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)

09.03.01 - Informatics and Computer Engineering
Focus (profile): Software Engineering and Computer Science

COURSEWORK

Authors

Petrov Vladimir 22216
Khamidullin Rustam 22216
Yakutina Sveta 22216

Job topic:

Project B
«TV Tennis»

Novosibirsk, 2023

TABLE OF CONTENTS

Introduction

For our group project, we chose the topic "TV-Tennis", since it is the most interesting.

The project provides an opportunity to apply and consolidate the knowledge gained on the course “Digital Platforms”.

We have developed a TV-Tennis on the Logisim and CdM-8 platforms.

Logisim is a logic simulator that allows you to design and simulate circuits using a graphical interface.

CDM8 Assembly Language is a language for development on the cdm8 platform.

Our project contains 6 main parts:

- Problem statement
- User Manual
- Hardware
- Interfacing the game with the CdM-8 platform
- Software
- Conclusion
- Appendix

Problem statement

In our project we decided to improve the basic version of the game TV-Tennis, make it more diverse and interesting. To do this, we added features to the game:

- Choose the difficulty level of the game
- Change ball speed on Y after hitting the racket
- Change the color of the ball
- Display the result at the end of the game

User Manual

The playing field is 32x32 pixels – the ball moves on it. On the left side is the player’s racket, which can be moved up or down. On the right side is the bot's racket.The game continues until someone scores the ball 11 times.

All movements are made with the joystick.

Other settings for the game:

- Start – responsible for the start of the game
- Reset – stops the game and resets the score
- Level – responsible for selecting the level of difficulty of the game
- Ball color – changes the color of the ball
- Ball weight – responsible for selecting the ball weight

Hardware

The display panel, the bats and the ball

We have a general-purpose display panel made up of 1024 pixels, arranged into 32 columns of 32 pixels each. The columns are numbered 0..31 from left to right, and the pixels in each column are numbered 0..31 from bottom to top.

Each column has a 32-bit input, where each bit is responsible for a different pixel, and more correctly for its color (#4EFFF6 - if the signal on this pixel is high, #404040 - if the signal is low, and red, if an unidentified value came to the pixel (we use this value to change the color of the ball)). In this way you can display any picture on the screen.

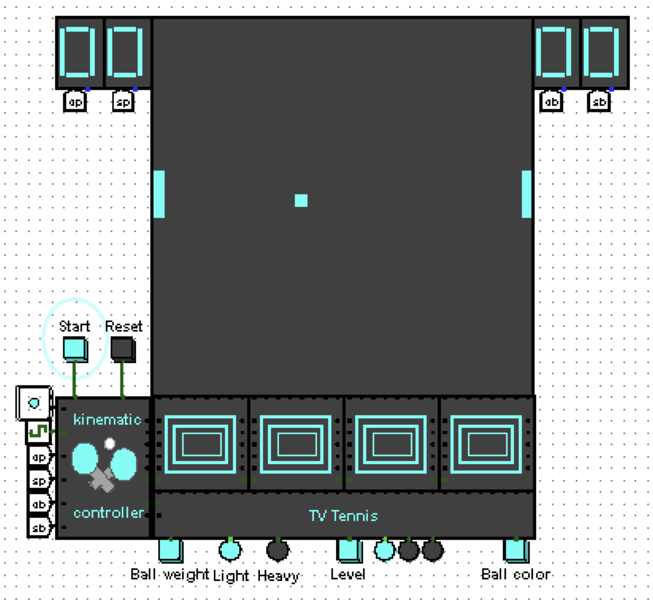
In TV Tennis we need to display two bats and a ball. The racket is a line with a height of 4 pixels. The bats are located at the extreme indexes (0 and 31) to increase the playing space. The ball is displayed as a single pixel on the screen and can be located anywhere on the field, except for the positions taken by the bats.

In order to display the ball on the screen we need to know its position in the X and Y coordinates. To do this, we use a 32-bit representation of the ball's Y position to immediately display it on the screen and a 5-bit representation of the ball's X position. The values are generated, stored and updated in the controller and then sent to the video system and displayed.

