A New Hardware Architecture for Parallel Shortest Path Searching Processor Based-on FPGA Technology

Mohammed A Ali Al-Ebadi
University of Basrah
5 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Majid A Alwan
University of Basrah
13 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Available Online at www.ijecse.org ISSN- 2277-1956

A New Hardware Architecture for Parallel Shortest Path Searching Processor Based-on FPGA Technology

Jassim M. Abdul-Jabbar [1], Majid A. Alwan [2], Mohammed A. Ali Al-Ebadi [3]
[1] *Computer Engineering Department, University of Mosul, Mosul, Iraq.*
[2&3] *Computer Engineering Department, University of Basra, Basra, Iraq.*
[1] *drjssm@yahoo.com*, [2] *altimimee@yahoo.com*, [3] *mohammed.alebadi@yahoo.com*

Abstract: In this paper, a new FPGA-based parallel processor for shortest path searching for OSPF networks is designed and implemented. The processor design is based on parallel searching algorithm that overcomes the long time execution of the conventional Dijkstra algorithm which is used originally in OSPF network protocol. Multiple shortest links can be found simultaneously and the execution iterations of the processing phase are limited to $\mathrm{O(n-1)}$ instead of $\mathrm{O(n^2)}$ of Dijkstra algorithm. Depending on the FPGA chip resources, the processor is expanded to be able to process an OSPF area with 128 routers. High speed up factors of our proposal processor against the sequential Dijkstra execution times, within (76.77-103.45), are achieved.

Keywords –Dijkstra Algorithm, Parallel Computing, OSPF, FPGA.

I. INTRODUCTION

The dynamic routing protocols run specific algorithms to automatically setup and update their routing tables responsible to forward the incoming packets to their destinations. Open Shortest Path First (OSPF) routing protocol is an instance of a link state protocol based on hop-by-hop communication of routing information, specifically designed for intradomain routing in an IP network [1], [2]. It runs Dijkstra algorithm for finding the next router in the shortest path from the source to destination routers. Dijkstra algorithm one of the first algorithms created for shortest path finding, determines the shortest path from a single source vertex to every other vertices [3].

Because of the sequential behavior of the original Dijkstra algorithm, it needs $O(n^2)$ to find the results, where n is the number of routers in the OSPF area [1]. In this paper the word 'node' denotes a router and the word 'network' denotes an area of OSPF network. So, when the network becomes large, the computation time of shortest paths will increase exponentially. The long computation time of the routing table setup makes the network performance slow and the traffic speed will be affected and many problems will appear, such as queue size problem and network congestion [2]. In the literature, the researches attempted to solve this problem and speedup the computation time of the routing table setup, generally by two ways. The first way uses the parallel processing systems. The routing table is partitioned into many regions; each region contains the information of a part of the network. Sub-routing table of each region can be calculated independently by its own processor or by its own thread in multi-thread systems [4]-[6]. By this way, not all routing table need to be updated when only some regions have the change. So, the time of computing small size sub-tables to handle the changes will be shorter than when the whole routing table is computed from the original complete network. The second way is to design parallel hardware architectures capable to process the routing information in a parallel fashion in order to produce the routing tables, such as reconfigurable processors and field programmable gate array FPGA technology [7- 12]. For such concepts, parallel shortest path searching algorithms are proposed and implemented in application specific integrated circuits ASIC's.

The goal of this paper is to design and implement high speed hardware processor for shortest path searching for OSPF networks. The proposed architecture is based on the parallel shortest path searching algorithm and targeted to Xilinx Vertix-7 (XC7V2000T) FPGA chip. Virtex-7 FPGA is a rich logic elements and high speed device [13], which can offer a high resources capacity and a high degree of parallelism. Such properties are usually required to handle large scale network topologies. VHDL codes and synthesis, functional and post-route simulations are accomplished using Xilinx ISE suite.

The rest of this paper is organized as follows: an illustration of conventional Dijkstra algorithm and the parallel shortest path searching algorithm is presented in section II. Section III explains the proposed hardware implementation of parallel shortest path searching (PSPS) steps of operation. In section IV, the design of an FPGA-based PSPS processor and its units are explained. Section V shows detail statistics of FPGA resources occupation and performance evaluation. Simulation results of 8-node PSPS processor is presented in section VI. Finally, the conclusion of this paper is given in section VII.

II. PRELIMINARIES

Dijkstra algorithm is one of shortest path searching algorithms [6]. It is used to find the shortest path or minimum cost path from a single source node to the destination node(s) in a graph [3]. The destination node may be a single node or multiple nodes. By running Dijkstra algorithm in all nodes, OSPF protocol computes all shortest path from the source node to the rest nodes in the network [1]. The OSPF network can be considered as a directed graph consists of n nodes and m links that connect the nodes. Given that we have a graph of n nodes (or vertices) and m links (or edges) with each link has a non-negative cost (or weight). The input of the algorithm is a weighted directed graph $G(V,E)$, where V is a set of nodes $\{v_1, v_2, v_3, \ldots, v_n\}$ and E is a set of links $\{e_1, e_2, e_3, \ldots, e_m\}$. The cost of each links is set by a weight function $w: E \rightarrow [0, \infty)$. The cost of link connecting two nodes, $w(u,v)$ where $u, v \in V$, represents the distance of moving from u node to v node in the graph. The shortest path is the minimum cost path from the source node to the destination node along all available paths. If $s \in V$ is the source node and $v \in V$ is the destination node, $d(v)$ is the shortest path from s to v. Dijkstra algorithm is described in the following steps [7].

1. Set S to empty, where S is a set of nodes whose shortest paths from the source node s have already been determined.

2. Add the source node s to S and $d(s) = 0$. If there is a link from s to v, $d(v) = w(s,v)$, for all other nodes $d(v) = \infty$.

