uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Mohamed Abou-Gabal
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Architecture for a Hardware Implementation of the OSPF Protocol

TITRE DE LA THÈSE / TITLE OF THESIS

D. Ionescu
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

V. Groza

I. Lambadaris

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Architecture for a Hardware Implementation of the OSPF Protocol

By

Mohamed Abou-Gabal

A thesis Submitted to
The Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering (SITE)

University of Ottawa
Ottawa, Ontario
January 2005

Your file Votre référence
ISBN: 0-494-11197-6
Our file Notre référence
ISBN: 0-494-11197-6

Table of Contents

List of Tables

List of Figures

Legend

Term	Definition
ABR	Area Border Router.
ALU	Arithmetic Logic Unit.
AR	Arc list Register.
Arc	A link between two nodes.
AS	Autonomous System.
ASBR	AS Boundary Router.
ASIC	Application Specific Integrated Circuit.
BGP	Border Gateway Protocol.
CE	Control Elements.
DD	Database Description.
Demux	Demultiplexer.
DR	Distance label Register.
Dynamic Memory	Refer to the registers (AR, DR, PR, SPR, SR).
EGP	Edge Gateway Protocol.
FPGA	Field Programmable Gate Array.
GE	Gigabit Ethernet.
GR	General Register.
GSPS	Group Shortest Path Set.
HPNNI	Hierarchical Peer Network to Network Interface.
IP	Internet Protocol.
IS-IS	Integrated Scientific Information System
LSA	Link State Advertisement.
LSAck	Link State Acknowledgement.
LSB	Least Significant Bits.
LSReq	Link State Request.
LSUpd	Link State Update.
MSB	Most Signifiant Bits.
Mux	Multiplexer.
Node Id	Used to describe Node index in an array.
OSPF	Open Shortest Path First.
PE	Processing Elements.
PNNI	Peer Network to Network Interface.
PR	Predecessor Register.
QoS	Quality of Service.
RAM	Random Access Memory.
SP	Shortest Path.
SPP	Shortest Path Processor.
SPR	Refers to static memory address register.
SR	Status Register.
Static Memory	Refers to the memory where the network topology is stored.
ToS	Type of Service.

Abstract

The shortest path problem is common in many different fields (transportation systems, mechanical systems, etc). Most of the telecommunication industry protocols such as PNNI, OSPF and IS-IS use Dijkstra's algorithm or Bellman-Ford's algorithm to solve the shortest path problem. Today, the majority of the shortest path computations are performed in software, which is inefficient for real-time (voice and multimedia over IP) applications that are sensitive to delay. This research proposes a hardware architecture for the OSPF protocol. It also provides a hardware based shortest path processor architecture which is used by the OSPF system.

Acknowledgments

I would like to thank my supervisor, Dr. Dan Ionescu, for his guidance, encouragement, optimism and support throughout my research. His ability to set lofty goals and challenge one to achieve them is responsible for the magnitude of this thesis.

Special thanks go to Raymond Peterkin for his continuous support through-out my research. Also, I would like to thank my parents (Rafiek and Seham), sister (Rania) and brothers (Mahmoud, Ahmed, Salah and Ramee) for their daily support.

1 Introduction

1.1 Motivation and research objectives

Today's packet networks are becoming increasingly complex. Consequently, poor performance and scalability issues are being experienced. Poor performance implies long delays and low network throughput. In order to overcome and engineer some of these performance issues, hierarchical and logical routing architecture protocols were developed in the area of packet networks. Examples of IP routing protocols are Open Shortest Path First (OSPF) [1], Intermediate-System to Intermediate-System (IS-IS) [2], Border Gateway Protocol (BGP) [3] and edge gateway protocol (EGP) [4]. An example of an ATM routing protocol is Peer Network to Network interface (PNNI) [5].

A network experiences large delays due to bottlenecks on system resources such as memory, CPU and bandwidth. Larger memory helps in minimizing packet overflow and packet discards in a node. Moreover, to address memory limitations and packet overflow, traffic management shaping and policing algorithms (such as InterServ, DiffServ [6], Bandwidth Brokers [7], Leaky Bucket, Token Bucket and Weighted Fair Queuing [8], [9]) have been developed. Faster CPU speed plays a major role in a node's throughput. To address CPU limitations, many of today's packet networks are equipped with network processors [10]. These processors can be programmed to process packets, check for damaged packets, error flow, execute control flow and perform other CPU intensive tasks. Thanks to optical technology, today's networks no longer experience bandwidth shortage. In fact, most of today's deployed optical networks have very little bandwidth consumption.

Since most of the system bottlenecks have been addressed and there is still an end-to-end delay in a network which is crucial for services like voice and multimedia. This fact leads the motivation of this research to optimize algorithms by implementing them in hardware to meet the increased demand on quality of service (QoS). Furthermore, traffic must be quickly

re-routed in the presence of a link failure, especially in high speed (Gigabit Ethernet (GE)) networks. A one second delay in a GE network means massive data loss.

Since this research focuses only on the hardware implementation, only hardware related solutions will be discussed in this thesis. Numerous researches in this domain are limited to algorithms with architectures that enhance the arithmetic operations such as matrix multiplication [11] and fast division algorithms [12]. Other research as in [13] focuses on memory structural designs that are suitable for network graphs storage.

One of the hardware solutions that were proposed to solve the shortest path problem is the "HAGAR: Efficient Multi-Context Graph Processors" [14] where the architecture of the solution involves a microprocessor and FPGA co-working together. The solution assumed that the network graph was a direct graph and had all edges with cost equal to 1. The article presented a shortest path algorithm in hardware, which was composed of two hardware arrays (registers), RAM where the contexts were stored and a Register File where the shortest traversed path was saved. The mechanism used to find shortest path was efficient. However, it does not solve the problem of shortest path for edges or arcs that are not unique in distance.

Matti and Jorma [15] have presented a FPGA-based version of Dijkstra's algorithm and a performance comparison between the FPGA-based and a microprocessor-based version of the same algorithm. The end result of their study illustrated that the FPGA-based version of Dijkstra's shortest path algorithm was ten times faster than the microprocessor based version.

Latest research, [16], [17], introduced a hardware matrix architecture that consists of processing elements (PE) and control elements (CE). This solution is adequate when the network topology has a small number of nodes and the link costs are not large. The performance of this architecture is heavily dependent on the number of nodes and link cost magnitude, which may not be too scalable if deployed in large networks. Other researchers focus more on providing a network processor performance benchmark guidelines [18], [19] by modeling and analyzing network processor [20], [21].

Further details on the above mentioned researches are given in section 2.2. The novelty in this research presents a Field Programmable Gate Array (FPGA) hardware architecture for OSPF to minimize time computation while calculating the shortest path. The reasons this study is going with an FPGA based solution instead of an Application Specific Integrated Circuit (ASIC) are:

1) The industry has shifted its focus towards IPv6 where IP addresses are 128 bits. Hence, an FPGA based system designed with IPv4 can be easily upgraded to IPv6 through reconfigurable computing.

2) The design and implementation costs are lowered significantly and therefore it allows for the possibility of implementing an optimized design on an ASIC in the future.

Unlike the aforementioned solutions, this architecture allows for a network topology to be directly entered or removed as OSPF protocol performs link state advertisements (LSAs).

1.2 Contributions of this thesis

The main contribution in this study is the hardware architecture implementation for the OSPF routing protocol system. To the best of the author's knowledge, this work has not been considered in the open literature by any other researcher.

Specific contributions include:

1) A novel a hardware architecture solution for the OSPF protocol. The hardware solution presented improves the performance of the OSPF protocol. The design of the hardware architecture was achieved by tackling the implementation from the following prospective: i) Network level, which deals with the interactions and operations of multiple nodes. ii) Node level, where router state machine is designed to full-fill the operation requirements at the network level. iii) Port level, where an internal messaging mechanism was developed to ensure that the router process OSPF packets properly.
2) Developed Shortest Path Processor (SPP) module that is interoperable with other routing protocol system such as IS-IS and PNNI.

Publications [22], [23], [24], deals with shortest path computation using the proposed architectures. [22] describes the SPP module architecture which is based on Dijkstra's algorithm. [23] provides an architecture proposed for the IS-IS routing protocol system. Finally, [24] presents an architecture proposed for the OSPF routing protocol system

1.3 Organization of the thesis

The thesis is organized to provide an incremental understanding of the problem of performing shortest path computations for the OSPF protocol. Mechanisms for improving the functionality of OSPF and determining the shortest path are presented along with implementation results.

Chapter 2 describes the shortest path problem and its solution using Dijkstra's algorithm. Examples of shortest path problems are expressed mathematically and graphically. Furthermore, background of related worked from the open literature is provided.

Chapter 3 provides a brief overview of OSPF protocol. This overview includes a general description of the protocol in addition to a router state machine representation, packet types

and formats. Various scenarios and examples are examined to illustrate the concepts more clearly.

Chapter 4 describes the OSPF hardware architecture and implementation. All relevant design decisions are given as well as an overview of the entire system, including the control unit and data path designs.

Chapter 5 describes the shortest path processor module design. As with the OSPF architecture description, the control unit and data path are fully described.

Chapter 6 provides analysis and experimental results. Results are given for both the OSPF system and SPP module. Observable trends and their explanations are also offered.

Finally, in Chapter 7 the dissertations is concluded and some suggestion for future work are presented.

2 Shortest path problem and reconfigurable computing with computer communications

One of the major OSPF protocol requirements is to execute a shortest path algorithm that finds the least cost distance from one router to all other routers. This section describes the shortest path problem and its solution.

2.1 Shortest path problem description and solution

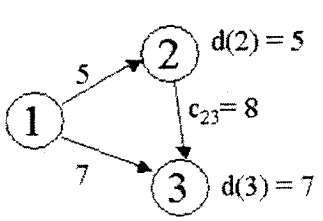
The shortest path problem exists in many different fields and there are many algorithms that address this problem. Some of the most popular algorithms are Dijkstra [25], Bellman-Ford, Prim and others that are listed in [26], [27], [28], [29], [30], [31], [32], [33]. Each one of these algorithms comes with different implementations to optimize its running time. For example, in the case of Dijkstra's algorithm, Table 1 presents a variety of implementations with their computation time complexity. Table 2 presents some graph notations that are used in the upcoming subsections. Furthermore, some of these notations are consolidated together into Figure 1 for illustration purposes.

Table 1 Different implementation of Dijkstra's algorithm [34]

Dijkstra Algorithm Implementations	Time Complexity
Original	$O(n^2)$
Dial's	$O(m + nC)$
d-Heap	$O(m \log_d n)$, where $d = m/n$
Fibonacci heap	$O(m + n \log n)$
Radix heap	$O(m + n \log (nC))$

Table 2 Graph notations and their description

Notation	Description
A	Symbolizes a set of arcs
A(i)	Denotes an arc list of node i.
C	The largest arc cost.
c_{ij}	The cost of arc between node i and node j.
d(i)	A numerical distance label of node i.
G	Denotes a graph.
M	Total number of arcs in a graph.
N	Represents a set of nodes in a network.
n	Total number of nodes in a graph.
S	An empty array.
s	Symbolizes the source node.
t	Denotes the termination node.
O(1)	The time computation. O(1) means the computation time is in unit of 1.
pred(i) = j	In a tree graph, this would mean that node j is the father of node I or node i is the child of node j. "pred" is short for predecessor.



- A(1) = {1-2, 1-3}
- $c_{12} = 5$
- N = {1,2,3}
- d(2) = 5
- n = 3
- m = 3

Figure 1 Example of a network and its notations.

Part of this thesis is dedicated towards Dijkstra's algorithm and therefore only Dijkstra's algorithm will be explored. In the next subsection, the shortest path problem is described mathematically followed by an explanation of Dijkstra's algorithm.

2.1.1 Mathematical model of the shortest path problem

Using notations shown in Table 2, the shortest path problem can be modeled as follows

$$G = (N,A) \quad (1)$$

$$\{x_{ij} \mid (i,j) \in A\} \quad (2)$$

where x_{ij} denotes the arc connecting the node i to node j (x_{ij} is equal to value of one). Hence, the shortest path problem is described as the minimization of the following equation

$$\sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3}$$

Equation (3) is minimized subject to the following constraints:

$$\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = \begin{cases} 1, i = s \\ -1, i = t \\ 0, otherwise \end{cases} \tag{4}$$

$$0 \le x_{ij}, \forall (i,j) \in A \tag{5}$$

For example, the shortest path for network shown in Figure 2 with s = 1 and t = 4, can be described as

Minimize : $5\,X_{12} + 7\,X_{13} + 8\,X_{23} + X_{24} + X_{34}$

Then, applying the constraints:

$$\begin{aligned} X_{12} + X_{13} &= 1 \\ -X_{24} - X_{34} &= -1 \\ X_{12} - X_{23} - X_{24} &= 0 \\ X_{13} + X_{23} - X_{34} &= 0 \end{aligned}$$

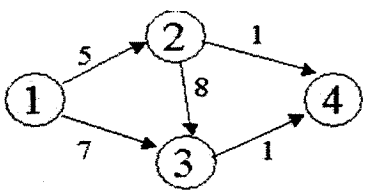


Figure 2 Network graph example

2.1.2 Dijkstra's shortest path algorithm

The original Dijkstra's algorithm developed back in 1959 is shown in Figure 3. The algorithm inputs are an array "N" of "n" nodes and an array of arcs "A" of "m" arcs. The algorithm output is a directed out-tree, rooted at the source and is connected to all other nodes with finite distance labels.

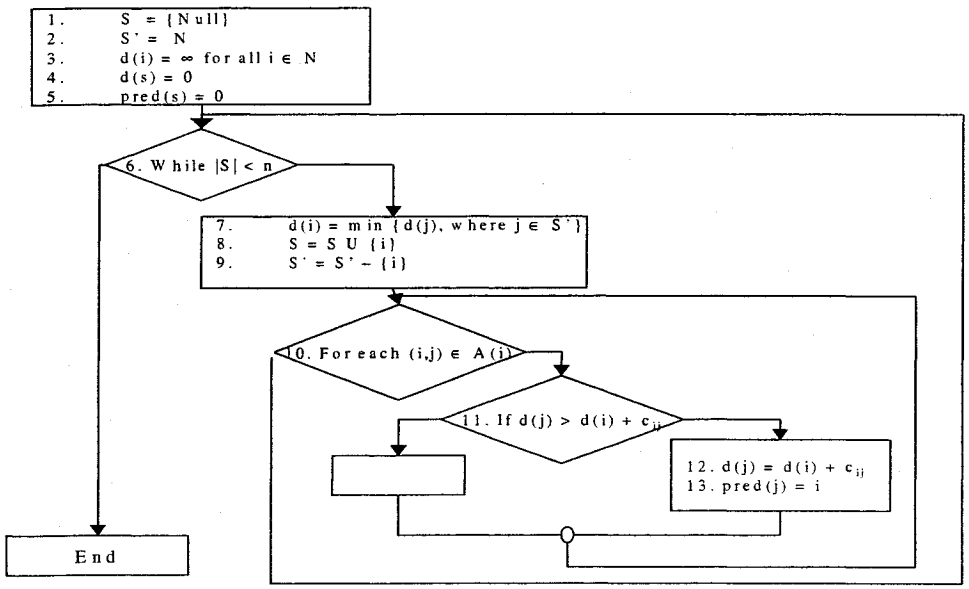


Figure 3 Flow chart of Dijkstra's algorithm

To explain the operation of this algorithm, assume the network graph shown in Figure 2 is given. The requirement is to find the shortest path from source node "1" to all the other nodes.

In Figure 3, S at step 1 is an empty array, S' contains {1,2,3,4}, N includes {1,2,3,4} and A is set to {(1,2), (1,3), (2,3), (2,4), (3,4)}. All nodes will have a distance label equal to infinity and the tree has a root set to node 1 (see Figure 4(a)). Since node 1 is selected, its arcs are processed first and as a result the distance label for arc (1,2) is set to 5 and arc (1,3) is set to 7 (see Figure 4 (b) and (c)). After completing the distance updates, all distance labels are compared and since node 2 has the lowest distance, it is added to the set of processed nodes (which is array S) and then iteration continue. Node 2 arcs are processed and updated as

shown in Figure 4 (d) and (e). Then, node 4 is selected because it has the lower distance label than node 3. However, since node 4 have no arcs, the algorithm moves on and selects node 3 for arc processing. Finally, from the predecessor initializations (statement 13), a final direct-out path tree can be drawn as shown in Figure 4 (g).

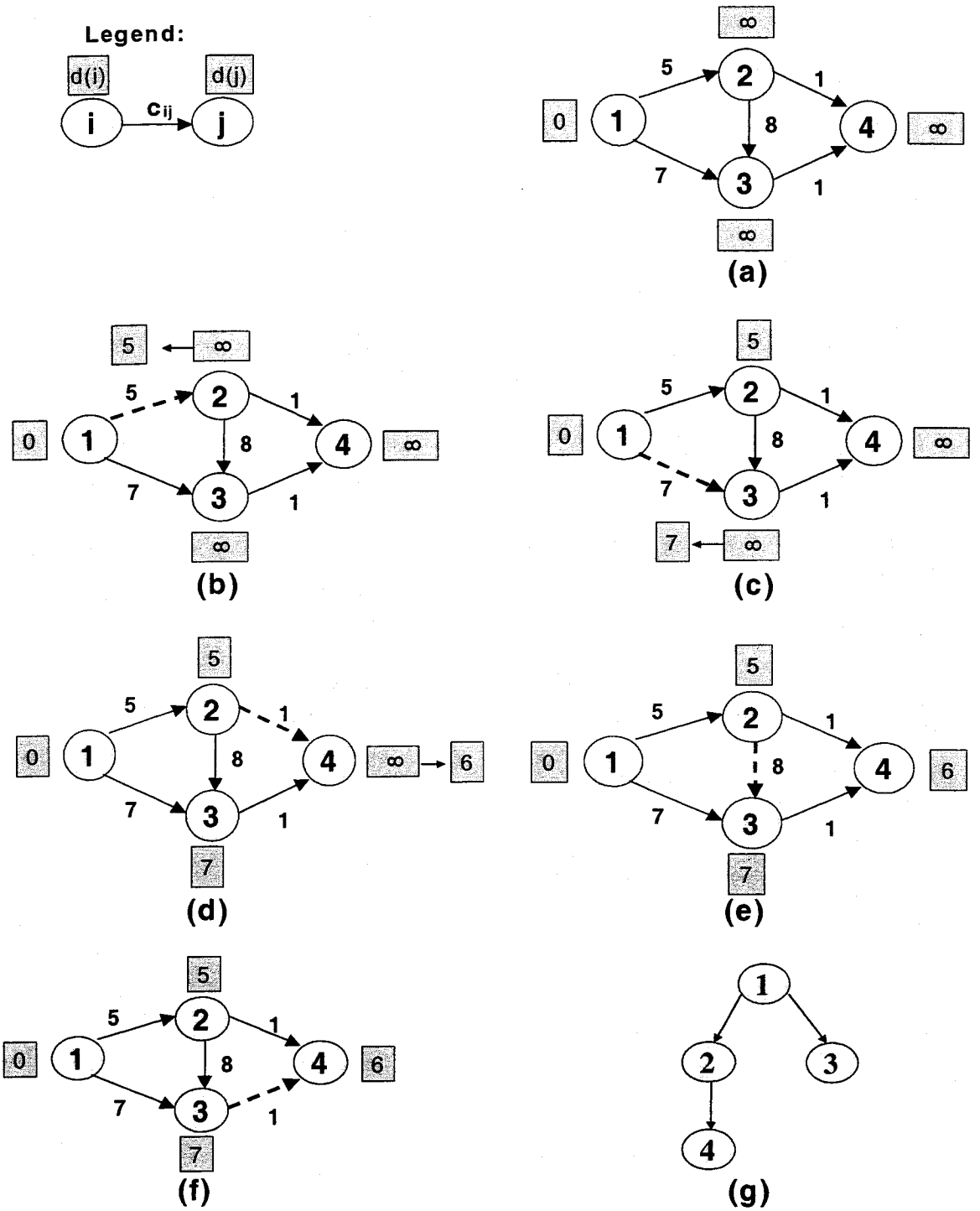


Figure 4 Example of Dijkstra's Algorithm.

2.2 Related work

2.2.1 FPGA-based implementation

The suitability of reconfigurable computing architectures for computer communications and networking protocols in general is discussed in [15]. Increasing QoS requires the efficient use of computing resources as much as possible. Abandoning software solutions as much as possible for alternatives like microprocessor based solutions and FPGA based technologies is explored in this paper.

When considering routing types and their applicability to reconfigurable computing, adaptive routing seems most appropriate. Routing algorithms are run in reconfigurable hardware where parallelism is fully exploited so the speed up is most observable. When network conditions change, different routing algorithms can be swapped in and run on the same hardware. Reconfigurable hardware can support various types of hardware algorithms to account for varying network conditions. Central and distributed algorithms can be adaptive. Centralized implementations relieve node computation and are vulnerable to the state of the single processing centre while distributed implementations have a large tolerance to failures but are susceptible to oscillations when link costs change.

Various algorithms exist to find the shortest path for a given network topology. Dijkstra's algorithm is the most common and well known for finding the shortest path from one node to all other nodes in a network. Open Shortest Path First (OSPF) uses Dijkstra's algorithm in its computations. Computation time for Dijkstra's algorithm is $O(|N^2|)$. That complexity can be reduced with the implementation of different data structures like priority queues but their implementation is particularly costly in reconfigurable logic.

An FPGA based algorithm for Dijkstra's algorithm was implemented where the network topology was stored in ROM using a matrix format. A separate control block was implemented that was used to retrieve information about the network topology and send it to

a comparator bank. Based on the results of the comparators, results were temporarily stored in RAM. The FPGA based implementation was able to store a network topology of up to eight nodes. Since the topology was stored in ROM, there was no ability for the user to change the cost values without making internal changes to the structure of the FPGA design. Dijkstra's algorithm for eight nodes fit onto an Altera EPF10K20TC144-3 FPGA using 1152 logic devices (approximately 20000 gates). An equivalent solution with the same network topology was implemented in Linux Redhat 6.2 with gcc to observe the effects with microprocessor-based solution.

The speedup factor for the FPGA-based version was dependent on the number of nodes in the network. As the network size grew, the FPGA-based version grew linearly while the microprocessor-based version exhibited quadratic growth. In addition to the linearly trend that was observed, the FPGA-based version also displayed running times that were tens of times faster when processing networks of 8, 16, 32, and 64 nodes. The speed up factor increased from approximately 23 to approximately 67 as the size of the network grew.

So reconfigurable computing exhibited superior performance to microprocessor-based solution in terms of processing times and trends of growth for calculating the shortest path as the network became larger. While a lack of overhead is a contributing factor to the increased performance, parallelization in the FPGA that is not present in a microprocessor or software solution appears to be the main reason why such an improvement was observed. Any design that maximizes parallelization wherever possible will exhibit significant improvements in calculating the shortest path. A highly parallelized design that is robust in allowing users to modify network topologies (particularly the number of nodes) in a significant manner will provide a significant advantage in computer communications tasks where determining the path for network traffic is of primary concern.

2.2.2 Neural network based solution for determining the shortest path in hardware

A shortest path processor based on recurrent spatiotemporal neural network principles is described in [17]. The processor architecture is a collection of processing elements and control elements interconnected in rows and columns. The architecture has an equal number of rows and columns where each row has one control element in the column corresponding to the row number (i.e. row one has a control element in column one, row two has a control element in column two, etc.) So the control elements effectively form a diagonal line in the processor. The remaining elements of each row are processing elements. Processing elements in a given row and column represent the cost from traveling from the node with the column number to the node with the row number in a given network topology. So the cost of traveling from node 3 to node 1 would be held in the processing element in row one, column 3. Consequently the number of processing and control elements in the architecture limits the size of the network that can be processed.

Control elements effectively at as OR gates while processing elements are counters that hold the cost in question. The counter is modified whenever necessary to reflect the computations performed in determining the shortest path.

When the network topology is asymmetrical the proposed architecture can solve the shortest path problem. When the topology represents a symmetrical graph, the topology can be modified to eliminate approximately half the processing elements. In that scenario, all the processing elements below the diagonal of control elements could be eliminated to produce a more efficient architecture to determine the shortest path. So an architecture that previously had 25 elements (5 control elements, 20 processing elements) would only require 15 elements (5 control elements, 10 processing elements) when a symmetrical network is being processed.

In processing a five node network, the shortest path was calculated between 5 and 15 clock cycles inclusively in four separate scenarios where the shortest path was 2 or 3 nodes long from source to destination. When separate tests were performed involving networks with 5, 6, 7, and 8 nodes the logic element usage for the FPGA in question (Altera FLEX10K EPF10K20RC240-4) is 28%, 41%, 57% and 75% respectively.

While the proposed design calculates the shortest path with relatively few clock cycles, the number of nodes that can be processed is extremely limited. Processing networks of significantly more nodes (10, 20, 30, etc.) would require significant effort to implement. Given that an 8 node network required 75% of the logic elements of a FLEX10K FPGA, processing networks with more than 10 nodes is not possible. So implementing this solution for large networks would require highly advanced reconfigurable technology and the use of symmetric networks as much as possible to reduce the size of the architecture.

2.2.3 Optimized matrix based solution for shortest Path calculations

A proposal for a graph processor is given in [14] where the adjacency matrix representation of a graph is mapped into an array representation of the graph in hardware. Each cell of the hardware array represents an entry in the corresponding adjacency matrix. As with the prior approach, this architecture implements elements in rows and columns to determine the shortest path. The row and column numbers indicate the numbers of the nodes that could be connected to each other. The diagonal of this configuration is a series of switches (tri-state buffers) while the other elements are one-bit flip-flops with switches. Since the flip-flops are one bit, the value '1' represents a cost of one while the value '0' represents a cost of zero, indicating that the nodes in question are not connected.

Circuits were implemented on the Xilinx Virtex 300E FPGA to test reachability and determining the shortest path. Observed improvements were found by reducing the complexity of the algorithm involved. The reachability cycle time for the architecture was

relatively linear while the cycle time for the shortest path was less sensitive to the increase in the number of nodes. So increasing the number of nodes in the network topology had a relatively low effect on the total processing time. The total number of gates used for network topologies from approximately 5 to approximately 30 gates when processing network topologies of 4 to 32 nodes. The previous architecture used 872 logic elements for a network topology of eight nodes.

This architecture represents a significant improvement on neural network based design in terms of gate usage and computation time. The gate usage for this architecture is so significantly reduced that there is no significant limit to the number of nodes that can be processed in a conventional FPGA. However, a reimplementation of the entire architecture must still occur if more modes must be processed. The lack of scalability in this design can lead to significant implementation and verification time for a processor whose functionality remains identical with reach modification.

One issue with this architecture that does not exist with the neural network design is that there cannot be a cost exceeding '1' for any link in the network. The improved processing time is partly due to the fact that there is no differentiation among existing costs in the topology. Consequently, it is not possible for this architecture to calculate the shortest path of a network representing a realistic scenario. The improved processing time of this matrix architecture has come at the expense of being unable to find the topology of relevant networks.

3 OSPF protocol

The OSPF protocol summary provided below summarizes readings completed from [1] and [35], [36], [37], [38], [39], [40], [41], [42], [43]. The discussions below are organized in a fashion that provides the reader with a view on how OSPF functions at the network level, router level and port level. The first section provides a description of the OSPF network and outlines how the network elements operate with each other. The second section presents a description of an OSPF router state machine. The third section explains in details the different OSPF protocol packets.

3.1 OSPF network in a nutshell

OSPF is an interior gateway protocol and a link state protocol used to find the shortest path for an Internet Protocol (IP) autonomous system (AS). Figure 5 shows the position of OSPF from a protocol stack prospective.

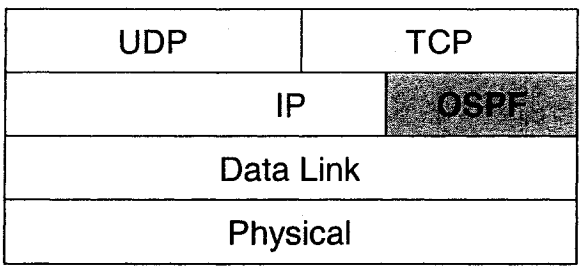


Figure 5 OSPF in the IP protocol stack [1]

One of the major advantages of deploying OSPF is that it reduces system overhead by only advertising network changes. However, OSPF is very resource (CPU and memory) intensive, largely due to the computation of the shortest path.

Using Figure 6 , the network elements of OSPF will be briefly explained.

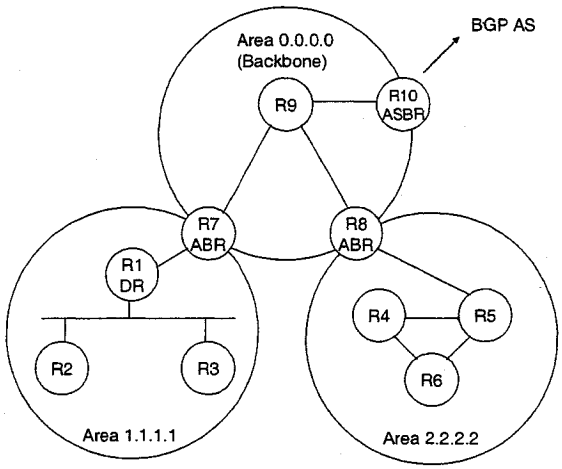


Figure 6 OSPF network

Figure 6 illustrates a network where OSPF is used by all the routers labeled R1 through R10. The network consists of three areas (0.0.0.0, 1.1.1.1, 2.2.2.2). All areas are connected through a backbone area, whose area Id must be 0.0.0.0. Area 1.1.1.1 represents a multi-access network. Hence, a designated router (DR) must be elected (during auto discovery) to cut down on the routing advertisement traffic and the workload on routers. In some multi-access networks, it is preferable to have a backup designated router (BDR). An Area Border Router (ABR) is used to connect multiple areas. In Figure 6, R7 and R8 are ABRs used to connect areas 1.1.1.1 and 2.2.2.2 to the backbone respectively. An AS Boundary Router (ASBR) is used to exchange information with routers from other ASs.

A link state advertisement (LSA) is used to describe the state of a router or a network. The LSAs heard and collected from neighboring routers form a topological database. There are many types of LSAs in OSPF. However, only four types are presented in this thesis. The first type is the router link state advertisement which is flooded within an area and it contains the neighbor router's links. For example, in area 1.1.1.1, this type of LSA will be exercised by routers, R1, R2, R3 and R7. The second type of LSA is the network link state advertisement which is flooded within an area by the DR. It includes information regarding all routers on its multi-access network. The only router that generates a type two LSA in Figure 6 is R1. The third type of LSA is the area summary link state advertisement. This

LSA is flooded into an area by an ABR and it describes the network accessibility from an outside area. From Figure 6, Routers R7 and R8 advertise type three LSAs. The final type of LSA presented is the ASBR summary link state advertisement. This type of LSA is flooded into an area by the ABR and it includes the cost from itself to an ASBR. From Figure 6, routers R7 and R8 advertise the final type of LSA.

In order to explain the operation of OSPF, assume that R6 in Figure 6 is a router that's being added to the area 2.2.2.2. Using Figure 7, a demonstration of the OSPF operation is provided.

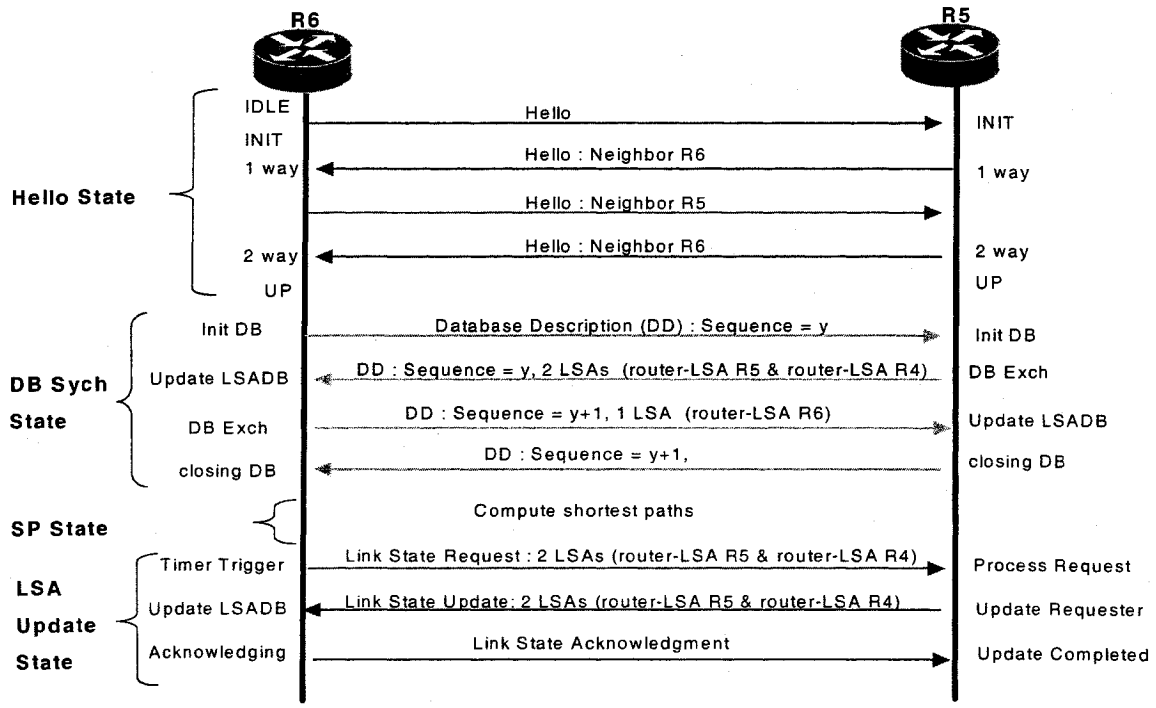


Figure 7 OSPF operation

The first state is the Hello state, where auto discovery occurs by the exchange of OSPF hello packets. R6 sends a Hello (with a null neighbor) packet to R5. R5 receives the Hello packet and checks if it is already in its database. If it is not in the database then R5 responds with a Hello packet. Otherwise, R5 disregards the Hello packet. Upon receiving a Hello packet, R6 acknowledges R5 by sending a Hello (with neighbor = R5) packet. Finally, R5 acknowledges R6 by sending a Hello (with neighbor = R6) packet.

The second state is the database synchronization, where R6 and R5 will synchronize their databases by the exchange of Database Description (DD) packets. The DD packets are equipped with a sequence number field. The sequence number is a random integer assigned sequentially; it helps the receiver to determine if messages are missing. As shown in Figure 7, R6 initiates the DD with a sequence number y (an integer). R5 replies with DD packet and appending to it are the LSAs that are contained in its database. R6 receives the packets and stores the LSAs. R6 increments the sequence number and replies to R5 with router-LSA R6. R5 stores the received LSA and acknowledges R6 with an empty DD packet.

In the third state, Shortest Path (SP), both routers will be computing their shortest paths. The fourth state, LSA Update, is a state triggered by a timer known as LS Refresh Time (usually set to 30 minutes). In Figure 7, the R6 timer happens to trigger this event first. Hence, R6 sends a Link State Request (LsReq) packet, asking for a refresh on the two router LSAs. R5 responds to the LsReq with a Link State Update (LsUpd) packet. Upon receiving the LsUpd, R6 refreshes its database with the new LSAs and it acknowledges R5 with Link State Acknowledgement (LsAck). Following the LSA update state is the Shortest Path state. In this way, the shortest path is always updated.

3.2 OSPF router state machine

Figure 8 depicts a high level view of an OSPF router state machine. During node initialization, a router goes through two states known as Hello and Database Description. Upon completing these two states the router has discovered its neighbors and familiar with their link cost information. Therefore, the router proceeds by computing the shortest path and then goes into the IDLE state.