Main circuit

This circuit is the link between all the other subcircuits. It contains:

- **32 by 32 pixels display**
- **kinematic controller**
- **4 video system**
- **joystick and 2 buttons (start and reset)**
- **scheme for displaying victory and defeat screens**
- **settings block**



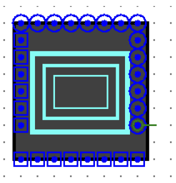
How it works?
In our version of tennis we give you the ability to choose difficulty levels, ball weight and color, so before or even during the game we can select the comfortable conditions of the game.

First, the weight of the ball, if you want to make the bat speed have a greater effect on the velocity of the ball, then a light ball is definitely your choice, otherwise go for a heavy ball. Next, we need to choose the level of difficulty. At first level your opponent can make a lot of mistakes, so if you are interested in interesting gameplay, I advise you to try medium and high levels of difficulty. And at the end you can change the color of the ball. After the settings you should press the start button and the game will start.

WARNING: All circuits except the processor run at 1/4 of the current frequency. One 1 clock cycle (4 ticks of the frequency selected in the logisim settings) of these circuits in our work we call a recalculation cycle.

Below you can read a detailed description of each component.

Video system



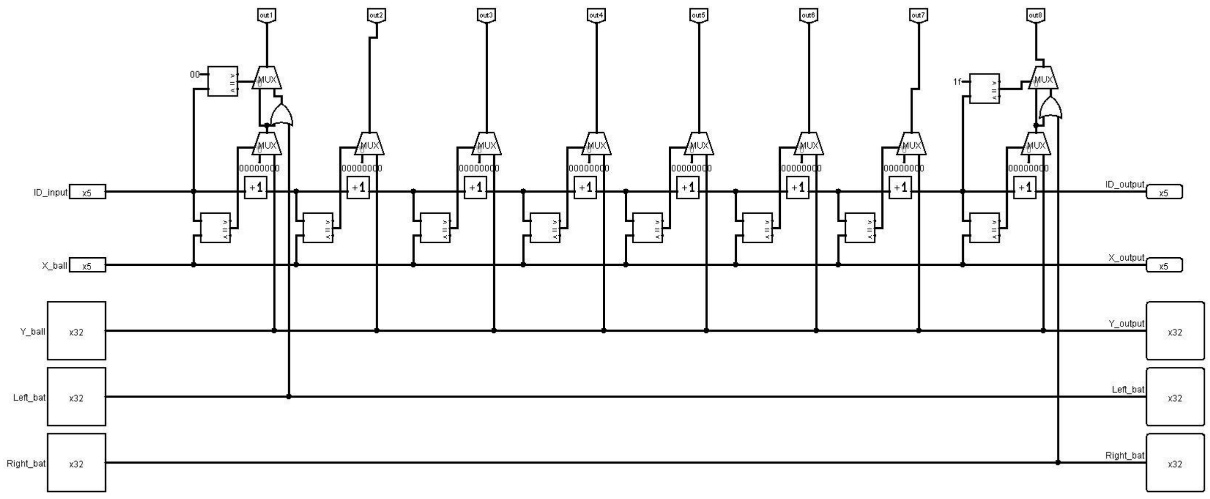
Takes in the input:

- **Position of the ball (by X (5 bits) and Y (32 bits) coordinates)**
- **Positions of both rackets (32 bits)**

