Mobile Robot Path Planning Software and Hardware Implementations

Lucia Vacariu, Flaviu Roman, Mihai Timar, Tudor Stanciu, Radu Banabic, Octavian Cret

Computer Science Department, Technical University of Cluj-Napoca, Romania
{Lucia.Vacariu, Octavian.Cret}@cs.utcluj.ro

***Abstract*—This paper presents the use of search algorithms and discrete environments to find the path in mobile robots systems navigation. Implementations of Breadth-First search algorithm are made both in software and in hardware. The results using the hardware approach provide significant time gain, compared with the software implementation. The time gain and the dynamical hardware reconfigurability allow multiple tasks implementation and the possibility to choose the appropriate one for the application.**

***Index Terms*— hardware design, mobile robots, path planning, search algorithms.**

I. INTRODUCTION

Mobile robots systems are driven by control systems. The architecture of these control systems provides different blocks to accomplish specific robotic tasks. Traditionally, the research in the field of Robotics is focused on the algorithms used to accomplish fundamental tasks, like path planning [1].

The path-planning task, part of a mobile robot navigation system, involves searching and finding the path between a Start Point and a Goal Point in the environment where the robots navigate. The environment is static or dynamic and may have a discrete or continuous representation. Mobile robots need to avoid obstacle collisions and generate an optimal result with respect to path dimension or to execution time. These imply different possible implementations [2].

To reach the goal, mobile robots need to have the ability to act as to assure an efficient and reliable navigation. A great variety of approaches for this strategic task have been experimented [3].

The approaches that solve path planning are generally based on software implementation. There are only a few hardware implementation directions that have demonstrated navigation competence [4].

Nowadays the Field Programmable Gate Array (FPGA) circuits and other hardware reconfigurable systems makes it possible to achieve implementations with a degree of flexibility that only software applications were considered to have until now. FPGA circuits allow running high-speed hardware applications and provide a high degree of parallelism [5].

While the quantity and diversity of the environments increases, the criteria and applications that imply huge computing power consumption for the mobile robots navigation continue to increase, the encapsulated real-time architectures for the mobile robots navigation systems must respond with flexibility and enhanced processing capacity and performance. Many repeating and time-consuming operations are adequate to be implemented in reconfigurable hardware. The establishment of a frame within which the hardware applications can be dynamically reconfigured to respond to the real-time requirements would allow new criteria approaches in robot applications [6].

The search algorithms represent a useful and reliable technique in solving path-finding, path-planning and obstacle-avoidance types of problems. They are successfully used to determine a valid path [7].

The basic idea of our approach is to highlight differences between software and hardware implementations of path generation from a Start Point to a Goal Point, path that will be followed by mobile robots. At the same time we show that the hardware approach provides significant time gain and also easy and fast switching between multiple algorithms. We started using the Breadth-First (BF) algorithm to obtain the path generated in a given static discrete environment. Implementations are made both in software and in hardware.

The paper is structured as follows: section 2 deals with the overall description of BF used algorithm and its software implementation and results. Section 3 explains the hardware implementation chosen for the algorithm and presents the tests and experimental validation. Section 4 reports the comparative results obtained by the two implementations, and shows conclusions and ideas for future work.

II. SOFTWARE IMPLEMENTATION

The environment where the path planning is made is one with a typical bi-dimensional discrete representation. Most often it is represented as a matrix of squares (cells), and denotes the positions of the cells using the combination of the

two coordinates. This way, the environment presents the cells as Nodes, and each node has neighbors in 4 directions (N, S, W, E). There is no connection on diagonals. The dimension of the matrix is known. The matrix contains free spaces, static obstacles, the Starting Point (S) and the Goal Point (G). It is assumed that the matrix is closed, bordered by obstacles.

There are several algorithms used to obtain a path in such environments. In a finite environment all the algorithms assure that a path will be found between the Start Point and the Goal Point. The characterization of algorithms is made with respect to the length of the generated path and with respect to the time spent to obtain it. These two criteria are independent. The shortest path does not necessarily imply the minimum time, or vice-versa.

By implementing path planning algorithms and their alternative usage with respect to the application demands, we aim to obtain an optimal path and minimum execution time.

