IMPLEMENTATION OF GAME OF LIFE ON FPGA

Submitted by
Adrian Sarbu
Sumit Ahuja

I. INTRODUCTION

This project implements the Game of life on Altera FPGA EP20K200EFC484. Game of Life works with the principle that a cell got birth when it has three neighbors; it dies when it has more then three neighbors or less then two neighbors and cell survives when it has two or three neighbors. These patterns can be realized using finite automata principles.

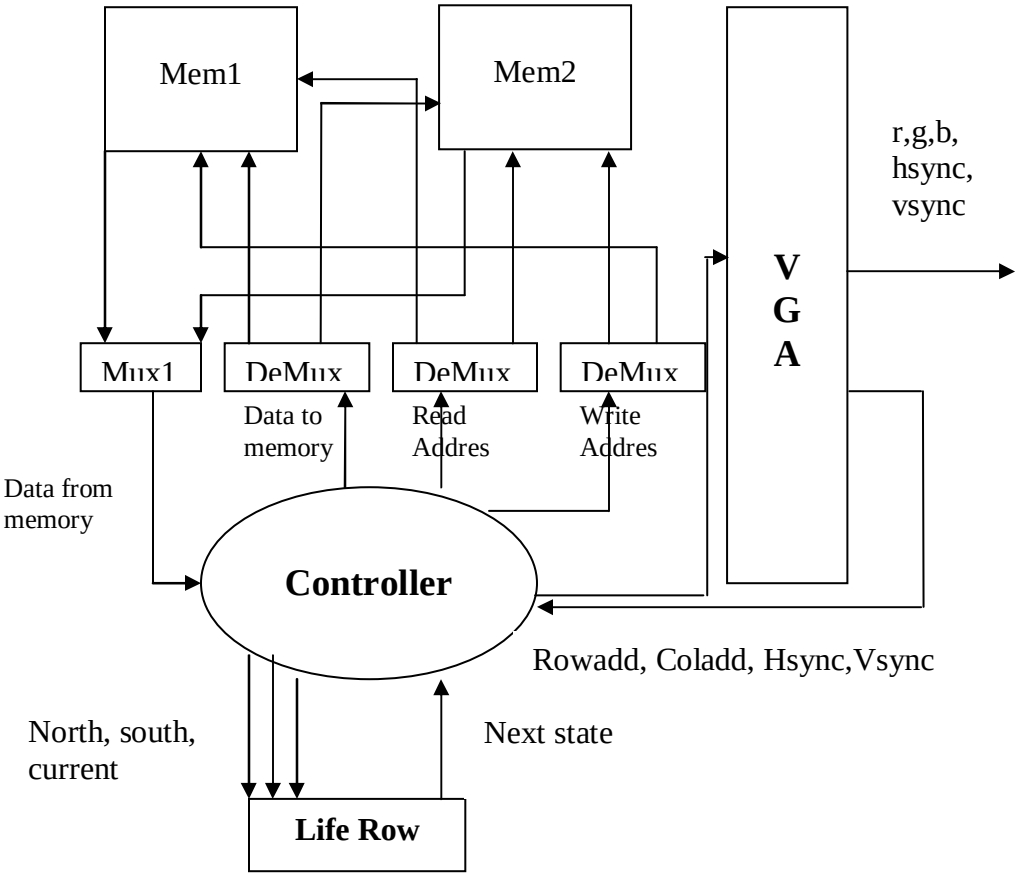
There can be many ways of implementation to display game of life using VGA. One of the implementation is totally parallel in which all the 8 X 8 cell states are calculated on single cycle. From there it is transmitted to video memory and from there vga reads and display the patterns.

In this project, we have used two different ways of implementation, In first we are using two memories (the internal memory provided by Altera) while in second implementation we are using the flip flops inside the controller units instead of using the memories. We are making the dedicated hardware for a row of the display units. This architecture, which is not fully parallel implementation, writes one row (Life row) at the video memory. Next sections discuss how we have implemented the game of life.

II. IMPLEMENTATION WITH TWO MEMORIES

In our implementation we are implementing the Life row in hardware that is connection of eight cells in horizontal direction as hardwired. So we can calculate the next eight states in one clock cycle. For synchronizing with vga we are using vsync signal from the output of the vga. The topmost view of the design can be viewed in figure1.