the bats mapping is calculated in the kinematic controller

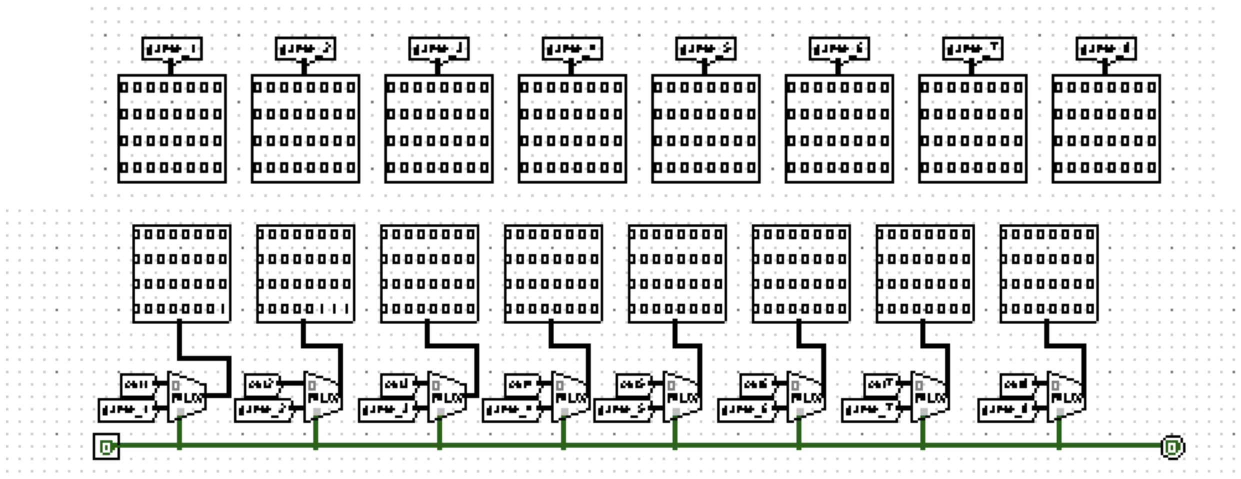
- **Video chip ID (5 bits)**

There are also (8) 32-bit inputs at the bottom of the circuit. They are needed to display the defeat and victory screens at the end of the game.
In order to optimize the video system during development we decided to remove the concept of "video chip" as a special circuit. This decision helped to minimize the number of multiplexers, and as a consequence, to improve the speed of operation at high frequency.

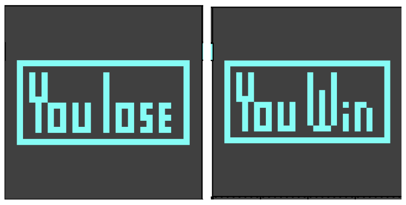


Ball output: When the x-coordinate of the ball equals the ID of one of the "video chips", a 32-bit representation of the y-coordinate of the ball is sent to the output.

Bat output: Bats are located on "video chips" with ID = 0 and ID = 31, therefore, for optimization reasons, the check for comparing IDs with these indices is run only on every 1st and 8th positions of the section. After that the four singular bits are moved by Y(coordinate of the racket) positions to the left.
Game Ending Screen: if the ending_game Input pin is up, data is taken to the bottom 8 inputs and later displayed on the screen.



The main task of the video system is to quickly and correctly display the state of the game on the screen.

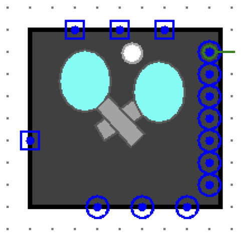


Kinematic controller

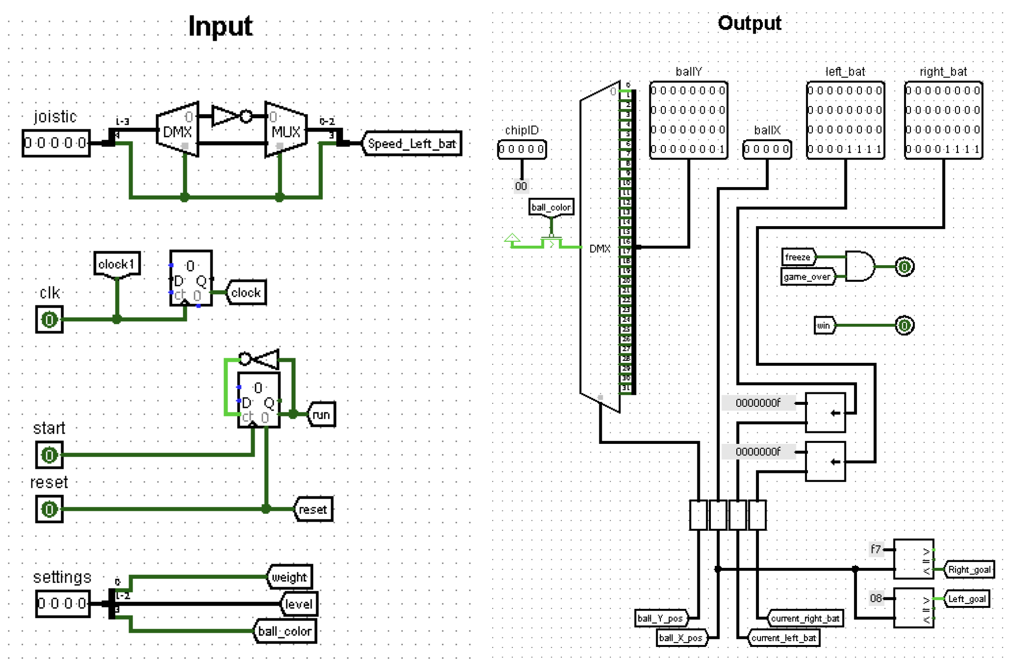
Controller is the link for generating, updating and storing data. It does all calculations of object positions, collision handling, and storage of previous object values.