A. *Breadth-First Search Algorithm*

We start using the Breadth-First (BF) algorithm to obtain the path. BF is a technique for searching and returning a path from a given Starting Point to a given Goal Point. The algorithm guarantees finding a solution, if there exists one. As for its complexity, it is a linear algorithm with respect to the number of considered nodes, while the way it searches is based on maintaining a queue of all neighbors found to be accessible through means of vertices until the Goal is reached. Keeping a similar queue, in which for each node we keep the one from which our node was found as neighbor, makes it possible to reconstruct the path.

Given a random map of the environment, that respects the criteria mentioned above, a Starting Point and a Goal Point, the implementation of BF will return (if there exists) a valid path between the two points. The path will be marked, assuming that one can pass only through the free spaces and the directions of motion are only the ones stated. The algorithm is presented in Listing 1.

The reconstruction of the solution is done by means of a special routine, which takes the list of previous elements and reconstructs the visited points of the map (Listing 2).

B. *Implementation and results*

For the software implementation and experiments, we have chosen Java as programming language. The implementation required Object-Oriented techniques, Graphical User Interface and Data/Maps saving and exporting, the Java compiler and emulator [8], and the Eclipse IDE [9]. The tests were performed on an Intel Pentium M at 1.73GHz, with 1G RAM.

The procedure to measure the time interval in which the algorithm runs was to capture the system time and to compute the difference of the values before and after the execution of the algorithm routine (Listing 3).

We performed various tests on manually generated maps, random maps, and small to large maps. The maps are displayed using Black for obstacles and White for free spaces. The starting point is Blue, and the goal is Yellow. We observed that the running time increases significantly for large maps. Each increase in the length of an assumed square matrix produces a 2nd order polynomial increase (with respect to the matrix dimension) in the execution time, because of the spreading (the cells included in the search queue but which are not on the final path).

```
//BF Search Algorithm

queue q          // the process queue
point start      // starting point
point goal       // goal point
point crt        // current point
list nb          // list of free neighbors of current point
list visited     // list of booleans <=> "true" = "visited"
list prev        // list of marks, for each node we mark the
                 // node which "discovered" the current one

// beginning of algorithm

push(q,start)
prev[first] <- -1 // mark that there is no "prev" for first
set all visited <- "false"
visited[first] <- "true"

while (q not empty) execute
    crt <- pop(queue)//pop current node
    if (crt == goal)
        return {point,goal,last} //solution found
    else
        nb <- all FREE neighbors of crt //4 directions in our case
        for all items in nb
            if (not visited[item])
                push(q,item)
                last[item] <- crt
                visited[item] <- "true"
            end if
        end for
    end if
end while

return "No solution" // if we arrive here the goal was not reached

// end of algorithm
```

Listing 1. BF Search Algorithm

```
// Reconstructing the solution path

point start
point goal
list prev
point current

// beginning of code

current <- goal;

while (current != -1) execute // "-1" is "prev" of first
    mark current as "in the path"
    current <- prev[current]
end while

// end of code
```

Listing 2. Reconstruction of path

```
Discrete d = new Discrete();  //space of matrix
d.readFromFile("ten.txt");    //acquire matrix

long time = System.currentTimeMillis();    //clock before
d = bf.startBFSearch(d);
long time2 = System.currentTimeMillis();   //clock after

elapsed = (time2 - time);                  //elapsed
new ShowPath(d).drawPath();                //map drawing
```

Listing 3. Execution time computing

Results prove very short paths (80% of the times optimal) and fast times only for small matrices (Fig.1.).

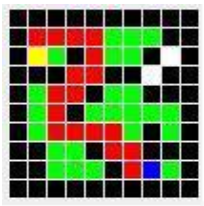

Fig.1. 10x10 map dimension

For larger maps (e.g. 100x100 cells) the time increases very much (Fig.2).