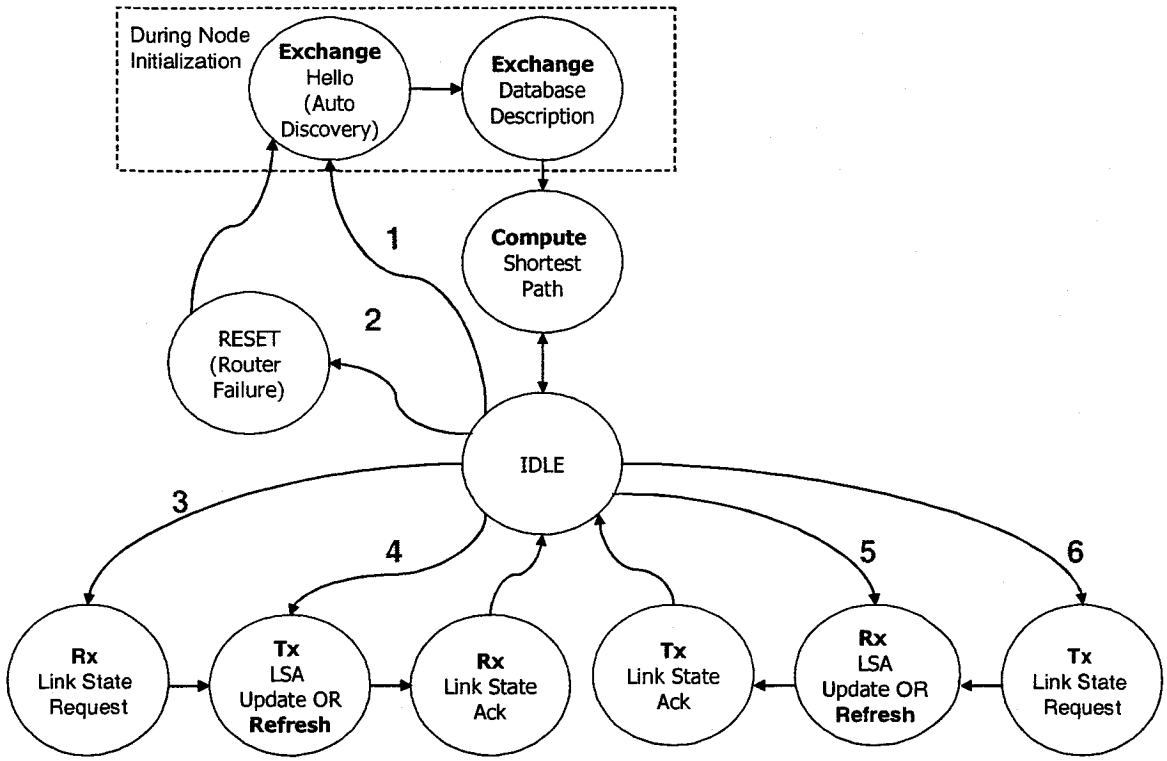


Figure 8 High level of OSPF router state machine

From the IDLE state there are six possible invocations scenarios. The first invocation shown in Figure 8 is that a new router has joined the area and it wants to establish adjacency with the current router. Hence, the current router goes into node initialization stage (described above). The second invocation is that the current router experienced some sort of failure and therefore it needs to go through the node initialization stage. The third possibility is a link request has arrived at the current router. The router looks up the LSA to see if there's any match. If a match is found, the current router transmits an LSA update containing the information found and then waits for an acknowledgement. The fourth possibility is when the current router needs to flood its LSAs to the network. This state is also known as Refresh, which occurs every 30 minutes. The fifth possible invocation occurs when an LSA update is received by the current router as a result of other routers LSA update Refreshes. Consequently, the router stores the LSA updates, re-computes its shortest path and replies to the advertising router with a link state acknowledgement packet. The sixth invocation is

triggered when a stored LSA age has reached one hour. In this case the router transmits a Link state request and then waits to receive an LSA.

3.3 OSPF protocol packets

This section explains the OSPF protocol packets types. There are two major breakdowns to the OSPF protocol packets:

- **OSPF packets,** which involve packets such as:
  - Hello,
  - Database Description (DD) ,
  - Link State Request (LsReq),
  - Link state Update (LsUpd) and
  - Link State Acknowledgement (LsAck).
- **Link state advertisement packets,** which involve packets such as:
  - Router LSA,
  - Network LSA,
  - Network or ASBR summary LSA and
  - AS external LSA.

All these packets are used among routers to communicate and exchange new network topologies or changes occurring within the network. The discussion below contains packet diagrams that depict the different OSPF protocol packets. Each of the packet diagrams is accompanied with a brief description of each field.

3.3.1 OSPF header

The OSPF header shown in Figure 9 is appended on top of each OSPF packet. Below are descriptions of each field:

- Version (8 bits): Since this thesis only deals with OSPF version 2 (OSPFv2). This field will be set to 2.
- Type (8 bits): Varies from 1 to 5. Please refer to Table 3 for details.
- Packet Length (16 bits): Total length (in bytes) including all the rest of packets preceding it (such as Hello, LSA Header, LSA, etc). Hence, packet length size will vary and it will depend on the scenario.
- Router ID (32 bits): Source IP address.
- Area ID (32 bits): Autonomous area Id.
- Checksum (32 bits): Allows receiving router to determine whether the packet was damaged.
- Authentication fields (total of 80 bits): Used for security purposes.

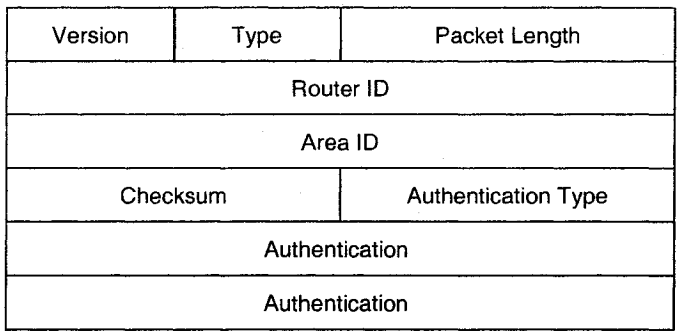


Figure 9 OSPF header

Table 3 OSPF Type field values and description

Type Value	Description
1	Hello packet.
2	Database description packet.
3	LS request packet.
4	LS update packet.
5	LS acknowledgement packet.

3.3.2 Link state advertisement header

The link state advertisement header shown in Figure 10 is appended on top of each LSA. Below are descriptions of each field:

- LS Age (16 bits): Time in seconds. It indicates the age of the LSA since it was generated. If LS Age reaches one hour the router will trash the LSA.
- Options (8 bits): These bits indicate that an LSA deserves a special treatment during flooding and path computations.
- LS Type (8 bits): Depends on the scenario and it goes hand in hand with Link state ID, please refer to Table 4 for further details.
- Link State ID (32 bits): Depends on the scenario and it goes hand in hand with LS Type, please refer to Table 4 for further details.
- Advertising Router (32 bits): Originating router's Id.
- LS Sequence Number: OSPFv2 uses linear space. Initially, the LS sequence number will be set to 0x80000001 and will be incremented by 1 until it reaches 0x7fffffff. If two instances are detect the router picks the LSA with largest sequence number and deletes the LSA with smallest sequence number.
- LS Checksum (16 bits): Allows receiving router to determine whether the LSA packet was damaged.
- Length (16 bits): The length (in bytes) of LSA and the LSA Header.

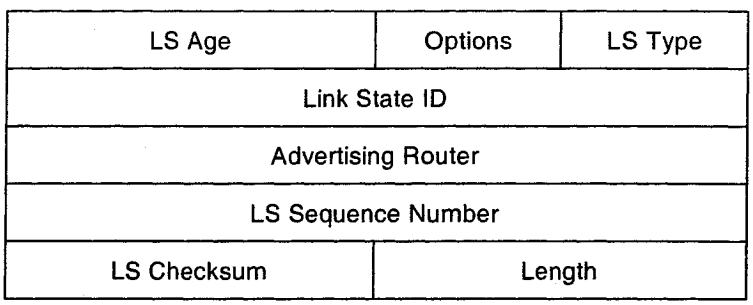


Figure 10 LSA header

Table 4 LS Type and Link State ID values

LS Type Value	Description	Link State ID Value
1	Router LSA.	Originating Router's Id.
2	Network LSA.	DR's IP Address.
3	Summary LSA (IP Network).	IP Network Address.
4	Summary LSA (ASBR).	ASBR Router Id.
5	AS External LSA.	IP Network Address.

3.3.3 OSPF packets

In this section a description of OSPF packets will be provided.

3.3.3.1 Hello packet

The hello packet shown in Figure 11 is used by routers during the auto-discovery (hello state) phase. Below are the fields descriptions:

- Network Mask (32 bits): This field is determined by the IP Class Type (A, B, etc).
- Hello Interval (16 bits): The normal period in seconds between hello messages.
- Options (8 bits): Refer to section 3.3.2 for details.
- Router Priority (8 bits): Integer used in selecting this router as Backup Designated Router.
- Dead Interval (32 bits): Time in seconds after which a non-responding neighbor is considered dead.
- Designated Router (32 bits): Router Id, used in multi-access networks (such as Ethernet)
- Backup Designated Router (32 bits): Router Id used when the designated router is no longer functional.
- Neighbor (32 bits): Router Id of all neighbors from which the sender has recently received hello messages.

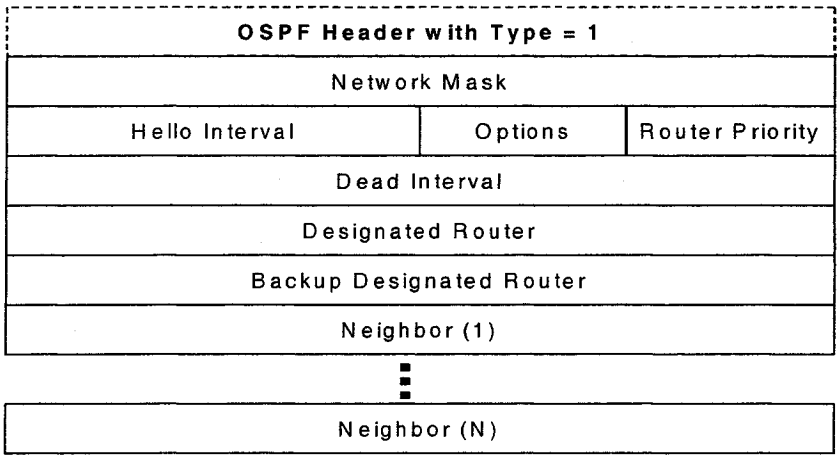


Figure 11 Hello packet

3.3.3.2 Database description packet

The database description packet shown in Figure 12 is used by routers to exchange network and link information. Below are the field descriptions:

- Options (6 bits): Refer to section 3.3.2.
- I (1 bit): When set to 1, it denotes initial message.
- M (1 bit): When set to 1, it denotes additional message.
- S (1 bit): When set to 0, it means that the router will act as a Slave (i.e., Router will acknowledge each received DD packet with an empty DD packet). If set to 1, this means that the router will act as a Master.
- DD Sequence Number (32 bits): Starts with arbitrary integer R and it's incremented with every additional message (i.e. with the next message should be R+1).

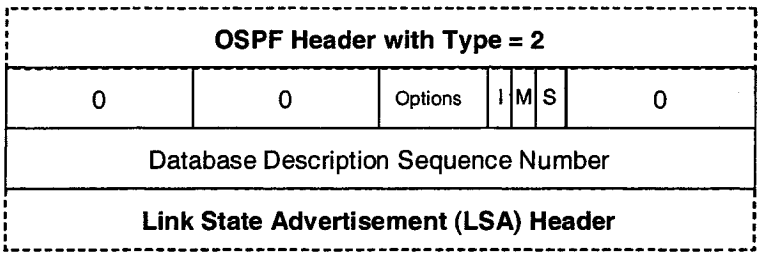


Figure 12 Database description packet

3.3.3.3 Link state request packet

The link state request packet shown in Figure 13 is used by routers to request other routers for an update on a specific link state. Below are the field descriptions:

- LS Type (32 bits): Refer to section 3.3.2.
- Link State ID (32 bits): Refer to section 3.3.2.
- Advertising Router (32 bits): Originating router's Id.

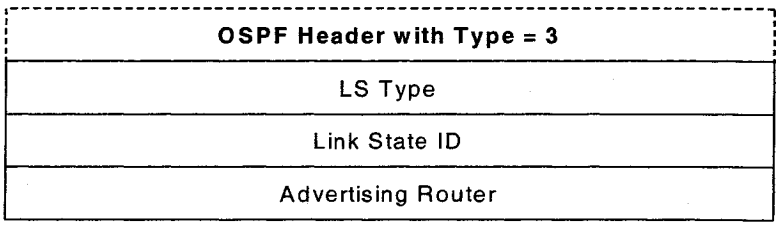


Figure 13 Link state request packet

3.3.3.4 Link state update packet

The Link state update packet shown in Figure 14 is used by routers into two scenarios. The first scenario is to respond to a link state request and the other scenario is when the router wants to send an LSA update refresh (which occurs every 30 minutes). Below are the field descriptions:

- Number of Advertisement (32 bits): Number of LSAs attached to this LSA update.
- LSA 1 to N (32 bits each): This depends on the LSA scenarios which are discussed in section 3.3.4.

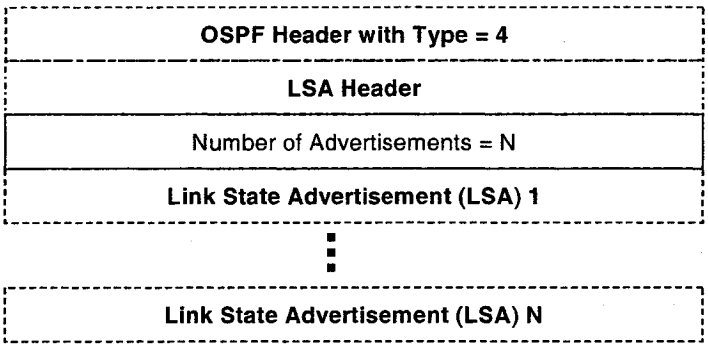


Figure 14 Link state update packet

3.3.3.5 Link state acknowledgement packet

The Link state acknowledgement packet shown in Figure 15 is used by routers to acknowledge each other when packets have been received successfully.

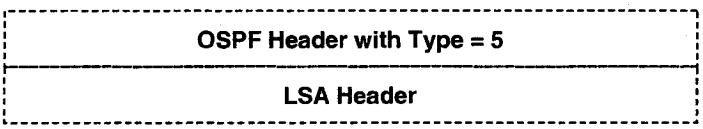


Figure 15 Link state acknowledgement packet

3.3.4 Link state advertisement packets

The link state advertisements packets are scenarios of LSA updates. In the sections below, descriptions of LSA types are provided.

3.3.4.1 Router LSA

The router LSA packet shown in Figure 16 is sent by a router to advertise the links attached to its direct neighbors. Below are the field descriptions:

- V: When set to 1, it refers to virtual link router
- E: When set to 1, it indicates ASBR router.
- B: When set to 1, it indicates ABR router.

- Number of Links: Number of links connected to the router.
- Link Type: The value varies and it has a relationship with Link ID and Link Data. Consult with Table 5 for more details.
- Link ID: The value varies and it has a relationship with Link Type and Link Data. Consult with Table 5 for more details.
- Link Data: The value varies and it has a relationship with Link Type and Link ID. Consult with for more details.
- Number of TOS: Number of Type of Service.
- Metric: Which translates to link cost.

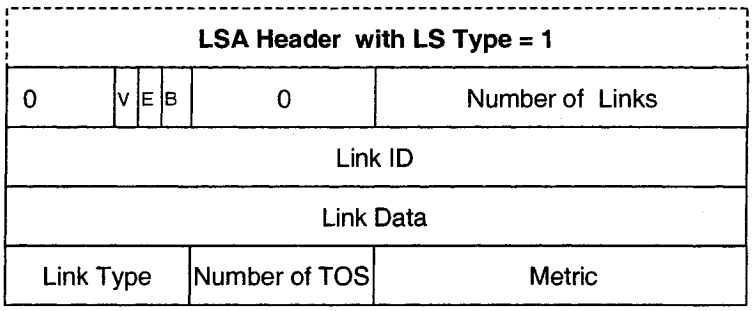


Figure 16 Router LSA

Table 5 Link Type, Link Id and Link Data Values and descriptions

Link Type Value	Description	Link ID	Link Data
1	Point-to-Point link	Neighbor Router ID	IP Address
2	Transit Network link.	IP Address of DR	IP Address
3	Stub Network link	IP Network or Subnet	Network Mask
4	Virtual link.	Neighbor Router ID	IP Address

3.3.4.2 Network LSA

The network LSA packet shown in Figure 17 is used by the designated router to flood information throughout an area. This packet describes all the routers attached to the multi-access network (such as Ethernet).

- Network Mask (32 bits): Refer to section 3.3.3.1.
- Attached Router(s) (32 bits): List of router Id(s) attached to this router.

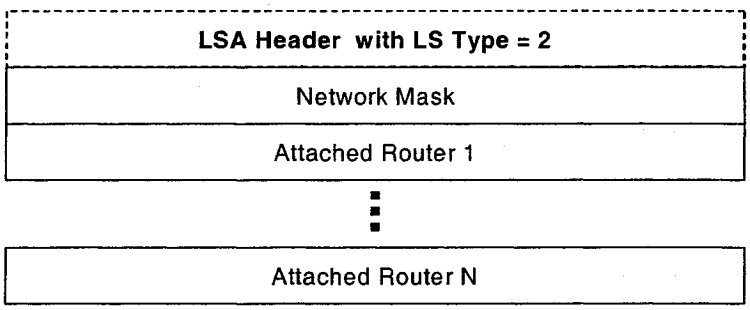


Figure 17 Network LSA

3.3.4.3 Network or ASBR summary LSA

In case of network summary LSA (Type 3), this packet is used by an ABR to flood information into other areas. The packet describes networks reachable from another area. In case of ASBR summary LSA (Type 4), the packet is used by an ABR to flood information into an area. It describes the distance from this router to the other ASBR. Below are the field descriptions:

- Network Mask (32 bits): refer to section 3.3.3.1.
- TOS (8 bits): Type of Service.
- Metric (24 bits): Link cost.

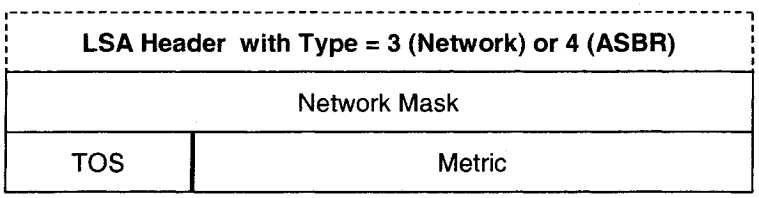


Figure 18 Network or ASBR summary LSA

3.3.4.4 AS external LSA

The AS external LSA packet shown in Figure 19 is used by ASBR to flood routes into external OSPF domains.

- Network Mask (32 bits): Refer to section 3.3.3.1.
- E (1 bit): If set to 0, it means External Type (i.e. ASBR). If set to 1, it means External Type 2 (i.e. Network).
- TOS (7 bits): Type of Service.
- Forwarding Address (32 bits): The address of the router on the other area (BGP, IS-IS or RIP).
- External Route Tag (32 bits): Used to convey information across an OSPF routing domain (i.e. info will be transparent to OSPF).

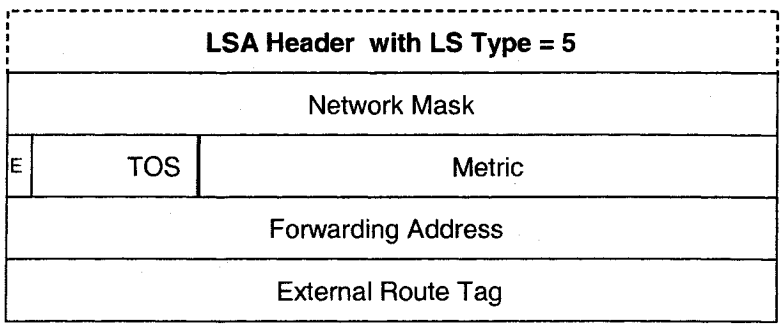


Figure 19 AS external LSA

3.4 OSPF packet example

This section provides an example of how incoming LS updates look like at port level. Using the same example shown in Figure 7, assume that the router Id for R5 is 10.10.10.5 (with link cost of 5) and the router Id for R4 is 10.10.10.4 (with link cost of 6). Hence, the incoming packets arriving at router R6 would like the following (bolded Xs means it is not important for this example):

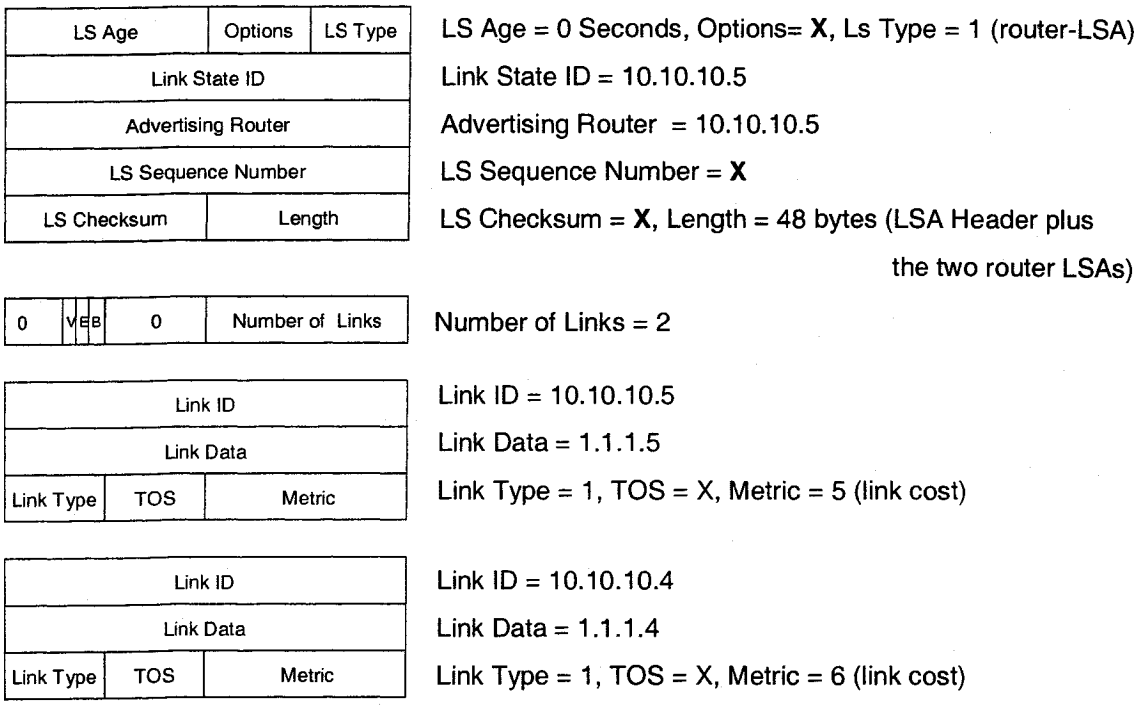


Figure 20 OSPF packet example

4 OSPF system implementation architecture

In this chapter, the OSPF system description is broken down into four major sections. The first section lays out the design decisions made. The second section provides roadmap design. The third section explains the OSPF system inputs, outputs and the high-level modules. The third section portrays the control unit behavioral model. Finally, the fourth section provides insights on the data path architecture. Hardware designed methodologies were explored from [44], [45], [46], [47].

4.1 Design decisions and criteria

Some design decisions were made in order to simplify the implementation of the OSPF protocol. These decisions are:

- The OSPF system will only work with OSPF packets. Hence, all Layer 3 (L3) and Layer 2 (L2) packets must be stripped off before entering the OSPF system (refer to Figure 5). L3 and L2 packets processing are beyond the scope of this thesis. Furthermore, the packets must be feed in segments of 32 bits every clock cycle.

- Since the shortest path is only computed within an autonomous system (area). Hence, only router LSAs will be processed in the implemented OSPF system. All other LSAs will be accepted and stored in the databases designed within the OSPF system.

- The number of nodes that will be supported is 128.

- No error detection (checksum) is considered in the designed OSPF system. This task should be done by other external modules connected to the OSPF system. Furthermore, this task needs to be executed before the OSPF system handles the incoming packets. In other words, these tasks are left for the data link layer.

- Authentications, Options and ToS fields presented in section 3.3 are accepted however not processed. The usage of these fields varies from one service provider to another. Hence, this can be performed as a future work where these parameters need to be provisioned through software.

4.2 Design Breakdown

To optimize the OSPF performance, the hardware architecture implementation is split into two major modules; the OSPF system and the Shortest Path Processor (SPP). The OSPF system is responsible for handling incoming and outgoing OSPF packets, while the SPP is responsible for computing shortest path. SPP architecture is based on Dijkstra's Algorithm.

The remaining of Chapter 4 will only focus on the OSPF system design and Chapter 5 will only focus on the SPP design. Figure 21, illustrates the connectivity between the two modules. Both modules are feed the same clock and reset control signals.

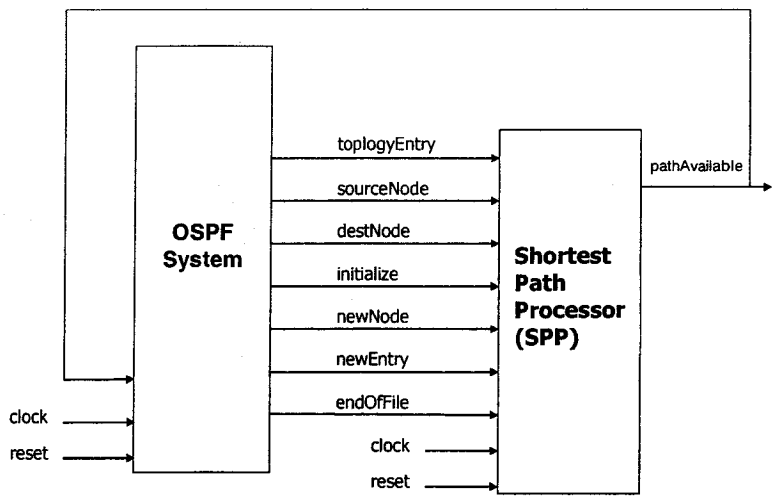


Figure 21 Connectivity of OSPF system and SPP module

4.3 OSPF System

4.3.1 OSPF system Input and Outputs

The OSPF system takes input and output signals as shown in Figure 22. Data signals are signals required to initialize the OSPF system. While, control signals (clock and reset) are usually provided by another module (oscillator) on the FPGA board. The OSPF system takes packets arriving at the "packetIn" bus, processes them and sends packets out via the "packetOut" bus. "routerId" and "areaId" are required to initialize the OSPF header packet. Signals "networkMask" through "backupDesignatedRouter" inclusive are data signals required to form a proper hello header. The "slaveMaster" input is required to inform the OSPF system whether it will be configured as a slave or as a master during the database description state. "routerType", "networkType" and "linkType" are fields required to initialize the LSA packets. The "ospfSystemState" is used for debugging purposes (it indicates to the user which state is the ospfSystem is running). The "topologyEntry" all the way to the "endOfFile" signals are used to synchronize the SPP module. Description of these signals are provided in section 0.

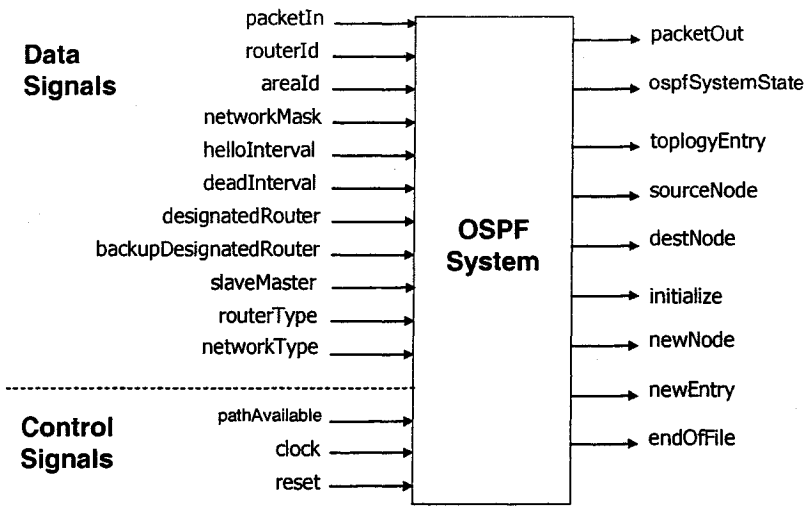


Figure 22 OSPF system inputs and outputs

4.3.2 OSPF System view

Figure 23 illustrates the interactions between high-level modules that are performing OSPF tasks. The Ingress processor stores data going into the system while the Egress processor performs packet assembly and sends packets out of the system. The lsaDb processor performs maintenance activities such as LS age checking and removing out of date LSAs from the memory. The synchronization of the OSPF protocol tasks are performed in the main processor module while the shortest path is computed in the shortest path processor (SPP) module. Details of the SPP design are provided in chapter 5.

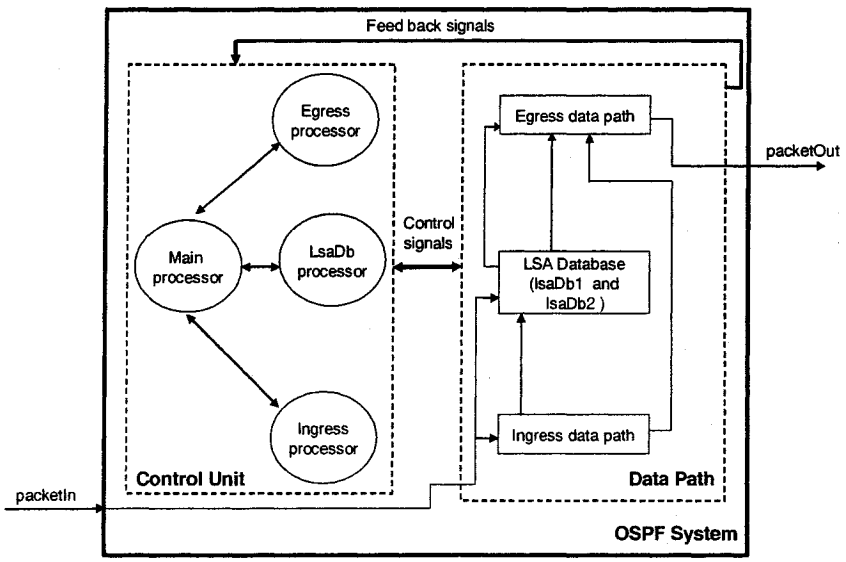


Figure 23 OSPF system high-level modules

4.4 Control unit design

The purpose of this section is to provide a zoomed in picture on the components of the control unit shown in Figure 23. Hence, an overview of the main processor, ingress processor, egress processor and lsaDb processor are provided.

4.4.1 Main processor

The core functionality of the main processor is to coordinate signal flows between internal processors (ingress, egress and lsaDb). During router initialization (sysINIT shown in Figure 24) the main processor performs specific tasks such as reading system inputs and sending commands to the egress processor. These commands force the egress processor to send out Hello packets and DD packets. After router initialization, the main processor moves to sysIDLE state where it basically coordinates control flow between internal processors. If the reset signal is set to high during any of the states mentioned above then the main processor move to the sysReset state.

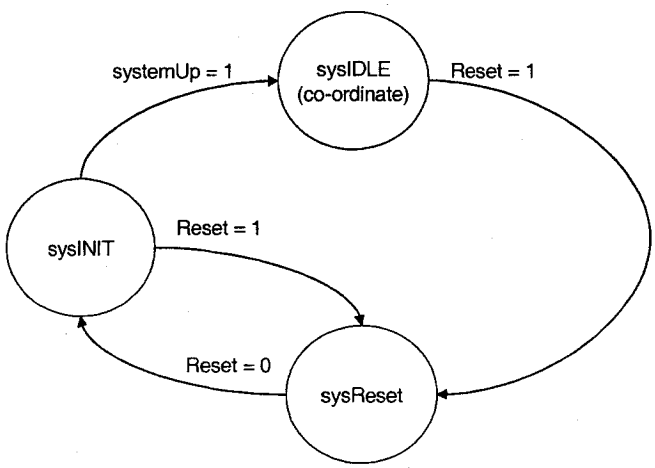


Figure 24 Main processor state machine

Figures 25, 26 and 27 explain the interactions and signal flows that occur during the sysIDLE state of the main processor. These figures denotes to the router state machine described in section 3.2 For example, set 1 shown in Figure 25 denotes the operation and signal flows that will be performed if the first invocation has occurred.

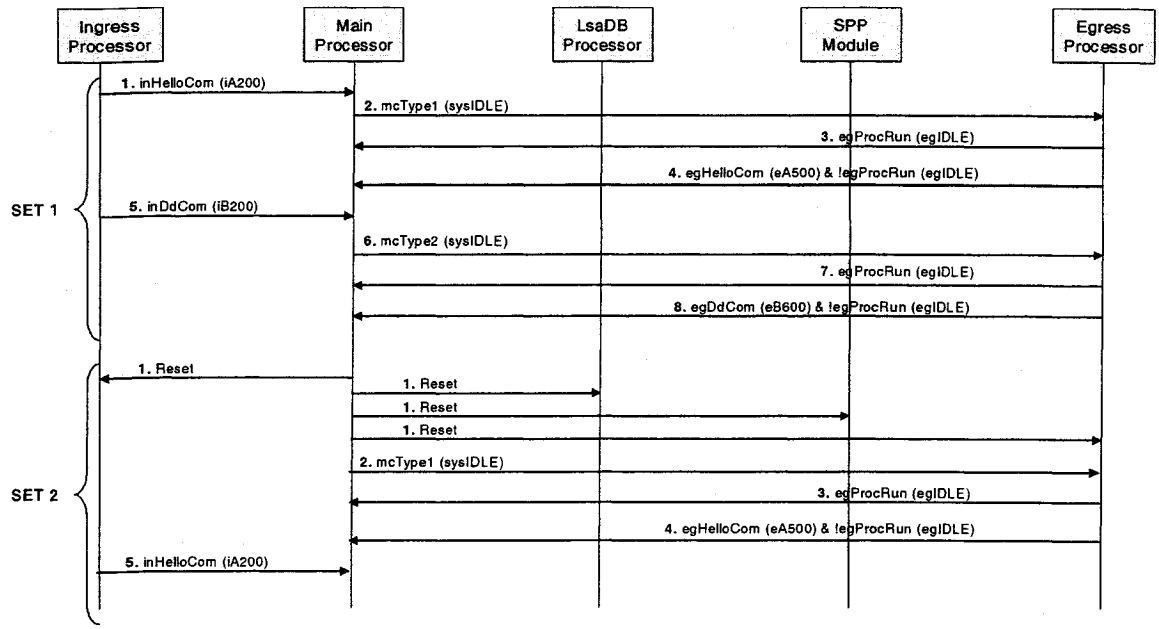


Figure 25 First two sets of router state machine

In Figure 25, set 1 depicts signal flows during Hello packet arrival. The inHelloCom signal is set high when the ingress processor receives the Hello packets. The main processor then immediately requests the egress processor to send out a Hello packet response. In return, the egress processor informs the main processor that it has begun assembling Hello packets and sets the egProcRun signal to one which indicates that the egress processor is busy. Once the Hello packet has been placed onto the egress path, the egress processor informs the main processor that the Hello packets were sent out and it is ready to accept the next command.

Set 2 in Figure 25 exemplifys the signal streams during router (system) failure. It is basically similar to set 1. In this scenario, the router engages the network by sending its own Hello packets first.

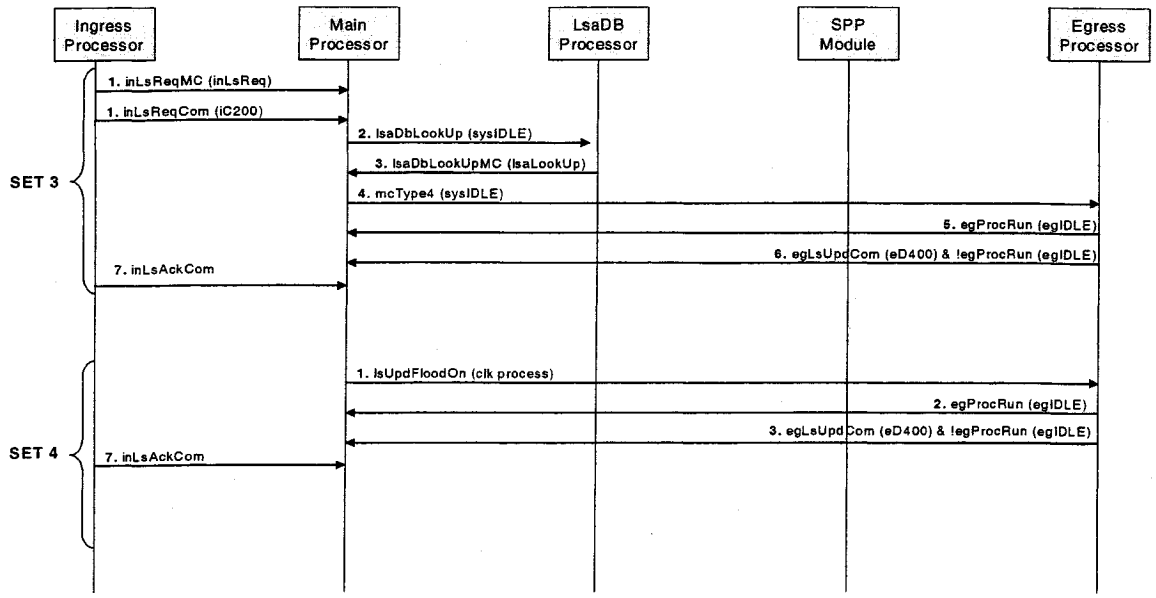


Figure 26 Set 3 and Set 4 of router state machine

Set 3 in Figure 26 illustrates the signal flows during the process of ingress LS requests. When inLsReqCom signal is set high, it informs the main processor that a request packet has arrived at the ingress path. Hence, the main processor invokes lsaDb processor to start looking up the request. Once look up has been completed, the lsaDb processor updates the main processor. Then, the main processor invokes the egress processor to send out LS update packets (this is done by setting mcType4 signal to high). Upon completing the transmission of egress LS update packets, the egress processor notifies the main processor that it is ready to process the next command (this is done by setting egProcRun signal to zero).

The process of LS Update flooding is demonstrated by set 4 in Figure 26. This state (LS Update flooding) is triggered by the timer processor which is always running within the OSPF system. The main processor invokes the egress processor to start transmitting the router's own LSAs. Once completed, the egress processor informs the main processor that it's ready to processor the next command.

