Implementation of Recurrent Neural Network Algorithm for Shortest Path Calculation in Network Routing

Nasir Shaikh-Husin, Mohamed Khalil Hani, and Teoh Giap Seng
utmseng@hotmail.com
Universiti Teknologi Malaysia

Abstract

This paper describes the architecture and implementation of a shortest path processor, both in reconfigurable hardware and VLSI. This processor is based on the principles of recurrent spatiotemporal neural network. The processor's operation is similar to Dijkstra's algorithm and it can be used for network routing calculations. The objective of the processor is to find the least cost path in a weighted graph between a given node and one or more destinations. The digital implementation exhibits a regular interconnect structure and uses simple processing elements, which is well suited for VLSI implementation and reconfigurable hardware.

1. Introduction

In packet switched computer networks, data are routed along communication links making up the network. There are numerous possibilities for a data packet to navigate from its source node, through multiple nodes, to its destination(s). Each path has a measurable parameter or cost representing the desirability of using that particular link. The cost may reflect the available bandwidth, path delay, or bit error of the network. The shortest path is the path for which the total cost from the source node to the destination node is the minimum. In the Open Shortest Path First (OSPF) link-state network routing protocol, Dijkstra's algorithm is the most popular shortest path algorithm being used [1].

A significant improvement in performance could be realized if the shortest path computation is ported to hardware. In this paper, we show how Dijkstra's algorithm [2] can be implemented, both in reconfigurable hardware and VLSI. The design concept is derived from recurrent spatiotemporal neural network principles [3]. The processor is controlled in such a way that only certain processing elements (space) are enabled at a particular time (tempo.)

The digital hardware implementation described in this paper is similar to works on the analog implementation [4], [5]. The analog shortest path processor uses D/A converters to translate the network cost and capacitors and voltage comparators for computation. A better alternative is to use current comparators. A circuit for this, called Loser-Take-All circuit, has been proposed [6]. For digital implementation, we use counters as the main components doing the computations. A digital implementation approach using RAM and digital comparators has also been reported [7].

2. Shortest path algorithms

2.1 Processor architecture

Fig. 1 shows the architecture of the shortest path processor for a five-node problem. The processor consists of identical processing elements and control elements, which are interconnected in rows and columns in a regular structure. Each processing element (PE) is associated with a particular path. For example, the PE in row 2, column 4 is assigned the cost for traversing node 2 from node 4. Similarly, the PE in row 4 and column 2 is assigned the cost for traveling from node 2 to node 4. The control element (CE,) placed diagonally in Fig. 1, is basically an OR gate.

Besides the PE and CE, the only other components needed for the processor are a couple of decoders. One

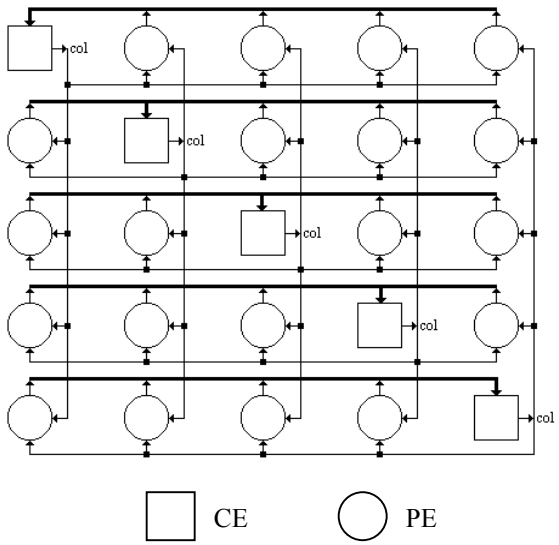

Fig. 1. Block diagram of processor

decoder is used to address each PE in sequence in order for each PE to be initialized with a preset cost. The other decoder decodes which node is the starting node.

When enabled, a PE will start counting down and it will fire (i.e. produce a HIGH output) when it counts down to zero. Among those enabled, the one assigned the least cost will be the winner. The winner then triggers the CE in its row. The CE output is fed back to all PEs in that row. This signal inhibits the rest of the PEs in the row from firing, hence making sure that only the PE that reaches zero can fire in a particular row. If two or more PEs reach zero at the same time, then all of them will fire. With this mechanism, multiple shortest paths can be determined, if the paths exist. The output of the CE is also routed to all the PEs in the corresponding column. Here, the signal acts as the enable flag; the PEs must wait for this signal before they can start counting. Previously inhibited PEs are precluded from firing.

