MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION
FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
“NOVOSIBIRSK NATIONAL RESEARCH UNIVERSITY STATE UNIVERSITY”

(NOVOSIBIRSK STATE UNIVERSITY, NSU)

15.03.06 - Mechatronics and Robotics
Focus (profile): Artificial intelligence

TERM PAPER

Polyansky Vladislav
Seigel Arina
Zyablov Vladimir

The theme of the paper:

“Snake Game”

Novosibirsk, 2024

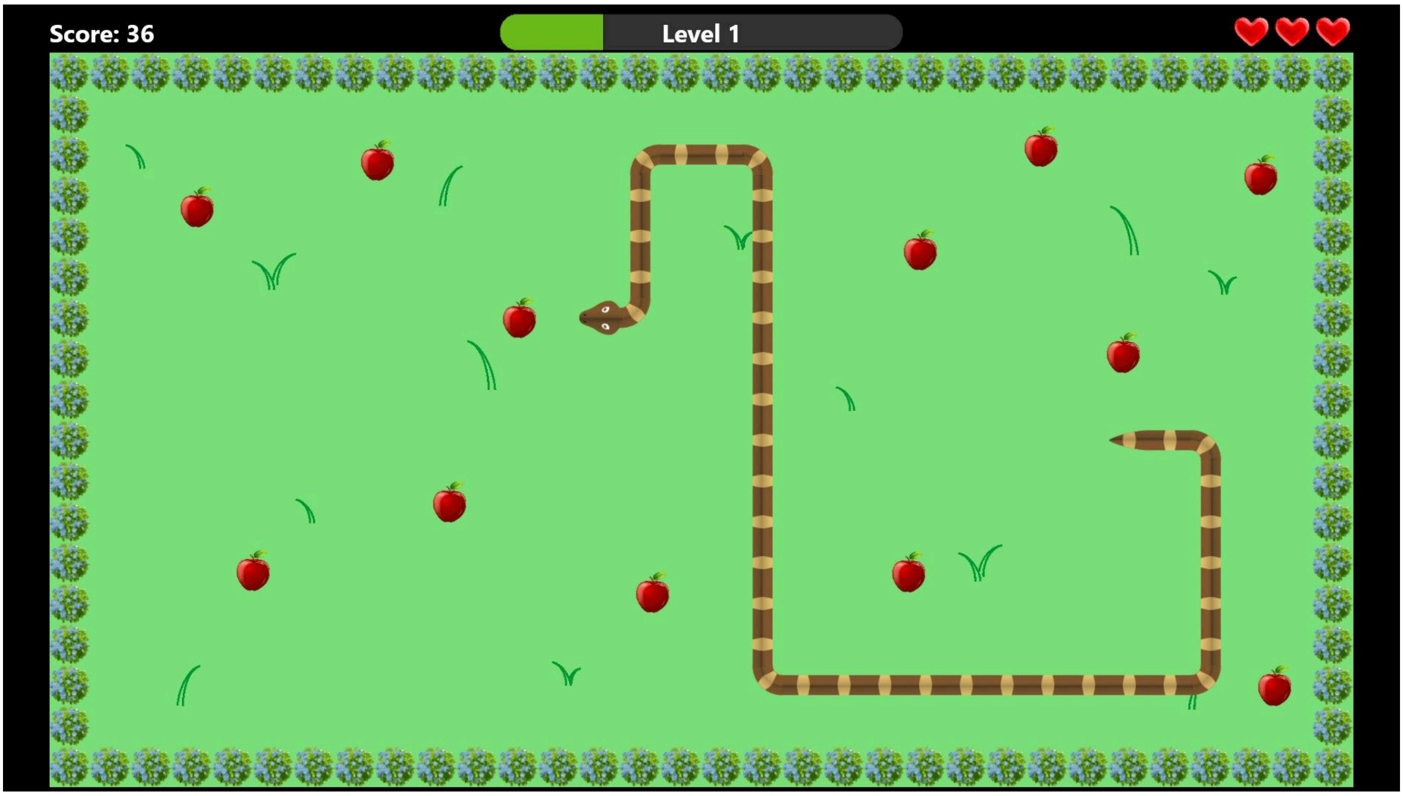
Contents

Novosibirsk, 2024

Introduction

Our goal was to recreate the iconic game "Snake" by bringing our own touch to the gameplay. To bring our idea to life, we used Logisim along with the CDM-8 processor simulator. In addition, we used CocoIDE as a coding platform.

Analogues



When creating our project we looked at several versions of the game Snake. The first is the original version of the game Snake known to all and the second is a more modern version of this game.

- This game is a modern version of the classic snake game. Players control a snake that grows by eating different foods such as apples, tomatoes, cherries and cakes. They must avoid walls and obstacles to stay alive as the snake gets longer and faster. Advancing to the next levels introduces new challenges such as different foods and bigger obstacles. Players can climb levels, increasing their score, and encounter surprises such as earthquakes that bring bonus points. Controls are simple: players swipe in the right direction to move the snake.

Concept

The concept of this game is as follows: the player controls a snake that moves around the playing field, where food randomly appears. When the snake eats the food, it becomes longer and the player gets five point. The game ends if the snake collides with its own tail. It is important that the snake does not move "inside itself", that is, that it does not rotate 180 degrees.

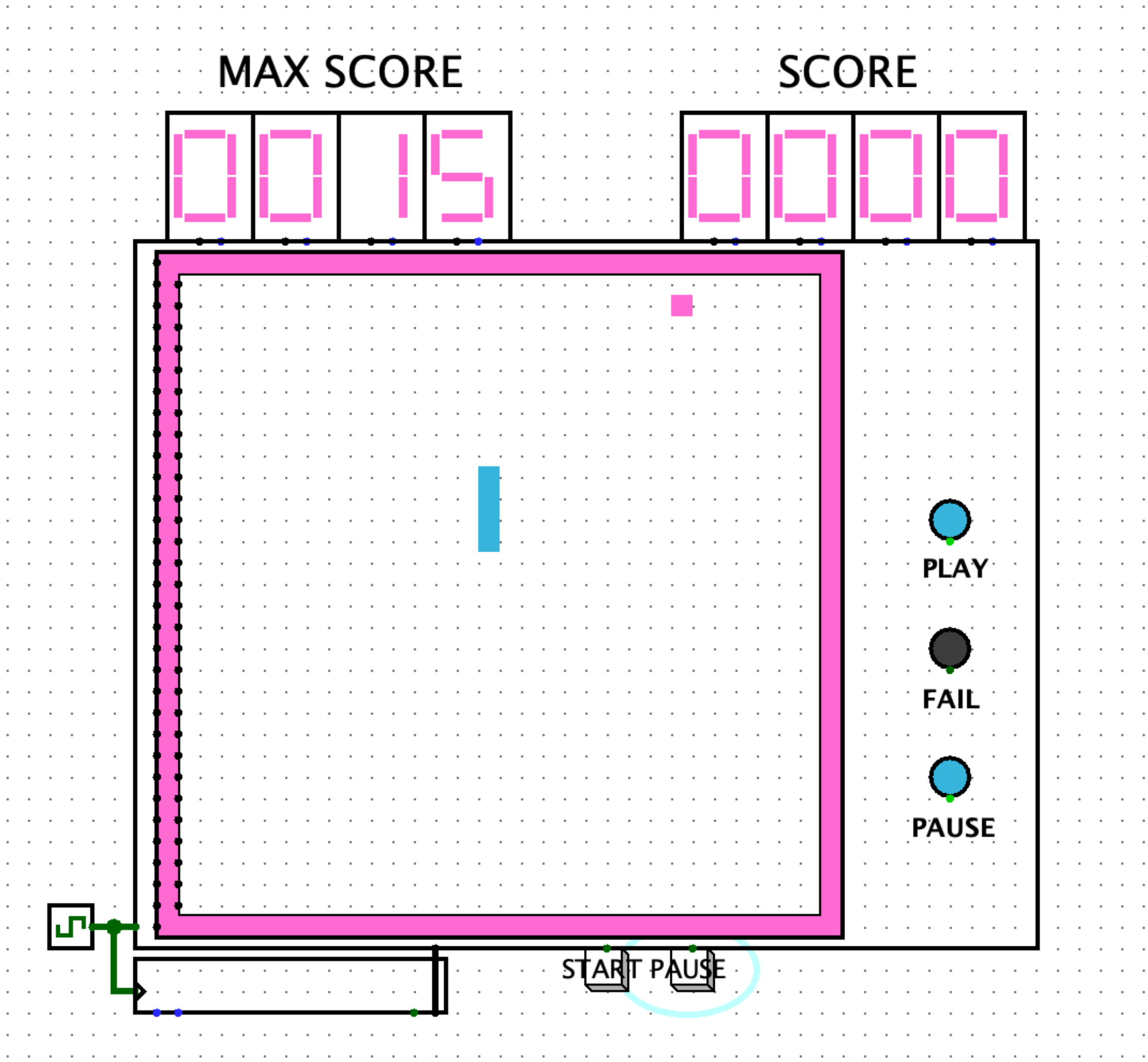
Project Objectives

The project objectives include creation of screens for displaying the playing field, creation of a snake, realization of its movement and generation of apples. Creation of a score counter, realization of interaction with the CDM-8 processor, connection of additional Logisim modules to it, as well as realization of the game concept with the help of software and hardware of the project.

Hardware

Start screen

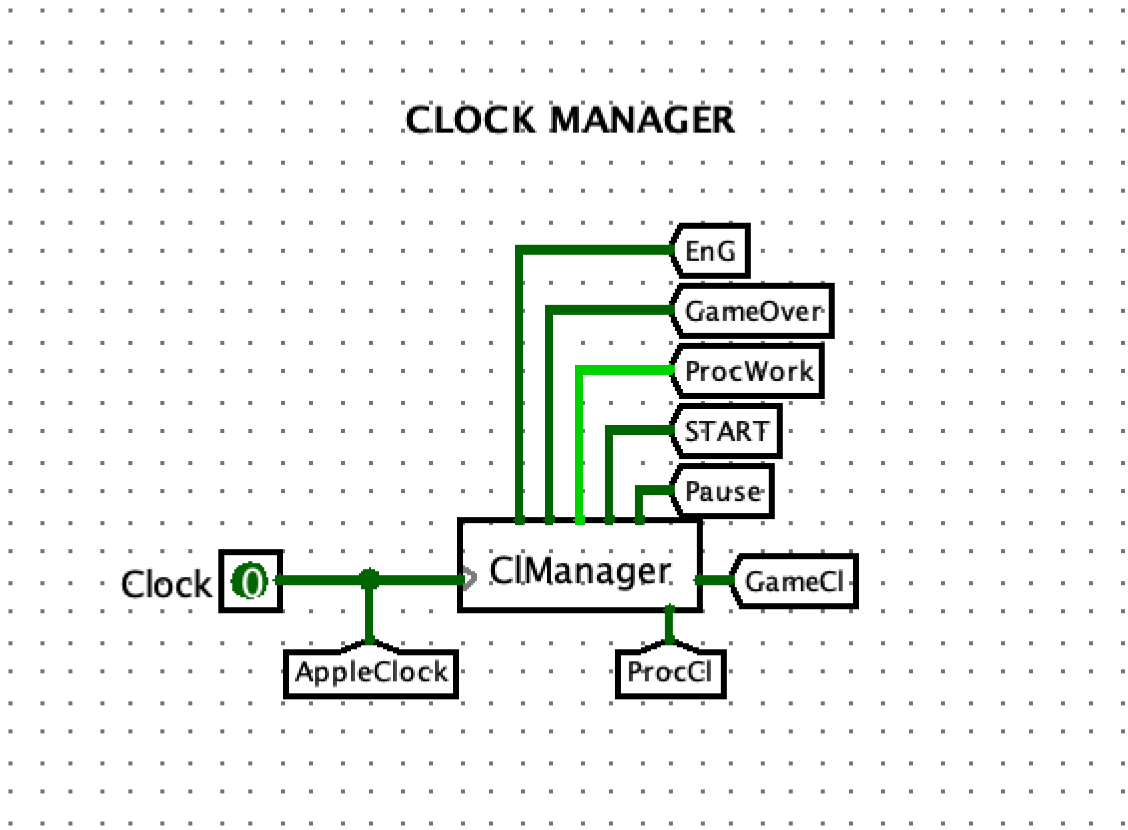
Here is the main game matrix, which displays the snake, apples, two score counters - one records the current score, the other the maximum score, the start and pause buttons, and on the side panel - LEDs showing the state of play, loss and pause.



