International Journal of Engineering Sciences & Emerging Technologies, Feb. 2013.
ISSN: 2231 – 6604 Volume 4, Issue 2, pp: 110-116 ©IJESET

DIJKSTRA ALGORITHM IMPLEMENTATION ON FPGA CARD FOR TELECOM CALCULATIONS

BENAICHA Ramzi[1], TAIBI Mahmoud[2]
[1] LERICA laboratory, Badji-Mokhtar University, Annaba, Algeria
ramzi.benaicha@hotmail.fr
[2]LASAlaboratory,Badji-Mokhtar University, Annaba, Algeria.
mahmoud.taibi@univ-annaba.org.

ABSTRACT

In Information Technology networks for data packets coming from a source and arriving to the input of a network (WAN for example) according to specifications, the routing system must assign to the network the best and optimal path in order to avoid the congestion phenomenon.
In this paper, we suggest a new approach for the implementation of the DIJKSTRA routing algorithm by using an FPGA development card (Xilinx), this is for accelerating the routing process of the IT networks,whatever the number of connected node where the network must provide combine flexibility andspeed.
In our investigation, we considered the following: at the beginning, we present the routing system in the IT networks and explain the DIJKSTRA routing algorithm. We present the new implementation architecture, we do a simulation and present the obtained results and we compare them with the ordinary processors.

KEYWORDS: *IT networks, protocol OSPF, DIJKSTRA's Algorithm, VHDL language, FPGA.*

I. INTRODUCTION

Today, with the spectacular advanced that experienced technology of manufacture of integrated circuits, in particular that of the FPGAs, many complex applications that fall within the field of digital signal processing could see the day. These include: computer networks, data security, etc. In their majority, these applications are developed either based on microcontrollers, microprocessors or complex logic circuits [5].
A computer network is a set of interconnected devices that are used to deliver a flow of information. To properly route information, network uses the routing process. The network must be able to combine flexibility and speed, hence the need to turn to new technologies such as the FPGAs that than those encountered on current processors, in other words to combine the flexibility of the software, hardware speed.
One of the first implementation is DIJKSTRA algorithm at Paris central school, he implemented the algorithm dijkstra with C software in a regular microprocessor, he tested its implementation with graphs in other work proposed in reconfigurable computing [4]. In the communication systems, it is an approach that promotes the hardware implementation compared to the software implementation [10], all of these reasons motivated by the desire to focus on this work.
The remainder of this article is organized as it follows.In Section 2, we describe the routing system in the IT networks. In Section 3, we present the new implementation architecture. We make a simulation and present the obtained results that we compare with the ordinary processors in Section 4, and concludethe paper by discussing open problems and researchchallenges in Section 5.

II. THE ROUTING IN COMPUTER NETWORKS

2.1. The Routing

Routing is a function of Layer3 (network layer) of the OSI (Open System Internet) model. It is a hierarchical system that brings together individual addresses. These are treated as a whole until the address of destination is required for final delivery of data [1].

2.2. The Operation of Routers

Routers allow interconnecting segments of a network or entire networks. Their role is to route frames of data between networks, based on the information of the layer3. They make logical decisions as to the best possible delivery of data and then redirect packets to appropriate port of exit so that they are encapsulated for transmission. Encapsulation and de-encapsulation phases occur at each passage of a packet in a router. The router must unwrap the fabric of the layer2 data to access the address of layer3 and consider it. Encapsulation is to split the flow of data into segments and add headers and in queues appropriate before transmitting the data. Process of encapsulation, is to remove the headers and the queues, and then recombine the data in a continuous stream [1].

2.3. Topology Suggested

To verify the ability to route the data collected by different sensors, a multiple nodes network topology is selected, as shown in figure1.