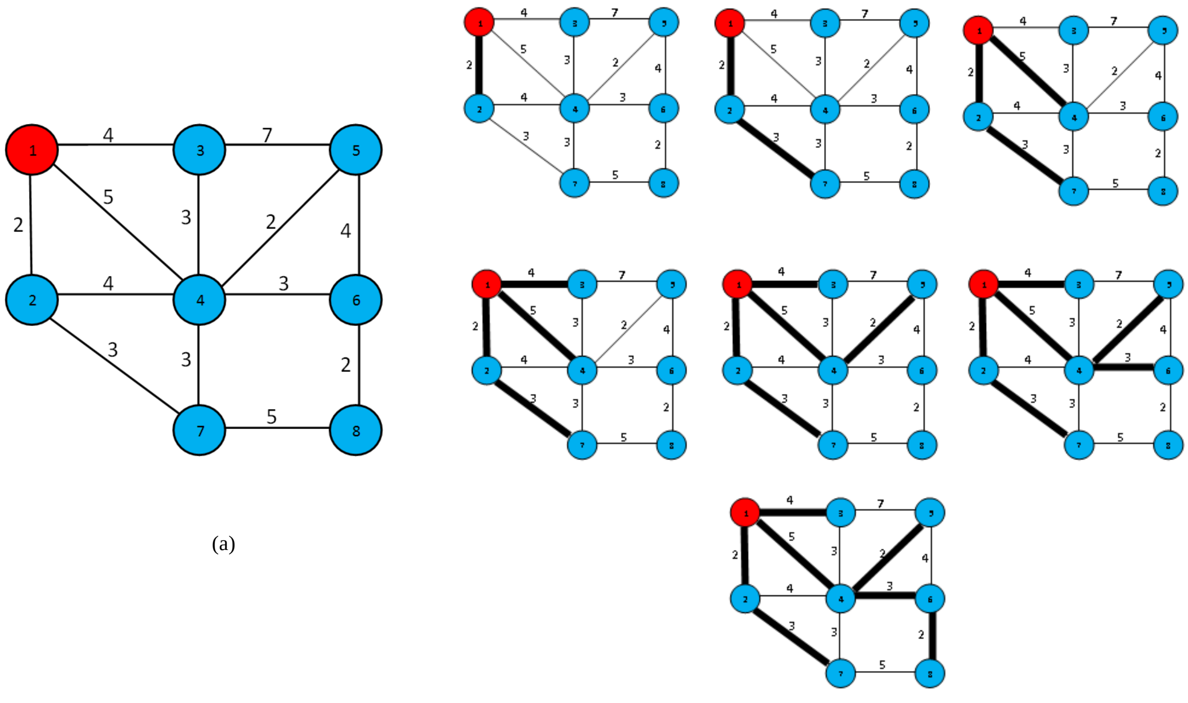
3. Add a node u to S, where $d(u)$ is the smallest in $V - S$. If $S = V$, complete the task.

4. If there is a link from u to $v \in V - S$, $d(v) = min\,\{d(v), d(u) + w(u,v)\}$. Then go to step 3.

To demonstrate the algorithm, Fig. 1(a) shows an example of a network topology of eight nodes and 13 links. A cost is assigned to each link. Assuming that the source node is node 1, so shortest paths from node 1 to the rest nodes will be calculated by Dijkstra algorithm. Firstly, $d(1) = 0$, $d(2) = 2$, $d(3) = 4$, $d(4) = 5$, and all other link costs set to infinity. Add node 2 to S and update $d(4) = 5$ and $d(7) = 5$. Add node 7 to S and update $d(4) = 5$, $d(8) = 10$. Add node 4 to S, and update $d(3) = 4$, $d(5) = 7$, $d(6) = 8$. Add node 3 to S and update $d(5) = 7$. Add node 5 to S and update $d(6) = 8$. Add node 6 to S and update $d(8) = 10$. Finally, add node 8 to S and the algorithm is completed. The algorithm steps are shown in Fig. 1(b) with bold lines of the shortest paths found.

It is clear that just one node is reached at a time because the sequential execution of the Dijkstra algorithm in conventional microprocessor-based systems. Therefore, the computation time for a n-nodes graph is $O(n^2)$. Also, Dijkstra algorithm can find multiple equal cost paths (MECP) that used to balance the network load by dividing the load into multiple paths which have equal cost, evenly. If this option is active, in our example, node 8 can be reached by two equal cost paths which are (1 2 7 8) and (1 4 6 8) both with cost 10.

A parallel shortest path searching algorithm based on original Dijkstra algorithm was proposed in [7]. Simultaneous multi-path search from the source node depending on cost determines the shortest paths to each node in the graph. A function $\gamma(u,v)$ is introduces to calculate how far it is from the current position to v in a link (u,v). The following steps describe the algorithm:



(a)

(b)

Figure 1. (a) Network topology, (b) Steps of Dijkstra algorithm

1. Set S to empty. Add the source node s to S and $d(s) = 0$. For all other nodes $d(v) = \infty$, and for all links $\gamma(u, v) = w(u, v)$.

2. Set L to empty, where L is a set of links in which current position is proceeds. If there is a link from s to v, add it to L.

3. Take $\gamma(p, q)$ steps for all $l \in L$, where $\gamma(p, q)$ is the minimum value for $(p, q) \in L$. Add q to S and $d(q) = d(p) + w(p, q)$. Set $\gamma(u, v) = \gamma(u, v) - \gamma(p, q)$ for all $(u, v) \in L$. If $S = V$, complete the task.

4. Delete (v, q) from L if there is a link from $v \in S$ to q. Add (q, v) to L if there is a link from q to $v \in V - S$. Then go back to step 3.