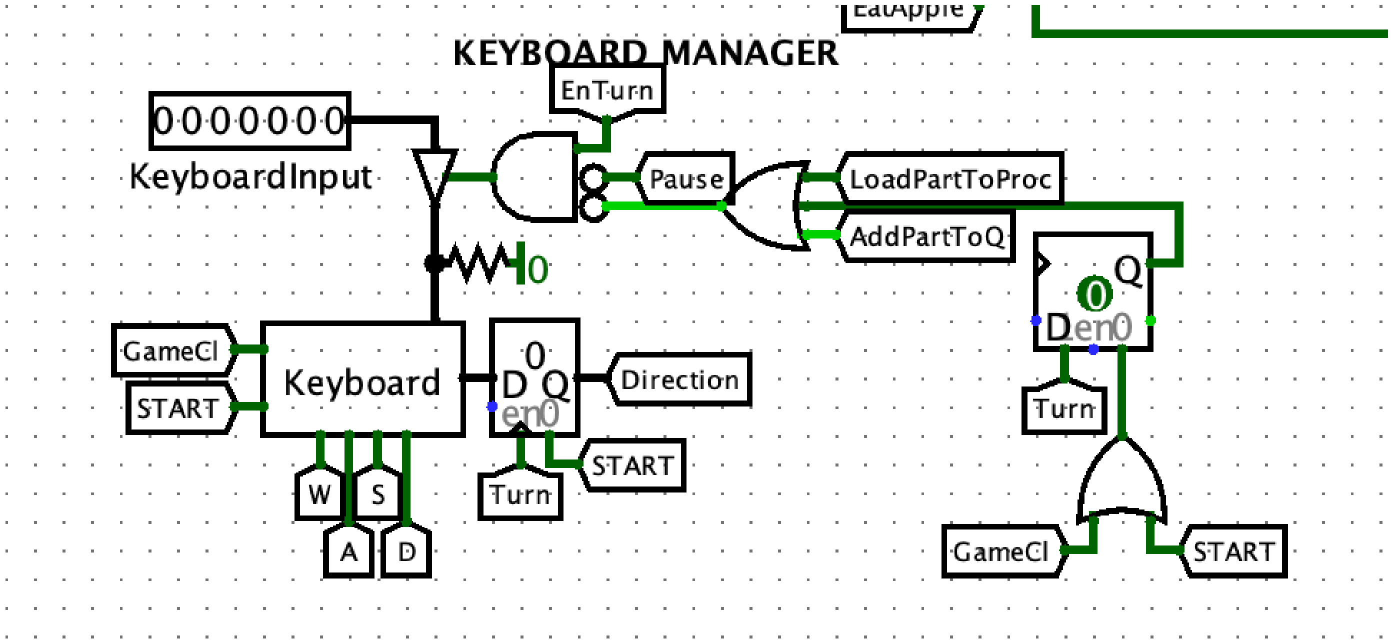
Main

Clock manager

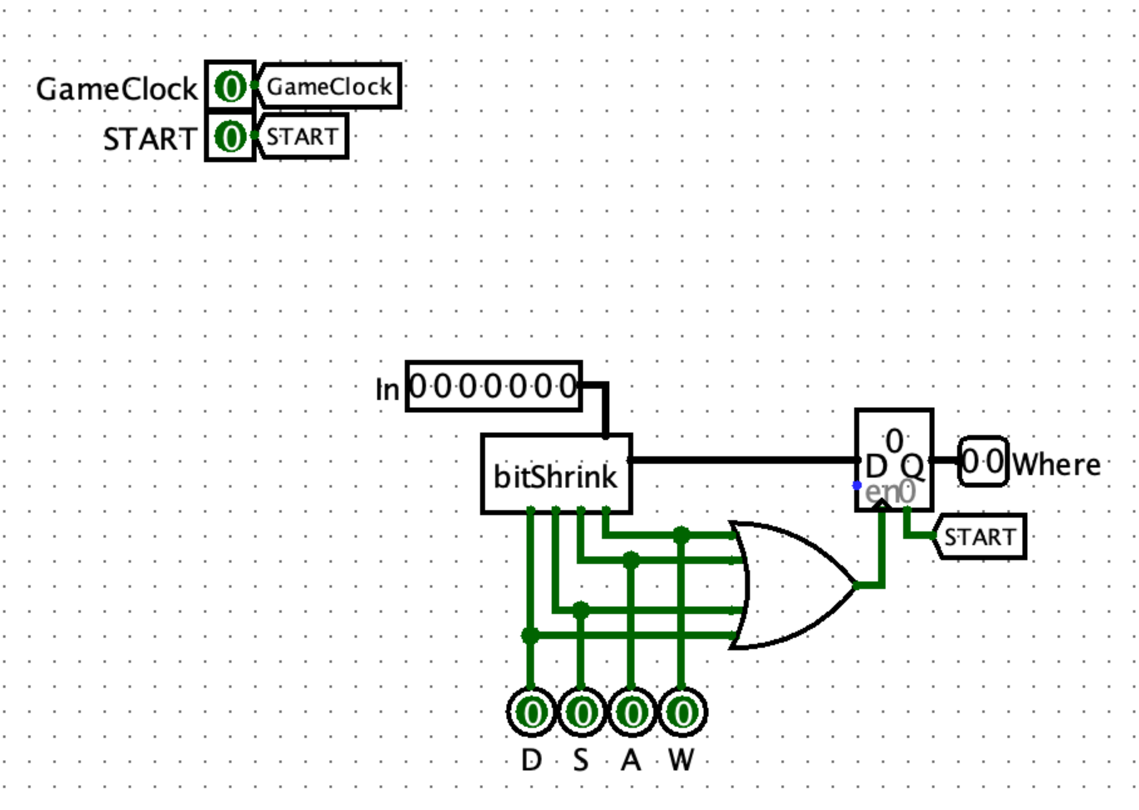
This diagram allocates the total number of clock cycles for each module depending on its state. It also manages the state of GameCL (GameClock) and updates its value based on the game and processor state.



Keyboard manager



Keyboard

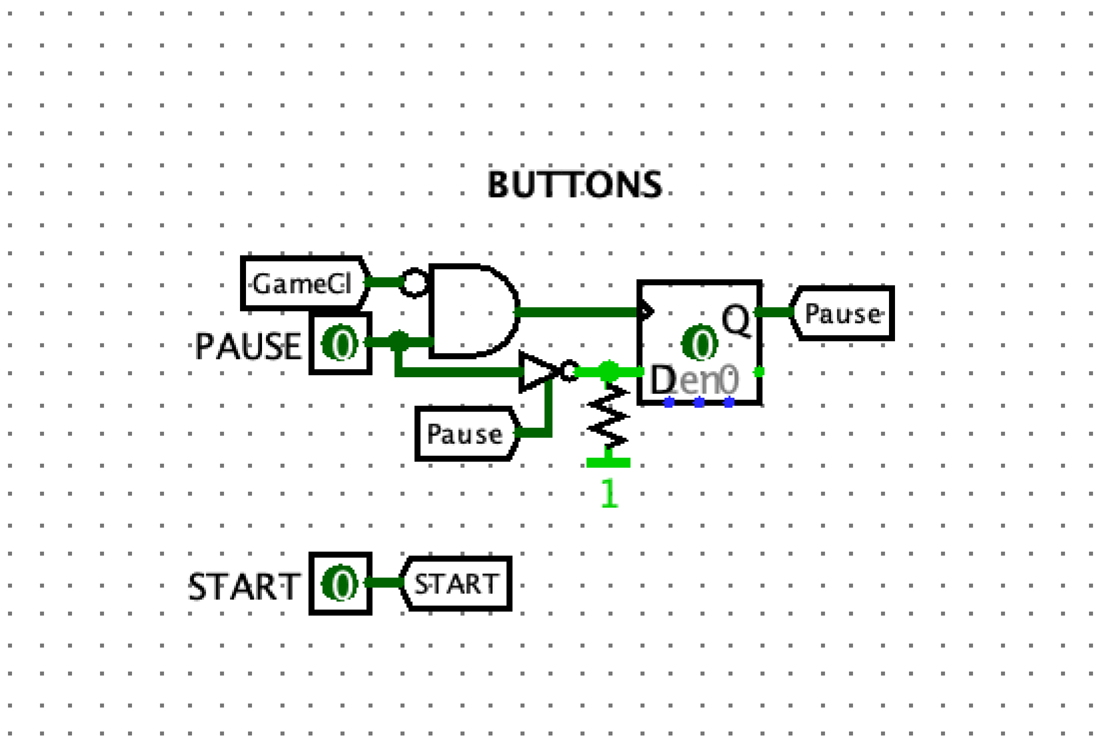


These two circuits are related to the input of values from the keyboard.

- The first circuit allocates the total number of clock cycles for each circuit depending on its state.
- The second circuit has a set of valid values, handles which key is currently pressed and overwrites the direction of the snake's movement, it also has a permissive input as a buffer.
- Nothing new can be entered until the snake makes a turn.

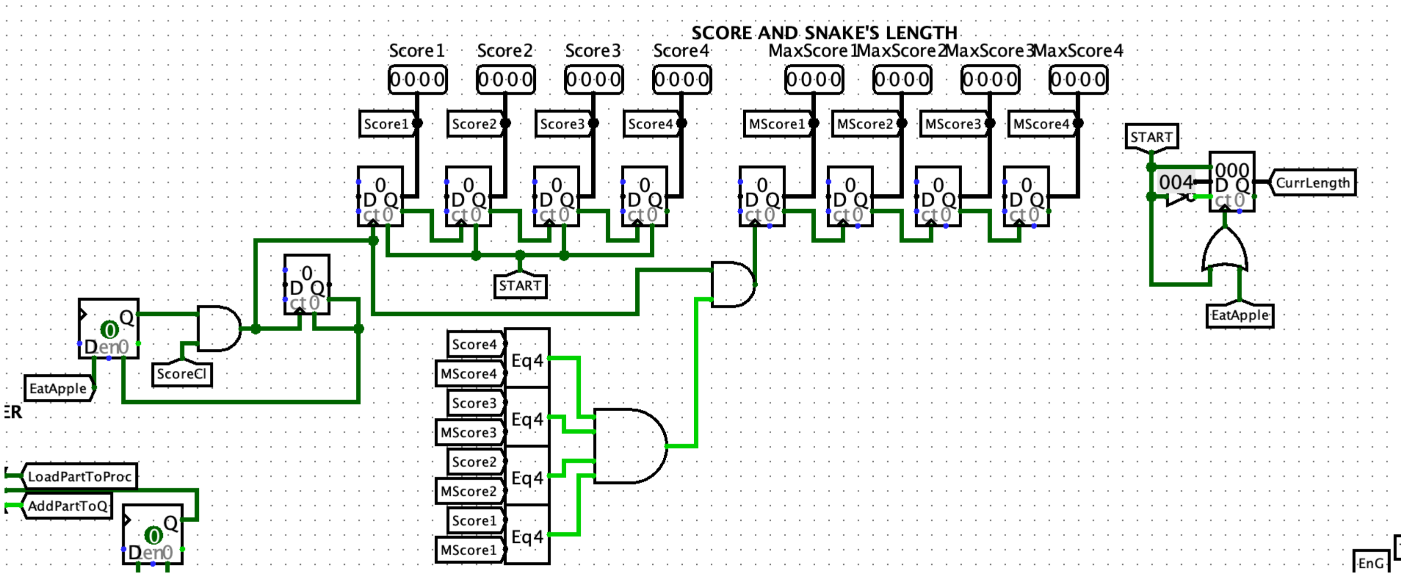
Buttons

This is a schematic of the start and pause buttons.