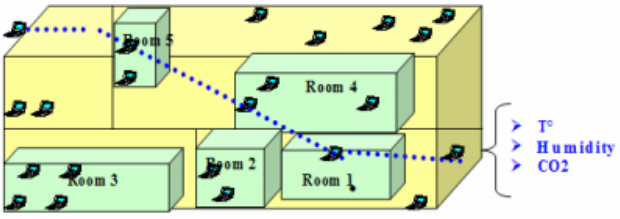

Figure1: Network topology

This is the most simple network topology which can exist, where all users have wireless cards and talk to each other directly without any intermediary. Features are taken up by the stations themselves. Other functions are not usable in this case. This technique is the most economical for some stations as it does not require a relay base station called "Access Point" or AP. [9]

III. THE DIJKSTRA'S ALGORITHM

Published in 1959, the Dijkstra algorithm is its usefulness in the calculation of the routing process. The weight of the arcs can be distance (for the shortest route), the time estimated (for the fastest route), the most economical (with fuel consumption and the price to be paid).
An application of the most common of the Dijkstra algorithm is Protocol open shortest path first (OSPF) which allows a very efficient internet routing of information in seeking the most effective route. IS - IS (Intermediate system to intermediate system) routers also use this algorithm [2].

3.1. Description

Dijkstra's algorithm is a type greedy algorithm: at each new stage, dealing a new Summit.
Remains to define the choice of the Summit to address and it apply the algorithm quickly.
Throughout the calculation, it will therefore maintain two sets:

- C, all of the summits which are to visit; initially C = S-{source}
- D, all Set of summit for which we already know their smallest distance from the source; Initially, D = {source}.

The algorithm ends well obviously when C is empty.
For each summit s in D, it will keep in a distances table, the weight of the shortest path to the source, and in an array route, the Summit which in a shortest path from the source to s. Thus, to find a shortest path, it will suffice to back predecessor in predecessor to the source, which can be done with a single recursive call.

3.1.1. Initialization

In the algorithm, the shortest known between the source and each of the summits is the direct path, with an edge of weight infinite if there is no connection between the two peaks. It initializes therefore distances table, by the weight of the edges connecting the source to each of the summits, and route table, the source for all summits.

ist step:

Assuming we have already treated i summits, routes and distances respectively contain the weight and the predecessor of the shortest paths for each of the already processed summits.

Either s on top of C realizing the minimum distances [s]. It removes C s and added to d. rest to update distances tables and routes for the t summits directly connected to s by an edge as follows:

If distances [s] + F (s, t) < distances [t];

Then it replaces distances [t] by distances [s] + F (s, t)

And route [t] s... and that's it!

(n-2) th step:

Initially, there are (n-1) summits to visit, but as discussed below, the last step is unnecessary. Thus, as soon as the (n-2) th step, distances and routes contain all the information necessary to find shortest paths from the source to each of the other summits (since then D = S):

- distances [s] is the weight of the shortest path from source to s.
- route [t] is the predecessor of s in a shortest path from the source to s.

3.1.2. Proof of the Algorithm

It remains to prove that this algorithm works indeed, which at first sight is not frankly obvious. The recurrence to perform is based on two assumptions:

- Distances and routes tables can be obtained for all points s d a short path that leads from source to s.
- Distances and routes tables can be obtained for all points t C a more short path that leads from source to t passing by D points.

These two hypotheses are well verified in the initialization of tables, if they are still on arrival, as D = S, the second hypothesis provides proof of the algorithm.

IV. ARCHITECTURE AND SIMULATION

In this section, we present the architecture forthe implementation of the algorithm of Dijkstra on FPGA using aVirtex5 XC5VLX110T card,a PC with an Intel core 2 duo 2.10 GHz processor and 3 GB RAM and 32-bit operating system windows 7. [3]

The program is written in VHDL and synthesized using the Xilinx ISE12.1 tool, the program is set over four parties, the command block, the memory block, the operator block & comparator block.

