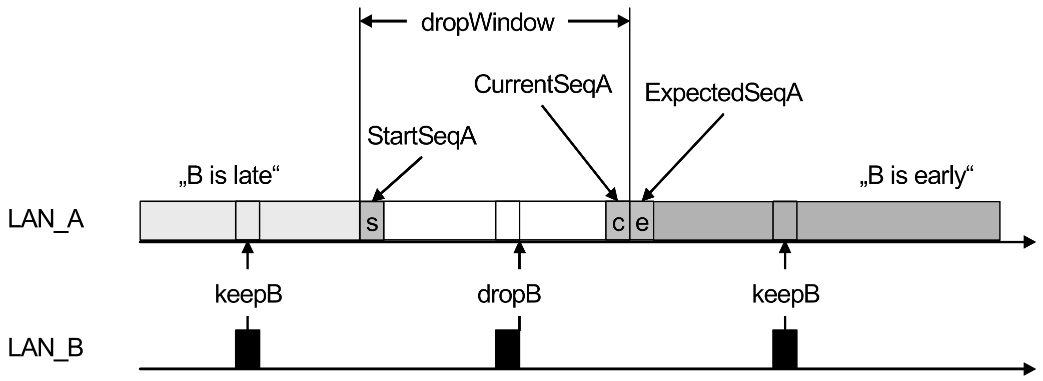


Figure 2.6 The drop window algorithm [4]

2.1 The Problem with the Drop Window Algorithm in HSR

The drop window algorithm works well in PRP because each node in PRP only sends or receives frames, it never relays a frame. In HSR, each node also forward frames except receiving and sending. As stated before, the situation may happen that a multicast frame losing the sender or a unicast frame losing both the receiver and the sender will circulate in the ring. There must be a mechanism to remove such circular frame from the ring. Since it is not possible to use the “Drop Window” algorithm for such purpose, it is preferred to find another algorithm which can not only be able to reject the duplicated frames but also the circular frames.

Another reason why the Drop Window algorithm is not used here is that the implementation of the “Drop Window” algorithm in hardware is less efficient than the lookup table method which will be introduced later in this Thesis [8]. For these reasons, new solutions are found and proposed in the next chapter.

3 The Proposed Algorithms for Rejecting the Duplicated and Circular Frames

In this chapter, three algorithms are proposed for finding an efficient way to reject the duplicated and circular frames. Their operation principle is explained, the advantages and disadvantages of each algorithm are discussed in respect to the collision possibilities, the searching time and the hardware implementation complexity.

3.1 General Principle

Instead of the “Drop Window” algorithm applied in PRP, algorithms based on look-up tables are applied. The basic idea is:

Each frame in HSR is uniquely identified by its destination address, source address and the sequence number (recall that the sequence number for each destination address is increased by one when a frame is sent to this destination address). Therefore the destination address, the source address and the sequence number can be stored as an entry in the table to show that this frame has already been received.

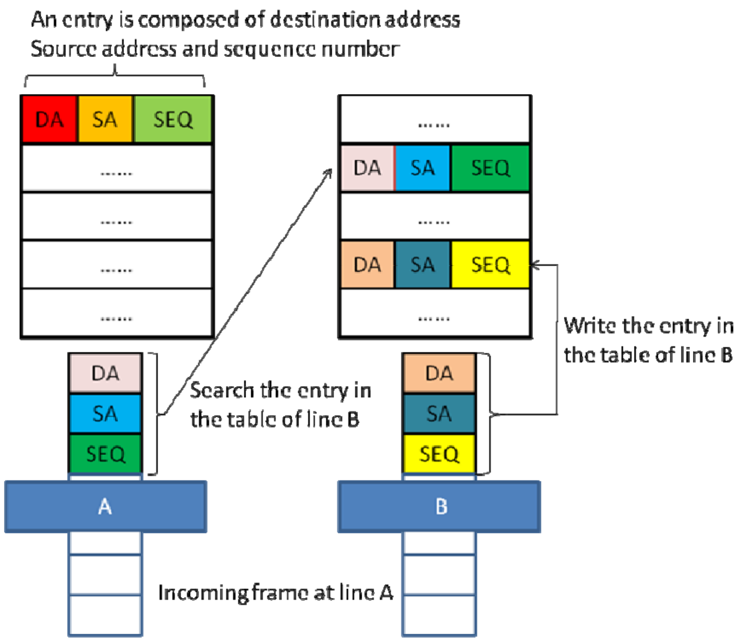


Figure 3.1 The searching and writing operation in the table for rejecting the duplicated frames

The working principle is stated as below***:***

1. **For rejecting the duplicated frames** (the frames can be unicast or multicast frames)

   There is one table for each line to store the entry of frames which have been successfully received on this line (means no error occur during receiving).

   During receiving at one line, after the destination address, the source address and the sequence number of the frame have been received, the frame is searched first in the table of the other line. If it is found in the table of the other line, it will be discarded and the receiving process is aborted otherwise

If the entry of this frame is not found in the table, the receiving process will continue. If the frame turns out to be a good frame at the end of the receiving, the entry of the frame will be stored in the table of the line at which it is received.

If frames of a pair are received on both lines when there is no entry of either frame in the table (for example two frames are received on both lines at the same time), the Window Function described in Section 5.4.3 is applied.

The case of rejecting the duplicated frames is illustrated in Figure 3.1.

2. **For rejecting the circular frames** (the frame can be any traffic type of frames)

   To reject the circular frames, there is one table on each line to store the entry of frames which have been successfully received on this line (means no error occurs during receiving).

   During receiving at one line, after the destination address, the source address and the sequence number of the frame have been received, the frame is searched first in the table of this line. If it is found in the table of the this line, it will be discarded and the receiving process is aborted otherwise

   If the entry of this frame is not found in the table, the receiving process will continue. If the frame turns out to be a good frame at the end of the receiving, the entry of the frame is stored in the table of the line at which it is received.

The case of rejecting the circular frames is similar with the case of rejecting the duplicated frame except that the entry is searched and written in the same table of the line on which the frame is received but not in the table of the other line.

The entry of a frame is written in the table only after the frame is completely received and verified to be a good frame. If a frame turns out to be a bad frame it will not be registered in the table, so when the other frame of this pair is received on the other line, the entry will be not be found in the table and the frame is therefore received. The same reason applies for the circular frames.

The operation of rejecting the duplicated frame stated above implies that the table can be read and written at the same time. For this purpose, the Dual Port RAM should be used.

The general principle is clear by now, the issue left is to find an efficient data structure to implement such table. There are several methods to lookup an item in table. The conventional methods are like the binary search algorithm and hash table. Binary search is based on the sorted table. But here the sorting is difficult to define (e.g. what criteria should be used to sort the entries) and the hardware is difficult to implement. There are different ways to implement a hash table, some methods are suitable in our situation, some not. The discussion of the hash table will be conducted in later sections. In the following sections, three algorithms are proposed to implement the lookup table described above.

3.2 Algorithm 1: Circular Buffer

The Circular Buffer method is introduced by the Zurich University of Applied Sciences (ZHAW, Winterthur) to implement the rejection of the duplicated frames in the Redundancy Box [7].

In this method, the entries are simply registered in the table one after another. The write pointer moves downwards by one after an entry has been registered. In this way, the entry of higher position is older. After the write pointer reaches the end of the table, it will go back to the beginning of the table and start over. This is shown in Figure 3.2. In this way, the older entry is automatically replaced by the new entries.

In this algorithm, the length of the table has to be selected according to the receiving delay of the frame of a pair between the two lines [8]. Take a ring with 6 nodes for example (Figure 3.3). If node 1 send a unicast frame to node 2, one frame of the pair goes the clockwise direction and arrives at node 2; the other frame of the pair travels the counter-clockwise and must goes through 4 nodes until arrives at node 2. Assuming that the length of all the frames in the ring is the same, node 1 could receive 4 other frames, until the frame sent by node 1 arrives at node 2, which means that 4 more entries are registered in the table during this time. Therefore the table should have places for at least 5 entries so that the entry of the first frame of a pair sent by node 1 is not overwritten by other entries before the second frame of the pair arrives at node 2.

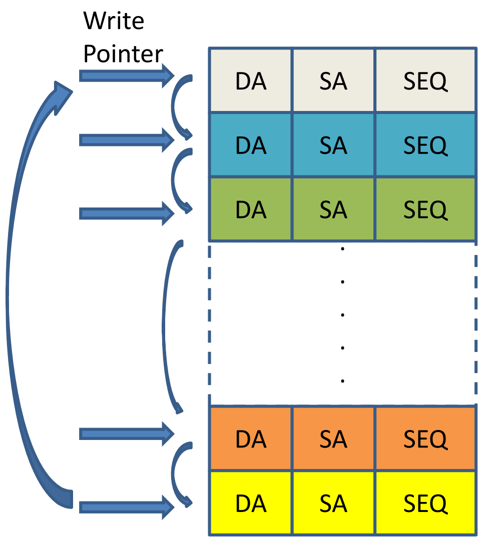


Figure 3.2 The operation principle of a circular buffer

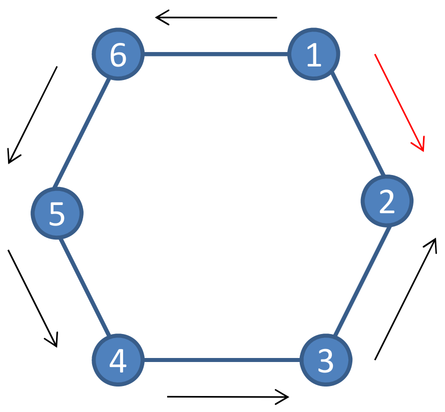


Figure 3.3 The receiving delay of a pair unicast frame in a ring

This is just a simple example to illustrate that the length of the table is related to the receiving delay between the two lines, the reality is more complex.

Because the entries are simply registered in the table one after another, the hardware implementation is simple. If the table is large enough, frames' entry can be found in the table before their entries are overwritten. But one should go through all the entries to verify that the received frame is not in the table. In a ring with more nodes, the search will cost more time and therefore cause longer delay when forwarding. If cut-through operation is required, this method is not suitable.

3.3 Algorithm 2: Hash Table with Open Addressing and Aging

3.3.1 Why Open Addressing

To increase the search efficiency, hash table is used. There are many algorithms to implement a hash table and resolve the collisions. The conventional ones are like chaining, open addressing and so on.

Chained hash tables have advantages over open addressed hash tables in that the removal operation is simple and resizing the table can be postponed for a much longer time because performance degrades more gracefully even when every slot is used. Indeed, many chaining hash tables may not require resizing at all since performance degradation is linear as the table fills. For example, a chaining hash table containing twice its recommended capacity of data would only be about twice as slow on average as the same table at its recommended capacity [9].

But chained hash tables need to allocate memory for adding elements to the linked list. The overhead required by the operation of allocating new memory will cause more delay and therefore is not preferred in real-time system. Furthermore, a memory allocator is more difficult to implement than the open addressing.

Compared with chaining, open addressing is [9]:

More space-efficient since it doesn't need to store any pointers or allocate any additional space outside the hash table, this makes it more suitable to be implemented in memory constrained devices like FPGA.

The Insertion of elements avoid the time overhead of memory allocation, and can even be implemented in the absence of a memory allocator.

Because it uses internal storage, open addressing avoids the extra indirection of the external storage required by chaining. It also has better locality of reference, particularly with linear probing. With small record sizes, these factors can yield better performance than chaining, particularly for lookups.

At last, they can be easier to serialize, because they don't use pointers.

3.3.2 Open Addressing Algorithm and its Constraints

The open addressing algorithm can be briefly explained as follows [10]:

A hash table *T* is an array *T[0,….m-1]*, m is a positive integer called the size of the table.

If we have a sequence of hash functions $< h_0, h_1, h_{2,} ......, h_{m-1} >$, such that for any item *x*, the *probe sequence* $< h_0(x), h_1(x), h_{2,}(x)......, h_{m-1}(x) >$ is a permutation of $< 0,1,2......, m-1 >$. In other words, different hash functions in the sequence always map x to different locations in the table.

x is searched by using the following algorithm, which returns the array index i if T[i] = x, “absent” if x is not in the table but there is an empty slot[11], and “full” if x is not in the table and there no empty slots. This is shown in figure below.

```
for $i = 1$ to $m-1$
  if $T[h_i(x)] = x$
    return $h_i(x)$
  else if $T[h_i(x)] = \phi$
    return Absent
return Full
```

Under the ***strong uniformity assumption***, that is for any key $k \in U$

$$\Pr\{(h_i(k), h_j(k)) = (i, j)\} = \frac{1}{m(m-1)} \quad (3.1)$$

the expected lookup time is calculated as

$$E(T(m,n)) \leq 1/(1-\alpha) \quad (3.2)$$

Here $\alpha = n/m$ is called the load factor of the table and n is the number of current element in the hash table.

Here we can see, with the increasing of the load factor α, the expected lookup time increases dramatically, if the table is almost full, the lookup will take increased to certain degree that one has to go through the entire table to found out whether an element is in the table or not.

Deleting an element in the table is also not simple. We cannot simply clear out the slot in the table, because we may need to know that *T[h(x)]* is occupied in order to find other items. Instead simply deleting a slot, we should mark it as a wasted slot. But a sufficiently long operation of insertions and deletions could eventually fill the table with marks, leaving little room for any real data and causing searches to take linear time. Therefore the size of the table should be increased when the load factor reaches a threshold value. The time costs of such operation can be very expensive. This can be shown by the amortized analysis but not an issue in this thesis.

On the other hand, the implementation of a hash table in real-time system cannot afford the time cost of enlarging the hash table all at once, because it may interrupt time-critical operations. And the device like FPGA with a constrained memory may not allow the increasing of the table size [9].

Because it is not desirable to go through the entire table until to find out whether the element is in the table and it may not possible to resize the table, we have to find another way to implement the table. A commonly used technique “Aging” is applied here.

3.3.3 Open Addressing With Aging

Figure 3.4 shows the entry structure in open addressing with aging. A one-byte “Aging Tag” (AT) is added at the beginning of each entry [12]. The meaning of the AT is:

x“00”	An empty bin
x”FF”	The maximum bin

At the beginning, the value of the AT is *x"00".* Every time an entry is written to a bin, the AT of this bin is assigned to the value of *x"FF".* An aging process runs in the background with certain time distance, which is determined by the scale of the network. The value of the AT is subtracted by one until the AT becomes *x"00"* again. Therefore, the smaller the value of the AT is older the entry.

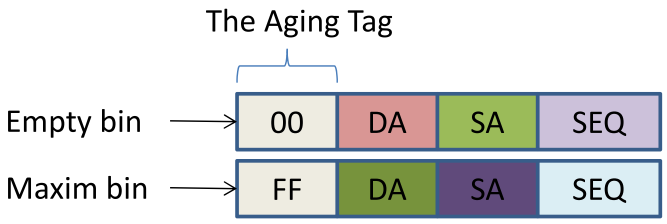


Figure 3.4 The entry format with "Aging Tag"

The searching algorithm is described as following

```
$for\ i = 1\ to\ \max\_bin$
  $if\ \ T[h_i(x)] = x$
    $T[h_i(x)].AT = x"00"$
    $return\ h_i(x)$
*return Absent*
```

This algorithm states if an entry is found in the table, the corresponding place will be cleared by writing AT of the bin with x"00". The ***max_bin*** defines the maximum probe time, after the ***max_bin*** is reached, the searching stops and return ***Absent*** to indicate that the entry is not found in the table.

The writing entry algorithm is a little more complicated, which is described as follows

```
$for\ i = 1\ to\ \max\_bin$
  $if\ \ T[h_i(x)].AT = \phi$
    $T[h_i(x)] = x$
    $T[h_1(x)].AT = x"FF"$
    $return$
  $else$
    Oldest_Pos = $T[h_i(x)].AT \le Oldest\,?\,h_i(x) : Oldest\_Pos$
    Oldest = $T[h_i(x)].AT \le Oldest\,?\,T[h_i(x)].AT : Oldest$
$return$ Oldest_Pos
$T[Oldest\_Pos] = x$
```

The figure above states, when writing an entry in the table:

If an empty bin is found, the entry will be written at this bin. If the bin is not empty, the AT of this bin will be stored in the variable ***Oldest***, and the position of this pin is stored in the variable ***Oldest_Pos.***

Every time when the bin is not empty, if the AT of this bin is older than the ***Oldest***, the ***Oldest*** will be replaced by the AT of this bin and the ***Oldest_Pos*** is replaced by the position of this bin.

If no empty bins are found in the end, the entry will be written at the place of the ***Oldest_Pos*** bin.

The algorithm described above is illustrated in Figure 3.5.

The advantage of introducing the ***max_bin*** is that it constraints the searching and writing time in the worst case to ***max_bin*** steps, so that the decision can be made in much shorter time than going through the entire table, which is the case in the conventional open addressing hash table.

The aging mechanism is necessary for deciding the bin to write when all the bins are filled with entries. In other words, the resizing of the table is replaced by just writing at the bin with the eldest AT. This takes place in the same store area, and do not require additional memories. Another use of AT, although dose not likely to occur under heavy traffic, is to time out the entries which are not received at the other line for a long time.

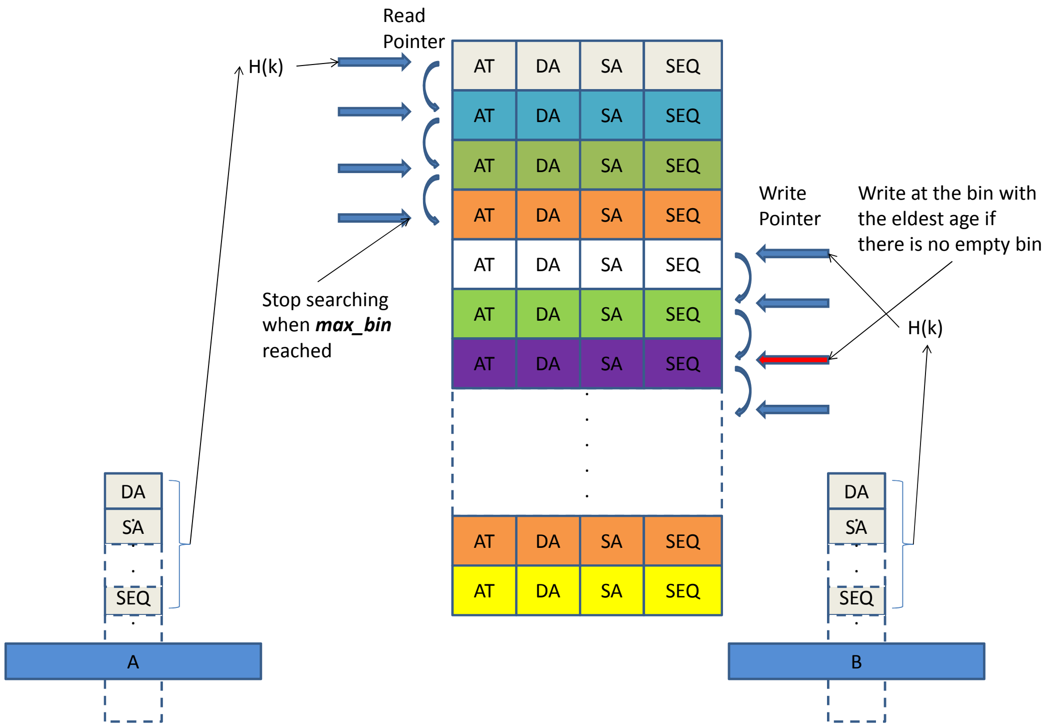

Read Pointer
H(k)
AT
DA
SA
SEQ
Stop searching when ***max_bin*** reached
Write Pointer
Write at the bin with the eldest age if there is no empty bin
H(k)
DA
SA
SEQ
A
DA
SA
SEQ
B

Figure 3.5 The search and write operation in modified hash table with open addressing

The time distance of subtracting the aging tag should consider the scale of the network. Take a look at the simple example of unicast traffic again, shown in Figure 3.6.

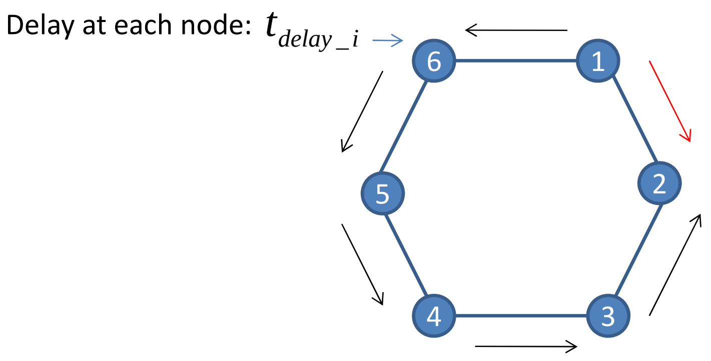


Figure 3.6 The receiving delay between the two lines of a unicast frame

In the ring having nodes of number *n*, the transmit delay (the time delay between completely receiving a frame and completely sending it out) is t_{delay} at each node.

When for example a unicast frame of a pair sent by node 1 arrives at node 2, its entry is registered in the table. The time delay of the arriving of the second frame of this pair on the other line is calculated as

$$t_{delay\_total} = \sum_{i=1}^{n-2} t_{delay\_i} \qquad (3.3)$$

Therefore

$$t_{subtract\_\text{int}eval} \geq \frac{t_{delay\_total}}{256} \quad (3.4)$$

Recall that the maximum of AT is x”FF”.

Therefore the time distance of aging cannot be so frequent that the entry times out before the t_{delay_total} is reached. In reality the traffic is more complex. One should configure the time distance of subtracting the aging tag enough long to ensure that the case described above does not happen.

3.3.4 Choose the Random Probe Sequence

In practice the truly random probe sequence $< h_0(x), h_1(x), h_{2,}(x)......, h_{m-1}(x) >$ is difficult to find, so one of the heuristics can be used

1) Linear probing

Using a single hash function h(x), and define

$$h_i(x) = (h(x) + i) \bmod m \quad (3.5)$$

The operation in this equation is simple, but it suffers from a phenomenon known as primary clustering, in which large chains of occupied positions begin to develop as the table becomes more and more full. This results in excessive probing

2) Quadratic probing

Also using a single hash function h(x), and define

$$h_i(x) = (h(x) + i^2) \bmod m \quad (3.6)$$

Unfortunately, for certain values of m, the sequence of hash values $< h_i(x) >$ does not hit every possible slot in the table; we can avoid this problem by making m a prime number. Although quadratic probing does not suffer from the same clumping problems as linear probing, it does have a weaker clustering problem known as secondary clustering: If two items have the same initial hash value, their entire probe sequences will be the same.

3) Linear Double Probing

We use two hash functions h(x) and g(x), and define

$$h_i(x) = (h(x) + ig(x)) \bmod m \qquad (3.7)$$

To guarantee that this can hit every slot in the table, the stride function g(x) and the table size *m* must be relatively prime. This can be guaranteed by making m prime. The key advantage of linear double hashing over linear probing is that it is possible for both h(k) and g(k) to vary with k. Thus, in $h_i(x)$ the probe sequence depends on k through both h(k) and g(k), and is linear in h(k) and g(k). A widely used member hash function proposed by Knuth is [13]

$$\begin{aligned} h(k) &= k \bmod m \\ g(k) &= k \bmod (m-2) \end{aligned} \quad (3.8)$$

3.3.5 Randomization of the Un-uniform Distributed Keys

The ordinary hash function h(x) used in equation (3.4) and equation (3.5) has a dramatic impact on the performance of linear probing and quadratic probing. A common choice like:

$$h(x) = x \bmod m \quad (3.9)$$

This performs only well when the key *x* is uniformly distributed, so that this ordinary function can generate uniformly distributed sequences. When the key *x* diverges from the uniform distribution, the performance of linear probing and quadratic probing degrades dramatically[14].

This is unfortunately our case. Assume a ring has 10 nodes, the variation of the key which is the concatenation of the destination address, source address and the sequence number is limited. This is because:

- First, the first 3 bytes of the MAC address is the manufacturers Organizational Unique Identifier (OUI). In a ring, the manufacturers are not likely to exceed 20. So the variation of this part is very limited
- Second, if the traffic is multicast, the destination is always the multicast address. The most traffic in the ring is multicast traffic.
- Third, although the sequence number varies most frequently (from 0 to 65535), it varies only at the end of a key.

Therefore the key in our case is far from uniformly distributed.

Two approaches are commonly used to address this problem. First, one can apply a randomizing transformation to the keys prior to supplying them to Equation 3.9. This is actually a natural step to take in many applications. For example, consider compiler symbol tables, where strings must be converted into numeric key values in order to “hash” them into the table. One such popular algorithm, called ***hashPJW()***[15]**,** takes a string as input, and output an integer in the range $[0, 2^{32} - 1]$. The transformation performed by ***hashPJW()*** tends to do a good job of producing numbers that appear uniformly over certain interval, even when the strings being hashed are very similar.

A second approach involves using a more complicated ordinary hash function h(x) so that the initial probe into the hash table is more random. In addition, by randomizing the choice of h(x) itself we can guarantee good average-case performance (relative to any fixed ordinary hash function) through the use of universal hashing[16].

A set H of hash function is universal if it satisfies the following property:

> For all pairs of distinct keys $x \neq y$, if a hash function *h* is chosen uniformly random from the hash function family set H, then

$$\Pr[h(x) = h(y)] \leq 1/|V| \quad (3.10)$$

|V| denote size of V, the number of possible hashed values.

A good example of a universal hash function is

$$h(k) = ((ak + b) \bmod p) \bmod m \quad (3.11)$$

Here $a \in \mathrm{Z}_p^*, b \in Z_p$, Z_p^* denotes the set {1, 2, 3, ……, p - 1} and p is a prime number large enough so that every possible $k \in U$ is in $[0, p-1]$. Thus, for fixed p and m, there are p(p – 1) different hash functions in this family.

Although the universal hash functions provide a good performance, the multiplication operation in the FPGA is very expensive especially when the key *k* is more than 32 bits. There are ways like addition tree can work around this problem, but again it sacrifices time and causes longer delay. Besides, the mod operation of a prime is difficult to realize. Only the mod operation of the order of 2 is able to be synthesized. Therefore, the randomizing transformation method is chosen.

3.3.6 The Randomness of Double Hashing

If the goal is to minimize the total number of collisions and thus memory accesses, then from a probabilistic perspective, the ideal case for open address hashing is uniform hashing[17][18].

A uniform hash function always produces probe sequences of length m (in the table space), with each of the $m!$ possible probe sequences being equally likely. The obvious way of implementing a uniform hash function involves the generation of independent random permutations over the table space for each key $k \in U$. However, the computational costs associated with this strategy make it completely impractical.

Through probabilistic analysis, the function described by Equation (3.7) offers a reasonable approximation to uniform hashing[19][20]. This conclusion is based on the ***strong uniformity assumption*** shown in Equation (3.1). Thus, these results only hold under the assumption that the keys will produce hash value pairs that are jointly uniformly distributed over the table space. This strong assumption has requirements both on the initial data distribution and the choice of h(k), g(k). As the most data set are far from uniform (which is indeed our case), and the popular candidate for h(k), g(k), which is described previously, has to be considered poor choices to satisfy the Equation (3.1).

3.3.7 Memory Access Serialization

According the algorithm applied by the hash table with open addressing and aging. It can be seen that there are three processes that need both read and write access to the table:

1) The aging process, which runs in the background. It needs read the aging tag, and subtracts AT by one if it is not equal to zero.
2) The searching process, which read the entry in the table, and clear the entry by writing AT to x"00" if it is found.
3) The writing process, which first reads AT in the table and register the entry in the empty bin or the bin with the oldest AT if no bins are empty.

Since the table is located in the dual port memory, it is allowed that reading and writing happens at the same time. But if two reading or writing operations are to be executed at the same time, the two operations must be serialized. A collision between two writing operation in the table for rejecting duplicated frames can be show in Figure 3.7

For the table used for rejecting the circular frames, because the searching and writing process occurs at the same line and therefore naturally serialized (means the searching operation always come previous to the writing process), the serialization only have to be done between the aging process and the reading or writing process. Here the reading and writing process always have the higher priority than the aging process, because we want to make the decision as early as possible so that the cut-through can be performed earlier.

For the table used for rejecting the duplicated frames, the searching and writing is required by different lines, the sequence of their occurrence is not deterministic. Therefore the searching, writing and aging process should be serialized respectively. The searching process has the highest priority, the reason is same as stated above to enable earlier cut-through. Writing process has the second priority because it is not time critical like searching process. If during writing searching is required by other line, writing should be suspended and give the memory access control to the searching process. Finally the aging process has the least priority.

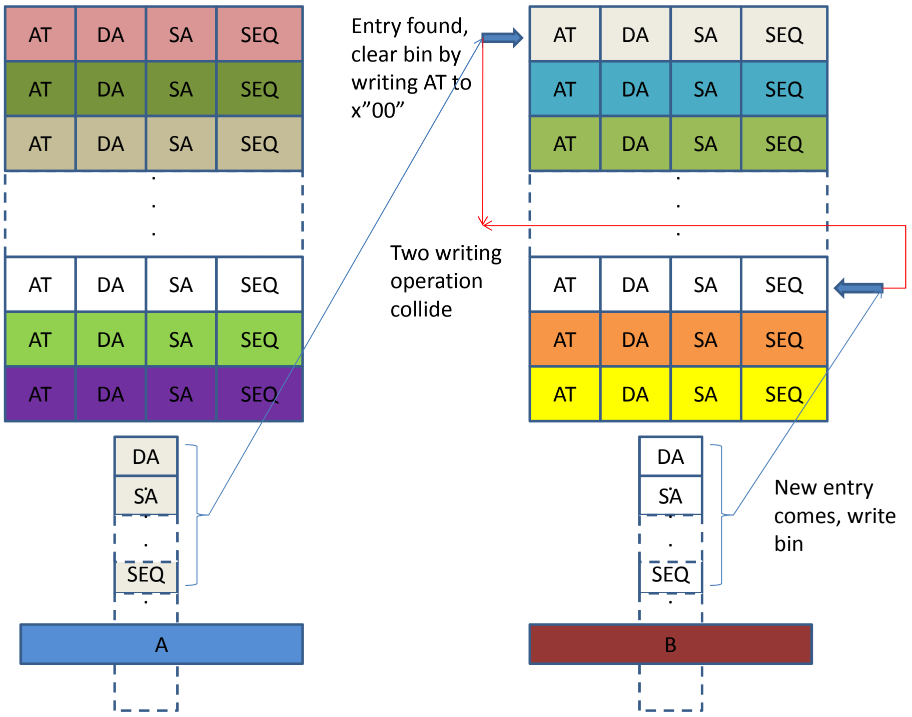
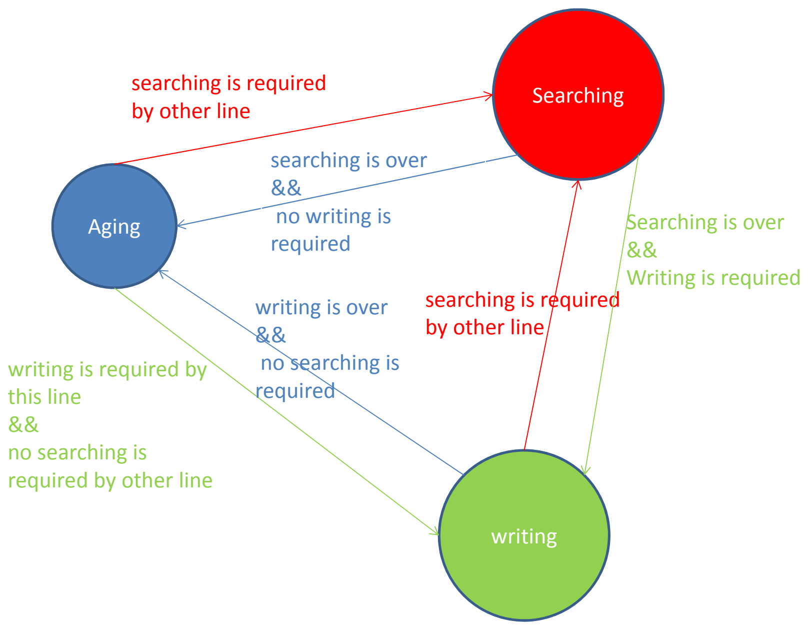

AT
DA
SA
SEQ
Entry found, clear bin by writing AT to x"00"
Two writing operation collide
New entry comes, write bin
A
B

Figure 3.7 Clearing bin operation collides with the writing entry operation

The time period from the sequence number to the end of the frame is plenty for the aging process, because the subtracting of the AT is only executed in a pre-configured time distance, but not at very instance. The memory access control transfer state machine is shown in Figure 3.8.



searching is required by other line
Searching
searching is over && no writing is required
Aging
Searching is over && Writing is required
searching is required by other line
writing is over && no searching is required
writing is required by this line && no searching is required by other line
writing

Figure 3.8 The memory access control transfer state machine of the hash table for rejecting the duplicated frames

3.4 Algorithm 3: Hashing Combined with Circular Buffer

In the previous section, the hash table with open addressing is modified by adding the ***max_bin*** parameter and the aging functionality to fix the worst case searching time and avoid resizing the whole table. But the aging functionality needs a process running on the background, and the serialization of the memory access among the aging, searching and writing process is also needed, these increase the complexity of the hardware implementation. To reduce the complexity of the implementation and still maintain a similar performance, the third algorithm is proposed.

The proposed hash table structure combined with circular buffer is illustrated in Figure 3.9. The table is divided into several regions, the hash value of the entry decided into which region the entry falls. In each region there are number of ***max_bin*** bins. The entry is written at the bin position.

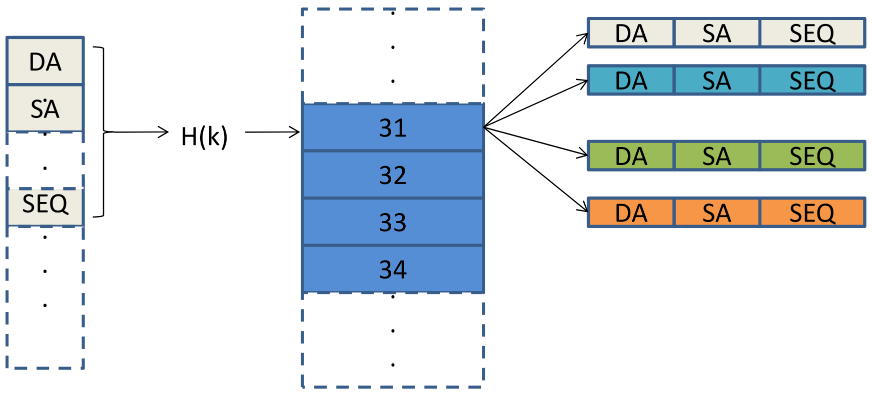


Figure 3.9 The structure of the hash table combined with circular buffer

The concept is similar to the chained hash table, except the element attached to each bucket is limited to the number given by ***max_bin*** parameter. If all the bins are taken, the entry is simply overwritten the eldest bin, and thus there is no need for memory allocator.

The operation principle is explained as follows:

- When searching an entry of a received frame:

  When an entry is received, after the destination address, the source address and the sequence number are read, the hash value is calculated. The calculated value is mapped to certain area of the table. This mapping can be for example a mod operation.

  The position of the read pointer in this area is the position after last searching in this area. If the entry is not found at the current position, the read pointer moves downwards by one. If entry is found, the read pointer stays at the next position to the position where the entry is found.