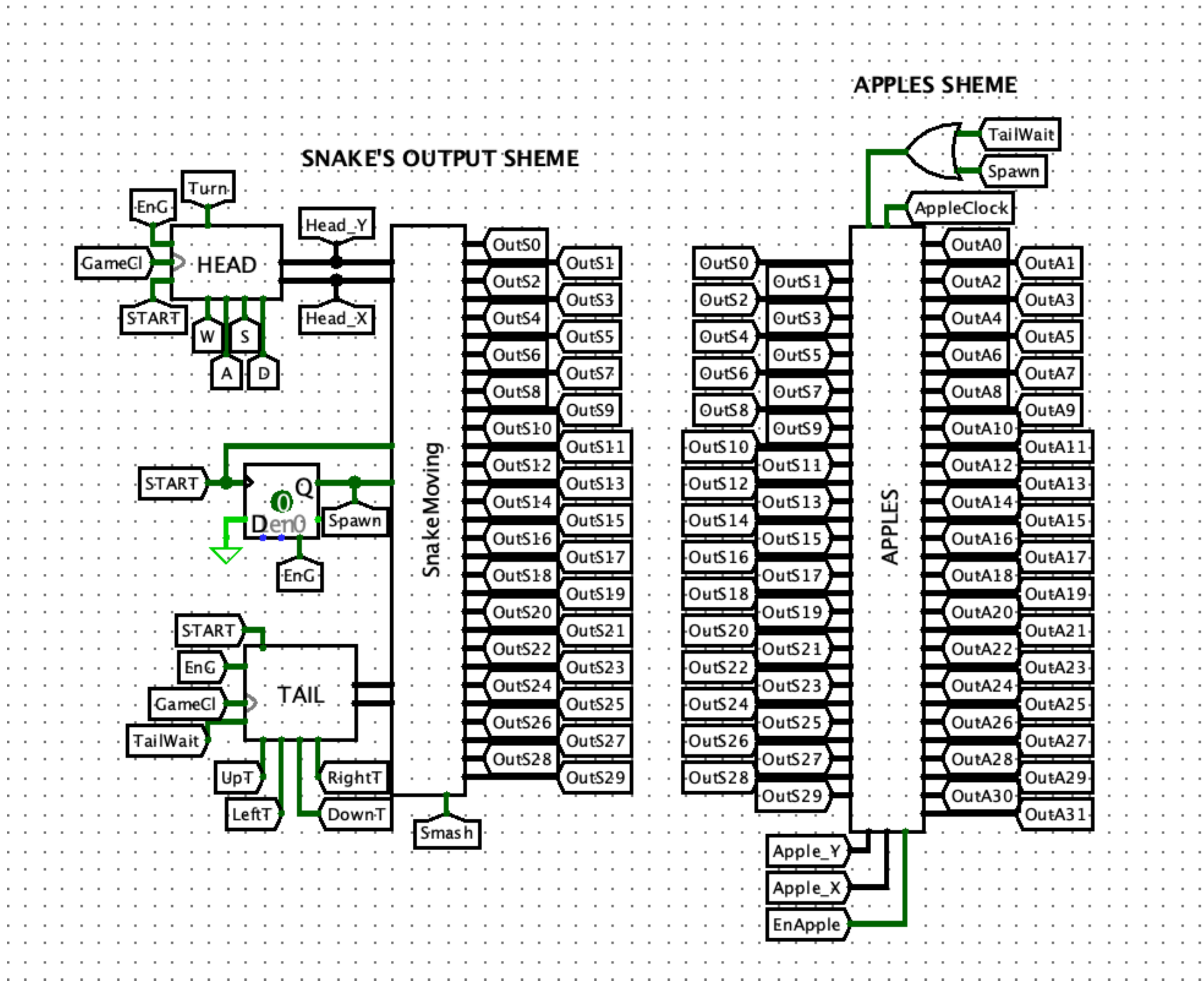
Score and snake`s length

In this circuit we have a current and a maximal count. if the current count is equal to the maximal count, it changes with the maximal count. On restart, the current account is reset and the maximum account is not changed



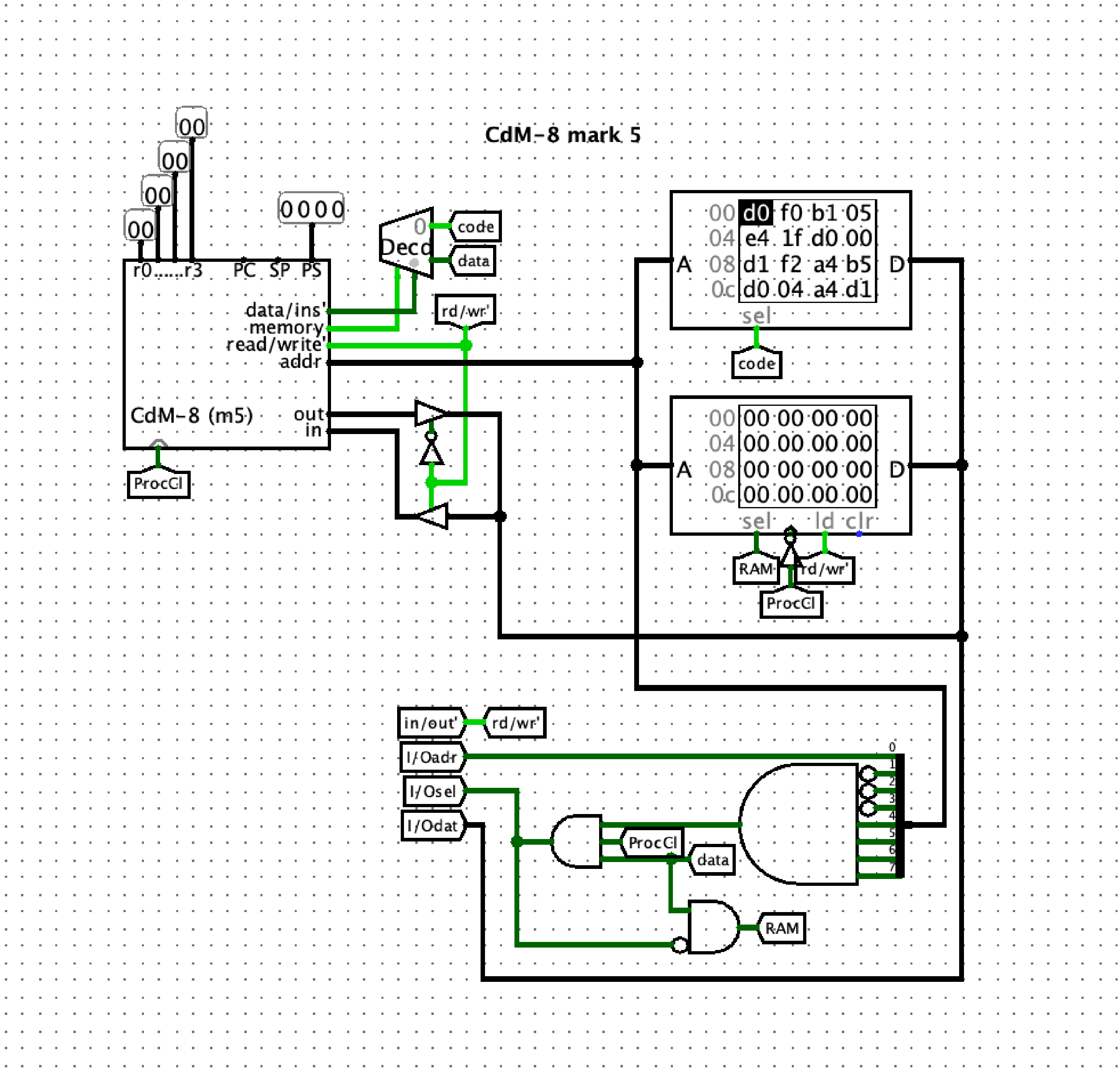
Snake`s output sheme and Apples sheme

- Apples sheme this circuit is responsible for where to spawn the apple on the main matrix
- Snake output circuit This circuit has the following inputs: the coordinates of the snake's head and tail. It also has an input that sequentially clears the map using a D-trigger, and an input that places the snake in initial form on the main matrix. This circuit determines the collision of the snake with its tail.



CdM-8-mark 5-reduced

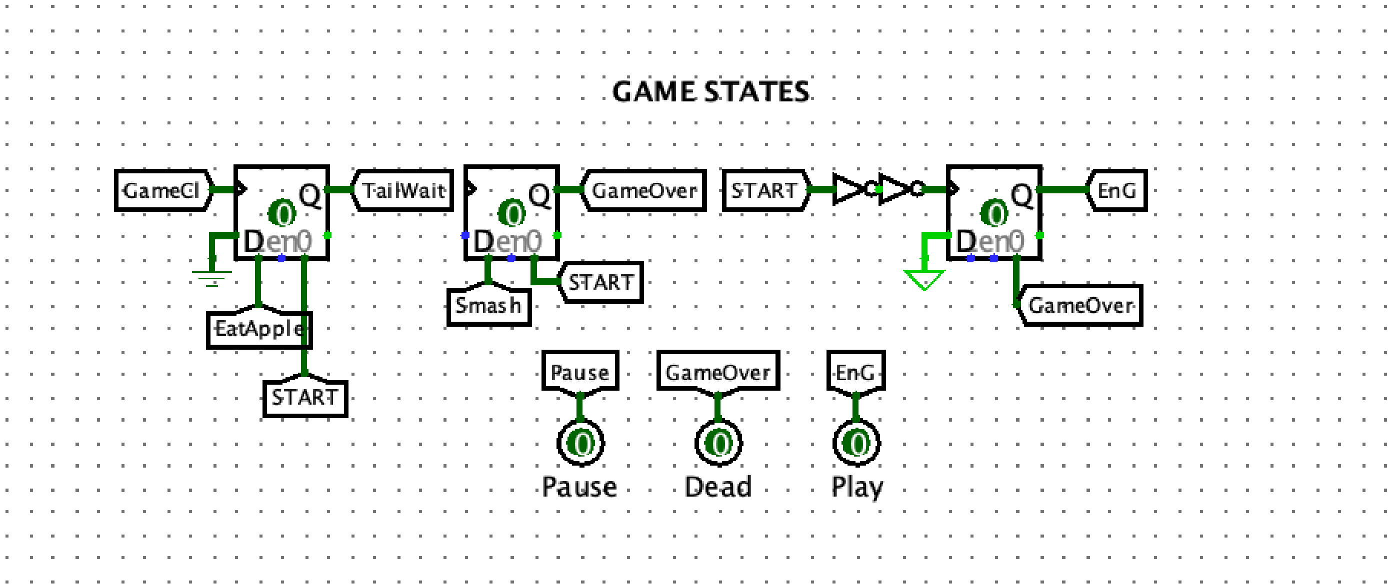
This is the CdM-8-mark 5-reduced processor circuit on which the snake segments control is implemented