The circuit is visually divided into 6 sections:

- Inputs (1)
- Outputs (2)
- Generating (3)
- Calculating (4)
- Score (5)
- Processor (6)

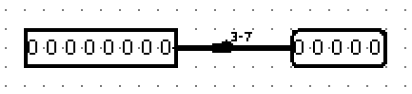


(1) Contains the current joystick position, frequency, reset contact and game start contact. And also a set of settings represented as a 4 bit number, where the first bit is responsible for the configuration of coefficients, the second two for the level of difficulty and the last for the color of the ball (red or blue). We use a 5 bit joystick representation so that the user can change the speed of the bat and influence the ball.

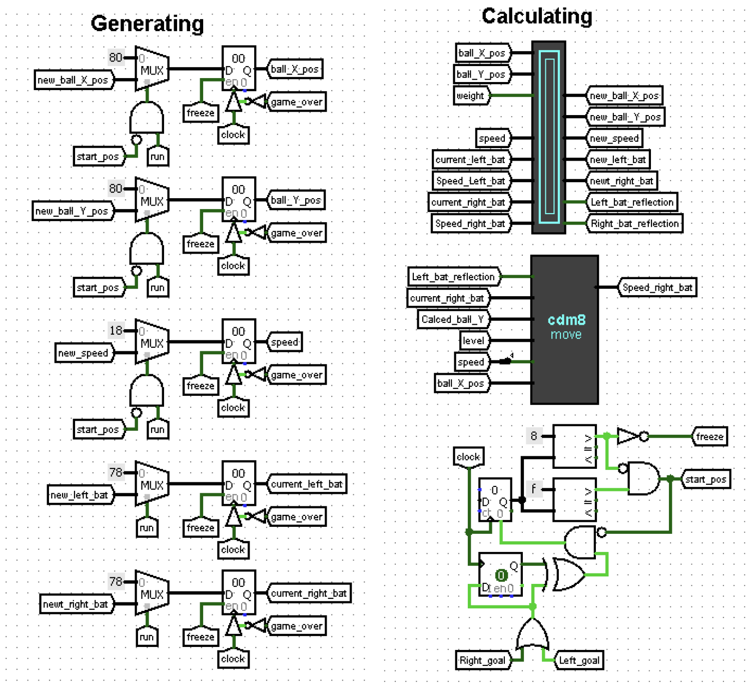


(2) Contains ball and bats positions, ID, frequency and game completion indicators.Also in this place we limit or allow the high signal to the decoder, in this way obtaining the color change of the ball.

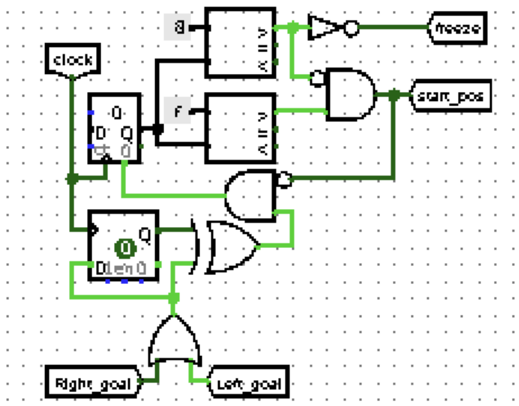
(3) The section where the positions of the bats and ball as well as the speed are generated, stored and updated. In our project, we use an 8-bit representation of the bats and ball coordinates; when the data is output to the video system, the lowest 3 bits are cut off by the 8_to_5 scheme. The velocity representation is more complicated, it is a 5-bit number, where the first bit is the direction of the X coordinate, the second bit is the direction of the Y coordinate, and the last three bits are the velocity modulus of the Y coordinate. Also on the scheme you can see the freeze tunnel, which slows down the data update for a while, and the end game tunnel, which stops frequency access when the game is over.