  When searching steps has reached the ***max_bin,*** searching stops and the read pointer go back to its last initial position.

- When writing an entry of a good frame

  Write the entry at the position of the write pointer and move the write pointer downwards by one.

  If the write pointer has reached the end of the area, it goes back to the beginning of this area

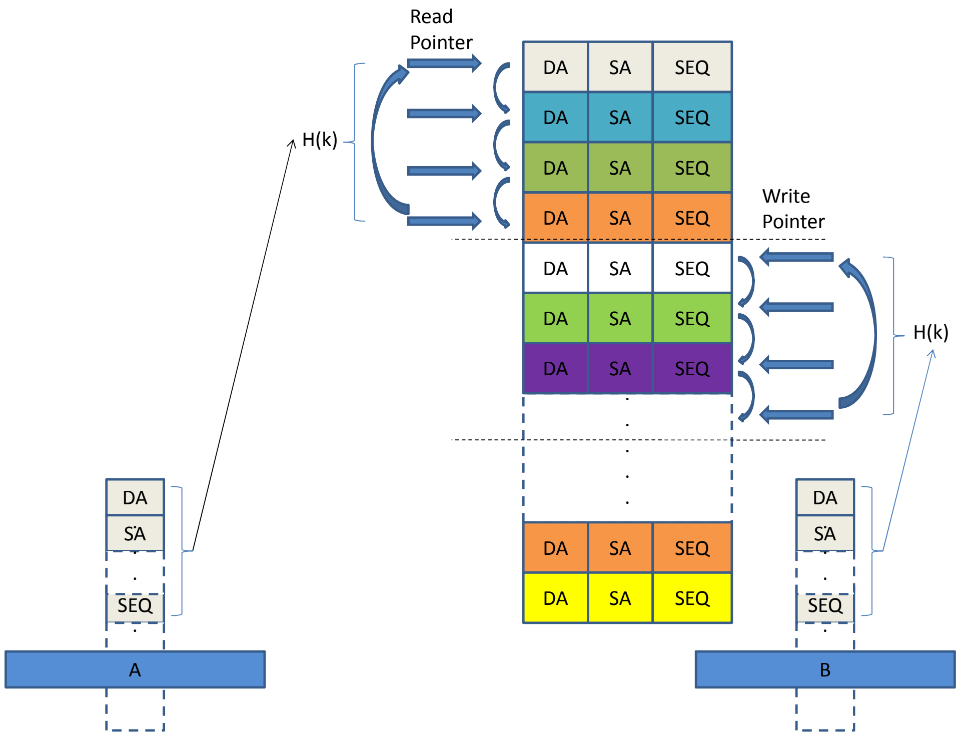


Figure 3.10 Read and write operation in the hash table combined with circular buffer for rejecting the duplicated frames

In this algorithm, the maximum searching step is also limited to ***max_bin*** times, and there is no aging mechanism being applied. Each area has its own read and write pointer. The writing operation iterates through the area, the oldest entry is simply overwritten. The writing entry process only does write operations, and the searching process only does read operation. Therefore there is no more than one process reading or writing at the same time and thus the memory access does not need to be serialized (Figure 3.10).

This simplifies the circuit dramatically compared with the hash table with open addressing and aging.

The map of an entry to certain area is done by the hash function

$$h(x) = x \bmod m \quad (3.10)$$

Again, the key x here is far from uniform distributed, so the function for randomization of the un-uniform distributed Keys introduced in Section 3.3.5 should be used here before the hash operation.

A drawback of this method is that it does not find an empty bin to write the new entry which is done in the hash table with aging algorithm. Instead it just overwrite the oldest entry in the area. So the probability that an unused entry is overwritten is higher than the hash table with aging.

3.5 Comparison of the Proposed Algorithms

In Table 3.1 the proposed algorithms are compared in respect to their collision probability, worst case searching time, implementation complexity.

Proposed algorithms	Collision Probability	Worst Case Searching time	Implementation Complexity
Circular Buffer	No collisions as long as the length of the table longer than frame arriving delay between the two lines	must go through the entire table	simplest
Hash Table With Open Addressing and Aging	Depending on the chosen probe sequence linear probe: primary clustering quadratic probe: secondary clustering double hashing: most unlikely to collide compared with two others With the aging functionality, the entry always search first an empty bin to write and then the oldest	limited by the ***max_bin*** parameter	Most complex because of the aging functionality and the memory access serialization
Hash Table Combined with Circular Buffer	The oldest entry in each area is simply overwritten, but it is possible that the overwritten entry is not used yet	limited by the ***max_bin*** parameter	Moderate

Table 3.1 Comparison of the proposed algorithms

3.6 Conclusion to the Proposed Solutions

In this chapter, three algorithms based on look up table are proposed to replace the drop window algorithm to reject the duplicated and circular frames in HSR.

The circular buffer method has no collision as long as the length of the table is larger than the frame arriving delay between the two lines, and it is simplest to implement. But to find out whether the entry is in the table or not, one should go through the entire table which causes long delay, this method is not suitable for realizing cut-through.

The hash table with opening address and aging functionality introduce the ***max_bin*** to limit the maximum searching time. Aging kicks out the entries which already times out and ensures to write the entry only in the empty bin or bin with the oldest age. This can somehow reduce the collision probability and avoid resizing operation. But the need for implementation of the aging function and serialization of the memory access makes the circuit more complex compared with circular buffer and hash table combined with circular buffer method.

The hash table combined with circular buffer is similar with the chained hash table except that the oldest entry is simply overwritten if all the bins attached to the bucket are taken. By doing so the memory allocator is saved. But one drawback of this method is that a not yet used oldest entry may be overwritten.

A last it should be pointed out that the key of the hash function in our case is far from uniform, the randomization of the key should be done previous to hashing.

In the next chapter, the simulation environment will be built. The performance of the proposed algorithms will be simulated in respect to their rejecting ratio to the duplicated and circular frames.

4 The Software Simulation of the Proposed Solutions

In this chapter, a simulation environment is set up to evaluate the proposed algorithms. The basic unit in the simulation is a node. A node can accomplish all the tasks of a real end node in HSR and simulate the timing behavior of sending or receiving process. Because the program is written in C++, the operations of the nodes in the ring must be parallelized to simulate the parallel process in real hardware. The configuration of the simulation is explained and the simulation result is presented and discussed. Finally the hash table combined with circular buffer is chosen as the algorithm to be implemented in FPGA.

4.1 The Setup of the Simulation Environment

In this section, the structure of a node, which is the basic unit in the simulation environment, is illustrated. The components in the node and their way of work are explained in more details. The way of parallelizing the operations of a node is explained.

4.1.1 The Node Class

The node is the basic unit in the simulation environment. A node unit represents an end node in the HSR network. The node class designed here can accomplish all the tasks of the link layer of an end node. The tasks of a node in HSR have already been introduced in chapter 2. The structure the node class is illustrated in Figure 4.1.

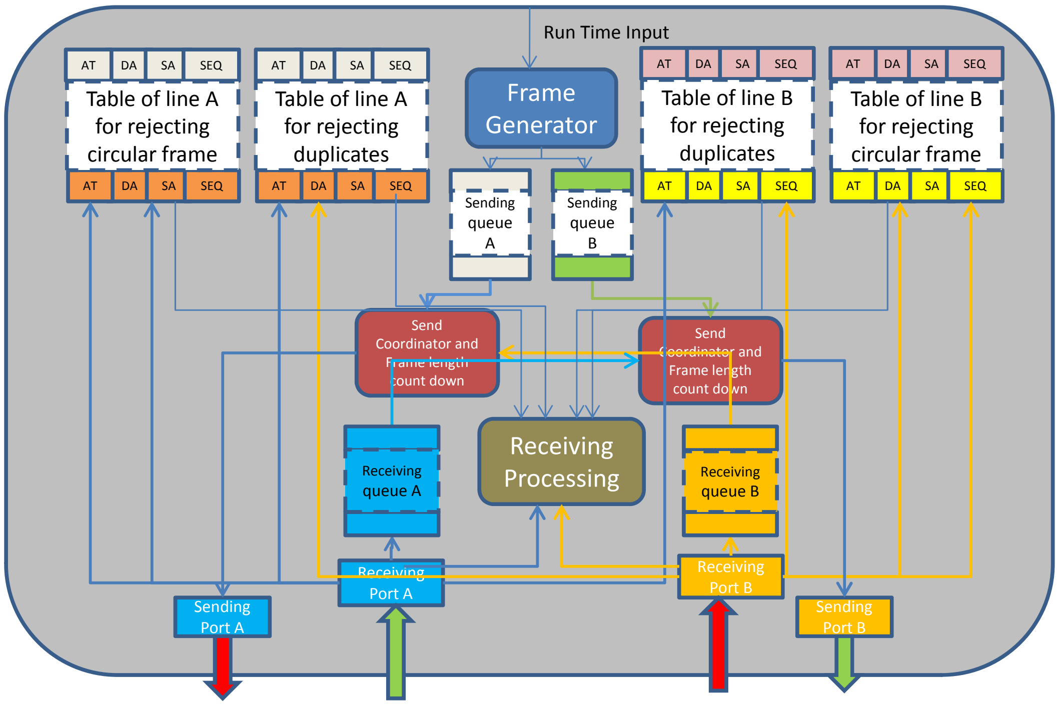

Run Time Input
AT DA SA SEQ
Table of line A for rejecting circular frame
Table of line A for rejecting duplicates
Frame Generator
Table of line B for rejecting duplicates
Table of line B for rejecting circular frame
Sending queue A
Sending queue B
Send Coordinator and Frame length count down
Receiving Processing
Receiving queue A
Receiving queue B
Receiving Port A
Receiving Port B
Sending Port A
Sending Port B

Figure 4.1 The structure of the node class

As illustrated in Figure 4.1 one node unit contains 8 member objects:

1) One frame generator,
2) Two sending queues on each line,
3) Two receiving queue on each line,
4) Two tables for rejecting duplicated frame one each line,

5) Two tables for rejecting the circular frames on each line.

6) The “send coordinator and frame length countdown”.

7) The “receiving processing” are two main control functions during sending and receiving.

8) The “sending port A, B” and “Receiving ports A, B” are a group of structure variable for storing the frame structure and status variables for labelling the sending and receiving status. This will be explained in next sections.

4.1.2 The Frame Structure

Before going to the detail of single components in the node unit, the frame structure used in this environment should be explained first.

According to the HSR frame format, the frame here is a structure contains the destination address, the source address, the sequence number, the length of the frame, and a counter. The frame structure is shown in Figure 4.2.

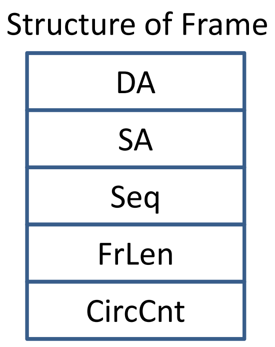


Figure 4.2 The structure of the frame in the simulation

The destination address, source address and the sequence number together compose the entry in the hash table. The length of the frame *FrLen* is used to simulate the time behavior of receiving or sending a frame. The counter *CircCnt* records the number of the nodes this frame has gone through before being accepted or rejected. This can be used to evaluate the performance of the table for rejecting the circular frames.

4.1.3 Generating Nodes in the Ring

A node is uniquely identified by its MAC address in the ring. To generate *n* nodes in the ring means to generate *n* different MAC addresses. According to the Ethernet protocol, the MAC address is divided into two parts: the first three octets of the MAC address is the Organizationally Unique Identifier (OUI) of the manufacturer, which is assigned by the IEEE Registration Authority; the last three Octets are assigned by the manufacturers.

There are two parameters can be configured. One is the OUI_NUM, which determines how many manufacturers there is in the network. The second parameter is the number of nodes *n* in the network. If the OUI_NUM is smaller than the number of nodes in the network, it means that there are some devices with the same manufacturer. This often corresponds to the reality considering the switch element used by the devices in the ring are from limited number of manufacturers.

When generating the nodes in the network, the OUIs are read out from the OUI tables at random locations, and assigned to the first three octets of the MAC address. Because the number of the nodes in the ring is larger than OUI, some OUIs are reused. The second three octets are generated randomly. Every time a MAC address is generated, it will be checked if it already exists in the network; if so, the second three octets should be regenerated until this MAC address is not same with any MAC addresses generated before.

After nodes generating is over, a table of *n* different MAC addresses is created

4.1.4 The Frame Generator

The frame generator is a member object in the Node class. As shown in Figure 4.3, it has a table (the destination address table) of the MAC addresses of all other nodes in the ring plus a multicast frame address. This table is created from the table of the generated nodes. For example, if this node is the Nr. 2 node in the ring, then the MAC address at the second position in the generated nodes table will be taken as the source address of this node, and other addresses are put into the destination address table. Besides the MAC address of the node itself, it also contains a MAC address which does not exist in the ring. This MAC address is used to simulate a unicast frame losing sender and receiver or a multicast frame losing sender. The reason why this may happen is explained in Section 2.3.

Together with each address in the address table there is also a sequence number. Each time a frame with is generated, the sequence number of the corresponding destination address will be increased by one.

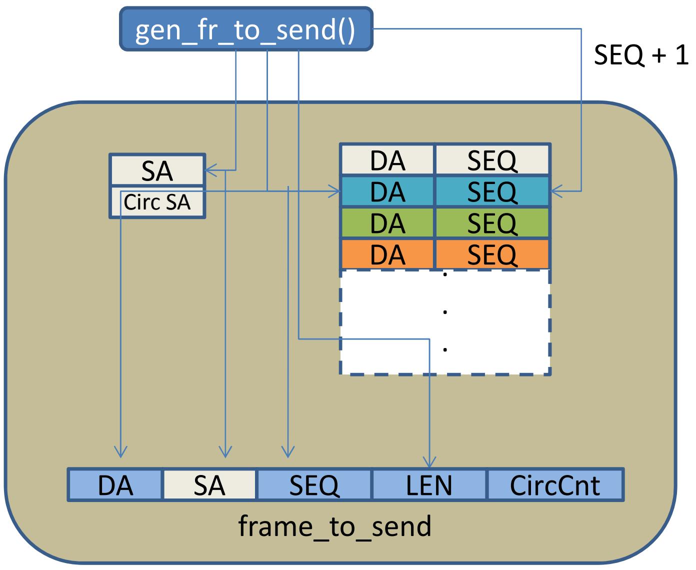


Figure 4.3 The structure of the frame generator and its

The probability of generating a multicast or a circular frame can be configured. When generating a frame, the function gen_fr_to_send() randomly select a destination address from the destination address table and a source address from the source address table with the preconfigured probability and increase the corresponding sequence number by one. gen_fr_to_send() also generate randomly the length of the frame between 64 and 1522. The chosen destination address, the source address, the sequence number, the generated frame length and the counter CircCnt are assigned to the variable frame_to_send.

4.1.5 The Queue

The two sending queues and the two receiving queues are normal queues defined with the Frame type using the queue template in the Standard Library of C++. It can execute normal operations for a queue structure like push, pop, and return the size of the queue.

4.1.6 The Lookup Tables

The proposed lookup tables are circular buffer, hash table with open addressing and aging, hash table combined with circular buffer. Because the structures and operating principles of each kind of table are already introduced in the previous chapter, it will not be repeated here. But in order to show how the concepts of each table are implemented, three important functions, namely the hash function, the searching entry function and the writing entry function are explained in detail for each type of table.

4.1.6.1 Circular Buffer

The circular buffer method does not use any hash functions. The flow charts of search entry and writing entry functions are illustrated in Figure 4.4 and Figure 4.5 below.

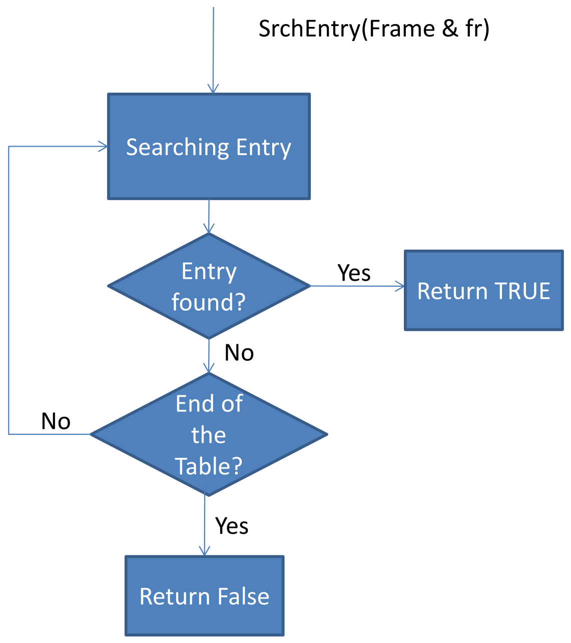


Figure 4.4 The flow chart of the searching entry function in circular buffer

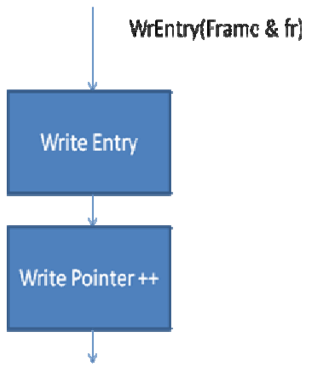


Figure 4.5 The flow chart of the writting entry function in circular buffer

4.1.6.2 Hash Table with Open Addressing and Aging

As stated in the chapter 3, the key which is the entry composed of the destination address, source address and the sequence number, is far from uniform, therefore the randomization should be performed before hashing.

The randomization function is chosen as the *Hashpjw()*. It coerces a key into a permuted integer through a series of bit operations on each byte in the key.

The code of *Hashpjw()* is shown in Figure 4.6 bellow:

```
int hashpjw(const void *key) {
   const char *ptr;
   int val;
   /*Hash the key by performing a
      number of bit operations on it. */
   val = 0;
   ptr = key;
   while (*ptr != '\0') {
      int tmp;
      val = (val << 4) + (*ptr);
      if (tmp = (val & 0xf0000000)) {
         val = val ^ (tmp >> 24);
         val = val ^ tmp;
       }
       ptr++;
    }
}
```

Figure 4.6 The hashpjw() funtion

This function is simple and proven to have a nice performance on randomizing the key. The bit operation is very suitable for implementing in the FPGA.

There are 3 probing sequences used: the linear probing, the quadratic probing, and the double hashing. The hash functions of each probing sequence are listed as following:

For linear probing and quadratic probing

$$h(x) = x \bmod m$$

For double hashing

$$h(x) = x \bmod m \quad g(x) = x \bmod (m-1)$$

Note here m is not chosen as a prime number. The reason for doing this is that the mod operation of a prime number cannot be synthesized in FPGA, other reason is mod a number of power of 2 can simply be realized by only taking the $\log_2 m$ number of the least significant bits. The effect of such choice must be evaluated through the simulation.

The flow charts of searching and writing entry function are shown respectively in Figure 4.7 and Figure 4.8.

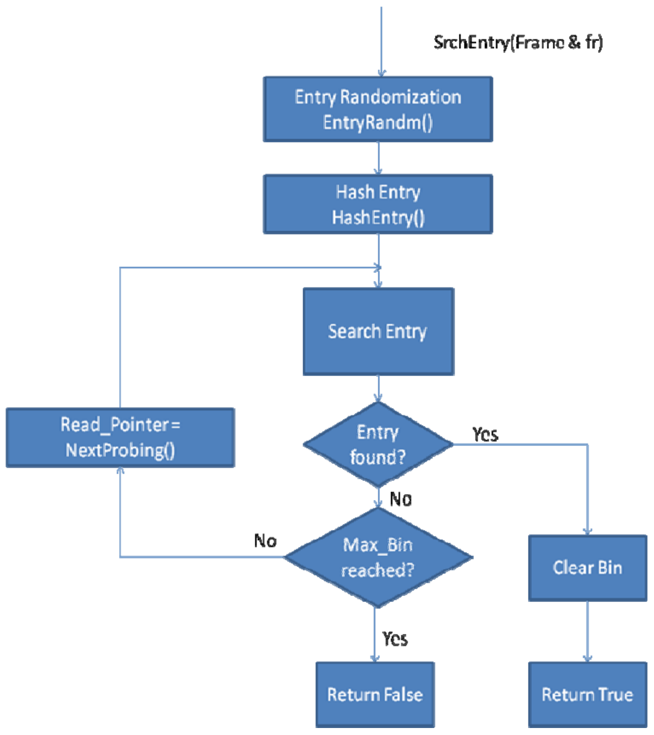


Figure 4.7 The flow chart of the searching entry function in hash table with open addressing and aging

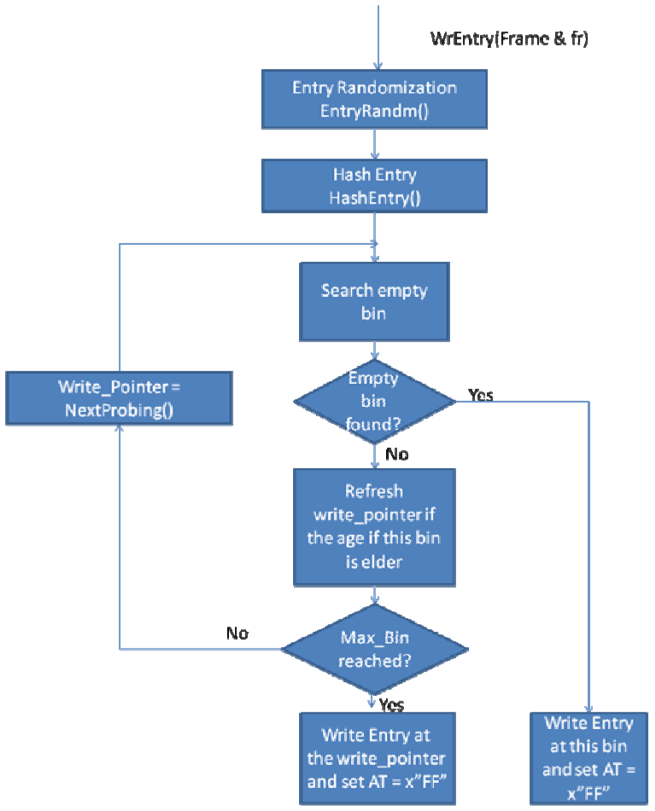

WrEntry(Frame & fr)
Entry Randomization EntryRandm()
Hash Entry HashEntry()
Search empty bin
Empty bin found?
Yes
No
Write_Pointer = NextProbing()
Refresh write_pointer if the age if this bin is elder
Max_Bin reached?
No
Yes
Write Entry at the write_pointer and set AT = x"FF"
Write Entry at this bin and set AT = x"FF"

Figure 4.8 The flow chart of the writing entry function in hash table with open addressing and aging

4.1.6.3 Hash Table Combined with Circular Buffer

For the hash table combined with circular buffer, the hash randomization function is still *Hashpjw().* The hash function is:

$$h(x) = x \bmod m$$

The flow charts of the searching and writing entry function are shown in Figure 4.9 and Figure 4.10 respectively.

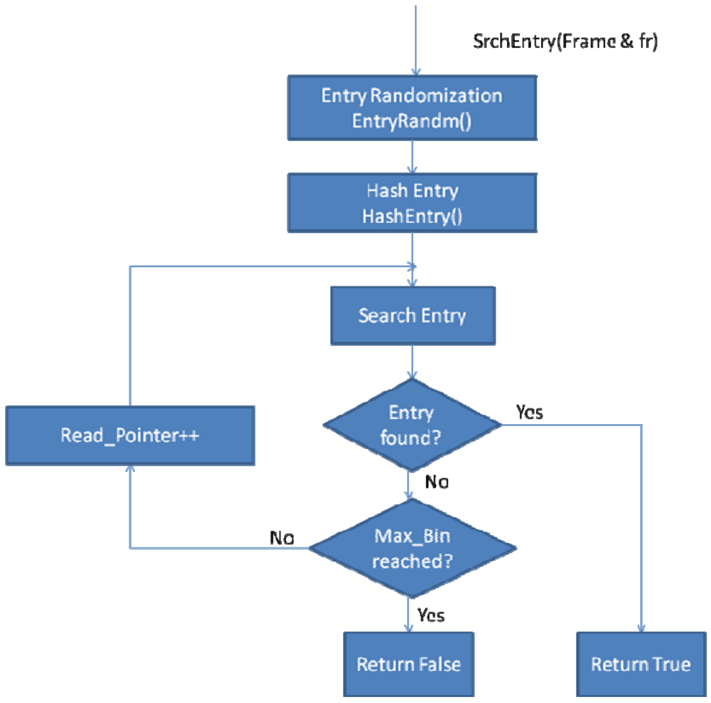


Figure 4.9 The flow chart of the searching entry function in hash table combined with circular buffer

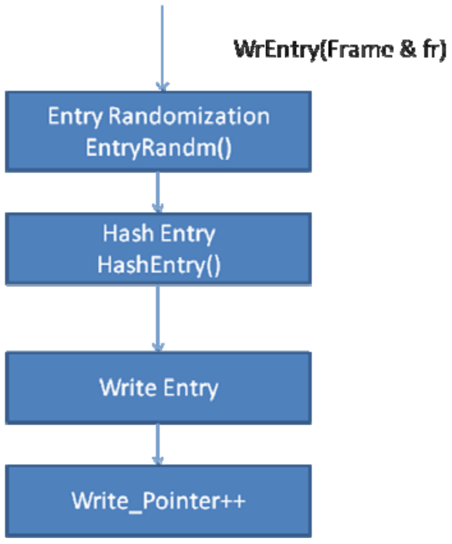


Figure 4.10 The flow chart of the writing entry function in hash table combined with circular buffer

4.1.7 The Receiving Processing

The receiving processing is a function which watches the receiving status on the two lines. Its tasks include searching and writing the entry of the received multicast frame and to the node dedicated unicast frame in the table for rejecting duplicated and circular frames, pushing the frame in the receiving queue if the frames are not found in either of the two tables. If frame comes at the same time, it should compare the two frames and make decisions. The way this function works is shown in Figure 4.11.

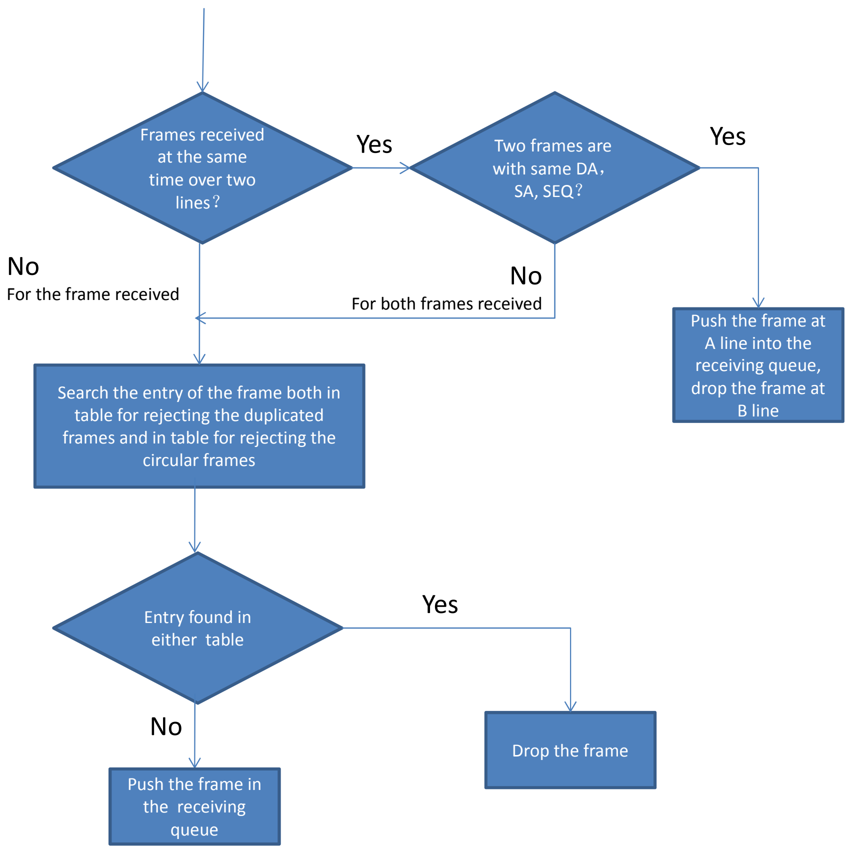

Frames received at the same time over two lines？
Yes
Two frames are with same DA，SA, SEQ？
Yes
No
For the frame received
No
For both frames received
Push the frame at A line into the receiving queue, drop the frame at B line
Search the entry of the frame both in table for rejecting the duplicated frames and in table for rejecting the circular frames
Entry found in either table
Yes
No
Drop the frame
Push the frame in the receiving queue

Figure 4.11 The flow chart of the receiving processing

4.1.8 Send Coordinator and Sending Port Use Count Down

Because each node has only one send port on each line, the using of the sending port must be coordinated between sending and forwarding. The basic principle of coordination is: the ***forwarding always has the priority***. The reason for doing that is to ensure the traffic in the ring is not delayed by the frames injected by the nodes.

The functionality of send coordinator and frame length counting down is realized by two functions: CntFrLen() and *IsSending().*

The CntFrLen() implement the count-down functionality and thus simulates the time behavior of the network. For example, if the receiving queue is not empty, the length of the frame on top of the queue will be loaded to the counter and the counter is counted down by one at each simulation time. During counting down, no new loading is allowed.

This stands for the time of using the sending port by this frame. After the counter reaches zero, the frame will be popped from the queue, and assigned to the sending port waiting to be received by the node next to it.

For the node next to this node, the receiving port should also be used when count-down is ongoing at the sending porting of this node. The node next to this node should sense that it is sending a frame. This *IsSending()* function tells the neighbor node that it is sending. If a frame requires to be sent a frame to the ring, but it sense the neighbor node is sending a frame towards it, the sending to the ring should not be allowed, because the receiving frame will probably be forwarded later. This operation principle is shown in Figure 4.12.

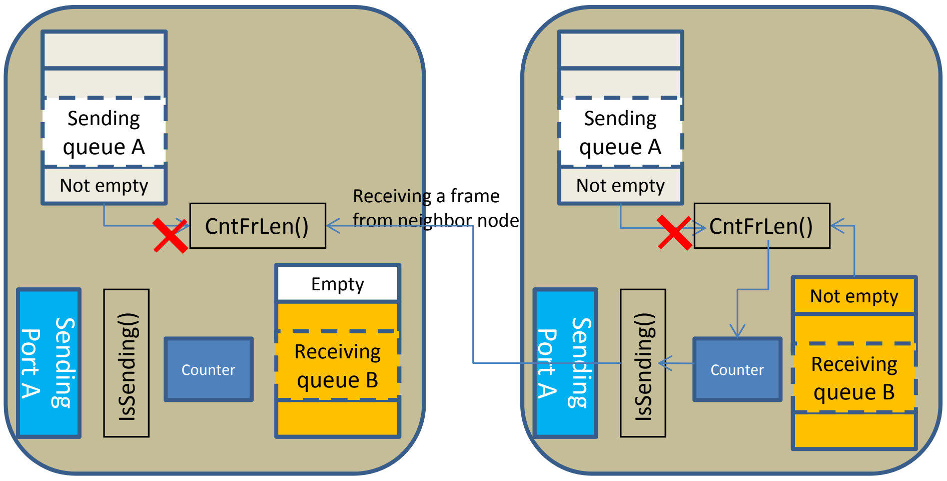


Figure 4.12 The sending coordination principle and frame length counting down

With such principle, the number of the frames in the receiving queue will never exceed 1 on the precondition that all frame are of the same length. Let us consider the following scenario:

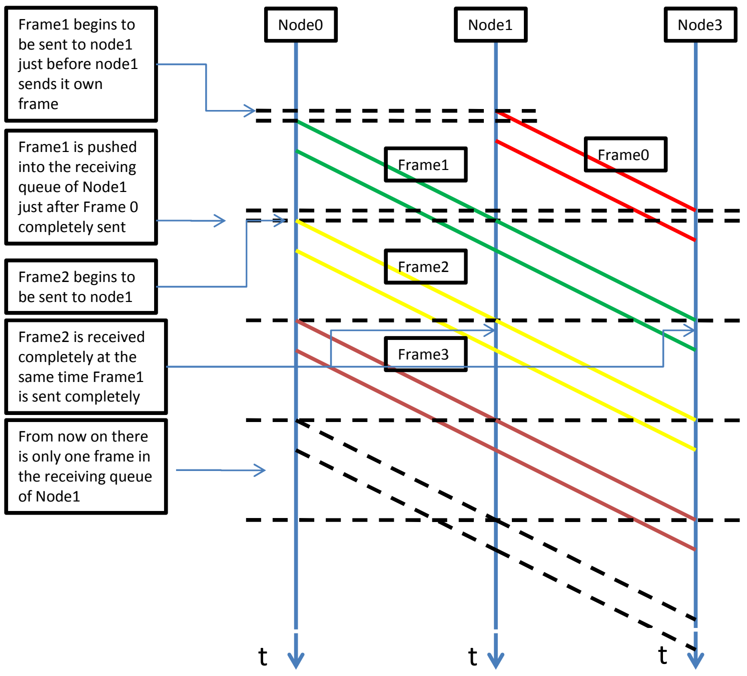

Frame1 begins to be sent to node1 just before node1 sends it own frame
Frame1 is pushed into the receiving queue of Node1 just after Frame 0 completely sent
Frame2 begins to be sent to node1
Frame2 is received completely at the same time Frame1 is sent completely
From now on there is only one frame in the receiving queue of Node1
Node0
Node1
Node3
Frame1
Frame0
Frame2
Frame3
t
t
t

Figure 4.13 The number of frames in the receiving queue will never exceed 1

In Figure 4.13 there are always frames in the sending queue of Node1, and the neighbor node sends frames without breaking. Assuming just short before the node0 begin to send frame to node1, node1 begin to send its own frame (frame1). After node1 finishes sending its own frame1, other frames in its sending queue cannot be sent because Node1 has sensed that Node0 is sending a frame to him. Frame0 is pushed into the receiving queue shortly after Fame0 is completely sent and should be sent next. During sending the frame0, the node1 still continues receiving frame (frame2) from node0. When frame1 is completely sent, frame2 is pushed into the receiving queue. From now on, the number of frames in receiving queue stays constant at 1 and does not increase any more.

4.1.9 The Sending Port

The sending port is implemented by two functions, a variable of *Frame* type and a variable of type *bool.*

For line A, the function *SendForwardFrToPortA()* pop the frame from the receiving queue when the counter has reached zero, then assign the frame to the variable *m_OutPortA* of type *Frame* and set the variable *FrPortA TRUE*.

The function *SendPortA()* should be called by the receiving function of the neighbor node.

It returns the value of *m_OutPortA* and set the *FrPortA* back to *FALSE.*

The same applies for line B.

4.1.10 The Receiving Port

The receiving port is implemented by one function, a variable of type *Frame* and variable of type *bool* .

For line A, the function *ReceiveFrA()* calls the *SendPortA()* function of its neighbor node, assign the variable of type *Frame* returned by *SendPortA()* to the variable *ReceivedFrameA* and set the status variable *FinishReceivingFromPortA TRUE.*

The ReceivedFrameA and *FinishReceivingFromPortA* are used by the receiving processing functionality. After the processing, *FinishReceivingFromPortA* is set back to *FALSE.*

The same applies for line B.