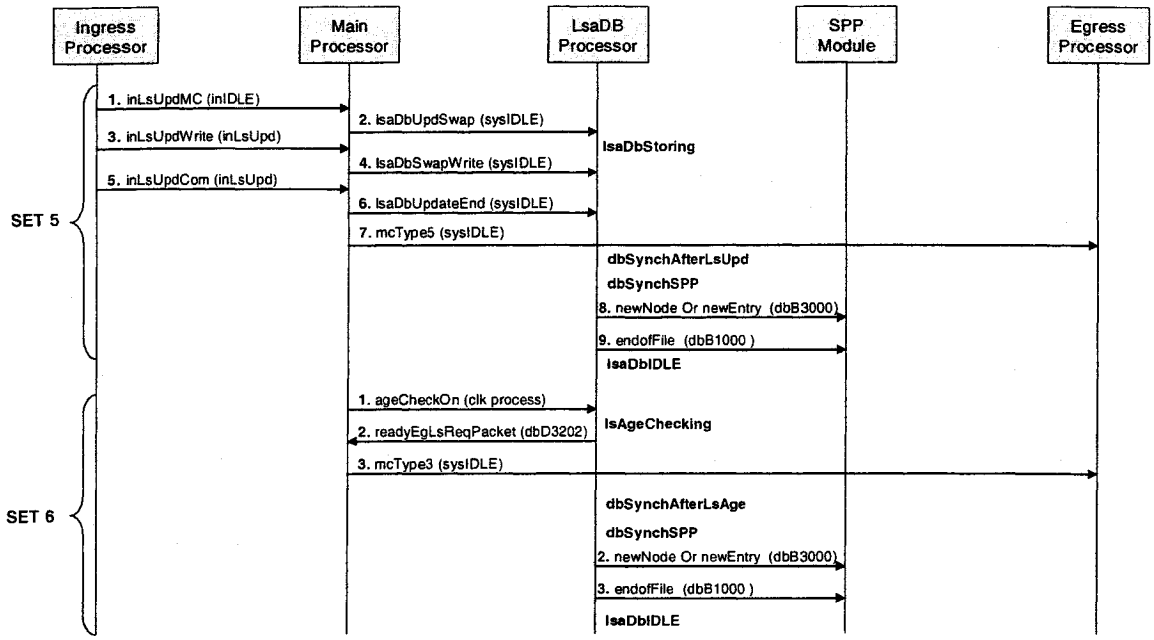


Figure 27 Set 5 and Set 6 of router state machine

Set 5 in Figure 27 shows the signal flows that are involved during the processing of ingress LS Update packets. The first signals (1 and 2) from the ingress processor and the main processor inform the lsaDb processor that it needs to swap the databases. In this way, the lsaDb processor will be writing the incoming LS Updates into a separate database. The second bundle of signals (3 and 4) notifies the lsaDb processor to start writing the LS updates. The third bundle of signals (5 and 6) informs the lsaDb processor to stop writing to the database. Then, the main processor will invoke the egress processor to assemble and transmit an LsAck packet. In parallel to the LsAck task, the lsaDb processor synchronizes its databases (lsaDb1 and lsaDb2) and then synchronizes the lsaDb1 database with the SPP database. The synchronization with the SPP is performed using newNode and newEntry control signals. Once the lsaDb processor has completed the synchronization, it sets the endofFile control signal to high which causes the SPP to compute the shortest path.

The procedure of LSA age checking processing is demonstrated through set 6 in Figure 27. The purpose of the LSA age checking is to remove any LSAs that are over half hour in age and it's triggered by the timer processor. The timer processor sets ageCheckOn to high which forces the lsaDB processor to start the lsAgeChecking state. Once the lsaDb processor has completed this state, the egLsReq queue will be populated with the LsReq packets. The main

processor is notified (signal 2) and in return it invokes the egress processor to transmit the LsReq packets. In parallel to this activity, the lsaDb processor synchronizes its databases (lsaDb1 and lsaDb2) and then synchronizes the lsaDb1 database with the SPP. In this way, the shortest paths are reflecting active links only.

Figure 28 illustrates the main actions performed by the timer processor. The timer processor is responsible to invoke the lsAgeChecking state by setting the ageCheckOn signal to one every 45 minutes. Moreover, it is responsible to set the lsUpdFloodOn signal to one every 30 minutes, which causes the OSPF system to flood its own LSAs to the network.

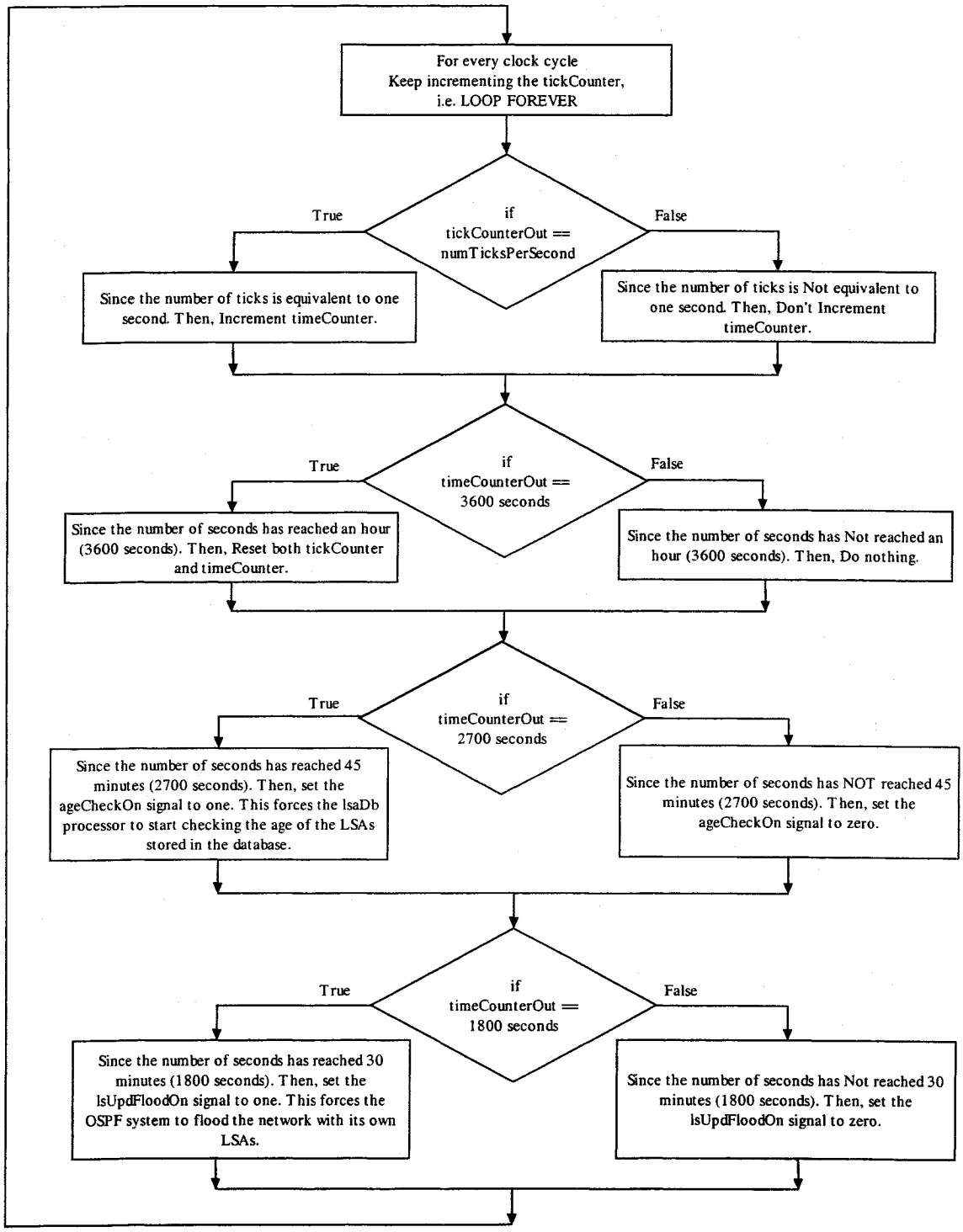


Figure 28 Timer processor flow chart

4.4.2 Ingress and egress processors

The ingress processor and egress processor are designed in the manner shown in Figure 27 and Figure 31. The ingress components are denoted by "in" and the egress components are denoted by "eg". For example, in Figure 27 the "inHello" state would mean ingress Hello. The ingress processor constantly checks incoming packets in state inIDLE. Since there are five types of OSPF packets, the design has dedicated five major processing states, one for each OSPF packet type.

The inIDLE state shown in Figure 29 reads the OSPF headers and determines the type of packet by reading the Type field (refer to Figure 9). Once the type has been determined, the inIDLE state invokes the appropriate OSPF related state (inHello, inDD, inLsReq, inLsUpd or inLsAck). Each one of the OSPF related states is composed of commands that act on the ingress data path. Once an OSPF related state is completed, the ingress processor rolls back to the inIDLE state and feedback signals are sent to the main processor to indicate that an OSPF state was accomplished.

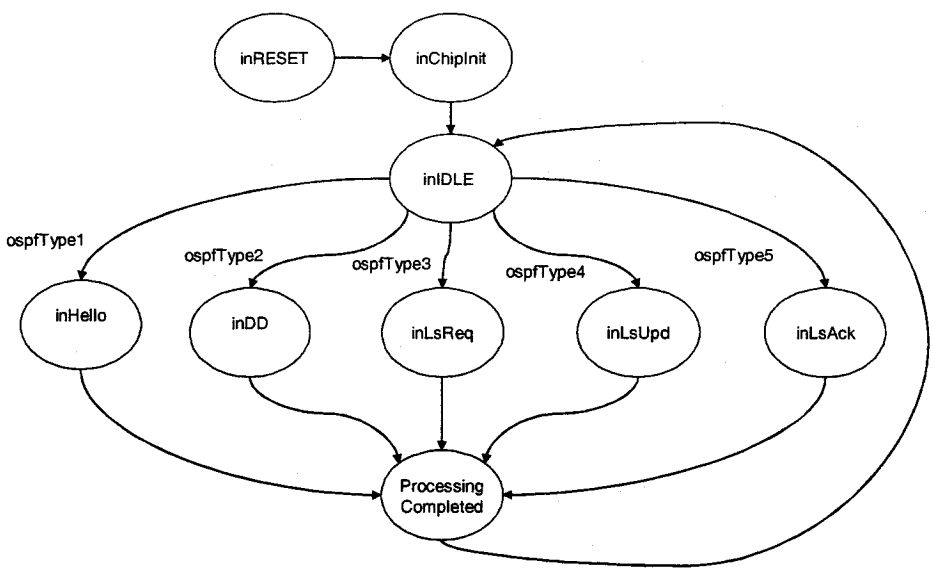


Figure 29 Ingress processor state machine

The flow chart shown in Figure 30 illustrates the type of control commands that are acting on the data path during the inHello state. When the inIDLE state classifies the packet, it also loads the inPacketLength counter with the packet length field.

Figure 30 shows, the inPacketLength counter is used to determine whether all Hello packet segments have been received. The inHelloC21 counter is used to point to the next available free space in the inHello register. Hence, the inHelloC21 counter is incremented each time after storing segments (32 bits) of a Hello packet. The inHelloC22 counter is used to track the number of Hello packets received from external routers.

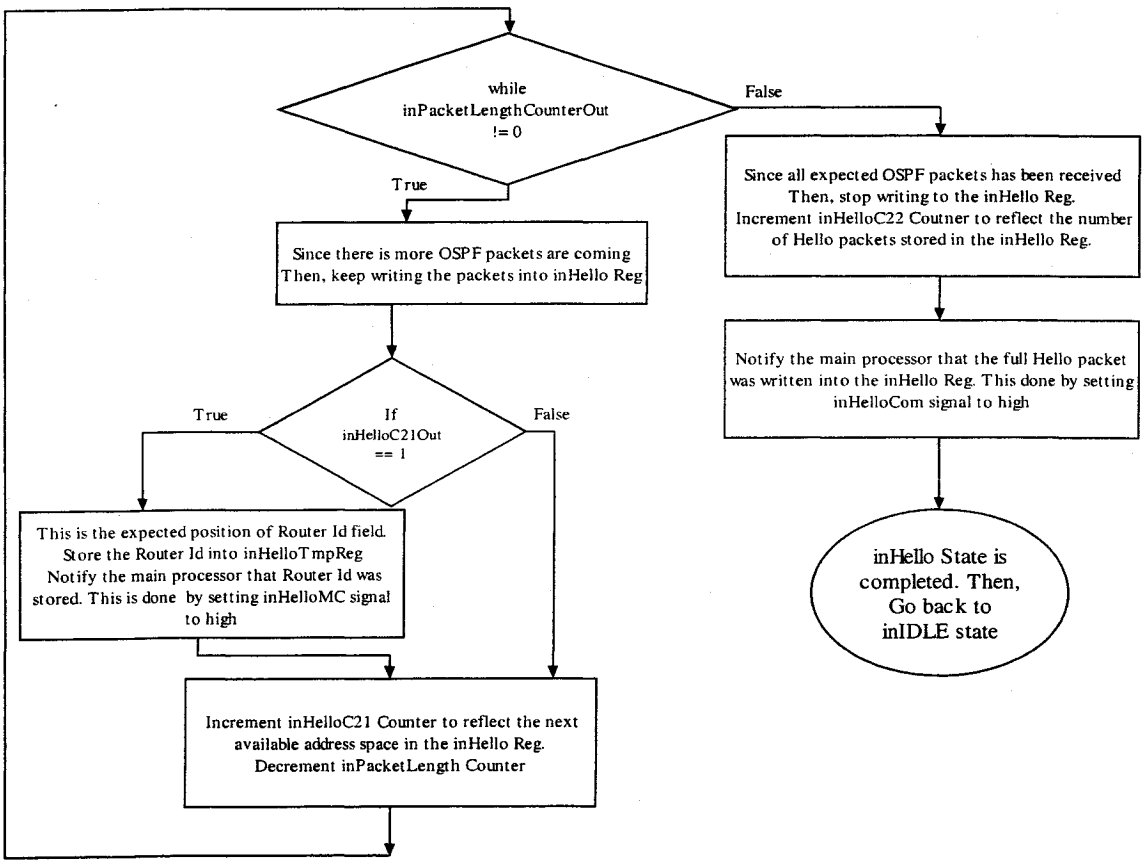


Figure 30 inHello high-level flow chart

All other states (inDd, inLsReq, inLsUpd and inLsAck) function the same way as illustrated in the Figure above. The only difference is each state has a different interest in their arriving packets. For example, the inHello state is interested in reading the router Id field because the

OSPF system needs to reply to that external router. In case of inDd state, the OSPF system is interested in the sequence number field because it needs to reply to the external router with the updated sequence number. In case of inLsReq state, the OSPF system requires the LS type, link state Id and advertising router fields. These three fields are required to allow the OSPF system to search for a matched LSA in the OSPF database. In case of inLsUpd state, the OSPF system does not require any specific fields. In case of inLsAck, OSPF is interested in the advertising router field because it acknowledges that router which ensures that the external does not re-send its packets.

The egress processor shown in Figure 31 is designed in the same manner as the ingress processor. The only difference is that instead of checking for incoming OSPF packet types, it receives commands from the main processor regarding the operation it needs to perform (mainly packet assembly to respond to arrived packets on the ingress path).

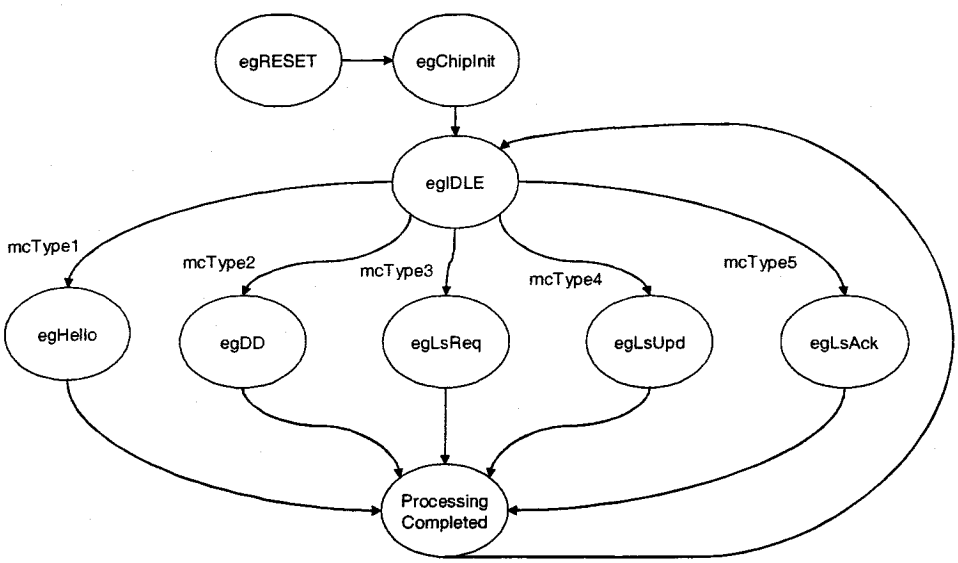


Figure 31 Egress processor state machine

The flow chart shown in the figure below demonstrates the type of control commands that are acting on the egress data path during the egHello state. The major goal of egHello state is to send out a reply Hello packet to the external router (occurs only when main processor invokes it). The first item on egHello state's list is to send out the OSPF header followed by Hello packet. Hence, the design has a dedicated egOspfhMC counter which keep tracks of

how far is the OSPF system from transmitting the OSPF header. Once, the OSPF header transmission has been completed. The design uses a egHelloMC counter to determine how far the OSPF system is from transmitting the Hello packet. The egHelloC21 counter is used to point to the next available address space in the egHello register (contains all external router Id(s) that are exchanging Hello packets with OSPF system). The egHelloC31 counter and egHelloC21 counter are used to determine if all router Id(s) have been placed on the egress path.

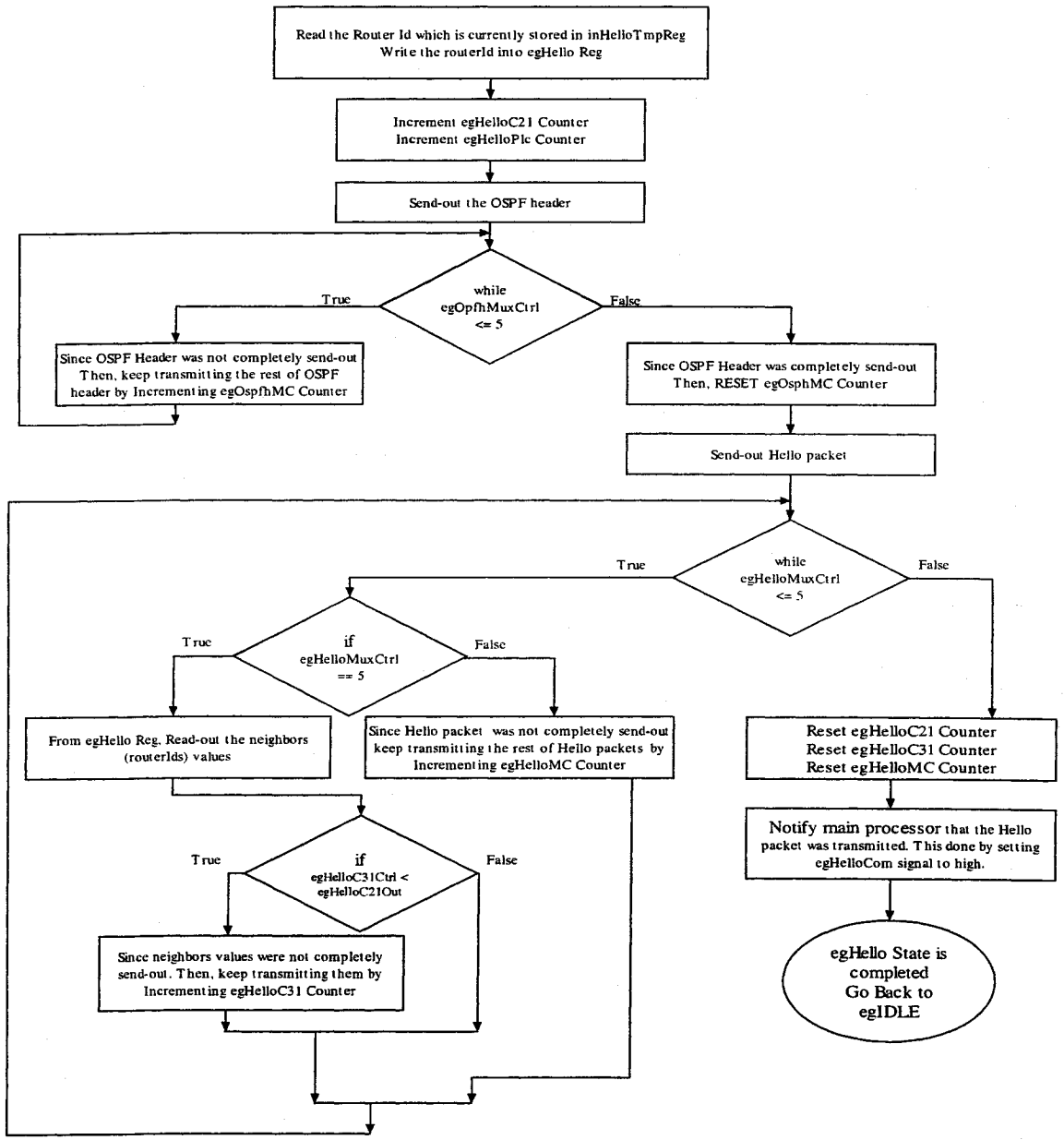


Figure 32 egHello high-level flow chart

All other states (egDd, egLsReq, egLsUpd and egLsAck) follow a somewhat similar infrastructure to the flow chart shown above.

4.4.3 LSA database (lsaDb) processor

The lasDb processor shown in Figure 33 is invoked during three main stages. The first stage is when ingress LS updates packets arrive. The processor moves from lsaDbIDLE state to lsaDbStoring state, where database swapping occurs. The database swapping basically activates lsaDb2 and deactivates lsaDb1. Hence, the ingress LS update packets are stored into lsaDb2. The processor remains in that state until all ingress LS updates have been written and waits for the main processor to set dbSynchGo signal to high. Upon setting dbSychGo signal the processor moves to synchDb2Db1 state where synchronization between the databases (lsaDb1 and lsaDb2) occur. The purpose of the database synchronization is to consolidate the LSAs into one database and to ensure that the active database has the latest LSAs. During synchDb2Db1, old LSAs are marked by inserting zeros in the location of the LSA header. Once the synchDb1Db2 state has been completed, the lsaDb processor moves to zeroCleanUp state where it removes the LSAs with the header set to zero (refer to Figure 34 for graphical view on the process handling the incoming LS updates). Moreover, the lsaDb processor then moves to synchDb1SPP state, where it sends updates to the SPP processor. These updates constitute the updated topology entries, newNode and newEntry control signals.

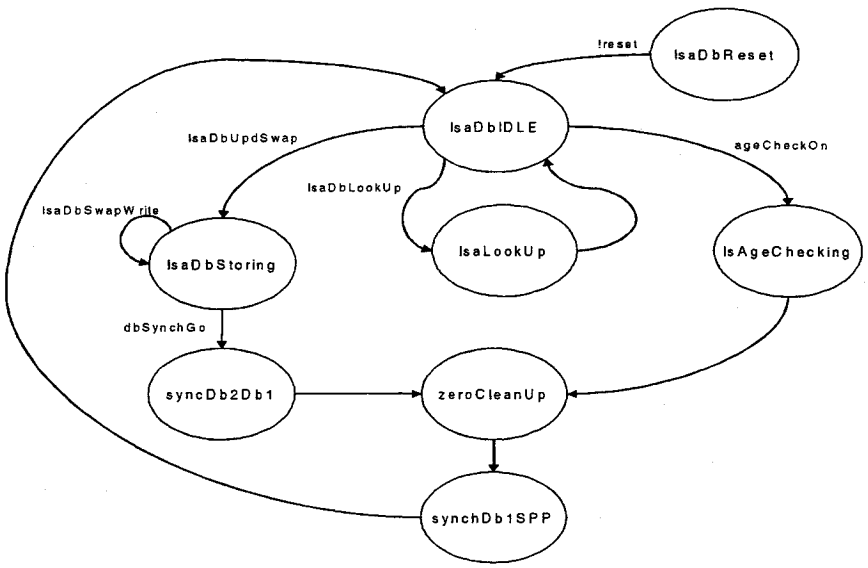


Figure 33 lsaDb processor state machine

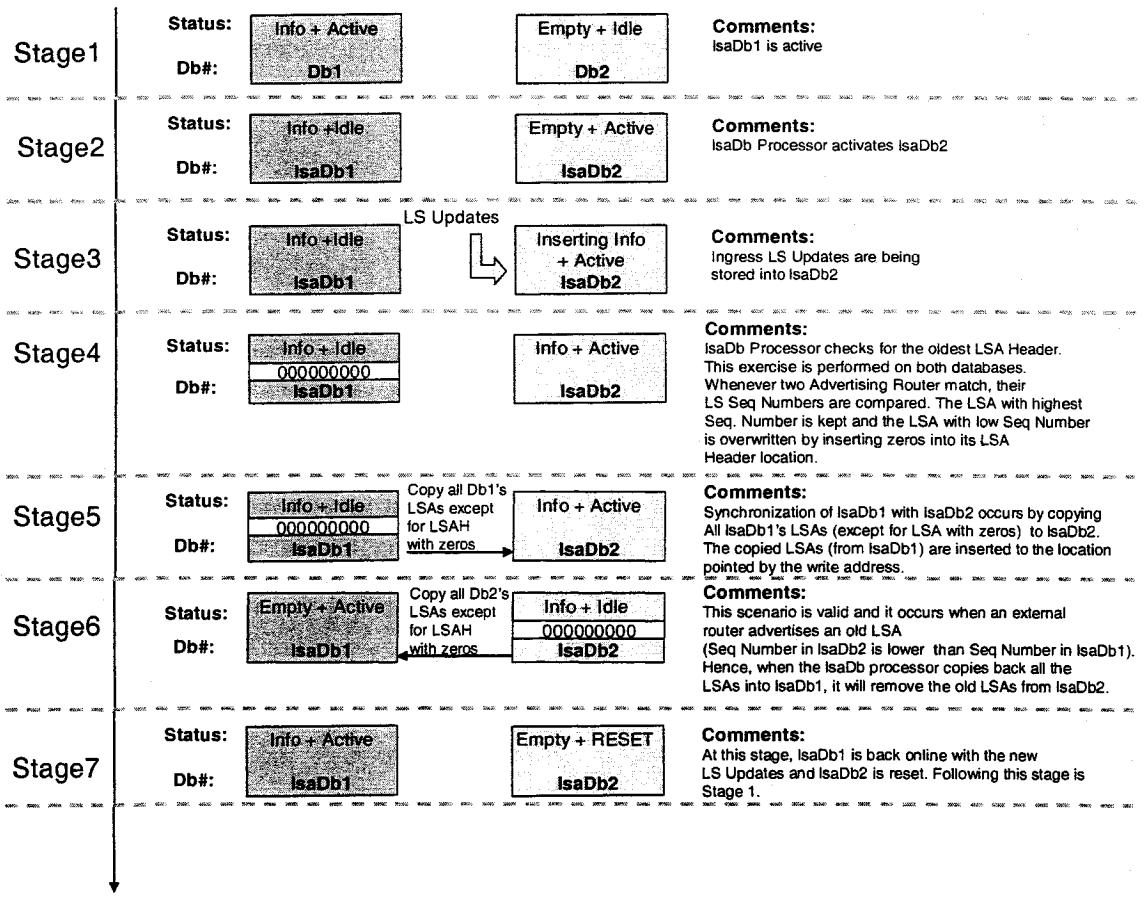


Figure 34 Database stages

The second stage where the lsaDb processor is invoked is at the arrival of ingress LS requests packets. In this case, the lsaDb processor moves from lsaDbIDLE to lsaLookup as shown in Figure 33, where the lsaDb1 entries are used to find a match for the LS request.

The third invocation is done by the timer processor. This invocation occurs (every half hour) when it is time for lsaDb processor to check the age of the LSAs stored in the lsaDb1 database. Hence, the lsaDb processor moves from lsaDbIDLE state to lsAgeChecking state. The results of completing lsAgeChecking state are two scenarios. The first scenario is if the LSA age was higher than 50 minutes then the LSA is overwritten by zero because it's out of date. The second scenario is if the LSA age was near 30 minutes then an LSA request packet is built and placed into the egLSReq register (these packets are then send out to the network by the egress processor). Upon terminating the lsAgeChecking state the lsaDb processor moves to zeroCleanUp state, where database synchronization occurs. The purpose of this database synchronization is to remove LSAs that were filled with zeros. Moreover, the lsaDb processor will move to SyncDb1SPP, where once again it updates the SPP processor with accurate topology entries.

The subsections below will provide further details to illustrate the control unit instructions given to the data path during synchDb2Db1, zeroCleanUp, synchDb1SPP, lsAgeChecking and lsaLookUp.

4.4.3.1 synchDb2Db1 state instructions

The synchDb2Db1 state is invoked right after all ingress LS update packets have been stored into lsaDb2. The flow chart below describes the operations performed by the control unit during synchDb2Db1 state.

Since the latest LS updates are written into lsaDb2. The control unit starts examining each newly stored LS update. This task is done by simply comparing whether the idDb2C31 counter output value has reached the idDb2C21 counter output value. The idDb2C31 can be thought of as the while loop index and the idDbC21 counter output (which is constant after all ingress LS updates have been stored) contains the total number of LS updates stored in the lsaDb2.

The lsaDb processor keeps reading and comparing advertising router fields from both databases (advertising router fields are stored in lsaDb1Out register and lsaDb2Out register as indicated by Figure 35). If advertising router fields match, then the lsaDb processor will only keep the LS update that has the highest sequence number (loaded and stored in lsaDb2OutRegExtra and lsaDb2OutRegExtra registers). The old LS update will be marked with zeros at its LSA header position.

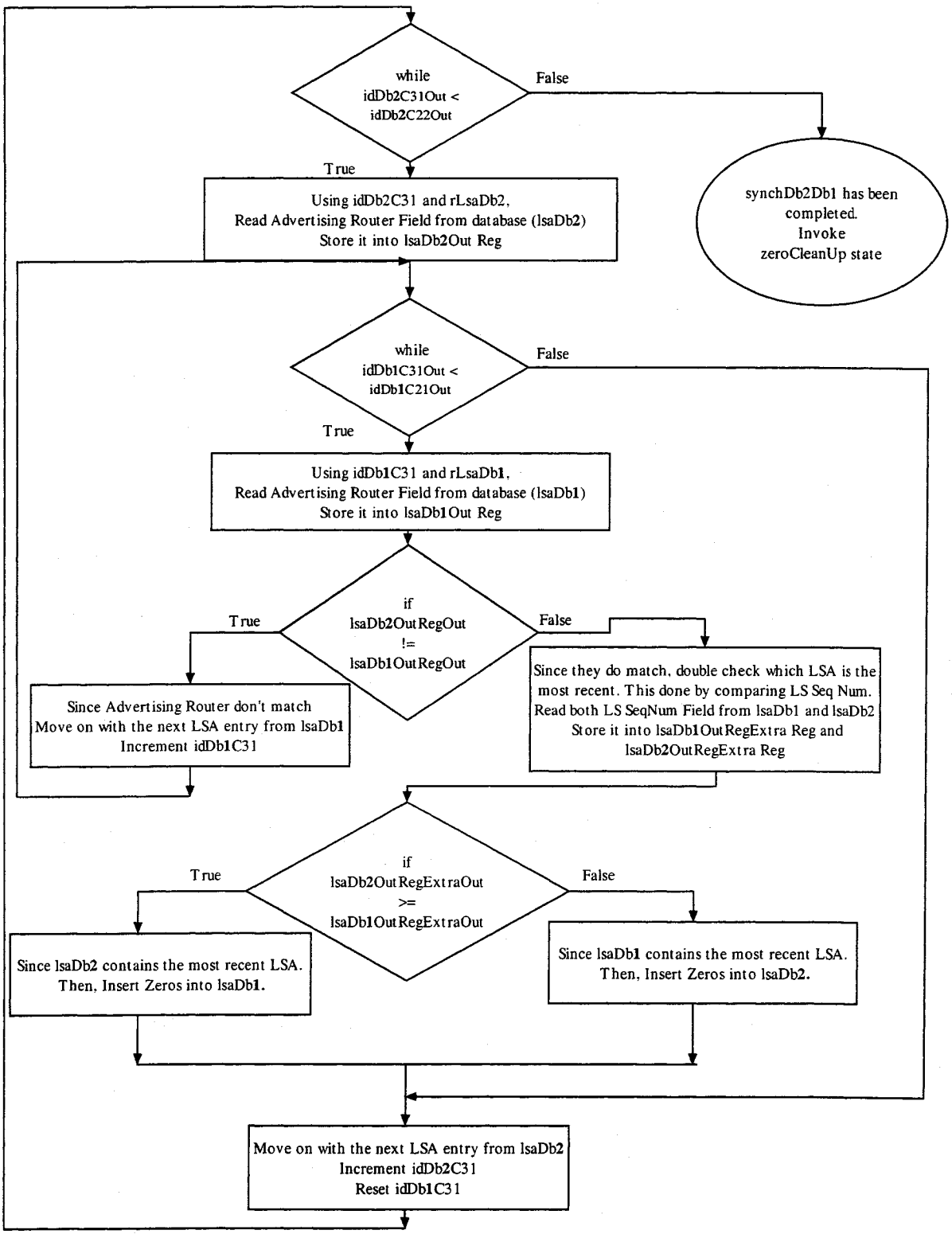


Figure 35 synchDb2Db1 flow chart

4.4.3.2 zeroCleanUp state instructions

The zeroCleanUp is invoked in two scenarios. The first scenario is right after synchDb2Db1 and the second scenario is right after lsAgeChecking. In either case, the lsaDb processor has inserted zeros to indicate that the LSA is no longer valid due to either out of date sequence number (the case if coming from synchDb2Db1 state) or LS age field is higher than 50 minutes (the case if coming from lsAgeChecking state).

The flow chart below describes the commands and conditions that the lsaDb processor go through to clean up the unwanted LSA. Again, the idDb1C31 counter output is used as a while loop index and the idDb1C21 counter output is the total number of LSAs stored. Hence, the lsaDb processor starts copying all the LSAs stored in lsaDb1 to lsaDb2 except for the LSAs with an LSA header marked with zeros. The next step is the reverse procedure, where lsaDb processor copies all the LSAs into stored in lsaDb2 into lsaDb1.

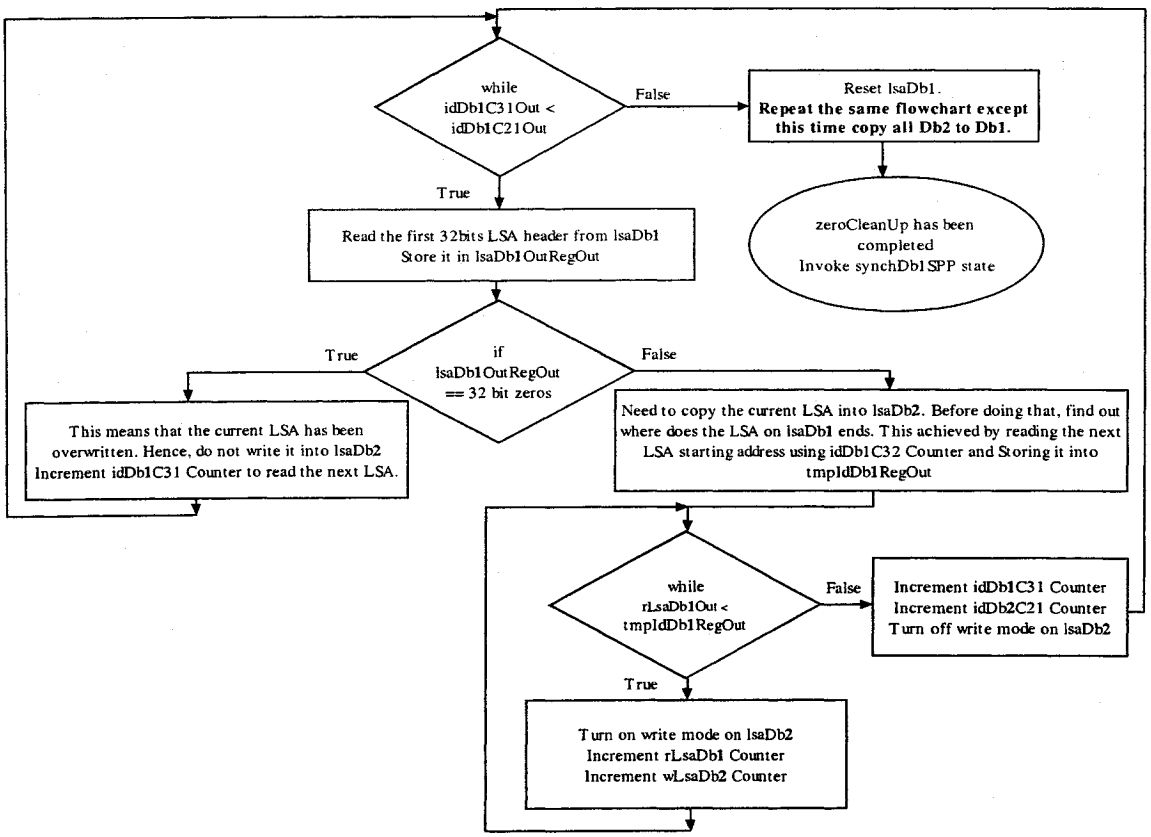


Figure 36 zeroCleanUp flow chart

4.4.3.3 synchDb1SPP state instructions

The synchDb1SPP state is invoked immediately after the zeroCleanUp state and the purpose of this state is to provide the SPP module with the correct topology entries.