Game states

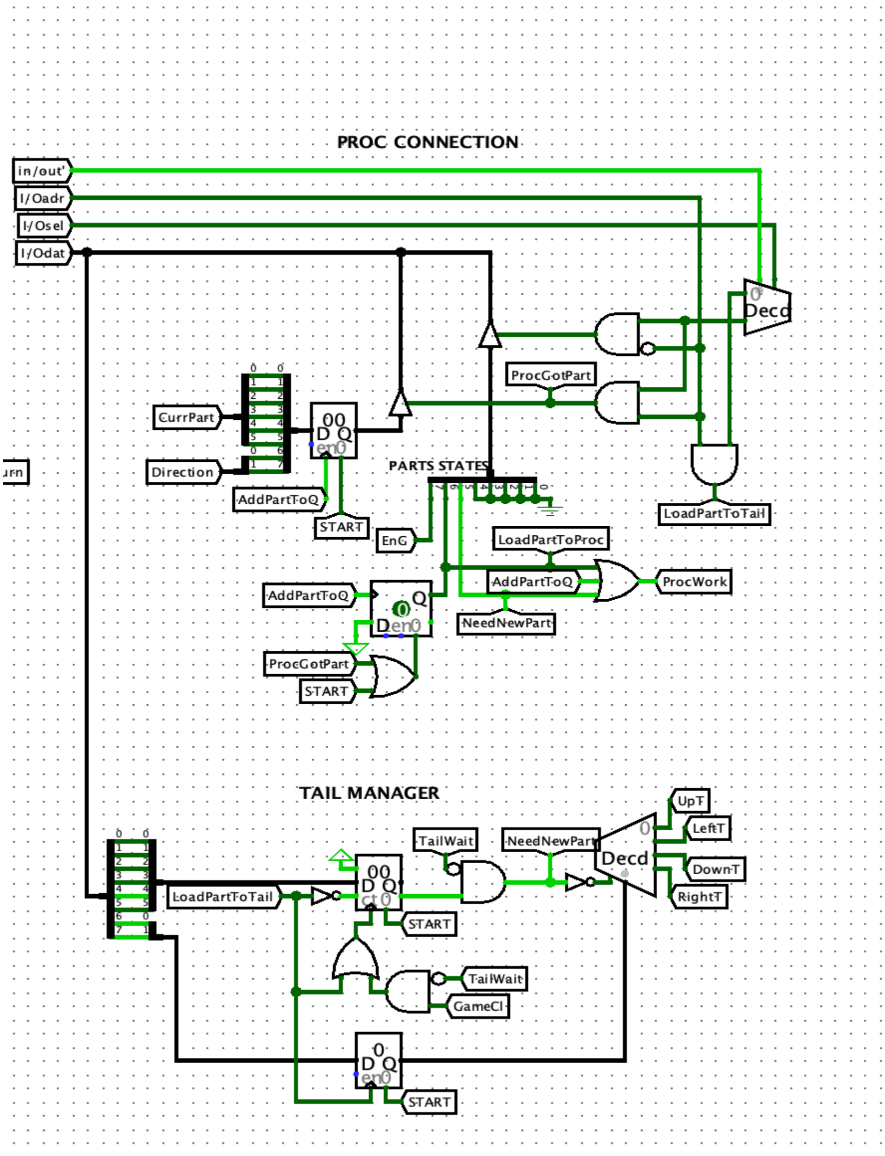
This circuit changes states depending on the calculations.

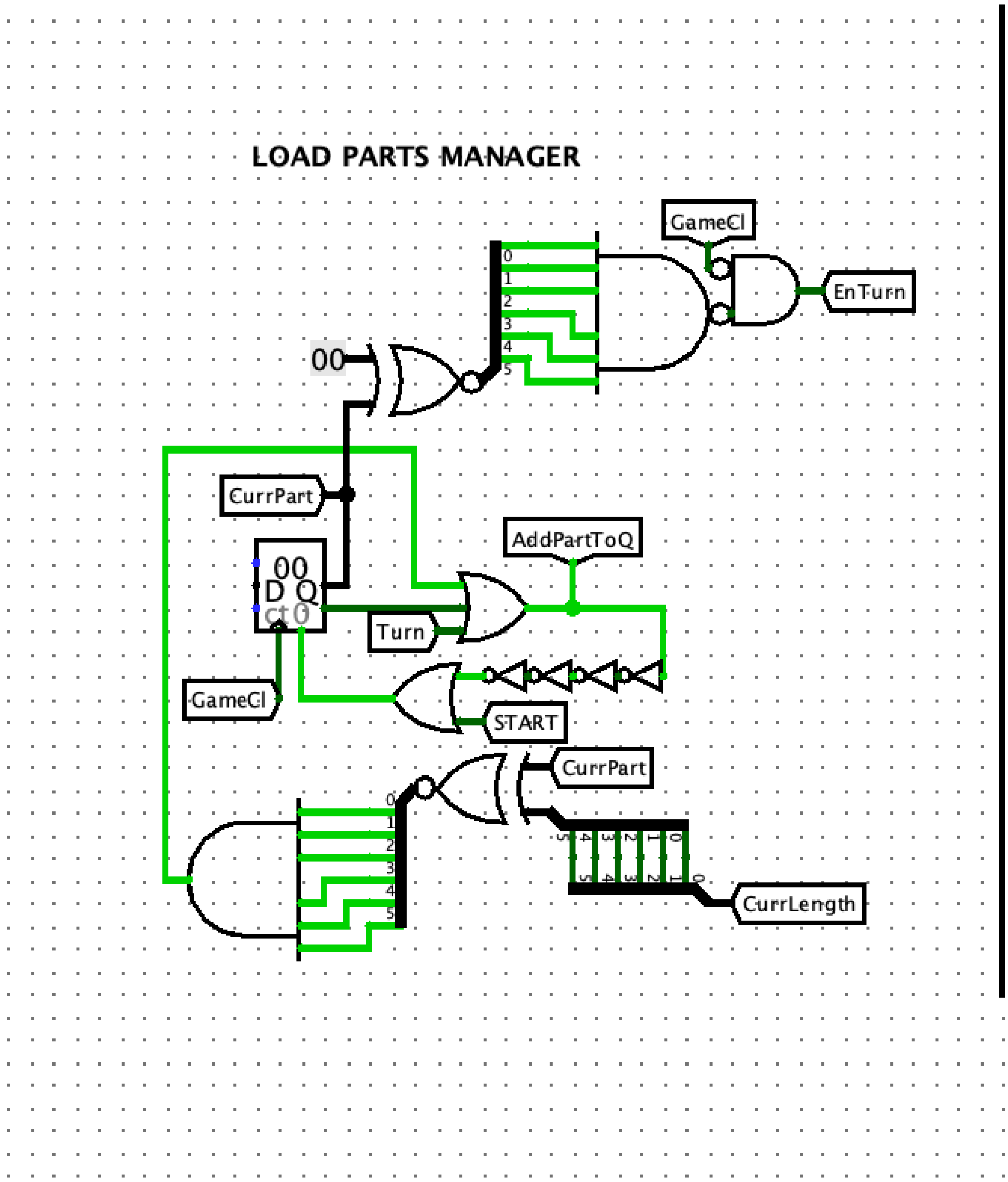
- TailWait increases the snake's length by one unit.
- GameOver is changed when the snake's head crashes into the tail.
- EnG - changes when all start values are loaded, gameclock clock input is opened and the player can start the game.



Proc Connection, Tail manager and Load parts manager

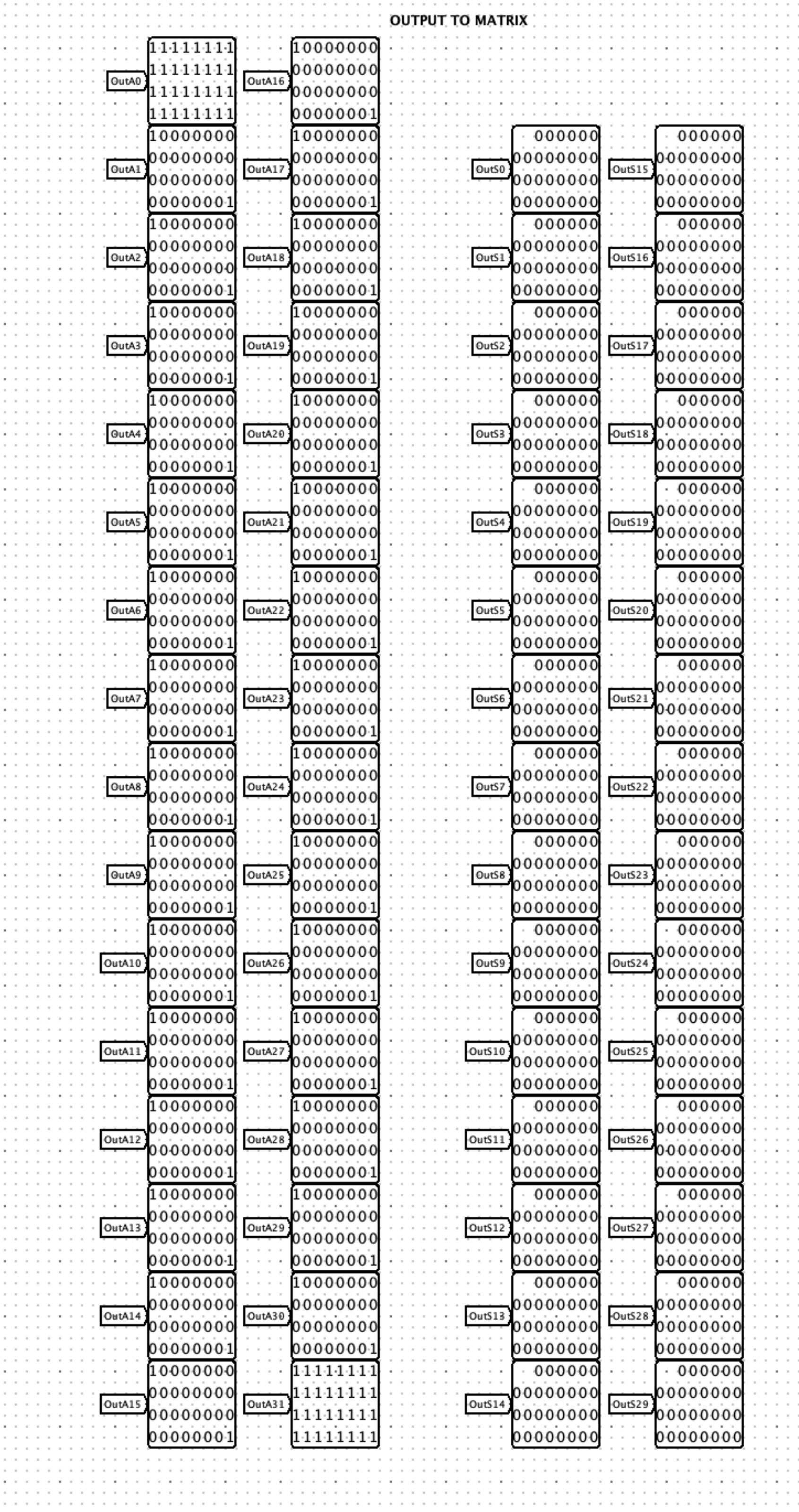
- We pass two values to the processor (software is registers F0 F1) if we write a value on F0 it means that we have passed some segment and it should be written to memory.
- The circuit supplies the value to be written to the state parts.
- The value is written, then a bit is raised which is passed to the processor, the process is executed and the bit is lowered.
- The "tail manager" determines where the tail needs to go and how long it needs to go
- the upper counter is the length and the lower counter is the direction (determined through the decoder, values: 00 - up 01 - left 10 - down 11 - right).
- As soon as the counter becomes equal to 0, it signals that a new element should be loaded into it.




LOAD PARTS MANAGER
GameCl
EnTurn
00
CurrPart
AddPartToQ
00
D Q
ct0
Turn
GameCl
START
CurrPart
CurrLength

Output to Matrix

There are outputs of this circuit for further display on the main matrix.