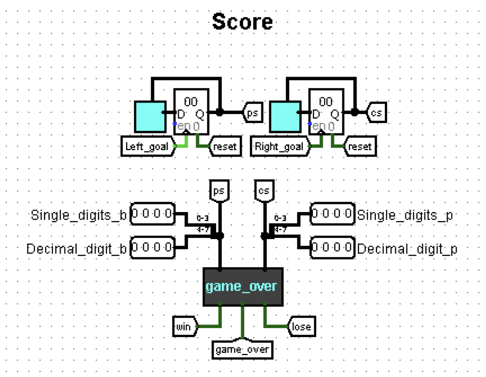
(4) Contains 2 subcircuits and freezer, which you can read about below.
- Calculation scheme
- CDM8 bat move

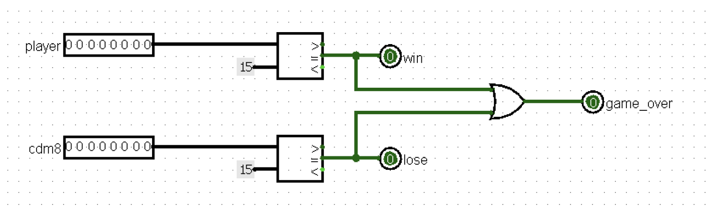


Next scheme is responsible for stopping the game after the goal for 16 ticks (on which most of the scheme works, varies depending on the level)

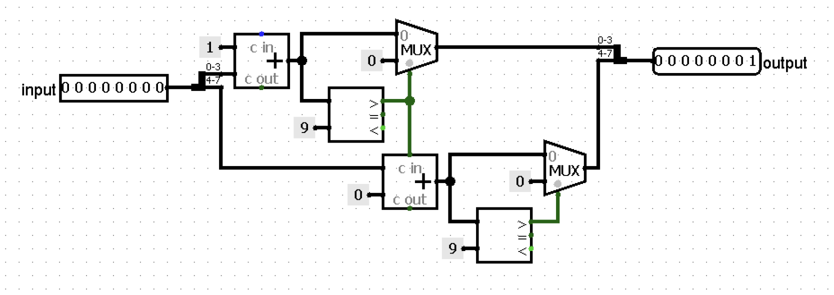


(5) It has two counters that increase their values when the wall collision indicator is raised. On top of that it contains a subcircuit that identifies the end of the game and a sub-circuit that increases the score by 1.



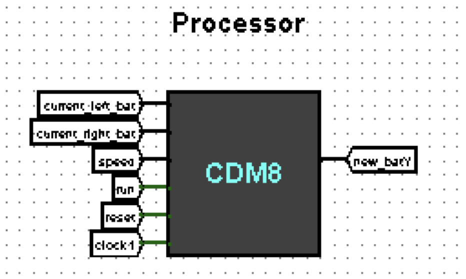


We store the score in BCD format.
How our scoring works?



The value of units is increased until they overflow, then the value of tens is increased by 1. The number for storing the score itself is an 8-bit storage, where the lowest 4 bits are for units, and the highest 4 for tens.

(6) **Section for the subcircuit in what the connection to the software part of the project is realized.**



Calculating circuit

Inputs

- **ball position (8 bits)**
- **weight (1 bits)** #determines the configuration of the coefficients
- **ball speed (5 bits)**
- **bats positions (8 bits)**
- **bats speed (4 bits)**

Outputs

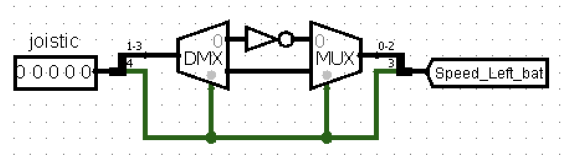
- **new ball position (8 bits)**
- **new speed (5 bits)**
- **new bats position (8 bits)**
- **reflections (of left and right bats) (1 bit)**

This scheme contains all the logic of ball and bats movement, as well as collision handling and recalculation of velocity vectors.