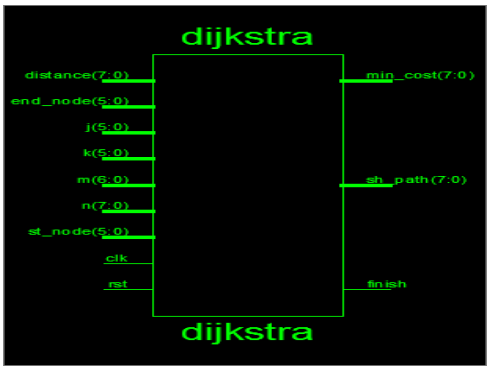

Figure 2: schematic dijkstra I/O block

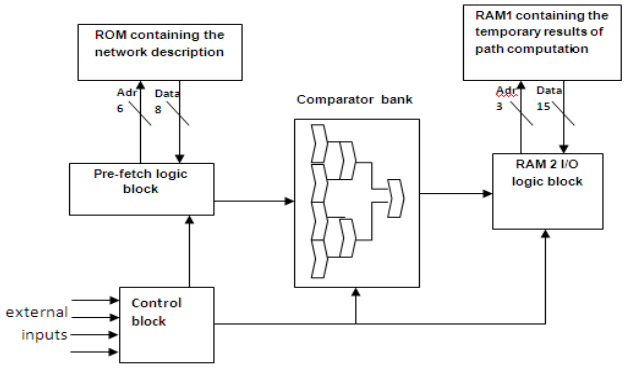

Figure 3: Schema of implementation of the dijkstra algorithm approach

4.1. The functioning of the Architecture

The execution of the program, the ROM contains the description of the network, the RAM1 is initialized to 0 for 'a' and at infinity for the rest of the nodes. The RAM2 it contains a list of all nodes (except 'a') because they have not yet been processed by the program.
The command block reads the list of nodes in the RAM2 untreated and forwards with the node of current blocks RAM1 and ROM. This transmission is made a form of address, block RAM1 receiving so these addresses returns not to command block, but rather to the operator block, indices of nodes passed to it, same ROM as it also refers to the block operator not the indices of the nodes in the network, but the cost of their links. The operator block performs calculation expected, and then performs an update of indices of the nodes of the network contained in the RAM1, and finally blocks space containing data on the processes node (which is here ' a' to the first loop of the program), it is so that the comparator block may not access.
The comparator block compares then all the values of the indices of the nodes that are visible, and returns the minimum index in the command block node. It performs an update of list of nodes in the RAM2 by sending this list the minimum index node that has now become the addresses node, and the cycle is repeated.
The program will stop when the RAM2 will be empty.
When the program is complete, there is still a small program which allows to unlock the RAM1 data and these data to generate a file containing paths different detailed of the node that contains the packet of data to all other nodes of the physical network (it is actually the table routing).

4.2. Results and Discussion

Dijkstra program coded in VHDL tool 12.1 of Xilinx ISE was implemented, the same algorithm in C is already implemented on a regular processor. A common example of network architecture was tested separately on each of the two processors, the results are presented in figures4 and 5:

Table 1: logical elements used by the number of nodes in the network

Environment (nature of objects)	Slices %	LUTs %	Bonded IOBs
10 nodes (and 14 edges)	7,915	15,011	25
20 nodes (and 32 edges)	16,233	32,161	25
30 nodes (and 64 edges)	17,477	34,796	27

We can say that the number of logic elements entering in the configuration of FPGA increase with the importance of the network which is quite logical because it needs more operating resource memory which consums more energy by the FPGA.

Number of nodes; Time of executions of Dijkstra (µs)VHDL on FPGA; (C) a µP; Time report µP/FPGA

Table 2: Time of executions of Dijkstra's algorithm

Number of nodes	*Time of exécutions of Dijkstra's Algorithm (µs)*		*Report of time µP /FPGA (%)*
	VHDL on FPGA	**C on µP**	
10	10	259	25,9
20	13	474	36,46
30	17	832	48,94
40	21	1104	52,57
50	25	1456	58,24