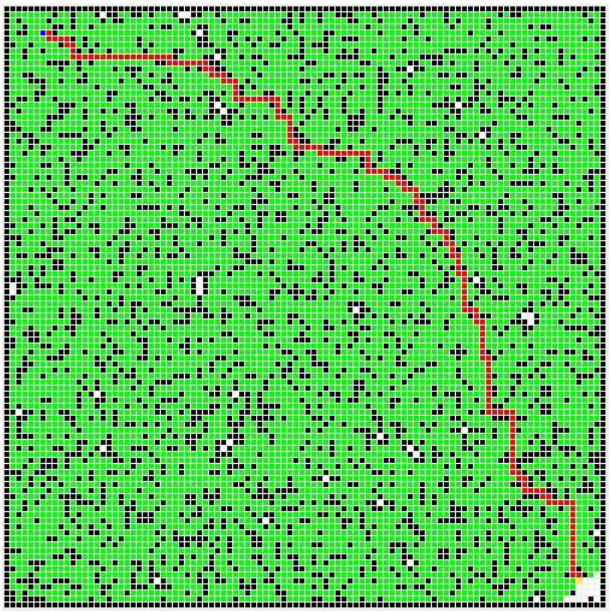

Fig.2. 100x100 map dimension

A comparison table (Table 1.) has been generated, based on the obtained time and the spreading results. A larger spreading implies larger execution time because the spreading is included in the search queue, so each node is expanded. For same dimension maps, but with fewer obstacles, the execution time was greater. This is because a less obstacle-filled map generates a larger spreading. The time is given in microseconds.

Width	Height	Total	Obstacles Fill	Spreading	Time (µs)
30	30	900	25%	High	10000
30	30	900	50%	Medium	< 10000
40	40	1600	25%	High	40000
40	40	1600	50%	High	10000
100	100	10000	25%	High	120000
100	100	10000	50%	High	40000

Table 1. Software BF implementation results

III. HARDWARE IMPLEMENTATION

Our hardware-based environment is built upon Xilinx FPGA core technologies, which is the well-known developing technology in reconfigurable-based computing functionalities.

A. *Hardware Design*

For the hardware implementation, we chose from the FPGA family, a Xilinx Virtex 2Pro Board (XC2VP30-FF896-6), manufactured by Digilent Inc. The board is equipped with VGA-output, used for visualizing test results on the monitor. It required Xilinx ISE 8.1 Environment [10], which was used for VHDL code synthesis, implementation and board programming.

For the hardware implementation, a series of adaptations of BF algorithm had to be done, in order to exploit the logic resources of the Virtex FPGA device. The input matrix is stored in BlockRAMs. Depending on the space available in the FPGA device, the process memory can be implemented inside or outside the chip (in the Virtex BlockRAMs or in an external Dynamic RAM).

All components have been described in parameterizable VHDL code. Thus, the design becomes portable on any hardware support system.

1) Components design: The hardware solution is based on a structural description with component-style design. It uses interconnected components, each of them performing a certain task. The most important component is the BF component, which implements the algorithm (Fig.3.).

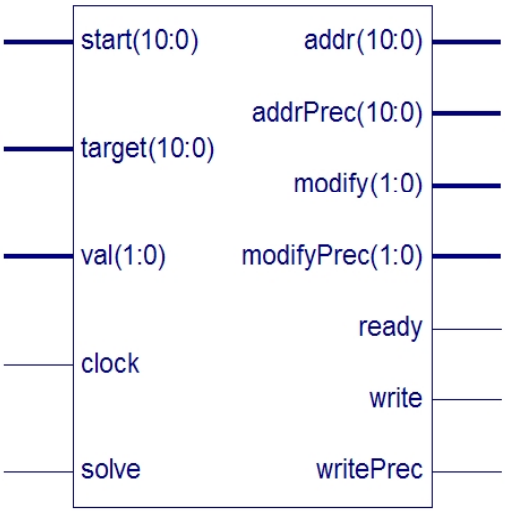

Fig.3. BF component

The signals interact with the required memory for applying the BF algorithm, while some of the signals (*clock*, *solve* as inputs and *ready* as output) are signals that belong to the hardware configurations and interconnections.

To implement the memory, FPGA Virtex BlockRAMs were used. This way, the speed of the configuration increases because specialized components are used, and also the number of available Virtex slices makes it possible to implement larger designs.

As mentioned in Section II, the BF algorithm uses a structure that retains all the neighbors of the current element. In our case this structure is implemented in hardware by an array, and uses more auxiliary array signals, all stored in BlockRAMs. BF uses the *start* and *target* signals to identify the position in the neighbors' structure. It also contains a *direction* signal, which is the most important, as it points to the direction in which the algorithm searches for empty

spaces. This signal is looped in the 0-3 sequence, pointing to the 4 directions.