4.1.11 Operation Parallelization and Simulation Process

Operation Parallelization

Because the simulation program is written in C++, in order to simulate the behavior of the hardware, the operations of nodes must be parallelized. The parallelization is realized by the method shown in Figure 4.14.

By doing this way, all operations of a node can be executed at the same run time and the sequence that one node executes an operation before or after the other node does not play a role. However, the sequence that one operation should be executed before the or after the other operation is still to be considered.

The operation of assigning a frame to the sending port should be executed before the reception of a frame from the neighbor node otherwise the reception of a frame is delayed to the next runtime. The receiving processing operation should executed after the receiving from neighbor node operation, so that it can deal with the situation where two frames is received on both line at the same time. The send coordination and send port use counting down is executed last because it needs the information of the sending queue and of the receiving queue. Only after the receiving from neighbor node operation and processing of received frames operation are finished, the information in the receiving queue is ready.

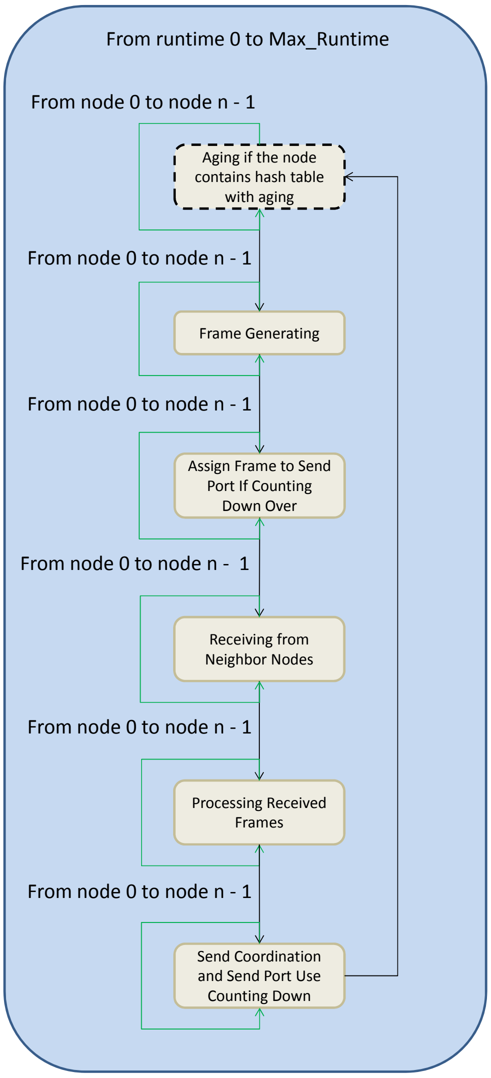

From runtime 0 to Max_Runtime
From node 0 to node n - 1
Aging if the node contains hash table with aging
From node 0 to node n - 1
Frame Generating
From node 0 to node n - 1
Assign Frame to Send Port If Counting Down Over
From node 0 to node n - 1
Receiving from Neighbor Nodes
From node 0 to node n - 1
Processing Received Frames
From node 0 to node n - 1
Send Coordination and Send Port Use Counting Down

Figure 4.14 The paraellization of the node operations

Simulation Process

The simulation process is performed in following steps:

MAC address table generation: the MAC address of each node is generated in the way described in the section 4.1.3. If there are *n* nodes in the network, *n* MAC addresses are generated. Some MAC addresses can have the same OUI.

An array of *n* nodes is created: Each element in the array is assigned by one MAC address from the created MAC address table.

An array of *n* time points is generated: the element in the array indicates at what time the corresponding node should generate a frame.

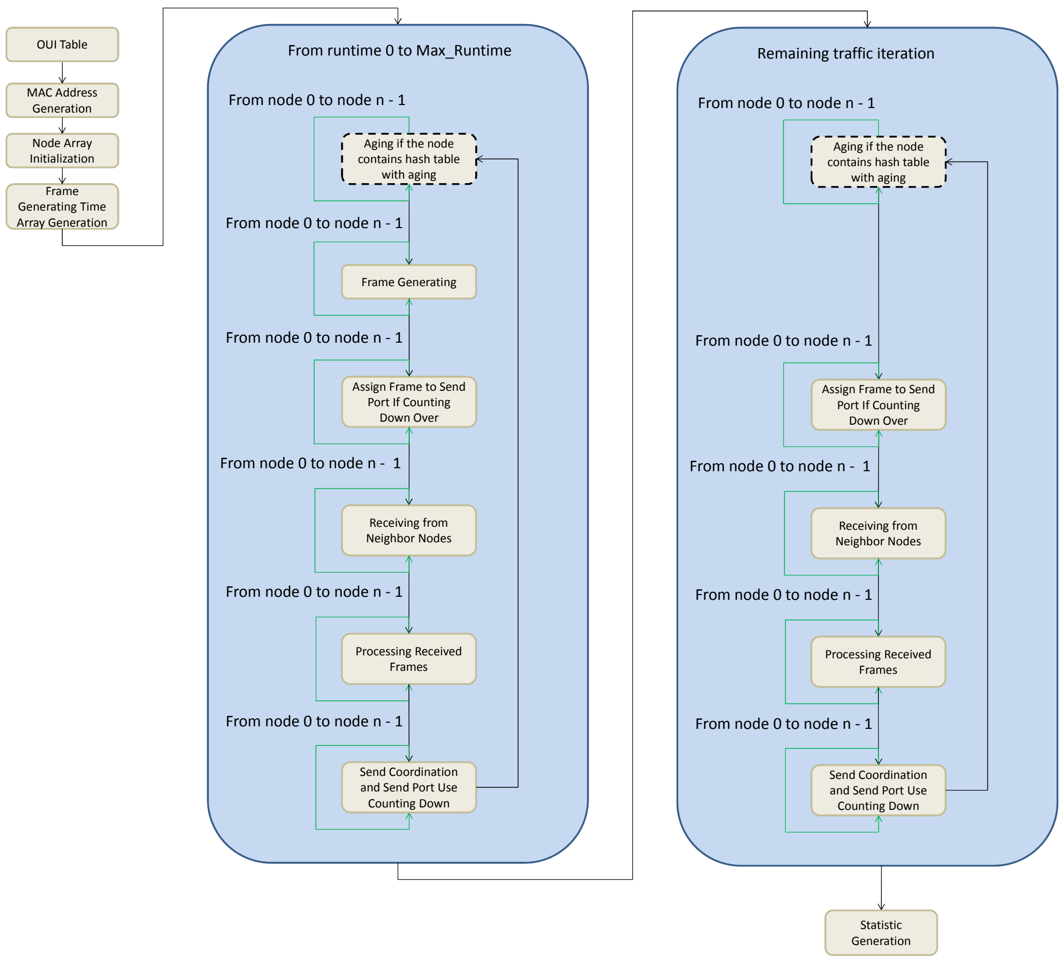
The main iteration: the simulation begins here. In each iteration, each node should perform 5 or 4 operations, depends on if the node contains the hash table with aging:

1) (Aging, if the node contains the hash table with aging).
2) Generates frame if its frame generating time is reached.
3) Assign the frame to the send port if counting down for this frame is over.
4) Receiving frames from the neighbor nodes at both lines if there are frames presented at the SentPort of the neighbor nodes.
5) Process the received frames.
6) If there are frames in the sending queue or the forwarding queue, load the frame according to the sending coordination principle introduced in section 4.1.8 to the counter and begin counting down.

The remaining traffic iteration: after the main iteration is over, there are still frames stays in the network. In this iteration, the nodes perform the same actions listed in main iteration, except no node generates new frames. The iteration will stop until no frame is left in the ring.

During the main iteration and the remaining traffic iteration, statistics are performed. When the simulation is over, the statistic result is generated.

The flow of the simulation is illustrated in Figure 4.15.



OUI Table
MAC Address Generation
Node Array Initialization
Frame Generating Time Array Generation
From runtime 0 to Max_Runtime
From node 0 to node n - 1
Aging if the node contains hash table with aging
From node 0 to node n - 1
Frame Generating
From node 0 to node n - 1
Assign Frame to Send Port If Counting Down Over
From node 0 to node n - 1
Receiving from Neighbor Nodes
From node 0 to node n - 1
Processing Received Frames
From node 0 to node n - 1
Send Coordination and Send Port Use Counting Down
Remaining traffic iteration
From node 0 to node n - 1
Aging if the node contains hash table with aging
From node 0 to node n - 1
Assign Frame to Send Port If Counting Down Over
From node 0 to node n - 1
Receiving from Neighbor Nodes
From node 0 to node n - 1
Processing Received Frames
From node 0 to node n - 1
Send Coordination and Send Port Use Counting Down
Statistic Generation

Figure 4.15 The simulation process

4.2 Simulation Configuration

In this section, the meaning and configuration of the parameter used in the simulation are explained. The time unit used in the simulation is explained, the methodology applied in the simulation and the contents of the simulation are introduced.

4.2.1 Parameters Configuration

In the Simulation, there are several parameters should be considered.

- For the network configuration:

  NODE_QTY : The number of the node in the network

  MAX_FRAME_LEN : The maximum frame length

  GEN_FRAME_INTV : The time interval with which a node should generate a frame

  MULTI_CAST_PERCENTAGE : the percentage of the multicast frames in the generated frames

  CIRC_PERCENTAGE : the percentage of the circular frames in the generated frames

- For the configuration of the circular buffer:

  HASHTB_CIRC1_UNIMULTI_DEPTH : The size of the circular buffer for rejecting the duplicated frames

  HASHTB_CIRC1_CIRC_DEPTH : The size of the circular buffer for rejecting the circular frames

- For the configuration of the hash table with open addressing and aging

  HASHAGEMAX: the maximum age of each bin

  HASHTB_AGE_UNIMULTI_DEPTH : The size of the table for rejecting the duplicated frames

  HASHTB_AGE_UNIMULTI_BIN_DEPTH : The max_bin of the table for rejecting the duplicated frames, introduced in the previous chapter

  HASHTB_AGE_CIRC_DEPTH : The size of the table for rejecting the circular frames

  HASHTB_AGE_CIRC_BIN_DEPTH : The max_bin of the table for rejecting the circular frames

- For the configuration of the hash table combined with circular buffer

  HASHTB_CIRC0_UNIMULTI_DEPTH : The size of the table for rejecting the duplicated frames

  HASHTB_CIRC0_UNIMULTI_BIN_DEPTH : The max_bin of the table for rejecting the duplicated frames

  HASHTB_CIRC0_CIRC_DEPTH : The size of the table for rejecting the circular frames

  HASHTB_CIRC0_CIRC_BIN_DEPTH : The max_bin of the table for rejecting the circular frames

The consideration to each parameter is discussed as bellow.

For the network configuration

- NODE_QTY:

  As discussed in section 3.2, the number of the node in the network has an influence on the size of circular buffer. If a network has *n* nodes, then for the unicast traffic, the size of the circular buffer should be at least *n – 2.*

  For multicast frames, the requirement is released. As shown in Figure 4.16, node1 send a multicast frame. If there is no delay during the traffic, the two frames will reach node4 at the same time. Therefore the delay between the two lines is zero in this case.

  However, because the table is used both for the multicast and unicast frames, the minimum table size should be *n - 2*.

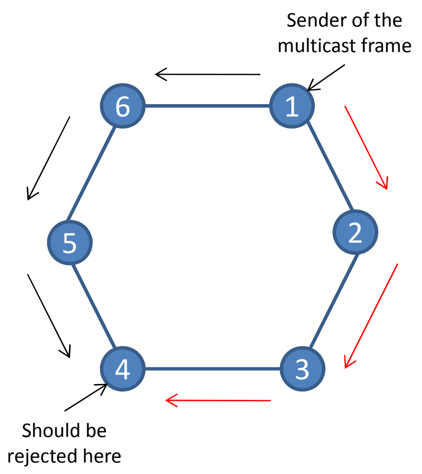


Figure 4.16 The multicast traffic in the ring, a multicast frame is sent in two directions and reaches the destination with zero delay

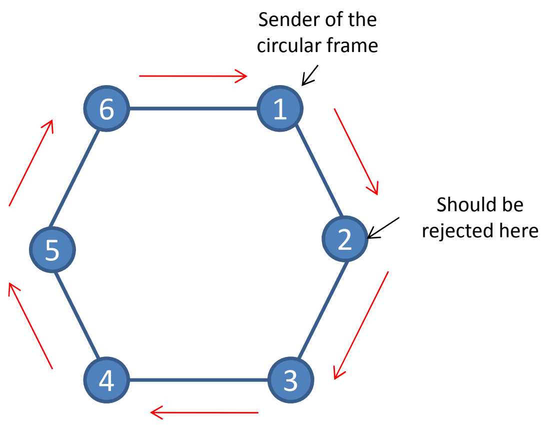


Figure 4.17 The circular frame should be rejected after it goes through the ring one round

For the circular frames, it should be rejected after it goes through the ring one round. Therefore the minimum size of the circular buffer for rejecting the circular frame should be at least *n.* This is shown in Figure 4.17.

- MAX_FRAME_LEN and GEN_FRAME_INTV

  A typical application scenario of HSR is the process bus. It carries real-time data from the measuring units, which requires a deterministic mode of operation. This means the period of generation of a frame by the units is deterministic. The generation of the frame takes place at a fixed time in a period. Because the information gathered by the unit on the bus has a fix format, the length of all the frames is the same, which is given by the MAX_FRAME_LEN in the simulation

- MULTI_CAST_PERCENTAGE, CIRC_PERCENTAGE

  On the process bus, most of the frames are mulicast frame. According to the application experience the MULTI_CAST_PERCENTAGE parameter is set to 0.9. The CIRC_PERCENTAGE is set to 0.0001.

For circular buffer:

- HASHTB_CIRC1_UNIMULTI_DEPTH

  The size of the table should be at least *n – 2*, if *n* is the number of nodes in the network. Here it is chosen as *n*.

- HASHTB_CIRC1_CIRC_DEPTH

  The size of the table should be at least *n*, if *n* is the number of nodes in the network

For hash table with open addressing and aging:

- *HASHAGEMAX*

  The maximum age of each bin should be at least $(n-2) * MAX\_FRAME\_LEN$, as stated in section 3.3.3.

- *HASHTB_AGE_UNIMULTI_DEPTH*

  It should be at least *n – 2,* if *n* is the number of nodes in the network. Here it is chosen as *n*.

- HASHTB_AGE_UNIMULTI_BIN_DEPTH

  Varies from 2 to 5

- *HASHTB_AGE_CIRC_DEPTH*

  It should be at least *n*, if *n* is the number of nodes in the network

- *HASHTB_AGE_CIRC_BIN_DEPTH* :

  Varies from 2 to 5

For hash table combined with circular buffer:

- *HASHTB_CIRC0_UNIMULTI_DEPTH*

  It should be at least *n – 2,* if *n* is the number of nodes in the network. Here it is chosen as *n*.

- HASHTB_CIRC0_UNIMULTI_BIN_DEPTH

  Varies from 2 to 5

- *HASHTB_CIRC0_CIRC_DEPTH:*

  It should be at least *n,* if *n* is the number of nodes in the network

- HASHTB_CIRC0_CIRC_BIN_DEPTH

  Varies from 2 to 5

4.2.2 Time Unit of the Simulation

The time unit of the simulation is defined by the time needed for transmitting an octet on the Ethernet line.

If the line speed of the Ethernet is 100Mbits/s, then the one time unit in the simulation stands for $2 * 40ns = 80ns$; if the line speed is 1Gbits/s, one time unit stands for 8ns.

4.2.3 Simulation Methodology and Contents

This simulation is parameterized for the bus traffic according to the standard IEC 61850-9-2 [5]. In this standard, the frame generating interval of each unit on the ring should be fixed to 250us. Which is 250us/80ns = 3125 after conversion to the simulation time unit. The frame length is defined as 138 Bytes. Therefore, the rejection ratio is measured with the node number as variable.

There are five methods to be measured:

1) The circular buffer
2) The hash table with open addressing and aging with linear probing
3) The hash table with open addressing and aging with quadratic probing
4) The hash table with open addressing and aging with double hashing
5) The hash table combined with circular buffer

For each method, the rejecting ratio of the duplicated frames and circular frames are measured given the same node number. Each measurement will be repeated 5 times; the average rejecting ratio is calculated. For the hash table with aging, and hash table combined with circular buffer, measurements are performed for each bin number. Besides, the number of the frame in the sending queue and the frame discarding ratio are also measured. These two parameters are helpful in explaining the communication situation in the ring.

The simulation will run 1000000 time units, which corresponds a time of 80ms in reality with the 100Mbits/s Ethernet.

Table 4.1 shows the contents of the simulation.

Note for the hash table with circular buffer method, some simulation cannot be performed. Because the size of the table is chosen as the number of nodes in the ring, the bin number should fully divide the table size. The shaded square in Table 4.1 indicated the simulation which cannot be performed.

Some node numbers are deliberately chosen as power of 2, because the mod operation is then just choosing the corresponding least significant bits of the key.

Node Number																	
				8	16	20	25	32	40	45	50	55	60	64	70	75	80
Circular Buffer																	
Hash table with Aging	linear probing	2 bins	***Multicast Rejection Ratio***														
			Unicast Rejection Ratio														
			Frame Discarded Ratio														
			Num of Frame in the														

			Sending Queue														
		3 bins	*Same as above*														
		4 bins	*Same as above*														
		5 bins	*Same as above*														
	quadratic probing	2 bins	*Same as above*														
		3 bins	*Same as above*														
		4 bins	*Same as above*														
		5 bins	*Same as above*														
	double hashing	2 bins	*Same as above*														
		3 bins	*Same as above*														
		4 bins	*Same as above*														
		5 bins	*Same as above*														
Hash table with circular buffer		2 bins	*Same as above*				▨			▨		▨				▨	
		3 bins	*Same as above*	▨	▨	▨	▨	▨	▨		▨	▨		▨	▨		▨
		4 bins	*Same as above*				▨			▨	▨	▨			▨	▨	
		5 bins	*Same as above*	▨	▨			▨						▨			

Table 4.1 The simulation contents

4.3 Simulation Results and Discussion

In this section, the simulation results are presented, compared and discussed. The relation between the frame rejection ratio and the number of nodes in the network is explained.

The result of this simulation is statistics with the following contents:

- The number of the unicast frames generated in the network during the runtime (N_{uni_gen})
- The number of the multicast frames generated in the network during the runtime (N_{multi_gen})

- The number of the circular frames generated in the network during the runtime (N_{circ_gen})
- The accepted unicast frames in the network (N_{uni_acpt})
- The rejected unicast frames in the network (N_{uni_rej})
- The accepted multicast frames in the network (N_{multi_acpt})
- The rejected multicast frames in the network (N_{multi_rej})
- The number of nodes a circular buffer has gone through before being rejected

The rejection ratio of the unicast frame is calculated as

$$R_{unicast} = 1 - \frac{(N_{uni\_acpt} - N_{uni\_rej})}{(N_{uni\_acpt} + Nuni_{uni\_rej})} \quad (4.1)$$

For example, if the number of accepted unicast frame equal to the number of rejected unicast frame, the rejection ratio is 100%.

The rejection ratio of the multicast frame is calculated as

$$R_{unicast} = 1 - \frac{[N_{multi\_acpt} - N_{multi\_gen} * (Node\_Num - 1)]}{N_{multi\_gen} * (Node\_Num - 1)}$$

$$= 2 - \frac{N_{multi\_acpt}}{N_{multi\_gen} * (Node\_Num - 1)} \quad (4.2)$$

For example, if there are 20 nodes in the network, 100 multicast frames is generated during the runtime, the accepted multicast frame should be 19*100 = 1900. If the number of accepted multicast frame is 2000, the rejection ratio of the multicast frame is 94.3%.

A successful rejection of a circular frame is expressed by the number of nodes it has gone through before being rejected. If there are 20 nodes in the network, a successful rejection means that the circular frame is rejected just after it has gone through the ring for one round.

4.3.1 The Simulation Results of Circular Buffer

The simulation results of circular buffer are presented below.

Figure 4.18 shows that the multicast frame rejection ratio stays at the level of 100 percent, but drops dramatically when the number of nodes in the ring reaches 80. Figure 4.19 shows that the unicast rejection ratio stays at the level of 100 percent, but begins to drop dramatically after the number of nodes in the ring exceeds 60.

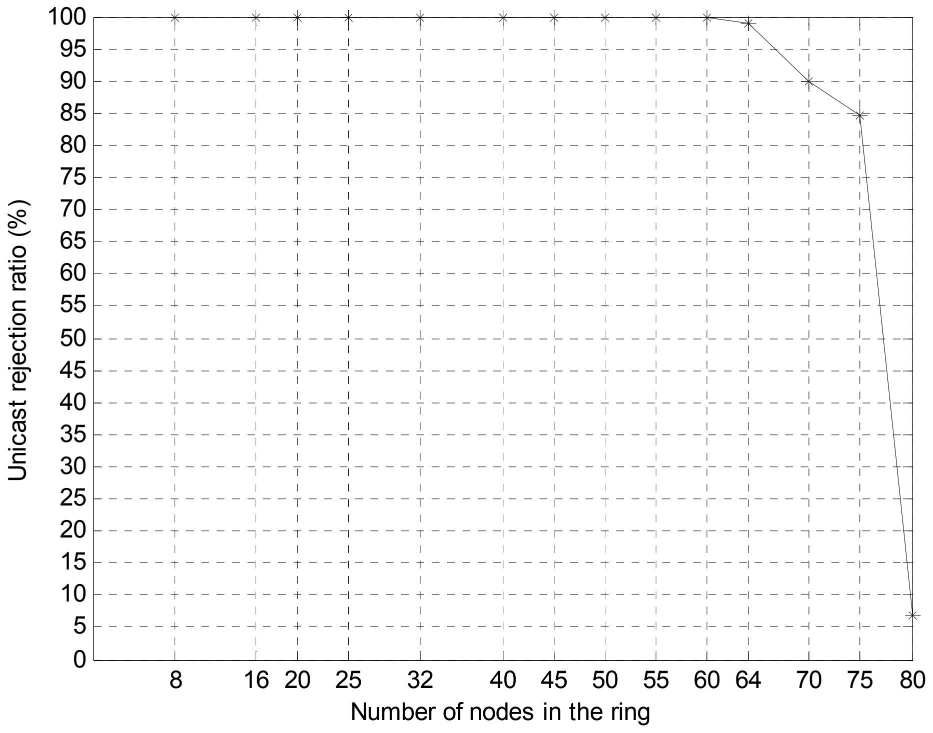


Figure 4.18 The multicast frame rejection ratio of circular buffer

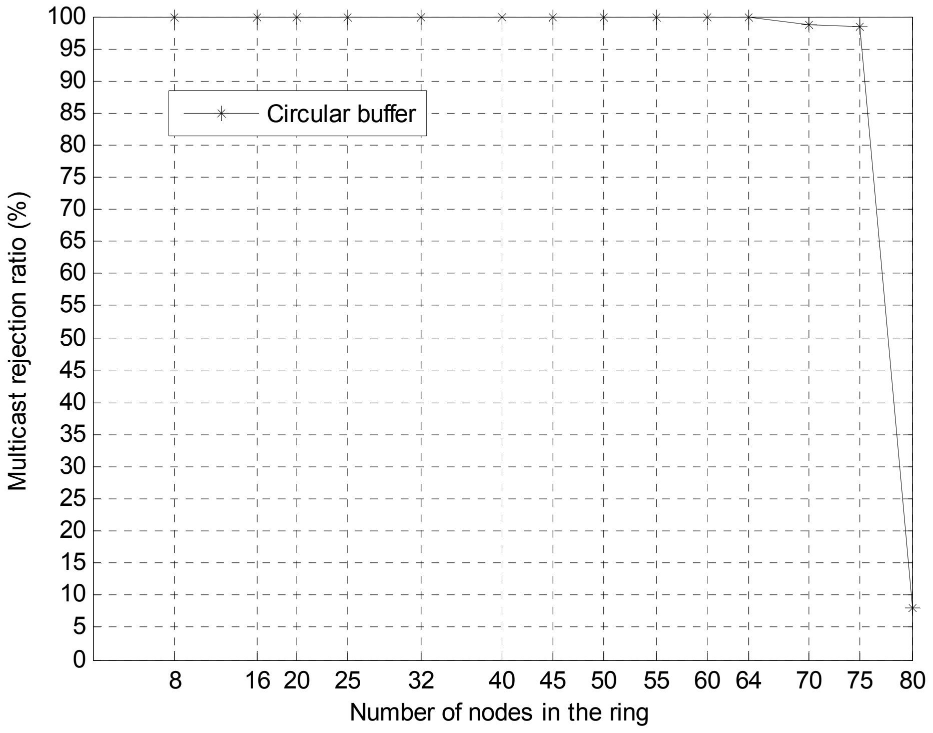


Figure 4.19 The unicast frame rejection ratio of circular buffer

4.3.2 The Simulation Results of Hash Table with Aging

Linear Probing

Figure 4.20 and Figure 4.21 shows the multicast frame rejection ratio and unicast frame rejection ratio of a hash table with aging and linear probing respectively. The rejection ratios of different bin numbers are drawn on the same figure.

The mulicast frame rejection ratio drops dramatically after the number of nodes in the ring exceeds 64, which is also observed in the circular buffer. The unicast frame ratio has the similar tendency, besides, the lower the bin number, the faster the rejection ratio degrades. The linear probing with 5 bins has the best performance.

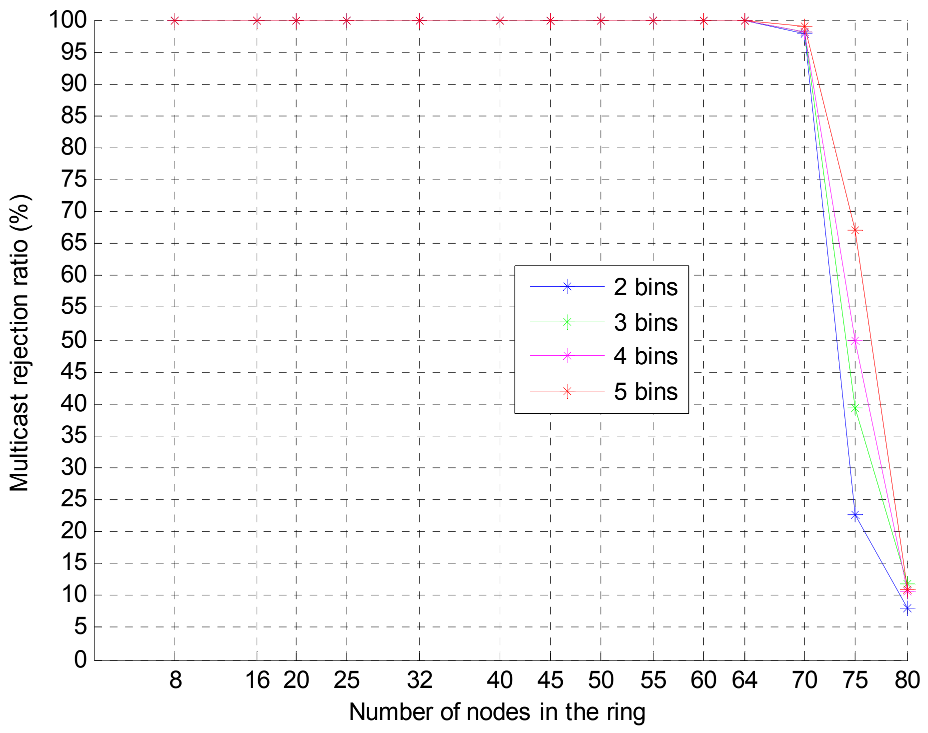


Figure 4.20 The multicast frame rejection ratio of hash table with aging and linear probing

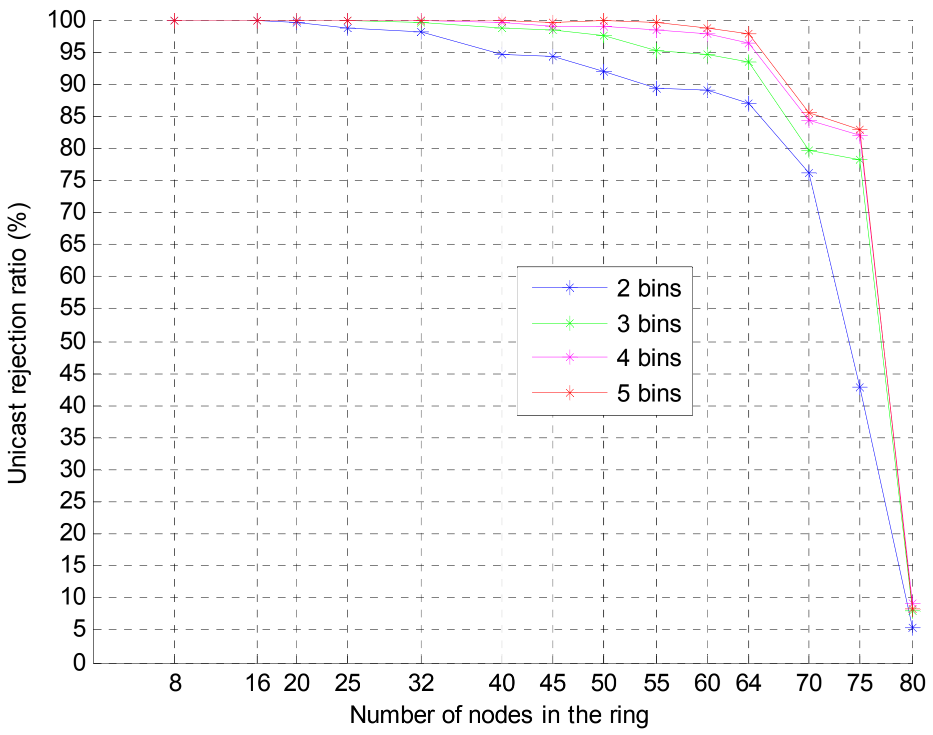


Figure 4.21 The unicast frame rejection ratio of hash table with aging and linear probing

Quadratic Probing

Figure 4.22 and Figure 4.23 shows the multicast frame rejection ratio and unicast frame rejection ratio of hash table with aging and quadratic probing respectively. The rejection ratios of different bin numbers are drawn on the same figure. The same tendency is repeated again as it is in the quadratic probing.

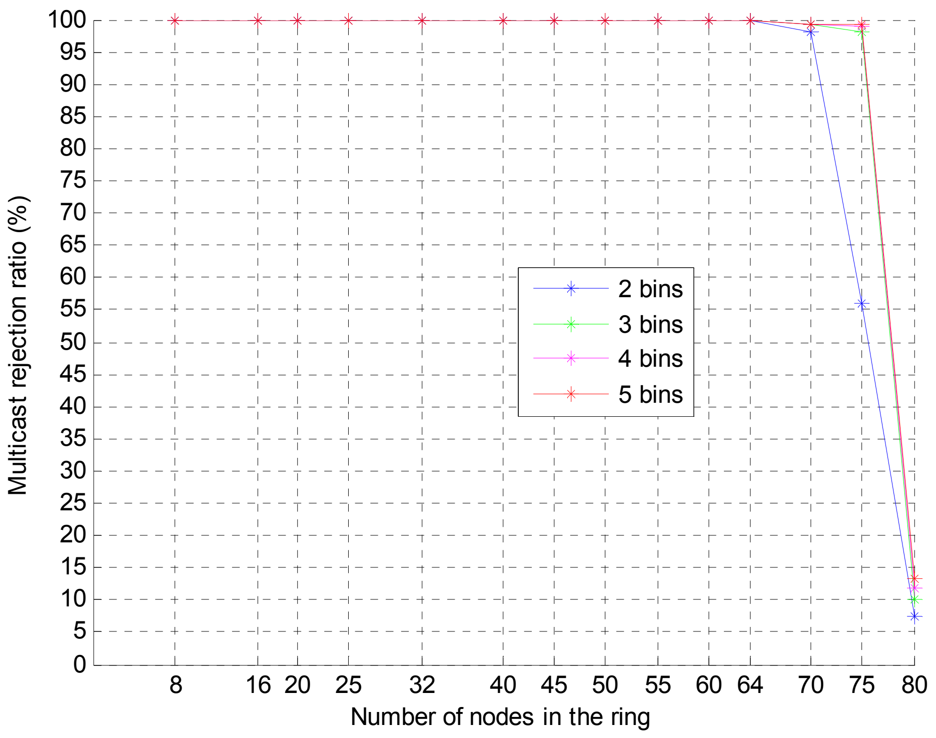


Figure 4.22 The multicast frame rejection ratio of hash table with aging and quadratic probing

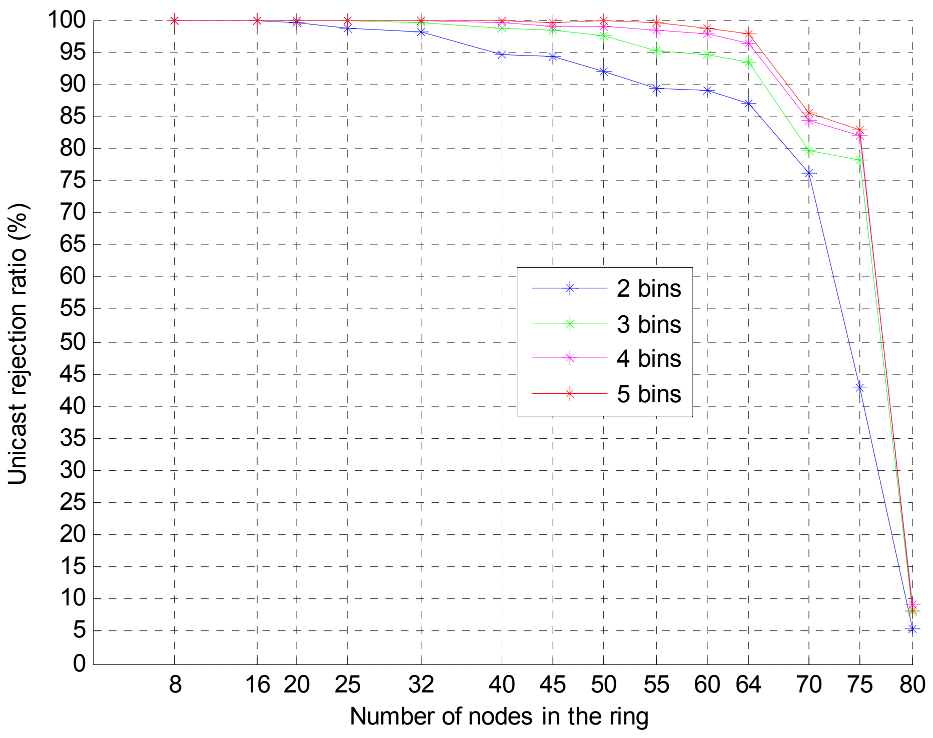


Figure 4.23 The unicast frame rejection ratio of hash table with aging and quadratic probing

Double Hashing

Figure 4.24 and Figure 4.25 shows the multicast frame rejection ratio and unicast frame rejection ratio of hash table with aging and double hashing respectively. The rejection ratios of different bin numbers are drawn on the same figure. The same tendency is repeated again as it is in the linear probing.

