Conway's Game of Life

On the 8-bit Picoblaze Microcontroller

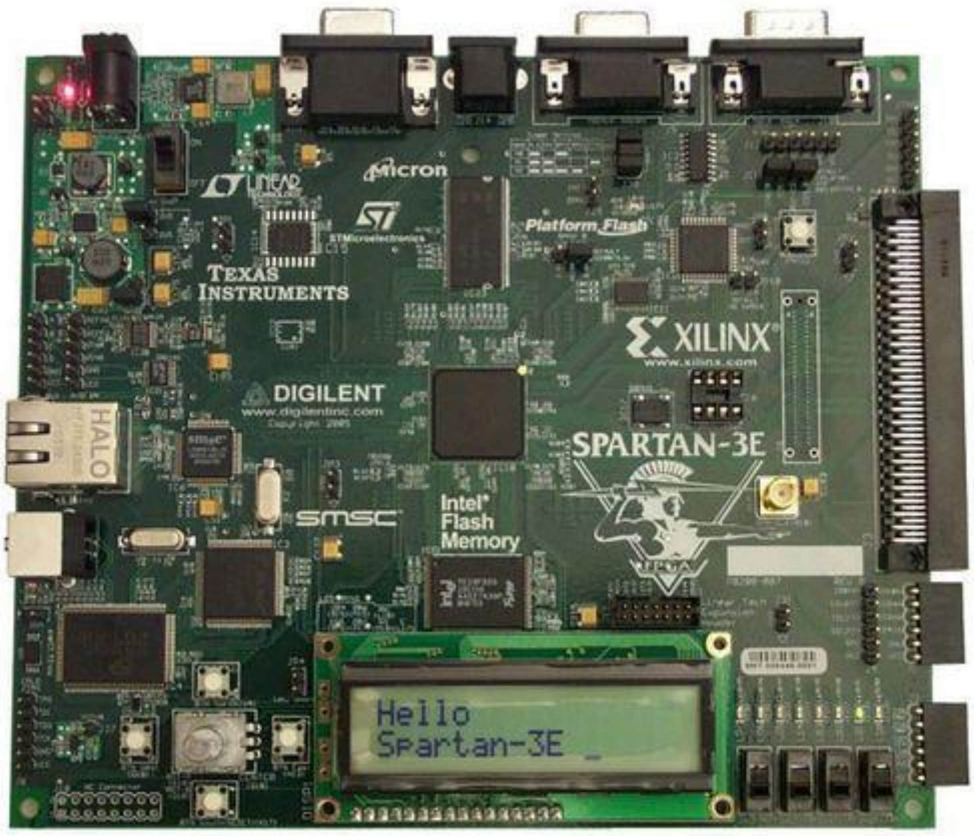
Technical Documentation

Michał Loska

29.11.2020

1. Introduction

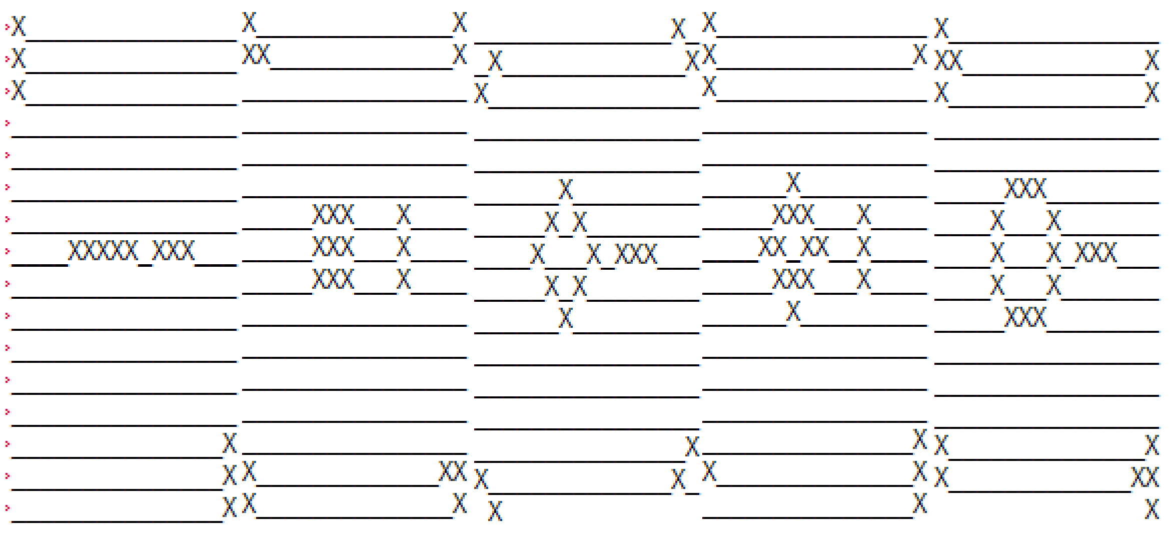
Conway’s Game of Life was implemented in Assembly language in Picasso IDE/Simulation Environment with full support for Xilinx Spartan 3E Starter Board.