The enabling and disabling of the PEs continues until all the PEs are disabled. At this time, all the paths in the graph have been compared, and the PEs that fire indicate the shortest paths.

2.1.1. PE Block Diagram. The main component of each PE is an 8-bit down counter with parallel load and enable inputs. Fig. 2 shows the block diagram of a PE. The path cost is loaded into the counter via the data bus when the *address* signal is asserted. The feedback signals from CE, together with *start* signal determine whether the counter is enabled or disabled. If the counter is not disabled, it counts down until it reaches zero. When this happens, the PE output becomes HIGH, and activates the CE in the PE's row. The HIGH output also would disable the counter itself, and hence maintains the output permanently.

2.2 Processor operation

A weighted graph with symmetrical costs as shown in Fig. 3 is used to illustrate how the processor operate. The graph is mapped into the processor by initializing the PEs

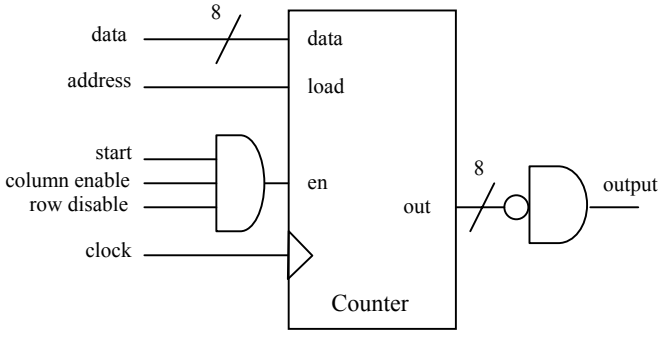

Fig. 2. PE block diagram

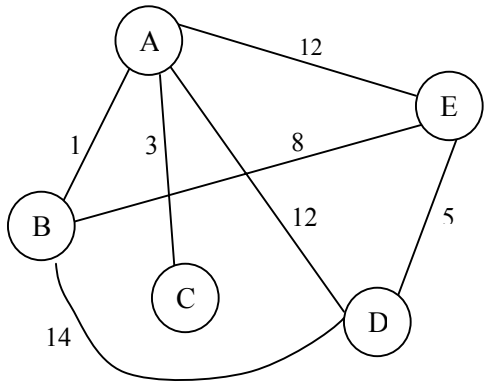

Fig. 3. Sample graph with symmetric costs

as depicted in Fig. 4. If two nodes are not connected, the PEs representing the path are assigned the maximum cost, which is 255. These PEs will be disabled and will not participate in the shortest path determination.

Assuming node D as the starting node, all PEs in D row are immediately disabled, while at the same time the PEs in column D are all enabled. However, the counting will begin only when *start* signal is asserted. When the counting begins, the PE in column D and row E will reach zero first, while the cost of the other PEs decrease accordingly, as shown in Fig. 5. This means the shortest path found so far is from D to E.

The zero PE now triggers the CE in row E, which subsequently disables all PEs in that row and simultaneously enables the PEs in column E (see Fig. 6.) The competition then continues until the processor representing path D-A fires. The same process goes on until all CEs are triggered. The shortest paths from D to all destinations are summarized in Table 1.

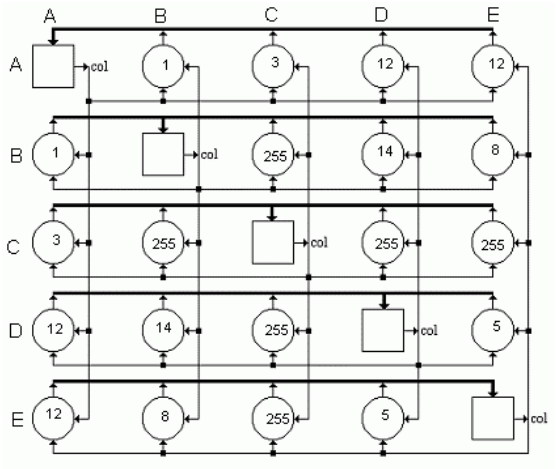

Fig. 4. PE initialization corresponding to sample graph in Fig. 4

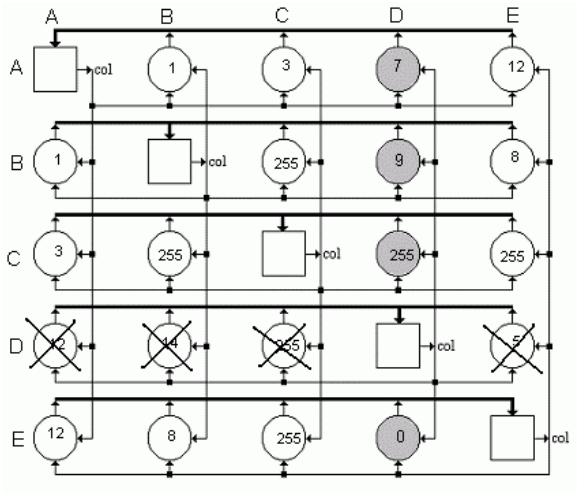

Fig. 5. Processor state after finding shortest path D-E. Disabled PE is indicated with a cross.