In our design we are using two memory units, VGA, Life row and multiplexers. Two different memories are used for the write and read purposes. From first memory, the previous states are read and on second the next states are written. When eight rows are written the memories are swapped so the memory which was initially reading now the controller will be writing on that memory and vice versa.



GAME OF LIFE

Figure1. Schematic for Implementation of Game of Life.

II(a). Result of first Implementation: The table below shows the Area results we have obtained on Altera EP20K200EFC484.

Area Report:

Resource	Used	Available Resources	Utilization
IOs	9	136	6.62%
LCs	538	8320	6.47%
Memory Bits	1024	106496	0.96%

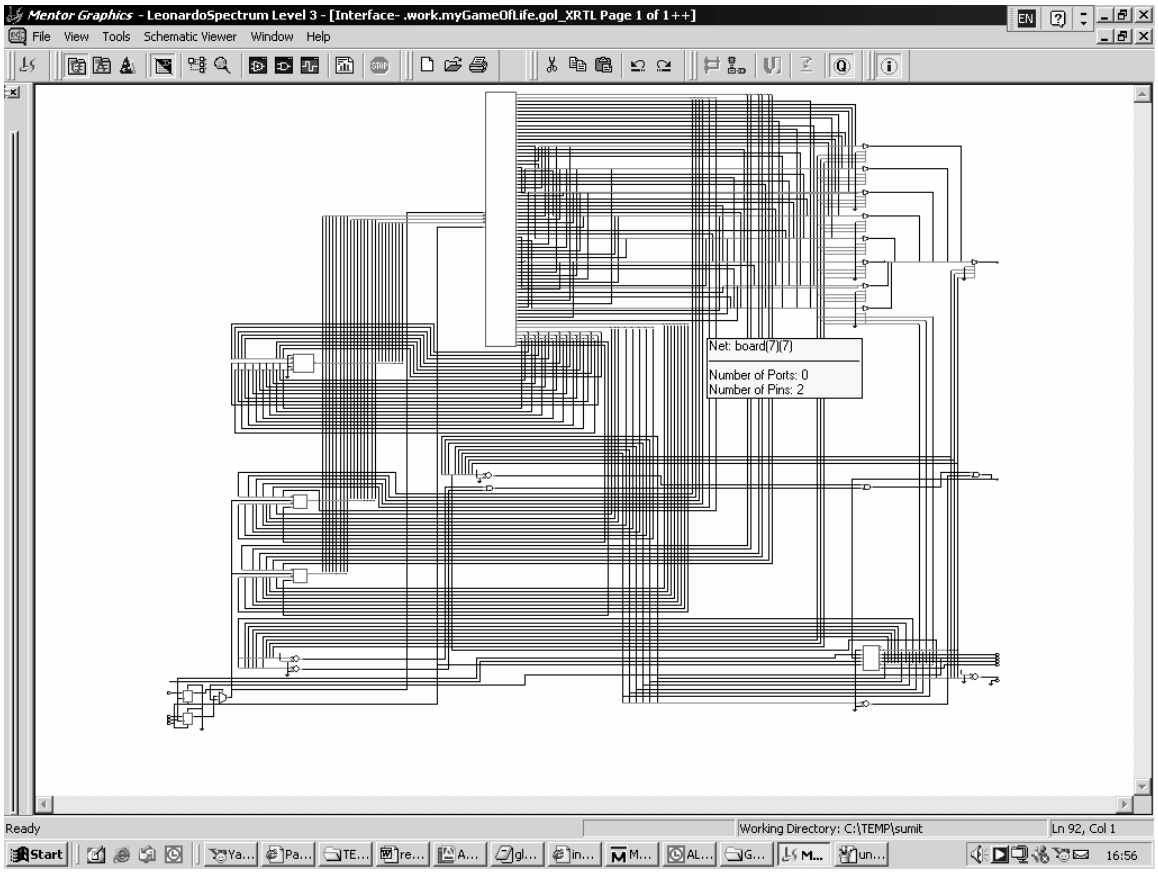


Figure 2 Implementation of Game of Life on Leonardo Spectrum with two memories

III. SECOND IMPLEMENTATION

In second Implementation instead of using two memories we are using flip flops to process the data i.e. the data which is written from the Life row to the one bank of the flip flops and read from the bank of flip flops instead of using memories.
Figure below shows the schematic and implementation style used for the implementation of Game of Life.