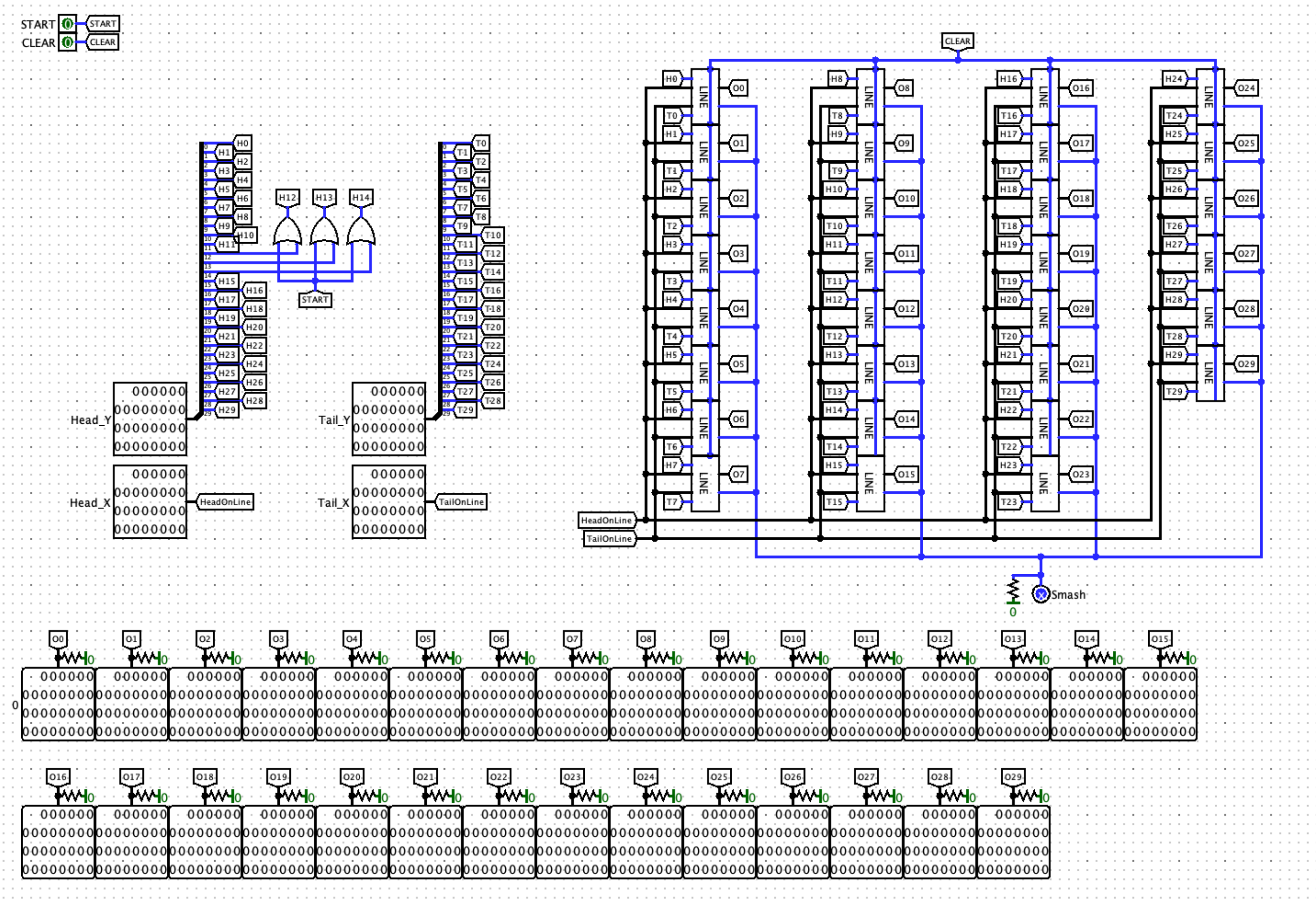
Pixel, Line, toMatrix

900 pixel elements are used to store and output a snake on a 30-bit matrix, each 30 pixel elements represent one line on the matrix and, accordingly, one Line element.

The final snake output and storage scheme - toMatrix - is assembled from 30 Line elements

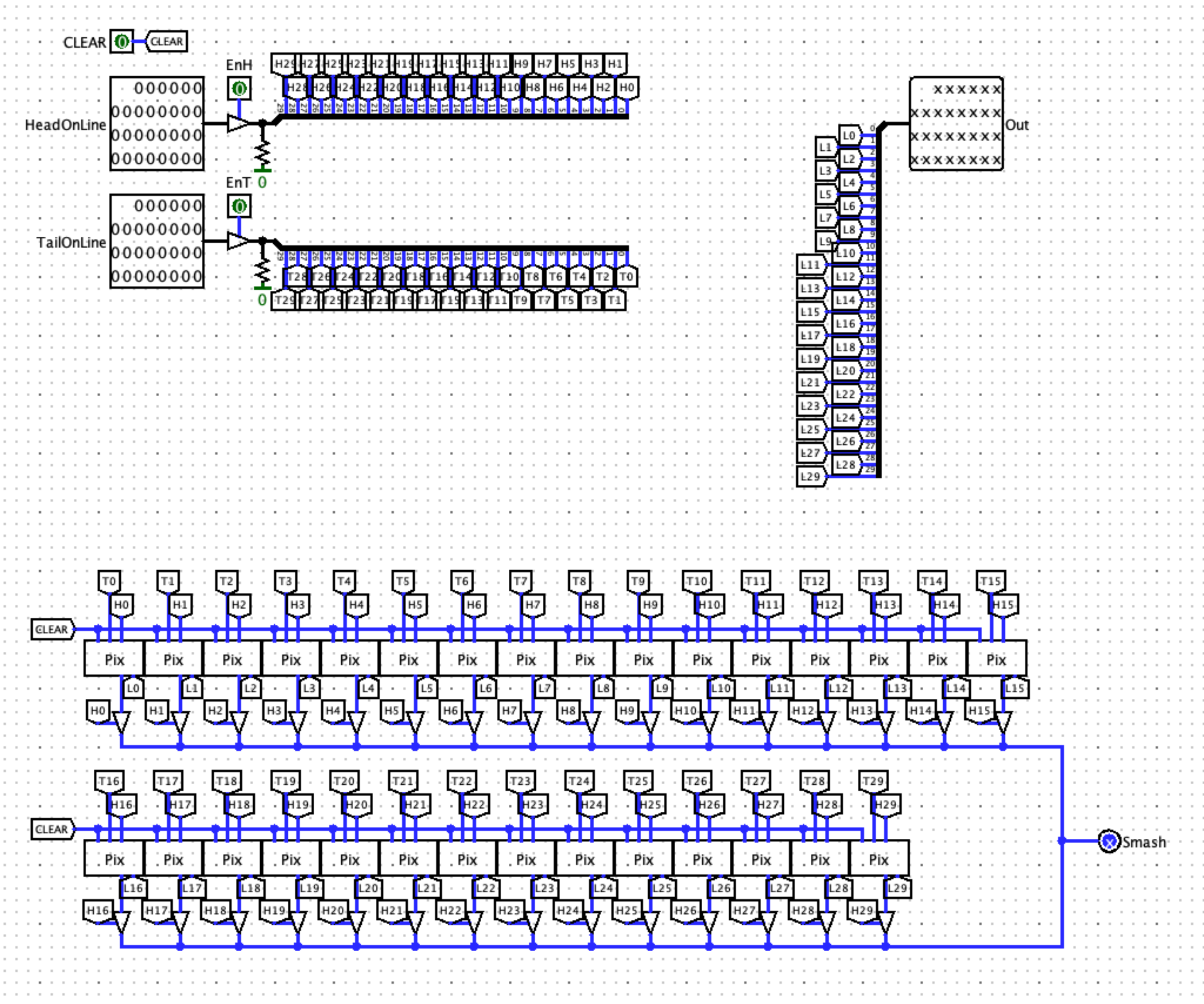
ToMatrix

- Takes the 30-bit coordinates of the Tail and Head as input.
- On the Y coordinate, it selects which line to feed the X coordinate to. In other words, Y coordinates raises the enable input on the desired Line element and feeds Head_X and Tail_X values to it.
- There are also START and CLEAR inputs
- START spawns the starting body of the snake with the size of three pixels, and after the head is spawned and we get a body with the length of 4 pixels.
- CLEAR - clears all available pixel D-triggers.
- ToMatrix outputs Lights values from all Line elements to the main 30-bit matrix, and combines all smash outputs into one.



Line

- The Line inputs are supplied with 30-bit HeadOnLine and TailOnLine values and enable inputs to them.
- HeadOnLine raises the corresponding pixel elements, TailOnLine clears them.
- All Light values of the 30 pixel elements combined through the splitter are fed to the Line output.
- The smash outputs with pixel elements are combined via controlled buffers and go to the common smash output.



Pixel

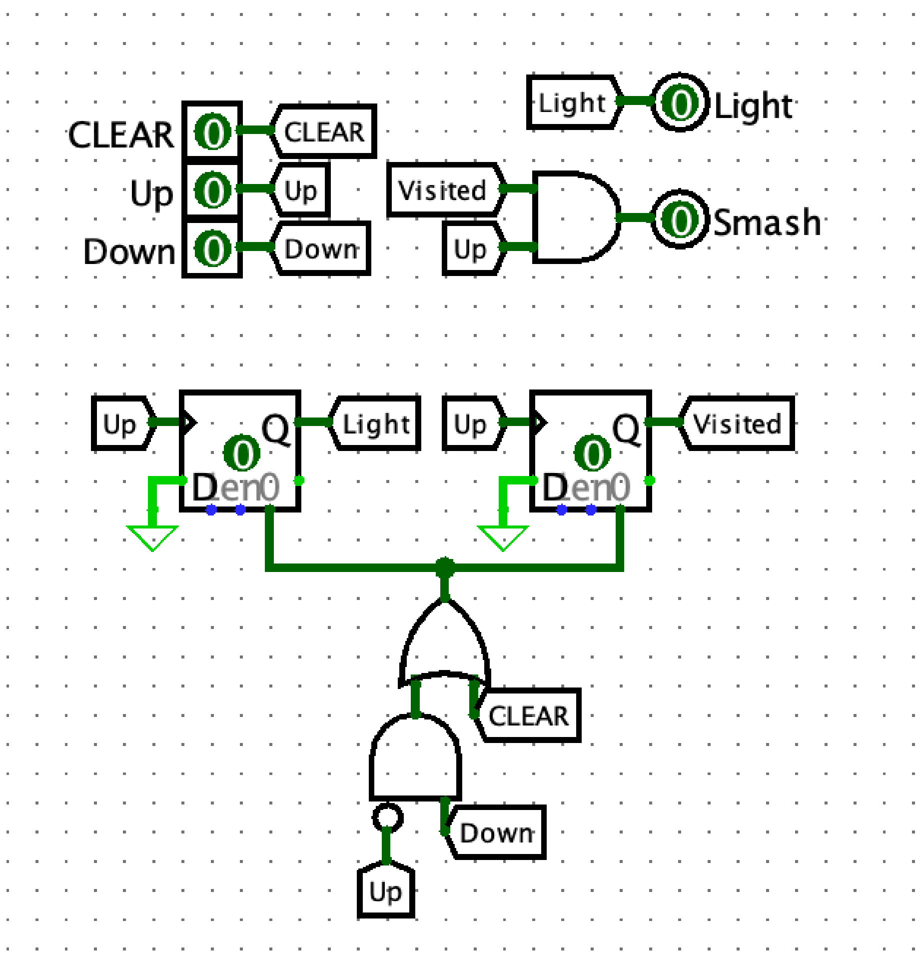
It consists of two D-triggers with the same connection, except for the edges

The first D-trigger Light triggers on the leading edge and is responsible for the state of a particular pixel (LED) on the matrix

The second D-trigger Visited triggers on the trailing edge and is responsible for the collision of the snake with the tail.