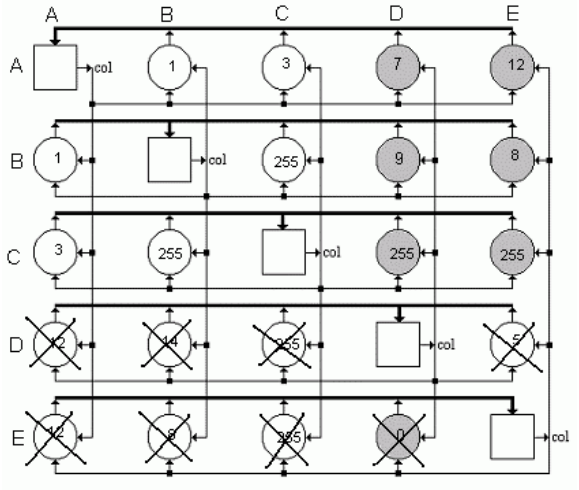

Fig. 6. Active PEs after shortest path D-E is found

Table 1. Shortest paths for sample graph in Fig. 3

Destination	Shortest path(s)	Path cost
D to A	D-A	12 clock cycles
D to B	D-A-B or D-E-B	13 clock cycles
D to C	D-A-C	15 clock cycles
D to E	D-E	5 clock cycles

2.3 Modified design to solve symmetrical graph

The processor architecture described so far can solve asymmetrical graph problem. For a symmetrical graph, the processor can be modified so that the number of PEs required is reduced by half. For a symmetrical problem, the cost from node A to node B for example is the same as the cost from node B to node A. Hence, only one PE is needed for both paths. Therefore, the PEs in the bottom triangle can be mapped to the respective PEs in the top triangle. Each PE has to be modified slightly however. Each one has a 2:1 multiplexer and a 1:2 demultiplexer to steer appropriate signals to the proper paths. The modified processor is shown in Fig. 7. The modified PE now is activated by two CEs, and its output is fed back to two CEs instead of one.

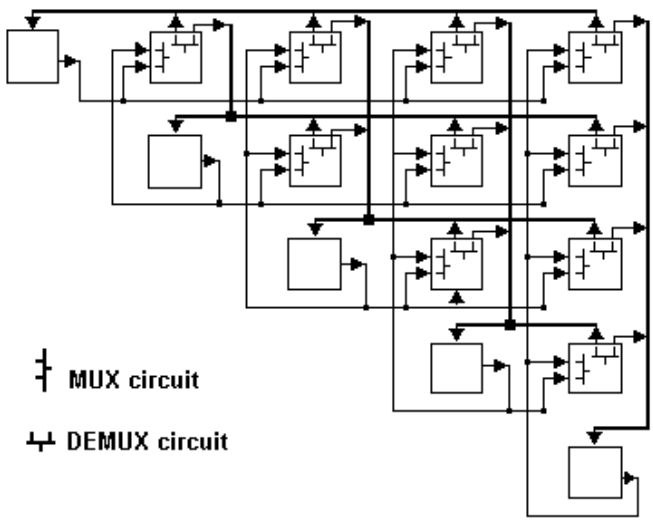

Fig. 7. Modified processor for solving symmetrical graph problem

3. Hardware resource utilization

The hardware for the processor is obviously dependent on the problem size. More hardware resource is needed to solve for a larger problem.

According to the algorithm, initially a row is disabled. After that, more rows will be disabled. Let n be the number of disabled rows and N be the size of the problem. The number of PEs that are operating at any iteration is given by

$$\text{Number of active PEs, } X = n(N-n) \qquad (1)$$

The maximum number of active PEs in a particular iteration, which is the minimum possible PEs needed in a processor, X_{min}, can be obtained by normal differential equation, $dX/dn = 0$, which results in $N - 2n = 0$, giving

$$n = \frac{N}{2} \qquad (2)$$

Hence,

$$X_{min} = \frac{N}{2}\left(N - \frac{N}{2}\right)$$

$$X_{min} = \frac{N^2}{4} \quad (3)$$

Therefore, for an N node problem, theoretically, we need the minimum $\frac{N^2}{4}$ (for even number of N) or $\frac{N^2-1}{4}$ (for odd number of N) processing elements (PE) in a processor. However, for an N node problem, we need $N^2 - N$ PE to store the asymmetric network cost, and $\frac{N^2-N}{2}$ PE to store the symmetric network cost. Therefore, the X_{min} given by (3) can only be achieved by using the external memory to store and hold the temporary data, which may increase the hardware area.