The same topology in Fig. 1(a) can be searched by the parallel algorithm. Fig. (2) shows the algorithm steps. To trace the algorithm steps, $w(u, v) - \gamma(u, v)/w(u, v)$ is presented with each link in the Fig.(3). Node 1 is the source node. First, S and L are empty. Add node 1 to S. By step 2, links 1-2, 1-3, and 1-4 are added to L and $\gamma(1, 2) = 2$, $\gamma(1,3) = 4$, and $\gamma(1,4) = 5$. According to step 3 of the algorithm, the minimum value is 2, so link 1-2 reaches node 2, add node 2 to S with $d(2) = 2$, and Links 1-3 and 1-4 proceed 2 steps that update $\gamma(1, 3) = 2$, $\gamma(1,4) = 3$. By step 4, link 1-2 will be removed from L and links 2-4 and 2-7 are added to L with $\gamma(2,4) = 4$ and $\gamma(2,7) = 3$. In the second iteration, the minimum $\gamma(p, q)$ belong $\gamma(1, 3) = 2$, $\gamma(1,4) = 3$, $\gamma(2,4) = 4$ and $\gamma(2,7) = 3$ is 2. Therefore, node 3 will be reached and added to S with $d(3) = 4$, and also link 1-3 will be removed from L. links 1-4, 2-4 and 2-7 will proceed 2 steps. Add links 3-4 and 3-5 to L and set $\gamma(3, 4) = 3$ and $\gamma(3,5) = 7$. Now, $\gamma(1, 4) = 1$, $\gamma(2,4) = 2$ $\gamma(2,7) = 1$, $\gamma(3,5) = 7$, and $\gamma(3,4) = 3$. The minimum is 1, so all links belong to L will proceed one step. According, nodes 4 and 7 are reached simultaneously from nodes 1 and 2 respectively with $d(4) = 5$ and $d(7) = 5$ and added to S. Links 1-4, 2-7, 2-4 and 3-4 is now removed from L because nodes 4 and 7 have been reached. During the third iteration, links 4-5, 4-6, and 7-8 are added to L. Update $\gamma(p, q)$ values to be $\gamma(3,5) = 6$, $\gamma(4,5) = 2$ $\gamma(4,6) = 3$, and $\gamma(7,8) = 5$. The minimum value is 2, so node 5 is reached from node 4 and added to S with $d(5) = 7$. Other links belong to L proceed 2 steps to update $\gamma(3,5) = 4$, $\gamma(4,6) = 1$, and $\gamma(7,8) = 3$. Because of node 5 is reached, links 3-5 and 4-5 will be removed from L while link 5-6 will be added with $\gamma(5,6) = 4$. Now, the minimum value is 1, and all links belong to L will proceed one step, resulting in node 6 being reached from node 4 and added to S with $d(6) = 8$, and updating $\gamma(5,6) = 3$ and $\gamma(7,8) = 2$. Remove links 4-6 and 5-6 from L and add link 6-8 to L with $\gamma(6,8) = 2$. Finally, the minimum value is 2 and node 8 will be reached simultaneously from both node 7 and node 6 with $d(8) = 10$. Add node 8 to S and complete the task because $V = S$ now.

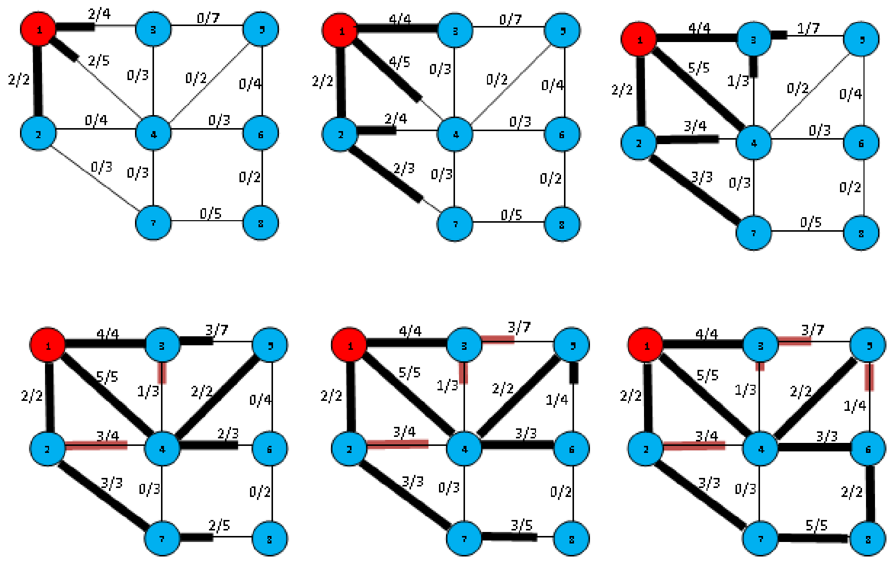


Figure 2. Steps of parallel shortest path searching algorithm

The computing time of sequential Dijkstra algorithm for n nodes network is $O(n^2)$ while for the parallel shortest path searching algorithm is $O(n)$ [7].

III. IMPLEMENTATION OF PARALLEL SHORTEST PATH SEARCHING (PSPS) ALGORITHM

The network topology will be presented as an adjacency square matrix called cost matrix. Each element in cost matrix represents the cost of travelling from node i to node j, $w(i,j)$. In our design, the PSPS algorithm is executed by the following steps:

1. A flag n-bit vector is associated with the cost matrix. The bit position of the flag vector works with the row and column of the cost matrix that have a corresponding index numbers (*i.e.*, flag bit 1 with row 1 and column1). In other words, each bit of the flag vector is associated with a single node and its links to the other nodes. Initially, flag bit position of the row of source node is set to 1, and other flag bits to 0. Flag bit status will decide if the corresponding row will be searched for minimum value in the next steps.

2. Set all elements of the column of cost matrix that has flag bit equals 1 to infinity.

3. Find the minimum value among the elements of rows whose flag bits are 1.

4. Subtract the minimum value that found in the step 3 from all values of rows whose flag bits are 1.

5. Zero elements of the resulted cost matrix have minimum costs from source node to their destinations.

6. For columns that have zero elements, update their flag bits to 1.

7. If all flag bits are 1's, all shortest paths are completely found and the matrix will be totally infinity next step, else, go to 2.