As a final result, BF will set the algorithm path to Red. Spreading will be marked with Green. The starting point is Blue, and the goal is Yellow. Here Black is for free spaces and White is for obstacles.

The MarkPath component is the one that implements the reconstruction of the path after the algorithm has been applied (Fig.4.).

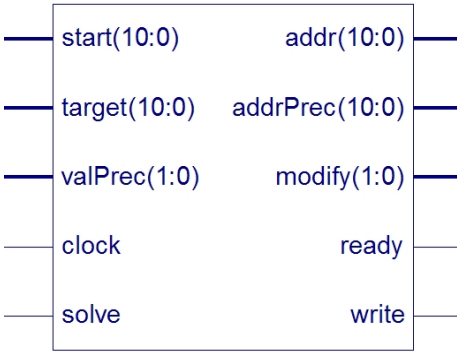

Fig.4. MarkPath Component

In hardware, no initialization of values is made automatically, therefore it is required that before any algorithm is applied, the system is brought to an initial known state, which must be a beginning state for the algorithm. Every signal must be initialized, and the memory map also. The component that deals with all these and with other clock synchronization and enable / ready types of signals is called CentralUnit (Fig.5).

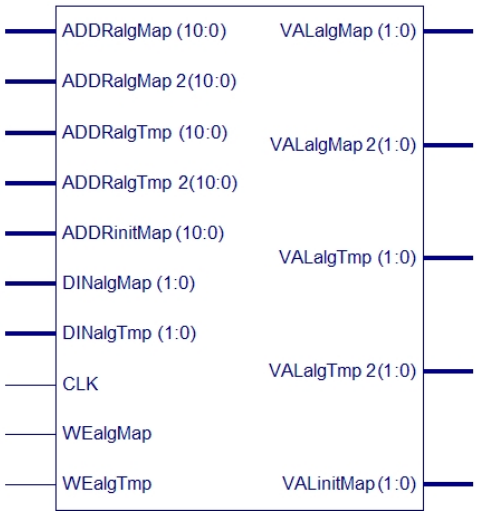

Fig.5. CentralUnit Component

The Serial component implements the interface that assures the receiving and transmission of the information from / to the computer through the serial port connection. The data format use: 1 start bit; 8 data bits; 1 parity bit; 1 stop bit. The environment map is received and the path given by the algorithm is transmitted (Fig.6.).

Fig.6. Serial Component

The VGA module contains the necessary signals to synchronize the output on a regular monitor (the VGA controller that provides the image on an 800x600 pixels screen display). The component captures the value from the matrix, which represents the type of cell to be displayed, and it outputs the color corresponding to the value (Fig.7). The matrix acts like a video memory for the component.

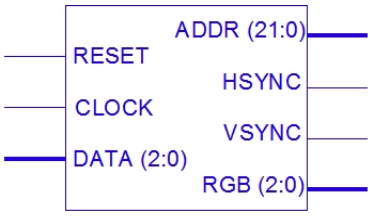

Fig.7. VGA Component

2) Integrated system: The architecture of integrated system has been designed as the collection of components. The connection between components is made by relatively simple Finite State Machines (FSM's), by signals, and with a few processes that handle the synchronization, command all the components, acquire signals from board inputs, and send data to outputs (Fig.8.).

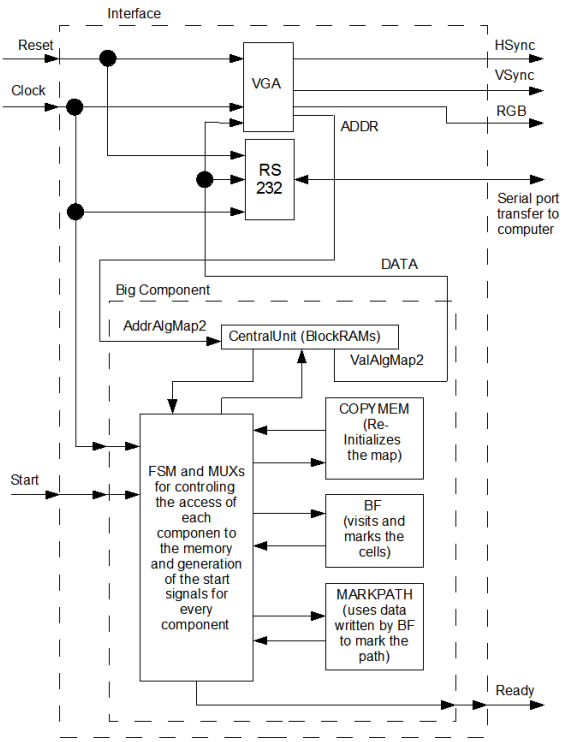

Fig.8. Integrated system

B. Hardware Results

To test the hardware implementation we used the same maps that were used for the software implementation. The system was tested several times to verify its reliability.