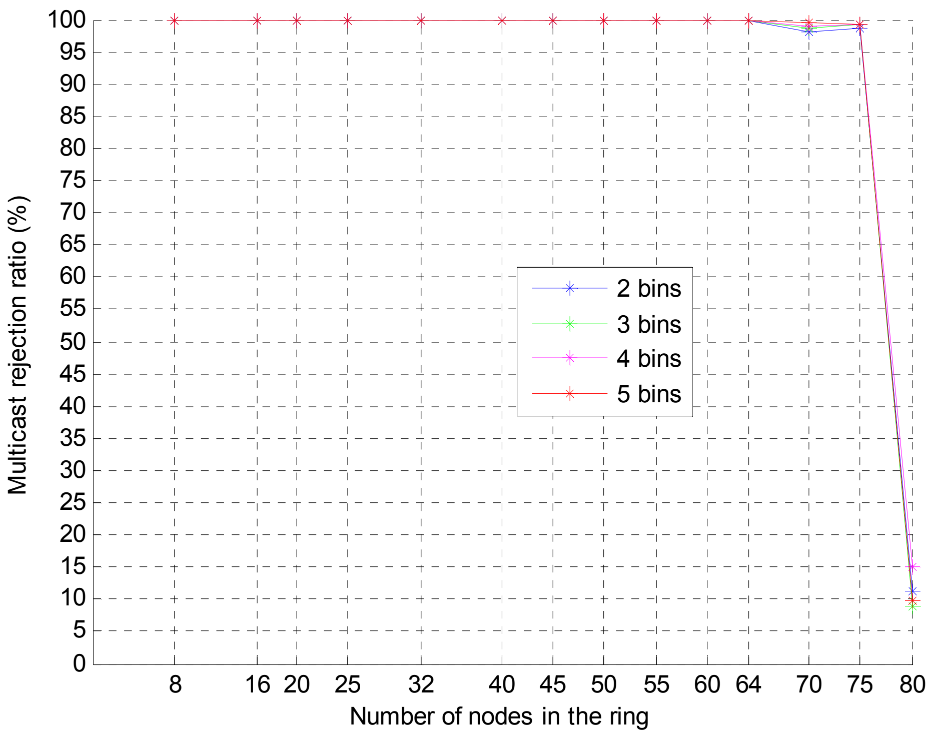


Figure 4.24 The multicast frame rejection ratio of hash table with aging and double hashing

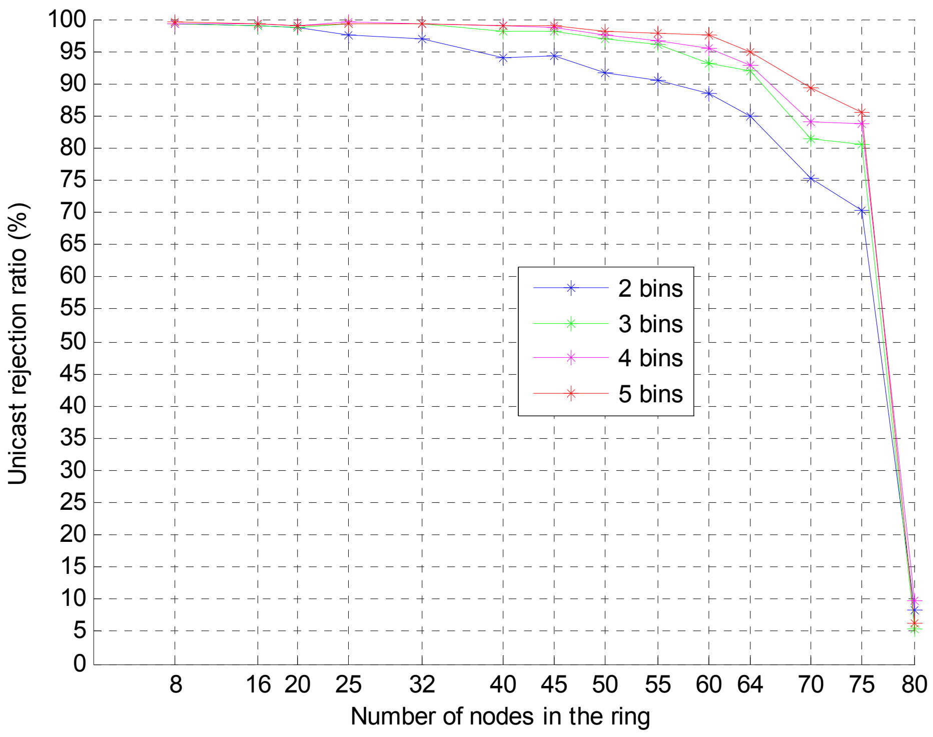


Figure 4.25 The unicast frame rejection ratio of hash table with aging and double hashing

4.3.3 Simulation Results of Hash Table Combined with Circular Buffer

Figure 4.26 and Figure 4.27 shows the multicast frame rejection ratio and unicast frame rejection ratio of hash table combined with circular buffer. For each node number, only the bin numbers which can divide it are simulated.

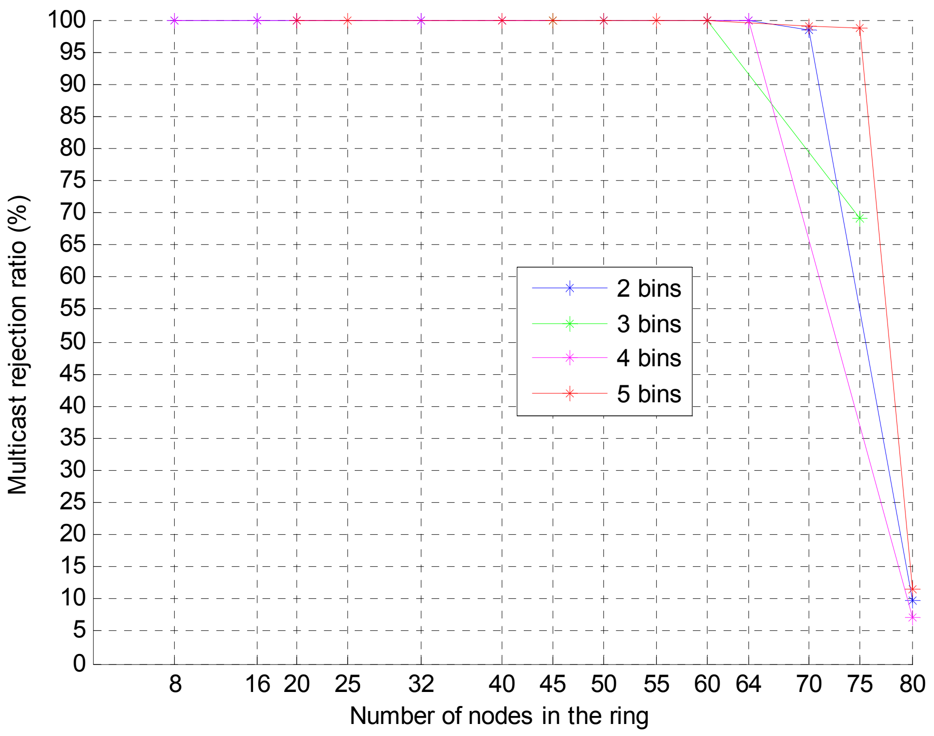


Figure 4.26 The multicast frame rejection ratio of hash table combined with circular buffer

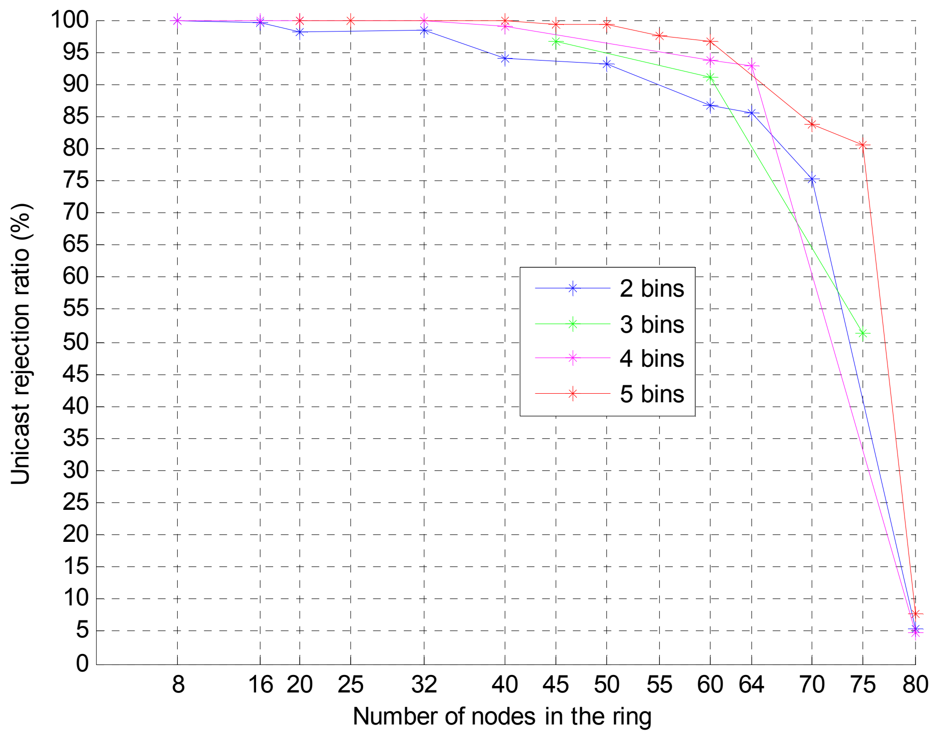


Figure 4.27 The unicast frame rejection ratio of hash table combined with circular buffer

4.3.4 Comparison of the Proposed Methods

The simulation results of the hash table with aging and hash table combined with circular buffer shows that the greater the bin number, is the rejection ratio higher. Therefore, the following methods are compared:

1) Circular buffer
2) Hash table with aging and linear probing, 5 bins
3) Hash table with aging and quadratic probing, 5bins
4) Hash table with aging and double probing, 5 bins
5) Hash table with aging and double probing. 5 bins

Figure 4.28 shows the simulation results of the rejection ratio of the multicast frames. All the proposed methods have almost 100 percent rejection ratio when the node number is below 60. The performance of all the methods begins to degrade dramatically when the node number exceeds 60. When the node number reaches 80, the methods lose their functionality totally.

Figure 4.29 shows the simulation results of the rejection ratio of the unicast frames. The performance of all the methods begin to degrade after the node number exceeds 45. The performance of the circular buffer drops slightly slower than other methods. The performance of the hash table combined with circular buffer drops slightly faster than other methods.

Figure 4.30 and Figure 4.31 give a closer view of the multicast and unicast rejection ratio in the region between the node number of 0 to 50.

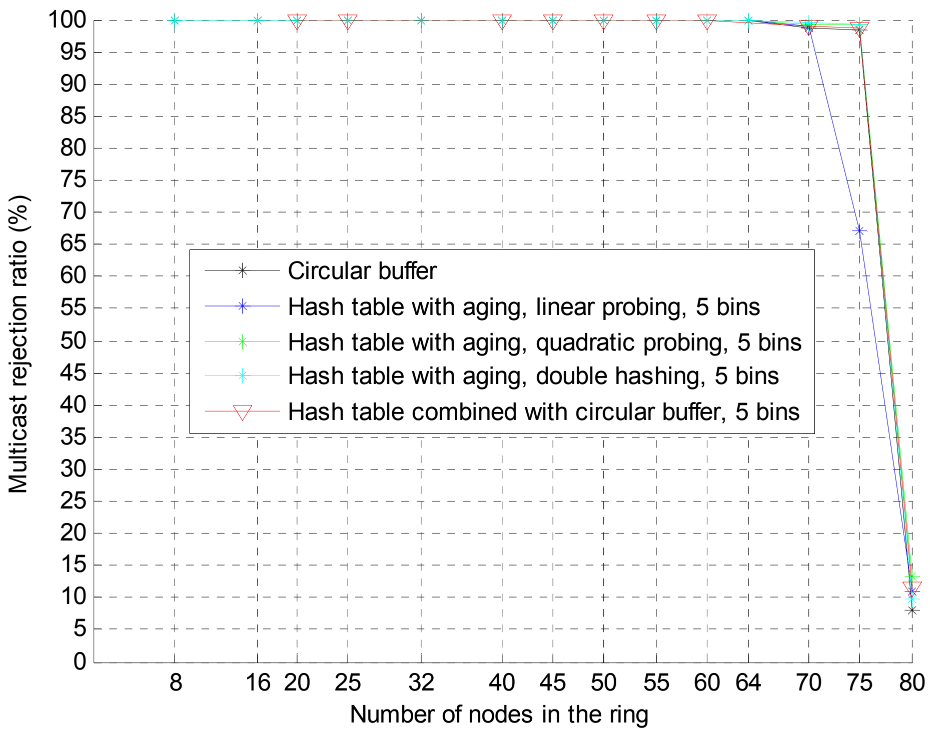


Figure 4.28 The multicast frame rejection ratio of proposed methods

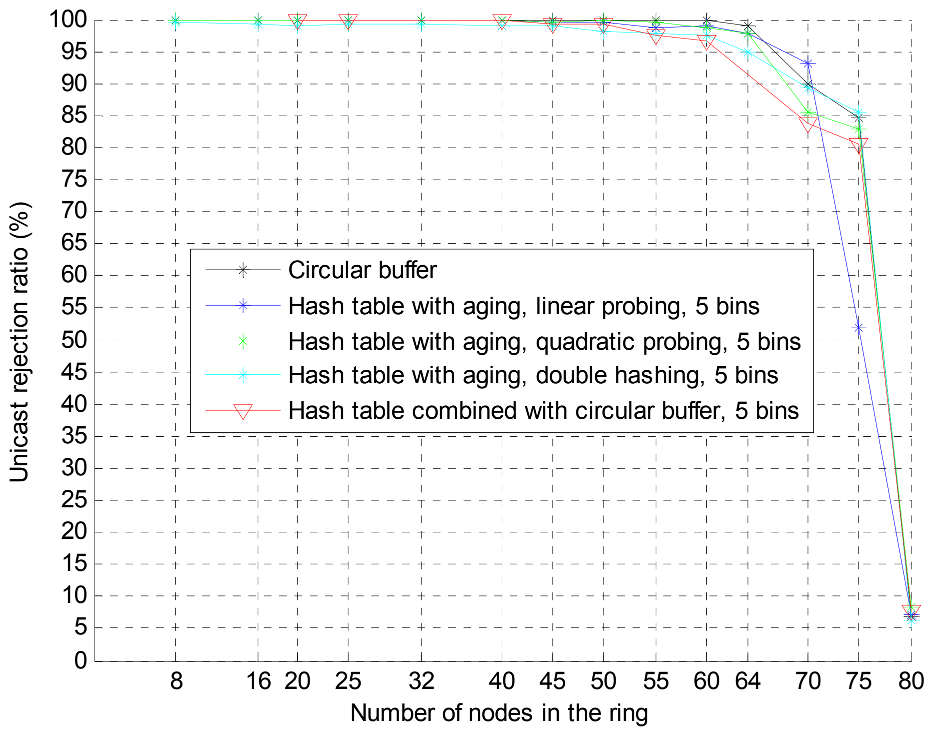


Figure 4.29 The unicast frame rejection ratio of proposed methods

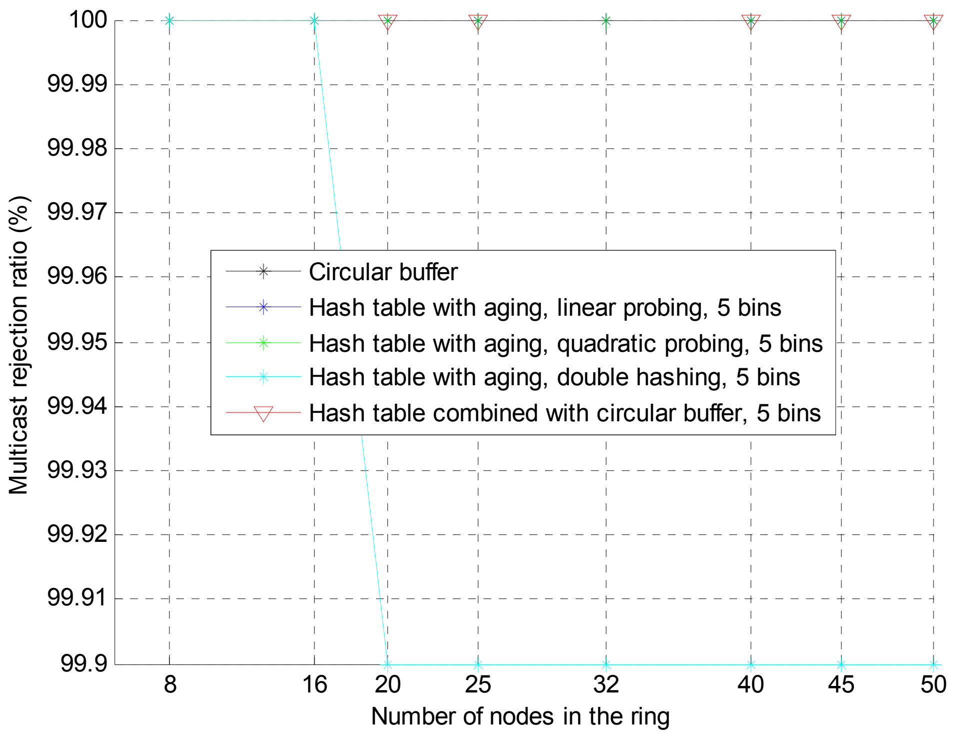


Figure 4.30 The multicast frame rejection ratio of proposed methods in the region of node number 0 to 50

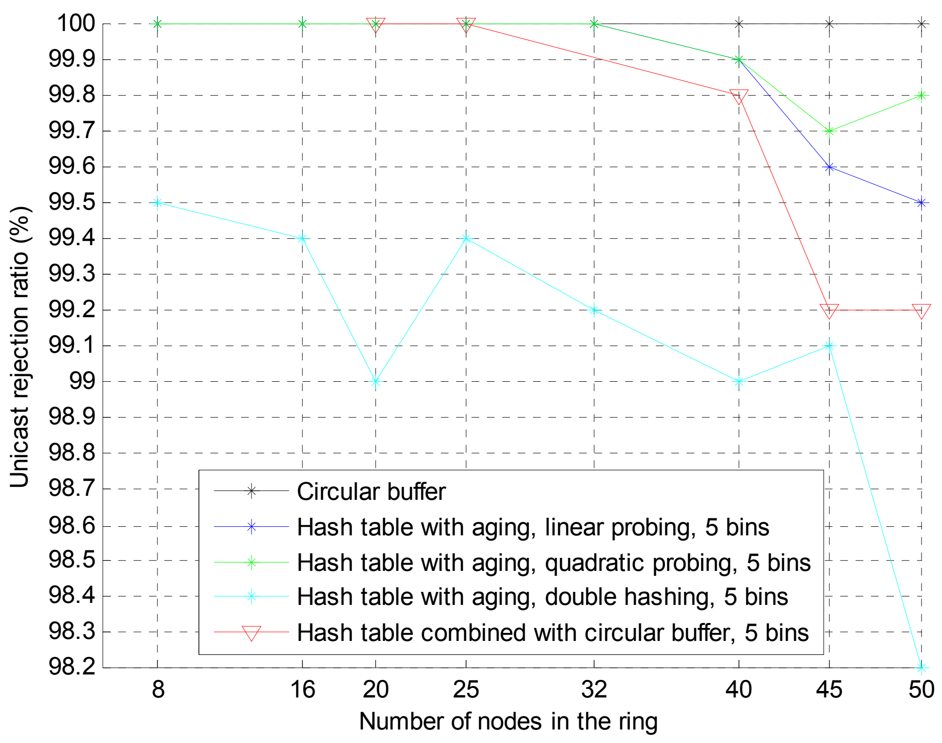


Figure 4.31 The unicast frame rejection ratio of proposed methods in the region of node number 0 to 50

For multicast frames, it can be observed from Figure 4.30 that the hash table with circular buffer has the best performance; the hash table with aging double hashing has the worst performance. The performance of the circular buffer, hash table with aging and quadratic probing, hash table with aging and linear probing and hash table combined with circular buffer maintains above 99 percent in this region.

4.3.5 Further Discussions

Why Rejection Ratio of Multicast Higher than Unicast

In the simulation it is observed that the rejection ratio of multicast frames is higher than unicast frames. This is because the frame receiving delay of multicast frames between the two lines are smaller than unicast frames. The smaller the delay, the smaller the possibility with which the entry of a frame is overwritten. This point has already been explained in Section 4.2.1.

The Reason for the Performance Degradation

In the simulation it is observed that the rejection ratio of both multicast and unicast frames degrades dramatically after the nodes in the ring exceeds a certain number. To see the reason of this, the scenario illustrated below is first introduced.

In Figure 4.32, all nodes generate multicast frames at the same time. After generation, each frame can be sent at once without any delay. After each frame has traveled half number of nodes in the ring, they are rejected at the same time. Now there are no frames in the ring. If at this time all the nodes generate frame again, the generated frames can be sent without any delay and the same happens as before. If the frame generation period is smaller than this time, the generated frames will accumulate the sending queue. And there is no such time that there is no frame in the ring.

The above situation is just a special case, if the nodes generate frames at different time in a period, the situation is more complex.

Therefore, the period with which nodes generate multicast frame should satisfy the following conditions:

$$T_{frame\_generation} \geq \frac{n_{node\_number}}{2} * t_{delay\_at\_node} \qquad (4.3)$$

For the unicast frame the condition should be

$$T_{frame\_generation} \geq n_{node\_number} * t_{delay\_at\_node} \qquad (4.4)$$

Here $t_{delay\_at\_node}$ is the time needed by a node for transmitting a frame. It is determined by the length of a frame and plus some overhead time for processing the frame.

In our situation the length of the frame is fixed in the IEC 61850-9-2 standard, the frame generation period is thus proportional to the number of nodes in the ring.

$$T_{frame\_generation} \propto n_{node\_number} \qquad (4.6)$$

The period with which a node generates a frame is also fixed in the IEC 61850-9-2 standard, which means $T_{frame\_generation}$ is fixed. Therefore, the following condition should be satisfied:

$$T_{frame\_generation} \leq T_{fixed} \qquad (4.7)$$

Therefore, the number of nodes in the ring cannot exceed a certain number.

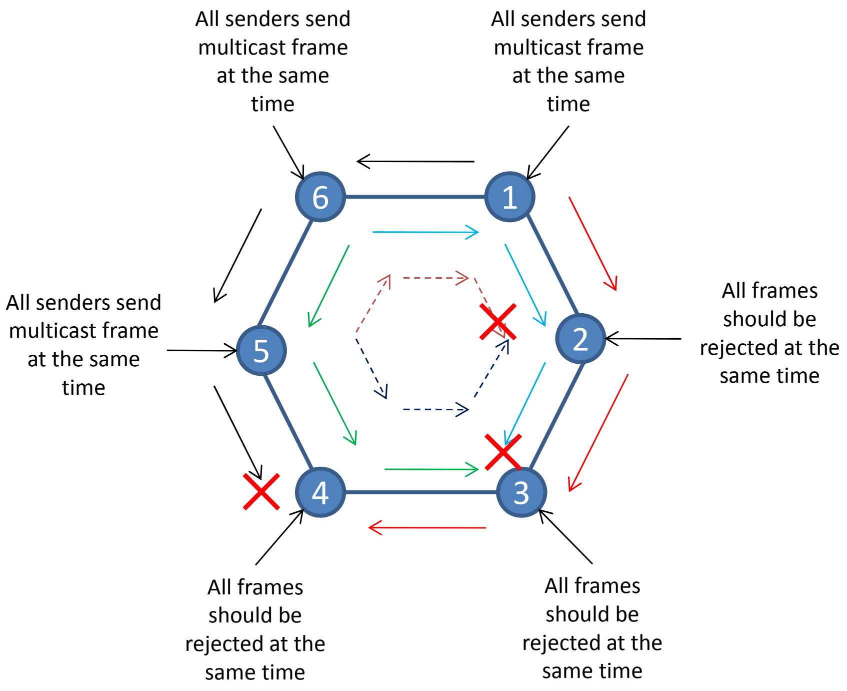

All senders send multicast frame at the same time
All senders send multicast frame at the same time
6
1
All senders send multicast frame at the same time
5
2
All frames should be rejected at the same time
4
3
All frames should be rejected at the same time
All frames should be rejected at the same time

Figure 4.32 All nodes generate a frame at the same time

Now let us consider the situation that the supposed frame generation period is greater than the fixed frame generation period, which means in turn that the number of nodes in the ring has exceeded a certain number.

In Figure 4.33, some frames accumulate in the sending FIFO of each node. The traffic on both lines is heavy. According to the sending forwarding coordination principle introduced in Section 4.1.8, as long as there are frames needing to be forwarded, the frames in the sending FIFO should wait. Now a multicast frame in the sending queue of B line has its chance to send, this chance can be created by accepting a unicast frame or rejecting a duplicated frame. The sent frame begins to traveling in the ring. During this time, the frame in the sending queue of line A does not get its chance to send, this could happen if it always failed to reject multicast frames or there is no unicast frame dedicated to it. In both cases, the node has to forward the frame so that the frame in the sending queue A always does not have its chance to send.

Because the frame on the A line is not sent, its entry does not exist in the table of all nodes, the frame sent on B line can thus go through the ring and come back to Node1 and be discarded there.

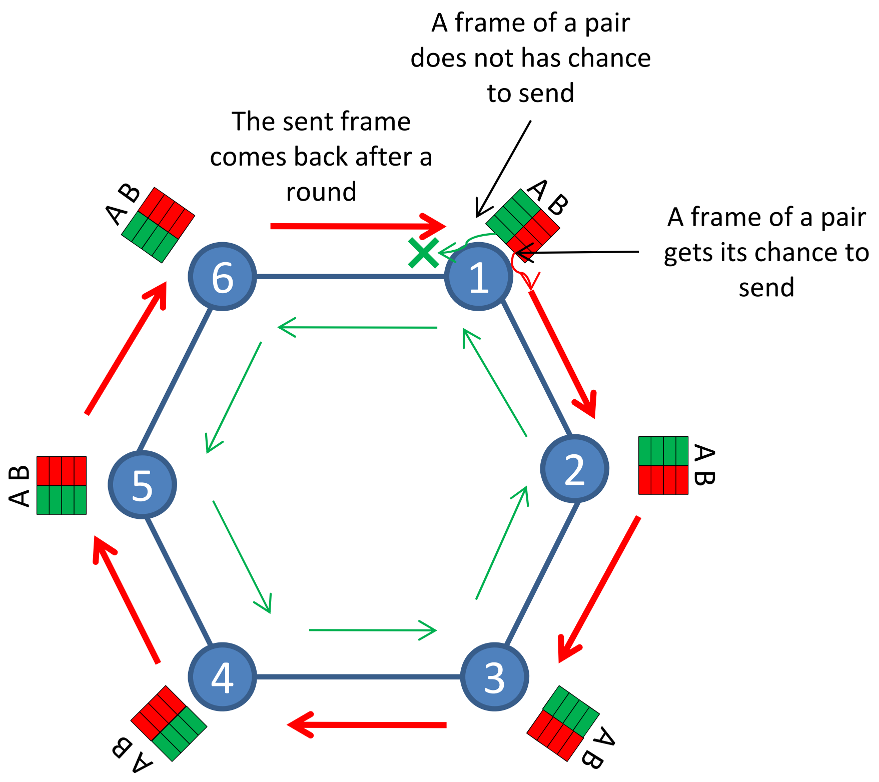

A frame of a pair does not has chance to send
The sent frame comes back after a round
A frame of a pair gets its chance to send
A B
1
2
3
4
5
6

Figure 4.33 The frame in sending queue B gets chance to send, whereas the frame in sending queue B has to wait for a long period such that the entry of frame sent at line B are all overwritten

Now, the frame in the sending queue A has to wait for such a long time that the entries of the frame sent on line B is overwritten in the table of all nodes, until it is sent. The frame sent on line A thus also travels through the ring and discarded when it returns Node1.

In such a way, the rejection ratio can dramatically degrade and the number of discarded frames, the number of frames in the sending queue increases dramatically.

In Figure 4.34 and Figure 4.35 the percentage of discarded frames and the number of frames in the sending queue are presented for the methods circular buffer, hash table with aging and linear probing, hash table with aging and quadratic probing, hash table with aging and double hashing and hash table combined with circular buffer. The results confirm the above considerations.

It can be observed in Figure 4.29 that the discarded frame ratio begin to rise at the node number of 60. According to equation 4.1 the frame generation period is supposed to be

$$T_{frame\_generation} = 138 * 80ns * 60 / 2 = 331us > 250us$$

, which verifies the consideration stated before.

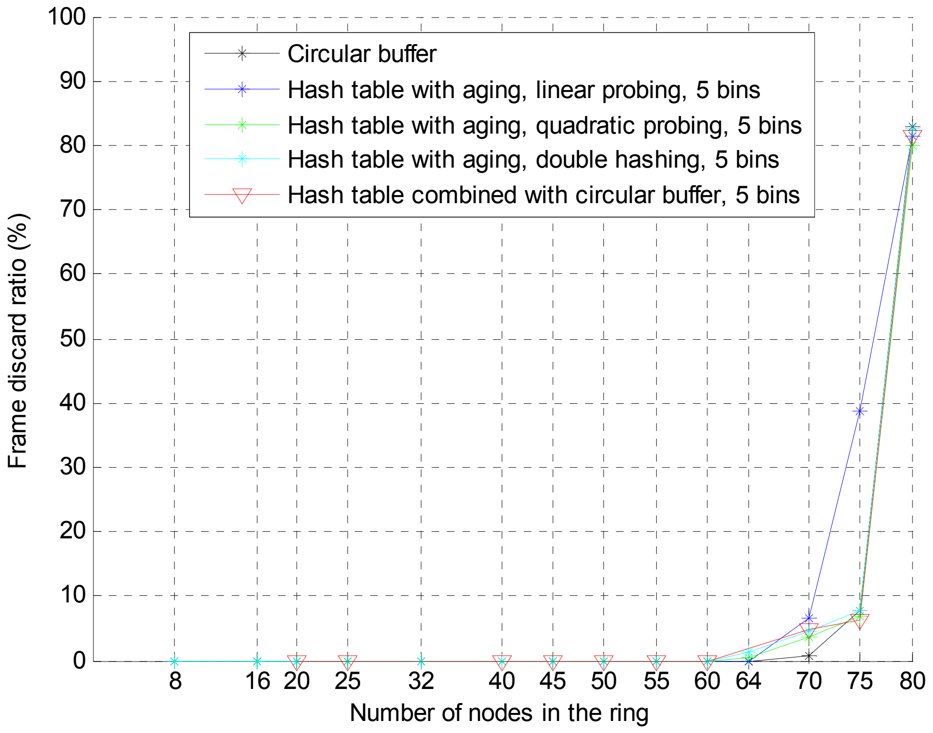


Figure 4.34 The discarded frame ratio

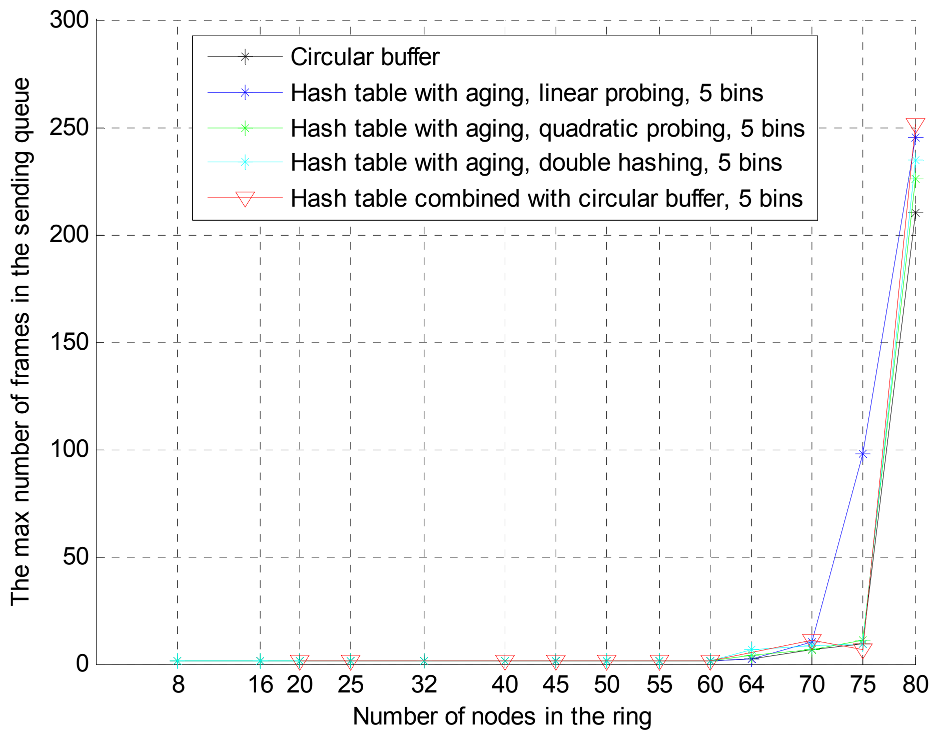


Figure 4.35 The max number of frames in the sending queue

4.4 The Chosen Algorithm and Modification

In this section, an algorithm is chosen for the implementation of the switch element. Some modification is done for the chosen algorithm to achieve better performance.

4.4.1 Choosing an Algorithm for Implementation

As shown in Figure 4.30, the multicast frame rejection ratio of the circular buffer, hash table with aging and quadratic probing, hash table with aging and linear probing and hash table combined with circular buffer are the same. Whereas the Figure 4.31 shows that the circular buffer has the highest unicast rejection ratio; hash table with aging and double hashing has the lowest rejection ratio; the rejection ratios of hash table with aging and quadratic probing, hash table with aging and linear probing and hash table combined with circular buffer are in between.

Although the circular buffer has the best performance, one should go through the entire table to verify if an entry is in table or not. As stated in chapter 3, the searching time is too long. The hash table with aging requires complex circuit implementation because its aging mechanism and memory access serialization among the aging, searching and writing process. Hash table combined with circular buffer achieves almost the same good performance as the hash table with aging, but requires only much simpler circuit implementation, therefore, the hash table combined with circular buffer is chosen to implement.

4.4.2 Modification before Implementation

Figure 4.26 shows that unicast frame rejection ratio of hash table combined with circular buffer is slightly worse than the rejection ratio of hash table with aging and quadratic probing, hash table with aging and linear probing. To improve the rejection ratio of the unicast frames, two hash tables are used for unicast and multicast frames separately instead of one. There are three advantages by doing this

1) The entry of the unicast frame and multicast frame will not overwritten by each other.
2) Only the source address and the sequence number of the incoming frame need to be hashed, because the destination address of the frame in the table for unicast frames must be the MAC address of this node and in the table for multicast frames the destination address must be the multicast address.
3) The entry in both tables is reduced from DA, SA, SEQ to SA, SEQ, the reason is the same as above. More space in the memory can therefore be saved.

Because the bin number of the hash table combined with circular buffer must divide the hash table size and the *mod* operation is much simpler if the table size is a number of power of 2, the bin number and the table size are both chosen to a number of power of 2.

In the implementation, the size of the table for unicast is chosen to 64, the size of table for multicast is chosen to 32 and the size of table for circular frame is chosen to 64. The *max_bin* number is chosen to 4 for all tables.

Simulation is performed again to verify the performance of the modified algorithm. The result of multicast frame rejection ratio is shown in Figure 4.36, the result of unicast frame rejection ratio is shown in Figure 4.37..