Currently, for an N node problem, our optimized design use up to $\frac{N^2-N}{2}$ of PEs, which is about twice of the theoretical minimum value given by (3).

4. VHDL description and simulation results

A parameterizable shortest path processor was designed in VHDL and was synthesized using Synopsys FPGA Compiler. The processor can be scaled by changing the parameters for size of the graph and the bit length of the *source* and *addr* buses.

Fig. 8 shows the simulation results from Altera's MAX+plus II software. The sample problem is the graph in Fig. 3, with node D as the source node. The processor first of all is initialized with path costs. During this period, *start* must be LOW and *addr* is changed from 00_{16} to 13_{16} to address all the PEs while the corresponding cost assigned to each PE is supplied. The *source* bus is set to 3 to indicate node D is the source node. The processor then waits for *start* to be HIGH before it commences its calculations.

The simulation shows that five clock cycles after *start* is asserted, output4_5 (representing path D-E) goes HIGH. The next PEs that go HIGH are the PEs representing paths D-A, A-B and E-B (both become HIGH at the same time,) and A-C in the respective order. The result is consistent with that in Table 1.

The synthesized design was targeted to Altera

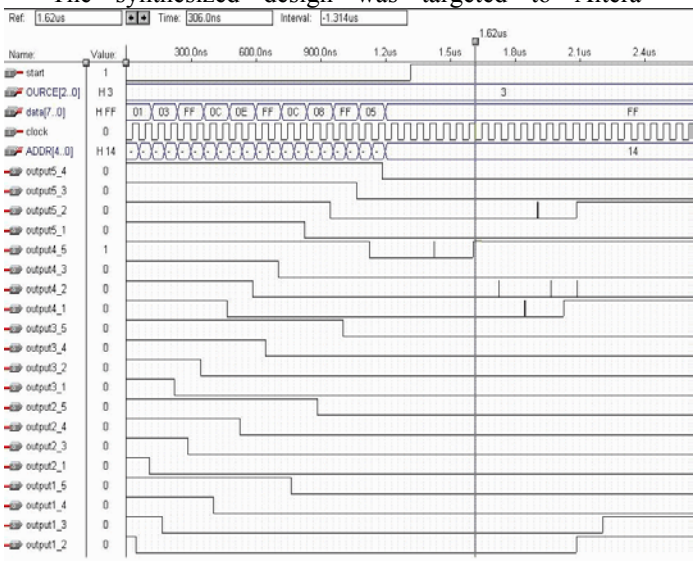

Fig. 8. Simulation results

FLEX10K reconfigurable device. The implementation data is shown in Table 2.

Table 2. Area usage and timing reports of the shortest path processor

Problem size (# of nodes)	Device	# of LEs	Min. clock period
5	EPF10K20RC240-4	323 (28%)	28.1ns
6	EPF10K20RC240-4	477 (41%)	30.0ns
7	EPF10K20RC240-4	660 (57%)	29.6ns
8	EPF10K20RC240-4	872 (75%)	29.8ns

5. VLSI implementation

A VLSI implementation of a processor for symmetric costs has been completed. The layout is for MOSIS 1.2 μm scalable CMOS N-Well process. The layout is done automatically using cell library. The circuit schematic was drawn using Tanner Tools S-edit, and the cells were placed and routed using Tanner Tools L-edit Standard Cell Place and Route (SPR). We used the Tanner provided MOSIS AMI 1.2 μ cell libraries to place and route the cells, and to fit the generated core cells into a 40

pin tiny chip which has an available core area of 2564 λ x 2574 λ.

Since only 10 PEs are required for a five-node problem with symmetrical costs, the *addr* bus is only four bits. To fit into a 40-pin chip, with active area limited to 2564 λ x 2574 λ, the counter bit length was reduced from 8-bit to 4-bit. Fig. 9 shows the chip's I/O. The generated core area occupies 1973 λ x 1474 λ. The complete layout is shown in Fig. 10.

Simulation result of the extracted layout using T-Spice is shown in Fig. 11. The sample problem is the graph in Fig. 3. The simulation gives exactly the same solution as in Fig. 8.