IJECSE
International Journal of Electronics and Computer Science Engineering
WWW.IJECSE.ORG

The PSPS algorithm has two main important features. First, multiple paths can be searched simultaneously and independent on the previous results. Second, multiple-equal-cost-paths (MECP) from source nodes to the same destination can be extracted from the results.

IV. HARDWARE ARCHITECTURE OF THE FPGA-BASED PSPS PROCESSOR

A new design of PSPS algorithm based on FPGA technology is proposed here. Fig (3) shows the block diagram of hardware architecture of PSPS of a network consists of n nodes. The adjacency matrix, or cost matrix, of the network topology is $n \times n$ elements, each element is 8-bits representing the cost of the connection link between upstream node to downstream node. The PSPS processor is mainly consisting of two blocks, latch block and processing block. The cost matrix will enter the shortest path processor row by row as 64-bit per clock. Internally, the latch block will latch the cost matrix to be provided then to the processing block. Latch block contains three units, address decoder, data latch, and flag generator units. When the complete cost matrix is latched, it will enter completely in parallel to the processing block. The processing block consists of seven main units. These are cost-matrix and flag multiplexers, set infinity columns, comparator bank, subtractors matrix, flag update, shift register, and routing table units. In the followings, each of these components are described.

1) Latch block:

The latch block consists of three units as mentioned above. The cost matrix will enter the data latch (DL) unit from the first row though the last row as 64-bit per clock. Simultaneously, the address will enter the address decoder (AD) unit in order to generate latch signal for the corresponding 64 flipflops that used to latch the data. Latch signals are used as clock enable signals of the flipflops. Because each latch signal generated by AD unit will be at logic 1 for just one clock period, the flipflops output will remain unchanged next clock periods. By this way, flipflops will acts as gated latches and will prevent the timing problem appears when using latches directly in FPGA-based design [14]. For n nodes network topology with 8-bit link cost values, the circuit requires $(n \times n \times 8)/64$ clocks for latching the complete cost matrix.

Flag generator (FG) unit is used to generate n -bit flag vector, which has its LSB bit at logic 1 and other bits at logic 0. It is a simple circuit, with a single input that is always at logic 1 and can be expanded to generate other $n-1$ inverted bits, to achieve the initial n-bit flag vector. Logic 1 LSB bit of the flag vector means that the first row of the cost matrix is for the source node of the shortest paths calculation process.

2) Processing block

The inputs of the processing block are the cost matrix and the flag vector, provided by latch block. If the i^{th} bit of the flag vector is at logic 1, i^{th} node's links will be used in the shortest path calculation, else, i^{th} node's links will not participate. Initially, just the flag bit of the source node will be at logic 1 and the rest bits will be 0's. During the shortest path calculation, the flag vector bits will be changed to 1's, and all bits will take the value 1 at the end of shortest path calculation. This means that all nodes and their links are tested for shortest paths.

As mentioned before, processing block is consisted mainly of seven units. Following is the description of each unit.

a) Cost-matrix and flag multiplexers (MUX unit):

The first unit of the processing block of the shortest path processor is the MUX unit. It consists of two 2-to-1multiplexers work in parallel. The first one is used to pass either the initial cost matrix provided from latch block or the resulted cost matrix provided from the intermediate unit of the processing block. The second multiplexer is used in the same way but for passing either the initial flag vector or the updated flag vector coming from the final processing unit. Two additional inputs supplied to the two multiplexers, selector signal for selecting one of the two multiplexer inputs and the enable signal for enabling the two multiplexers. The selector is generated by ORing all bits of the updated flag vector which is initiated to zero value. So, the first iteration of the processing block will supplied by the initial cost matrix and flag vector. After the first iteration is completed, at least two of the updated