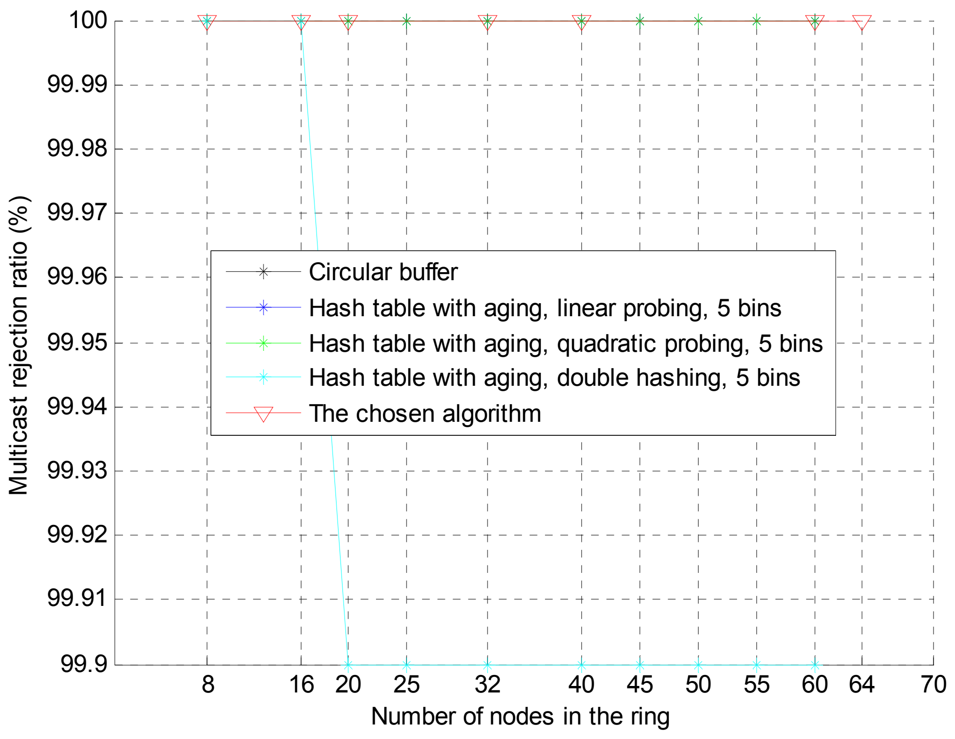


Figure 4.36 Comparison of the rejection ratio of the multicast frames between the chosen algorithm and the other proposed algorithms

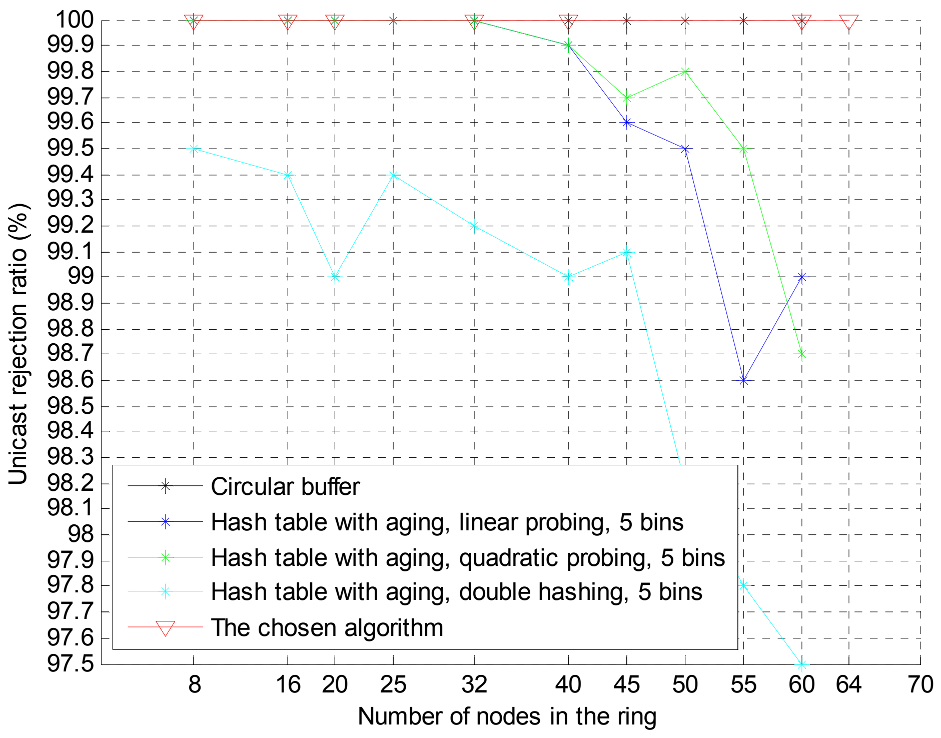


Figure 4.37 Comparison of the rejection ratio of unicast frame between the chosen algorithm and the other proposed algorithms

It is shown that both the rejection ratio of the multi- and unicast frame of the chosen algorithm have reached 100% in the node number range of 0 to 64.

5 FPGA Implementation

In this chapter, the HSR protocol introduced in Chapter 2 together with the chosen algorithm for rejecting the duplicated and circular frames in Chapter 4 is implemented in FPGA. The hardware platform, the structure of the design and the pin designation are introduced. The functionality of each component is shortly described. The sending flow, receiving flow and forwarding flow are illustrated. Several important design issues are explained. The design is synthesized in the Quartus design environment. After synthesis, the post-synthesis simulation is performed in ModelSim to evaluate all the functionalities.

5.1 The Hardware Platform and Interface

A multi-channel Ethernet interface is being developed at ABB, which can implement the HSR protocol. One configuration of this interface is shown in Figure 5.1.

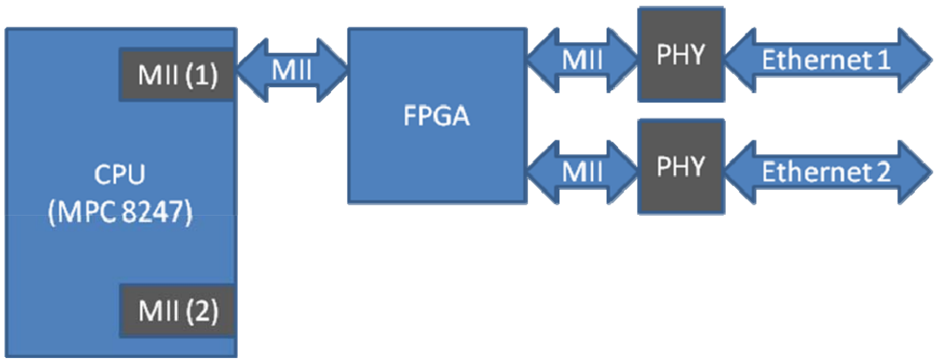


Figure 5.1 One possible configuration of the Ethernet interface on [21]

The CPU used here is the MPC8247 CPU from Freescale Semiconductor. It has two MII interfaces. The FPGA is configured to have 3 MII interfaces, one is connected to MII in the CPU and the other two are connected to two PHYs. With such a configuration, the switch element implemented in the FPGA can receive a frame from the CPU, duplicate it and send the two frames through the two MII interfaces to two PHYs which are connected to the ring. The switching of the frame received from Ethernet1 or Ethernet2 are performed by the switch element in the FPGA. Only a received frame dedicated to the receiving node is passed to the CPU.

5.2 Design Process and Methodology

Figure 5.2 shows the FPGA design process and methodology. The entry of this process is a VHDL file which describes the functionality of the circuit.

The Register Transfer Level (RTL) functional simulation verifies only the functionality, the timing information of the real components on the device is not included.

The Synthesis maps the high level functional description of the VHDL file to the gate level structure which can be realized on the Device. The Synthesis tool can generate a netlist of the real components on the device and also the timing information of these components. After synthesis, the Post-synthesis simulation is performed. The Post-synthesis simulation simulates not only the functionality of the design but also its timing behavior.

In “place and route”, the layout is performed by the device layout tools. This process generates a netlist and the timing delay file in Standard Delay Format (SDF). This file contains not only the timing information of the components but also the timing information of the layout.

The simulation after "Place and Route" is thus closest to the actual hardware[22]. In this thesis the process has been gone through till the post-synthesis simulation, the simulation after "place and route" and the hardware implementation is not performed due to the time constraints of the thesis.

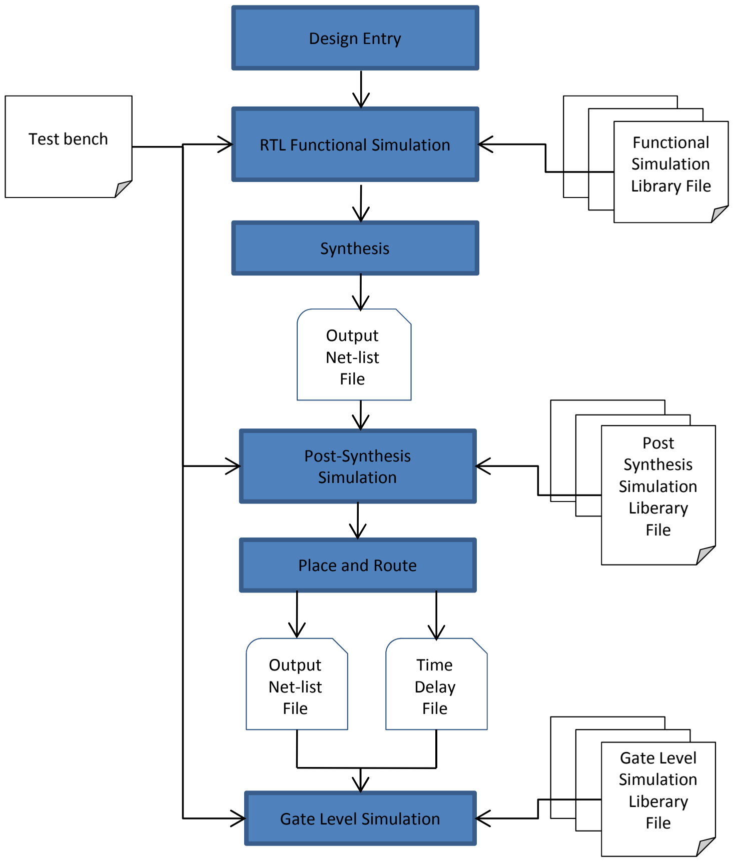

Design Entry
Test bench
RTL Functional Simulation
Functional Simulation Library File
Synthesis
Output Net-list File
Post-Synthesis Simulation
Post Synthesis Simulation Liberary File
Place and Route
Output Net-list File
Time Delay File
Gate Level Simulation Liberary File
Gate Level Simulation

Figure 5.2 The FPGA design process and methodology[23]

5.3 Functional Design

In this section, the functionality and the design considerations of the each component will be explained in detail. The basic technique used in the design is the Finite State Machine (FSM). Almost all the control blocks are implemented with FSM.

5.3.1 An Overview of the Switch Element Structure

Block Diagram of the Switch Element

Figure 5.3 shows the block diagram of the switch element. The three pairs of receiving and sending ports (RX_HST, TX_HST, RX_A, TX_A, RX_B, TX_B) are all MII interface. The Frame received from the host (CPU) is pushed into the sending FIFO waiting to be sent. Frames received from the A line or B line is pushed into the Receiving FIFO by the receiving main routine. The decision of discarding duplicate frames and the switch decision (forward, accept or discard) is made by the receiving processing unit after searching the hash tables. Frames to be forwarded are pushed into the forwarding FIFO, frames to be accepted are pushed into the accepting FIFO. The forwarding and sending are coordinated by the sending coordinator, the sending of the accepted frames to the host is coordinated between the two accepting FIFO by the receiving coordinator.

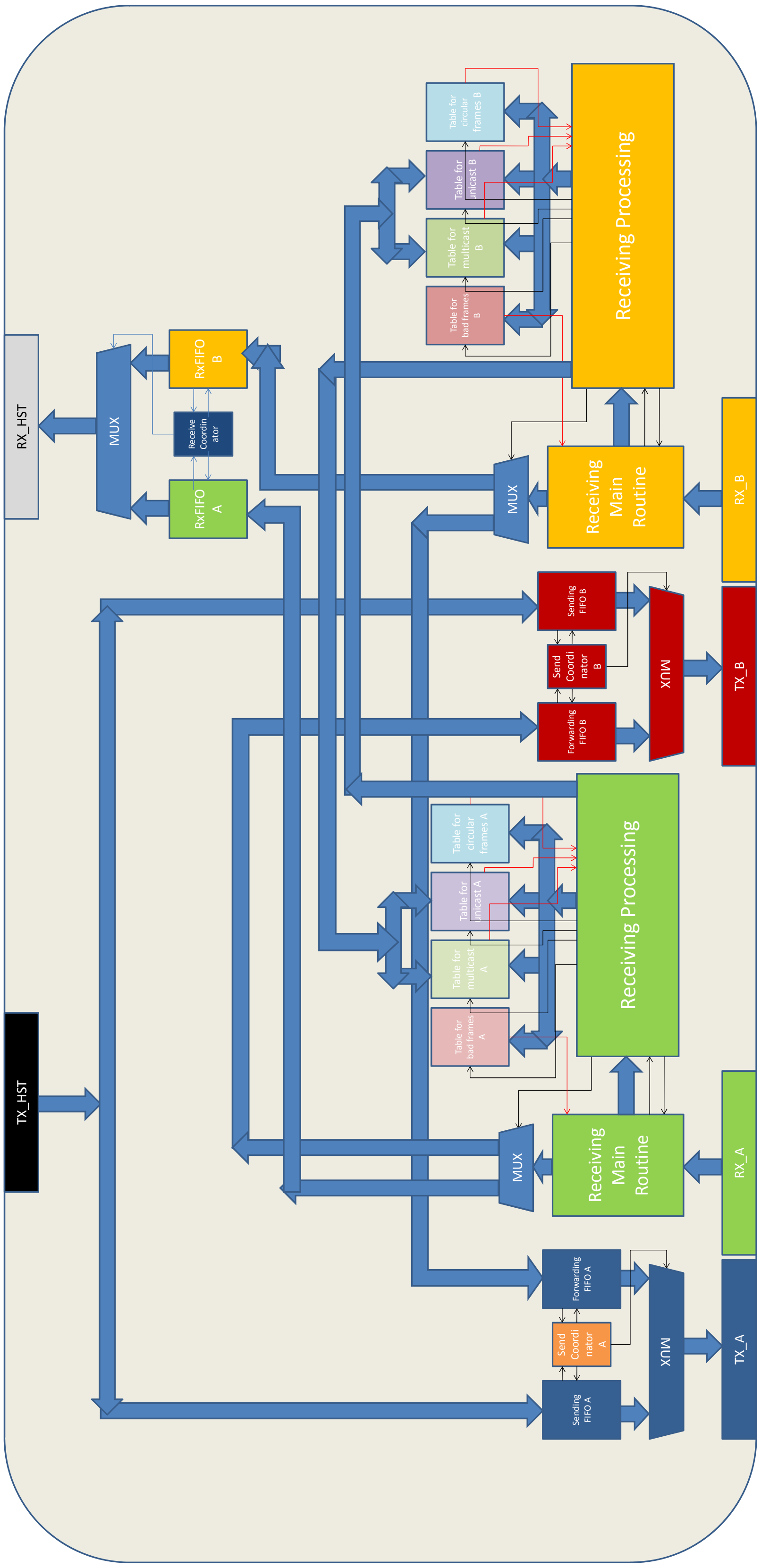


Figure 5.3 The block diagram of the switch element

Pin Definition of the Switch Element

The Pin Definition of the Switch Element	
Interface to Host	
Pin Name	Description
tx_clk_hst	TX clock provided by host
tx_data_hst[3:0]	Transmit data to switch element
tx_dv_hst	TX data valid
tx_sreset_hst	TX reset provided by host
tx_enable_hst	TX enable provided by host
tx_er_hst	TX error
rx_clk_hst	RX clock provided by host
rx_sreset_hst	RX reset provided by host
rx_enable_hst	RX enable provided by host
rx_data_hst[3:0]	Receive data from switch element
rx_dv_hst	RX data valid
rx_er_hst	RX Error
rx_crs_hst	Carrier sense
rx_col_hst	Collision
Interface to Port A	
tx_clk_a	TX clock provided by host
tx_data_a[3:0]	Transmit data to PHY at line A
tx_dv_a	TX data valid
tx_sreset_a	TX reset provided by host
tx_enable_a	TX enable provided by host
tx_er_a	TX error
rx_clk_a	RX clock provided by PHY at line A
rx_sreset_a	RX reset provided by host
rx_enable_a	RX enable provided by host
rx_data_a[3:0]	Receive data from PHY at line A
rx_dv_a	RX data valid
rx_er_a	RX Error
rx_crs_a	Carrier sense
rx_col_a	Collision
Interface to Port B	
tx_clk_b	TX clock provided by host
tx_data_b[3:0]	Transmit data to PHY at line B
tx_dv_b	TX data valid
tx_sreset_b	TX reset provided by host
tx_enable_b	TX enable provided by host
tx_er_b	TX error
rx_clk_b	RX clock provided by PHY at line B
rx_sreset_b	RX reset provided by host
rx_enable_b	RX enable provided by host
rx_data_b[3:0]	Receive data from PHY at line B
rx_dv_b	RX data valid
rx_er_b	RX Error
rx_crs_b	Carrier sense
rx_col_b	Collision

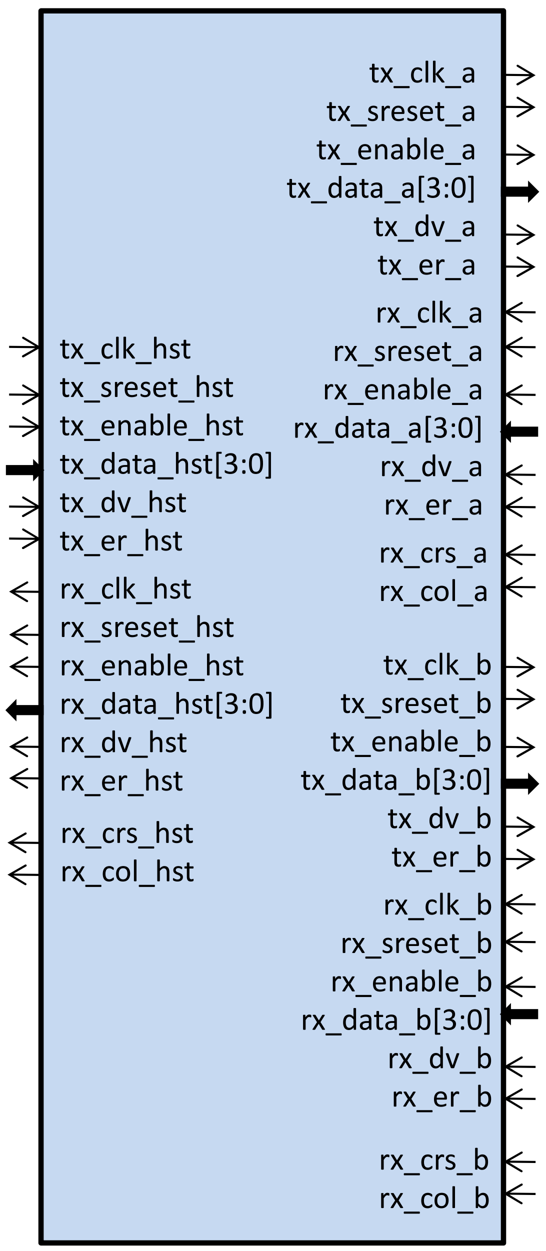


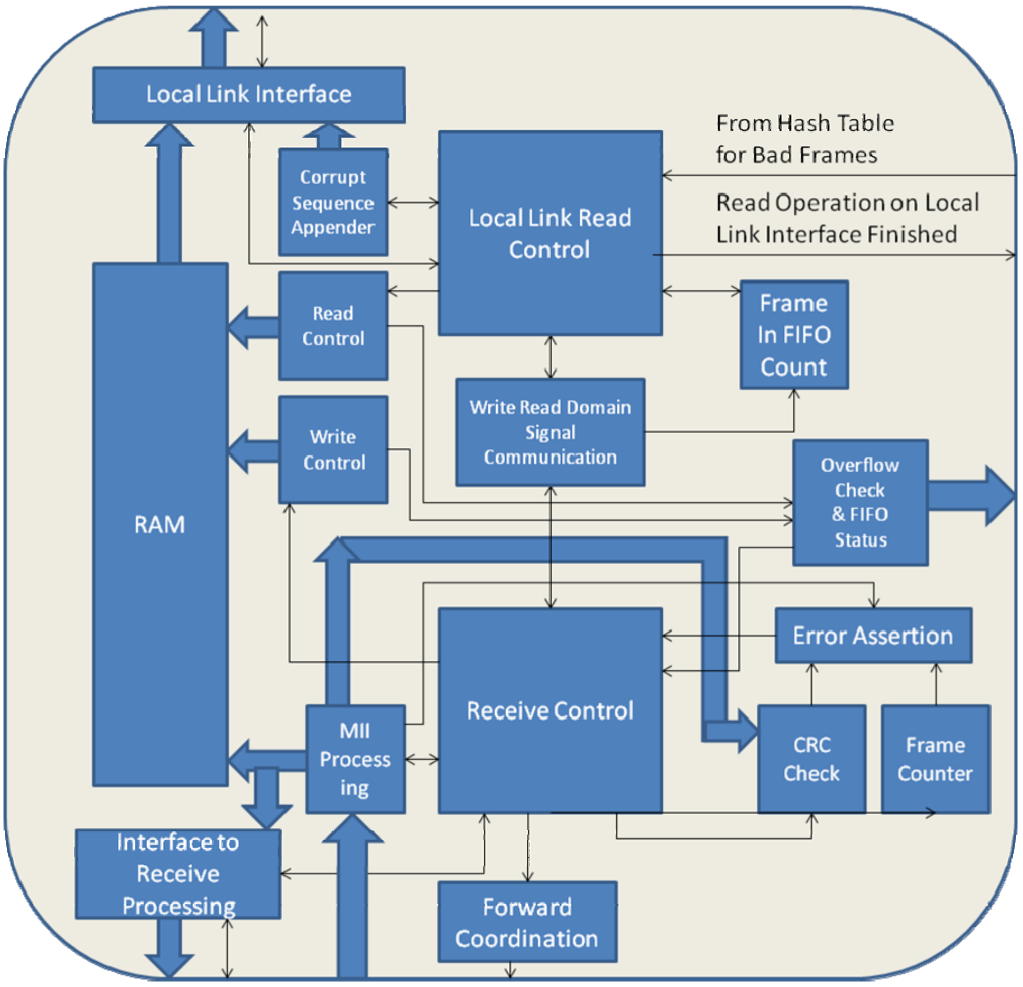
Figure 5.4 The pin definition of the switch element

Table 5.1 The pin description of the switch element

Figure 5.4 shows the pin definition of the switch element and Table 5.1 shows the pin description. The switch element designed in this thesis works only in full duplex mode, therefore the both the carrier sense pin and the collision pin from PHY are just connected to the switch element but not used inside the switch element. The rx_er_hst, rx_crs_hst and rx_col_hst signals are connected to the GND inside the switch element, which means there is no connection error, collision on the channel between the host and the switch element. The Management Data Input/Output (MDIO) interface is also not implemented in this thesis.

5.3.2 The Receiving Main Routine

The main tasks of the receiving main routine are pushing the received frame into FIFO, pass the received frame to the next unit through the local link interface, performing CRC check on the received frame, dropping the frame if the frame is found in the table for rejecting duplicated and circular frames, appending the garbling sequence when the received frame is a bad frame. The block diagram of the receiving main routine is shown in Figure 5.5. The functionality of each block is introduced shortly below.



Local Link Interface
From Hash Table for Bad Frames
Corrupt Sequence Appender
Local Link Read Control
Read Operation on Local Link Interface Finished
Read Control
Frame In FIFO Count
Write Control
Write Read Domain Signal Communication
Overflow Check & FIFO Status
RAM
Error Assertion
Receive Control
MII Process ing
CRC Check
Frame Counter
Interface to Receive Processing
Forward Coordination

Figure 5.5 The block diagram of the receiving main routine

MII Processing

The MII processing detects the rising edge of the rx_dv signal, informs the receive control block the arrival of the frame. It also transforms the 4-bits wide MII data format to 8-bits wide data format, this is used by the DPRAM block, the receive processing unit and the CRC check block. Besides, it sends the rx_er signal to the error assertion.

Frame Counter

The frame counter block counts the number of bytes of the received frame. Besides, the frame counter also tells the CRC check block when to begin the CRC check (the CRC check begins after the preamble and the frame start delimiter.

Error Assertion

The error assertion block asserts an error if either the received frame count number is smaller than 64 or greater than 1536, or the CRC check is not passed or the rx_er is set. If error is asserted, the receiving process will be stopped and the frame is dropped. The bad frame handling is introduced more in detail in the Section 2.3.5.

Overflow Functionality and FIFO Status

The overflow functionality tells if the FIFO is already full. The FIFO status tells how many percent of the FIFO has been used.

Write and Read Domain Signal Communication

The write and read domain signal communication block is responsible to synchronize the signal in one clock domain with another clock domain. For example, if bad frame is asserted in the write clock domain, the bad frame signal should be sensed by the read clock domain so that the local link read control can interrupt the reading process and append the garbling sequence. The transferring of signals from one clock to another clock domain will be explained in more detail in section synchronization between clock domains.

Frame in FIFO Count

The frame in FIFO count block counts the number of frames in the FIFO. It can inform the local link read control to begin the read process.

Read Address Control and Write Address Control

The read address control and the write address control manage the read address and the write address of the DPRAM. They store also the read and write address of last time a good frame received as the start address of this time. If a bad frame is received, the write and read address are reseted to the start address of this time.

Garbling Sequence Appender

The garbling sequence appender appends the garbling sequence at the end of a bad frame.

Local Link Interface

The frame stored in the FIFO is passed to the RX_FIFO or Forwarding FIFO through the local link interface.

Forward Coordination

The forward coordination asserts the signal *take_sending_port* and sends it to the send coordinator to inform that a frame is being received. The frame in the TX_FIFO on the other line should wait if the frame being received is going to be forwarded to the sending port of the other line. The *take_sending_port* signal can be deasserted by the Receiving Processing if this frame is a unicast frame dedicated to this node or the entry of this frame is found in the hash table for rejection duplicates or the frame should be discarded.

Interface to Receiving Processing

The interface to receiving processing informs the Receiving Processing unit when to begin loading the source address, destination address, and sequence number. It also provides the received data after the format transformation to the Receiving Processing unit. One the other hand, it passes the control signal from the Receiving Processing. For example, if the entry of a frame is found in the table for rejecting the duplicated frames, the Receiving Processing unit will send a signal back to the Receiving Main Routine, so that the receiving process is stopped and the frame is dropped.

Receive Control and Local Link Read Control

The receive control and the local link read control are two main control units which coordinate between different blocks.

In this unit, the receive control, lock link read control, CRC check, Overflow functionality are implemented with FSM.

5.3.3 The Receiving Processing

The tasks of the receiving processing are loading the source address, the destination address, and sequence number of the received frame, making the switch decision (forwarding, accepting and discarding), providing the entry for the hash tables, randomizing the key for the hash function. Figure 5.6 shows the block diagram of the Receiving Processing Unit. The functionality of each block is introduced shortly below.

Interface to the Write Domain of Receiving Main Routine

The source address, the destination address and the sequence number and the load enable signal are passed from the receiving main routine through this interface. The drop frame command is sent to the receiving main routine through this interface.

Hash Key Randomization Functionality

The hash key randomization function *Hashpjw()* is implemented here. The calculation of the key is done byte by byte every clock. The calculation of the hash key for the circular frame begins with destination address and ends with the sequence number. The calculation of the hash key for the unicast and multicast frame begins with the source address and ends with sequence number. After the calculation is done, the randomized keys are registered to the “Interface to Hash Table for Multicast, Unicast, Circular, Bad Frame”.

Interface to Hash Table for Multicast, Unicast, Circular, Bad Frame

The entry of the frame in the hash table for circular frame which consists of destination address, source address and sequence number, the entry of the frame in the hash table for unicast and multicast which consists of the source address and sequence number, are registered to this interface. The randomized hash keys are also registered to this Interface. The signal to start searching entry in the hash table is passed to the hash tables through this interface. The *entry_found* and *entry_not_found* signals are sent back by the hash table for unicast, multicast and circular frames respectively. If at least one entry is found in the three tables, the receiving process is stopped, and frame is dropped. Only when entry is not found in all the three tables, the receiving process continues.

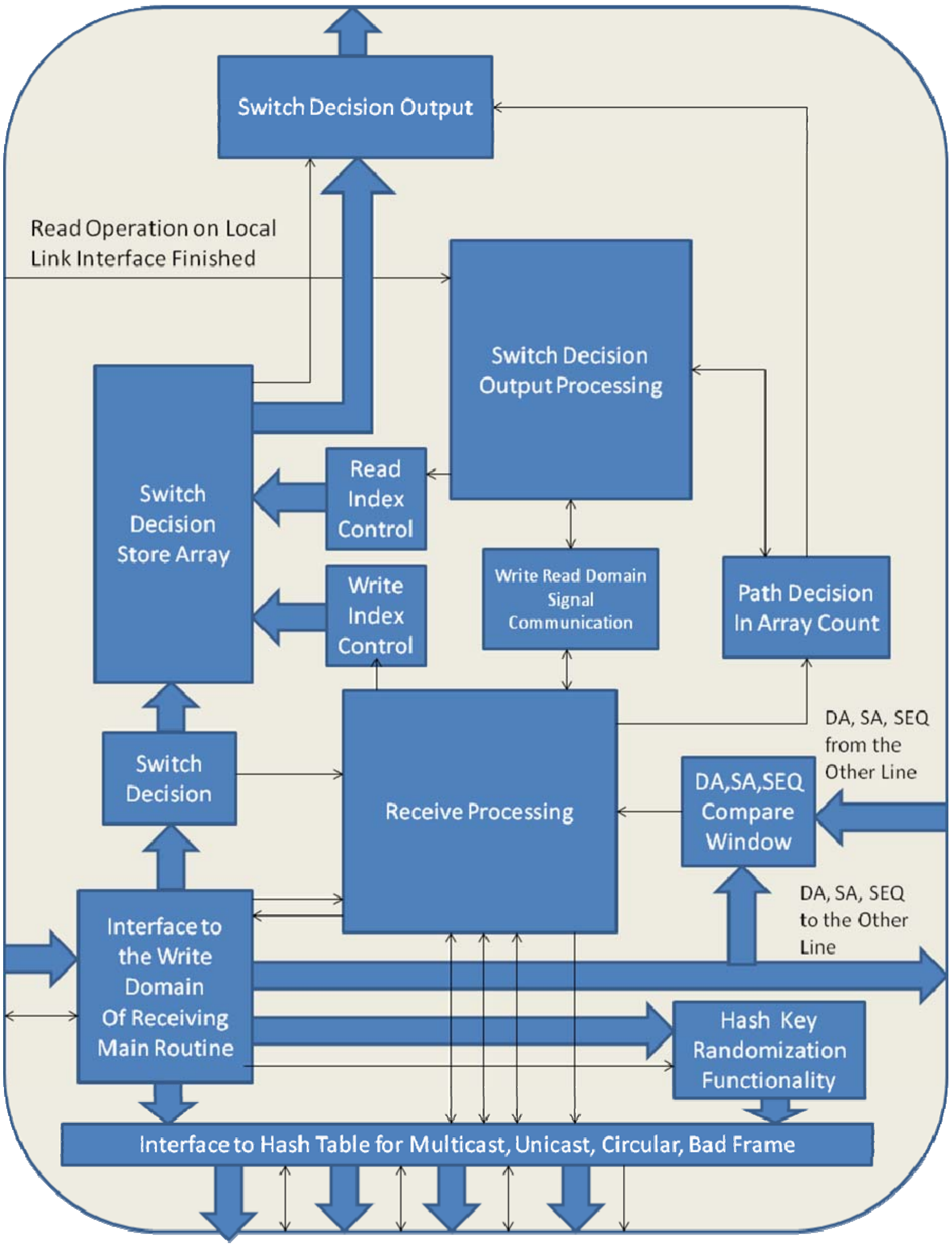

Switch Decision Output
Read Operation on Local Link Interface Finished
Switch Decision Output Processing
Switch Decision Store Array
Read Index Control
Write Index Control
Write Read Domain Signal Communication
Path Decision In Array Count
DA, SA, SEQ from the Other Line
Switch Decision
Receive Processing
DA,SA,SEQ Compare Window
DA, SA, SEQ to the Other Line
Interface to the Write Domain Of Receiving Main Routine
Hash Key Randomization Functionality
Interface to Hash Table for Multicast, Unicast, Circular, Bad Frame

Figure 5.6 The block diagram of the receiving processing

DA, SA, SEQ Compare Window

This block can handle the situation where two same frames are received at two lines at the same time. The way how it works will be explained in more detail in section DA, SA, SEQ compare window.

Switch Decision

The switch decision block makes the switch decision by comparing the received destination address and source address.

a) *Discard* when the source address of the received frame = MAC address

b) *Forward* when the destination address of the received frame != MAC address

&& the source address of the received frame != MAC address

c) *Accept* when the destination address of the received frame = MAC address

&& the source address of the received frame != MAC address

d) *Forward and Accept* when the destination address of the received frame = multicast address && the source address of the received frame != MAC address

e) *No Frame* if there is no frame, the switch decision is NULL, which does not allow any operation on the local link interface.

If the unicast frame is dedicated to this node, the receiving processing will send a signal to the forward coordination block in the receiving main routine to free the occupation of the sending port on the other line.

Switch Decision Store Array

If there are more than one frame in the FIFO of the receiving main routine, the corresponding switch decision should also be stored in the switch decision array, so that when the frame is popped from the FIFO, its corresponding switch decision is also popped. In the correct designed system there will be no more than one frame in the FIFO, this function is preserved for possible future use.

Path Decision in Array Count

This block count the number of switch decisions stored in the array. If the number is not equal to 0, the switch decision output processing should be started.

Read Operation on the Local Link Interface Finished

This signal is used to synchronize the local link read control in the receiving main routine and the switch decision output processing. If the read operation on local link interface is finished, this signal informs the switch decision output processing to register the new switch decision.

Switch Decision Output

The switch decision in the switch decision array is registered to the switch decision output. The decision is used to switch the MUX connected to the local link interface of the receiving main routine.

The block write index control, read index control and the write read domain signal communication is the same as in the receiving main routine and therefore not repeated here.

5.3.4 Hash Table

The block diagram of the hash table is shown in Figure 5.7. The entry, the randomized hash key and the signal to start searching are passed from the interface to receiving processing. The positions of the write bin and read bin in each bucket are stored in the bin store array. The content read from the DPRAM is compared with the entry input on the interface to receiving processing. If they are equal, the *entry_found* signal is asserted and sent back to the interface to receiving process. If the max_bin is reach and the entry is still not been found, the entry_not_found signal is asserted and sent back to the interface to receiving g process.

The write process is very simple. The entry to be written is presented on the interface to receiving processing of this line. If the write enable is set to “high”, the entry is written into RAM on the next rising edge of the clock at the position indicated by the write bin.