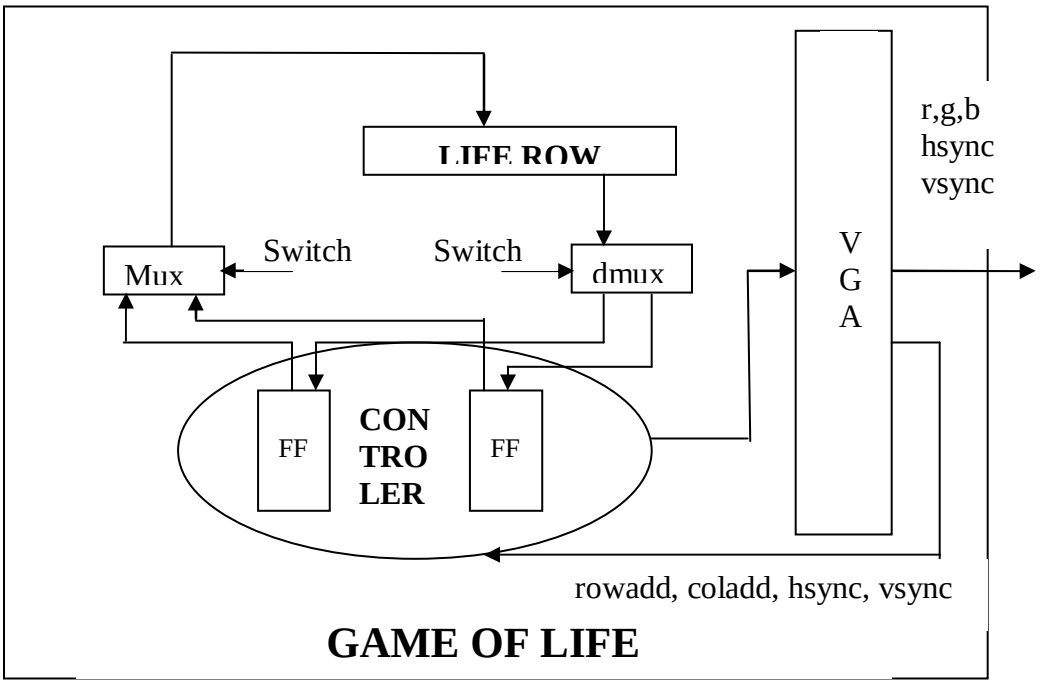


Figure3. Schematic for second Implementation of Game of Life.

III(a). Results of Second Implementation: The table below shows the Area results we have obtained on Altera EP20K200EFC484.

Area Report:

Resource	Used	Available Resources	Utilization
IOs	7	136	5.15%
LCs	649	8320	7.80%
Memory Bits	0	106496	0.00%