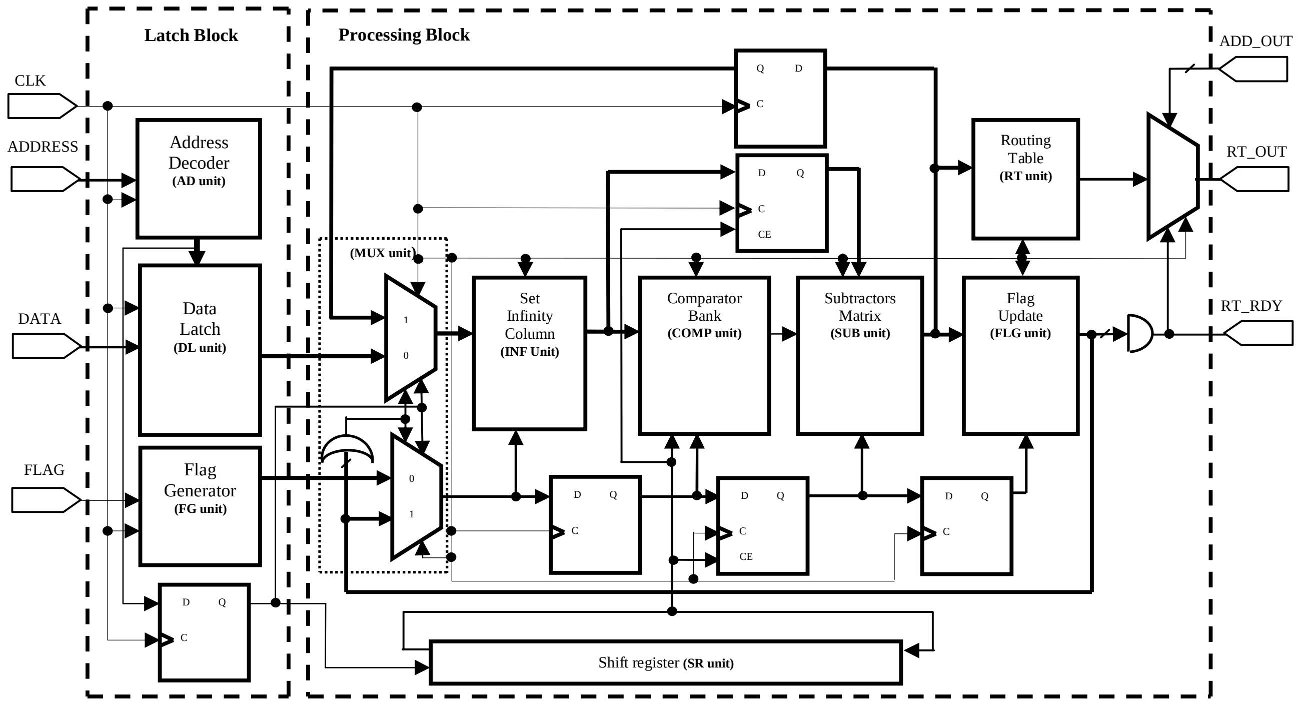


Figure 3. Block diagram of the proposed FPGA-based PSPS processor

flag vector bits will be changed to 1 and the output of the OR gate will also be changed to 1. Therefore, the second and the next iterations will be supplied by the updated cost matrix and flag vector. The enable signal is driven from the last latch signal with one clock delay to insure that the initial cost matrix and flag vector reach the multiplexers inputs.

b) Set-infinity-column unit (INF unit):

The operation of INF unit is to set all elements of each column of the cost matrix whose associated flag bit is at logic 1, to infinity. The inputs of INF unit are the cost matrix and the flag vector coming from MUX unit, and the output of INF unit is the updated cost matrix with some columns with totally infinity value elements. If the (i,j) element in the cost matrix has an infinity value, this represents no connection link is available between i^{th}node and j^{th} node. Setting some columns to infinity values by INF unit is to prevent update their element values (links costs) in the next processing units.

c) Comparator bank unit (COMP unit):

Now, the minimum cost among the desired links can be found using COMP unit. This unit is basically binary tree comparators. It is consist of $(\log_2 n \times n)$ stages of comparators, for n nodes network. The input to this unit is the complete cost matrix updated by INF unit and the flag vector. In the first stage, all rows of the cost matrix will be searched independently in parallel, for their local minimum costs. When the associated flag bit of the row is set, its local minimum cost will be pass to the next stages, else an infinity value will be placed instead. The rest stages of the comparator bank will search the global minimum among the local minimums of the first stage. Finally, the output of this unit is the minimum link cost of the participated links.

d) Subtractor matrix unit (SUB unit):

The subtractor matrix unit is a two-dimensional $n \times n$ array of simple subtractors. The updated cost matrix produced by INF unit, the minimum cost produced by COMP unit, and the flag vector are the inputs of this unit. The subtractor of a specific position in the subtractors array will subtract the minimum cost from cost value at the same position of the cost matrix. This is done only for the rows of the cost matrix which have flag bits equal to 1, other rows will stay unchanged. All subtractors will operate simultaneously, and as a result, the updated cost matrix will have zero values. The output of SUB unit is the updated cost matrix that will be used in the next iteration of the processing block as the second input of the cost matrix multiplexer of MUX unit.

e) Flag Update unit (FLG unit):

FLG unit receives the updated cost matrix produced by SUB unit and the flag vector. The function of this unit is that, when a zero value is found in each column of the cost matrix, the corresponding flag bit will be set to 1. All columns will be searched in parallel and all flag vector affected bits will be updated at the same time. The output of the FLG unit is the updated flag vector that will be used in the next iteration of the processing block as the second input of the flag multiplexer of the MUX unit.

f) Shift register unit (SR unit):

SR unit is used for synchronization purpose. In order to prevent the race conditions that my cause wrong result of SUB unit, the output of the INF unit and the flag vector as well as the output of the COMP unit should be arrived at the same clock period to the SUB unit. This can be achieved by using a shift register rotating its contents infinitely when its enable signal is active.

The SR unit has initial binary value equal to (00 …. 00100). It is arrange so that third LSB bit value will reach the MSB bit at the same time of arriving the result of COMP unit to its output flipflops. At this time, MSB bit will enable the flipflops of the last stage of COMP unit and the other synchronization flipflops, and the inputs of SUB unit will arrive at the same time. The length of the shift register should be $(4 + \log_2 n \times n)$ bits and it is enabled to start rotating by the same signal of MUX unit enable.

g) Routing Table unit (RT unit):

The RT unit will have the final information of the shortest paths of the network routs. It consists of $n \times n$ bit array. When (i, j) bit is set, this means that the link from i node to j node is participated in the shortest path because of its minimum cost. The input of this unit is the cost matrix resulted from SUB unit. The zero values of cost matrix elements will set the corresponding bits positions in the bit array of RT unit. At the end of the shortest path computations, the RT unit will have the shortest path routing table.

Finally, the routing table can be read out from the RT unit using a multiplexer. The multiplexer has $(n \times n)/64$ data inputs and one data output, each is of 64-bit wide. Also, it has an enable signal generated from ANDing all bits of the updated flag vector that produced by FLG unit. This signal indicates, when it becomes 1, that the RT unit has the complete routing table information.

V. FPGA-BASED DESIGN STATISTICS AND PERFORMANCE EVALUATION

The proposed hardware architecture of PSPS processor is designed in VHDL and Xilinx Vertix-7 (XC7V2000T) FPGA chip is selected as the target device. Various network scales are considered and the corresponding processors are designed and tested against many cases of network topologies. The FPGA chip performance and area utilization can be reported using Xilinx ISE v13.2 design suite [15].

First, the performance of the proposed PSPS processor will be explained. The system is driven by a single clock source for synchronous elements. The number of clocks consumed by the processor is dependent on the network size, *i.e.*, number of nodes. Table 1 shows the clock cycles required for each operation of PSPS processor, where n is the number of nodes of the network.

From Table 1 above, the following equation calculates the number of clocks required for calculating the shortest paths from a single node to all other node in an n nodes network:

$$Number\ of\ clocks = \frac{n^2}{8} + [(4 + \log_2 (n^2) \times (n-1)] + \frac{n^2}{64} \qquad …….. (1)$$

Table -1 Clock cycles required for operations.