The path found after BF algorithm execution was displayed on the monitor and was photographed in different stages of evolution. We used these pictures to compare them with the results from the software implementation.

The path is also saved in a file on the computer, after transferred through the serial port, and is used in the navigation system of the mobile robot.

At the beginning, after running Xilinx Synthesis software for our hardware implementation, the report indicated a maximal working frequency of 185 MHz (clock period: 5.4 ns).

The space available in FPGA Virtex devices allows the implementation of a large matrix (e.g. 100x100). Our maps use variable dimension matrices. The report generated by Xilinx synthesis tool shows that the FPGA circuit is used only at a small fraction of its capacity. For larger maps the amount of BlockRAMs increases by a square function. Therefore, if larger maps can not be stored into the BlockRAMs we can also use the external DynamicRAM.

We used the simulation environment ModelSIM XE III 6.1 [11] to test if our hardware design has a proper functioning. The waveforms generated in simulation helped us in choosing the working frequency. We decided to make the first test at the 120 MHz frequency.

The images taken from the hardware implementation of BF algorithm show the map on the monitor. In Fig. 9, 10, 11, 12, 13 maps with different dimensions are used to verify that the algorithm obtains the correct path and to check how much time is required for that.

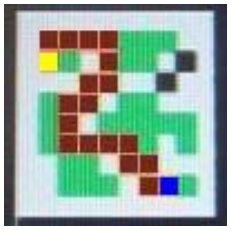

Fig.9. Result for 10x10 map dimension

Fig.10. Result for 30x30 map dimension

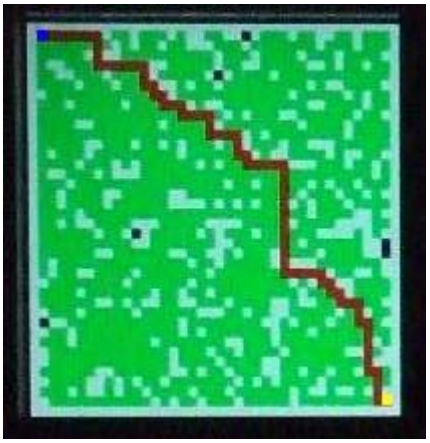

Fig.11. Result for 40x40 map dimension

Fig.12. Result for 50x50 map dimension

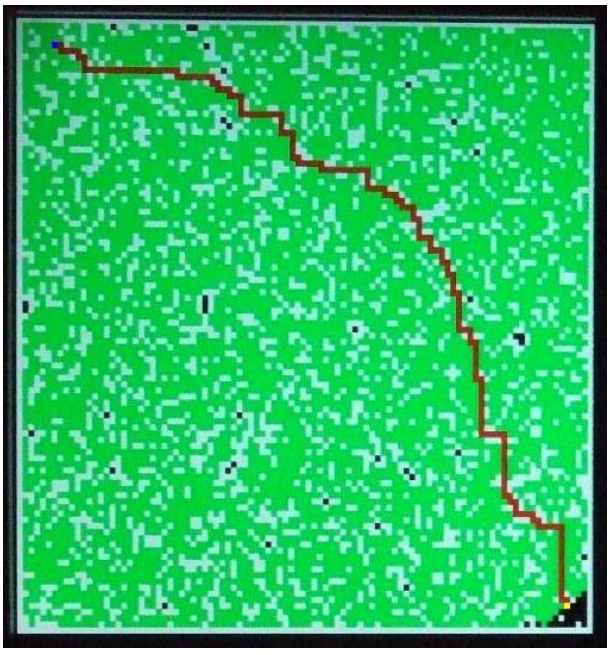

Fig.13. Result for 100x100 map dimension

After obtaining these encouraging results we continued testing with 150 MHz frequency. This affects neither the path found, nor the implemented algorithm, only the total running time, which decreases proportionally to the difference between the working frequencies.