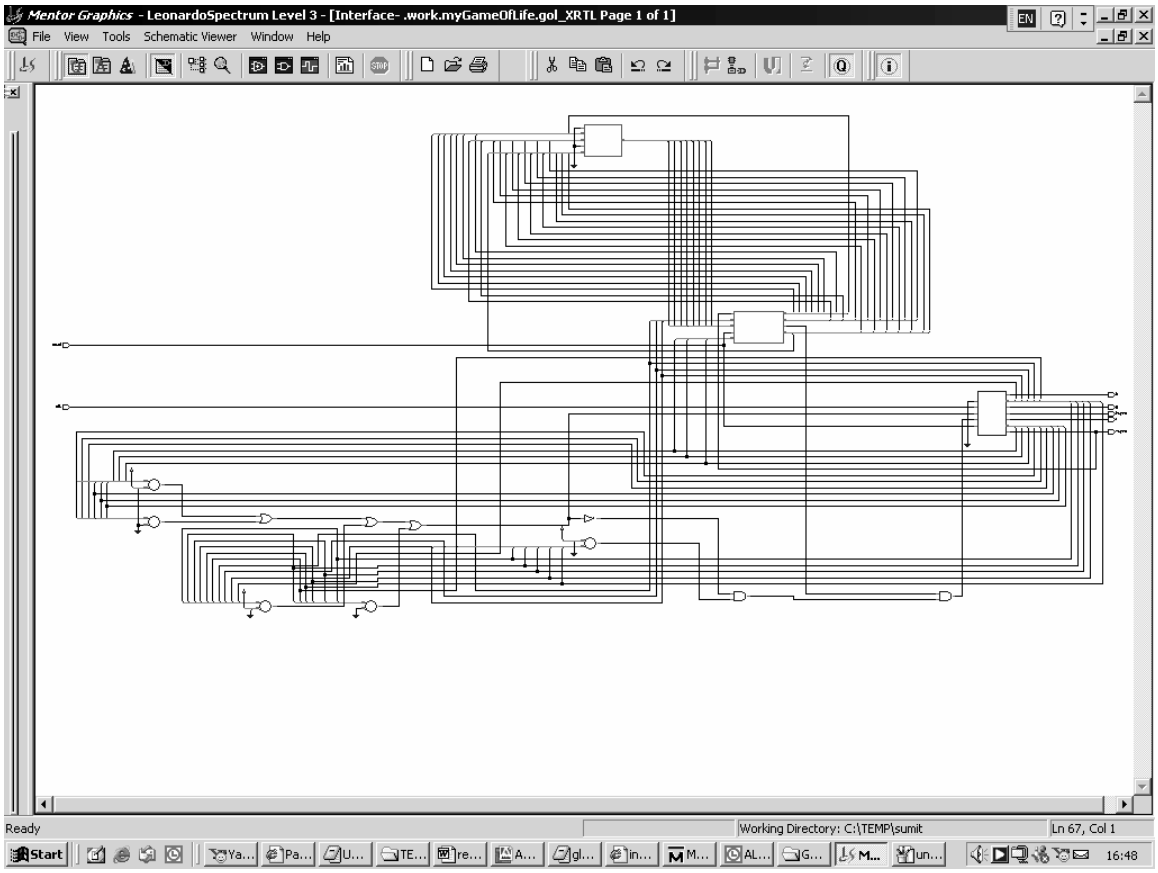
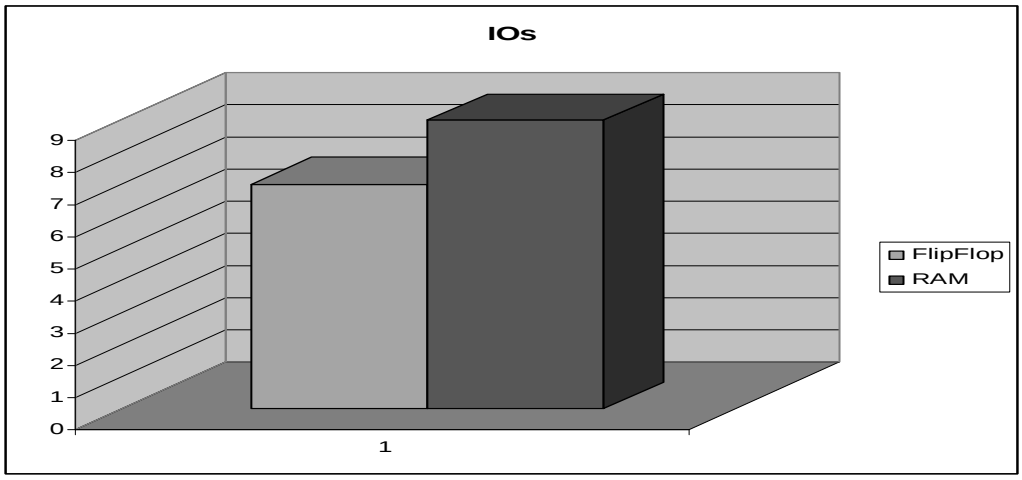