Operation	Clocks
Loading and latching cost matrix	$(n^2 \times 8)/64$
MUX unit	1
INF unit	1
COMP unit	$log_2 n \times n$
SUB unit	1
FLG unit and RT unit	1
Extract the routing table	$n^2/64$

Eq. (1) computes the maximum clock cycles required for shortest paths searching, but actually the clock cycles is mostly less than that number resulted from such equation because the processing block may not require all $n-1$ iterations to finish its operation. It depends on the network size and topology, and also on the variation of link cost values. The parallel searching of the minimum cost have the high impact to reduce the number of processing iteration of the processing block because multiple rows can be searched simultaneously.

Table 2 shows the FPGA chip resources utilization and the minimum clock period for different processors. The minimum clock period was specified by the timing constraints tool of Xilinx ISE software.

Table -2 XC7V2000T FPGA chip utilization and clock period. FF stands for flipflop and LUT for look-up-table.

Network size (n)	**Flipflops FFs**	**LUTs**	**Area Utilization ratio**		**Min. Clock period (ns)**	**Max. Number of clocks**
			FF	**LUT**		
8	3758	2531	1%	1%	7.60	79
16	19051	12614	1%	1%	7.80	216
32	75471	51493	3%	4%	9.11	578
64	300440	354272	12%	29%	9.95	1584
128	1044499	797389	42%	65%	12.6	4590

Table 3 shows the comparison of the execution time between the proposed FPGA-based PSPS processor and the sequential software Dijkstra algorithm for n nodes networks. For the sake of testing PSPS processors, cost matrices for different size n nodes networks are generated randomly with various considerations of connection density and link cost values. The software version of Dijkstra algorithm is executed on a standard PC with 2 GHz Intel i7 core processor and 8 GB RAM. Speed up factors are also computed in this table. The hard cases of network connectivity are selected for each network sizes. Those cases are reached by forcing various cost values and connection densities to be in the range of 65% to 95%. Also, FPGA-based PSPS processor is considered consuming all clock cycles calculated from Eq.(1).

Table -3 Software Dijkstra algorithm vs. FPGA-based PSPS execution time.

Network size (n)	**Software Dijkstra Min. execution time (sec)**	**FPGA PSPS Max. execution time (sec)**	**Speed up factor**
8	6.0038 e-5	600.4 e-9	99.99
16	1.3393 e-4	1684.8 e-9	79.49
32	4.0541 e-4	5265.58 e-9	76.99
64	0.0014	15760.8 e-9	88.82
128	0.0035	57834 e-9	60.51

IJECSE
International Journal of Electronics and Computer Science Engineering
WWW.IJECSE.ORG

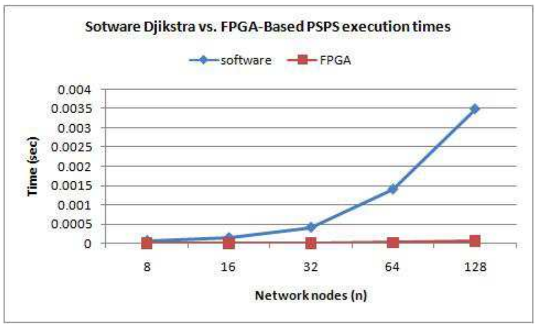
- *Early Termination:*

As mentioned before in this paper, the proposed FPGA-based PSPS processor is designed to terminate the processing block iterations when all updated flag vector bits become 1's. Because of the parallel fashion of the design, more than one flag bit may be set in the same iteration, so it does not need to complete all $n-1$ iterations. This feature provides the proposed design with an early termination, highlighting the execution time reduction while determining shortest paths. Table 4 presents the practical results of executing hard cases experiments for each network size using both Dijkstra algorithm and PSPS processor. It also shows that the maximum clock cycles required for 32, 64, and 128 nodes processors are less than those computed from Eq. (1).

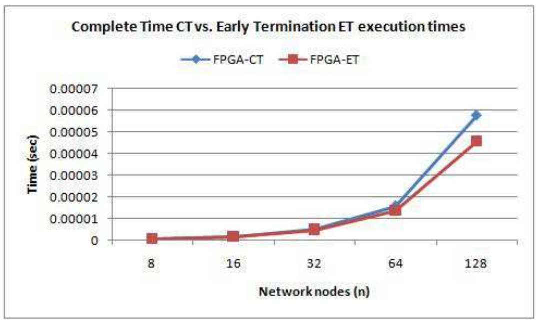
Table - 4 Software Dijkstra algorithm vs. FPGA PSPS processor execution time (early termination)

Network size (n)	**Software Dijkstra Min. execution time (sec)**	**FPGA-based PSPS**		**Speed up factor**
		Iterations	**Execution time (sec)**	
8	6.0038 e-5	7	600.4 e-9	99.99
16	1.3393 e-4	15	1684.8 e-9	79.49
32	4.0541 e-4	27	4755.4 e-9	85.25
64	0.0014	49	13532 e-9	103.45
128	0.0035	73	45586.8 e-9	76.77

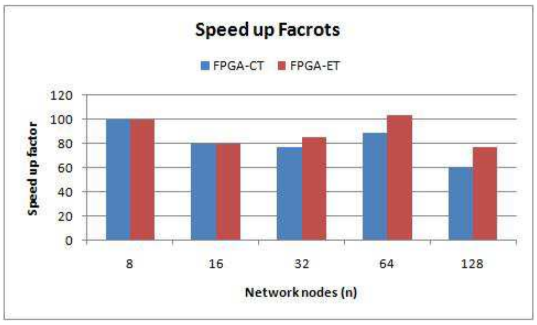
Figure 4 (a) illustrates the difference between software and hardware execution times for various network sizes as shown in Table 3. Figure 4 (b) shows the complete time (CT) and early termination (ET) execution times of the PSPS processor. Figure 4 (c) shows the speed up factors between the conventional software Dijkstra algorithm and the FPGA-based PSPS processor for the two cases, complete time (CT) and early termination (ET).