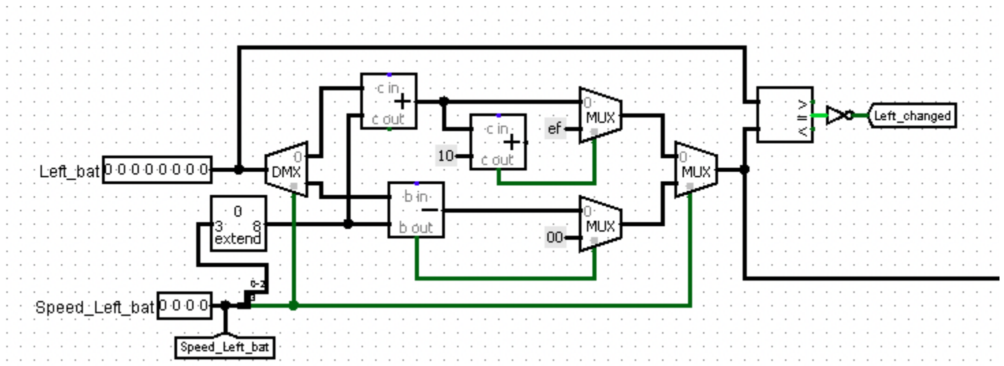
For easier understanding, let's divide it into logical parts and talk about each of them.

Left bat movement

The joystick values are manipulated in such a way that you can change the mouse speed with the joystick position. Bat speed can range from -7/8 pixels to 7/8 pixels per recalculation cycle.



After that we use this speed to change the mouse position.



How it works?

We separate the first bit responsible for the direction of the velocity from the velocity modulus, then expand the velocity value to 8 bits since the racket has an 8-bit representation. Then, using the velocity bit and the demultiplexer we split the calculations into addition and subtraction lines, increasing the speed of the circuits.

Right bat movement

After getting the value from the processor, we identify the velocity vector in which direction the bat should move.The circuit that is responsible for this is called cdm8_bat_move.

CDM8_bat_move

Inputs

- left bat reflection (1 bit)
- current right bat position (8 bits)
- calculated position (8 bits)
- level (2 bits)

- speed X direction (1 bit)
- ball X position (8 bits)

Outputs

- Speed right bat (4 bits)

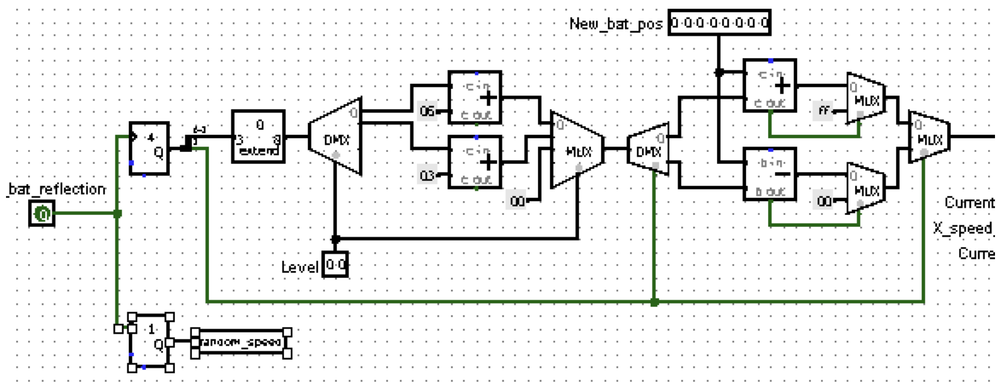
However, in addition to speed, this scheme has the mechanics of randomness and levels, because of which the bot can lose.

This scheme is divided into two important components.

The first of which is responsible for the random shift of the bat, which depends on the level of difficulty.