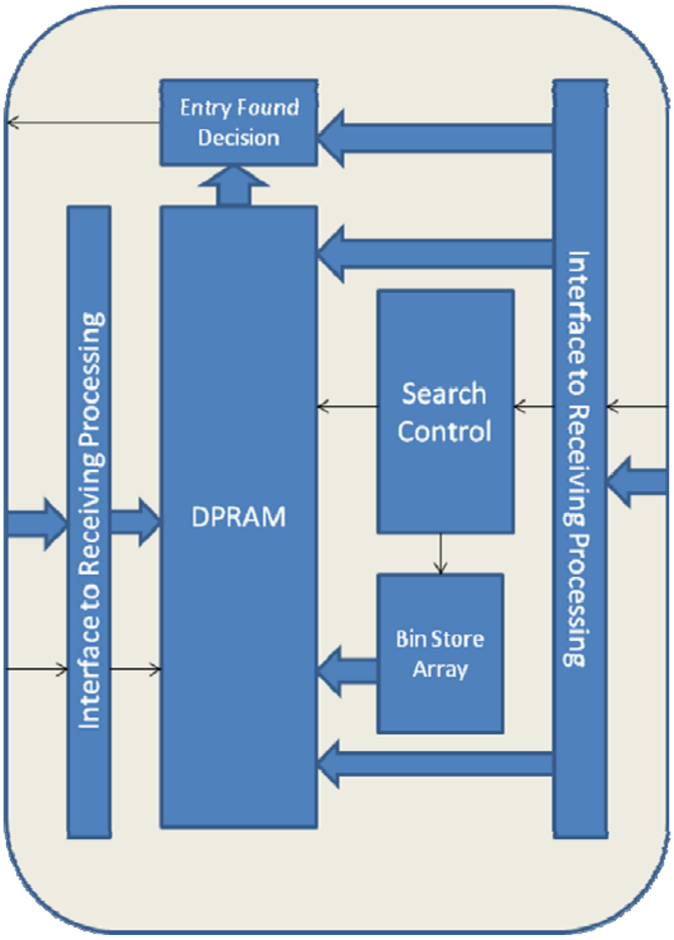

Entry Found Decision
Interface to Receiving Processing
DPRAM
Search Control
Bin Store Array
Interface to Receiving Processing

Figure 5.7 The block diagram of the hash table

5.3.5 Forwarding FIFO

The forwarding FIFO receives the frame passed by the receiving main routine through the local link interface, and send it without any modification through the MII interface to the PHY.

MII Processing

The MII processing block here transfers the 8-bits width data to 4 bits width data.

Forward Coordination

The Forward Coordination block sends signal to inform the sending coordinator that the forwarding process wants to take control of the sending port.

The functions of other blocks are the same as in the receiving main routine and will not be repeated here. Figure 5.8 shows the block diagram of the forwarding FIFO.

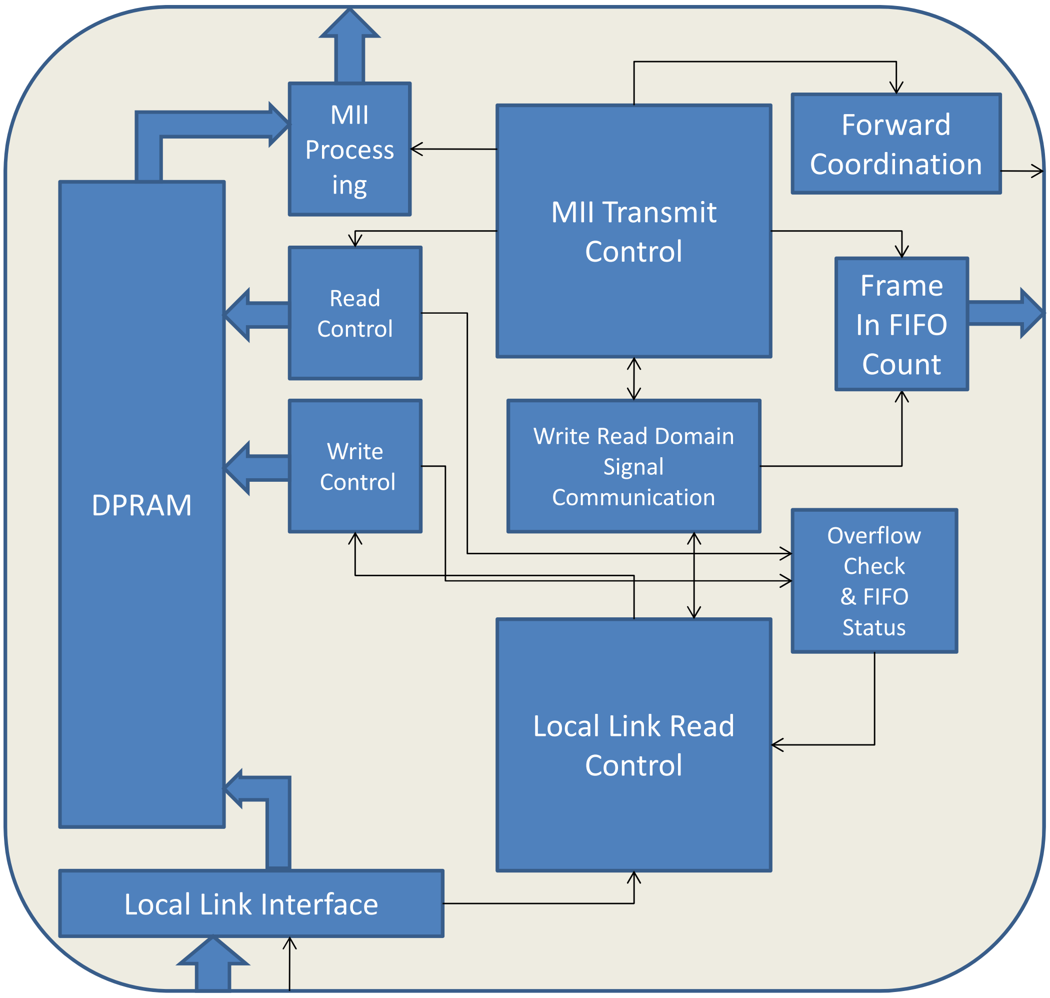


Figure 5.8 The block diagram of the forwarding FIFO

5.3.6 RX_FIFO

The RX_FIFO receives the frame sent by the receiving main routine and send it towards the MII interface to the host. The block diagram of the RX_FIFO is same as the It has a unit called garbling sequence detector which can detect the corrupt sequence. If a corrupt sequence has been detected, the frame is dropped. Figure 5.9 shows the block diagram of the forwarding FIFO.

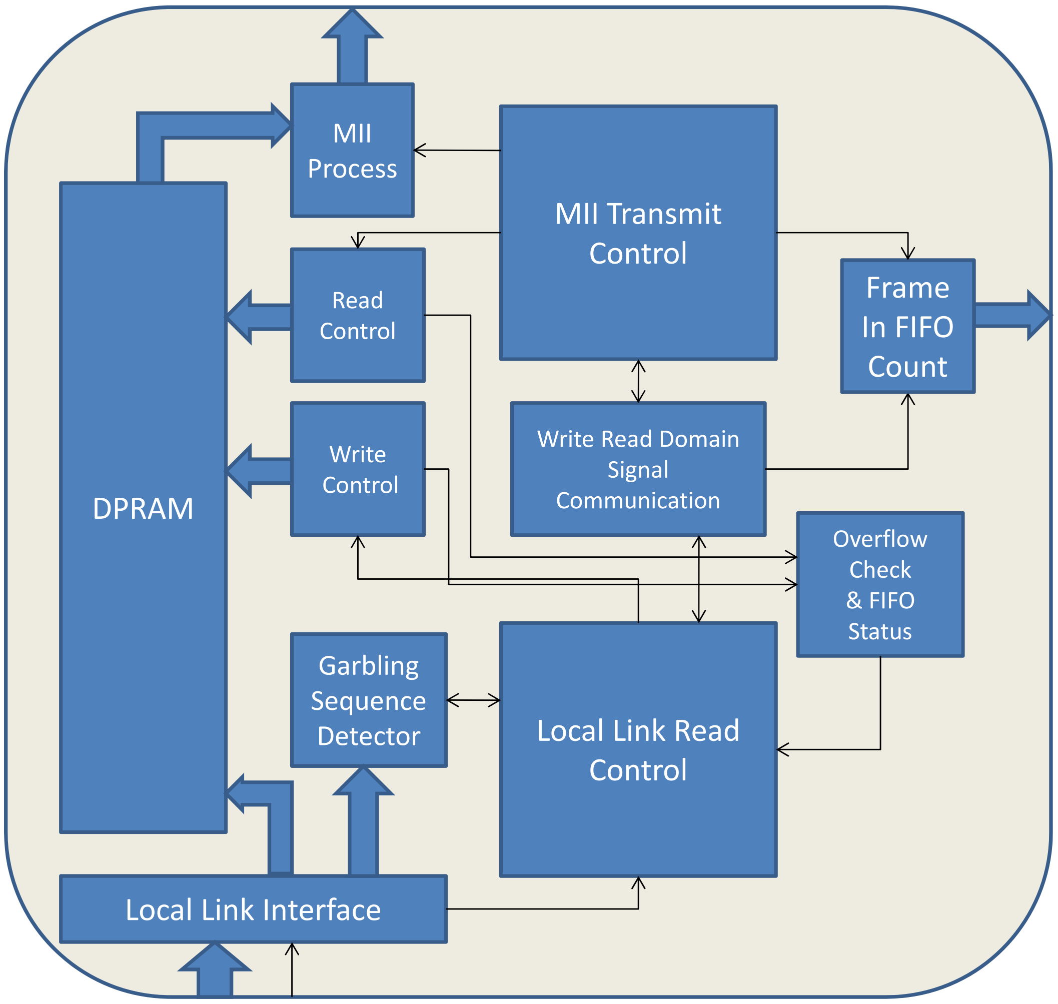


Figure 5.9 The block diagram of the RX_FIFO

5.3.7 TX_FIFO

The TX_FIFO receives frames from the MII Interface to the host, performs the CRC check and sends the frame over the MII interface to the PHY. Figure 5.10 shows the block diagram of the TX_FIFO.

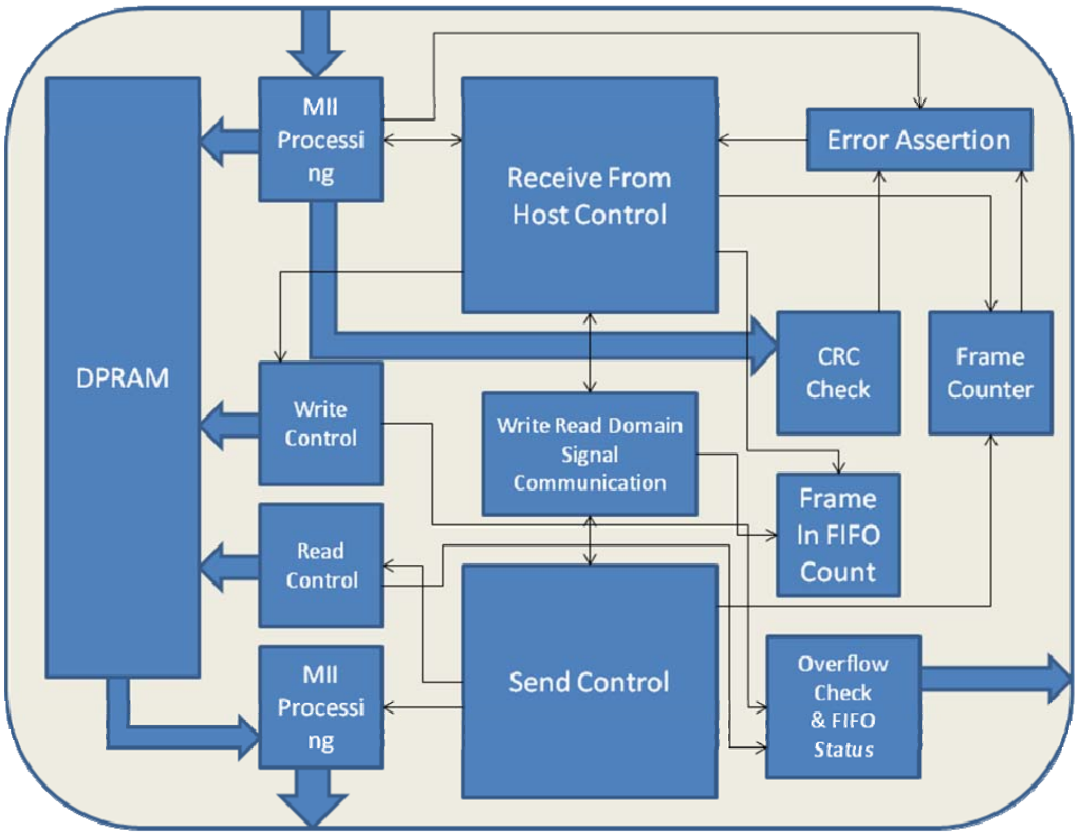


Figure 5.10 The block diagram of the TX_FIFO

5.3.8 Send Coordinator

The send coordinator receives the t*ake_port_control* acquirement from the forwarding FIFO and the receiving main routine, and coordinates the sending FIFO and forwarding FIFO according the "forwarding always has priority principle". Its state transfer is shown in Figure 5.11.

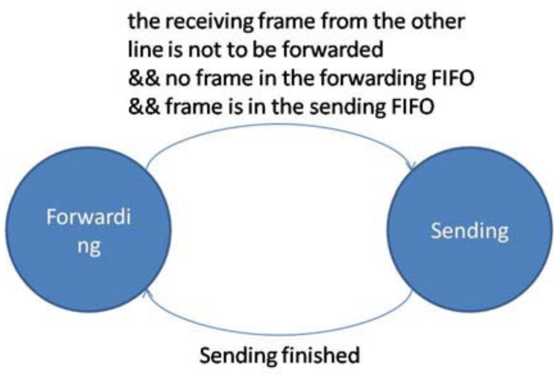

the receiving frame from the other
line is not to be forwarded
&& no frame in the forwarding FIFO
&& frame is in the sending FIFO
Forwardi ng
Sending
Sending finished

Figure 5.11 The state transfer of the sending coordinator

5.3.9 Receive Coordinator

Figure 5.12 shows the state transfer of the receiving coordinator. The receive coordinator coordinates the receiving of the frames between the RX_FIFO on line A and line B. The priority of frame on both lines is equal, except that at the start of the system the frame on A has higher priority if two frames are received at the same time

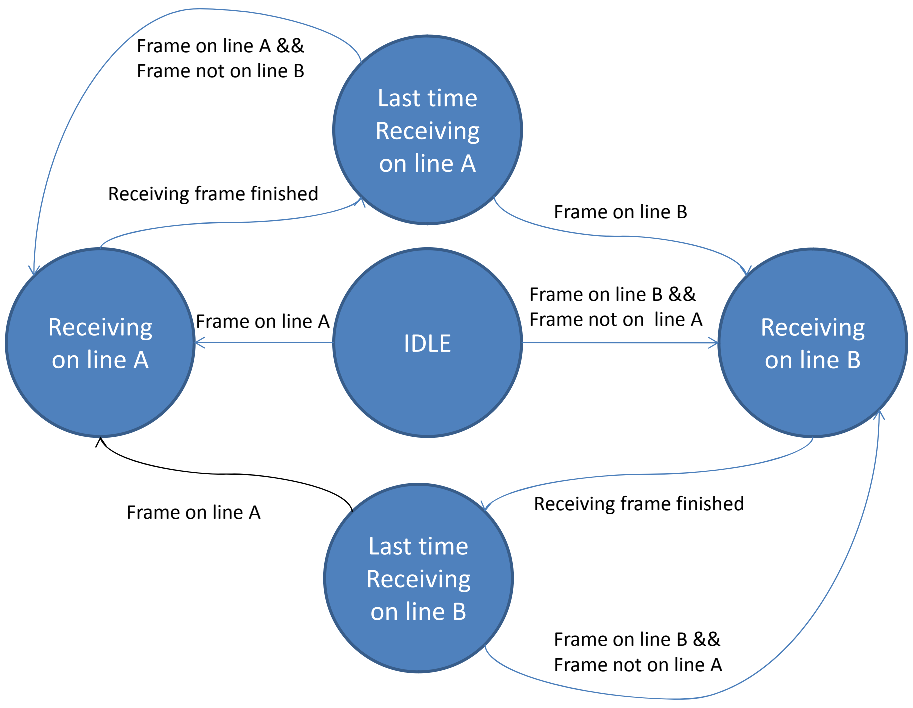


Figure 5.12 The state transfer of the receiving coordinator

5.3.10 Receiving Flow

Figure 5.13 shows the receiving flow of a frame. When a frame arrives, the receiving main routine pushes the frame into the FIFO, and performs CRC check and overflow check. The receive processing unit make the switch decision and start searching the entry of the frame in the hash table for duplicates frames (unicast and multicast) and circular frames. The hash tables return the result of the searching. The frame is dropped when it does not pass the CRC check, or overflow happens, or its entry is found in either of the hash tables.

According to the switch decision, the frame is popped from the FIFO in the receiving main routine and pushed into the forwarding FIFO or RX_FIFO or both if the frame is a multicast frame.

The frame in the RX_FIFO is then sent through the MII interface to the host, the frame in the forwarding FIFO is sent through the MII interface to the PHY.

5.3.11 Sending Flow

Figure 5.14 shows the sending flow of a frame. The sending is only allowed if there no frame in the forwarding FIFO or the frame being received by the receiving main routine is a unicast frame dedicated this node.

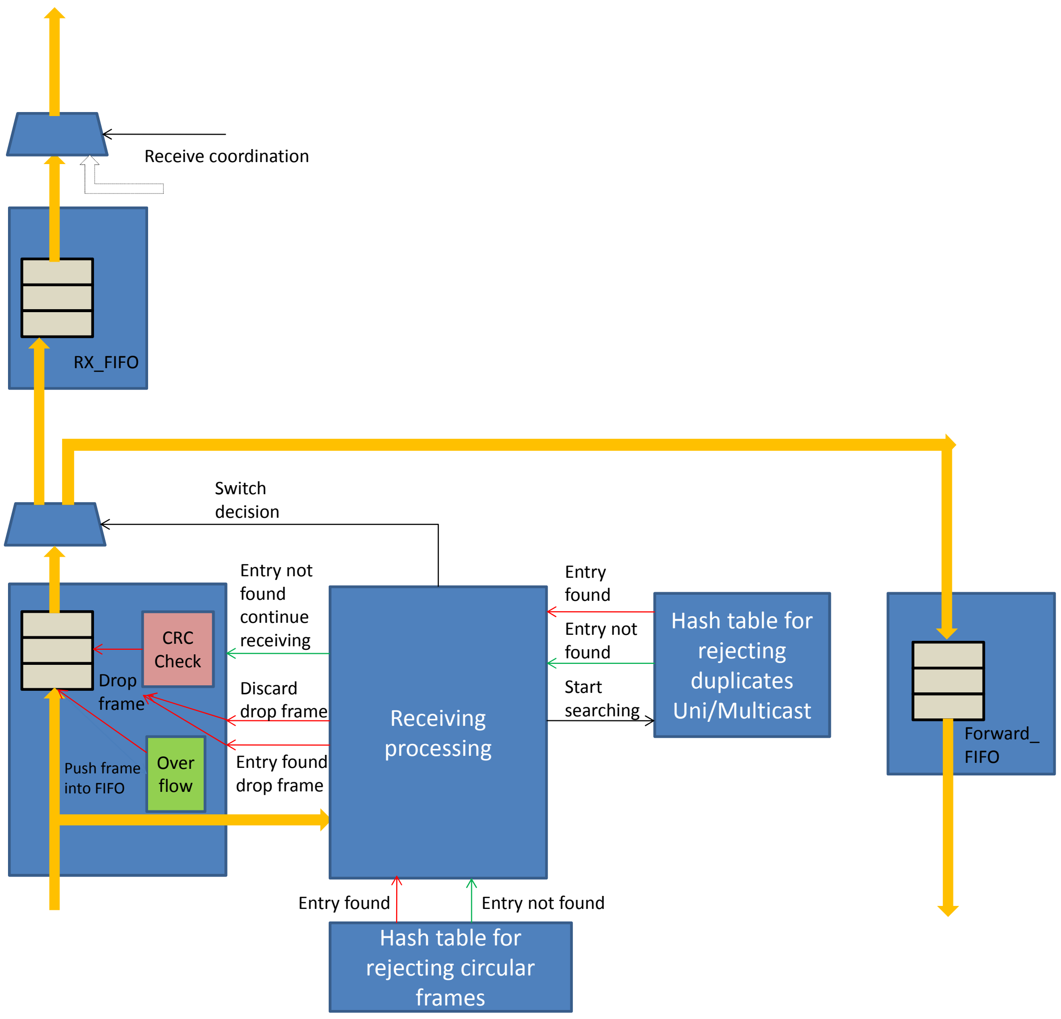


Figure 5.13 The receiving flow of a frame

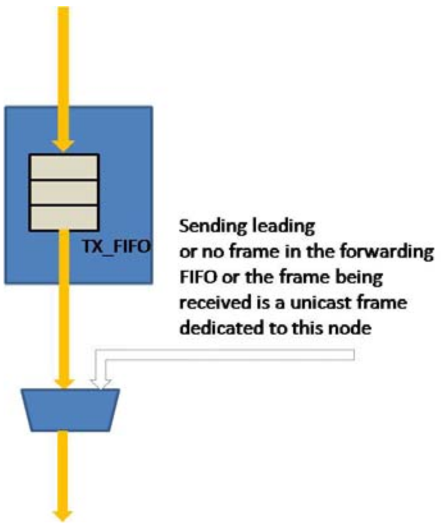

TX_FIFO
Sending leading or no frame in the forwarding FIFO or the frame being received is a unicast frame dedicated to this node

Figure 5.14 The sending flow of a frame

5.4 Functional Design Considerations

In this section, several important design considerations are introduced. The way how they are implemented is explained.

5.4.1 Cut-through Implementation

When a frame arrives at the receiving port, its destination address, source address and the sequence number are loaded to the receiving processing component. The receiving processing component searches the entry of this frame. The switch decision is only registered out when the entry of this frame is not found in the hash tables. After the switch decision is registered out, the receiving processing will assert the *switch_ready* signal to the local link read process block in the receiving main routine. The local link read process begins to send the frame to the forwarding FIFO or RX_FIFO after it sees that the *switch_ready* signal is set "high". After the sending is finished, the local link read process send a signal to tell the receiving processing that the sending is finished. After the receiving processing sees this signal, the *switch_ready* signal is deasserted. Figure 5.15 show the time flow of this process.

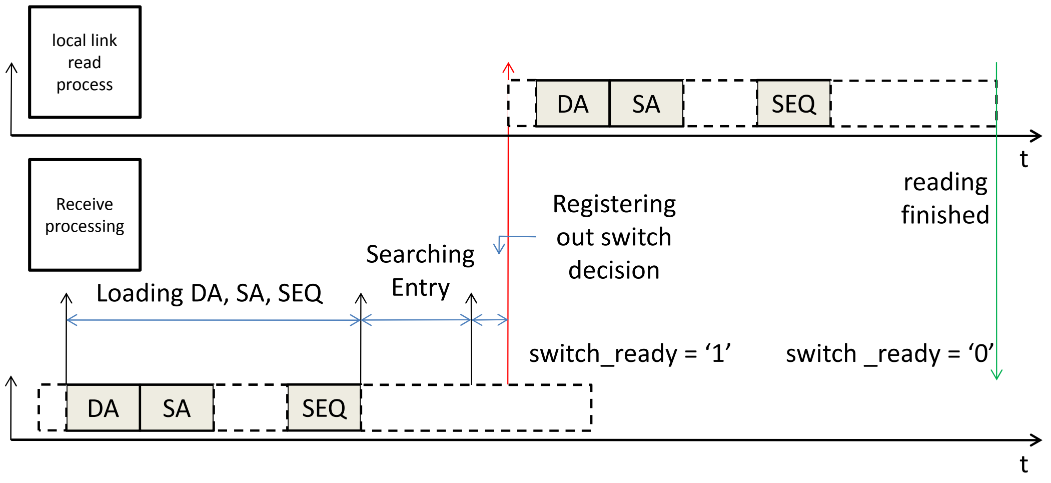


Figure 5.15 An illustration of cut-through implementation

In the forwarding FIFO, after the first byte of the frame passed by the receiving main routine is stored in the RAM, the *frame_in_fifo* asserted and transmitted to the MII transmit control. The MII transmit control begins transmitting the frame through the MII interface the PHY. After the transmitting is finished, the *frame_in_fifo* signal is deasserted.

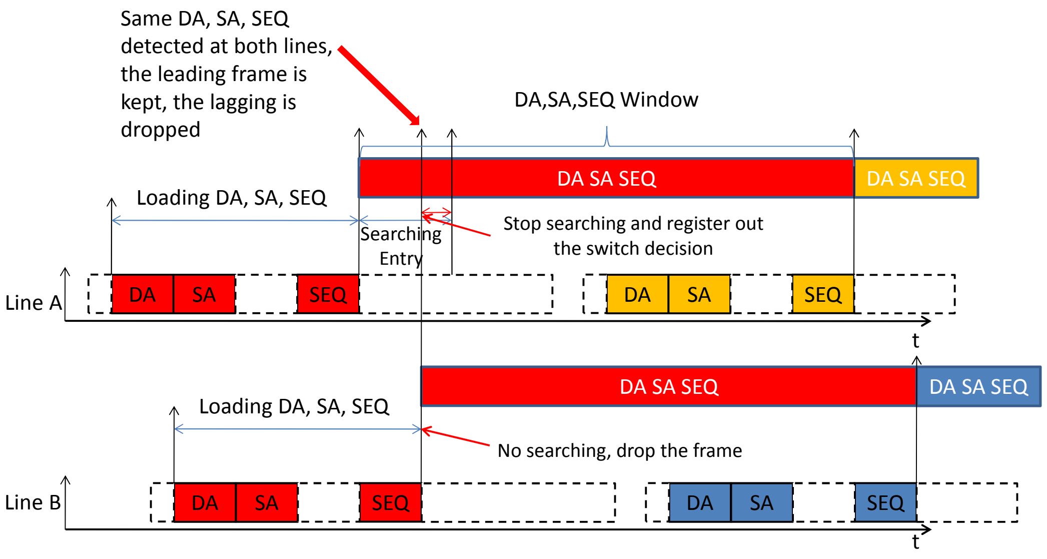
5.4.2 DA, SA, SEQ Comparison Window

In order to find out whether the frame has been received once or forwarded once, the entry of the frame is searched in the hash table. It only works when one frame of a pair comes after the other, so that the frame coming first will be registered in the table. But if two frames come at the same time, there is still no entry in the table, so both frames are accepted. To overcome this problem, a window should be built. Figure 5.16 illustrates the way how it works.

If the first frame of a pair arrives at Line A, the DA,SA,SEQ window is set up after the destination address, source address and sequence number is received. This window lasts until the destination address, the source address and the sequence number of the next frame are received. If the second frame of a pair comes later at Line B, its DA,SA,SEQ window is also build after its destination address, source address and sequence number are received. At this time the window on Line A finds that the window on Line B has the same value as it, because Line A is leading, it will keep the frame. Line

B also finds that the window on line A has the same value as it, but it is lagging, so it will drop the frame. If at the time when the window of Line B is built Line A is searching the entry in the hash table, the search is stopped, because there is no need to search anymore. The switch decision is then registered out.

If the frame arrives at exactly the same time, the frame on Line A is kept, the frame on Line B is dropped.



Same DA, SA, SEQ detected at both lines, the leading frame is kept, the lagging is dropped
DA,SA,SEQ Window
DA SA SEQ
DA SA SEQ
Loading DA, SA, SEQ
Searching Entry
Stop searching and register out the switch decision
Line A
DA
SA
SEQ
DA
SA
SEQ
t
DA SA SEQ
DA SA SEQ
Loading DA, SA, SEQ
No searching, drop the frame
Line B
DA
SA
SEQ
DA
SA
SEQ
t

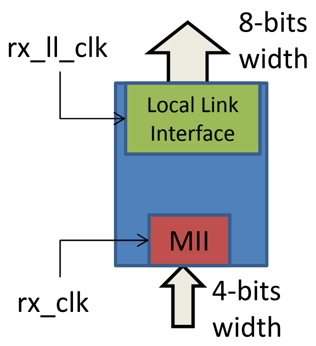
Figure 5.16 The DA, SA, SEQ comparison window

5.5 Design Considerations about Timing

In this chapter, some design considerations about timing are discussed. The solutions to overcome some timing problems are presented.

5.5.1 Clock Cooperation between Lock Link Interface and MII Interfaces

The receiving main routine has two clocks operating at different frequencies. The data width of MII interface is 4-bits. The received data is transferred to 8-bits width and stored in the DPRAM. The data on the local link interface which is read from the DPRAM is also 8-bits width. Therefore, the clock frequency of the local link interface rx_ll_clk must be half of the MII clock rx_clk. This illustrated in Figure 5.17.



$$f_{rx\_ll\_clk} = 1/2\, f_{rx\_clk}$$

Figure 5.17 The clock relation between the MII and local link interfaces

5.5.2 The Critical Clock Path

The propagation time of a signal along the clock path plays a very important role in the clock synchronized circuits. There are several basic concepts to be introduced here first[24].

- Tsu: setup time at pad, is defined as the length of time for which data that feeds a register via its data or enable input(s) must be present at an input pin before the clock signal that clocks the register is asserted at the clock pin.
- Tco: clock to output delay, is defined as the maximum time required obtaining a valid output at an output pin that is fed by a register after a clock signal transition on an input pin that clocks the register.
- Th: hold time, is defined as the minimum time required for the input signal stays stable after clock signal transition to obtain a valid output.
- Clock skew: The arriving time difference of the clock signal at clock input of different registers due to different interconnect paths and clock buffers

The above concepts are illustrated in Figure 5.18.

If a clock transition happens at register1, the output signal will arrives at the input of register 2 after Tco + Delay. It must be ensured the signal at the input of register2 is stable to generate a valid output at register2. This is equal to satisfy the following equation:

$$\mathrm{T}_{\mathrm{clk}} \geq T_{co} + T_{delay} + Tsu - T_{clkskew} \qquad (5.1)$$

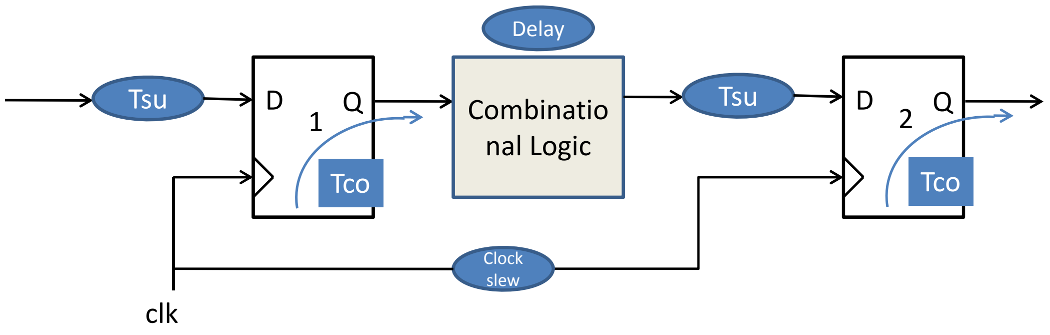
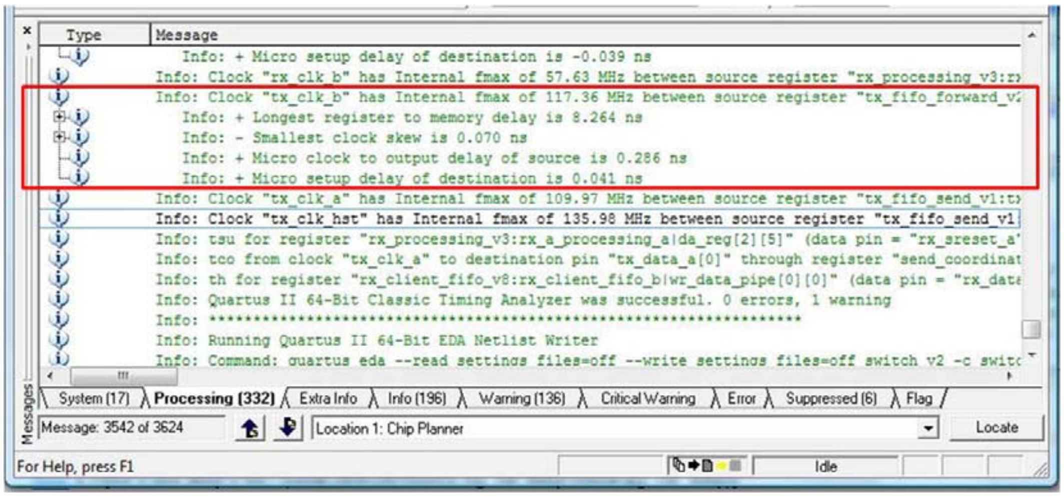


Figure 5.18 The signal propagation delay along the clock path

This equation thus also defines the maximum frequency allowed in the circuit.

$$\mathrm{f}_{\max} = \frac{1}{T_{co} + T_{delay} + Tsu - T_{clkskew}} \qquad (5.2)$$

Since it is difficult to calculate the Tco, Tdelay, Tsu and Tclkskew of all the component and paths by hand, this task is done by the synthesis tools. In this design, the slowest clock path which is called the critical clock path is twice as the MII clock frequency. This can be shown in the synthesis report below.



```
Type      Message
            Info: + Micro setup delay of destination is -0.039 ns
          Info: Clock "rx_clk_b" has Internal fmax of 57.63 MHz between source register "rx_processing_v3:r
          Info: Clock "tx_clk_b" has Internal fmax of 117.36 MHz between source register "tx_fifo_forward_v
            Info: + Longest register to memory delay is 8.264 ns
            Info: - Smallest clock skew is 0.070 ns
            Info: + Micro clock to output delay of source is 0.286 ns
            Info: + Micro setup delay of destination is 0.041 ns
          Info: Clock "tx_clk_a" has Internal fmax of 109.97 MHz between source register "tx_fifo_send_v1:t
          Info: Clock "tx_clk_hst" has Internal fmax of 135.98 MHz between source register "tx_fifo_send_v1
          Info: tsu for register "rx_processing_v3:rx_a_processing_a|da_reg[2][5]" (data pin = "rx_sreset_a"
          Info: tco from clock "tx_clk_a" to destination pin "tx_data_a[0]" through register "send_coordinat
          Info: th for register "rx_client_fifo_v8:rx_client_fifo_b|wr_data_pipe[0][0]" (data pin = "rx_data
          Info: Quartus II 64-Bit Classic Timing Analyzer was successful. 0 errors, 1 warning
          Info: *******************************************************************
          Info: Running Quartus II 64-Bit EDA Netlist Writer
          Info: Command: quartus eda --read settings files=off --write settings files=off switch v2 -c switc
```

System (17) | Processing (332) | Extra Info | Info (196) | Warning (136) | Critical Warning | Error | Suppressed (6) | Flag

Message: 3542 of 3624 — Location 1: Chip Planner — Locate

For Help, press F1 — Idle

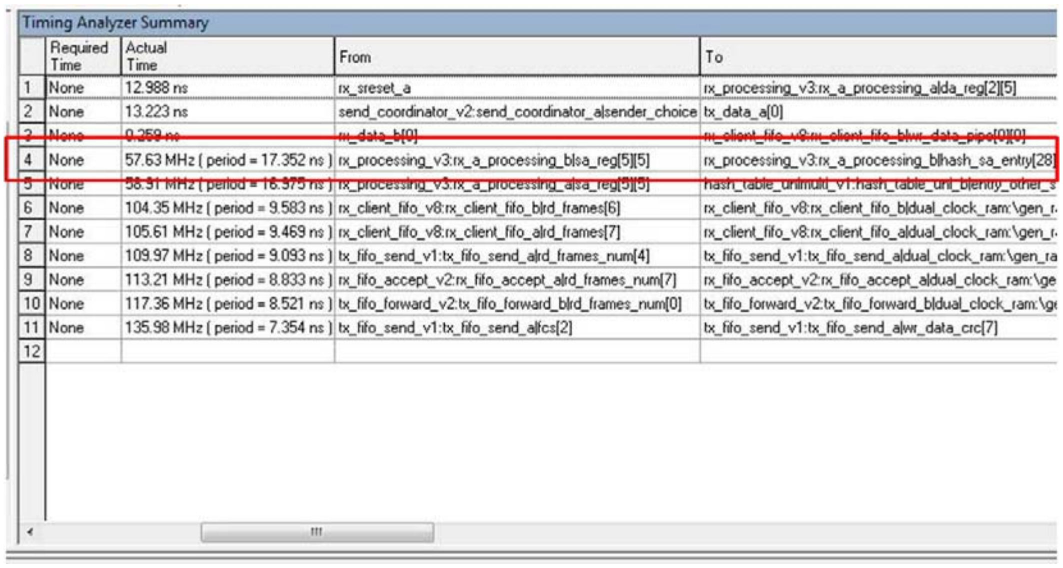
Figure 5.19 The timing analysis in Quartus synthesis tool

For the tx_clk_b clock, the Tco = 0.041ns, Tdelay = 8.264ns, Tsu = 0.286ns,

Tclkskew = 0.07 ns. Therefore the highest operating frequency of this path is

$$f_{\max} = \frac{1}{0.041 + 8.264 + 0.286 - 0.07} = 117.36MHz$$

The critical clock path is the path with the lowest *fmax*, in this circuit, the critical clock path is the path driven by the MII receive clock at port b, which has a *fmax* of 57.63 MHz (Figure 5.19, Figure 5.20).Since this frequency is twice as the MII clock frequency working at 100Mbits/s, which is 25MHz, the designed circuits operates without problem.