(a)



(b)



(c)

Figure 4. (a) Software Dijkstra algorithm vs. FBGA-based PSPS processor execution times.
(b) FPGA-based complete time (CT) vs. early termination (ET) execution times.
(c) FPGA-based PSPS processor speed up factors.

IJECSE
International Journal of Electronics and Computer Science Engineering
WWW.IJECSE.ORG

VI. ISIM SIMULATION RESULTS

FPGA-based PSPS processors for n=8, 16, 32, 64, and 128, are designed in VHDL codes according to the proposed architecture, and are targeted to FPGA chip, Xilinx Vertix-7 (XC7V2000T), and tested with various network topologies with different complexities by cost values and the connection density. As an example, Fig. (5) shows the post place and route simulation waveforms of the main input/output and other signals of the 8 node FBGA-based PSPS processor. The Post-Route simulation is accomplished by Xilinx ISim simulator [16]. The input cost matrix belongs to the network topology presented in Fig (1), and node 1 is chosen as the source node. Note that the link costs are represented here in hexadecimal numbers, and FF_h link cost represents the infinity. The number of clocks and the number of iterations consumed by FPGA-based PSPS processor are clear to be noted. The loading and latching cost matrix takes 60.8 ns (8 clocks). The computation process of the processing block takes six iterations to complete the algorithm and its elapsed time is 516.8 ns, *i.e.*, 68 clocks. It can be noticed that the flag vector is updated after each iteration as [03, 07, 4F, 5F, 7F, FF]. One additional clock cycle is needed to extract the routing table information, and the total loading-computing-extracting time will be 524.4 ns (69 clocks).

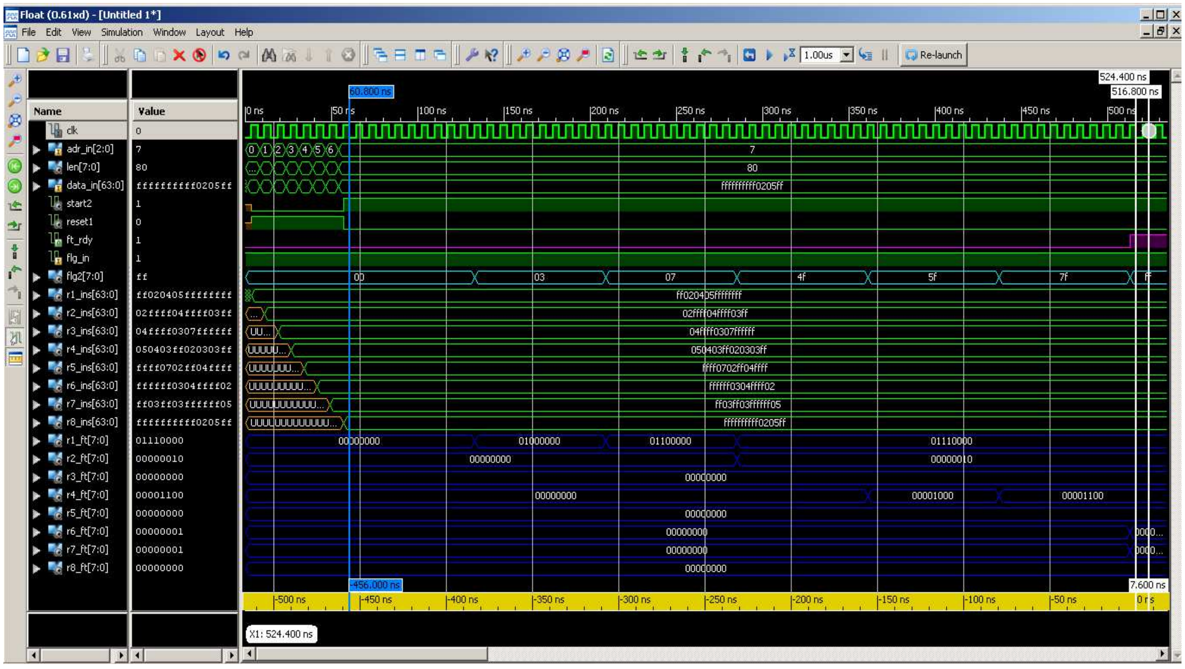

Name	Value
clk	0
adr_in[2:0]	7
len[7:0]	80
data_in[63:0]	ffffffffff0205ff
start2	1
reset1	0
ft_rdy	1
flg_in	1
flg2[7:0]	ff
r1_ins[63:0]	ff020405ffffffff
r2_ins[63:0]	02ffff04ffff03ff
r3_ins[63:0]	04ffff0307ffffff
r4_ins[63:0]	050403ff020303ff
r5_ins[63:0]	ffff0702ff04ffff
r6_ins[63:0]	ffffff0304ffff02
r7_ins[63:0]	ff03ff03ffffff05
r8_ins[63:0]	ffffffffff0205ff
r1_ft[7:0]	01110000
r2_ft[7:0]	00000010
r3_ft[7:0]	00000000
r4_ft[7:0]	00001100
r5_ft[7:0]	00000000
r6_ft[7:0]	00000001
r7_ft[7:0]	00000001
r8_ft[7:0]	00000000

Figure 5. Simulation results of 8 node FPGA-based PSPS processor

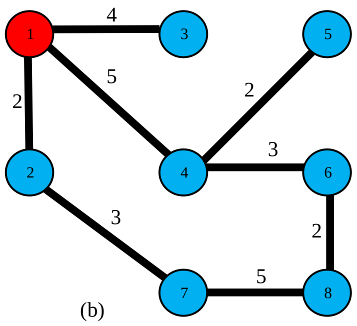
The final cost matrix is represented by the binary bit array [01110000, 00000010, 00000000, 00001100, 00000000, 00000001, 00000001, 00000000], which has the routing table information. Figure (6) illustrates the final binary bit array resulted from PSPS processor and how it represents the links that construct the shortest paths network. In Fig. 6 (a), bit position (i, j) of the binary bit array tells if the connection link from node i to node j is used in the shortest paths network or not, depending on its binary value. If its value is 1, then the link (i, j) will used, while if its value is 0, the link (i, j) will not be used and can be dealt with as unavailable link. Figure 6 (b) shows the final shortest paths network when the source node is node 1.