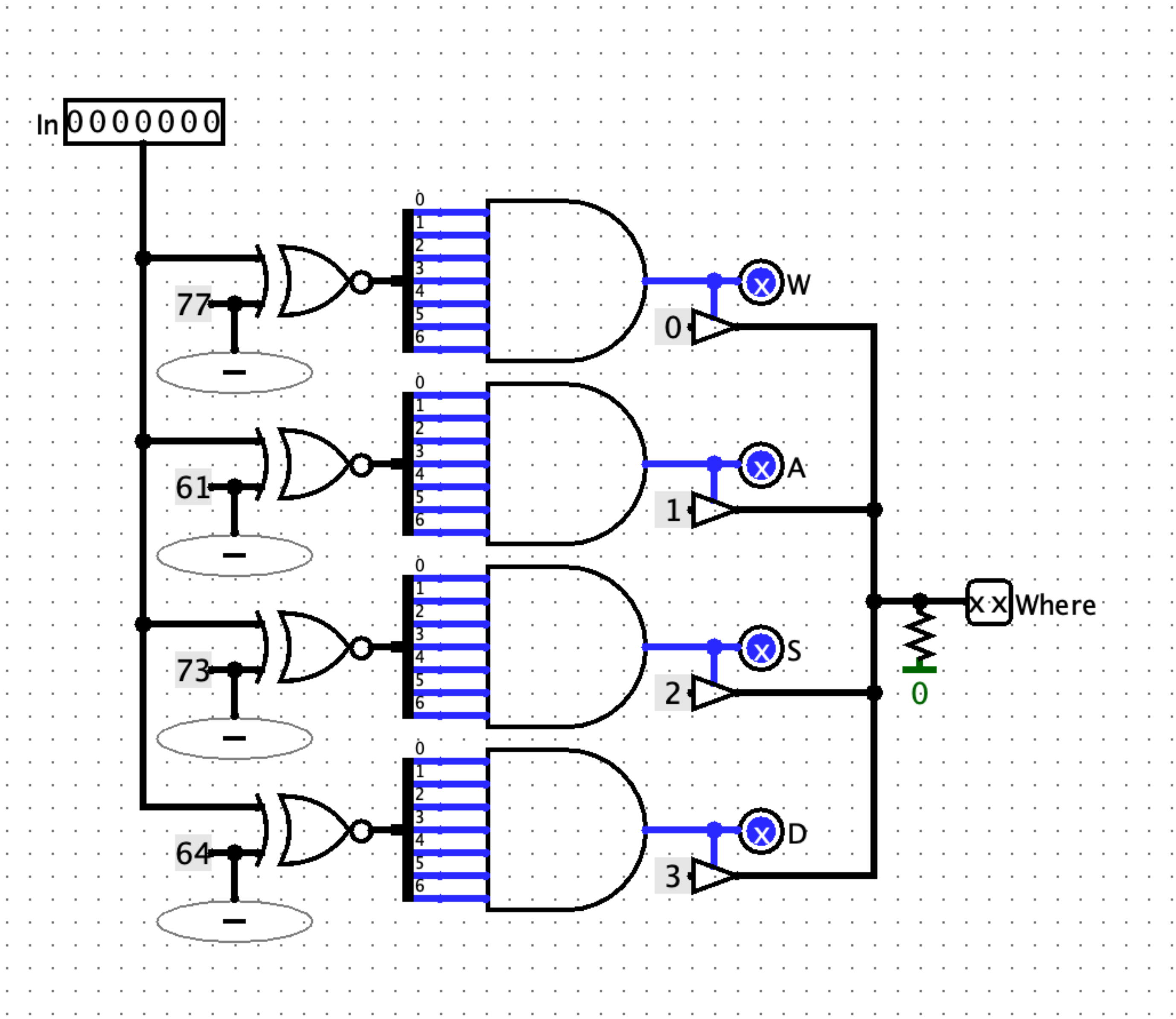
Inputs:

- Up - raises the D-triggers
- Down - clears, but only if Up is not raised
- Clear - clears and does not depend on Light and Visited states.
- Light and collision go to the output.



Bit Shrink

BitShtink compares input values from the keyboard, if WASD keys are pressed, it raises the corresponding outputs and also outputs the two-bit code of the pressed key.



Apples

This circuit has input values that signal that you need to make an apple input

- clock input
- game over input (resets the stored coordinates of the apple)

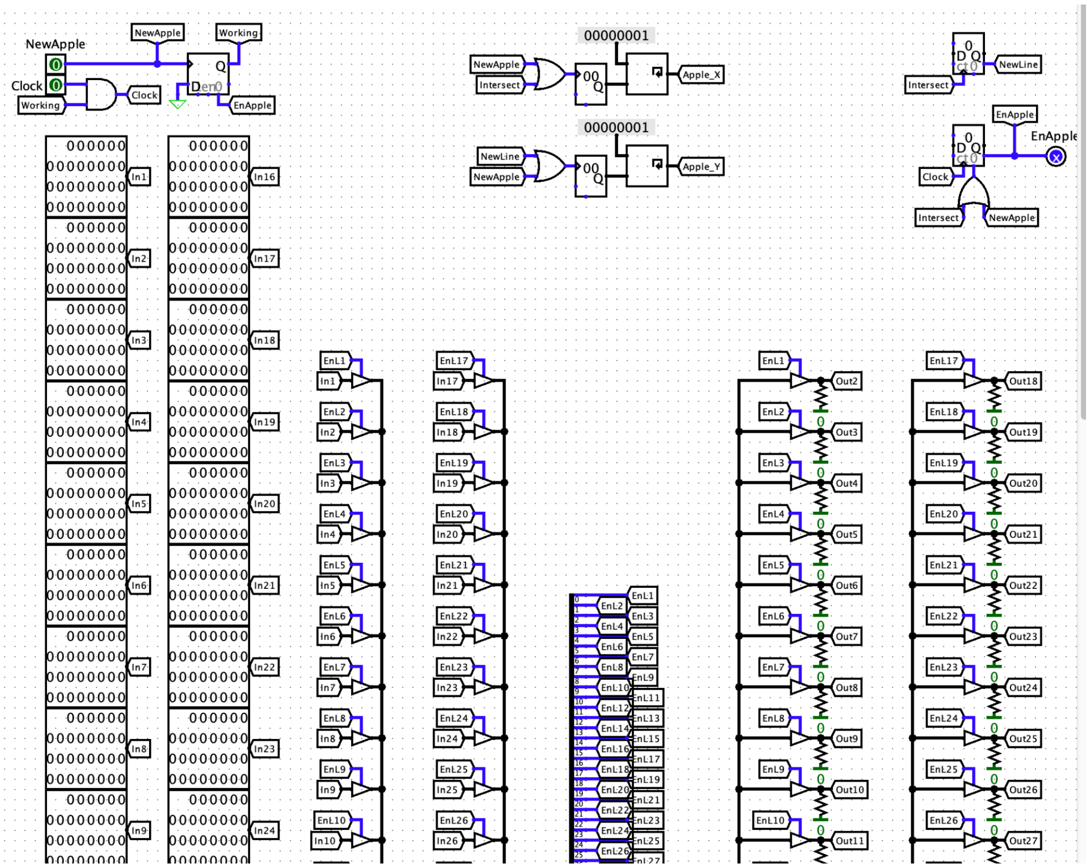
generation of apples - there are generators of random numbers they are connected to 30 bit engines, with their help we get coordinates Ax and Ay.

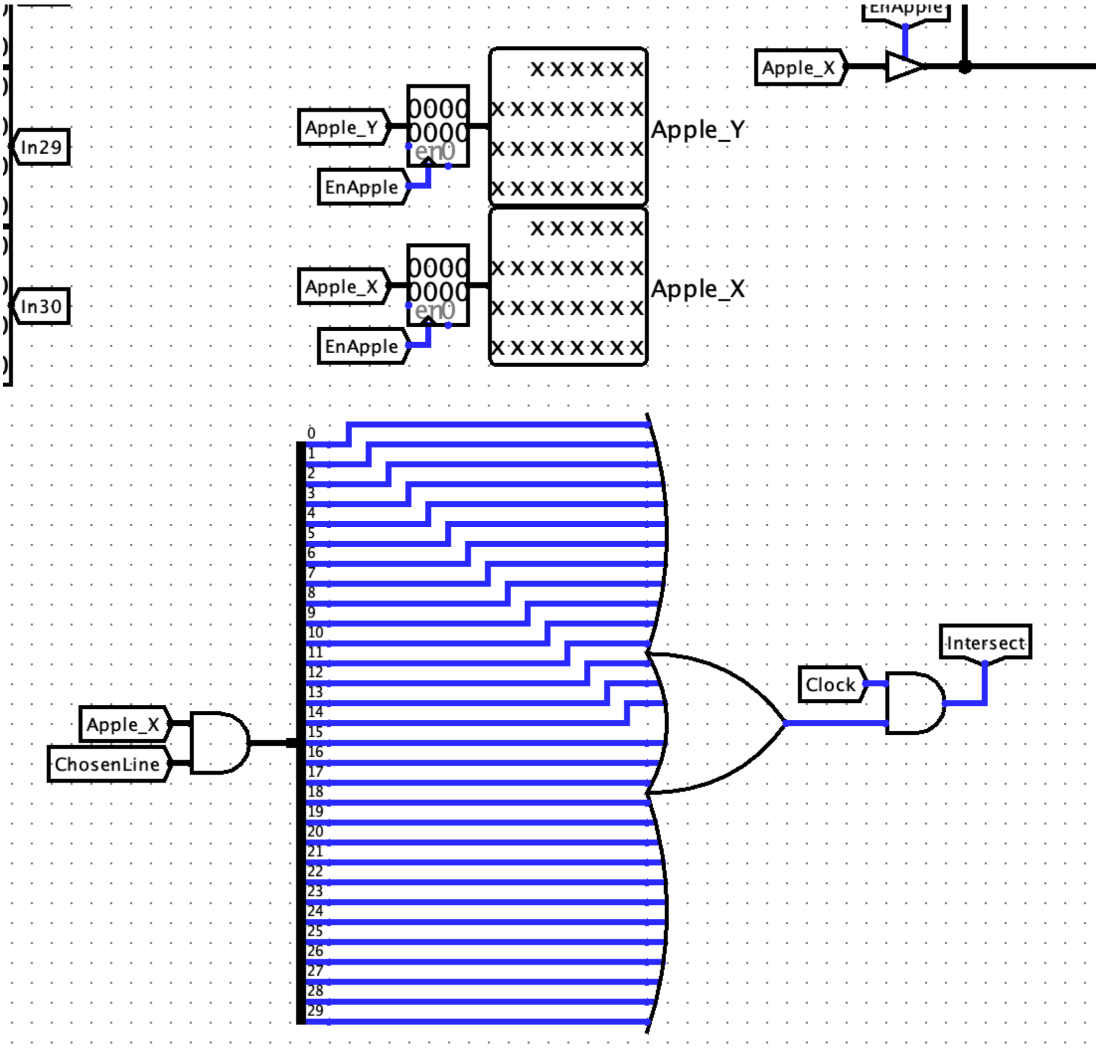
circuit below :