The flow chart below outlines the lsaDb processor instructions given to the data path. The summary of the flow chart instructions is that the lsaDb processor needs to regulate the flow of newNode, newEntry control signals and topologyEntry data signal which are used to populate the SPP network topology.

For each LS update entry, the lsaDb processor will be setting newNode to high and for each router LSA within that LS Update (refer to section **Error! Reference source not found.**) the lsaDb processor will be setting newEntry to high. In parallel to the setting of newNode and newEntry signals, the lsaDb processor initializes the topologyEntry data signal. The topologyEntry data signal contains two information. The first information (Most Significant Bits) is the node index (i.e. Link ID) and the second information (Least Significant Bits) contains the link cost (i.e. Metric).

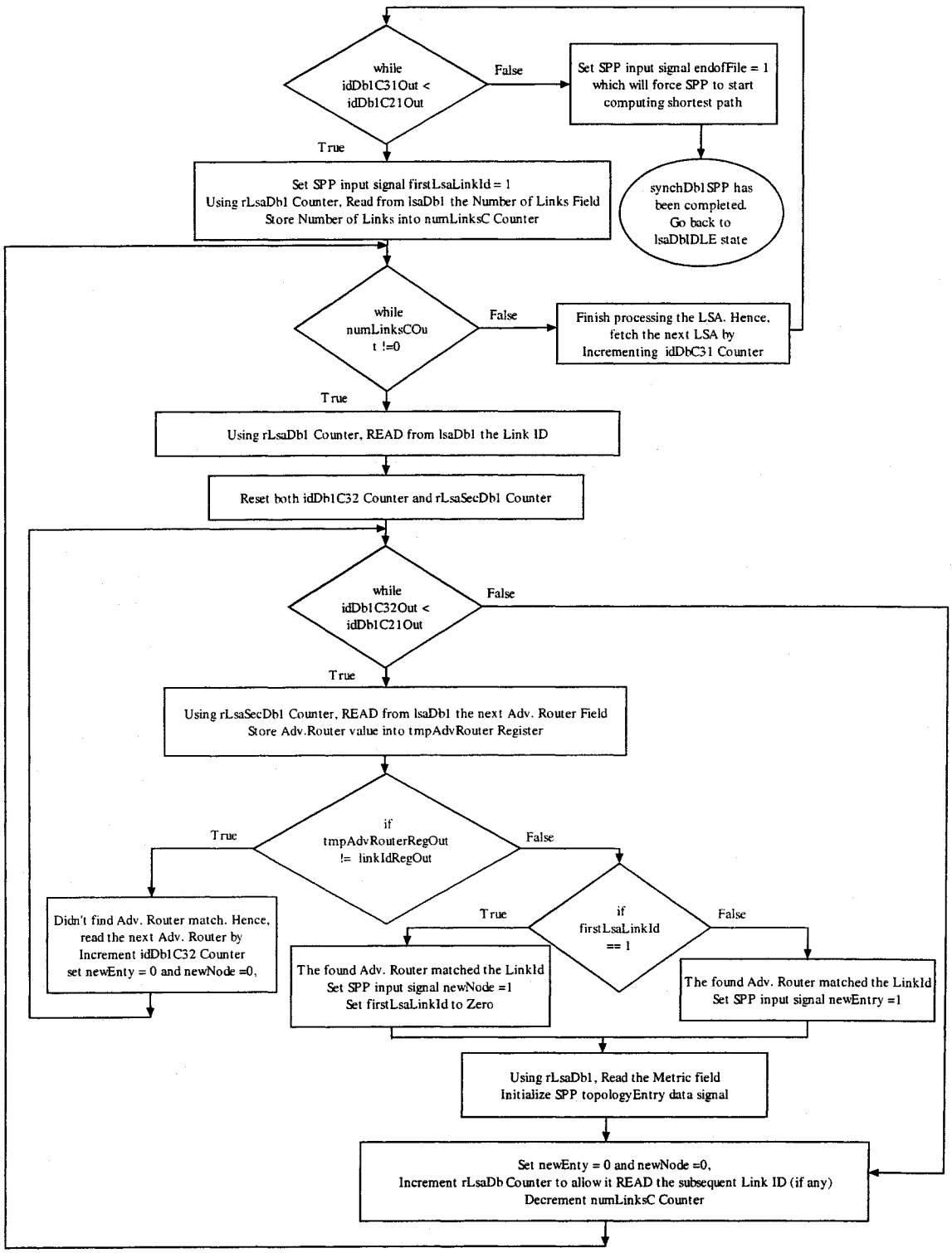


Figure 37 synchDb1SPP flow chart

4.4.3.4 lsAgeChecking state instructions

The lsAgeChecking state is called up by the timer process (running in the main processor). This state is invoked every half hour (1800 seconds) to check if any stored LS Age fields are above the limitation.

The flow chart below provides the lsaDb processor commands that are passed to the data path during lsAgeChecking state. The timeCounterOut is the output of the timer counter implemented within the OSPF system. The timerCounterOut along with the LS Age field are used by the ALU to compute the difference Z. Using Z, the lsaDb processor decides whether to trash the LSA or keep the LSA (as shown in the Figure below).

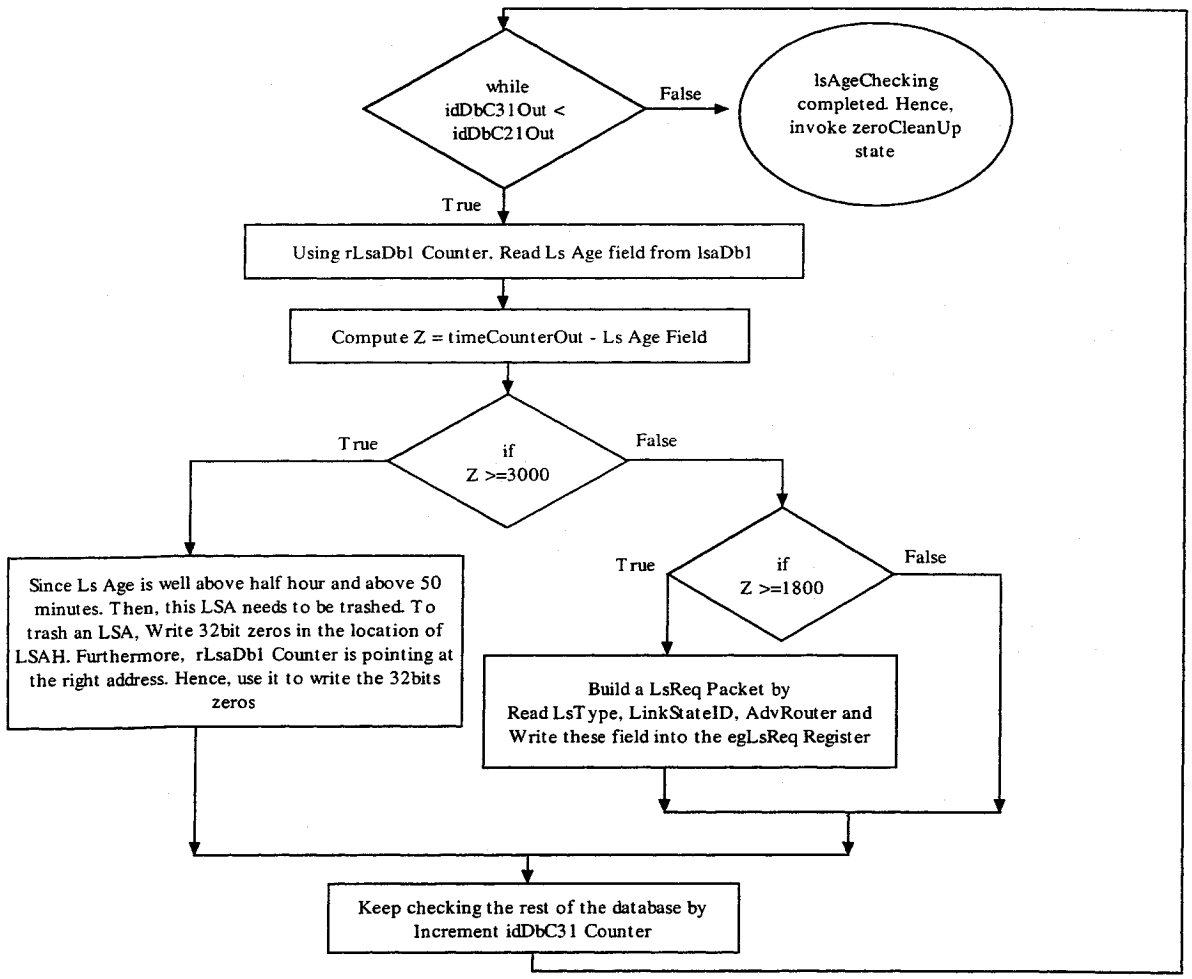


Figure 38 lsAgeChecking flow chart

4.4.3.5 lsaLookup state instructions

The lsaLookUp state is invoked at the appearance of LS request packets. The main goal of this state is to determine if it has any of the LS requests that are stored in the inLsReq register. If a match is found, then the lsaDb processor will populate the egLsUpdate register with the correct information.

The flow chart below provides a description of the operations performed by the lsaDb processor. The procedure used to determine if a match has occurred is done by comparing the stored (in lsaDb1) LSAs fields (Ls Type, Link ID, and Advertising Router) with the arrived LS request fields (inLsTypeRegOut, inLinkStateIdRegOut and inAdvRouterRegOut). This procedure (the look-up) is repeated from the beginning of the database all the way down to the end of the database or until a match entry has fulfilled the requirement.

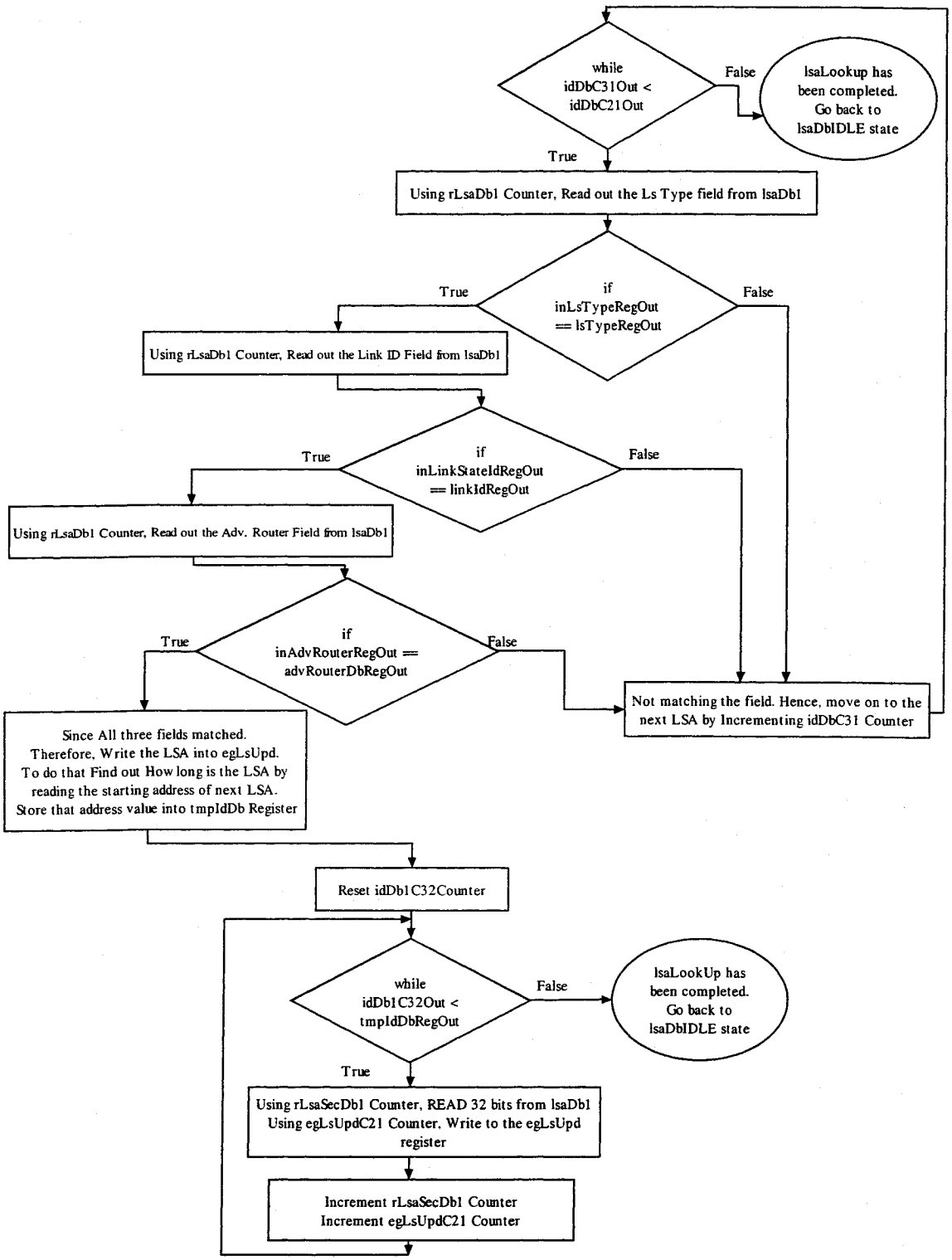


Figure 39 lsaLookUp flow chart

4.5 Data path design

The purpose of this section is to present more details on the data path components that were shown in Figure 23.

Figure 40 shows a very high level compressed view of the OSPF data path system. It consists of three major units and they are the ingress data path, egress data path and LSA databases (lsaDb1 and lsaDb2). The ingress components and egress components queues store packets according to their OSPF type. The OSPF data path has two LSA databases and only one can be active a time (refer to Figure 34). The systemInputs component represents registers that store the system inputs shown in Figure 22. The muxDemux component represents multiplexers and demultiplexers that were hidden to reduce the complexity of the diagram.

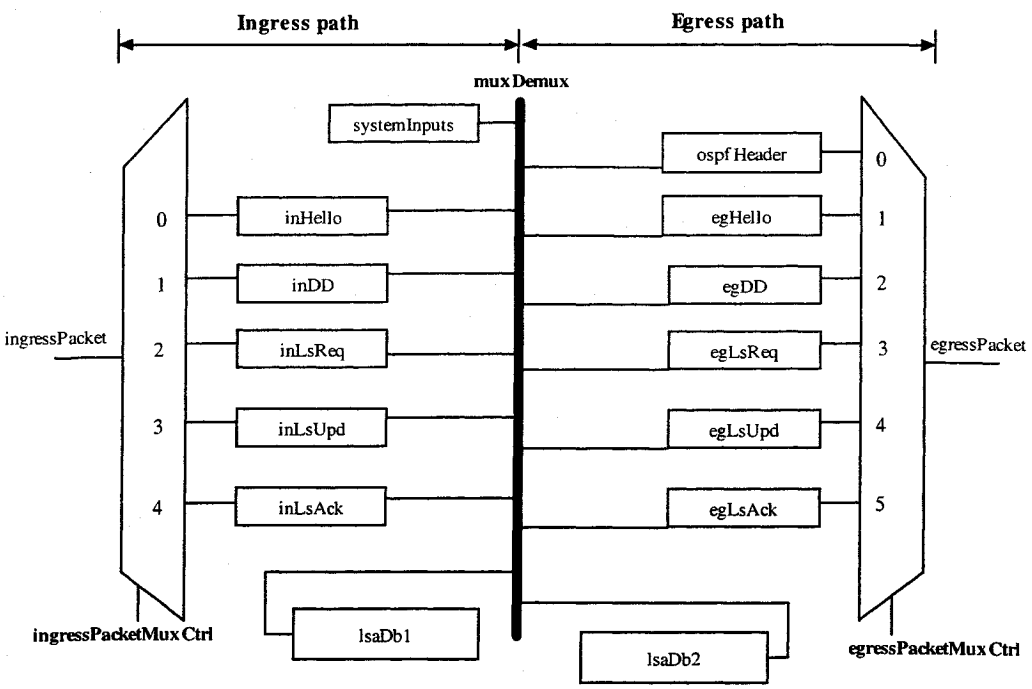


Figure 40 OSPF data path compressed view

In the subsections a more detailed view and explanation will be provided in the following manner; ingress path, egress path, LSA databases and the rest of data path elements.

4.5.1 Ingress data path

Ingress elements (signals, wires, buses and components) have the word "in" in front of them which denotes ingress. Figure 41 presents a design example of an ingress queue. In general, the middle block is the queue itself where the Hello packets get queued. The queue is 32 bits wide and its depth varies. The depth of a queue is determined by the number of nodes and links that need to be supported in an OSPF network. The muxs and demuxs are used to select the appropriate input and output data from the queue. Also, they are used to select the read and write addresses. Counters are used to track the read and write address locations. The counter size (number of bits) is dependent on the depth of the queue.

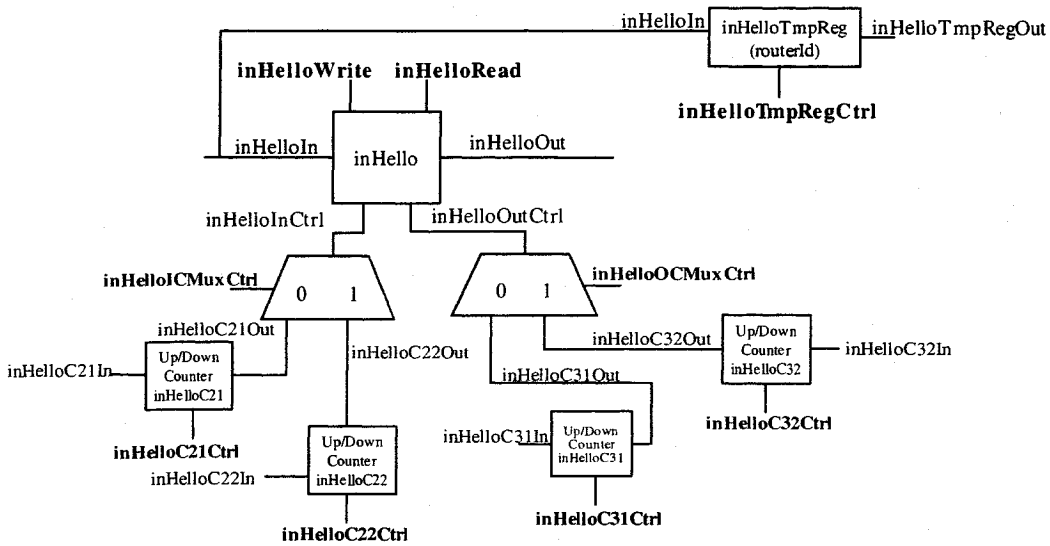


Figure 41 Example of ingress data path queue

To illustrate the specific functionality of the elements mentioned above, a detailed explanation of the inHello queue and its elements are provided below.

The inHello queue is used to store Hello packets arriving from ingress path. Hello packets are stored temporarily until the egress processor responds to other routers that are exchanging hello packets with the current router. From Figure 41, the inHelloICMux is a mux used to regulate the storage address. The address is read from the counters (inHelloC21

or inHelloC22). The inHelloC21 counter keeps track of the next free address space where packets can be written. The inHelloC22 counter used in scenarios where a specific field overwrites needs to occur. The same concept goes for inHelloC31 and inHelloC32, except they are there for reading purposes. The inHelloTmpReg register is used to immediately store and pass the advertising router Id to the egress processor. In this way, the egress processor can start working on assembling the Hello packet while the ingress processor is still writing the reset of the incoming hello packets. The full ingress data path is shown in the figures below. The usage and the functionality of these queues are more or less like the above mentioned example.

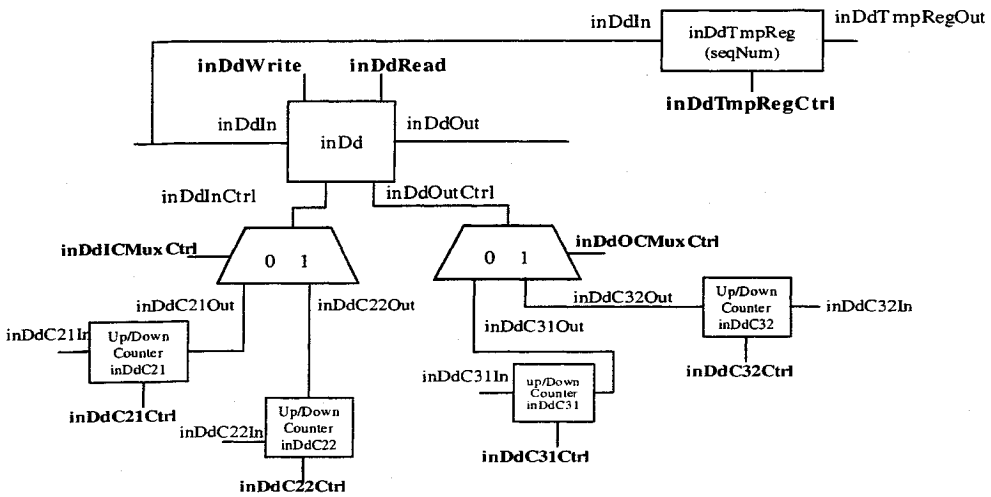


Figure 42 inDd queue

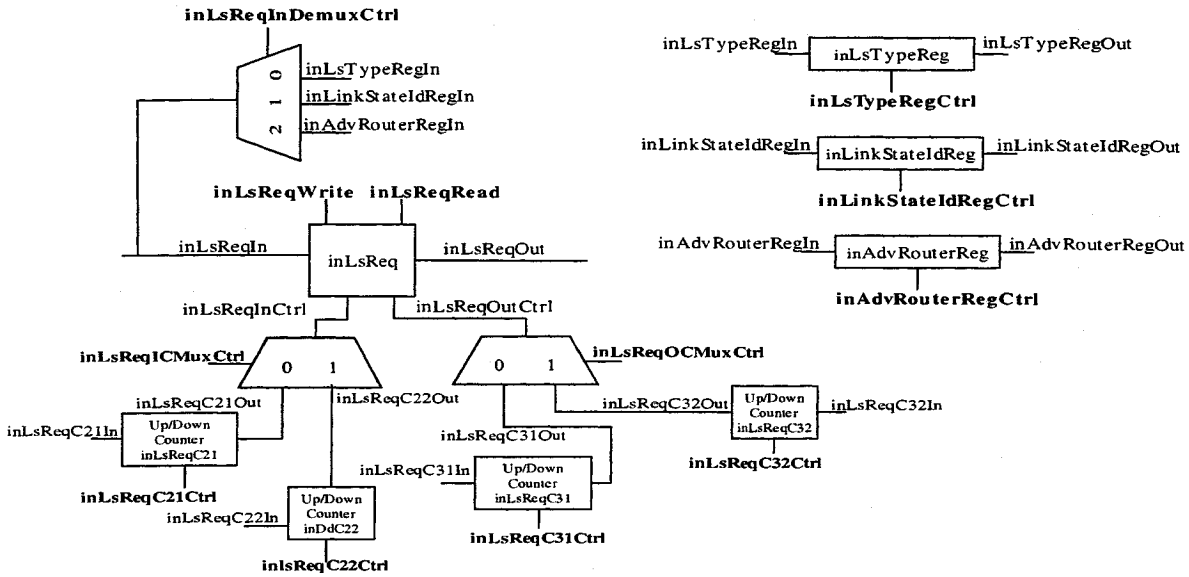


Figure 43 inLsReq queue

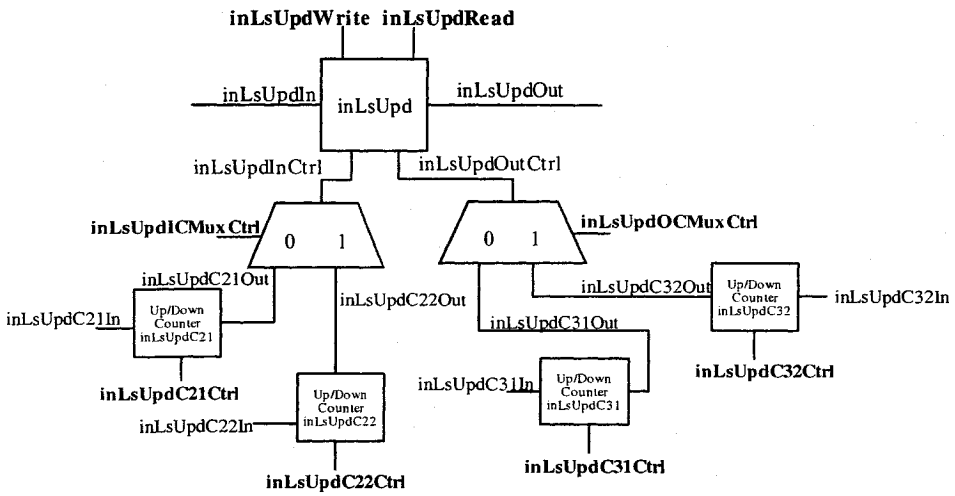


Figure 44 inLsUpd queue

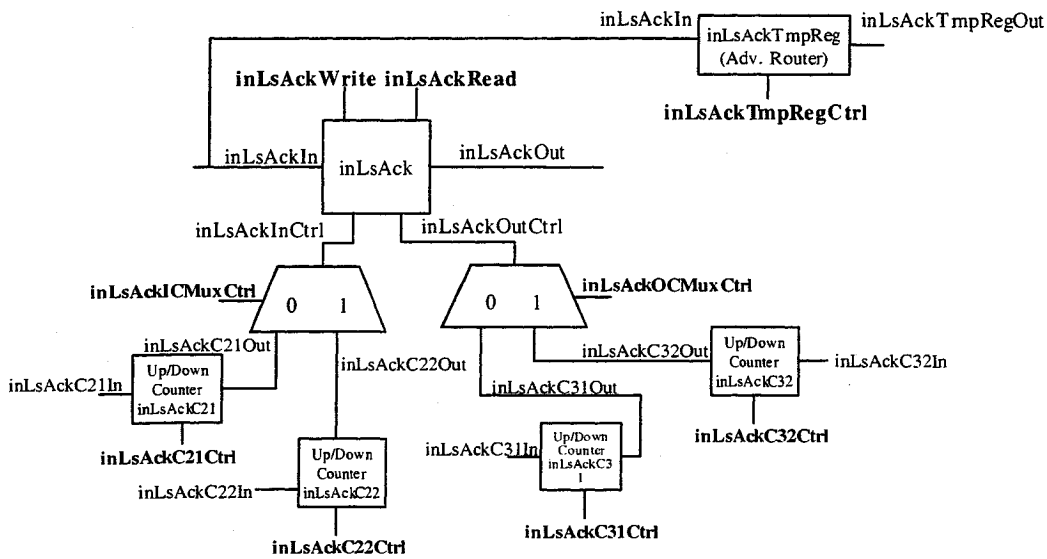


Figure 45 inLsAck queue

4.5.2 Egress data path

Figure 46 is zoomed in diagram on the attachments of the egress mux. Using this figure a description of how egress packets are transmitted will be given. For illustration purposes, the egress hello packet flow will be used.

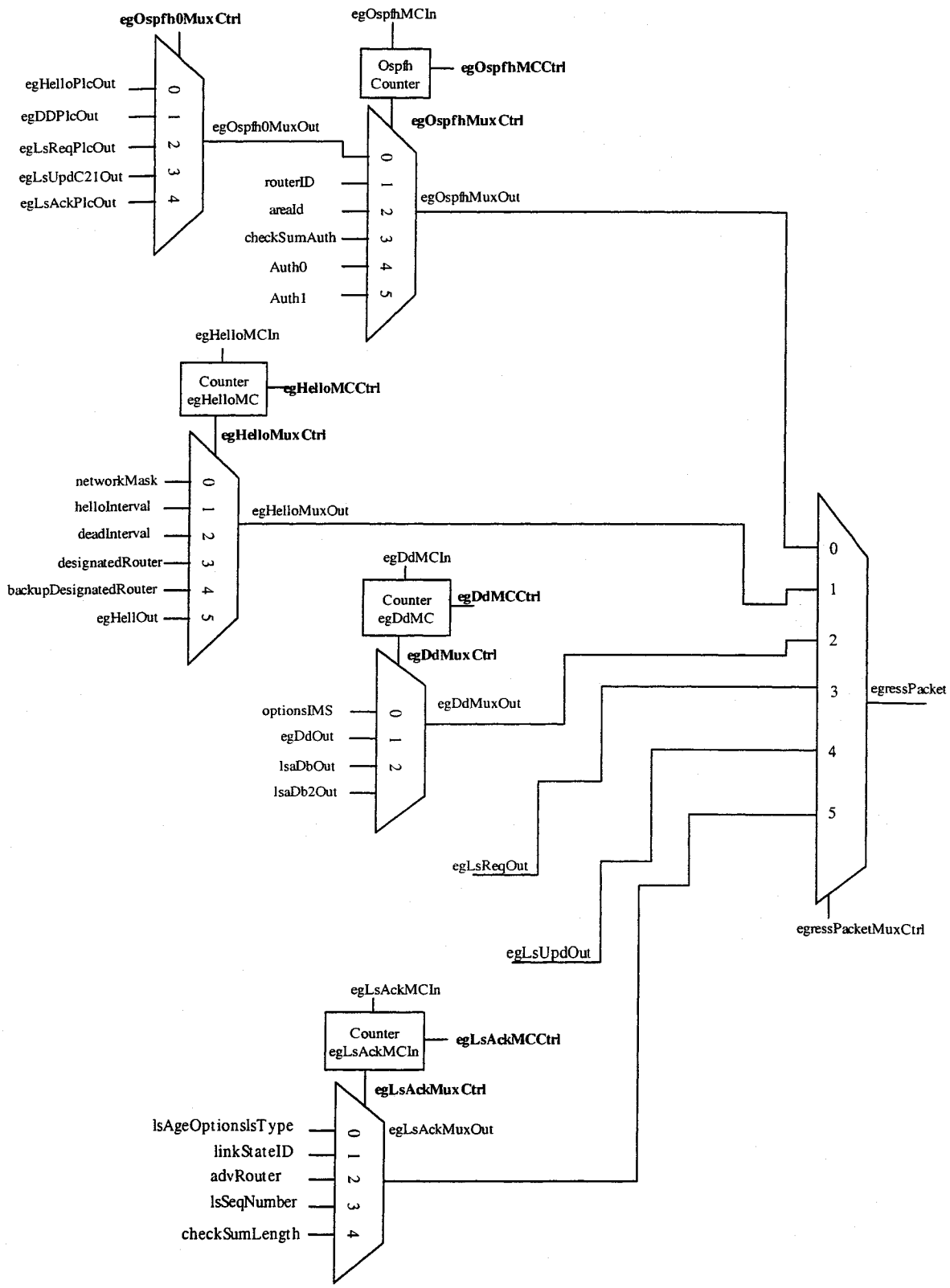


Figure 46 Egress mux attachments

Assuming that every field was initialized and it is time to transmit a Hello packet. The egress processor will first set egressPacketMuxCtrl to zero, egOspfhMuxCtrl to zero and egOspfh0MuxCtrl to zero. At this point, the first 32 bits that are put on the bus containing the Version, Type and Packet Length (refer to Figure 9). To keep outputting the rest of the OSPF header, the egress processor keeps incrementing the Ospfh counter until it reaches the value five. In this manner, the OSPF header has been transmitted. The next step is to send out the Hello portion of the packet. This task is accomplished by setting the egressPacketMuxCtrl to one and egHelloMuxCtrl to zero (basically reset egHelloMC counter). Moreover, the egress processor keeps incrementing the egHelloMC counter until it reaches the value five. Hence, the end result is an OSPF header followed by the Hello portion, which makes the Hello packet.

The figures below presents the egress queue designed to store and assemble egress packets. Once again, the egress queues are built the same way as the ones shown in the ingress path. Hence, for a description on the usage and functionality of the elements surrounding the queues refer to the ingress path section.