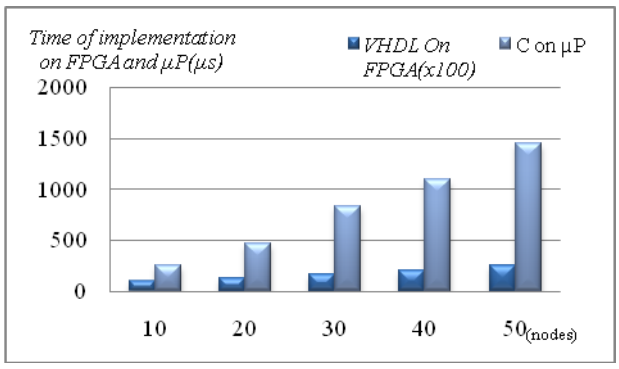

Figure 4: Time of implementation FPGA and µP

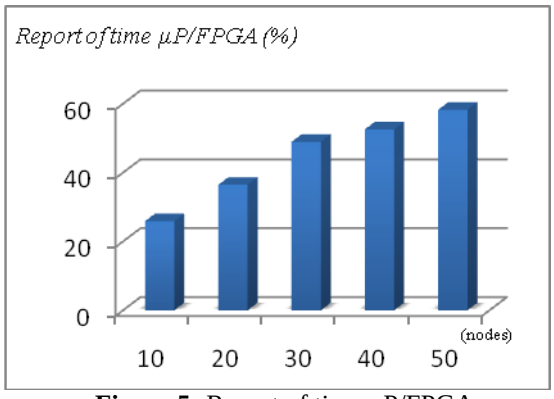

Figure 5: Report of time µP/FPGA

The second point that we should highlight is obviously on the different running time of the algorithm. First of all on the FPGA as on the single microprocessor, the running time of the algorithm increases with the importance of the network, which is obviously quite logical.then the temporal reports column shows us how the FPGA performs faster than a conventional processor.

This is due to two fundamental reasons:

First the difference is the level of the internal architecture of FPGA which is designed for that specific dedicated tasks ensure that during operation, it performs the task assigned to it, it's what gives it its speed. Conventional microprocessor performs several tasks at the same time which is a cause of delay compared to FPGA.

The FPGA also has the advantage that its programming combines the flexibility of software with the speed of the hardware, while the microprocessor is made in a software manner only and is therefore has not the hardware speed of the FPGA.

Multiple variables instructions are executed concurrently on the FPGA, multiple arithmetic operations, such as comparisons, are executed in parallel.

At the end of the implementation, a temporal analysis was conducted to determine the time performance of this algorithm. To this end, the timing tool analysis revealed a clock 'clk' of minimum use which is of 5.873ns. Which corresponds to a frequency fclk = 170.271 MHz.

To complete the design flow, the proper functioning of this algorithm has been validated by simulations at different levels of abstraction. This operation was conducted from a test file that is named test-bench and was conducted by using the tool of simulation Modelsim PE 7.0.

Temporal simulation results are shown on figure 6.

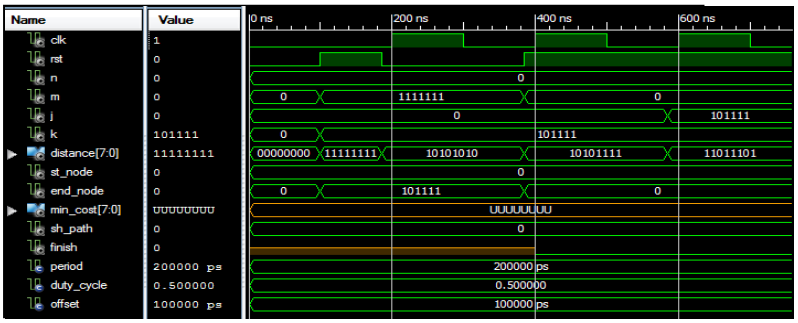

Figure 6: Temporal Simulation of Dijkstra's algorithm

A physical test of our implementation was carried out for the real operation. This means after the loading card. This test has given satisfactory results. The application used to perform this test is a network of connected node (30 knots and 64 edges) which is to determine the shortest path between two nodes.