Timing Analyzer Summary

	Required Time	Actual Time	From	To
1	None	12.988 ns	rx_sreset_a	rx_processing_v3:rx_a_processing_a|da_reg[2][5]
2	None	13.223 ns	send_coordinator_v2:send_coordinator_a|sender_choice	tx_data_a[0]
3	None	0.259 ns	rx_data_b[0]	rx_client_fifo_v8:rx_client_fifo_b|wr_data_pipe[0][0]
4	None	57.63 MHz (period = 17.352 ns)	rx_processing_v3:rx_a_processing_b|sa_reg[5][5]	rx_processing_v3:rx_a_processing_b|hash_sa_entry[28]
5	None	58.91 MHz (period = 16.975 ns)	rx_processing_v3:rx_a_processing_a|sa_reg[5][5]	hash_table_unimulti_v1:hash_table_uni_b|entry_other_s
6	None	104.35 MHz (period = 9.583 ns)	rx_client_fifo_v8:rx_client_fifo_b|rd_frames[6]	rx_client_fifo_v8:rx_client_fifo_b|dual_clock_ram:\gen_r
7	None	105.61 MHz (period = 9.469 ns)	rx_client_fifo_v8:rx_client_fifo_a|rd_frames[7]	rx_client_fifo_v8:rx_client_fifo_a|dual_clock_ram:\gen_r
8	None	109.97 MHz (period = 9.093 ns)	tx_fifo_send_v1:tx_fifo_send_a|rd_frames_num[4]	tx_fifo_send_v1:tx_fifo_send_a|dual_clock_ram:\gen_ra
9	None	113.21 MHz (period = 8.833 ns)	rx_fifo_accept_v2:rx_fifo_accept_a|rd_frames_num[7]	rx_fifo_accept_v2:rx_fifo_accept_a|dual_clock_ram:\ge
10	None	117.36 MHz (period = 8.521 ns)	tx_fifo_forward_v2:tx_fifo_forward_b|rd_frames_num[0]	tx_fifo_forward_v2:tx_fifo_forward_b|dual_clock_ram:\ge
11	None	135.98 MHz (period = 7.354 ns)	tx_fifo_send_v1:tx_fifo_send_a|fcs[2]	tx_fifo_send_v1:tx_fifo_send_a|wr_data_crc[7]
12				

Figure 5.20 The critical clock path (in red box)

5.5.3 Synchronization between Clock Domains

In digital designs, many variations of clocks are used. When clocks are not synchronous, signals that are used to communicate between two asynchronous clock domains require synchronization [25]. The synchronizations between clock domains are very important for the finite state machine design (FSM). If the signal from other clock domain is used by the state machine in this clock domain without synchronization, undefined states may appear.

Figure 5.21 shows a state machine operation. The asynchronous signal is from another clock domain; together with the signal “current state” the “next state” signal is determined. At the next rising edge of the clock signal, the “next state” signal is registered to the “current state” signal.

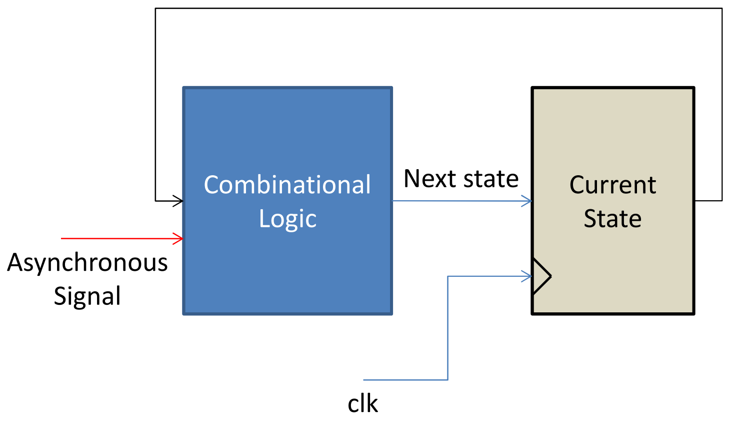


Figure 5.21 The state machine operation with asynchronous signal

Because the signal is asynchronous to the clock, the situation may happen that it changes just shortly before the rising edge of the clock. Because the combinational circuit introduces a small delay, the decoded next state is not stable when the rising edge of the clock comes. At this time the next state can be an arbitrary code and an undefined state can be thus registered to the “current state” signal. This case is shown in Figure 5.22.

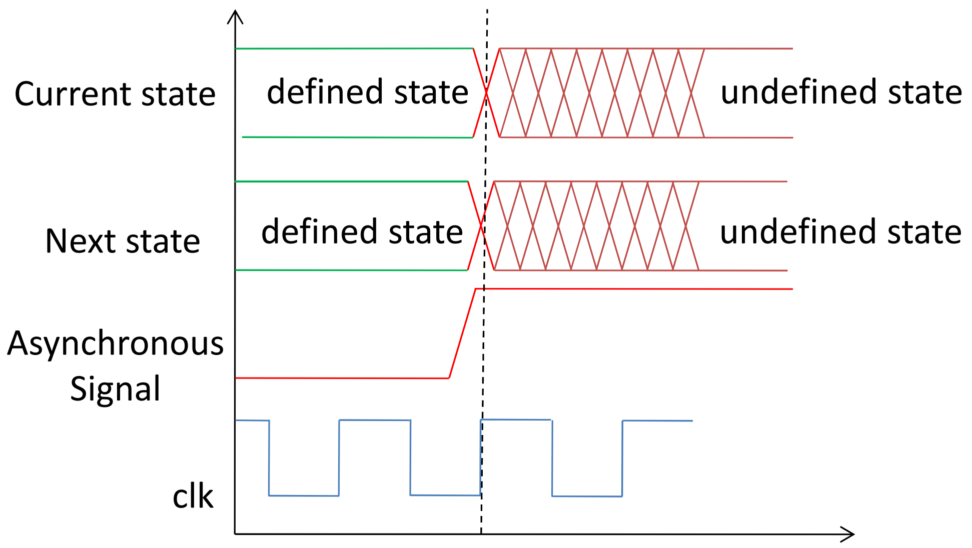


Figure 5.22 an undefined state is registered

To avoid this situation, the asynchronous signal must be first registered in this clock domain, so that decoded “next state” signal is always stable when the rising edge of the clock comes. The circuit is shown in Figure 5.22.

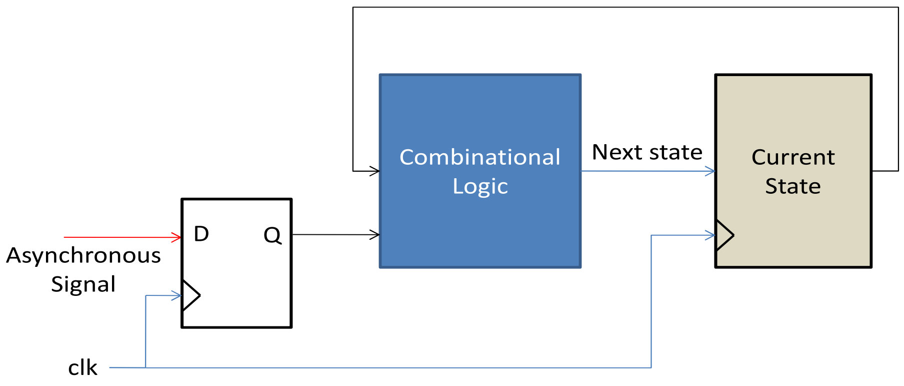


Figure 5.23 Sample the asynchronous signal

The waveform of this circuit is shown in Figure 5.24. The asynchronous signal is changing when the rising edge of the clock comes at time t1. The signal “Q” is registered to an unknown level which could be either ‘0’ or ‘1’. Because there is a delay of the propagation from signal “Q” to signal “Next State”, the “Next State” has not had time to change when the rising edge of clock comes at t1. The “Current State” stays thus unchanged after the rising edge of the clock. After a clock period at time t2, the “Next State” signal is already stable, but the value can be unchanged if the Q sampled at last clock is unchanged. At time t2, the asynchronous signal is sampled correctly to the signal “Q”, the decoded signal “Next State” changes after the rising edge of the clock. The “Current State” signal is registered to the value of the “Next State” signal of last clock period, which may be unchanged. At time t3, the correctly decoded “Next State” signal is registered to the “Current State” signal.

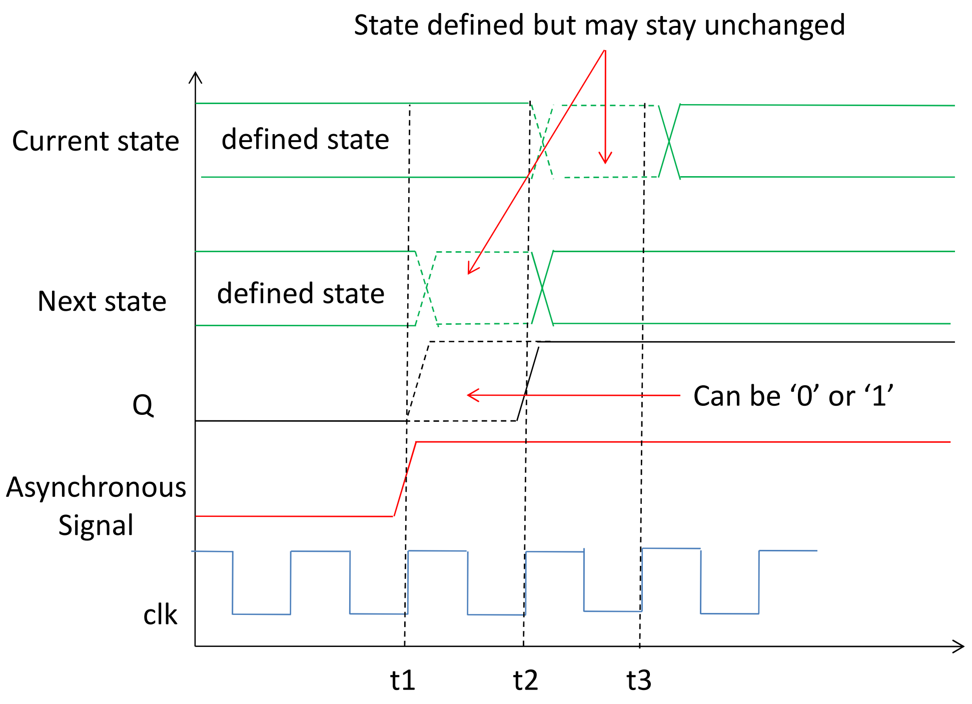


Figure 5.24 the state is always defined after registering the asynchronous signal

This process shows although the signal “Q” may not be sampled correctly when the asynchronous signal is changing at the rising edge of the clock, state is never undefined, but the state transfer is delayed for a clock period.

5.6 Synthesis and Evaluation of the Implemented Switch Element

In this chapter, the implemented switch element is synthesized in the QuartusII development environment. A short report of the synthesis is presented. After the Synthesis, evaluation is performed. The functionalities defined in the HSR protocol and the duplicated and circular frame rejection are tested.

5.6.1 Synthesis

After the functional design is free of error, the design is synthesized in Quartus design environment. In synthesis, the logic functionalities are mapped to the logic component of the goal device. The time requirement is examined. The resource on the device is allocated. After the synthesis, a netlist file (the .vho file) is generated which describes the interconnection of the components on the devices. A file (the .sdo file) contains the timing information is also generated, which can be used later in the Post-synthesis simulation.

The resource usage summery is shown in Table Table 5.2 . The percentage of the logic elements usage is 26%. The memory block usage is 76%. The timing analysis summary is shown in Table 5.3. The critical clock path is 59.3 MHz which is twice more than MII clock frequency in the 100Mbit/s Ethernet mode.

Total logic elements	4,794 / 18,752 (26 %)
Combinational with no register	2018
register only	400
Combinational with a register	2376
Logic element usage by number of LUT inputs	
4 input functions	2276
3 input functions	772
2 input functions	1346
register only	400
Logic elements by mode	
normal mode	3768
arithmetic mode	626
Total registers	2,776 / 19,160 (14 %)
Dedicated logic registers	2,776 / 18,752 (15 %)
I/O registers	0 / 408 (0 %)
Total LABs: partially or completely used	401 / 1,172 (34 %)
I/O pins	58 / 152 (38 %)
Clock pins	7 / 8 (88 %)
Global signals	8
M4Ks	52 / 52 (100 %)
Total memory bits	165,888 / 239,616 (76 %)
Total RAM block bits	239,616 / 239,616 (100 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
PLLs	0 / 4 (0 %)
Global clocks	8 / 16 (50 %)
Average interconnect usage	17%
Peak interconnect usage	32%
Maximum fan-out node	rx_clk_a~clkctrl
Maximum fan-out	776
Highest non-global fan-out signal	rx_sreset_b
Highest non-global fan-out	438
Total fan-out	24387
Average fan-out	3.25

Table 5.2 The resouce usage summery of the design

Type	Slack	Required time	Actual Time
Worst-case tsu	N/A	None	10.375 ns
Worst-case tco	N/A	None	11.923 ns
Worst-case th	N/A	None	-0.043 ns
Clock Setup: 'rx_clk_a'	N/A	None	59.53 MHz (period = 16.797 ns)
Clock Setup: 'rx_clk_b'	N/A	None	60.95 MHz (period = 16.407 ns)
Clock Setup: 'rx_clk_ll_b'	N/A	None	95.57 MHz (period = 10.464 ns)
Clock Setup: 'rx_clk_ll_a'	N/A	None	102.52 MHz (period = 9.754 ns)
Clock Setup: 'tx_clk_b'	N/A	None	127.70 MHz (period = 7.831 ns)
Clock Setup: 'rx_clk_hst'	N/A	None	129.70 MHz (period = 7.710 ns)
Clock Setup: 'tx_clk_a'	N/A	None	131.54 MHz (period = 7.602 ns)
Clock Setup: 'tx_clk_hst'	N/A	None	133.17 MHz (period = 7.509 ns)
Total number of failed paths	0		

Table 5.3 The timing analysis summary

5.7 Evaluation

5.7.1 Simulation Environment

The evaluation of the synthesized switch element is performed by simulation in Modelsim. For the post-synthesis simulation, the following files are needed.

- The switch*.vho* file generated by the synthesis tool
- The switch*.sdo* file which contains the timing information about the devices on the FPGA
- The *cycloneii* library which contains the components of CycloneII series FPGA
- The *switch_TB*.vhd file which is the testbench

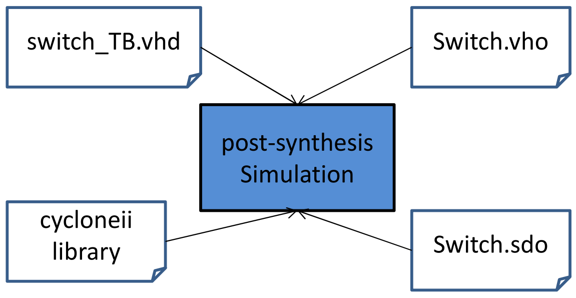


Figure 5.25 the pos-synthesis simulation environment

5.7.2 Tested Functionalities

To verify if the designed switch element can realize the communication rules defined in HSR, the following tests should be performed.

General Communication Rules

1. Sending frames to the switch element, the frames should be duplicated and sent over both sending ports.
2. Receiving unicast frames which is not dedicated to this node, the frames should be forwarded over the sending port of the other line with cut-through mode.
3. Receiving unicast frames which is dedicated to this node, the frame should be accepted and passed to the receiving port to the host.
4. Receiving multicast frames, the frame should be forwarded to the sending port of the other line and passed to the receiving port to the host.

Rejection of Duplicated Frames

Unicast frames dedicated to this node and multicast frames received on one line should be rejected when received again on the other line.

Rejection of Circular Frames

Receiving unicast frames, multicast frames on one line, the unicast can be dedicated to this node or not. Receiving the same frames again on this line, all frames should be rejected

Bad Frame Handling

If frame is asserted as bad frame before the cut-though operation is performed, it is dropped. Otherwise a garbling sequence is appended at the end of the frame and the source address of this frame will be registered in the hash table for bad frames. If a frame the same source address is received again on the same line, no cut-through is performed, the frame is sent only after it is completely received and verified to be a good frame. If the frame is a good frame, the entry in the hash table for bad frames is cleared. Next time a frame with the same source address is received, the frame is sent in cut-through mode again.

Send Coordination

A frame in the sending FIFO can only be sent if no frame is receiving on the other line or the frame being received on the other line is a unicast frame dedicated to this node (which means no forwarding for this frame).

5.7.3 Simulation Results

In this section the simulation results of the switch element are presented. The names of the pins on the waveform are already introduced in Table 5.1.

General Communication Rules

1. Sending frame to the switch element:
   Frames are sent sequentially to the sending port of the switch element. The frames are duplicated and sent over both lines. The simulation waveforms are shown in Figure 5.26.

2. Receiving unicast frames which is not dedicated to this node:
   In this test, eight different unicast frames which are not dedicated to are received from line A and line B. Each line receives 4 frames. The receiving on both lines is concurrent. All the eight frames are forwarded to the sending port of the other line. The simulation waveforms are shown in Figure 5.27. The delay between receiving a frame and forwarding it is 2720 ns.

3. Receiving unicast frames which is dedicated to this node:
   In this test, 8 different unicast frames dedicated to this node are received on line A and line B concurrently. Each line receives 4 frames; all the 8 frames are passed to the receiving port to the host. The simulation waveforms are shown in Figure 5.28.

4. Receiving multicast frames:
   In this test, 8 multicast frames with different source address are received on line A and line B concurrently. Each line receives 4 frames; all the 8 frames should be passed to the receiving port to the host and forwarded to the other line. The waveforms are shown in Figure 5.29.

Rejection of Duplicated Frames

1. Receiving on both lines is not concurrent
   In this test, unicast frames dedicated to this node and multicast frames are received on one line. The order of receiving is multicast, unicast, multicast, unicast. The two unicast frames should be accepted and passed to the receiving port to the host. The two mulicast frames should be accepted, passed to the receiving port to the host and forwarded to the sending port of the other line. After the receiving of these 4 frames is finished, begin receiving the same 4 frames from the other line, all the frames should be rejected and no frame is sent on the receiving port to the host and forwarded to the sending port of the other line. The simulation waveforms are shown in
2. Receiving on both lines is concurrent, this test the DA,SA,SEQ window functionality. The procedure is same as above, except that the receiving on both lines proceeds at the same time.

The simulation waveforms are shown in Figure 5.30 and Figure 5.31.

Rejection of Circular Frames

In this test, unicast frames dedicated to this node and multicast frames are received on one line. The order of receiving is multicast, unicast, multicast, unicast. The two unicast frames are accepted and passed to the receiving port to the host. The two mulicast frames are accepted, passed to the receiving port to the host and forwarded to the sending port of the other line. Then the same 4 frames are received again at the same line, all the frames should be rejected. The simulation waveforms are shown in Figure 5.32.

Bad Frame Handling

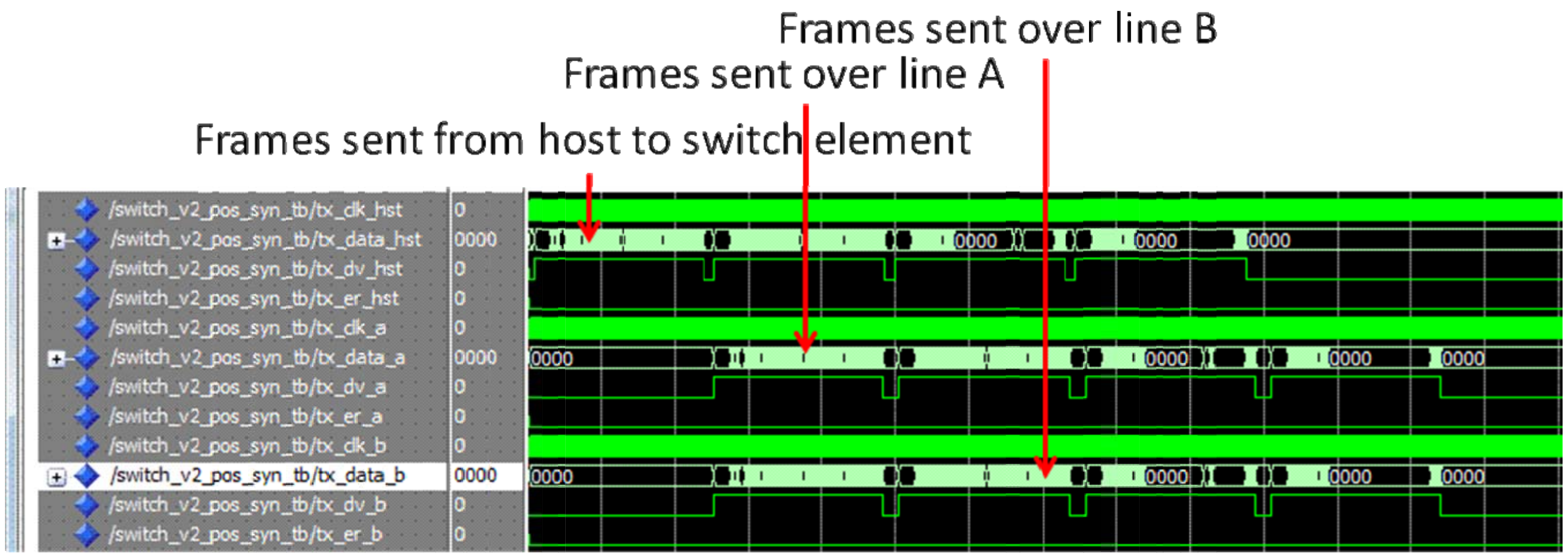
1. Two multicast frames with the same source address but different sequence number are received on line A.
   The first frame received on line A is asserted to be a bad frame at the last nibble of the source address. Because the cut-through is not performed when the bad frame is asserted there, the frame will be dropped, nothing is forwarded to the sending port of line B, nothing is sent to the receiving port to the host and the source address is not registered in the table for bad frames. Then the second frame with the same source address is received again on line A, without bad frame assertion. The frame should be forwarded to the sending port of line B and received at the receiving port to the host with cut-through operation.
2. Two multicast frames and a unicast frame not dedicated to the node received on line B, all the three frames have the same source address, and the two multicast frames do not have the same sequence number.
   The first multicast frame received on line B is asserted to be a bad frame at the middle of the frame (here is the 40 th byte of the frame). The cut-through operation is already performed if the bad frame is asserted at this position. Therefore the frame is truncated and appended a garbling sequence and its source address is stored in the table for bad frames. The second multicast frame is forwarded to the sending port of line A and the receiving port to host. Recall that the RX_FIFO to host has the ability to detect the garbling sequence. The frame is dropped in RX_FIFO and nothing is passed to the host. Then the second multicast frame with the same source address is received again on line B, without bad frame assertion. The frame should be completely received and forwarded to the sending port of line B and received at the receiving port to the host. The entry in the hash table for bad frames should be cleared. Then the unicast frame with the same source address is received on line B. Because the entry in the hash table for bad frame is already cleaned, this frame should be forwarded with cut-through operation.

The simulation waveform is shown in Figure 5.33.

Send Coordination

1. Line A begins receiving a multicast just short before a frame is completely received from the host. The frame in the sending FIFO on line B should wait until the received frame is completely sent on line B.
2. Line B begins receiving a unicast frame dedicated to this node just short before a frame (the same frame as above) is completely received from the host. After the receiving main routine finds that this unicast frame is dedicated to this node (which means that no forwarding for this frame), the frame in the sending FIFO on line A is allowed to send the frame on line A

The simulation waveform is shown in Figure 5.34



```
/switch_v2_pos_syn_tb/tx_clk_hst    0
/switch_v2_pos_syn_tb/tx_data_hst   0000
/switch_v2_pos_syn_tb/tx_dv_hst     0
/switch_v2_pos_syn_tb/tx_er_hst     0
/switch_v2_pos_syn_tb/tx_clk_a      0
/switch_v2_pos_syn_tb/tx_data_a     0000
/switch_v2_pos_syn_tb/tx_dv_a       0
/switch_v2_pos_syn_tb/tx_er_a       0
/switch_v2_pos_syn_tb/tx_clk_b      0
/switch_v2_pos_syn_tb/tx_data_b     0000
/switch_v2_pos_syn_tb/tx_dv_b       0
/switch_v2_pos_syn_tb/tx_er_b       0
```

Figure 5.26 Sending frames to the switch element

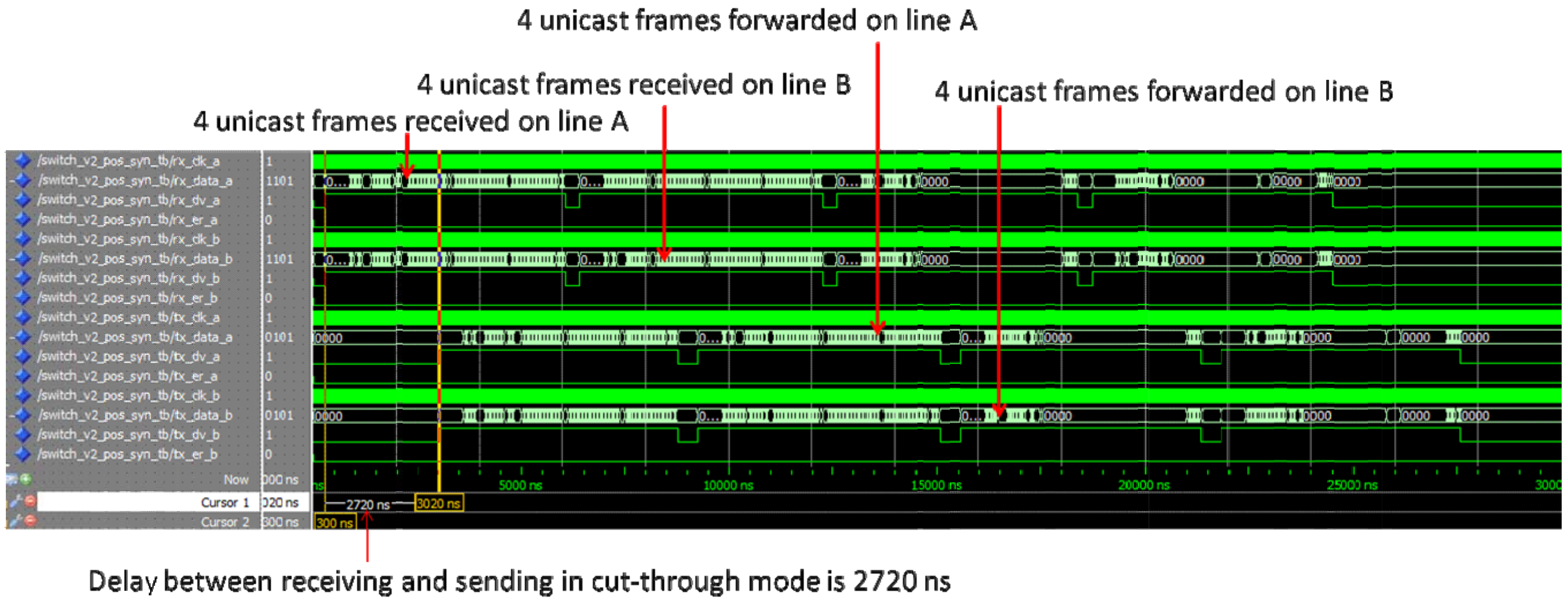
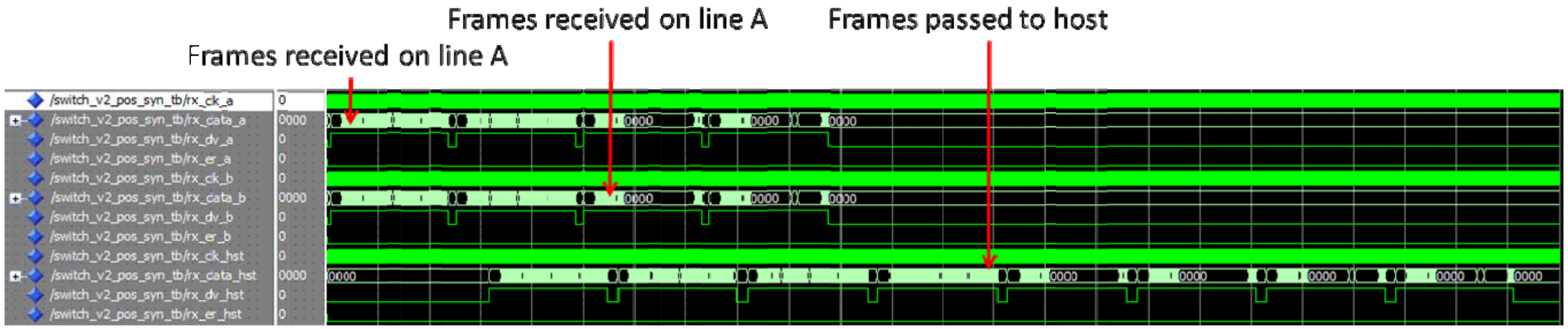

4 unicast frames forwarded on line A
4 unicast frames received on line B
4 unicast frames forwarded on line B
4 unicast frames received on line A
/switch_v2_pos_syn_tb/rx_clk_a
/switch_v2_pos_syn_tb/rx_data_a
/switch_v2_pos_syn_tb/rx_dv_a
/switch_v2_pos_syn_tb/rx_er_a
/switch_v2_pos_syn_tb/rx_clk_b
/switch_v2_pos_syn_tb/rx_data_b
/switch_v2_pos_syn_tb/rx_dv_b
/switch_v2_pos_syn_tb/rx_er_b
/switch_v2_pos_syn_tb/tx_clk_a
/switch_v2_pos_syn_tb/tx_data_a
/switch_v2_pos_syn_tb/tx_dv_a
/switch_v2_pos_syn_tb/tx_er_a
/switch_v2_pos_syn_tb/tx_clk_b
/switch_v2_pos_syn_tb/tx_data_b
/switch_v2_pos_syn_tb/tx_dv_b
/switch_v2_pos_syn_tb/tx_er_b
Now
Cursor 1
Cursor 2
5000 ns
10000 ns
15000 ns
20000 ns
25000 ns
2720 ns
3020 ns
300 ns
Delay between receiving and sending in cut-through mode is 2720 ns

Figure 5.27 Receiving unicast frames not dedicated to the node on line A and line B concurrently



```
/switch_v2_pos_syn_tb/rx_ck_a       0
/switch_v2_pos_syn_tb/rx_data_a     0000
/switch_v2_pos_syn_tb/rx_dv_a       0
/switch_v2_pos_syn_tb/rx_er_a       0
/switch_v2_pos_syn_tb/rx_ck_b       0
/switch_v2_pos_syn_tb/rx_data_b     0000
/switch_v2_pos_syn_tb/rx_dv_b       0
/switch_v2_pos_syn_tb/rx_er_b       0
/switch_v2_pos_syn_tb/rx_ck_hst     0
/switch_v2_pos_syn_tb/rx_data_hst   0000
/switch_v2_pos_syn_tb/rx_dv_hst     0
/switch_v2_pos_syn_tb/rx_er_hst     0
```