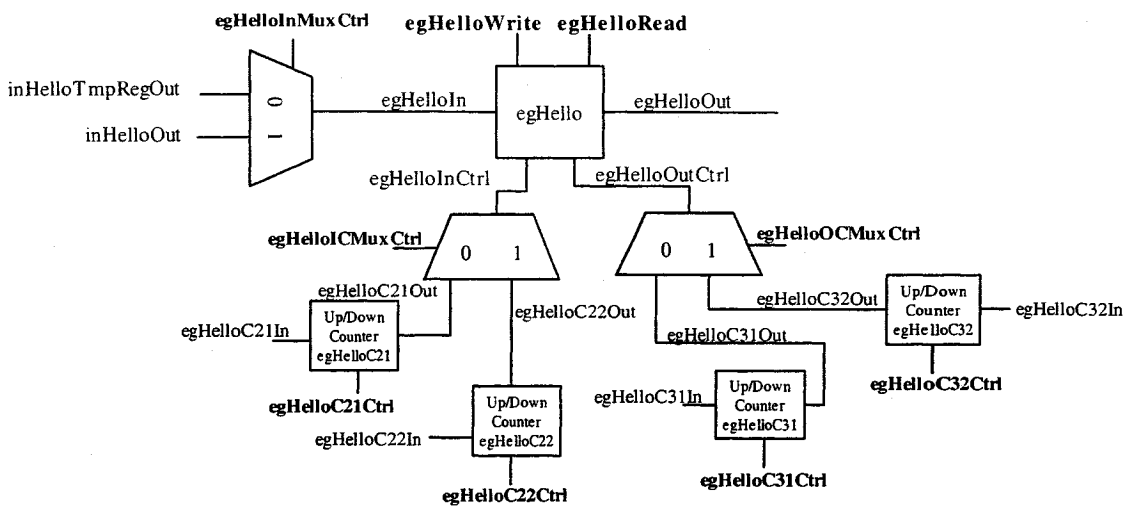


Figure 47 egHello queue

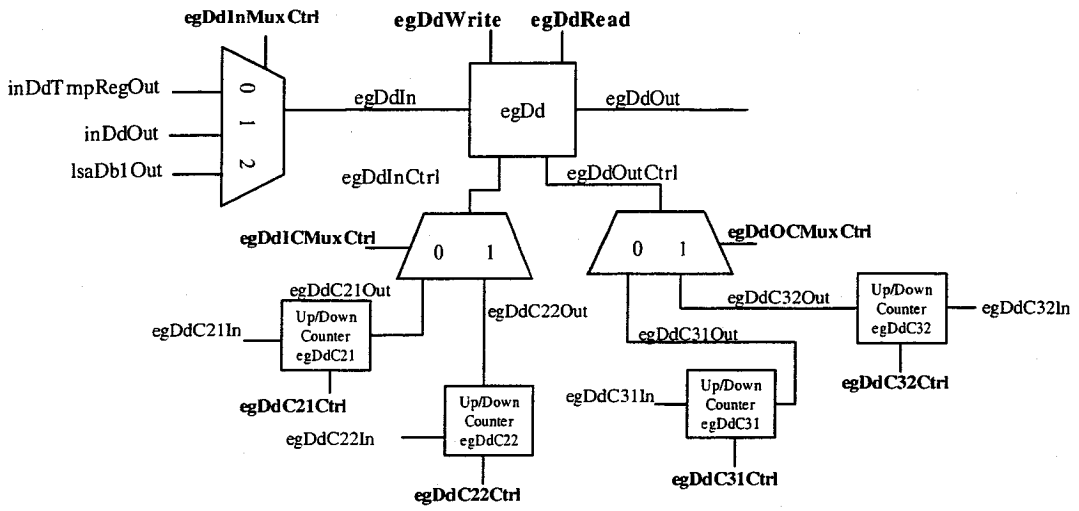


Figure 48 egDd queue

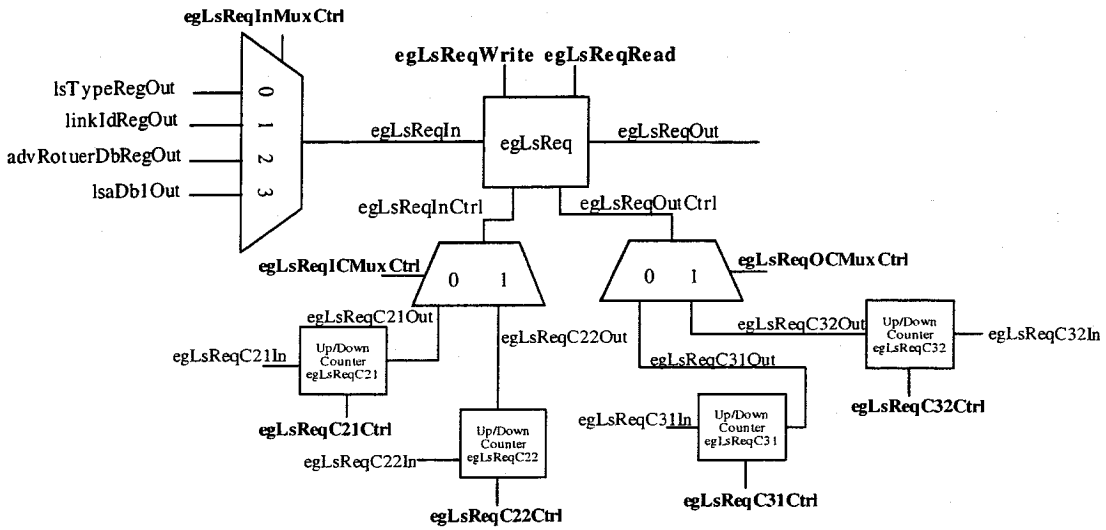


Figure 49 egLsReq queue

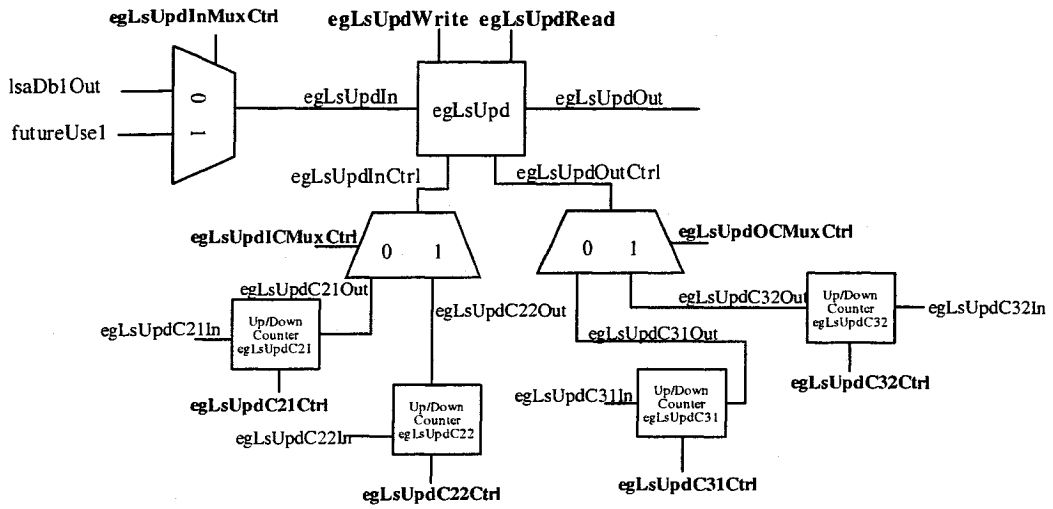


Figure 50 egLsUpd queue

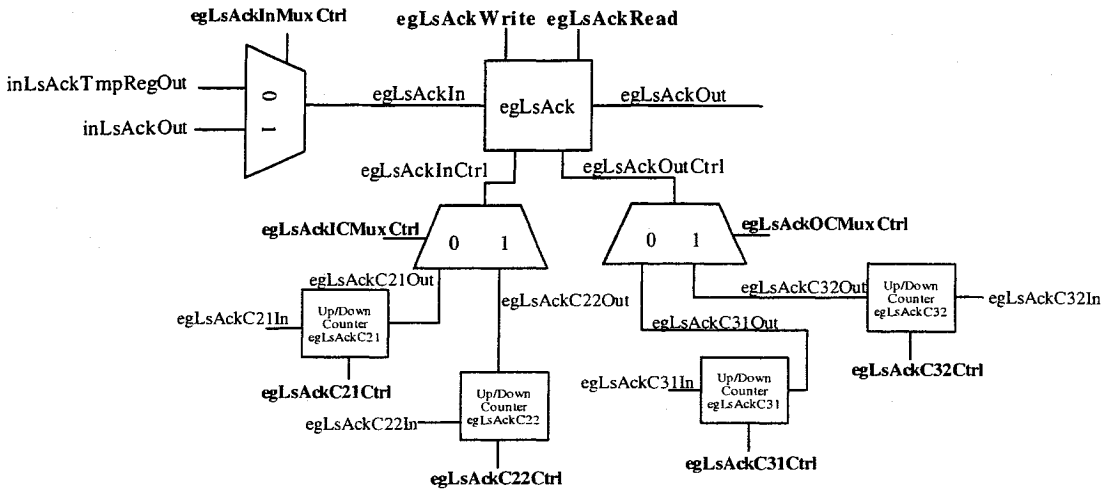


Figure 51 egLsAck queue

4.5.3 LSA database data path

The LSA Database (lsaDb) is where LSAs are stored. The sources of these LSAs are the DD packets and the LSA updates received from external router advertisements. There are two lsaDbs in the OSPF system and only one of them can be active at time. The active database contains the most recent processed LSAs. The standby database is only used during LSA

updates, where additional LSAs require storing or deletion of old LSAs needs to occur. Figure 52 and Figure 53 show the elements attached to the lsaDb1 and lsaDb2 respectively.

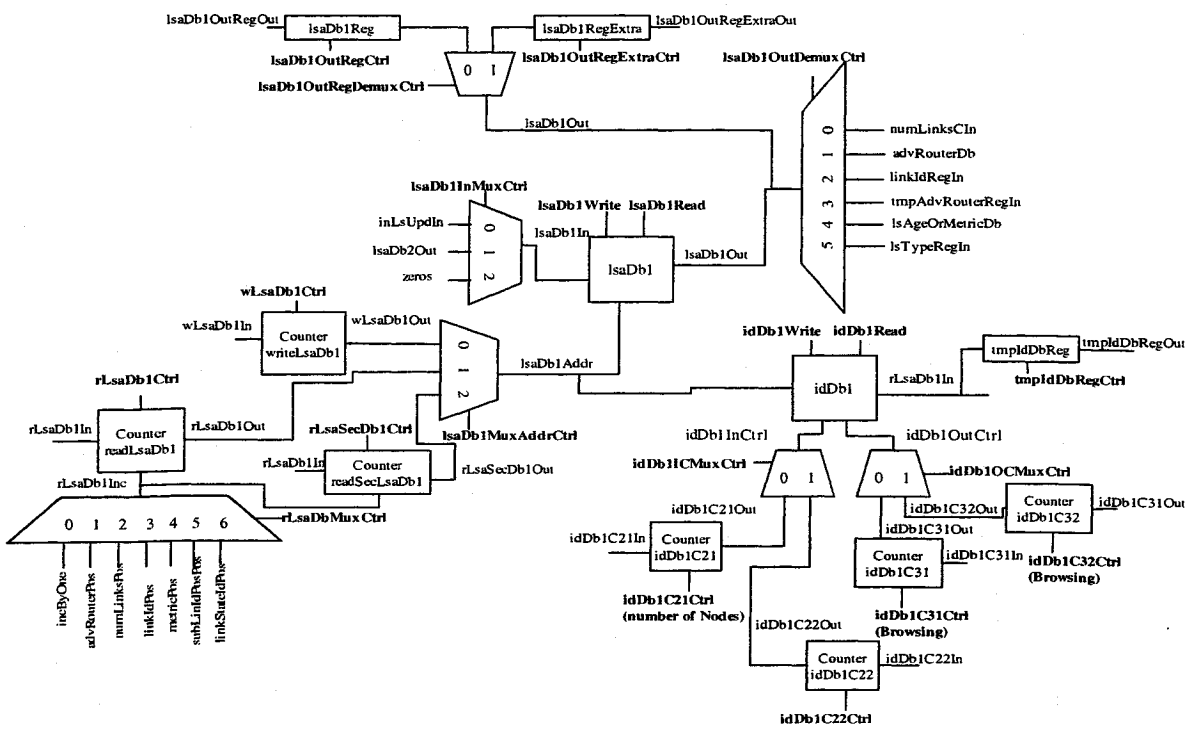


Figure 52 lsaDb1 data path surroundings

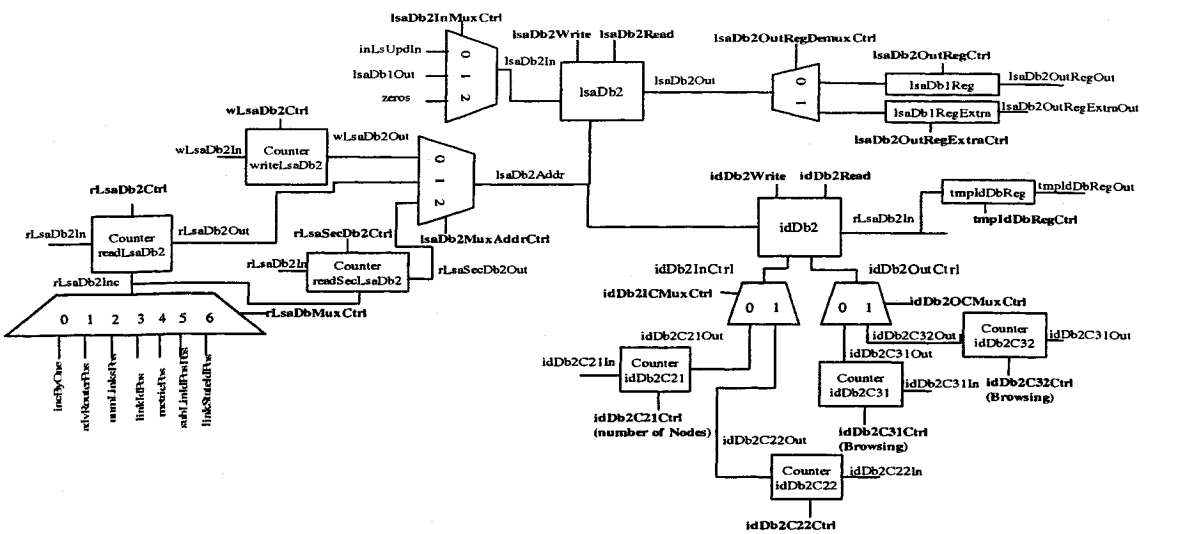


Figure 53 lsaDb2 data path surroundings

For each lsaDb, there is a writeLsaDb counter and a readLsaDb counter. These counters are used to track write and read addresses respectively. What's unique about the readLsaDb counter is that it can be incremented by different value. These values are selected by the demux right below it. In this way, any 32 bit field can be accessed very quickly. The readSecLsaDb is used in situations where an important read address needs to be saved for a specific operation. The idDb acts as an indirect memory address. It stores the starting address value of each LSA. In this way, LSA lookups and LSA age checking can be completed much faster. Once again, the counters (idDbC21 and idDbC22) are used to track address in the idDb queue. The tmpIdDbReg is used to temporarily store an LSA starting address for specific operations. The mux attached to the lsaDb is used to select specific inputs. For example, the inLsUpdIn wire is where the incoming LSA updates come through. The lsaDb (lsaDb1 or lsaDb2) wire is used during database synchronization after the LSA updates. The zeros wire is used during the synchDb2Db1 state and the lsAgeChecking state where 32 bits of zeros are required to mark unwanted LSAs.

4.5.4 The rest of the data path elements

The figure below shows five (One for each OSPF packet type) egress packet length counters. These counters compute the packet length fields as 32 bits segments are stored in the corresponding queue. For example, in the case of the egress Hello packet, the default packet length would be 11 segments (each segment is a 32 bits). The 11 comes from; 6 for the OSPF header and 5 for the Hello header (refer to section 3.3 for further details). Now during a Hello packet exchange the packet length may varies. Hence, the egHelloPlc counter assists in computing the exact length as more neighbors (router Ids) are stored in the egHello queue.

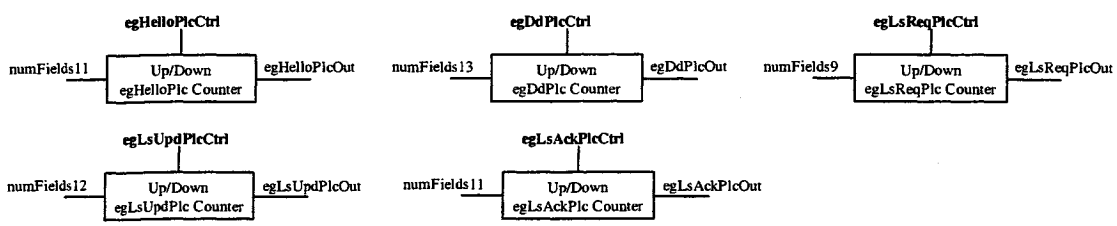


Figure 54 egress packet length counters

ALU surroundings are shown in the figure below. The primary user of the ALU is the lsaDb processor (during lsAgeChecking state). The ALU outputs are "xGy"which denotes X is greater than Y; "xEy" which denotes X is equal to Y; "Z" which contains the result of addition or subtraction.

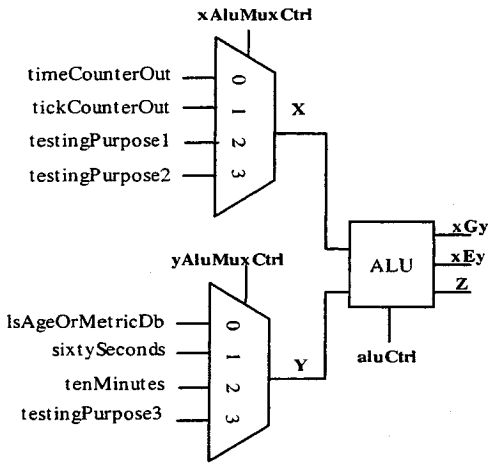


Figure 55 ALU in OSPF data path

The egQ shown in the figure below is used to store the main processor commands for the egress processor. This main processor commands storage occurs when the egress queue is busy sending or assembling packets.

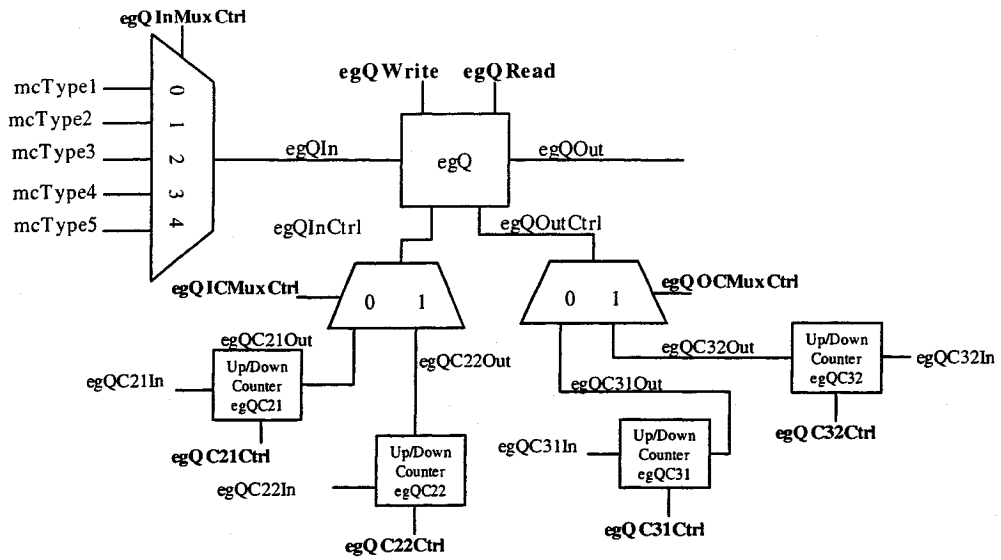


Figure 56 egQ queue

The usage of the counters and registers below were already discussed in the control unit section. They are presented here for the sake of the reader's reference.

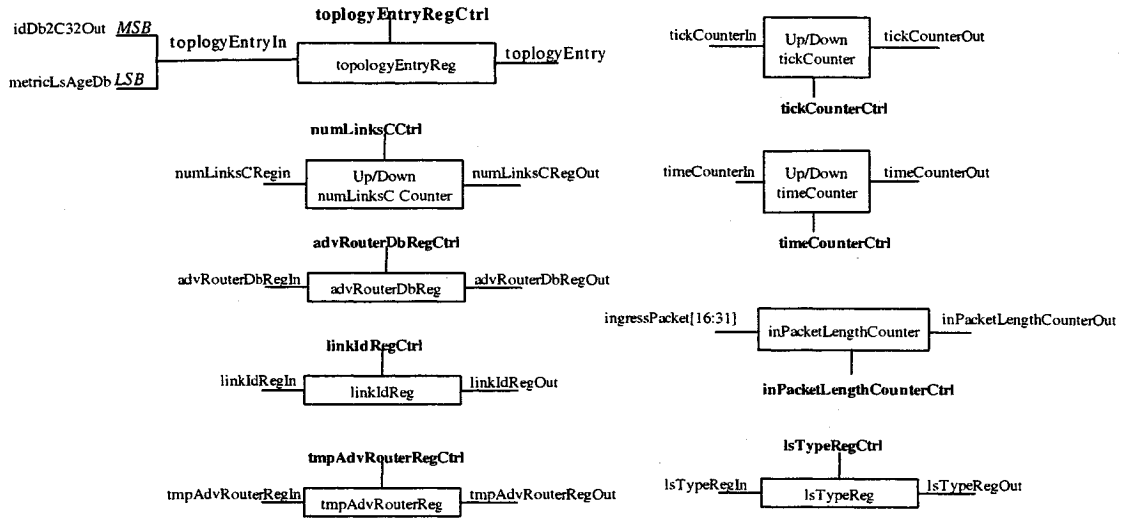


Figure 57 Other counters and registers

5 Shortest path processor design

In this chapter, the focus is to discuss SPP design only. Hence, for details on OSPF system design refer to Chapter 4 and for an overview between the connectivity between OSPF system and SPP modules refer to section 4.2.

The first section of this chapter presents the inputs and outputs of the SPP module. The second section provides insights on the control unit design. Finally, the last section explains the data path design.

5.1 SPP inputs and outputs

Figure 58 presents the input and output signals of the shortest path processor (SPP). The data inputs signals are "topologyEntry", "sourceNode" and "destNode". The input control signals are "initialize", "newNode", "newEntry", "endOfFile", "clk" and "reset". The SPP accepts topology entries through the "topologyEntry" signal when "initialize" is set to one. To inform the SPP that an upcoming entry is for a new node the user must switch "newNode" from zero to one. To notify the SPP that a new link is available, the user should switch "newEntry" from zero to one. To indicate that topology entries are completed, the user will have to set "endOfFile" to one, which in return causes the processor to start computing the shortest path for the source node, specified by the "sourceNode" signal. At any point in time, if the "reset" signal is set to zero, then the chip will clear all its topology contents.

For outputs, the processor will provide a shortest path through the "shortestPathFile" data signal. The "pathAvailable" signal is set to one to indicate that the shortest path computation has been completed.

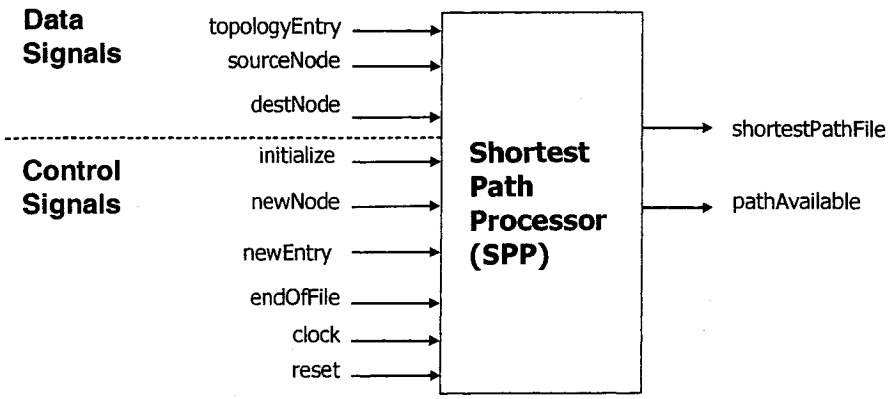


Figure 58 Inputs and outputs of SPP

5.2 Control unit design

5.2.1 Overview of control unit state machine

The state machine shown in Figure 59 describes the control unit. As an initial state the processor is in state1 where it is idle meaning that it has already computed the path or it is waiting for an initialize signal to be set high. When "initialize" is set to one, the processor will move from state1 to state2, where it's expected that the user will start providing network topology entries. The user can indicate that topology entries have completed by setting "endofFile" signal to one, this causes the processor to move to state 3. In state 3, the processor will search for the node with least distance label and as soon as that node is selected "minFound" is set high which causes the processor to move to state4. At this stage, the processor goes through the arc list of the node selected and updates the distance label register (DR). Upon the completion of distance label updates it sets "arcDone " to one, which in return goes back to state 3 for another node selection. This exchange of states between state 3 and state 4 continues until all the nodes of the network has been processed. As a result, the processor sets "nodeDone" signal to one, which forces the processor back to state1. Figure 59 shows that at any point of time if the reset signal is set high the control unit

clears all its memory content. The following subsections provide detailed descriptions for each of the four states of the control unit.

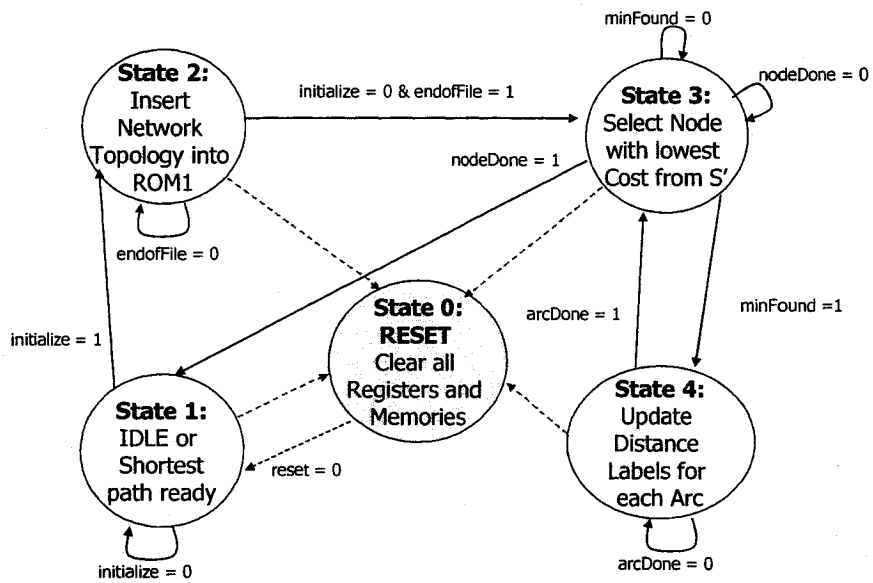


Figure 59 Control unit state machine

5.2.2 State1 and State2: Network topology and initialization

Figure 60 illustrates the state machine represented by state2 in Figure 59. The SPP only enters state2 when "initialize =1" (set by the user) and as a result the SPP clears all registers, counters and memory contents, otherwise the SPP will remain in its IDLE state. The purpose of state2 is to allow the SPP to store and update its database with information regarding the network topology. State2 will also permit the proper initialization of registers such as distance label register (DR).

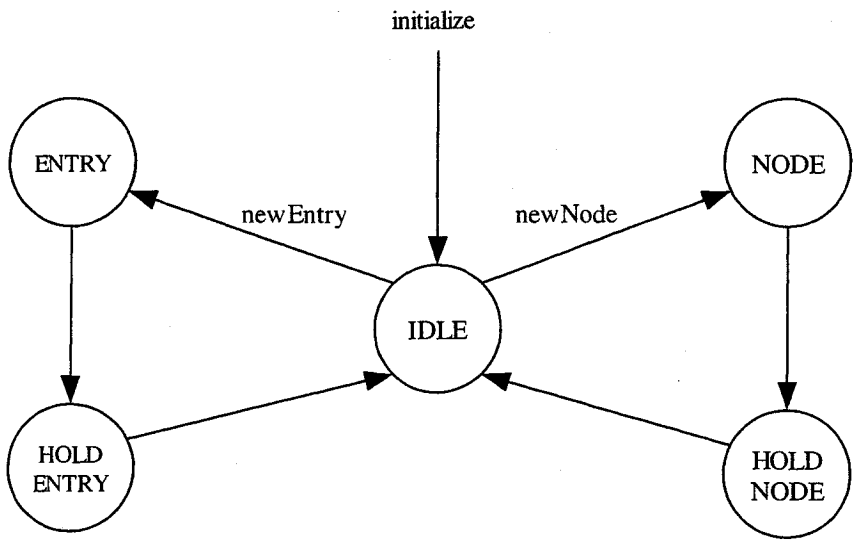


Figure 60 State1 and State2 flow diagram

Upon entering state2, the source node is stored into "iReg". Then the SPP will enable data transfers to static memory, DR, AR and PR. The static memory will be enabled to start storing topology entries as they come. The DR will be enabled to initialize all distance labels to infinity. AR is enabled to store the number of arcs per node. PR is enabled to initialize the source node predecessor to zero as shown in Figure 3 statement 5.

The SPP keeps accepting topology entries through the signal "topologyEntry" until "endofFile" is set to one. If "endofFile" is set to one, the SPP moves to state3. To manage the storage of topology entries, the SPP offers signals "newNode" and "newEntry". The "newNode" is used to indicate that the current "toplogyEntry" is a start of new node's arc list. The "newEntry" signal is used to inform the SPP that the current toplology entry belongs to the same node identfier that was set recently

If "newNode = 1" the SPP transitions to the "NODE" state where it increments the static memory address, DR index, AR index and SPR index. The static memory address is incremented to point to the next location available to store the next topology entry. The DR index is incremented to initialize the next node distance label to infinity. The AR index is incremented to store the number of arcs of the new node. The SPR index is incremented to store the start of static memory address of the new node arc list. A transition to the

"HOLDNODE" state occurs so read and write operations occur separate from address updates. When the topology entry for a new node is complete the SPP becomes idle.

If "newEntry =1", the SPP transitions to the "ENTRY" state where the "arC11" counter and the static memory address are incremented. "arC11" is incremented to keep track of the number of arcs connected to the current node. The static memory address is incremented to point to the next location available to store the next topology entry. A transition to the "HOLDENTRY" state occurs so read and write operations occur separate from address updates. When the topology entry for a new node is complete the SPP becomes idle. The entire process is repeated until "endofFile = 1" and as result forces SPP to transition to state3.

5.2.3 State3: Node selection

Figure 61 demonstrates the operation of state3. Once the SPP has entered state3 successfully, all the enabled registers are turned off. Furthermore, the SPP will execute statements 4 and 5 from Figure 3. The SPP uses "drC31" counter to navigate through the DR during processing.

To keep track of the number of nodes processed, SPP compares "arInCtrl" and "srInCtrl" which translates to statement 6 in Figure 3. "arInCtrl" is equivalent to "n" and "srInCtrl" is equivalent to "|S|" in Figure 3. It's important to note that "arInCtrl" is connected to the output of "arC21" counter and "srInCtrl" is connected to the output of "srC21" counter (which are shown in Figure 70 and Figure 71 respectively). In state2, the SPP kept updating "arC21" to reflect the number of nodes in a network and as will be explained later, "srC21" will be updated whenever a complete node arc list has been processed.

The SPP sets the "nodeDone" signal to 1 when the whole network topology has been processed to indicate that the shortest path is available in the PR register (i.e. SPP will go to IDLE state). Otherwise, SPP sets "nodeDone=0" for further processing on the network topology.

As mentioned earlier, the SPP uses "drC31" counter to navigate through DR. At the start of state3, the "drC31" counter was reset to zero to make sure that SPP will start processing the arc list of the source node. For this reason, the state machine in Figure 61 checks whether "drC31out" is equal to zero. If "drC31out" is not equal to zero, this means that SPP has already processed the "sourceNode" and it's time to look at other nodes arc list in the network. The node selection criterion is computed in the state "SELECTNONSRC', which will be discussed later. After the node selection is completed, SPP sets "minFound" to one and as a result invokes state4, or the "WAITFORUPDATE" state where the node arc list will be processed. Once the node's arc list is processed, the SPP invokes state3 by setting "arcDone" equal to one. Moreover, the SPP increments the "srC31" counter to reflect that number of nodes processed so far.

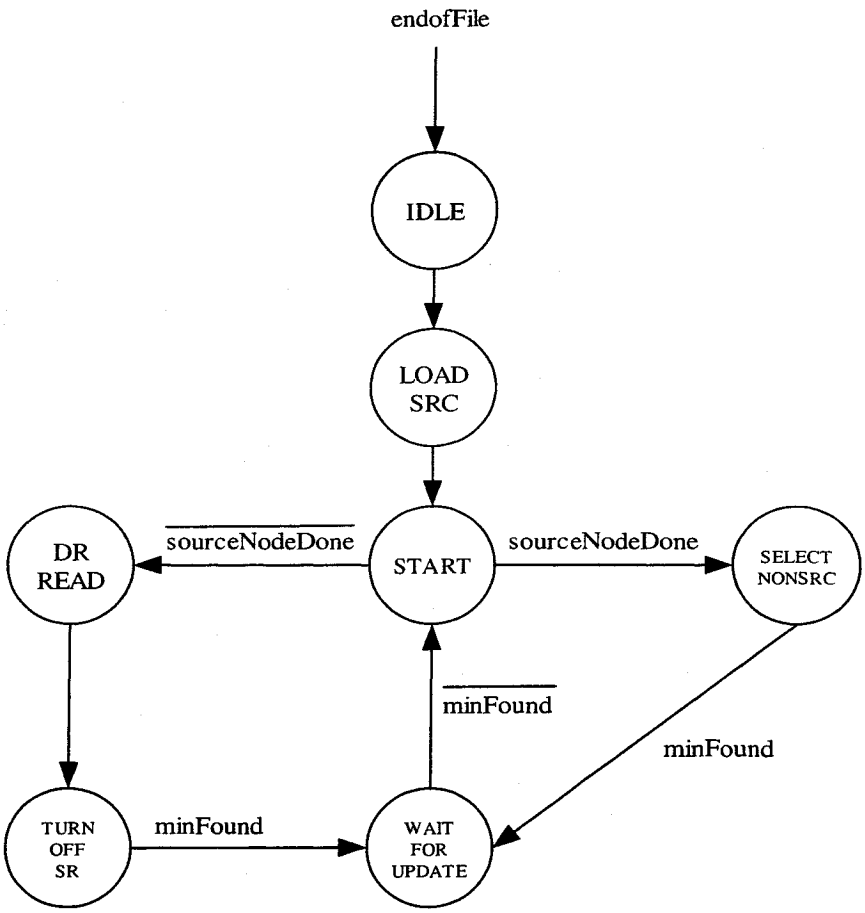


Figure 61 State3 flow diagram

As indicated earlier the "SELECTNONSRC" state performs the node selection. A node is selected based on two conditions; First, a node can only be selected once and second if selected this node has to have the lowest distance label. The two conditions mentioned above maps to statement 7 in Figure 3.

Figure 62 illustrates how the node selection process will be done in hardware. SPP starts by resetting "drC31" and "srC32" counters. "drC31" and "srC32" are used to navigate sequentially through DR and SR registers respectively (see Figure 69 and Figure 71). The SPP checks if "drC31out" has reached the total number of nodes in the network. This task is done to ensure that the SPP has gone through the DR register and picked the node with the lowest distance label cost. The next step is to confirm that the node chosen, node(i), does not appear in the SR. This step achieved by going through the SR and comparing each node id with node(i). If the node has not been previously chosen, then the SPP will increment "drC31" to pick a new node (node(i)). If the node has been chosen, the SPP still has to check with the rest of the SR register. To complete the verification of the SR, the SPP makes sure that "srC32Out" has reached the number of nodes inserted into the SR register ("srInCtrl"). Once the SPP has finished SR verification, it compares the current distance label with the distance label stored in "minReg" (this distance label belongs to the node identifier stored in "iReg"). If the distance label of the currently selected node is less than the distance label stored in "minReg", the SPP will update both "minReg" and "iReg". The "iReg" will contain the updated node(i) and "minReg" will conatin the distance label of node(i).

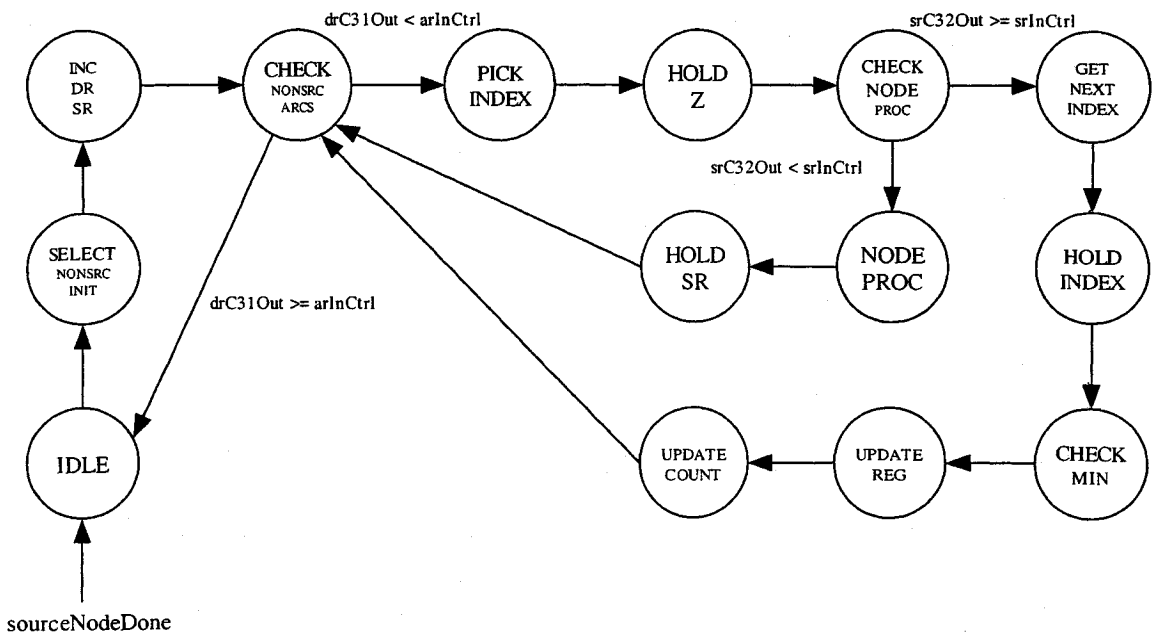


Figure 62 SELECTNONSRC state flow diagram

5.2.4 State4: Arc list processing

State4 is where the SPP focuses on processing the arc list for the node selected (node(i)) in state3. The state machine for state4 is given in Figure 63.In order to start processing an arc list of a node, the SPP needs to know the number of arcs of node(i). Therefore, the SPP will access this information from the AR register and load it in the "arC41" counter.

The "arC41" counter is used to keep track of the number of arcs processed and it is decremented every time the SPP processes an arc. To fetch the actual arc list, the SPP will refer to the SPR register to see where the pointer to the address memory of arc list is stored. If "arC41" has reached zero, this means that all the arcs of node(i) were processed, therefore the SPP sets "arcDone" to one. Otherwise, the SPP sets arcDone to zero and keeps processing the rest of the arcs. The actions described above maps to the statement 10 in Figure 3. Using the address of the arc list fetched from the SPR, the SPP fetches both the adjacent node identifier (node(j)) and the link cost (cij). The adjacent node identifier will be loaded to "jReg" and the link cost will be loaded to "cijReg". Using the ALU, the SPP will add the distance label of node(i) and link cost (cij). Moreover, the SPP will compare the

results of summation with the adjacent distance label (d(j)). This task maps to statement 11 from Figure 3. If d(j) is greater than the sum (xGy), then the SPP will update the DR and PR registers. The DR register will contain the update value for d(j), which equals to the sum (d(i)+cij). This task maps to statement 12 in Figure 3. The PR register will have the predecessor of the adjacent node equal to the node index of node(i).

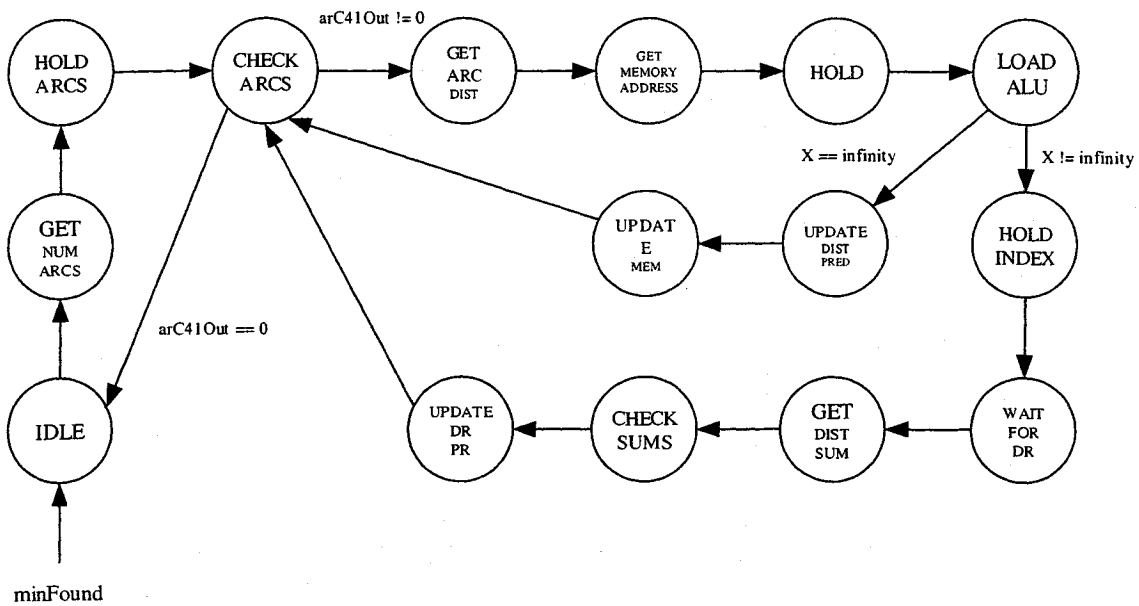


Figure 63 State 4 flow diagram

5.3 Data path design

The data path of the SPP is composed of static memory, five special registers (also referred to as dynamic memories AR, DR, PR, SR and SPR) and an arithmetic logic unit (ALU). The static memory is used to store the topology entries in the format of adjacency matrix. The dynamic memories are used to store various data for different purposes. The AR (Arc List Register) will store the number of arcs of each node, DR (Distance Label Register) will store the distance labels of each node, PR (Predecessor Register) will store the predecessor of each node and SR (Status Register) contains the list of nodes that were processed. The starting address of each node arc list in static memory will be stored in the SPR register. The ALU performs addition, subtraction and compare operations on the data passed from dynamic memories. In the upcoming subsections, design and implementation details for each data path element will be presented.