Table 2 presents samples of algorithm execution times (in μs) in our hardware implementation at the 120 MHz working frequency and Table 3 shows the execution times at the 150 MHz frequency.

Freq	Width	Height	Total	Obstacles Fill	Spreading	Time (μs)
120 Mhz	10	10	100	25%	High	2.78
	30	30	900	25%	High	34.95
	40	40	1600	50%	High	73.81
	50	50	2500	50%	High	112.15
	100	100	10000	50%	High	475.38

Table 2. Execution times at 120 MHz frequency

Freq	Width	Height	Total	Obstacles Fill	Spreading	Time (μs)
150 Mhz	10	10	100	25%	High	2.35
	30	30	900	25%	High	28.2
	40	40	1600	50%	High	59.54
	50	50	2500	50%	High	90.14
	100	100	10000	50%	High	380.3

Table 3. Execution times at 150 MHz frequency

IV. Comparison between implementations and Conclusion

The different results between the two kinds of implementation appear for the run time of the algorithm. Generally, all the measured times were at least two orders of magnitude better in hardware than in software (Table 4).

Width	Height	Total	Obstacles Fill	Spreading	Software Time (μs)	Hardware Time (μs)
10	10	100	25%	High	10000	2.35
30	30	900	25%	High	10000	28.2
40	40	1600	50%	High	10000	59.54
100	100	10000	50%	High	40000	380.3

Table 4. Execution times in software and hardware

The paths obtained after running the algorithm in hardware were the same as in software. This states that the implementation of the algorithm is valid.

Paths obtained by both implementations are very good or optimal. As for the time criterion, for large dimension maps the hardware implementation is preferable because of the much better running times obtained. Our results demonstrate that the hardware-level solution for path-planning algorithms is hundreds of times faster and proves to be a serious alternative in speed to usual software solutions.

The board with FPGA device is proper to use for mobile robot applications. It is now possible to take over a part of the necessary control system of mobile robots, which can be executed in FPGA.

The development of such hardware implementations is more difficult because of the high degree of details we need to cover. But the design using reusable components increases system adaptability and reduces the time spent with the implementation. Software implementations provide flexible solutions, easy to implement, manage and maintain. The costs of implementing software solutions are small, while solutions developed on hardware require greater costs associated to the boards. Nevertheless hardware solutions provide speeds that cannot be achieved by any means in software, while the same board can be used to perform other tasks, too.

In our future research, we intend to make hardware implementations for others used path-planning algorithms, and used them for the mobile robot navigation. We will make implementation for continuous maps too.

Acknowledgment

This work was supported by the Romanian Ministry of Education and Research, under grant type A, no. 1566/2007.

References

[1] T. Arai, E. Pagello, and L. E. Parker, "Advances in Multi-Robot Systems", *IEEE Trans. Robot. Autom.,* vol. 18, no.5, pp. 655-661, October, 2002.

[2] G. Dudek, and M. Jenkin, *Computational Principles of Mobile Robotics*, Cambridge University Press, UK, 2000, pp. 121-148.

[3] R. Siegwart, and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, The MIT Press, Cambridge, MA, 2004, pp. 258-290.

[4] M. Grieger. (2004, Nov.). A parallel implementation of path planning on reconfigurable hardware. Bielefeld University. Germany. [Online], Available: http://www.ti.uni-bielefeld.de/html/publications/diploma_theses/index.html.

[5] R. H. Katz, and G. Borriello, *Contemporary Logic Design*. 2nd ed., Pearson Prentice Hall, Upper Saddle River, NJ, 2005, pp. 421-451.

[6] M. Tommiska. (2005, March). Applications of Reprogrammability in Algorithm Acceleration. Helsinki University of Technology. Finland. [Online], Available: http://lib.tkk.fi/Diss/2005/isbn9512275279/isbn9512275279.pdf

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990, pp. 400-472.

[8] Java API, [Online]. Available: http://www.java.sun.com/reference/api

[9] Eclipse IDE, [Online]. Available: http://www.eclipse.org

[10] The Xilinx ISE 8.1i Environment, [Online]. Available: http://www.xilinx.com/support

[11] ModelSIM XE III 6.1, [Online]. Available: http://www.model.com