VII. CONCLUSIONS

Even though Dijkstra algorithm is an efficient algorithm for computing the shortest paths in a weighted directed graph problem, its run time is $O(n^2)$, that will consume a long execution time for large size networks.

From node#	To node#	1	2	3	4	5	6	7	8
1		0	1	1	1	0	0	0	0
2		0	0	0	0	0	0	1	0
3		0	0	0	0	0	0	0	0
4		0	0	0	0	1	1	0	0
5		0	0	0	0	0	0	0	0
6		0	0	0	0	0	0	0	1
7		0	0	0	0	0	0	0	1
8		0	0	0	0	0	0	0	0

(a)



(b)

Figure 6. Final routing table information of the network topology of Fig.(1).
(a) Binary bit array of routing table.
(b) The shortest path network of source node 1.

In this paper, a fully parallel FPGA-based processor architecture has been proposed for accelerating the shortest path searching and the routing table constructing of OSPF networks. The proposed PSPS processor design architecture is based on a parallel shortest path searching algorithm. The shortest paths searching process has three phases of its operation, the cost matrix loading , shortest path computing, and routing table information extracting. It has been noticed that even the times required for loading the cost matrix and extracting routing table information become longer for larger network sizes, they can further be reduced. For example, at n=128, the time of loading and extracting is 50.2% of the total computing time of the processor. Many network sizes have been considered and accordingly high speed up factors in the range (76.77-103.45) have been achieved.

It can also be noticed that the FPGA-resource consumptions are also increased exponentially with the increase of the network size. For n=128, the processor occupies 42% of the FFs and 65% of the LUT elements. It is clear that if we go on to increase the processor capability for large scale networks, such as 256-node network, with the same strategy, the required logic resources will exceed the total capacity of the FPGA chip. For this reason, currently we are trying to restrict the work to the implementation of an FPGA-based PSPS processor capable to handle 256-node networks with some modifications to handle larger network sizes.

VIII. REFERENCES

[1] D. Medhi and K. Ramasamy, Network Routing Algorithms, Protocols, and Architectures, Elsevier Inc, 2007.
[2] B. A. Forouzan, Data Communications and Networking, 4th edit, McGraw-Hill, 2007.
[3] E. W. Dijkstra, “A note on two problems in connection with graphs”, Numerische Mathematik, vol1.1, pp.269–271, 1959.
[4] X. Xiao and L. Ni, “Reducing Routing Table Computation Cost in OSPF”, Internet Workshop, 1999. IWS 99 , pp.119-125, 18-20 Feb. 1999.
[5] B. Xiao, J. Cao, Q. Zhuge, Z. Shao, E. H.-M. Sha, “dynamic Update of Shortest Path Tree in OSPF”, in Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN’04), pp. 18-23, May 2004.
[6] A. K. Phipps, “parallel algorithms for geometric shortest path problems”, Master thesis, School of Informatics, University of Edinburgh, 2004.
[7] H. Ishikawa, S. Shimizu, Y. Arakawa, N. Yamanaka, K. Shiba, “New Parallel Shortest Path Searching Algorithm based on Dynamically Reconfigurable Processor DAPDNA-2”, IEEE International Conference on Communications, 2007. ICC '07, pp.1997-2002, 24-28 June 2007.
[8] S. Shimizu, T. Kihara, Y. Arakawa, N. Yamanaka, K. Shiba, “A Prototype of a Dynamically Reconfigurable Processor Based Off-loading Engine for Accelerating the Shortest Path Calculation with GNU Zebra” , International Conference on High Performance Switching and Routing, 2008. HSPR 2008, pp.131-136, 15-17 May 2008.
[9] K. Sridharan, T. K. Priya, P. R. Kumar, “Hardware Architecture for Finding Shortest Paths”, TENCON 2009 - 2009 IEEE Region 10 Conference, pp.1-5, 23-26 Jan. 2009.
[10] I. Fernandez, J. Castillo, C. Pedraza, C. Sanchez, J. Ignacio Martinez, “Parallel Implementation of The Shortest Path Algorithm on FPGA”, 2008 4th Southern Conference on Programmable Logic, pp.245-248, 26-28 March 2008.
[11] G.R. Jagadeesh, T. Srikanthan, C.M. Lim“Field programmable gate array-based acceleration of shortest-path computation”,IET Comput. Digit. Tech., 2011, Vol. 5, Iss. 4, pp. 231–237.
[12] M. Y. Idris, S. Abu Bakar, E. Mohd Tamil, Z. Razak, N. Mohamed Noor,“High-Speed Shortest Path Co-processor Design”,ams, pp.626-631, 2009 Third Asia International Conference on Modelling & Simulation, 25-29 May 2009.
[13] Xilinx, “Virtex-7 FPGAs Data Sheet: DC and Switching Characteristics”, Advance Product Specification, DS183 (v1.2) November 7, 2011, available on www.xilinx.com.
[14] Xilinx, “Synthesis and Simulation Design Guide”, UG626 (v 12.3) September 21, 2010, available on www.xilinx.com.
[15] Xilinx, “ISE In-Depth Tutorial”, UG695 (v13.1) March 1, 2011, available on www.xilinx.com.
[16] Xilinx, “ISE Simulator (ISim) In-Depth Tutorial”, UG682 (v 13.2) July 6, 2011, available on www.xilinx.com.

IJECSE
International Journal of Electronics and Computer Science Engineering
WWW.IJECSE.ORG