5.3.1 Static memory design and implementation

The SPP can only support 256 distinct node identifiers and topology entries. The total cost of any shortest path cannot exceed 65,535. Therefore, taking these assumptions into consideration, the data going in ("topologyEntry") and out of the memory ("memDataOut") is 24 bits long (8 bits for the node identifier and 16 bits for link cost).

5.3.1.1 Static memory input/output

The static memory has inputs for the address, data, read operation and write operation illustrated by the "memAdd", "topologyEntry", "memRead" and "memWrite" signals respectively as Figure 64. Connected to the output of static memory are the "cijReg" and "jReg" registers, which are used during the distance update operations and they are feed to the "X" and "Y" inputs of the ALU.

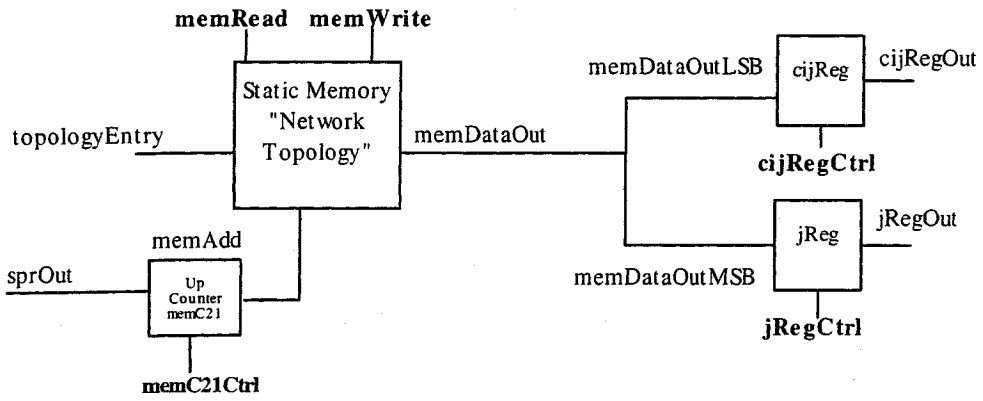


Figure 64 Static memory inputs and outputs

5.3.1.2 Static memory data path

The data path memory architecture is modified a version of [46]. This architecture was modified to fit the SPP needs. The static memory has 2 RAM columns The first RAM column (8bits) stores the node identifier and the second RAM column (16bits) stores the link cost. For example, storing the topology of the network graph shown in Figure 2 would cause the content of static memory to look like the Figure below.

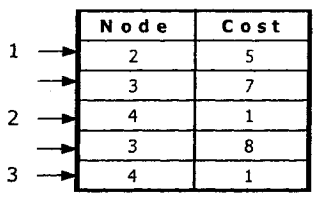


Figure 65 Memory storage example

5.3.2 General Register (GR) design and implementation

The general register is instantiated to build the AR, DR, PR, SPR and SR. The SPR is the only one that will be storing 10 bits long information and the other registers will store 8 bits long information such as the node identifier, link cost and distance labels.

5.3.2.1 General Register input/output

This general register takes in two control signals called "InCtrl" and "OutCtrl". The "InCtrl" signal is used as a select signal for the internal demultiplexer while the "OutCtrl" signal is used as a select signal for the internal multiplexer. The "DataIn" signal is used for inserting data into the register. Data is retrieved from the general register through the "DataOut" signal.

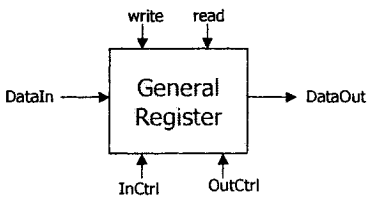


Figure 66 General register inputs and outputs

5.3.2.2 General Register design

The general register can be thought of as a hardware array with indices for input (InCtrl) and output (OutCtrl). The input index is implemented using a demultiplexer and the output index is implemented using a multiplexer as shown in the Figure 67.

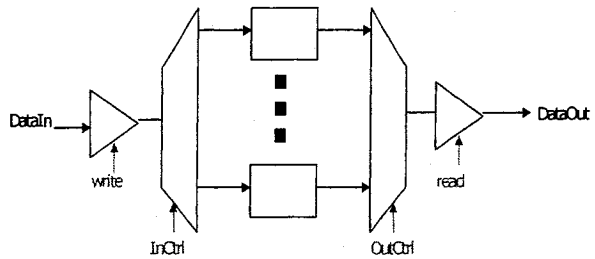


Figure 67 General register design

5.3.3 Arithmetic Logic Unit (ALU) design and implementation

As shown in Figure 68, the ALU takes two inputs from the data path (X and Y). Each of these inputs is 8 bits long. The third input, "aluCtrl" is used to instruct the ALU what kind of operations needs to be performed on "X" and "Y". The result of the operation gets stored in "Z". The "xGy" output is used in case of comparison puposes and it denotes "X > Y"

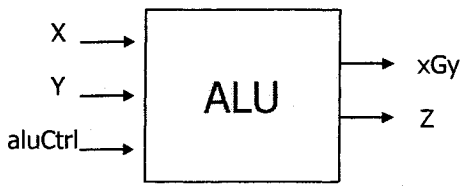


Figure 68 ALU inputs and outputs

5.3.4 Distance label register (DR)

The distance label register is used for storing the updated distance label for each node. At the beginning of Dijkstra's Algorithm, it is required to set all distance labels to infinity except for the source node, which will be set to zero. The DR design and implementation were already discussed in section 5.3.2. The goal of this section is to explain how the DR is integrated with the data path. Figure 69 illustrates how the DR is integrated with the ALU in the data path. Going from left to right, the "drInMux" manages what goes to "drIn", attached to this multiplexer is the following outputs:

- infinityRegister, which is used during initialization.

- zeroRegister, which is used to set the source node distance label to zero.
- ALU (zOut), which is used as feedback in case of distance updates.
- "drICMux" selects the appropriate address (or index) to where the "drIn" will be written, appended to this multiplexer is "drC21", which is a counter controlled by the control unit.
- "iReg", which is the node index (or node identifier) that identifies the current node's arc list being processed by SPP.
- "jReg", which is the node index of the node connected to node index in "I Register". (refer to Figure 3 flow chart for more details)
- "drOCMux" in dedicated for outputting the right info on "drOut", connected to this mux is the outputs of:
- "drC31", an up counter controlled by the control unit during the search for the minimum distance labels.
- "minInMux" is the multiplexer used to help "minReg" to store the value of the least cost distance label. At the beginning of Dijkstra's Algorithm (minInMuxCtrl = 0) the "minReg" signal is set low because of the source node. Otherwise, "minInMuxCtrl" is set to one to update lowest distance label computed by the ALU.
- On both of the ALU inputs, "X" and "Y", there are multiplexers to regulate different cases of input parameters that require various computations.

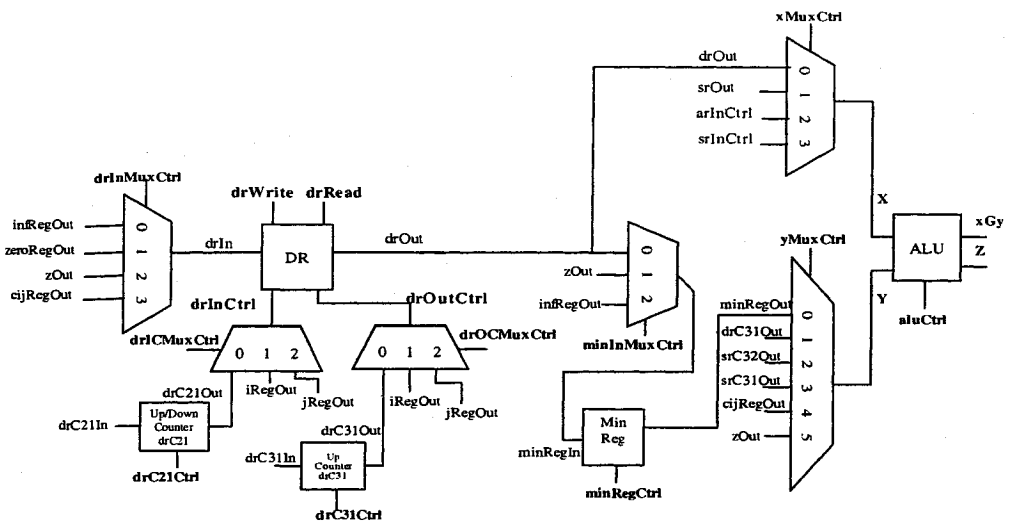


Figure 69 DR and ALU co-working

5.3.5 Arc List Register (AR) in data-path

The AR is used for storing the number of arcs for each node. This task is done by counting the number of new entries during the initialization period (see SPP control unit for more details). In terms of implementation, the AR is another instance of a general register. The building blocks around the AR in the data path are shown in Figure 70. "arC11", which is the counter that gets cleared every time "newNode" switches from one to zero, is incremented every time "newEntry" switches from one to zero. "arC21" is incremented every time "newNode" is switched from zero to one. The increment allows for the next available register in the array to be selected. "arC31" serves same function as "arC21" for retrieving values. "arC41" a counter that is used to evaluate if the selected node arc list has been all processed (i.e. it is used to perform statement 10 in Figure 3)

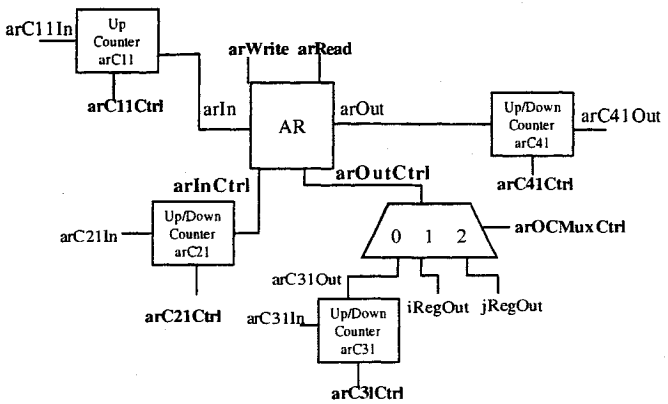


Figure 70 AR in data path

5.3.6 Status Register (SR)

SR's function is to capture the node identifier of previously processed nodes. The actual design of SR was already discussed in section 5.3.2. The components around SR in data path are shown in Figure 71.

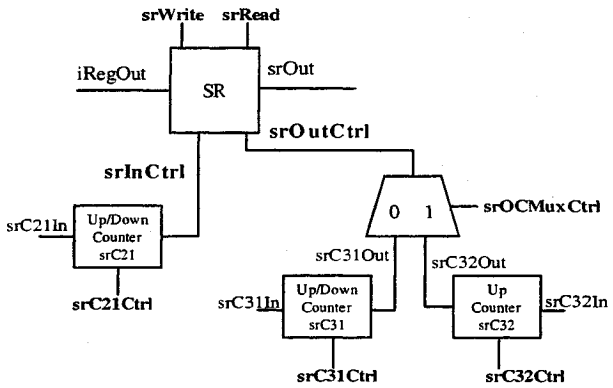


Figure 71 SR in data path

The "srC21" counter is used during the initialization state where it is required to set S to null. At this time "srIn" is connected to "zeroReg". The "srOCMux" multiplexer is used to regulate the selection of the node index by one of the counters "srC31" or "srC32". "srC32" is used to go through the SR array when the processor is avoiding nodes that have been already processed. The "srC31" counter is applied when the SPP is counting how many nodes it has processed.

5.3.7 Static Memory Address Register (SPR)

The SPR is used to store the start of each node's static memory address. This makes the task of retrieving each node's arc list easier. The implementation of the SPR was already discussed in section 5.3.2, the only difference between the SPR and the rest of the registers (AR, DR, SR and PR) is that it stores 10 bits long information address while the other registers store 8 bits long information (number of arcs, distance label value, node index or identifier). The elements that work with the SPR are illustrated in Figure 72. "sprC21" is incremented by the control unit every time "newNode" switches from one to zero. "sprC31" is incremented to select the node to be fetched from the static memory.

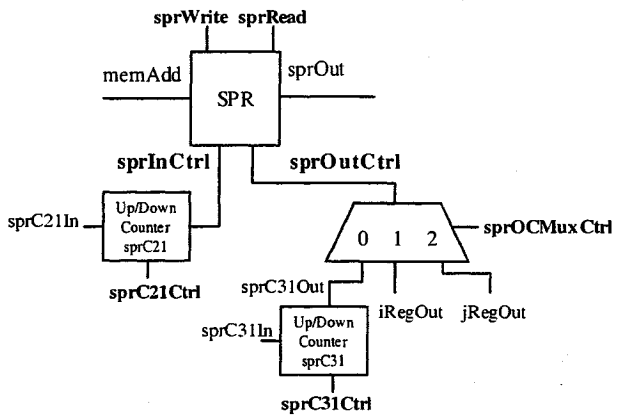


Figure 72 SPR in data path

5.3.8 Predecessor Register (PR)

The PR stores the predecessor of each node. PR is another instance of the general register. Figure 73 depicts the components that work with the PR.

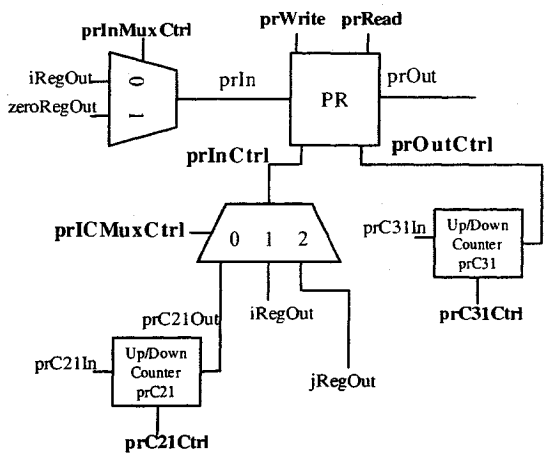


Figure 73 PR in data path

"prInMux" is used to load the PR with zeros during the initialization phase and with the "iReg" content when there is a successful distance update. "jReg" is used in selecting the address. Depending on the state of the control unit "prICMux" selects the right address. For example during initialization phase, counter "prC21" helps in going through the PR array and set all to zeros. "prC31" is used to read the PR contents sequentially.

5.3.9 Other data path blocks

Registers and counters (up counters and down counter) do exist in the SPP data path, most of these counters and registers have been already introduced in the previous sections. The rest of these registers are presented in this section. In Figure 74 , the three registers "iReg", "zeroReg" and "infReg" store the current values of the 'i' index, zero, and infinity respectively in performing iterations and comparisons for the various tasks of the SPP.

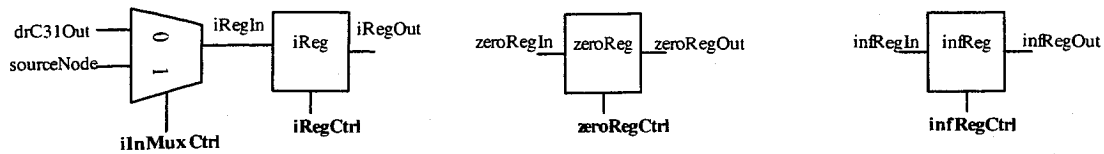


Figure 74 The rest of the data-path registers

6 Analysis and benchmarks

The testing and benchmarks required for the work presented in the thesis is quite large due to the nature of the topic. In reality, this nature of implementation requires a detailed test plan that is approved by hardware designer, firmware designer, software designer and network or system architect. This test plan includes tests scenarios from all levels. In this dissertation, the focus is more towards both hardware and networking level. Therefore, only these scenarios will be discussed.

From a hardware prospective, it is important to test all data path elements separately. Then, the data path elements can be integrated together to form what is known as the data path unit. Moreover, the full data path unit is tested along with the designed control unit. At this stage, the main focus is to ensure that the system inputs and outputs are processed properly as outlined by the control unit design.

From the networking prospective, the concerns are with the system level characteristics such as functionality, scalability and performance. This type of testing requires thorough analysis and therefore desires network simulators. Examples of performance testing are:

- Time to synchronize with neighboring routers.
- Time to react to a failure.
- Time to forward an LSA received from neighboring routers.
- Time to update database from LSA.

As the discussion above portrays, the networking tests required in order to conclude that the implementation is 100% functional, is limited to the shortage of network testing equipments.

The hardware testing part was completed for this study. This includes all the data path elements (counters, registers, memories, mux, demux, etc.) along with the control unit. The rest of this section describes the testing performed on the OSPF system (see section 6.1) and the SPP module (see section 6.2).

6.1 OSPF system testing scenarios

The tests that were completed for the OSPF system are presented in this section. The system inputs (see Figure 75) were constant through out the testing. However, the incoming packet simulation varies depending on the test scenario. The tests were executed using the ModelSim software.

```
//---------------------- Initial System Input Values Begin ----------
initial
begin
  newPacketBundle =1'b0;
  ingressPacketAvailable =1'b0;
  routerId= 32'b00000000000000000000000000000010;
  areaId= 32'b00000000000000000000000000000011;
  networkMask= 32'b00000000000000000000000000000100;
  helloInterval= 32'b00000000000000000000000000000101;
  deadInterval= 32'b00000000000000000000000000000110;
  designatedRouter= 32'b00000000000000000000000000000111;
  backupDesignatedRouter= 32'b00000000000000000000000000001000;
  slaveMaster= 1'b0;
  routerType= 32'b00000000000000000000000000001001;
  networkType= 32'b00000000000000000000000000001010;
  clk = 1'b0;
  reset = 1'b1;
end
//---------------------- Initial System Input Values End ------------
```

Figure 75 System inputs

6.1.1 Testing the exchange of Hello packet

6.1.1.1 Purpose

1) Ensure that the ingress processor is able to detect the type of OSPF packet (which is Hello packet).
2) Verify that the main processor is coordinating signals properly.
3) 0Confirm that the egress processor will reply with a Hello packet to the external router.

6.1.1.2 Results description

A Hello packet shown in Figure 76 was fed to the OSPF system.

```
//---------------------- Hello Packet Testing Begin -----------------
initial
begin
  #10 reset = 1'b0;
  #10
  #10 newPacketBundle = 1'b1;
      ingressPacketAvailable = 1'b1;
      ingressPacket= 32'b00000000000000010000000000101100; // Start of OSPFH (Version,
                                                              Type, packetLength)
  #10 ingressPacket= 32'b00000000000000000000000000000010; // RouterID
  #10 ingressPacket= 32'b00000000000000000000000000000011; // AreaID
  #10 ingressPacket= 32'b00000000000000000000000000000100; // CheckSum + Authentication
                                                              Type
  #10 ingressPacket= 32'b00000000000000000000000000000101; // Authentication
  #10 ingressPacket= 32'b00000000000000000000000000000110; // End of OSFPH
                                                              (Authentication)

  #10 ingressPacket= 32'b00000000000000000000000000000111; // Start of Hello packet
                                                              (Network Mask)
  #10 ingressPacket= 32'b00000000000000000000000000001000; // Hello Interval
  #10 ingressPacket= 32'b00000000000000000000000000001001; // Dead Interval
  #10 ingressPacket= 32'b00000000000000000000000000001010; // Designated Router
  #10 ingressPacket= 32'b00000000000000000000000000001011; // Backup designed Router
  #10
  #10 ingressPacketAvailable = 1'b0;
end

//---------------------- Hello Packet Testing End -------------------
```

Figure 76 Hello test packet

In Figure 77, the ingress processor detects that the incoming packet is a Hello packet (this done by reading OSPF type field) and hence activates inHello queue by selecting the ingressDemuxCtrl to be zero. The Hello packets are arriving through the ingressPacket data signal and they are being stored automatically into the inHello queue (see inHelloIn data signal and inHelloWrite control signal). The instant the advertising router field is detect, the ingress processor loads that value to into the inHelloTmpReg register (see inHelloTmpRegOut data signal). Along with that, the inHelloC21 counter is being incremented every time a 32 bits segment packet is written into the inHello queue (see inHelloC21Out data signal). The inHelloC22 counter is incremented after all the 32 bits segments have successfully stored (see inHelloC22Out at the line cross labeled with 145 ps). The inHelloMC control signal is set to high (see line cross labeled with 35 ps) upon detecting that the arriving packets are Hello packets and hence this informs the main processor that it

is time to set mcType1 control signal to one, which invokes the egress processor to start assembling a reply Hello packet. At line cross labeled with 145 ps, the inHelloCom is set to high to inform the main processor that all Hello packets have been queued.

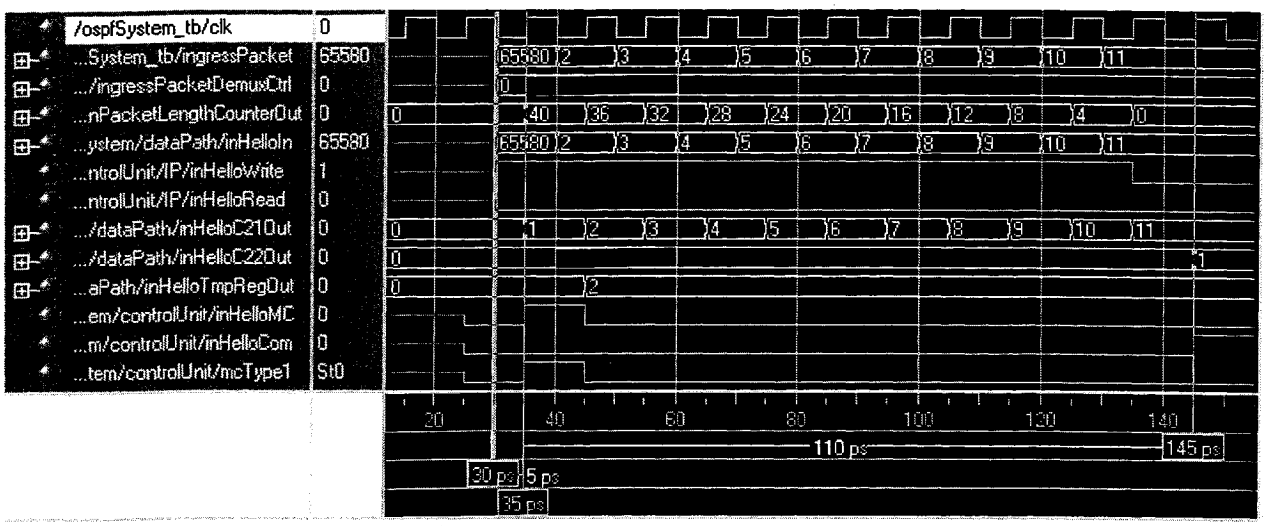


Figure 77 First Hello cross section timing diagram

Figure 78 is the continuation of the cross section of Figure 77. In Figure 78, the focus is on the egress path where it supposes to be outputting a reply Hello packet. At line cross labeled with 55 ps, the egress processor has stored the advertising router id passed by the inHelloTmpReg into the egHello queue (see egHelloIn data signal and egHelloWrite control signal). Also, the egHelloPlc counter has loaded the default packet length field (see egHelloPlcOut data signal = 33620012). At line cross labeled with 65ps, the egress processor increments the packet length field which is reflected on the egHelloPlcOut data signal (increment by 4 because 32 bits = 4 * 8 bytes). The increment occurred due to additional advertising router Id that was just added to egHello queue. Furthermore, the egress processor has started transmitting the Hello packets (starting with OSPF header first and then Hello segments), which can be seen on the egressPacket data signal.

As discussed in section 3.3.3.1, the last segments of Hello packet are the neighbors Ids, which translates to the advertising routers stored into egHello queue. In this testing scenario, there is only one advertising router Id stored, which is equal to 2. Hence, the line cross labeled with 175 ps shows that the advertising router Id (or neighbohr Id) has been read from

egHello queue (see egHelloOut data signal and egHelloRead control signal) and placed onto the egressPacket data signal.

At line cross labeled with 205 ps, the egress processor sets egHelloCom to one, which informs the main processor that a Hello reply packet has been successfully transmitted to the external router.

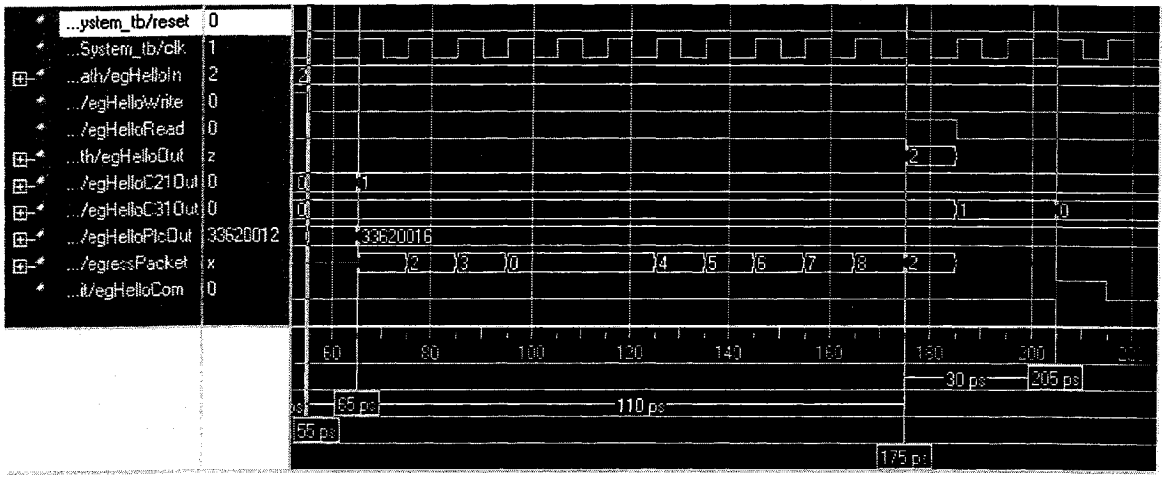


Figure 78 Second Hello cross section timing diagram

6.1.2 Testing the processing of DD and LS Update packets

From the implementation prospective, the DD packets and the LS updates packets are handled the same way. In case of DD packets the OSPF system has to reply to the external router with sequence number updated. In case of the LS updates, the OSPF system has to reply back with LS acknowledgement. Hence, the testing below will go with LS Updates because its more commonly used once the router has been initialized.

6.1.2.1 Purpose

1) Ensure that the ingress processor is able to detect the type of OSPF packet, which is LS update packet.
2) Verify that the main processor is coordinating signals properly.

3) Confirm that the egress processor will reply with an LS acknowledgement packet to the external router.
4) Validate that the OSPF system executes the database synchronization among lsaDb1 and lsaDb2.
5) Check if the OSPF system sends updates (topologyEntry, newNode and newEntry) to the SPP module.

6.1.2.2 Results description

For the purpose of this testing, assume that the network shown in Figure 79 is used to simulate the LS update packet (shown in Figure 80) was fed to the OSPF system.

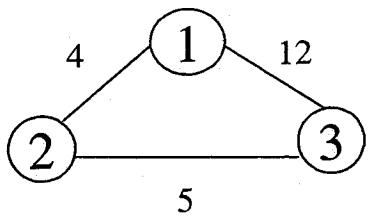


Figure 79 Network used for testing LS updates

```
//---------------------- LsUpd Packet Testing2 Begin -----------------
initial
begin

  #10 reset = 1'b0;
  #10
  // Simulating the router's own LSAs, i.e. Receiving provisioning commands from operator
  #10 newPacketBundle = 1'b1;
      ingressPacketAvailable = 1'b1;
      ingressPacket= 32'b00000000000001000000000001001000; // Start of OSPF Header
                                                           (Version, Type,
                                                           packetLength=18*4=72 bytes)
  #10 ingressPacket= 32'b00000000000000000000000000000001; // RouterID
  #10 ingressPacket= 32'b00000000000000000000000000000011; // AreaId
  #10 ingressPacket= 32'b00000000000000000000000000000100; // CheckSum + Authentication
                                                              Type
  #10 ingressPacket= 32'b00000000000000000000000000000101; // Authentication
  #10 ingressPacket= 32'b00000000000000000000000000000110; // End of OSPF Header
                                                           (Authentication)

  #10 ingressPacket= 32'b00000000000000001000000000000001; // Start of LSA Header
                                                           (LsAge, Options, LsType)
  #10 ingressPacket= 32'b00000000000000000000000000000001; // Link State ID
  #10 ingressPacket= 32'b00000000000000000000000000000001; // Advertising Router
  #10 ingressPacket= 32'b00000000000000000000000000001100; // Ls Sequence Number
  #10 ingressPacket= 32'b00000000000000000000000000110000; // End of LSA Header
                                                           (LsCheckSum and
                                                           Length=12*4=48 bytes)

  #10 ingressPacket= 32'b00000000000000000000000000000010; // Start of Router LSA
                                                           (VEB, Number of Links = 2)
  #10 ingressPacket= 32'b00000000000000000000000000000010; // Link ID
```

```
#10 ingressPacket= 32'b00000000000000000000000000000010; // Link Data
#10 ingressPacket= 32'b00000000000000000000000000000100; // LinkType,  NumberofToS,
                                                            Metric =4

#10 ingressPacket= 32'b00000000000000000000000000000011; // Link ID
#10 ingressPacket= 32'b00000000000000000000000000000011; // Link Data
#10 ingressPacket= 32'b00000000000000000000000000001100; // LinkType,  NumberofToS,
                                                            Metric =5
#10
#10 ingressPacketAvailable = 1'b0;

#10
  //Simulating live router-LSAs arriving from router Id 2
#10 newPacketBundle = 1'b1;
    ingressPacketAvailable = 1'b1;
    ingressPacket= 32'b00000000000001000000000001001000; // Start of OSPF Header
                                                            (Version, Type,
                                                            packetLength=18*4=72 bytes)
#10 ingressPacket= 32'b00000000000000000000000000000010; // RouterID
#10 ingressPacket= 32'b00000000000000000000000000000011; // AreaId
#10 ingressPacket= 32'b00000000000000000000000000000100; // CheckSum + Authentication
                                                               Type
#10 ingressPacket= 32'b00000000000000000000000000000101; // Authentication
#10 ingressPacket= 32'b00000000000000000000000000000110; // End of OSPF Header
                                                            (Authentication)

#10 ingressPacket= 32'b00000000000000001000000000000001; // Start of LSA Header
                                                            (LsAge, Options, LsType)
#10 ingressPacket= 32'b00000000000000000000000000000010; // Link State ID
#10 ingressPacket= 32'b00000000000000000000000000000010; // Advertising Router
#10 ingressPacket= 32'b00000000000000000000000000001100; // Ls Sequence Number
#10 ingressPacket= 32'b00000000000000000000000000110000; // End of LSA Header
                                                            (LsCheckSum and
                                                            Length=12*4=48 bytes)
#10 ingressPacket= 32'b00000000000000000000000000000010; // Start of Router LSA
                                                            (VEB, Number of Links = 2)
#10 ingressPacket= 32'b00000000000000000000000000000001; // Link ID
#10 ingressPacket= 32'b00000000000000000000000000000001; // Link Data
#10 ingressPacket= 32'b00000000000000000000000000000100; // LinkType,  NumberofToS,
                                                            Metric =4
#10 ingressPacket= 32'b00000000000000000000000000000011; // Link ID
#10 ingressPacket= 32'b00000000000000000000000000000011; // Link Data
#10 ingressPacket= 32'b00000000000000000000000000000101; // LinkType,  NumberofToS,
                                                            Metric =5
#10
#10 ingressPacketAvailable = 1'b0;
#10

//Simulating live router-LSAs arriving from router Id 3
#10 newPacketBundle = 1'b1;
    ingressPacketAvailable = 1'b1;
    ingressPacket= 32'b00000000000001000000000001001000; // Start of OSPF Header
                                                            (Version, Type,
                                                            packetLength=18*4=72 bytes)
#10 ingressPacket= 32'b00000000000000000000000000000011; // RouterID
#10 ingressPacket= 32'b00000000000000000000000000000011; // AreaId
#10 ingressPacket= 32'b00000000000000000000000000000100; // CheckSum + Authentication
                                                               Type
#10 ingressPacket= 32'b00000000000000000000000000000101; // Authentication
#10 ingressPacket= 32'b00000000000000000000000000000110; // End of OSPF Header
                                                            (Authentication)
#10 ingressPacket= 32'b00000000000000001000000000000001; // Start of LSA Header
                                                            (LsAge, Options, LsType)
#10 ingressPacket= 32'b00000000000000000000000000000011; // Link State ID
#10 ingressPacket= 32'b00000000000000000000000000000011; // Advertising Router
#10 ingressPacket= 32'b00000000000000000000000000001100; // Ls Sequence Number
#10 ingressPacket= 32'b00000000000000000000000000110000; // End of LSA Header
                                                            (LsCheckSum and
                                                            Length=12*4=48 bytes)
#10 ingressPacket= 32'b00000000000000000000000000000010; // Start of Router LSA
                                                            (VEB, Number of Links = 2)
```

```
  #10 ingressPacket= 32'b00000000000000000000000000000001; // Link ID
  #10 ingressPacket= 32'b00000000000000000000000000000001; // Link Data
  #10 ingressPacket= 32'b00000000000000000000000000001100; // LinkType,  NumberofToS,
                                                              Metric =12
  #10 ingressPacket= 32'b00000000000000000000000000000010; // Link ID
  #10 ingressPacket= 32'b00000000000000000000000000000010; // Link Data
  #10 ingressPacket= 32'b00000000000000000000000000000101; // LinkType,  NumberofToS,
                                                              Metric =5
  #10
  #10 ingressPacketAvailable = 1'b0;
end
//---------------------- LsUpd Packet Testing2 End ------------------
```

Figure 80 LS update test packet

Figure 81 shows that the ingress processor was able to detect that the incoming packet is an LS update packet (see inLsUpdWriteDb control signal, this signal is set by the main processor only if the ingress processor detected an LS update packet). The LS Update packets commence to be written into lsaDb2 the instant the LSA header is detected (see lsaDb2Write control signal and wlsaDb2Out data signal at line cross 85 ps all the way to line cross 205ps). Upon receiving all the LS update packet the ingress processor sets the inLsUpdCom control signal to one to inform the main processor that the first LS update packet was completely received. Then the main processor asks the lsaDb processor to update its indirect memory address (see idDb2Write control signal and idDb2C21Out data signal at line cross labeled with 225ps). In this way, the lsaDb processor is ready for other incoming packets.

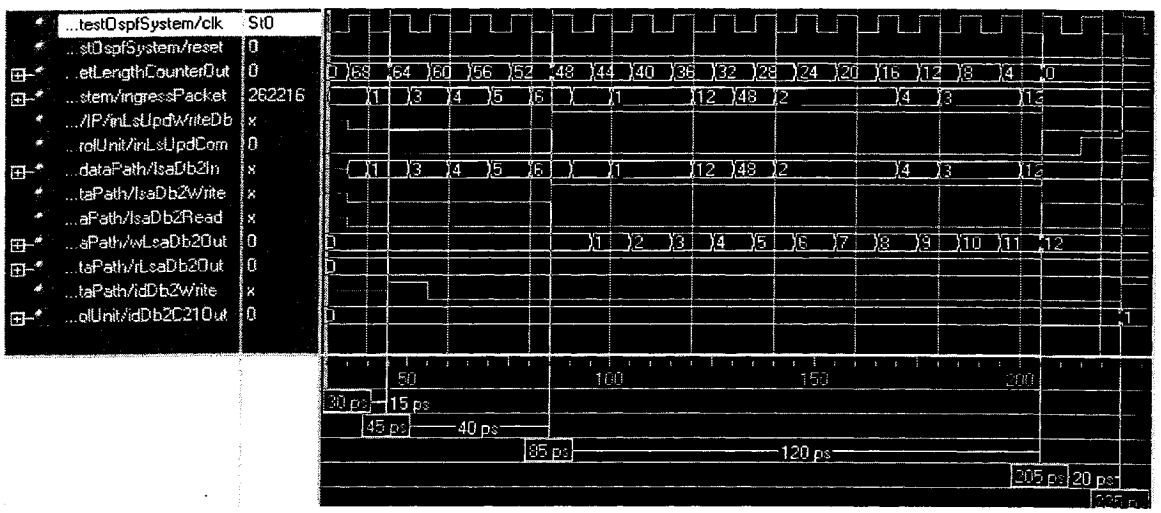


Figure 81 First cross section of LS update timing simulation

Since this test had 2 more LS Updates, below are two figures that correspond to LS updates from router Id 2 and router Id 3.