Figure 5.28 Receiving unicast frames dedicated to this node

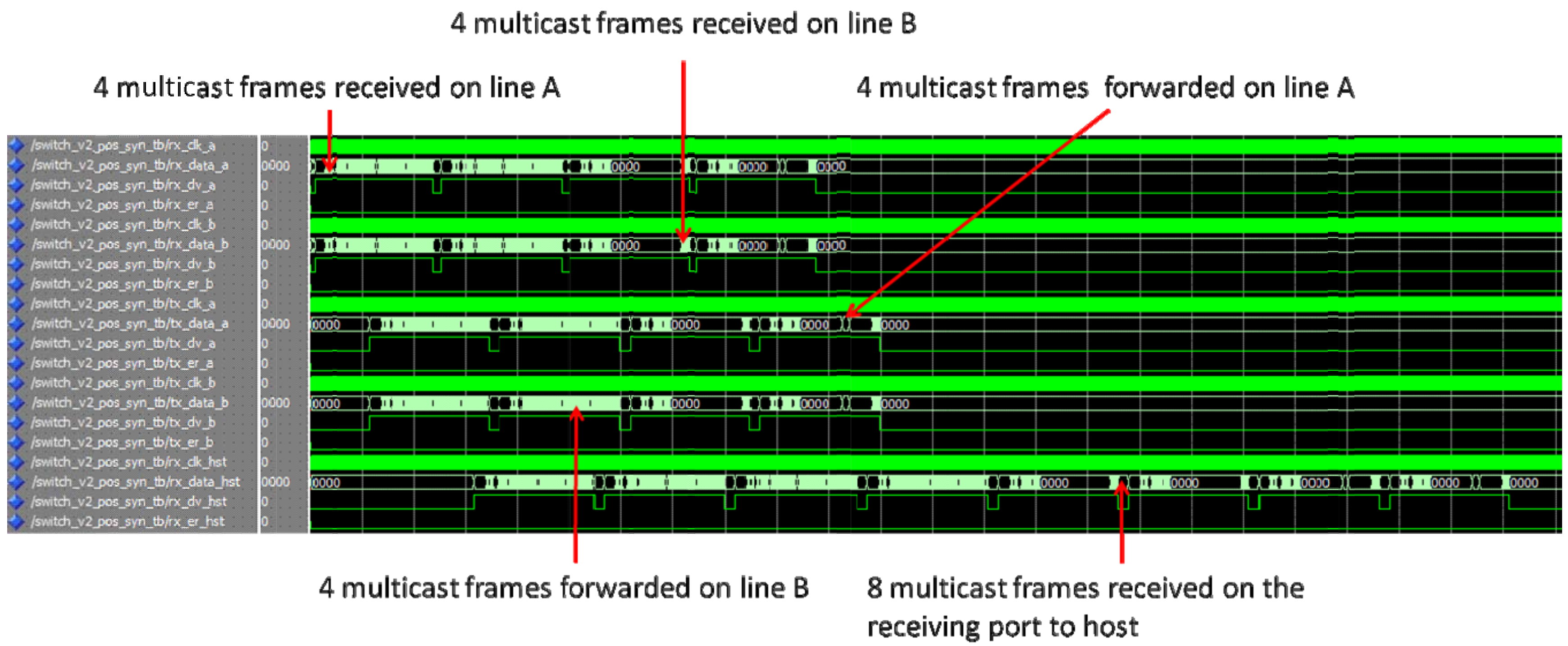

4 multicast frames received on line B
4 multicast frames received on line A
4 multicast frames forwarded on line A
/switch_v2_pos_syn_tb/rx_clk_a
/switch_v2_pos_syn_tb/rx_data_a
/switch_v2_pos_syn_tb/rx_dv_a
/switch_v2_pos_syn_tb/rx_er_a
/switch_v2_pos_syn_tb/rx_clk_b
/switch_v2_pos_syn_tb/rx_data_b
/switch_v2_pos_syn_tb/rx_dv_b
/switch_v2_pos_syn_tb/rx_er_b
/switch_v2_pos_syn_tb/tx_clk_a
/switch_v2_pos_syn_tb/tx_data_a
/switch_v2_pos_syn_tb/tx_dv_a
/switch_v2_pos_syn_tb/tx_er_a
/switch_v2_pos_syn_tb/tx_clk_b
/switch_v2_pos_syn_tb/tx_data_b
/switch_v2_pos_syn_tb/tx_dv_b
/switch_v2_pos_syn_tb/tx_er_b
/switch_v2_pos_syn_tb/rx_clk_hst
/switch_v2_pos_syn_tb/rx_data_hst
/switch_v2_pos_syn_tb/rx_dv_hst
/switch_v2_pos_syn_tb/rx_er_hst
4 multicast frames forwarded on line B
8 multicast frames received on the receiving port to host

Figure 5.29 Receiving multicast frames from both lines

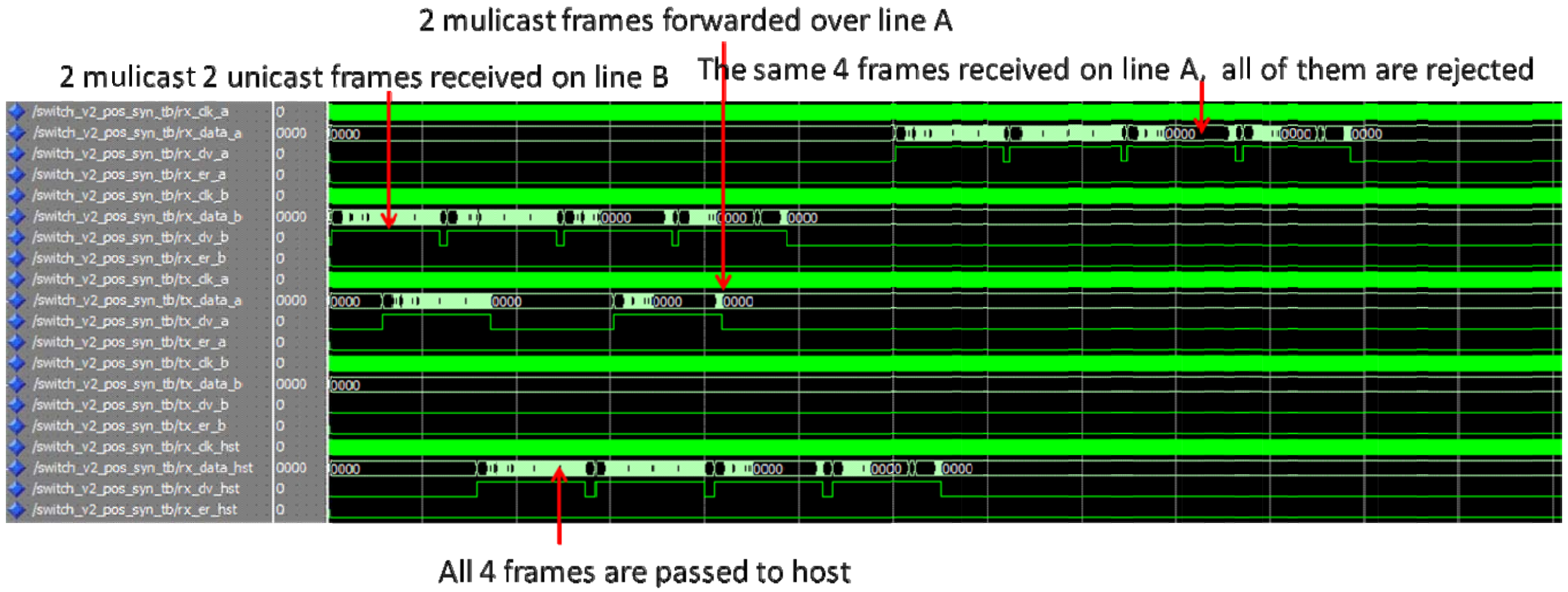

2 mulicast frames forwarded over line A
2 mulicast 2 unicast frames received on line B
The same 4 frames received on line A, all of them are rejected
/switch_v2_pos_syn_tb/rx_clk_a
/switch_v2_pos_syn_tb/rx_data_a
/switch_v2_pos_syn_tb/rx_dv_a
/switch_v2_pos_syn_tb/rx_er_a
/switch_v2_pos_syn_tb/rx_clk_b
/switch_v2_pos_syn_tb/rx_data_b
/switch_v2_pos_syn_tb/rx_dv_b
/switch_v2_pos_syn_tb/rx_er_b
/switch_v2_pos_syn_tb/tx_clk_a
/switch_v2_pos_syn_tb/tx_data_a
/switch_v2_pos_syn_tb/tx_dv_a
/switch_v2_pos_syn_tb/tx_er_a
/switch_v2_pos_syn_tb/tx_clk_b
/switch_v2_pos_syn_tb/tx_data_b
/switch_v2_pos_syn_tb/tx_dv_b
/switch_v2_pos_syn_tb/tx_er_b
/switch_v2_pos_syn_tb/rx_clk_hst
/switch_v2_pos_syn_tb/rx_data_hst
/switch_v2_pos_syn_tb/rx_dv_hst
/switch_v2_pos_syn_tb/rx_er_hst
All 4 frames are passed to host

Figure 5.30 Simulation waveform of rejecting duplicated frames. Frames are not received at the same time on both lines

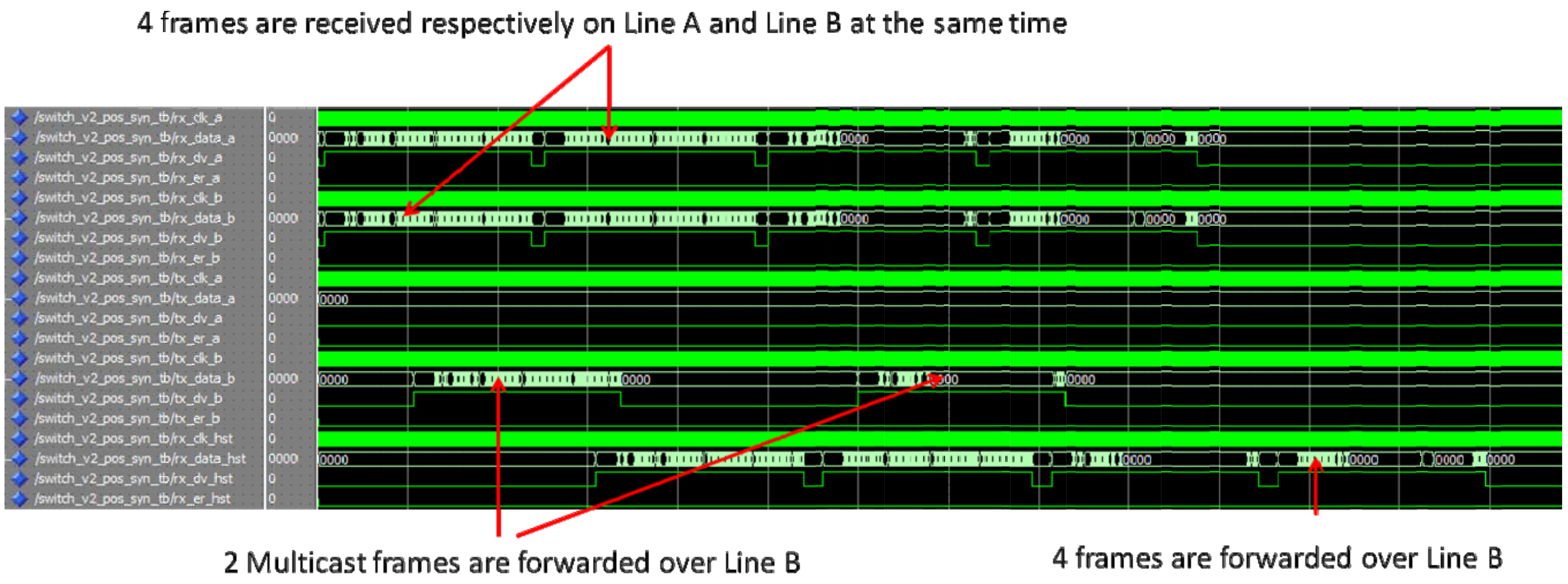

4 frames are received respectively on Line A and Line B at the same time
/switch_v2_pos_syn_tb/rx_clk_a
/switch_v2_pos_syn_tb/rx_data_a
/switch_v2_pos_syn_tb/rx_dv_a
/switch_v2_pos_syn_tb/rx_er_a
/switch_v2_pos_syn_tb/rx_clk_b
/switch_v2_pos_syn_tb/rx_data_b
/switch_v2_pos_syn_tb/rx_dv_b
/switch_v2_pos_syn_tb/rx_er_b
/switch_v2_pos_syn_tb/tx_clk_a
/switch_v2_pos_syn_tb/tx_data_a
/switch_v2_pos_syn_tb/tx_dv_a
/switch_v2_pos_syn_tb/tx_er_a
/switch_v2_pos_syn_tb/tx_clk_b
/switch_v2_pos_syn_tb/tx_data_b
/switch_v2_pos_syn_tb/tx_dv_b
/switch_v2_pos_syn_tb/tx_er_b
/switch_v2_pos_syn_tb/rx_clk_hst
/switch_v2_pos_syn_tb/rx_data_hst
/switch_v2_pos_syn_tb/rx_dv_hst
/switch_v2_pos_syn_tb/rx_er_hst
2 Multicast frames are forwarded over Line B
4 frames are forwarded over Line B

Figure 5.31 Simulation waveform of rejecting duplicated frames, frames are received on both lines at the same time

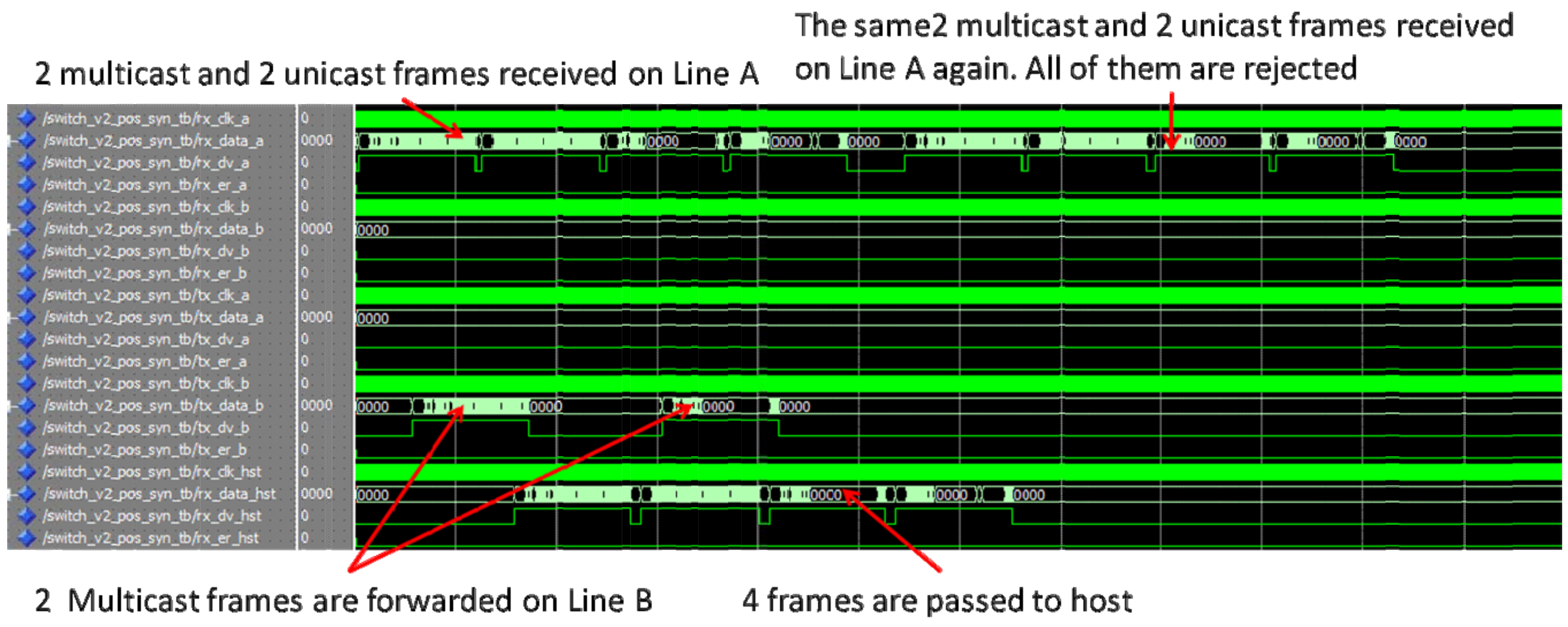

2 multicast and 2 unicast frames received on Line A
The same2 multicast and 2 unicast frames received on Line A again. All of them are rejected
/switch_v2_pos_syn_tb/rx_clk_a
/switch_v2_pos_syn_tb/rx_data_a
/switch_v2_pos_syn_tb/rx_dv_a
/switch_v2_pos_syn_tb/rx_er_a
/switch_v2_pos_syn_tb/rx_clk_b
/switch_v2_pos_syn_tb/rx_data_b
/switch_v2_pos_syn_tb/rx_dv_b
/switch_v2_pos_syn_tb/rx_er_b
/switch_v2_pos_syn_tb/tx_clk_a
/switch_v2_pos_syn_tb/tx_data_a
/switch_v2_pos_syn_tb/tx_dv_a
/switch_v2_pos_syn_tb/tx_er_a
/switch_v2_pos_syn_tb/tx_clk_b
/switch_v2_pos_syn_tb/tx_data_b
/switch_v2_pos_syn_tb/tx_dv_b
/switch_v2_pos_syn_tb/tx_er_b
/switch_v2_pos_syn_tb/rx_clk_hst
/switch_v2_pos_syn_tb/rx_data_hst
/switch_v2_pos_syn_tb/rx_dv_hst
/switch_v2_pos_syn_tb/rx_er_hst
2 Multicast frames are forwarded on Line B
4 frames are passed to host

Figure 5.32 the simulation result of rejecting cicular frame

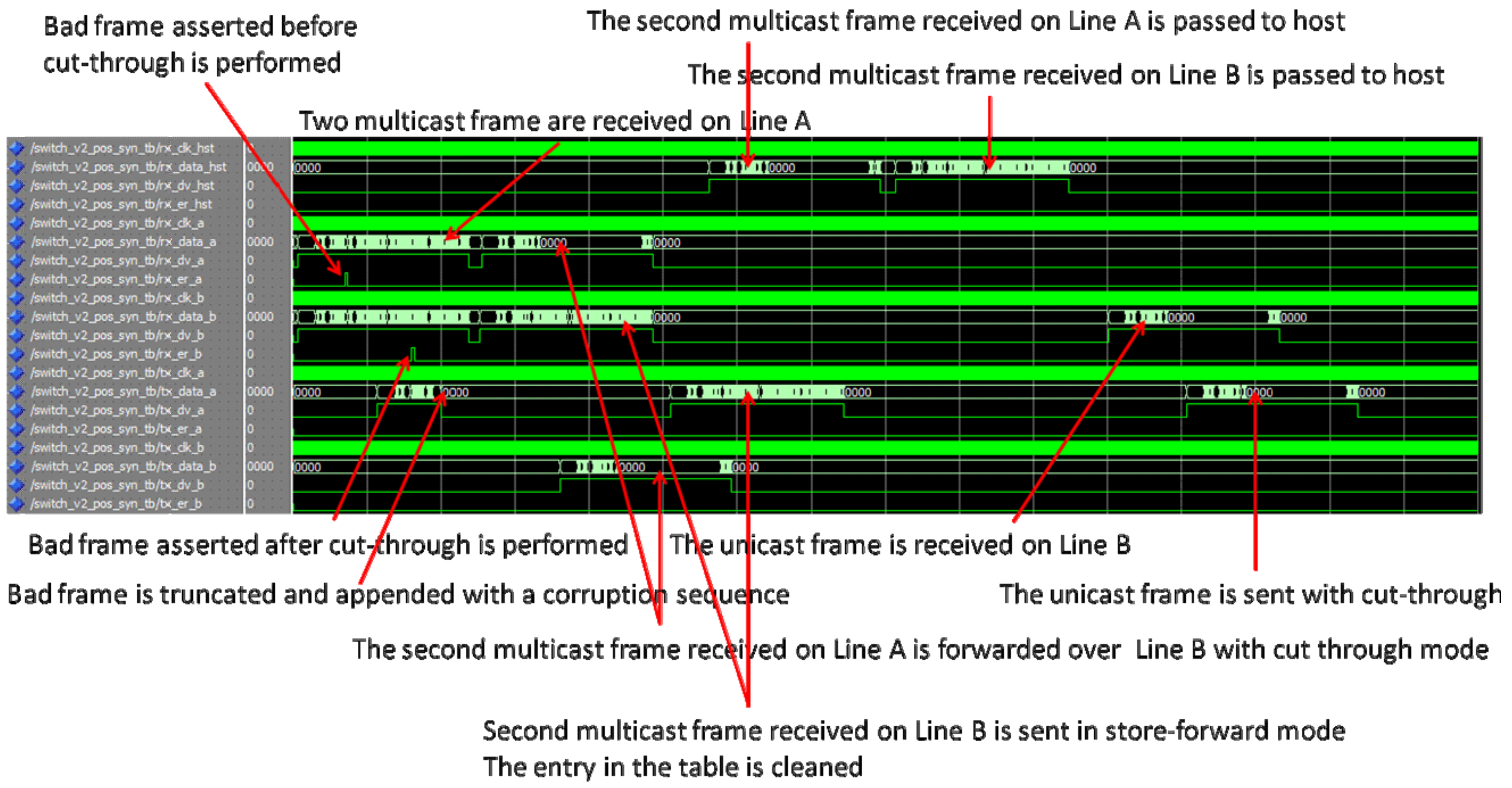


Figure 5.33 The simulation waveform of bad frame handling

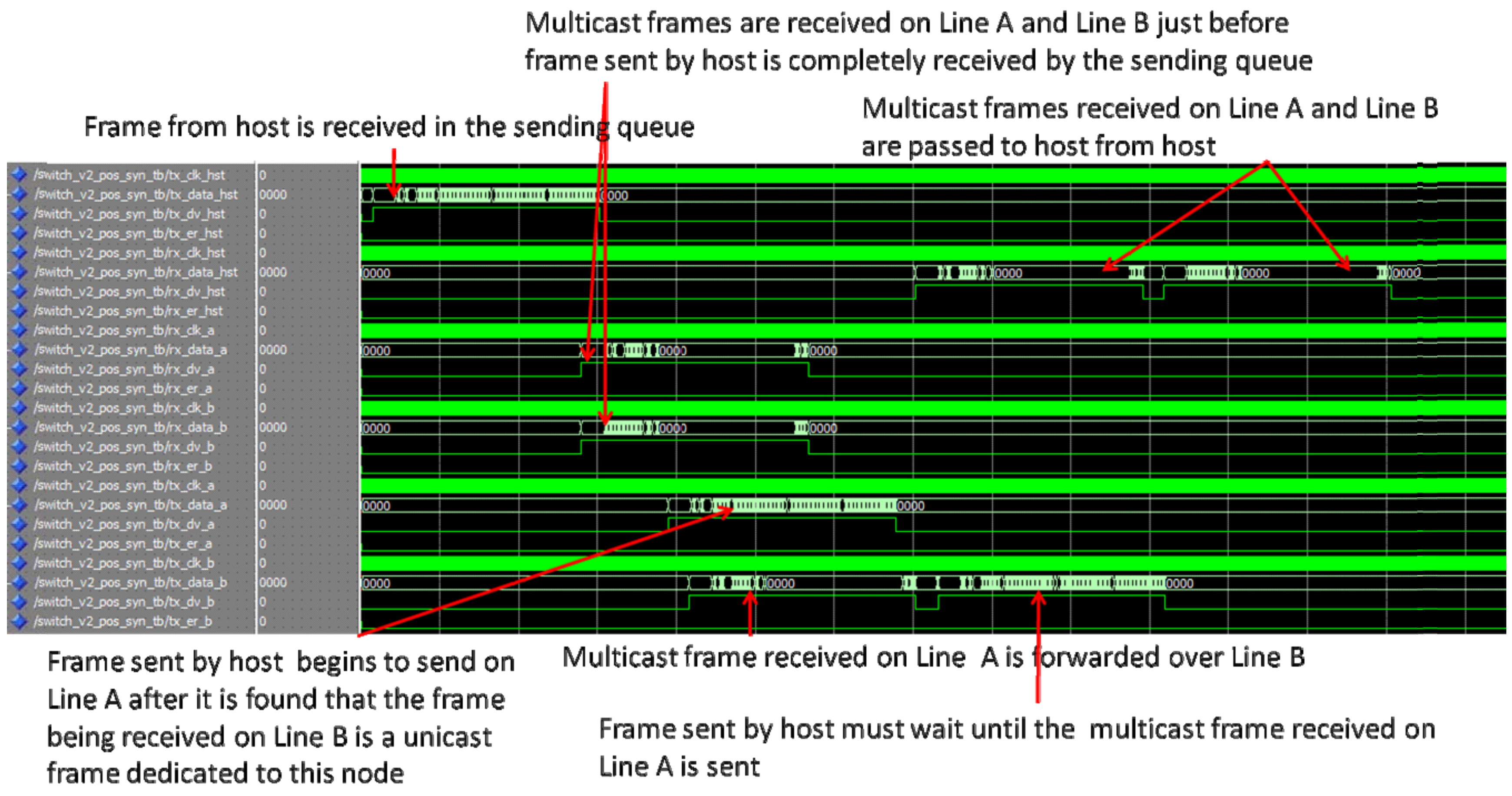

Multicast frames are received on Line A and Line B just before frame sent by host is completely received by the sending queue
Frame from host is received in the sending queue
Multicast frames received on Line A and Line B are passed to host from host
/switch_v2_pos_syn_tb/tx_clk_hst
/switch_v2_pos_syn_tb/tx_data_hst
/switch_v2_pos_syn_tb/tx_dv_hst
/switch_v2_pos_syn_tb/tx_er_hst
/switch_v2_pos_syn_tb/rx_clk_hst
/switch_v2_pos_syn_tb/rx_data_hst
/switch_v2_pos_syn_tb/rx_dv_hst
/switch_v2_pos_syn_tb/rx_er_hst
/switch_v2_pos_syn_tb/rx_clk_a
/switch_v2_pos_syn_tb/rx_data_a
/switch_v2_pos_syn_tb/rx_dv_a
/switch_v2_pos_syn_tb/rx_er_a
/switch_v2_pos_syn_tb/rx_clk_b
/switch_v2_pos_syn_tb/rx_data_b
/switch_v2_pos_syn_tb/rx_dv_b
/switch_v2_pos_syn_tb/rx_er_b
/switch_v2_pos_syn_tb/tx_clk_a
/switch_v2_pos_syn_tb/tx_data_a
/switch_v2_pos_syn_tb/tx_dv_a
/switch_v2_pos_syn_tb/tx_er_a
/switch_v2_pos_syn_tb/tx_clk_b
/switch_v2_pos_syn_tb/tx_data_b
/switch_v2_pos_syn_tb/tx_dv_b
/switch_v2_pos_syn_tb/tx_er_b
0000
Frame sent by host begins to send on Line A after it is found that the frame being received on Line B is a unicast frame dedicated to this node
Multicast frame received on Line A is forwarded over Line B
Frame sent by host must wait until the multicast frame received on Line A is sent

Figure 5.34 The simulation waveform of send coordination

5.8 Conclusion of the FPGA Implementation

In this chapter, the HSR protocol defined in Chapter 2 and the proposed algorithm for rejecting duplicated frames and circular frames are implemented in FPGA. Several design issues are discussed. The design is synthesized in Quartus and fulfills the timing requirement and has satisfied the resource constraints. The Post-synthesis simulations are performed for each communication rules in HSR protocol and the proposed algorithms. All the simulation results are successful.

6 Conclusions and Future Work

6.1 Conclusions

The High Availability Seamless Ring is a new industry automation communication network protocol standardized as IEC 62439. It applied the Parallel Redundancy Protocol (PRP) principle to provide in case of fault a bumpless switchover of the network, while maintaining a simple network topology and infrastructure. As a derivate of the PRP protocol, HSR needs to handle duplicate frames, and in addition must be capable to reject circular frames because of its ring topology. The drop window algorithm for rejecting duplicated frame applied in PRP cannot be transplanted to HSR because it is not designed for rejecting the circular frames. The challenge of this work was to find an algorithm which can handle both duplicate frames and circular frame. As recommended by IEC 62439, the rejection of the duplicated frames should be accomplished by hardware to offload the processor, which is indeed a necessity since HSR is used in applications such as substation which have a heavy traffic. This algorithm must be time-efficient to allow cut-through operation of the switches and be implementable with low-cost FPGAs. As an algorithm applied in hard real-time system, this algorithm must not disturb the deterministic delay times of the network.

This master thesis proposed the algorithms based on lookup table are considered as a reasonable substitute of the drop window algorithm. Different algorithms based on the lookup table were discussed and three algorithms were chosen as candidates for the hardware implementation. The three algorithms are circular buffer, hash table combined with circular buffer, hash table with aging which is classified into three methods according to the probing sequence algorithm applied: linear probing, quadratic probing and double hashing.

The evaluation of the proposed algorithms was based on three criteria: the rejection ratio of duplicated frames, the searching time efficiency and the hardware implementation complexity. Through the analysis of the operation principle of each algorithm, combined with the software simulation results, the following conclusions can be made for these three algorithms.

- The circular buffer is proven to have the best performance of rejecting the duplicated frames, but it has to go through the entire table to make the decision if an entry is in or not in the table. The searching time is the longest among the proposed algorithms
- The hash table with aging using quadratic probing, double hashing and the hash table combined with circular buffer, providing deterministic searching time. In the simulation they have the moderate performance among the proposed algorithms.
- The performance of the hash table with aging using linear probing also has deterministic search time but its performance is the worst among the proposed algorithms.
- The circular buffer has the lowest hardware implementation complexity among the proposed algorithms. The operation principle of the hash table with aging requires the highest hardware implementation complexity among the proposed algorithms. The hardware implementation complexity of the hash table combined with circular buffer is moderate.

Through the software simulation, a phenomena is discovered that the performance of all three algorithms degrade dramatically when the number of the nodes in the network exceeds a certain value. This value is influenced by the length of the frame and the time interval a node in the ring generates a frame. The IEC 61850-9-2 standard fixed the length of the frame to 138 bytes and the time interval of generating a frame to 250us.

Under such condition, the simulation shows the performance of all algorithms begins to drop at the node number of 60. The dependency to the node number in the network is a limit of the proposed algorithms.

After the simulation of the proposed algorithm and a discussion of the simulation results, the hash table combined with circular buffer was chosen as the algorithm to be implemented in FPGA. Before the implementation, two modifications were made to achieve higher rejection ratio of the duplicated frames and higher efficiency of memory usage. One is to separate the hash table for rejecting duplicated frames into a hash table for rejecting multicast frames and a hash table for rejecting the unicast frames. The other modification is reducing the entry in the two tables above to the source address and the sequence number. The collision between the unicast and multicast frames is therefore reduced and the memory usage can be configured differently for unicast and multicast frames, which is more flexible and efficient. At last the hash table for multicast frames of length of 32, max_bin number of 4, the hash table for unicast of length of 64, max_bin number of 4 and the hash table for circular frames of length of 64, max_bin number of 4 are implemented in FPGA. This configuration is dedicated to handle the network which has nodes less than 64.

The HSR protocol together with the hash table combined with circular buffer method was implemented in the Cyclone II, EP2C20 series FPGA. The maximum frequency of the critical clock path is more than twice of the MII frequency at the full speed of 100Mbits/s Ethernet. The resource consumption of the design is within the constraints of the device.

Finally, the pos-synthesis simulation was performed after the design is synthesized in the Quartus II design environment. The simulation evaluates the functionalities of the design with timing delay of the components on the device. The design realized all the functionalities defined HSR protocol and the functionality of rejecting the duplicated and circular frames.

6.2 Future Work

Although the performance of the algorithm is forecast by the simulation, and the limitation of the algorithm was observed, there is still lack of theoretical analysis of the communication in the ring and its relation to the proposed algorithm. It is of interest to discover the mathematical mechanism behind and give a more precise model to help configure the algorithm and the ring scale.

As the implementation of the design has done only down to the pos-synthesis simulation, the next step is to implement the design in the newly developed hardware platform. Software should be designed to simulate the behavior of the end node in the ring. The node can be connected together to a simple ring to test the implemented functionalities.

Also, a suited test environment should be designed to monitor the operation and perform a conformance test.

This Thesis also gives an inspiration to the design of the redundancy box (RedBox) which connects several singly attached nodes to HSR or couples the PRP network to HSR. The RedBox connected several singly attached node to HSR is shown in Figure 2.1. A RedBox should not only have a table of the received frames but also a table of all the nodes attached to it. To reject the duplicated frames, we can for example build tables for multicast and unicast frame at each line. The entry in the table for multicast still consists of source address and sequence number, but the entry in the table for unicast should consists of destination address, source address and sequence number because of the reason stated above.

Appendix A Abbreviations Used in the Thesis

AT: Aging Tag

DA: Destination Address

DANP: Double Attached Node

DPRAM: Dual Port RAM

HSR: High Availability Seamless Ring

IEC: International Electrotechnical Commission

PRP: Parallel Redundancy Protocol

RCT: Redundancy Check Tag

RTL: Register Transfer Level

SA: Source Address

SAN: Singly Attached Node

SEQ: Sequence Number

Bibliography

[1] Hubert Kirrmann, Steven Kunsman and Peter Rietmann, ABB Network Redundancy Using IEC 62439, Protection, Automation & Control World, Autum 2008

[2] Hubert Kirrmann, Requirements on redundancy, IEC Technical Committee 57 Work Group 10 Redundancy Requirements TF

[3] Hubert Kirrmann , Proposal for a seamless ring solution for IEC 61850, ABB Switzerland Ltd, Technical Committee 57 Work Group 10, Buenos Aires, November 2008

[4] IEC CDV 62439-3 International Electrotechnical Commission, Industrial Networks – Highly Available Automation Networks, Parallel Redundancy Protocol and High availability Seamless Ring.

[5] International Electrotechnical Commission, Communication Networks and Systems in Substations Part 9-2: Specific Communication Service Mapping (SCSM) –Sampled values over ISO/IEC 8802-3

[6] IEC 61850 International Electrotechnical Commission, Communication networks and systems for power utility automation – Part 8-1: Specific Communication Service Mapping (SCSM) – Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3

[7] Glattfelder, Ch, PRP Red Box FPGA Implementation, School of Engineering, InES Institute of Embedded Systems, Zurich University of Applied Sciences, October 2008

[8] Glattfelder, Ch, Architecture Study of the PRP Red Box, , School of Engineering, InES Institute of Embedded Systems, Zurich University of Applied Sciences, January 2007

[9] Hash Table, Wikipedia, http://en.wikipedia.org/wiki/Hash_table

[10] Jeff Erikson, Combinatorial Algorithm, Lecture 6, Hash Table, Department of Computer Science at the University of Illinois at Urbana-Champaign

[11] Tennenbaum, Aaron M.; Langsam, Yedidyah; Augenstein, Moshe J. (1990), Data Structures Using C, Prentice Hall, pp. 456–461, pp. 472

[12] Design Manual of Link Street 88E6063 Integrated 7-Port Qos, 802.1Q 10/100 Ethernet Switch, Address Management pp. 57-61, Marvell

[13] Donald Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Second Edition. Addison-Wesley, 1998. ISBN 0-201-89685-0. Section 6.4: Hashing, pp.513–558

[14] How Caching Affects Hashing, Gregory L. Heileman Department of Electrical and Computer Engineering University of New Mexico, Albuquerque, NM, Wenbin Luo, Engineering Department St. Mary's University, San Antonio, TX

[15] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, Compilers: Principles, Techniques, and Tools (Reading, MA: Addison-Wesley,1986)

[16] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. Journal of Computer and System Sciences, 18(2): pp 143-154, 1979.

[17] Jerey D. Ullman. A note on the efficiency of hash functions. Journal of the ACM, pp. 569-575, 1972

[18] Andrew C. Yao. Uniform hashing is optimal. Journal of the ACM, pp.687-693, 1985

[19] L. Guibas and E. Szemeredi. The analysis of double hashing. Journal of Computer and Systems Sciences, pp.226-274, 1978.

[20] G. Lueker and M. Molodowitch. More analysis of double hashing. In Proceedings of the 20th Annual ACM Symposium on the Theory of Computing, pp.354-359, 1988.

[21] Heikki. Bjoerkman, Kari Lappalainen, Recommendation of Multi-Channel Ethernet Interface for Tiger Platform, ABB Oy, Distribution Automation 2008

[22] Volnei A. Pedroni , Circuit Design with VHDL. Chapter 1, Design Flow pp. 3-6, MIT Press, second edition.

[23] Design Flow with ModelSim-Altera or ModelSim Software, Mentor Graphics ModelSim Support, October 2007 v7.2.0 , Altera.

[24] Clock Analysis, Quartus II Classic Timing Analyzer, November 2008 v8.1.0, Altera.

[25] K.C Chang, Digital Systems Design with VHDL and Synthesis An Integrated Approach, Chapter 8, Synchronization Between Clock Domains, pp 238-242 IEEE Computer Society

[26] Christoph.Klarenbach, FPGA basierte Echtzeit-Ethernet-Anschaltung, Elektronik, Fachschrift fuer Industrielle Anwender und Entwickler, 9.December.2008