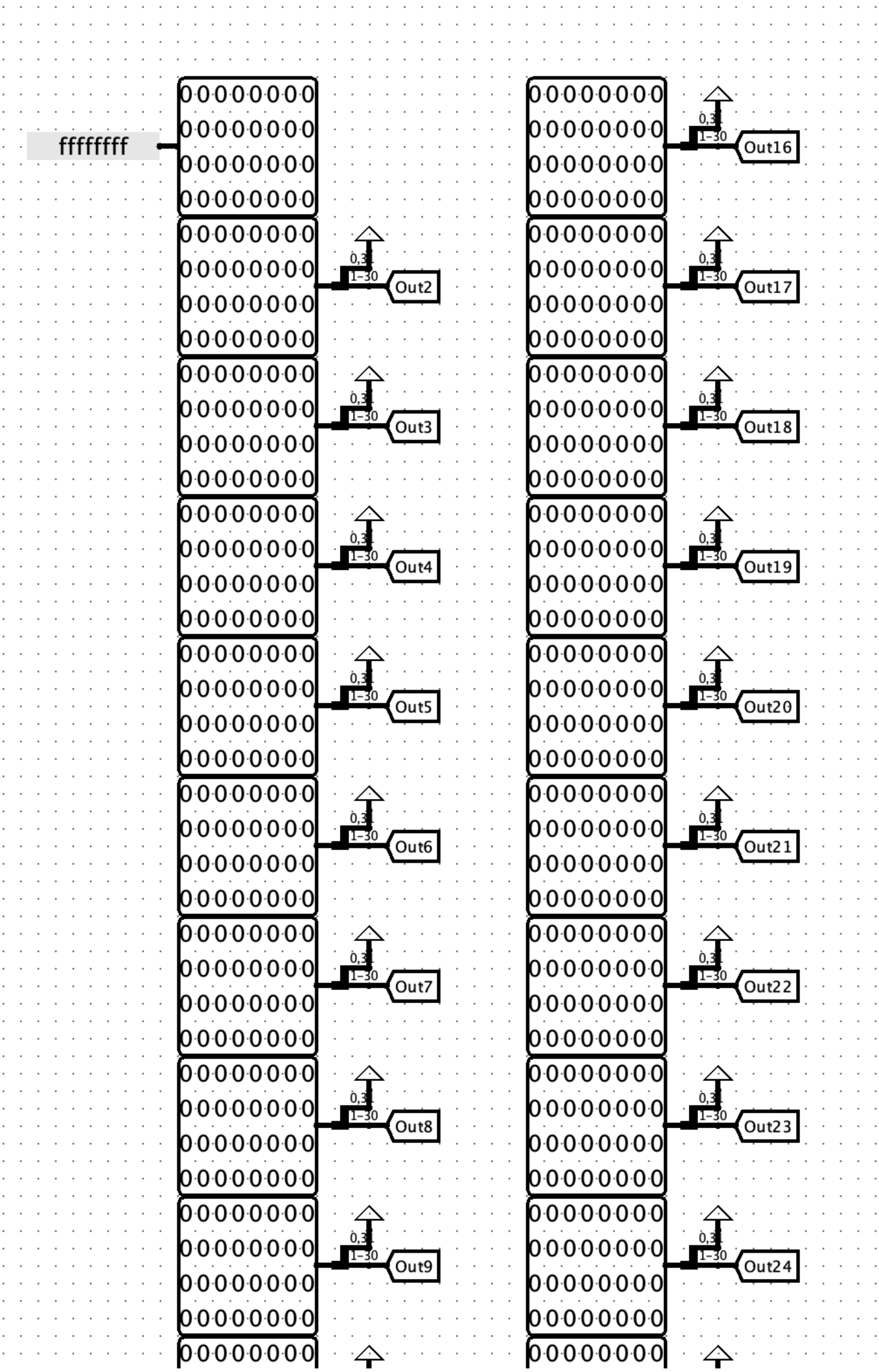
The selected string is compared with Ax and if there is a perpendicularity, we change the value of Ax.

If 16 times there is an intersection, we change the value of Ay.

Ay and Ax are changed to new ones at each generation of apple outputs and inputs are realized through buffers



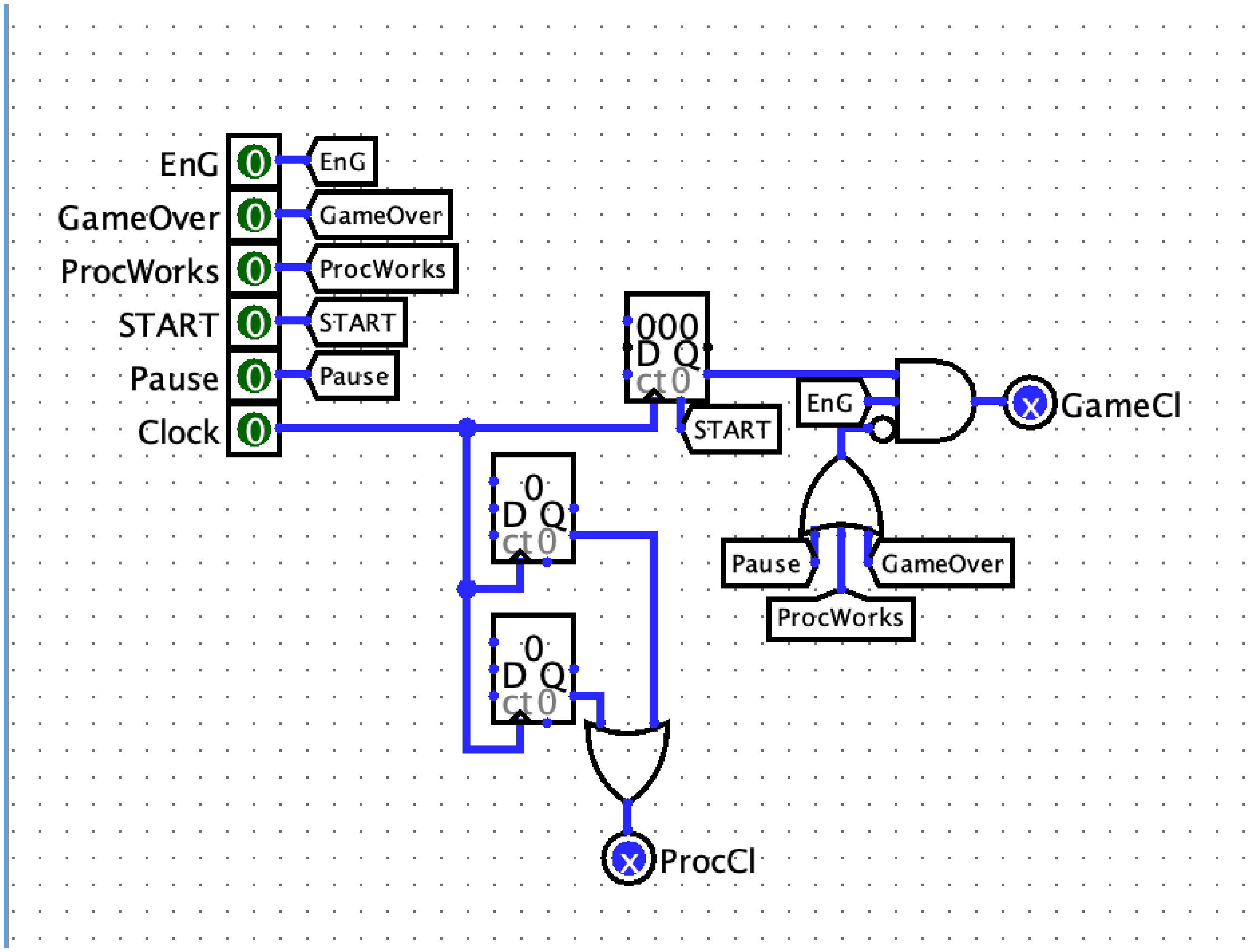

EnApple
Apple_X
In29
Apple_Y
0000
0000
en0
EnApple
Apple_Y
In30
Apple_X
0000
0000
en0
EnApple
Apple_X
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
Apple_X
ChosenLine
Clock
Intersect



- This is a circuit for two matrices, they are not the same size because we need to create a frame on the playing field.

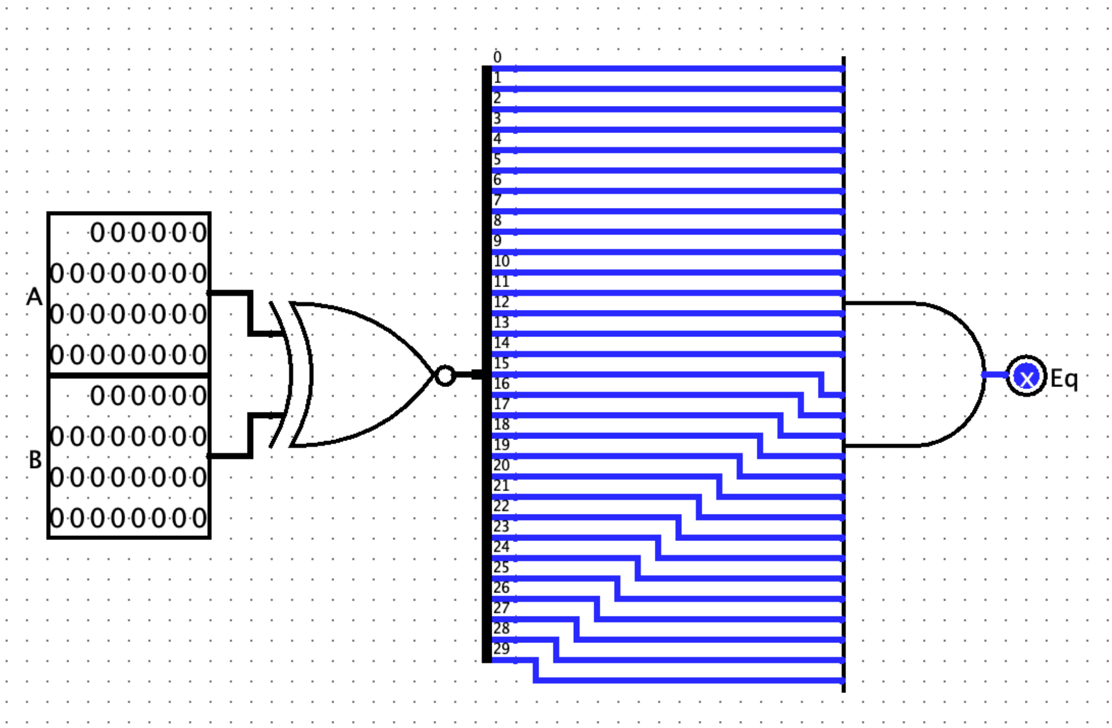
CIManager

Module CImanager consists of a circuit that is the manager of the continuous integration of other circuits.

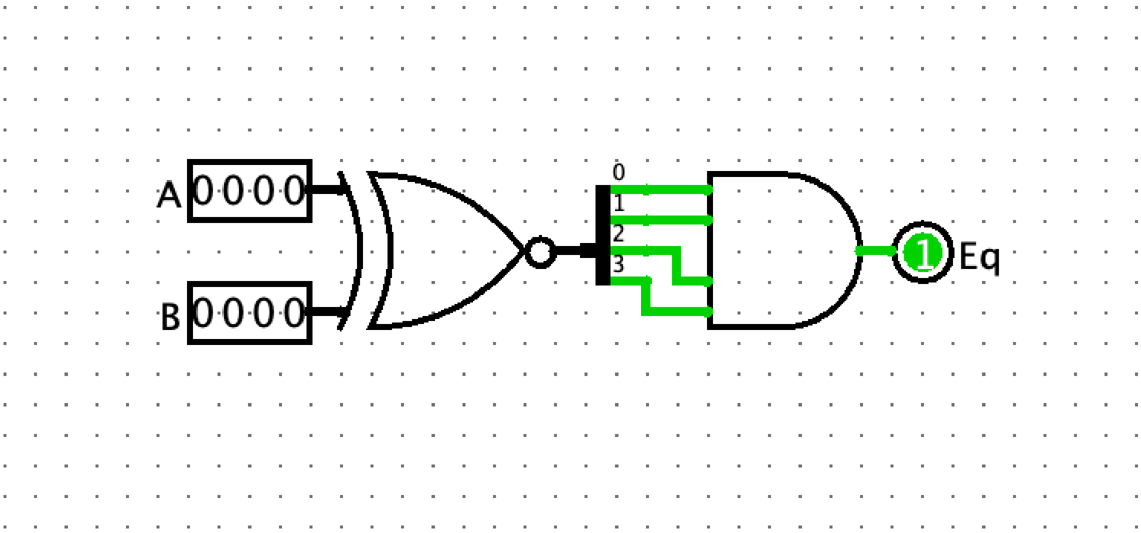


Eq30

30-bit equality



Eq4



Head

Head consists of four parts:

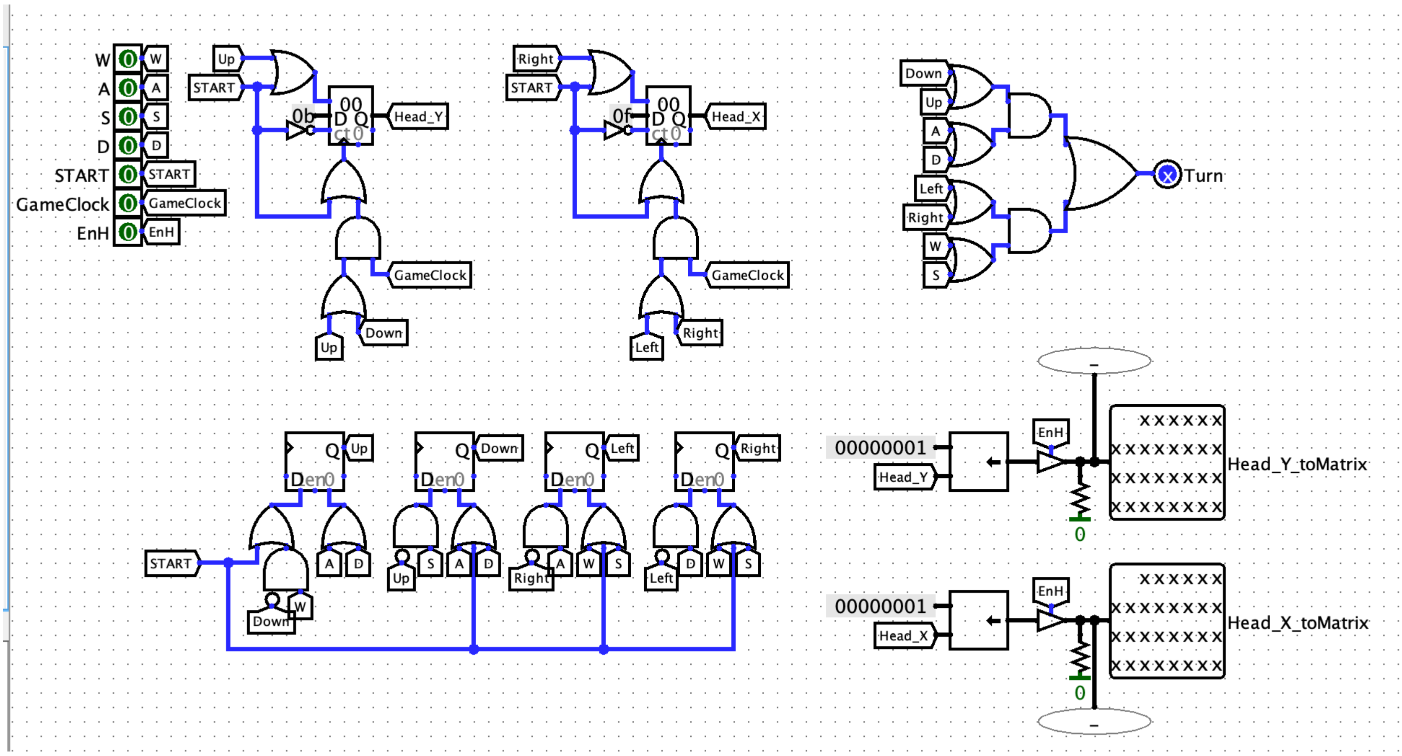
- Two 5-bit counters are the Y and X coordinates of the snake head on a 30-bit matrix. Depending on the Up, Down, Right and left signals, the counters are either decreased or increased. When the START button is pressed, they are set to 0b and 0f respectively.
- Below are the D-triggers that hold the direction of the snake until the player changes it, initially the snake moves up.

There is a circuit of the rotation sensor.

Below it, the 5-bit coordinates are converted to 30-bit coordinates using the left logic shifters. Each output has a enable input in the form of a buffer.

Input values:

- Wasd button signals, GameClock clock input, START, and En enable input



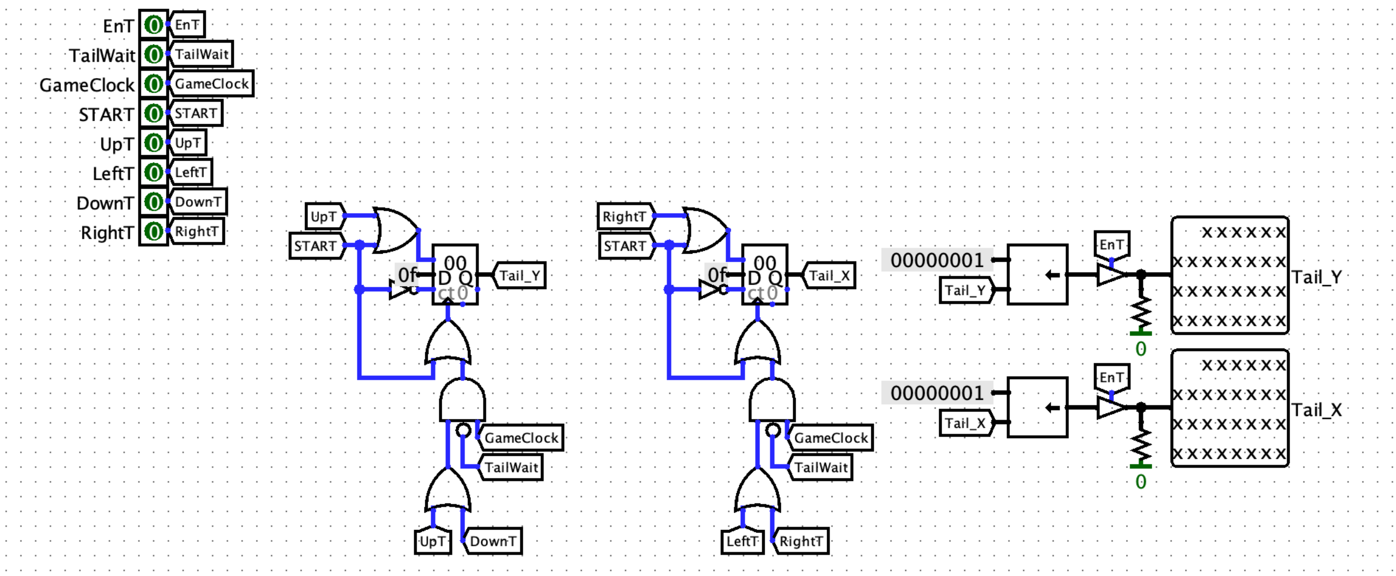
Tail

Tail consists of two parts:

- Two 5-bit counters are the Y and X coordinates of the tail of the snake on a 30-bit matrix. Depending on the UpT, DownT, RightT and leftT signals, the counters are either decreased or increased. When the START button is pressed, they are set to 0F and 0F respectively.
- The right side 5-bit coordinates are converted to 30-bit coordinates using the left side logic shifters. Each output has an enable input in the form of a buffer.

Input values:

- GameClock - Clock input
- TailWait - A signal that stops the clock on the counter inputs. It is raised when the snake needs to lengthen after eating an apple.
- START input
- Direction inputs (UpT, DownT, LeftT, RightT)



Software

```
1           asect 0xF0 # A memory cell used to send instructions
2   readyFlags:        # to the processor from external circuitry
3
4           asect 0xF1 # A memory cell used to exchange segments
5   parts:             # between the processor and external circuitry
6
7           asect 0xF2 # Pointer to the beginning of the queue
8   begQueue:
9           asect 0xF3 # Pointer to the end of the queue
10  endQueue:
11          asect 0xF4 # Memory cell storing the queue size
12  sizeQueue:
13          asect 0x00
14  main:
15          ldi r0, readyFlags
16          ld r0, r1
17
18          if # Game status check{
19              tst r1
20          is pl
21              # Initialization of variables{
22              ldi r0, 0x00
23              ldi r1, begQueue
24              st r1, r0
25
26              ld r1, r1
27              ldi r0, 0x04
28              st r1, r0
29
30              ldi r1, endQueue
31              ldi r0, 0x01
32              st r1, r0
33
34              ldi r1, sizeQueue
35              ldi r0, 1
36              st r1, r0 #}
37
38              # Waiting for the game to start
39              ldi r0, readyFlags
40              do
41                  ld r0, r1
42                  tst r1
43              until mi
44          fi
45          #}
46
47
48
49          if # Checking for a request to save a new segment to the queue
50              shl r1
51              tst r1
```

1-13: Setting variables: readyFlags - memory area through which the external circuit sends commands to the processor, parts - memory area through which segments are exchanged between the processor and the tail, begQueue - pointer to the beginning of the queue, endQueue - end of the queue, sizeQueue - queue size.

14: Start of the main program loop.

15-20: Load the status byte from the external circuit, if bit 7 is zero, i.e. the game is not running, then initialize the variables and wait for the game to start.

21-26:

22-24 - the pointer to the beginning of the queue takes the value 0x00

26-28 - value 0x04 (initial snake length) is loaded to the beginning of the queue

30-32 - pointer to the end of the queue takes the value 0x01

34-36 - queue size is equal to one

39-43 - loop waiting for the game to start, as soon as the seventh bit becomes equal to one, the number becomes negative and the loop is interrupted.

49-52: The status byte is shifted one to the left, so in the tst instruction, if the external circuitry has sent a command to the processor to load a new segment, bit 7 will be raised and the is mi condition will be triggered.

```
        tst r1
52  is mi
53      ldi r2, parts
54      ld r2, r2
55      move r2, r3
56      if # If the segment length is zero, we do not save it
57          shl r3
58          shl r3
59          tst r3
60      is nz
61          # Saving a new segment and incrementing the queue size {
62          ldi r0, endQueue
63          ld r0, r0
64          st r0, r2
65
66          inc r0
67          ldi r2, endQueue
68          st r2, r0
69
70          inc r2
71          ld r2, r0
72          inc r0
73          st r2, r0
74
75          ldi r0, endQueue
76          ld r0, r0
77          ldi r2, 0xF0
78          #}
79          if
80              cmp r0, r2
81          is eq
82              ldi r2, 0x00
83              ldi r0, endQueue
84              st r0, r2
85          fi
86      fi
87  fi
88
89
90
91  if
92      shl r1
93      tst r1
94  is mi
95      if
96          ldi r0, sizeQueue
```

53-60 - the segment is loaded into r2 and then copied into r3 and shifted twice to the left, so that r3 stores only the length of the segment, while the direction has been discarded. After that it is checked for the length, if it is not zero, the segment is loaded into memory and the queue size is incremented.

62-63 - saving the segment to the end of the queue

66-68 - incrementation of the pointer to the end of the queue

70-73 - queue size incrementation

76-86 - check for the end of memory - if it is reached, the pointer to the end of the queue takes the value 0x00

93-96: The status byte is shifted one to the left again, so at the tst instruction, if the external circuitry has sent a new segment unload instruction to the processor, bit 7 will be raised and the is mi condition will be triggered.

```
97                  ld r0, r0
98                  tst r0
99              is nz
100                 dec r0
101                 ldi r2, sizeQueue
102                 st r2, r0
103
104                 ldi r0, begQueue
105                 ld r0, r0
106                 ld r0, r0
107                 ldi r2, parts
108                 st r2, r0
109
110                 ldi r0, begQueue
111                 ld r0, r0
112                 inc r0
113                 ldi r2, begQueue
114                 st r2, r0
115
116                 ldi r0, begQueue
117                 ld r0, r0
118                 ldi r2, 0xF0
119                 if
120                     cmp r0, r2
121                 is eq
122                     ldi r2, 0x00
123                     ldi r0, begQueue
124                     st r0, r2
125                 fi
126             fi
127         fi
128 br main
    end
```

97-101 - if queue is not empty, unload new segment to external circuit - tail

102-104 - decrement queue size

106-110 - saving the value from the beginning of the queue to r0 and unloading it through parts to the tail

112-116 - incrementation of the pointer to the beginning of the queue

119-128 - check for end of memory - if it is reached, the pointer to the beginning of the queue takes the value 0x00

132 - end of the loop

Conclusion

The result of this work was the realization of the game “Snake”.

Our project was realized on the basis of logic circuits in Logisim and software written in Assembler in CocoIDE development environment.

In the process of creating our project we faced some difficulties, some of them seemed unsolvable, but we coped.

The process of creation was very interesting, we learned how our game works from the inside, what nuances can arise during its creation.

We improved our knowledge and of course we are grateful for this experience!

We were happy to present our project to you.

Sources

- http://ccfit.nsu.ru/~fat/Platforms/
- tome.pdf