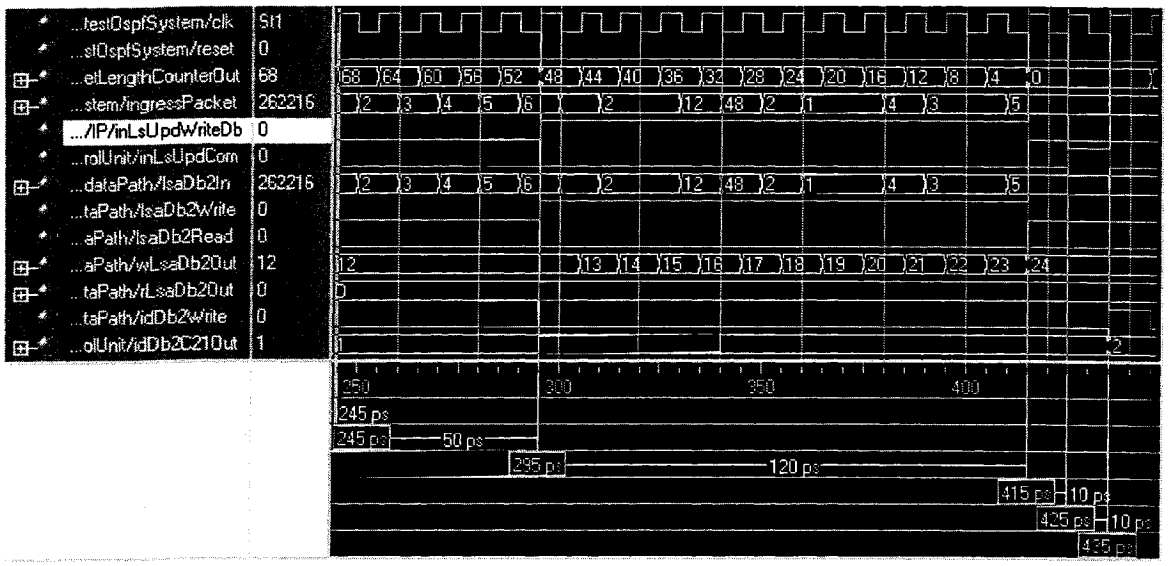


Figure 82 Second cross section of LS update timing simulation

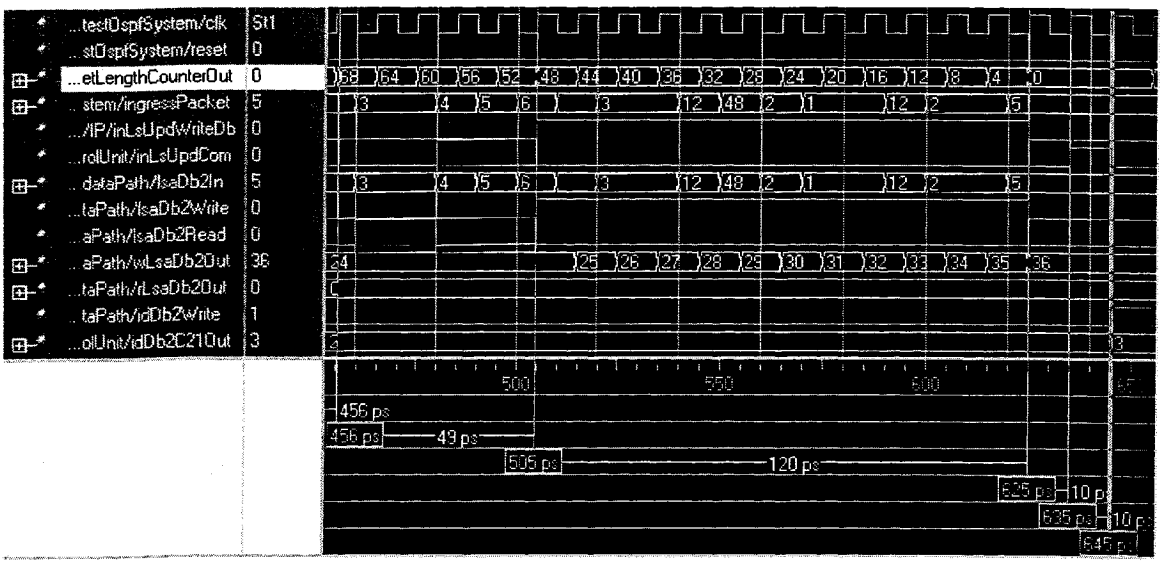


Figure 83 Third cross section of LS update timing simulation

From OSPF protocol prospective, each received LS update packet needs to be acknowledged by an LS acknowledgement packet. Hence, Figure 84, which is continuation of the simulation, shows that three acknowledgement packets were sent out on the egress path. One LS acknowledgment per router (Id 1, Id 2 and Id 3). Each time the inLsUpdCom control

signal was set to one by the ingress processor, it forces the main processor to send the egress processor a mcType5 command signal, which forces the egress processor to assemble and send out an LS acknowledgement packet. Figure 84 shows that the egressPacketMuxCtrl signal switches from value zero to value five. The value zero denotes that the egress processor is sending OSPF header and value five denotes the LS acknowledgement portion (refer to section 3.3.3.5 for more details)

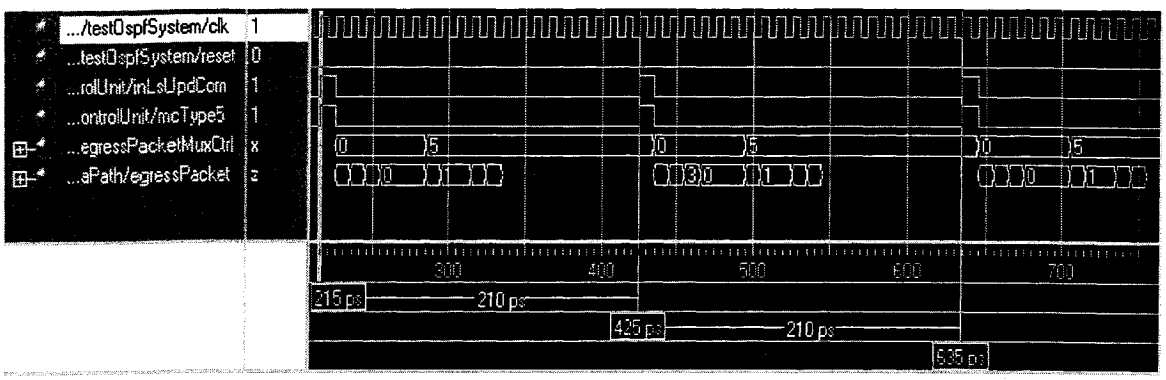


Figure 84 Fourth cross section of LS update timing simulation

Database synchronization between lsaDb1 and lsaDb2 was not performed because the system started with empty lsaDb1 database. The purpose of this synchronization was to ensure that the lsaDb processor will only keep the LSAs with the highest sequence number. Figure 85 shows that the lsaDb processor reads the advertising router and compares it with other advertising routers stored in the lsaDb1. However, the lsaDb1 database is empty. Since there were three LSAs stored, it can be seen from Figure 85 that the lsaDb processor actually goes to lsaDb2 and reads each advertising router (see lsaDb2Out at line crosses 745 ps, 775 ps and 805 ps)

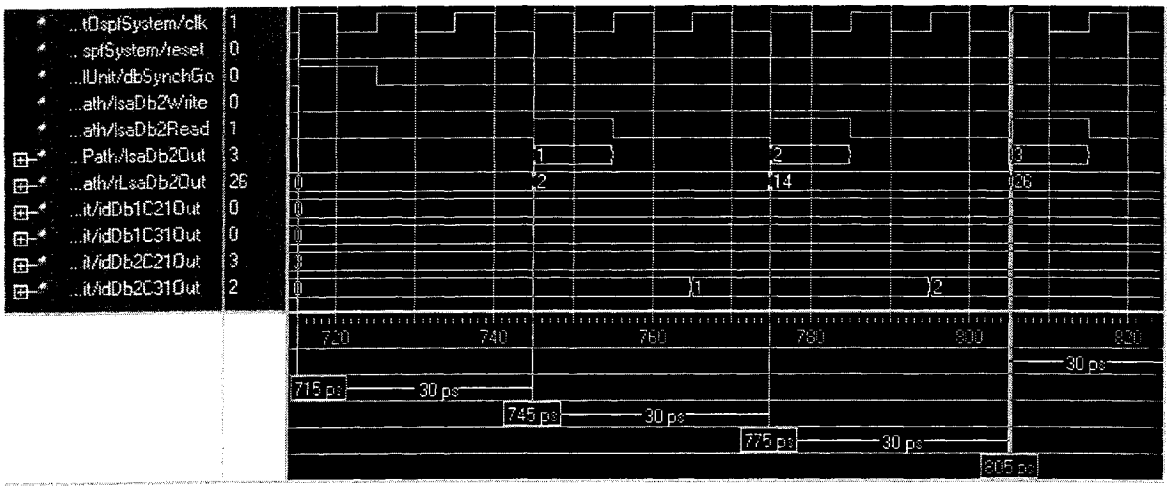


Figure 85 Fifth cross section of LS update timing simulation

The next state outlined by the lsaDb processor is the zeroCleanUp state. Since the lsaDb1 is empty and there were no zeros inserted into LSA's header. The zeroCleanUp state basically performs a database migration from lsaDb2 to lsaDb1. Given that there were three LSAs written into lsaDb2, Figure 86 shows the three LSAs have been migrated from lsaDb2 to lsaDb1 (see lsaDb2Out and lsaDb1In data signals at 885 ps, 1036 ps and 1185 ps).

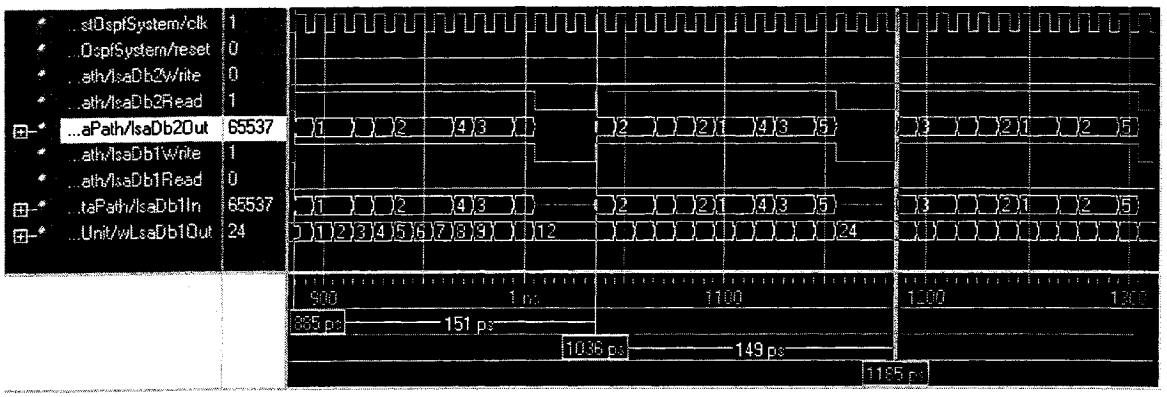


Figure 86 Sixth cross section of LS update timing simulation

The next state from the lsaDb processor is the synchDb1SPP state, where the lsaDb processor informs the SPP module about the changes occurring in the network. To convey these changes the lsaDb processor uses topologyEntry data signal, newNode and newEntry control signals. In this case, there are three nodes and each node has two links, therefore the

lsaDb processor should raise the newNode signal three times and raise the newEntry signal three times as well. In total, the topologyEntry signal will be entered six times. At the end of this cycle the lsaDb processor will set endOfFile to one, which forces the SPP module to compute the shortest path.

Figure 87 and Figure 88 show the lsaDb processor updating the SPP module with node 1 links information. Figure 87 shows the update done for the arc connecting node 1 and node 2 (see tmpAdvRouterRegOut at 1445 ps), where the newNode and the topologyEntry signals were set at the line cross labeled with 1465 ps. Figure 88 shows the update for the arc connecting node 1 and node3 (see tmpAdvRouterReg at 1595 ps), where the newEntry along with topologyEntry were set at 1615 ps.

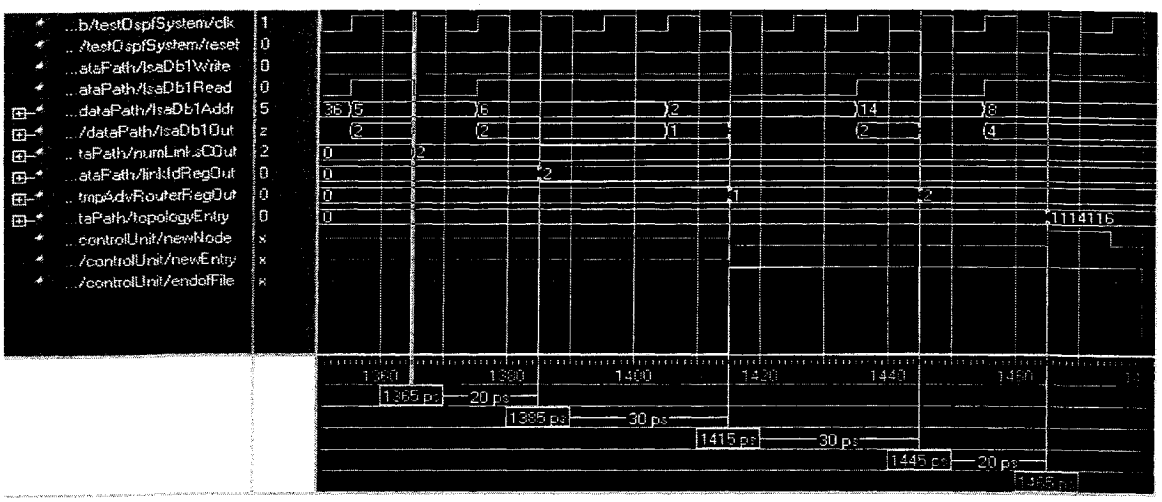


Figure 87 Seventh cross section of LS update timing simulation

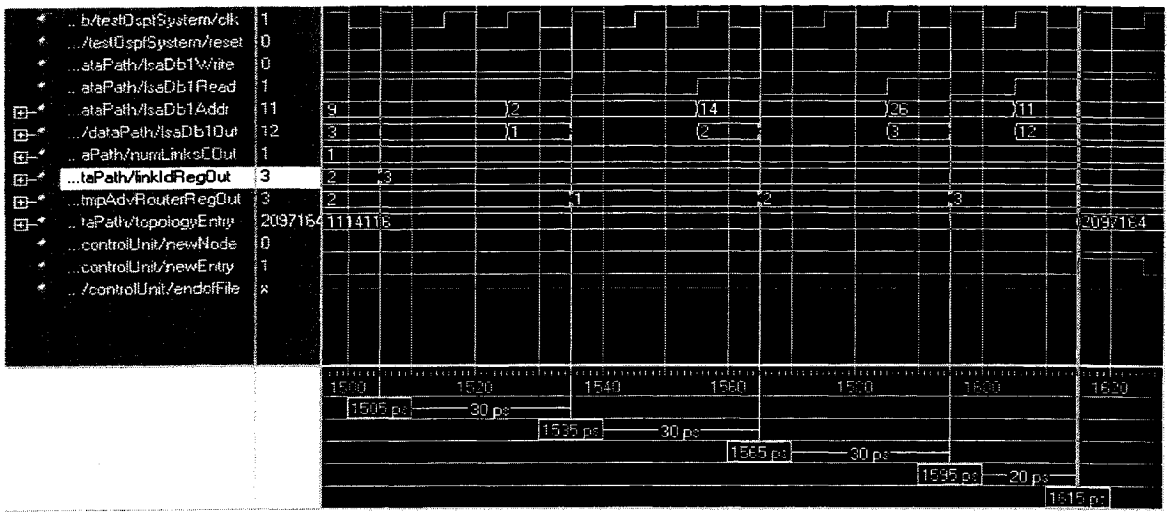


Figure 88 Eight cross section of LS update timing simulation

Figure 89 and Figure 90 show the lsaDb processor updating the SPP module with information related to node 2. Figure 89 shows the update for the arc connecting node 2 with node 1 and Figure 90 shows the update for the arc connecting node 2 with node 3.

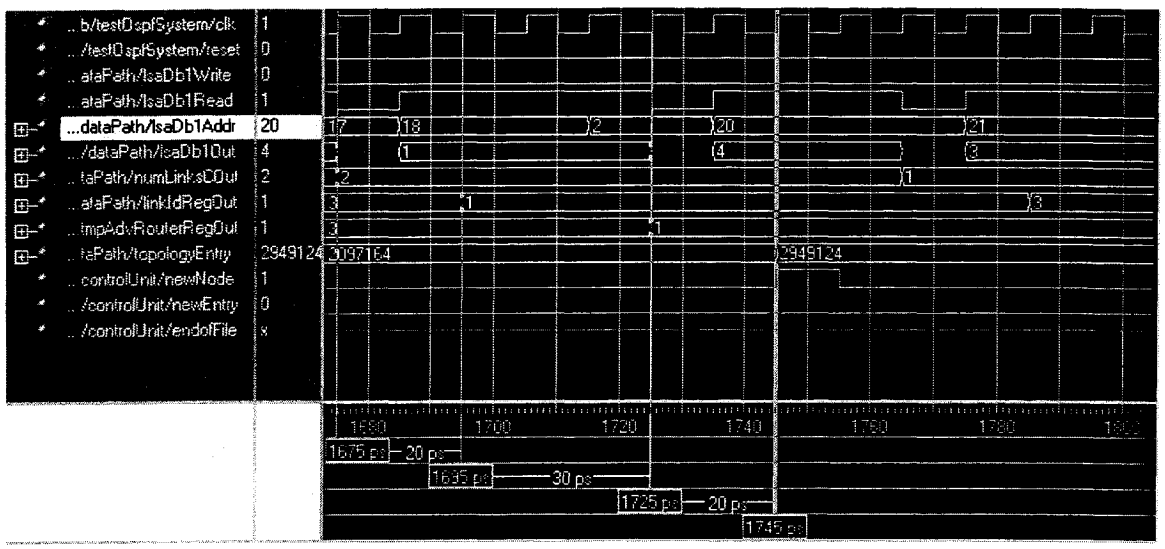


Figure 89 Ninth cross section of LS update timing simulation

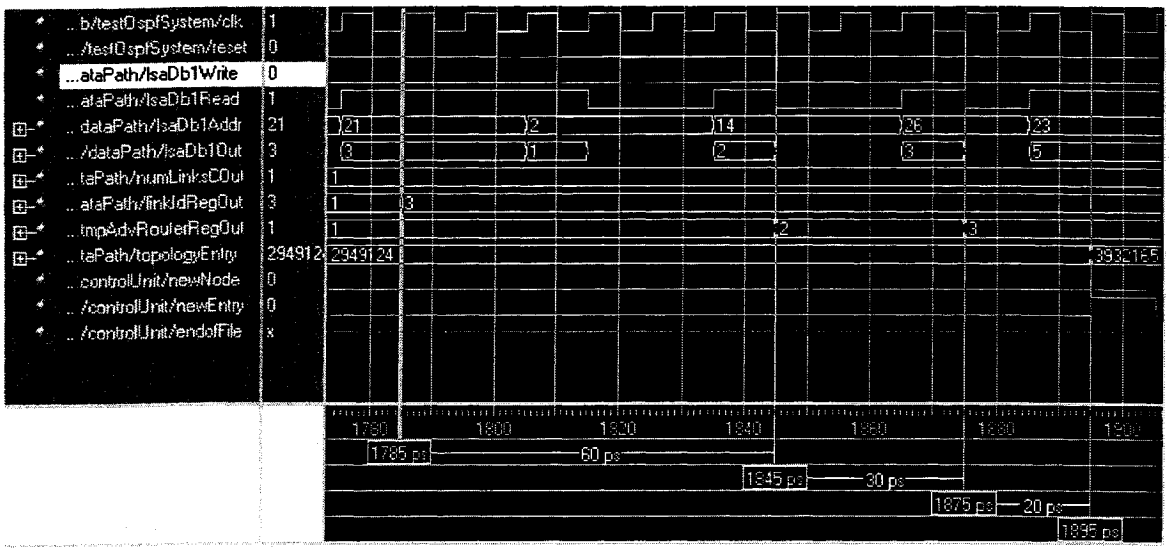


Figure 90 Tenth cross section of LS update timing simulation

Figure 91 shows the lsaDb processor updating the SPP module with node 3 links information. The update for the arc connecting node 3 with node 1 occurs at 2025 ps and the arc connecting node 3 with node 2 occurs at 2145 ps. Since, this sums all the network topology shown Figure 79, the lsaDb processor sets the endOffile signal to one at 2175 ps, which forces the SPP module to compute the shortest path.

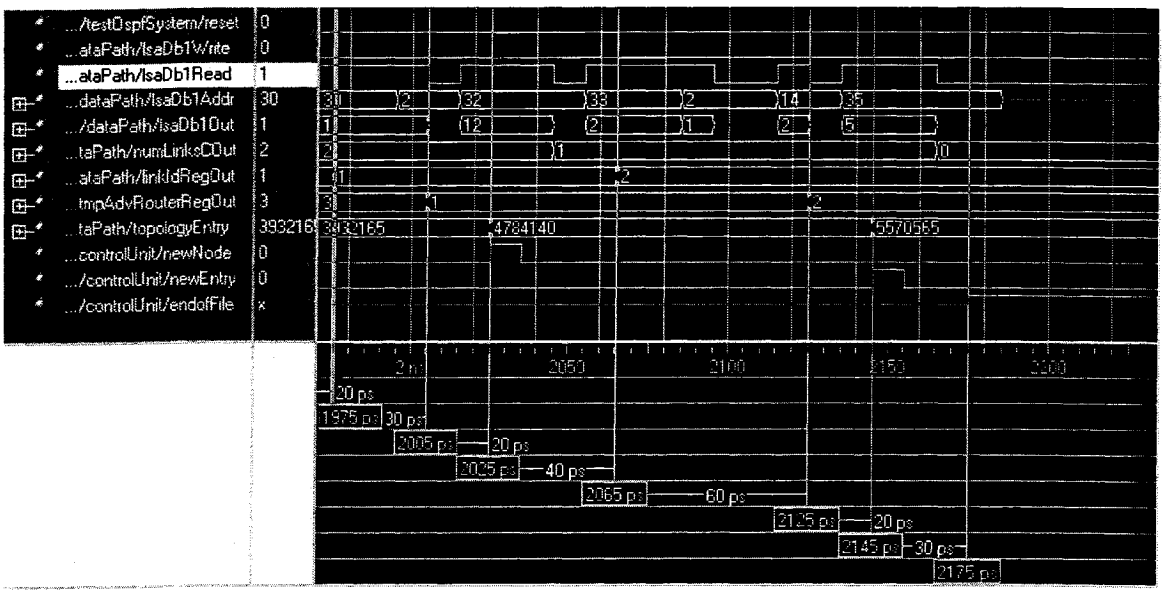


Figure 91 Eleventh cross section of LS update timing simulation

6.2 SPP testing scenarios

This section covers the testing performed towards the SPP module. It includes functionality testing plus performance benchmarks.

6.2.1 SPP functionality testing

The current version of the SPP was implemented using the Modelsim development environment. This version accommodates 7 nodes network.

6.2.1.1 Purpose

To verify if the SPP module computes the shortest path properly.

6.2.1.2 Results description

Figure 92 illustrates the network topology used for testing the SPP module, while Figure 93 and Figure 94 show a cross section of the simulation. Node 1 is the source node. The drWrite and prWrite signals are used to store the distance and predecessor of a node respectively. drIinCtrl and prInCtrl represent the node ID while drIn and prIn represent the distance and predecessor respectively. In Figure 93, the predecessor and distance of the source node are both zero. Node 2 has a predecessor of 1 with a distance of 1. In Figure 92, nodes 3 and 4 also have a predecessor of 1 with distances of 2, as indicated in Figure 93.

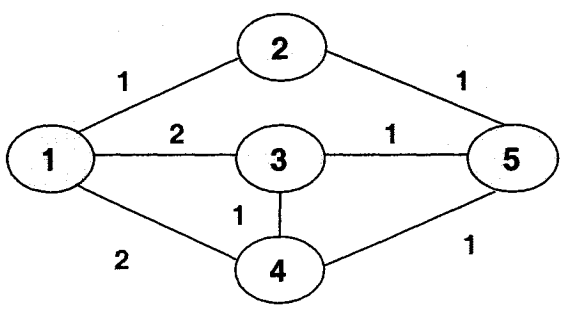


Figure 92 Network topology for simulation

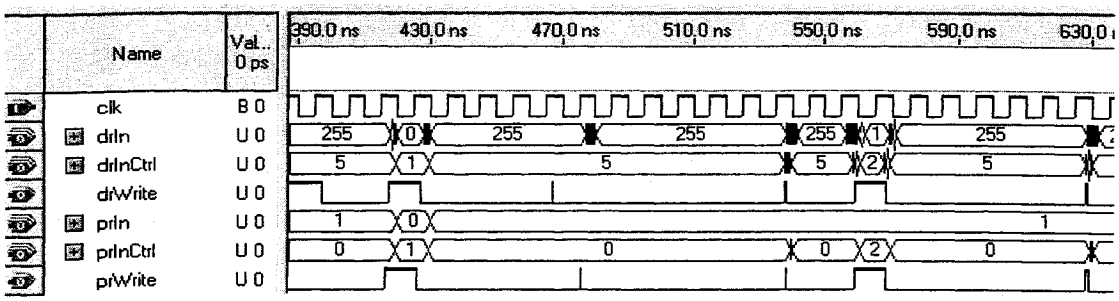


Figure 93 First section of SPP simulation

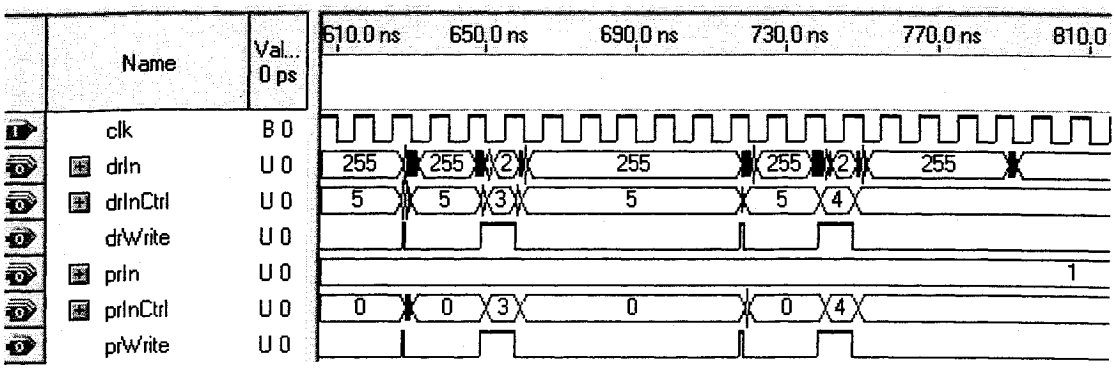


Figure 94 Second section of SPP simulation

6.2.2 SPP performance benchmarks

The SPP performance characterization will benchmark up to 100 nodes due to PC system resource (CPU and memory) constraints. The networks used to benchmark the SPP implementation are shown in Figure 96 through Figure 103. Code used in section 6.2.1 was expanded to perform this characterization.

6.2.2.1 Purpose

To characterize the SPP performance improvement (SW vs. HW)

6.2.2.2 Results description

The table and graph below illustrate experimental results for the SPP. All software benchmarks were performed using C based implementations of Dijkstra's algorithm.
The average software cycles shown in Table 6, was computed based on 50 simulation runs per network. Table 6 shows the percentage improvement gained using the proposed hardware architectures. Furthermore, the designed SPP shows an improvement in computation time, which contributes in minimizing the delay of recalculating the shortest path. The variations in percentage improvements are due to different network topologies (i.e. its dependent on the *number of nodes* and *number of arcs* in a network). Further performance improvements can be achieved if tasks computation can be parallelized (i.e. SPP Design can be optimized).

Table 6 SPP performance

Number of Nodes	Measured Peformance Data			Computation Time, (msec) with a speed of 450Mhz		Percentage Improvement
	Avg. Software cycles	Software Standard Deviation	Hardware cycles	Software	Hardware	
30	14,510	674	7,841	0.03	0.02	46%
40	22,350	394	12,306	0.05	0.03	45%
50	31,800	1,064	20,472	0.07	0.05	36%
60	41,550	954	29,117	0.09	0.06	30%
70	52,250	600	39,292	0.12	0.09	25%
80	64,140	1,010	49,662	0.14	0.11	23%
90	76,020	1,102	64,022	0.17	0.14	16%
100	90,880	1,372	78,637	0.20	0.17	13%