Table 3: the elements for a network (30 nodes,64 edges)

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	3559	301440	1%
Number of Slice LUTs	33191	150720	22%
Number of fully used LUT-FF pairs	2892	33858	8%
Number of bonded IOBs	59	400	14%
Number of BUFG/BUFGCTRLs	1	32	3%

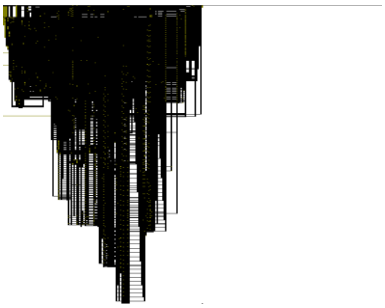

Figure 7: Schematic of the top-level block
(30 nodes and 64 edges)

V. CONCLUSIONS

In this investigation, we have suggested an approach for implementing one of the most used algorithms in communication networks via a development type Virtex5 FPGA card equipped with a chip of type XC5VLX110T.
Ordinary processors which executes routing algorithms have increasingly growing difficulties with the qualifications and the requirements of the specifications required by the networks.
In order to combine the flexibility of software and hardware speed, the trend was to turn to reconfigurable architectures in this case that of the FPGA.
This can be attributed to the following factors: multiple variables instructions are executed in a concurrent way, multiple arithmetic operations including comparisons are executed in parallel and tables and structures of data are directly implemented in the internal memory blocks.
In this paper, we first spoke of routing to place the subject in context, and then completed by a comparison between the implementation of the algorithm of Dijkstra on FPGA and on conventional processor.
Despite their relative slowness, more and more people are interested in the FPGA. Persons interested in the free Hardware use them to make a dedicated processor machine according to the tasks required. Most of these people use FPGA chips in parallel with a conventional processor. It allows to accelerate tasks by hardware while keeping a fast machine regardless of the task that is required.
The originality of our work deals with setting up an FPGA implementation approach. Then it is to define a methodology for the implementation of the Dijkstra algorithm in a topology of an ad-hoc by varying the number of node to extract the logic elements used and minimize the execution time of the algorithm. For our future works, we are working in the improvement and the adaptation of the

proposed approach in order to perform in real time. Furthermore, the implementation leads to the network congestion, that why we attempt to resolve this problem.

REFERENCES

[1] Pierre François, « Improving the convergence of IP RoutingProtocols ». L'Université catholique de Louvain, Octobre, 2008.
[2] Wikimedia, "Dijkstra's algorithm", 2008, http://en.wikipedia.org/wiki/Dijkstra's_algorithm.
[3] Xilinx Platform FPGAs. "Available from": http://www.xilinx.com.
[4] Dominique Hazail-Massieux, Implémentation de l'algorithme de dijkstra, http://www.nimbustier.net/publications/djikstra/index.htm
[5] Thoma Y., 2005. « Tissu numérique cellulaire à routage et configuration dynamiques ». Thèse de Doctorat en ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, Pays. 160p.
[6] Siew-KeiLam, Thambipillai Srikanthan., 2009. "Rapid design of area-efficient custom instructions for reconfigurable embedded processing", Journal of Systems Architecture 55 (2009) 1–14
[7] J.L. Träff, C. D. Zaroliagis, A simple parallel algorithmfor the single-source shortest path problem on planar digraphs, Journal of Parallel and Distributed Computing 60 (9) (2000) 1103–1124.
[8] M. Rauch, P. Klein, S. Rao, S. Subramanian, Faster shortest-path algorithms for planar graphs, Journal of Computer and System Sciences 55 (1997) 3–23.
[9] Fabrice LEMAINQUE, (2009) *Tout sur les reseaux sans fil*, Dunod, paris, p 107-113.
[10] MattiTOMMISKA, (1998*) Reconfigurable Computing In Communications Systems*, Thesis submitted for the degree of Licentiate of Technology in Engineering.

AUTHORS

Ramzi BENAICHA received his master in electrical engineering from UBMA University, ANNABA, Algeria in 2009, and searcher in LERICA laboratory at Annaba, Algeria since 2010. Currently, he is an assistant professor at Badji-Mokhtar University, Annaba, Algeria since 2011. His interests are in embedded systems.

Mahmoud TAIBIreceived his BSc in electrical engineering from USTO University, Oran, Algeria in 1980, then MSc from Badji-Mokhtar University, Annaba, Algeria in 1996. Currently, he is an assistant professor at Badji-Mokhtar University, Annaba, Algeria since 1983. His interests are in intelligent systems.