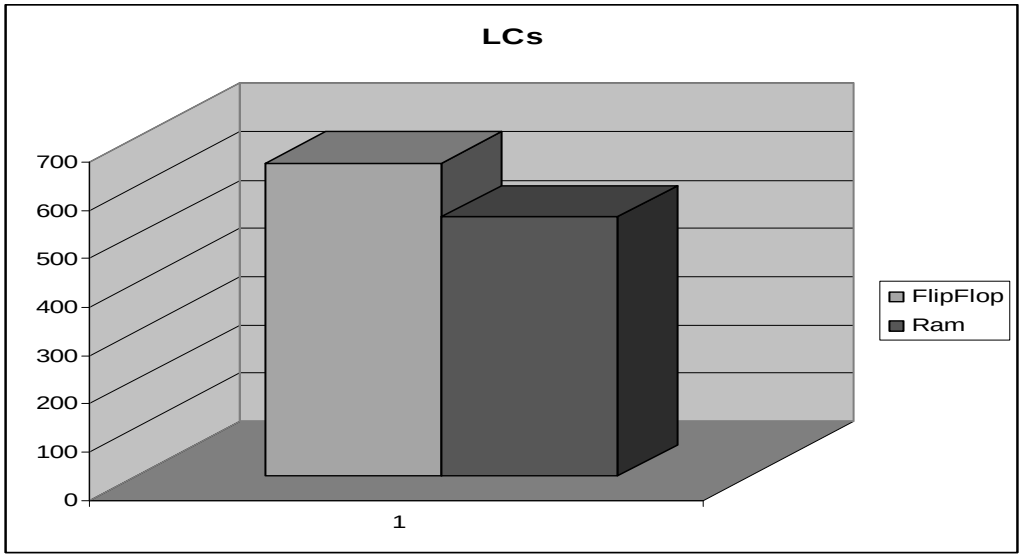
Figure 4 Implementation of Second implementation with Leonardo Spectrum.

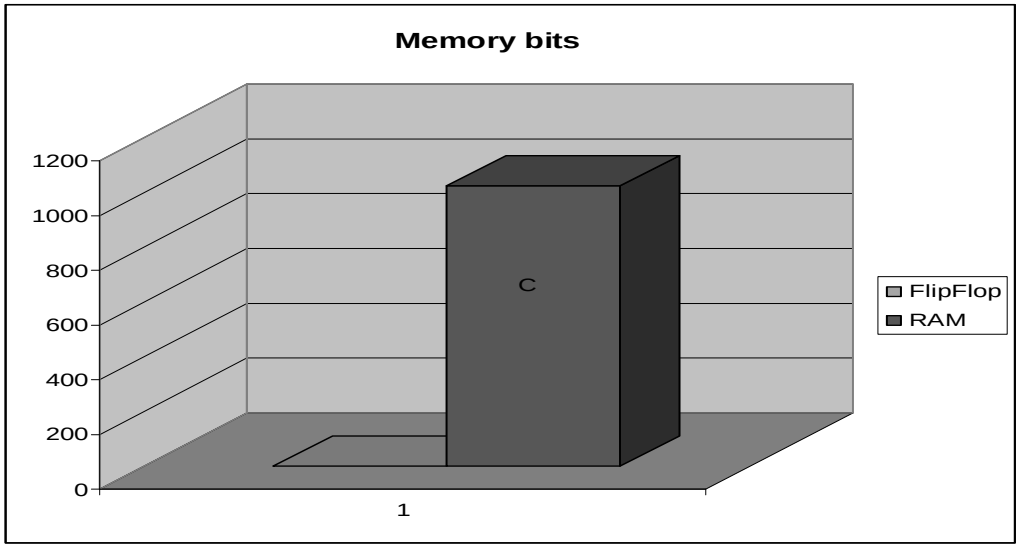
IV. RESULTS AND COMPARISON:

Among the two implementations the first implementation is more useful as it uses the internal Memory provided by Altera. The second implementation uses the flip-flops for storing the current and next state. While flip-flop based implementation functioned correctly when downloaded into the Altera FPGA, synthesis of RAM based implementation was faulty and did not produced the expected results on the FPGA. Both versions showed the exact same behavior while simulated under ModelSim but something happens during synthesis that renders the RAM based implementation non-functional. We could not continue our investigation as to why this is happening because the software licenses expired soon after we started working.

The graphs hereafter show the comparative resources requirements of the two implementations:


IOs
9
8
7
6
5
4
3
2
1
0
1
FlipFlop
RAM


LCs
700
600
500
400
300
200
100
0
1
FlipFlop
Ram


Memory bits
1200
1000
800
600
400
200
0
C
1
FlipFlop
RAM