6. Conclusion

An algorithm to solve shortest path problems and its implementations has been described. The algorithm can provide multiple shortest paths between a source node and a destination node, if equal paths exist. The regular and simple circuit structure required for hardware implementation can be easily expanded to solve larger node problem.

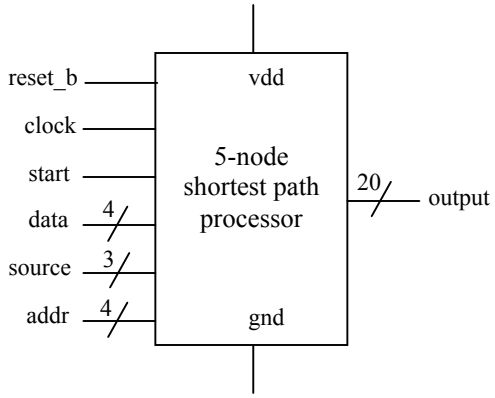

Fig. 9. Chip I/O

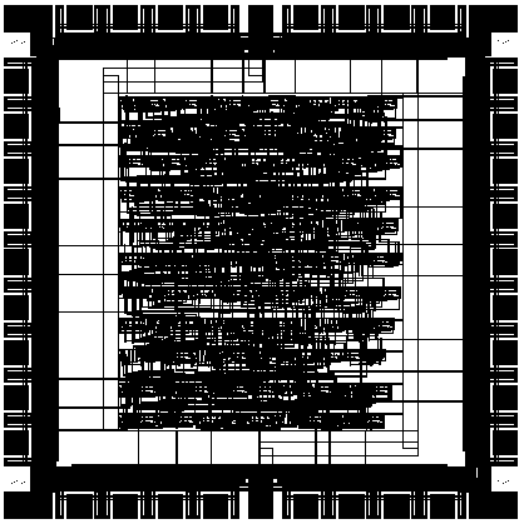

Fig. 10. VLSI layout

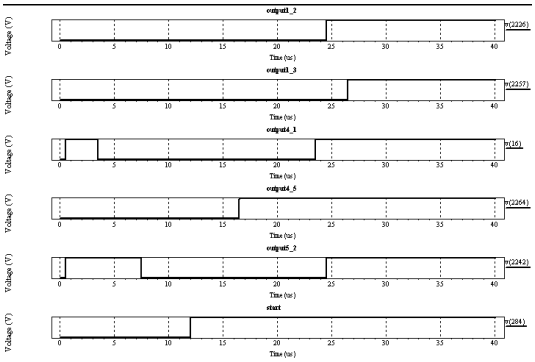

Fig. 11. Simulation results

A modification to the basic architecture to solve a symmetrical graph problem has also been described. The counter can be replaced with a comparator so that PEs with lowest cost can be determined more quickly [4].

A VLSI implementation for the basic processor has also been completed. Since the circuit structure is very regular, a VLSI implementation using Block Place and Route technique is currently underway, in order to get a better organized VLSI layout.

ACKNOWLEDGEMENT: This research project is supported by IRPA under vot number 72307.

7. References

[1] W. Stallings, *Data and Computer Communication*, NJ: Prentice-Hall, 1999.

[2] Dijkstra, A Note on Two Problems in Connection with Graphs, *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.

[3] J. L. Meador, "Spaciotemporal Neural Networks for Shortest Path Optimization," in *Proc. of IEEE International Symposium on Circuits and Systems,* Seattle, WA, vol. 2, pp. 801-804, 1995.

[4] N. Shaikh-Husin and J. L. Meador, "Mixed Signal Neural Circuits for Shortest Path Computation," in *Conference Record of the Twenty-Ninth Asilomar Conference on Signals, Systems, and Computers '95*, Pacific Grove, CA, vol. 2, pp. 876-880, 1996.

[5] N. Shaikh-Husin and J. L. Meador, "Spatiotemporal Neural Networks For Link-State Routing Protocols," in *Proc. of IEEE International Symposium on Circuits and Systems,* Atlanta, GA, vol. 3, pp. 547-550, 1996.

[6] N. Shaikh-Husin and J. L. Meador, "A High Precision Current Copying Loser-Take-All Circuit," in *Proc. of World Engineering Congress 1999*, Kuala Lumpur, Electrical and Electronic Engineering vol., pp. 177-179, 1999.

[7] M. Tommiska and J. Skytta, "Dijkstra Shortest Path Routing Algorithm in Reconfigurable Hardware," in *Proceedings of 11th International Conference FPL2001,* pp 653-657, 2001.