Xilinx Spartan 3E

The idea behind this project is to simulate a Cellular Automata graphically using the UART (TX) interface to output the visualization. Execution of the code is still possible without the Microcontroller as the Simulation Environment can simulate the UART communication.

Pictures below represent 5 iterations of the program execution:



```
X_______________ X______________X ______________X_ X_______________ X_______________
X_______________ XX_____________X _X_____________X X______________X XX_____________X
X_______________ ________________ X_______________ X_______________ X______________X
________________ ________________ ________________ ________________ ________________
________________ ________________ ________________ ________________ ________________
________________ ________________ ______X_________ ______X_________ _____XXX________
________________ _____XXX___X____ _____X_X________ _____XXX___X____ ____X___X_______
____XXXXX_XXX___ _____XXX___X____ ____X___X_XXX___ ____XX_XX__X____ ____X___X_XXX___
________________ _____XXX___X____ _____X_X________ _____XXX___X____ ____X___X_______
________________ ________________ ______X_________ ______X_________ _____XXX________
________________ ________________ ________________ ________________ ________________
________________ ________________ ________________ ________________ ________________
________________ ________________ ________________ ________________ ________________
_______________X ________________ _______________X _______________X X______________X
_______________X X____________XX X_____________X_ X______________X X_____________XX
_______________X X______________X _X______________ _______________X _______________X
```

5 Iterations of program execution

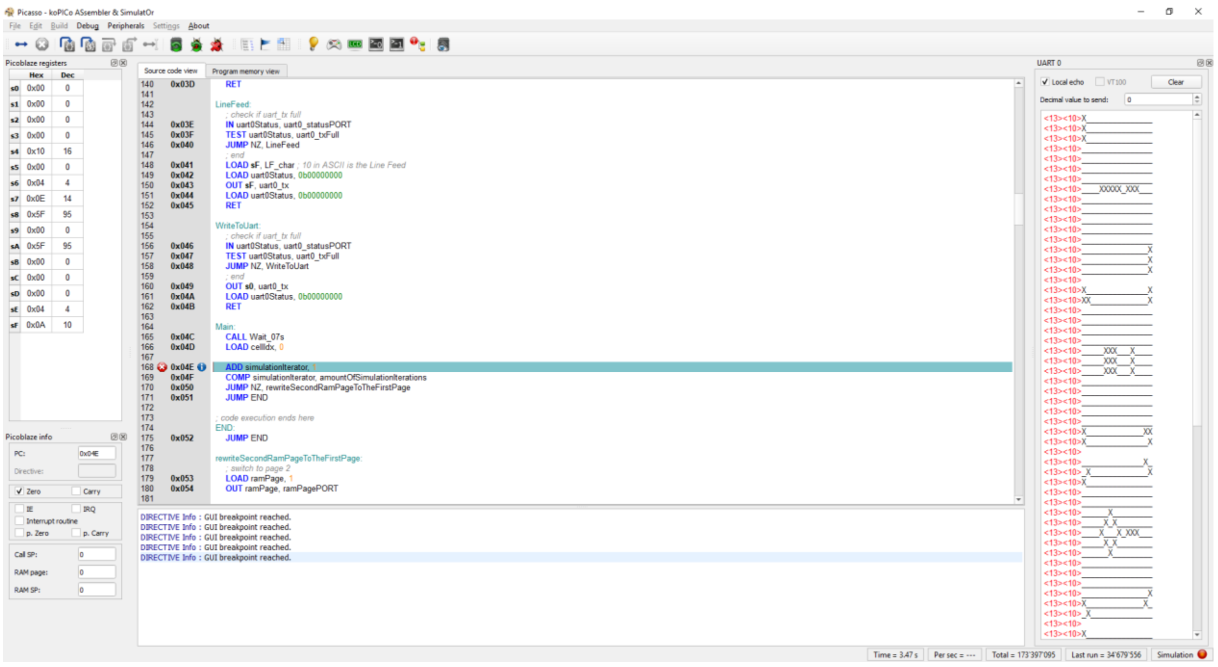
2. Initial Requirements

The initial project requirements were:

- Storing and Fetching board data in/from RAM
- Implementation of UART Interface
- Fixed board sized 16x16 (one full 256B page of RAM)
- Predefined initial pattern of alive cells
- Edges and corners are interconnected
- Program runs for the predefined amount of iterations
- "_" and "X" are used to represent dead and alive cells

3. Implementation

Project has been Implemented within Picasso overlay of Picoblaze Assembly language with use of Picasso Simulator.


Picasso - koPICo ASsembler & SimulatOr
File Edit Build Debug Peripherals Settings About
Source code view
Program memory view
UART 0
Local echo
VT100
Clear
Simulation

Picoblaze registers

	Hex	Dec
s0	0x00	0
s1	0x00	0
s2	0x00	0
s3	0x00	0
s4	0x10	16
s5	0x00	0
s6	0x04	4
s7	0x0E	14
s8	0x5F	95
s9	0x00	0
sA	0x5F	95
sB	0x00	0
sC	0x00	0
sD	0x00	0
sE	0x04	4
sF	0x0A	10

Picoblaze info

PC:	0x04E
Directive:	
✓ Zero	Carry
IE	IRQ
Interrupt routine	
p. Zero	p. Carry
Call SP:	0
RAM page:	0
RAM SP:	0

```
0x03D       RET

        LineFeed:
            ; check if uart_tx full
0x03E       IN uart0Status, uart0_statusPORT
0x03F       TEST uart0Status, uart0_txFull
0x040       JUMP NZ, LineFeed
            ; end
0x041       LOAD sF, LF_char ; 10 in ASCII is the Line Feed
0x042       LOAD uart0Status, 0b00000000
0x043       OUT sF, uart0_tx
0x044       LOAD uart0Status, 0b00000000
0x045       RET

        WriteToUart:
            ; check if uart_tx full
0x046       IN uart0Status, uart0_statusPORT
0x047       TEST uart0Status, uart0_txFull
0x048       JUMP NZ, WriteToUart
            ; end
0x049       OUT s0, uart0_tx
0x04A       LOAD uart0Status, 0b00000000
0x04B       RET

        Main:
0x04C       CALL Wait_07s
0x04D       LOAD cellIdx, 0

0x04E       ADD simulationIterator, 1
0x04F       COMP simulationIterator, amountOfSimulationIterations
0x050       JUMP NZ, rewriteSecondRamPageToTheFirstPage
0x051       JUMP END

        ; code execution ends here
        END:
0x052       JUMP END

        rewriteSecondRamPageToTheFirstPage:
            ; switch to page 2
0x053       LOAD ramPage, 1
0x054       OUT ramPage, ramPagePORT
```

```
DIRECTIVE Info : GUI breakpoint reached.
DIRECTIVE Info : GUI breakpoint reached.
DIRECTIVE Info : GUI breakpoint reached.
DIRECTIVE Info : GUI breakpoint reached.
DIRECTIVE Info : GUI breakpoint reached.
```

Decimal value to send: 0

```
<13><10>X_______________
<13><10>X_______________
<13><10>X_______________
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>____XXXXX_XXX___
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>_______________X
<13><10>_______________X
<13><10>_______________X
<13><10>
<13><10>X______________X
<13><10>XX_____________X
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>_____XXX___X____
<13><10>_____XXX___X____
<13><10>_____XXX___X____
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>X_____________XX
<13><10>X______________X
<13><10>
<13><10>______________X_
<13><10>_X_____________X
<13><10>X_______________
<13><10>________________
<13><10>________________
<13><10>______X_________
<13><10>_____X_X________
<13><10>____X___X_XXX___
<13><10>_____X_X________
<13><10>______X_________
<13><10>________________
<13><10>________________
<13><10>________________
<13><10>_______________X
<13><10>X_____________X_
<13><10>_X______________
<13><10>
<13><10>X_______________
```