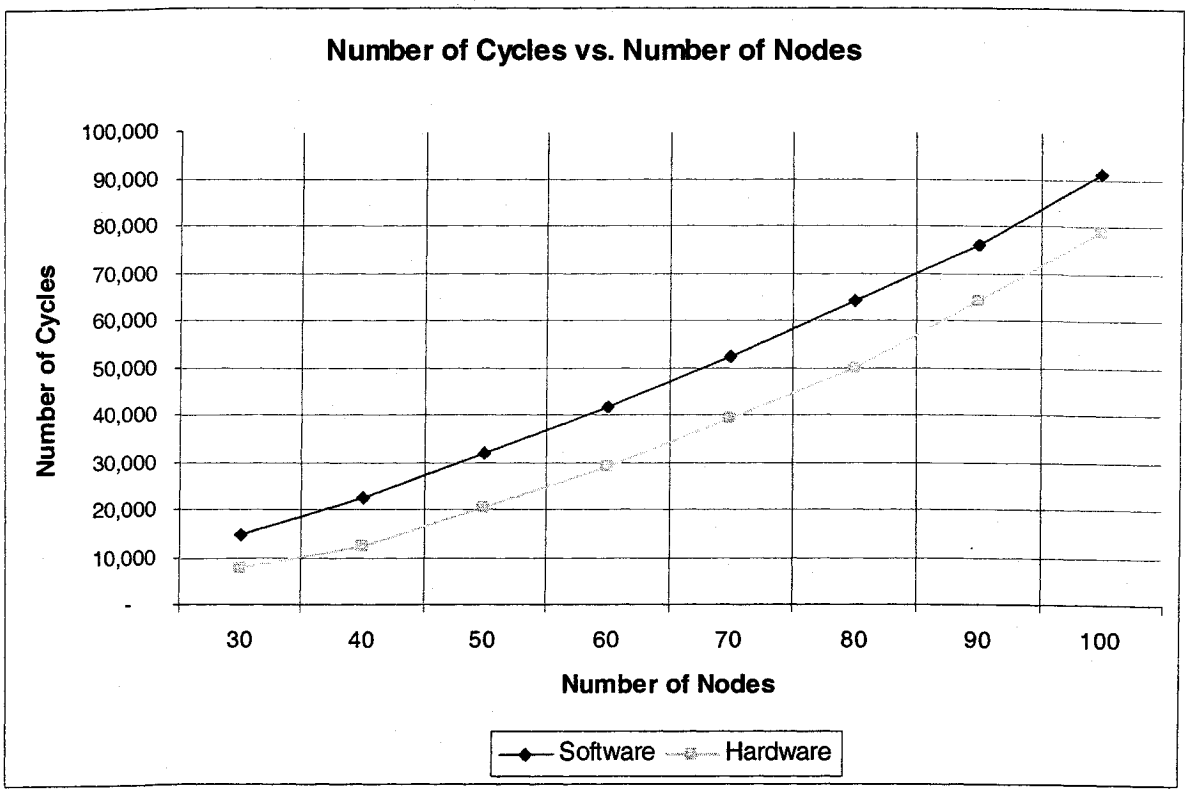


Figure 95 SPP performance graph

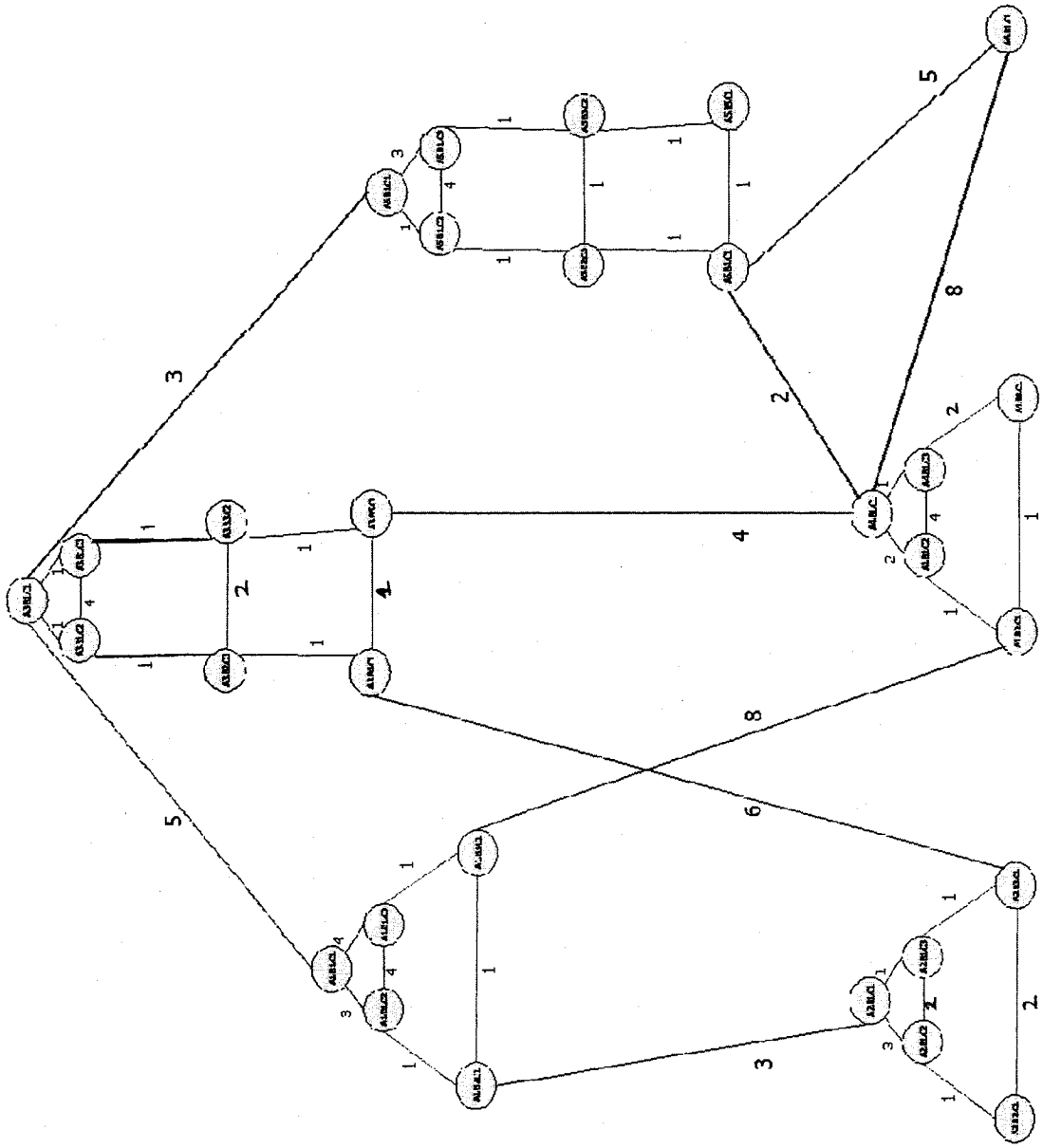


Figure 96 Thirty nodes network topology

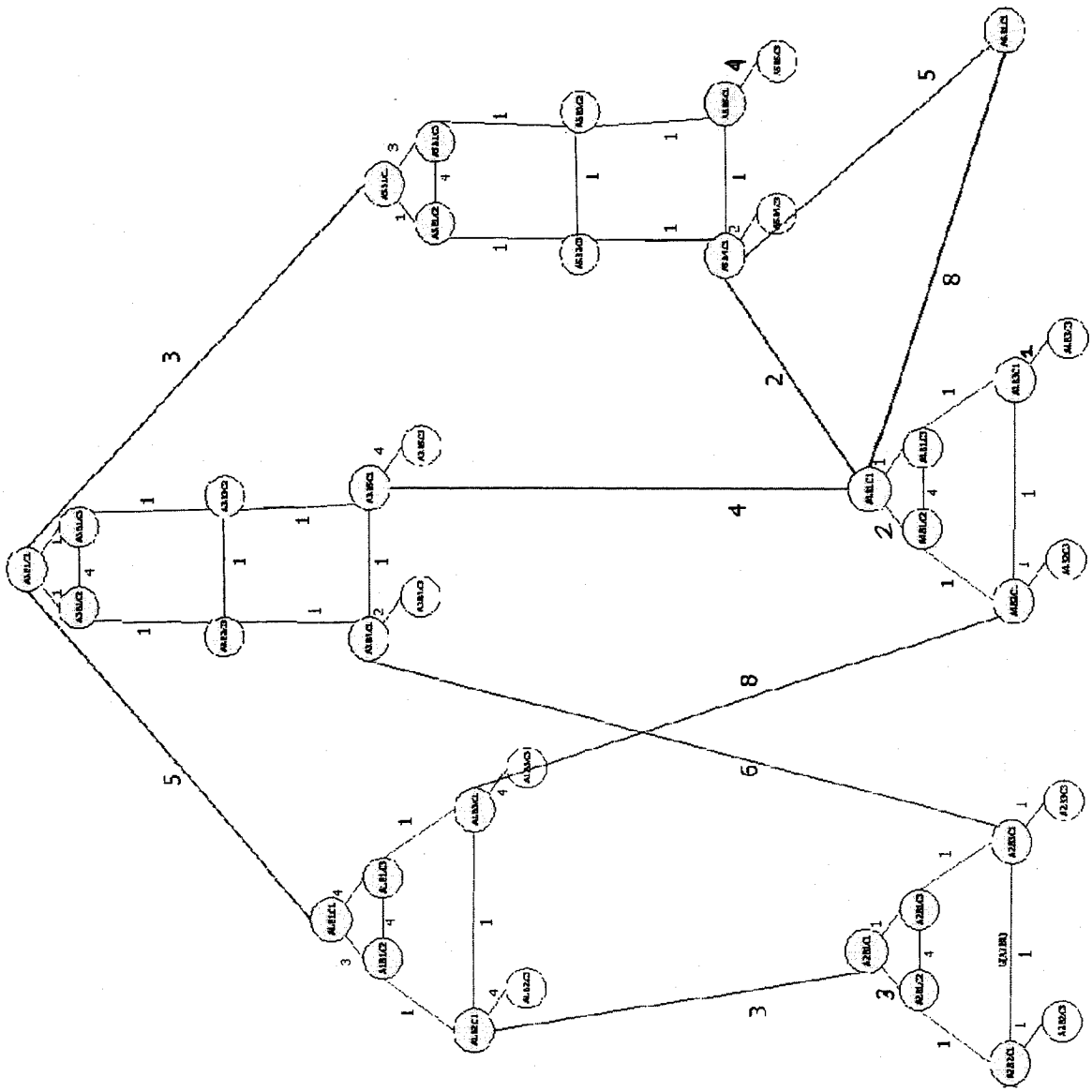


Figure 97 Forty nodes network topology

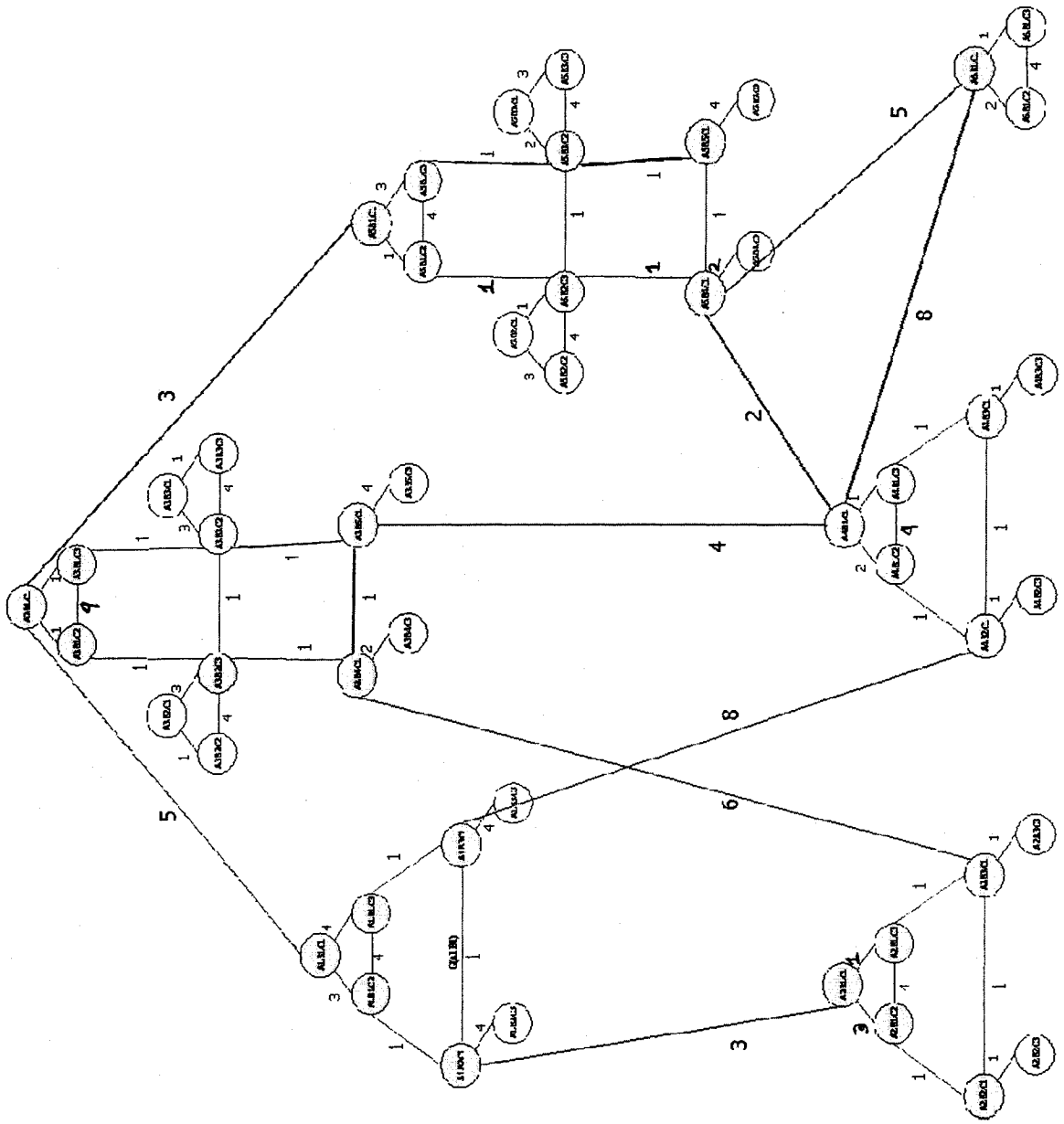


Figure 98 Fifty nodes network topology

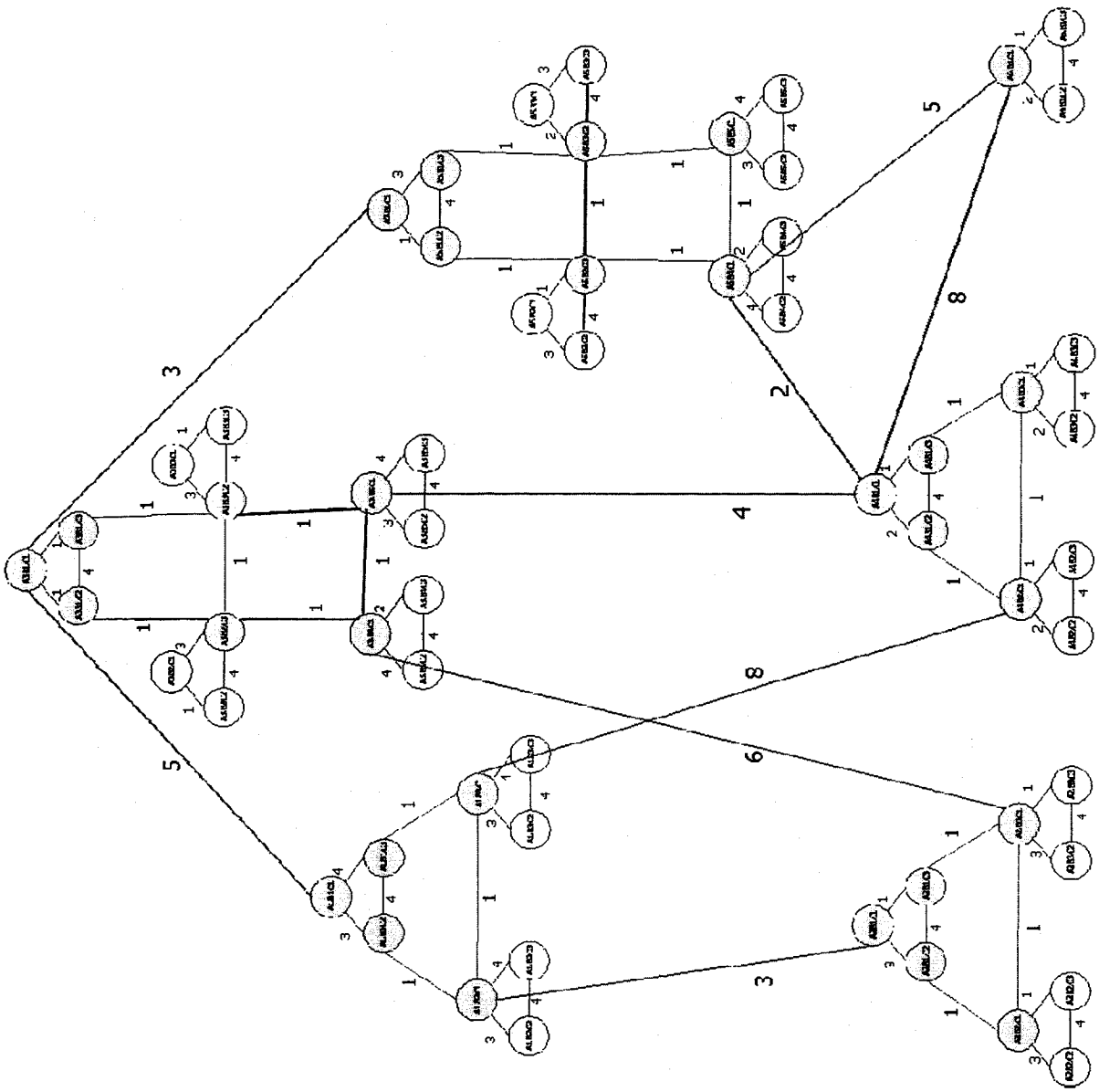


Figure 99 Sixty nodes network topology

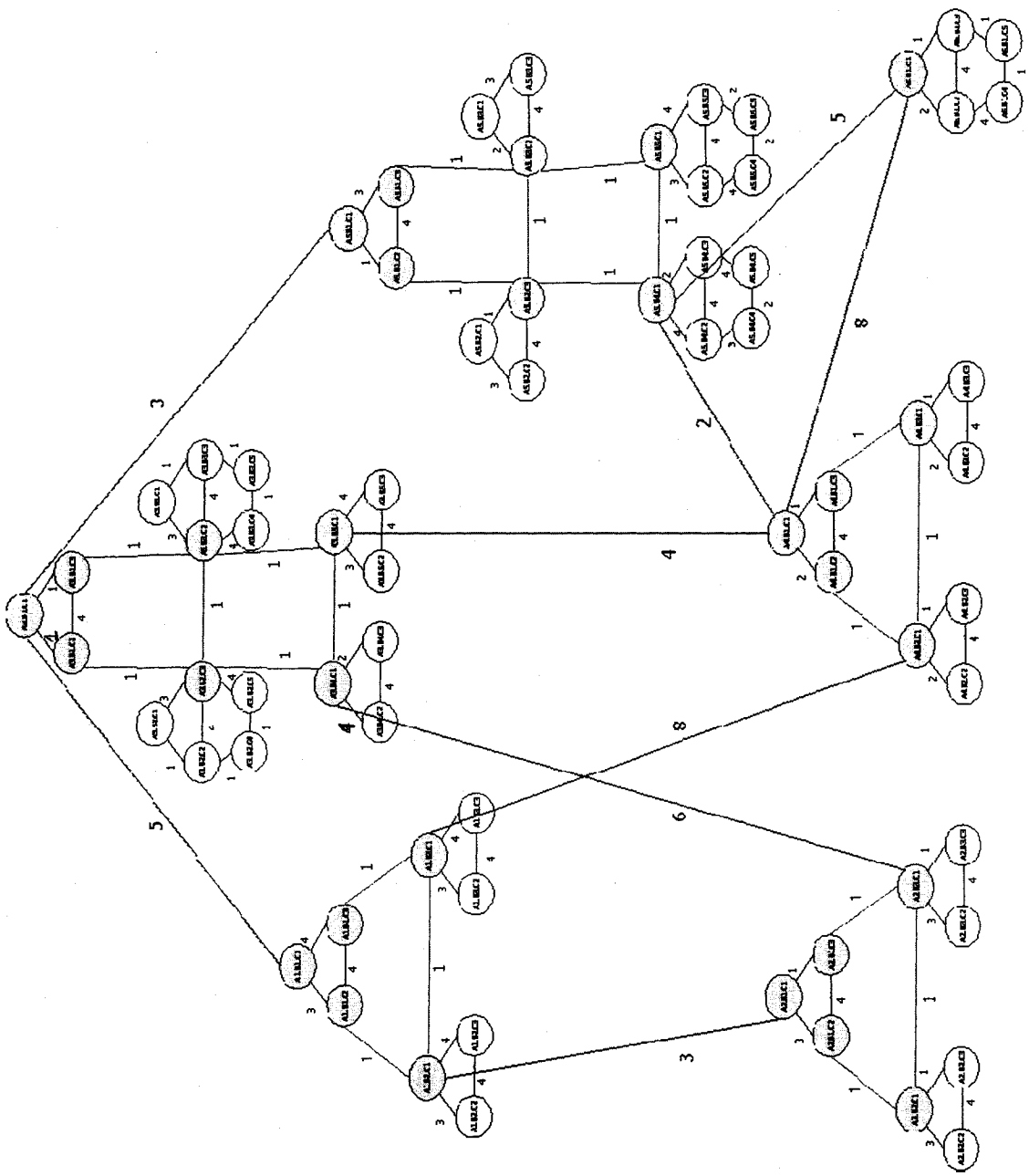


Figure 100 Seventy nodes network topology

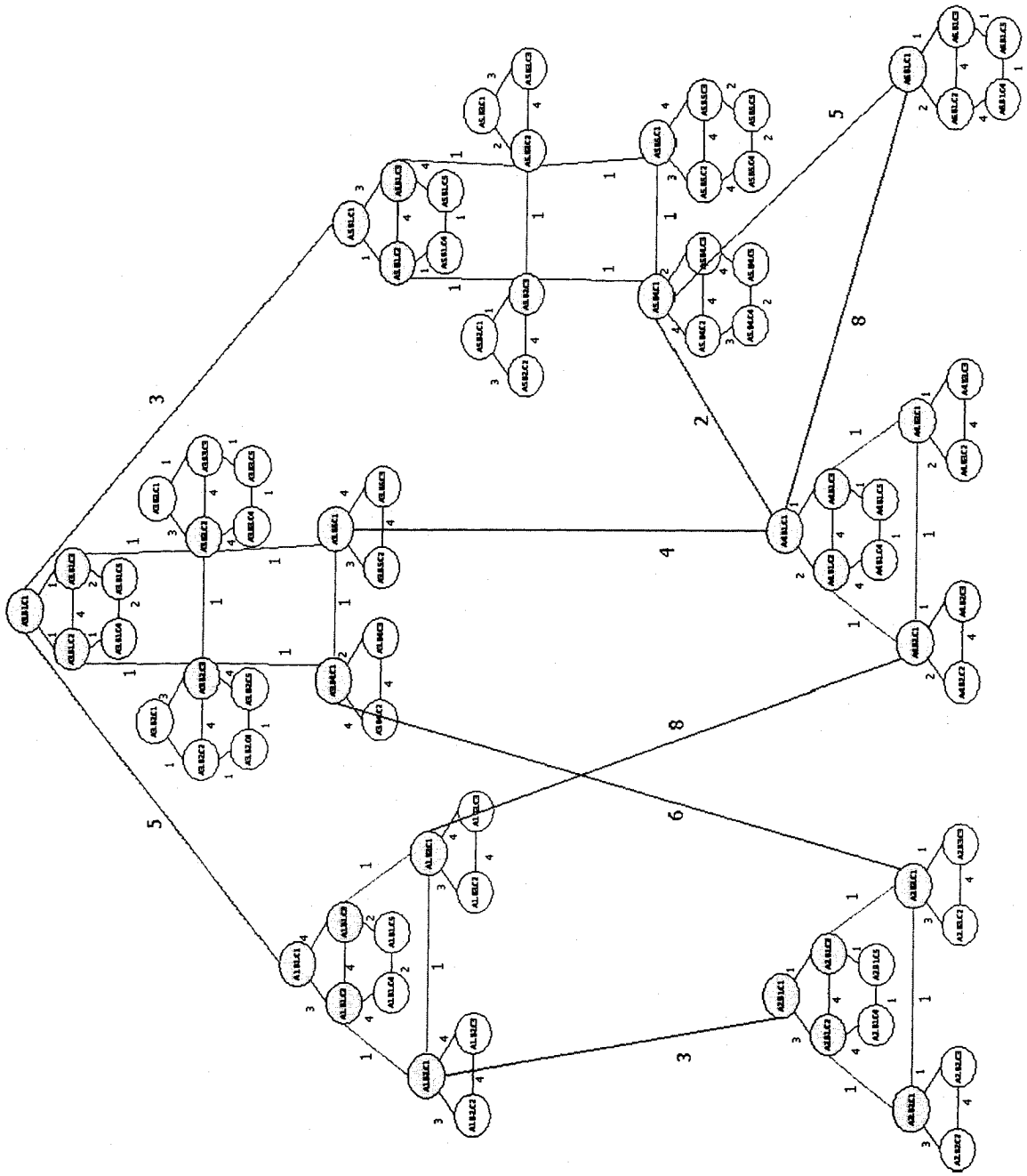


Figure 101 Eighty nodes network topology

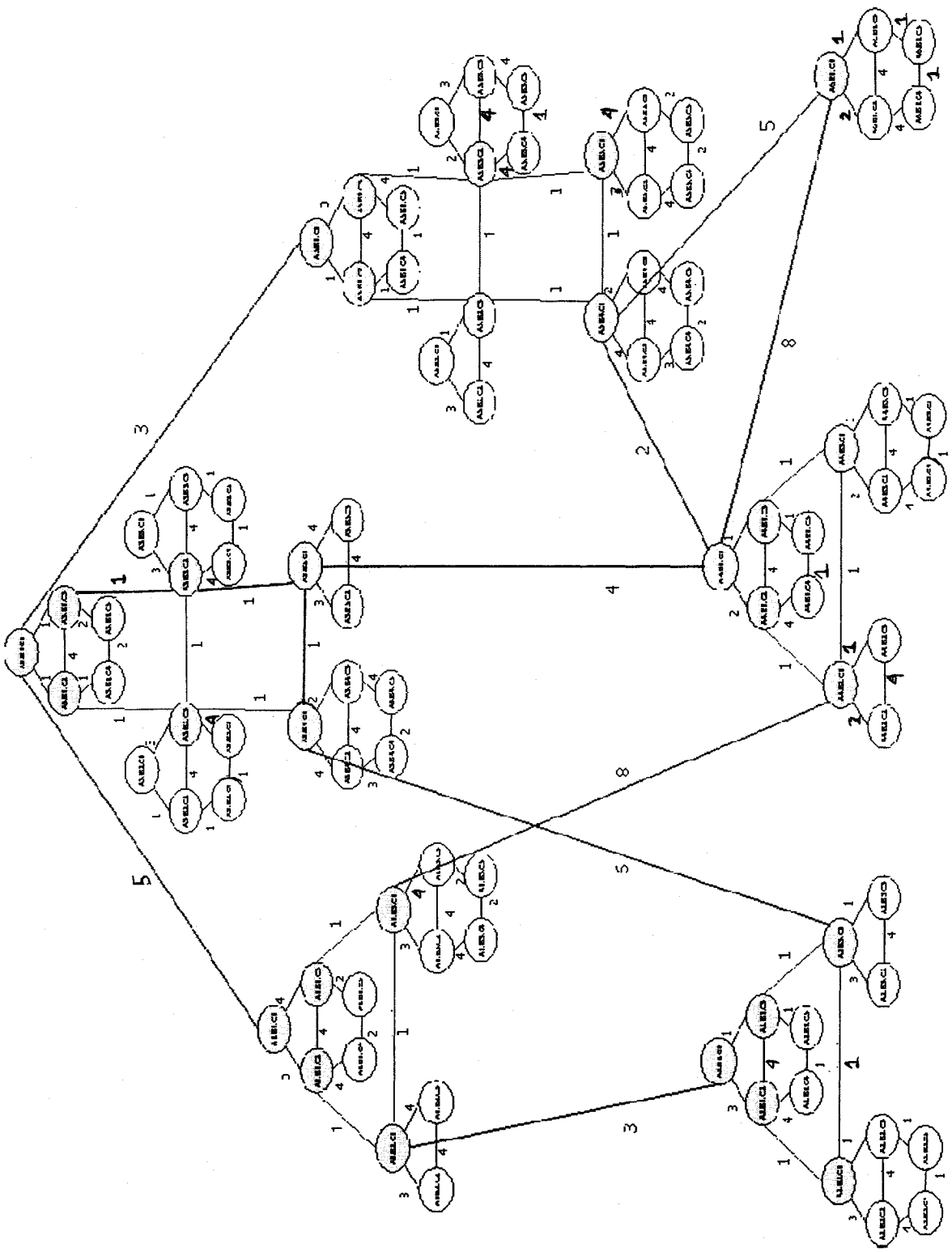


Figure 102 Ninety nodes network topology

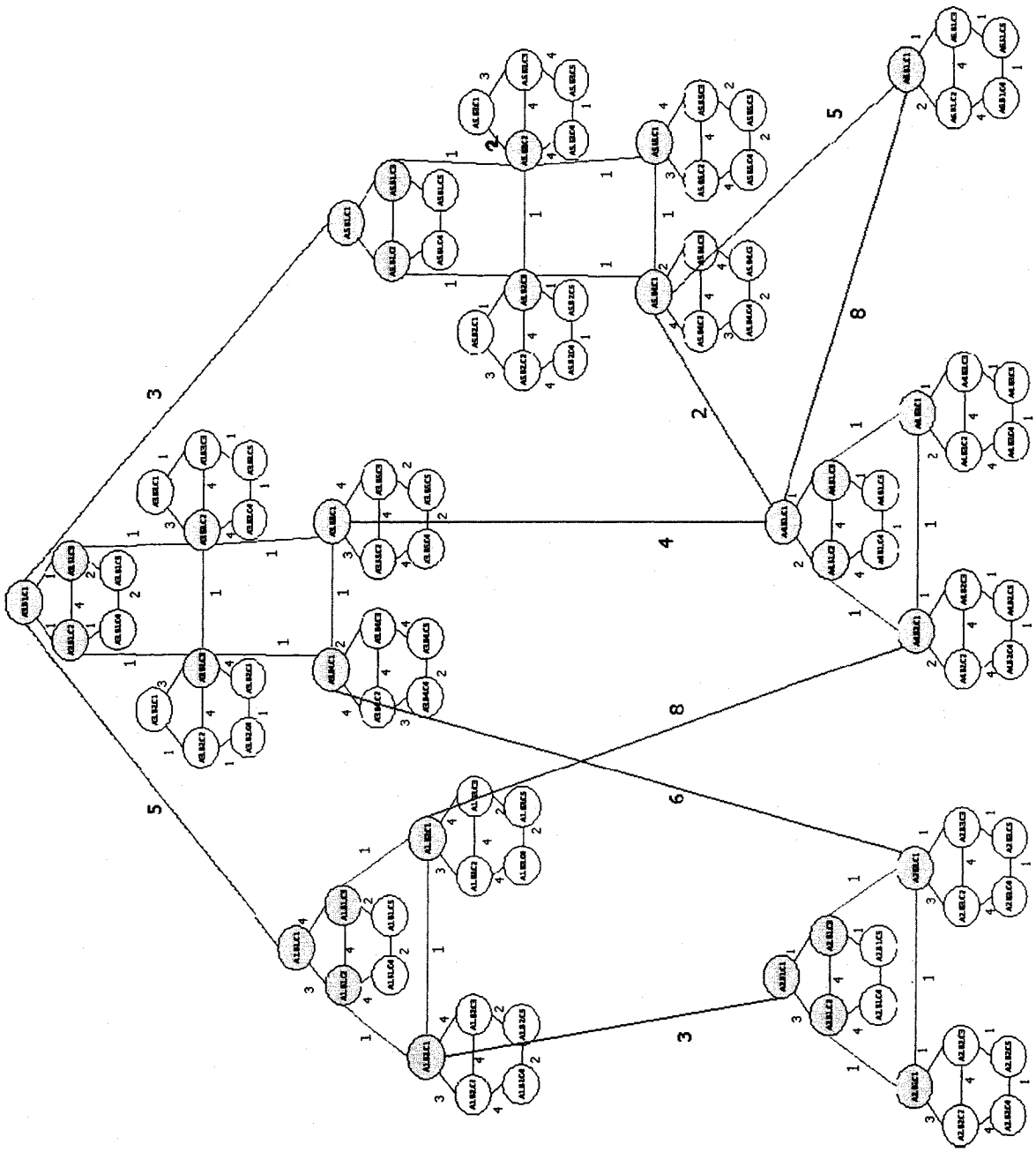


Figure 103 One hundred nodes network topology

7 Conclusions and future research

The demand of better QoS for stream-based applications (voice and multimedia) will keep growing. Hence, minimization of network delays and optimization of network throughput will play an important role in the next generation of newly developed switches and routers. Most existing switches and routers compute the shortest path algorithms in software, which is inefficient for real-time stream-based applications that are very sensitive to delays. The continuous need to improve performance and reduce network delays has led to the research of this thesis.

This thesis has introduced an innovative hardware architecture design and implementation for the OSPF protocol. The hardware architecture was divided into two modules (OSPF system and SPP module). The OSPF system responsibilities were to process the OSPF packets, respond to external OSPF routers and update the SPP module with network changes occurring. The SPP module is hardware architecture solution based on Dijkstra's algorithm, hence it computed the shortest path for the OSPF protocol. With the two architectures presented in this study, future requirements to reduce network delays can be achieved for two reasons. First, the OSPF network protocol convergence is greatly improved because the OSPF packets processing are handled at the hardware level. Second, with the SPP module, time computation for the shortest path is enhanced as shown in the experimental results section 6.2.2.

7.1 Concepts addressed in this thesis

Overall, this thesis has demonstrated the problem breakdown for implementing and architecting a hardware solution for a given routing protocol system (OSPF). The first break was identifying the major components of a protocol by approaching the problem from the following prospective:

- Network level, describing the interactions and operations of multiple nodes co-working together.

- Node level, describing the state machine needed for a node in order to operate and meet the functionality requirements at the network level.

- Port level, describing the packet format required in order to meet the protocol standard.

The second breakdown was towards the hardware architecture, where this study has demonstrated how to import the problem breakdowns described above into two major components of hardware architecture:

- Control unit, which performs the state machine of a routing protocol by acting and sending commands to the data path.

- Data path, which provides the control unit with the infrastructure (registers, memory, counter, muxs, demux, etc) that allows it to execute the state machine operations.

7.2 Contributions

The novel hardware solutions presented in this study, elevates the performance of the OSPF protocol from the following prospective:

- Synchronization time with neighboring routers.
- Time to update database after LSA(s) are received.
- Time to process and respond to LS requests.

Furthermore, the shortest path computation is an essential task for other networking protocols. Hence, the SPP module presented in this research can be integrated with different routing protocols (such as PNNI and IS-IS), which augments their performance when

computing the shortest path. Collectively, all these performance improvements assist in reducing the networks delays.

7.3 Future research

The implementation of a complete OSPF protocol for real-live networks is beyond the scope of this thesis. Work in this field should be continued in order to provide more features, enhance the interoperability and the compatibility of the OSPF system with other protocols. Below is a list of topics that require further study:

- Handling authentication, performing LS Checksum in the OSPF system. Furthermore, process the rest of the LSAs which are network LSA, ASBR summary LSA, AS external LSA and any newly added LSAs to the OSPF protocol.

- Investigate how to connect the OSPF system with higher layers protocol (layer 4 and layer 5). Hence, develop a device driver for the OSPF system with an operating system such as Linux.

- Integrate and deploy the OSPF system with other protocols such as IP or MPLS. Investigate what are the bottlenecks and enhancements needed for the OSPF system. Optimize the bottlenecks by parallelize tasks in hardware.

- Interconnect the OSPF system with other routing protocol such as RIP or BGP. Investigate what are the changes required for the OSPF system. Implement the changes and test the functionality.

- Using different hardware implementations of shortest path algorithms. Compare the performance of each implementation and outline the advantage and disadvantage of each implementation.

References

[1] J.Moy, "OSPF Anatomy of an Internet Routing Protocol", Addison-Wesley, February 1998.

[2] Russ White, Alvaro Retana, "IS-IS Deployment in IP Network", Addison-Wesley, February 2002.

[3] John Stewart, "BGP4: Inter-domain Routing in the Internet", Addison-Wesley, December 1998.

[4] http://www.ietf.org/, D.L. Mills, "Exterior Gateway Protocol Formal Specification", RFC 904, IETF, April 1984.

[5] http://www.atmforum.com/, ATM Forum Technical Committee, "Private Network-Network Interface Specification", March 1996.

[6] William Stallings, "Data Computer Communications", Prentice-Hall, Sixth edition, pg.582-598, 2000.

[7] Dinesh Verma, "Supporting Service Level Agreements on IP Networks", MacMillian Technical Publishing, 1999.

[8] Andrew S. Tanenbaum, "Computer Networks", Prentice-Hall, Third edition, pg.380-388, 1996.

[9] Th. Orphanoudakis, H.-C. Leligou, George Kornaros, Ch. Charopoulos, "Accurate scheduler implementation for shaping flows of variable length packets in high-speed networking applications", Recent Advances in Communications and Computer Science (WSEAS), pg. 95, 2003.

[10] R.Abielmona, S.Abielmona, M.Abou-Gabal, "The IP Architecuture of an IP over ATM processor", IEEE International conference, 2001.

[11] A. Dandalis, V. K. Prasanna. "Mapping homogeneous computations onto dynamically configurable coarse-grained architectures", IEEE Symposium on Field-Programming Custom Computing Machines, April 1998.

[12] Patrick Hung, Hossam Fahmy, Oskar Mencer, Michael J. Flynn, "Fast Division Algorithm with a Small Lookup Table", Asilomar Conference on Signals, Systems, and Computers, California, Nov. 1999.

[13] L. Huelsbergen, "A Representation for Dynamic Graphs in Reconfigurable Hardware and its Application to Fundamental Graph Algorithms", Eighth International ACM Symposium on Field-Programmable Gate Arrays Citation (FPGA'00), IEEE Computer Society, 2000.

[14] Oskar Mencer, Zhining Huang, Lorenz Huelsbergen, "HAGAR: Efficient Multi-Context Processors", FPL, 2002.

[15] Matti Tommiska, Jorma Skyttä "Dijkstra's Shortest Path Routing Algorithm in Reconfigurable Hardware", FPL 2001, LNCS 2147, pp. 653-657, 2001.

[16] Nasir Shaikh, Mohamed Khalil Hani, Teoh Giap Seng, "Design and Implementation of a Shortest Path Processor for Network Routing", 2nd world Engineering Congress (WEC2002) Sarawak, Malaysia, July 2002.

[17] Nasir Shaikh, Mohamed Khalil Hani, Teoh Giap Seng, "Implementation of Recurrent Neural Network Algorithm for Shortest Path Calculation in Network Routing", ISPAN 2002, IEEE Computer Society DL, 2002.

[18] M. Tsai, C. Kulkarni, C. Sauer, N. Shah, K. Keutzer, "A Benchmarking Methodology for Network Processors", 1st Workshop on Network Processors (NP-1), 8th Int. Symposium on High Performance Computing Architectures (HPCA-8), 2002.

[19] Robert Hass, Clark Jeffries, Lukas Kenel, Andreas Kind, Bernard Metzler, Roman Pletka, Marcel Waldvogel, Laurent Frelechoux, Patrick Droz. "Creating Advanced Functions on Network Processors: Experience and Perspectives", IEEE Network, 17(4), July 2003.

[20] http://www-2.cs.cmu.edu/~rajesh/, Rajesh Krishna Balan, Urs Hengartner, "Performance Analysis of the Intel IXP1200 Network Processor", Project Report for Graduate Computer Architecture Class, 15-740 ,Fall 00, CMU 2000.

[21] http://www.gigascale.org/, M. Gries, C. Kulkarni, C. Sauer, K. Keutzer. "Comparing Analytical Modeling with Simulation for Network Processors: A Case Study", Design, Automation and Test in Europe (DATE), Munich, Germany, March, 2003.

[22] Mohamed Abou-Gabal, Raymond Peterkin, Dan Ionescu, Cristian Lambiri, Voicu Groza, "A Shortest Path Processor for Traffic Engineering of VPN Services",6th International Conference on Technical Informatics, IEEE, 2004.

[23] Mohamed Abou-Gabal, Raymond Peterkin, Dan Ionescu, "IS-IS Protocol Hardware Implementation for VPN Solutions", WSEAS, 2004.

[24] Mohamed Abou-Gabal, Raymond Peterkin, Dan Ionescu, "An Architecture for a Hardware Implementation of the OSPF Protocol", 17th International Conference on Computer Applications in Industry and Engineering, ISCA, 2004.

[25] Dijkstra, E.: "A Note on Two Problems in Connexion with Graphs ", Numer. Math. Vol1, pg.269-271, 1959.

[26] Erwin Kreyszig, "Advanced Engineering Mathematics", Wiley, 8th Edition, 1999.

[27] Gen-Huey Chen, Biing-Feng Wang, Chi-Jen Lu, "On the Parallel Computation of the Algebraic Path Problem", IEEE transactions on parallel and distributed systems, October 1992.

[28] Ming-Yang Su, Gen-Huey Chen, Dyi-Rong Duh, "A Shortest-Path Routing Algorithm for Incomplete WK-Recursive Networks", IEEE transactions on parallel and distributed systems, April 1997.

[29] Shan Zhu, Garng M.Huang, "A New Parallel and Distributed Shortest Path Algorithm for Hierarchically Clustered Data Networks", IEEE transactions on parallel and distributed systems, September 1998.

[30] Ivan Stojemenovic, Xu Lin, "Loop-Free Hybird Single-Path/Flooding Routing Algorithms with Guaranteed Delviery for Wireless Networks", IEEE transactions on parallel and distributed systems, October 2001.

[31] Hristo N. Djidjev, Grammati E. Pantziou, Christos D. Zarliagis, "Improved Algorithms for Dynamic Shortest Paths", Algorithmica 28:4, pp. 367-389, 2000

[32] Jesper L. Traff, Christos D. Zaroliagis, "A Simple Parallel Algorithm for the Single-Source Shortest Path Problem on Planar Digraphs", Journal of Parallel and Distributed Computing 60:9, pp.1103-1124, 2000.

[33] http://portal.acm.org/, Sairam Subramanian, Roberto Tamassia, Jeffery Scott Vitter, "An Efficient Parallel Algorithm for Shortest Paths in Planar Layered Diagraphs", Technical Report, Duke University, 1993.

[34] Ravindra K.Ahuja, Thomas L. Magnanti, James B. Orlin, "Network Flows Theory, Algorithms and Applications", Prentice Hall, pg.122, 1993.

[35] Douglas E.Comer, "Internetworking with TCP/IP principles, protocols and architectures", Vol1, Prentice-Hall, 2000.
[36] Perlman, Radia, "A Comparison Between Two Routing Protocols: OSPF and IS-IS", IEEE Network Magazine, Vol.5, No.5, pp. 18 – 22, 1991.
[37] Sharon Oran, "Dissemination of Routing Information in Broadcast Networks: OSPF versus IS-IS", IEEE Network, Vol.15, No.1, pp. 64, 2001.
[38] Mijeong Yang, Jinho Hahm, Youngsun Kim, Sangha Kim, "Design and Implementation of the IS-IS Routing Protocol with Traffic Engineering", APCC 2003, the 9th Asia-Pacific Conference on Communications, Vol.3, pp. 1103, 2003.
[39] Cristian Lambiri, Dan Ionescu, Bogdan Ionescu, "Service Control in Connection Oriented Networks", Recent Advances in Communications and Computer Science (WSEAS), pp.266, 2003.
[40] Ashwin Sridharan, Roch Guerin, Christophe Diot, "Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks", INFOCOM 2003, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, Vol.3, pp. 1168, 2003.
[41] http://www.nortelnetworks.com/, "Passport IP Routing User Guide", Nortel Networks", 2003.
[42] http://www.cisco.com/, "OSPF Point-to-Multipoint Network with Separate Costs Per Neighbor", IOS Release 11.3, Cisco, 2004.
[43] http://www.juniper.com/, "Internet Software Configuration Guide: Routing and Routing Protocols", JUNOS Release 4.0, Juniper, 2004.
[44] M.Morris Mano, Charles R.Kime, "Logic and Computer Design Fundamentals", Prentice-Hall, pg.292-300, 1997.
[45] Ken Coffman, "Real World FPGA Design with Verilog", Prentice-Hall, 2000.
[46] Mark Gordon Arnold, "Verilog Digital Computer Design Algorithms to Hardware", Prentice-Hall, 1999.
[47] Douglas J. Smith, "HDL Chip Design", Doone Publications, 2000.