Time = 3.47 s | Per sec = --- | Total = 173'397'095 | Last run = 34'679'556

Simulation Environment Debugging Window

3.1 Initialization and defining stored values in RAM

The very beginning of the program consists of initialization of the 2 first pages of the RAM memory. Then the pages are filled with the value 95 which represents "_" character in the ASCII code. This character represents a dead cell. After this step an arbitrary pattern of alive cells represented by value 88 ("X" in ASCII) is set on the first page of the RAM memory.

```
; Set the initial alive cells (in RAM memory blocks)
LOAD s0, aliveCell
STORE s0, 0
STORE s0, 16
STORE s0, 32
```

Defining the initial pattern (numbers above represent RAM addresses)

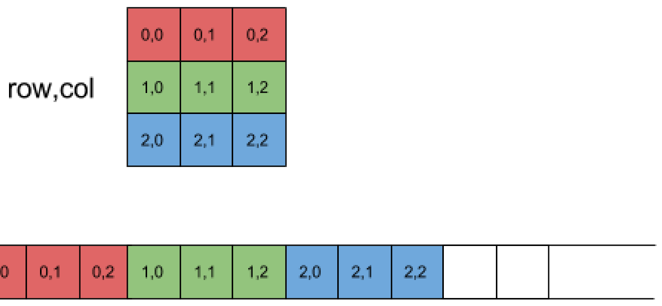
3.2 Main Loop

Program has its main loop in which the most logic happens such as:

1. Reading corresponding value from corresponding RAM address
2. Writing the value to UART TX
    a. Making sure UART_TX is not full
3. Count neighbors of the cell under the fetched RAM address
    a. Check if the current cell is a corner of the board
        i. Execute further logic specific to this case
    b. If not, check if the current cell is along side of the left or right edge
        i. Execute further logic specific to this case
    c. If none of the above was true
        i. Execute further default logic
    d. Check if the current cell is alive or dead
    e. If alive:
        i. DIES:
            1. Has 0 neighbors
            2. Has 1 neighbor
            3. Has 4 or more neighbors (overpopulation)
        ii. SURVIVES
            1. Has 2 or 3 neighbors
        iii. Overwrite alive cells to the next iteration
    f. If dead:
        i. Is born:
            1. Has 3 neighbors
4. Incrementation of regarding iterators in order to keep track of the current position in the code
5. Depending on the values of the iterators we continue or finish traversing the board
6. If the predefined amount of iterations is higher than the current iteration number then go to point 1 and go over again
    a. If the iterator is at the ending bound – STOP THE PROGRAM

3.3 Board Traversal

The addressing of a single RAM Page starts at number 0 and ends at 255 and it continues from row to row:



Logical RAM Representation

Incrementing of the iterator will start at 0 and end at 255 and the traversal will be conducted row by row from the top to the bottom. In order to

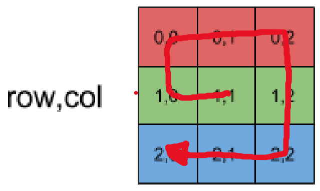
3 iterators were implemented:

- rowIdx – keeps the number of the currently traversed row
- isAtTheEOL – keeps the number of current column (is zeroed at the end of each row)
- cellIdx – keeps the number of the current cell in RAM (RAM block address)

3.4 Counting Neighbors

In every iteration a value under a given addresses in RAM is fetched. The previously mentioned iterators help distinguish which function has to be used for neighbor counting depending on the cell position.

All the corner cases use predefined addressing to check the neighbors and all the other functions use the clockwise order starting at the left-middle neighbor:



Orded of Neighbor counting

Where the edge cases need to jump to the opposite edges to fetch their neighbor's data.

3.5 Rewriting RAM pages

After all the actions mentioned above will have executed the second RAM page is rewritten to the first page. The second page plays the role of the temporary container for the upcoming iteration.

4. Summary:

Project was designed away from the actual hardware and was only tested in the supplied simulator. The proper functioning of this program can be checked in any online simulator.

Git Repo: https://github.com/michalloska/GameOfLife_Assembly