1 level -> [-13;-6] V [6; 13]
2 level -> [-10; -3] V [3; 10]
3 level -> the bot is never wrong

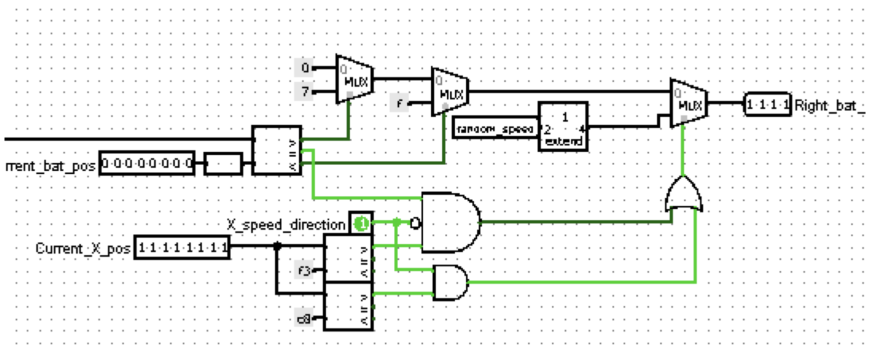
How we did it?
The random shift is recalculated each time the left racket hits, after that the random number is divided into two components, which are the shift direction and the random shift modulus. Next, the random shift modulus is added to the constants (6 at the first level and 3 at the second level) and then, depending on the direction of shift, we add or subtract this value.



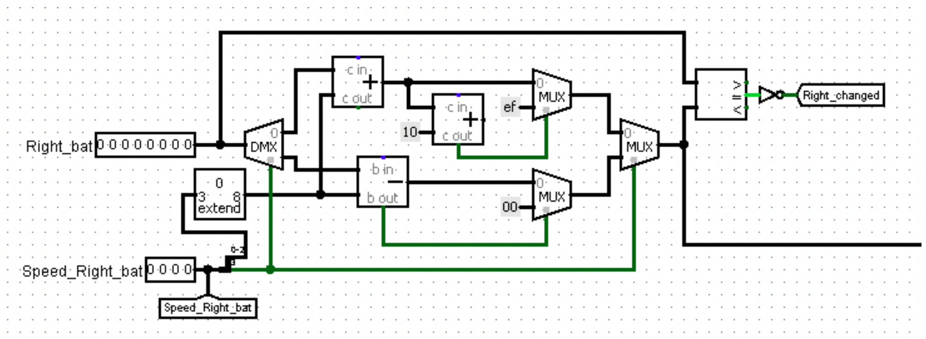
The second part of this scheme is responsible for the speed, the current position of the bat is compared with the value calculated by the processor + random number, if they do not equal, then the speed will be equal to 7/8 pixel + the right direction. If they are the same, the X coordinate of the ball is greater than 240 and it is moving towards the right racket, then the right racket starts

moving down with random speed to change the speed of the ball at Y and complicate the game.

The speed of the bot can vary from 1/2 pixel to 7/8 pixel per recalculation cycle.



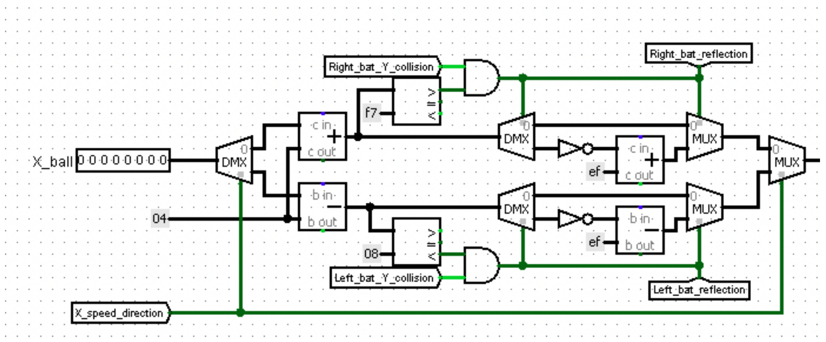
After that, the velocity vector is sent to the calculation scheme and the bat's position is moved in the right direction



Ball move

X Speed

Speed modulus of the ball on the x-coordinate is constant and equals 1/2 pixel in each operation, but depending on collisions with the objects speed vector changes its direction.

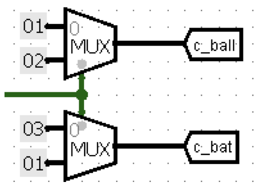


Y Speed

Ball velocity by Y coordinate can change in the range from -7/8 pixels to 7/8 pixels in one cycle of recalculation.

In our project we decided to change the speed only at the Y coordinate. In order to give the player more control of the game we added coefficients to the settings, they allow you to change the strength of the effect of the speed of the bat on the speed of the ball.

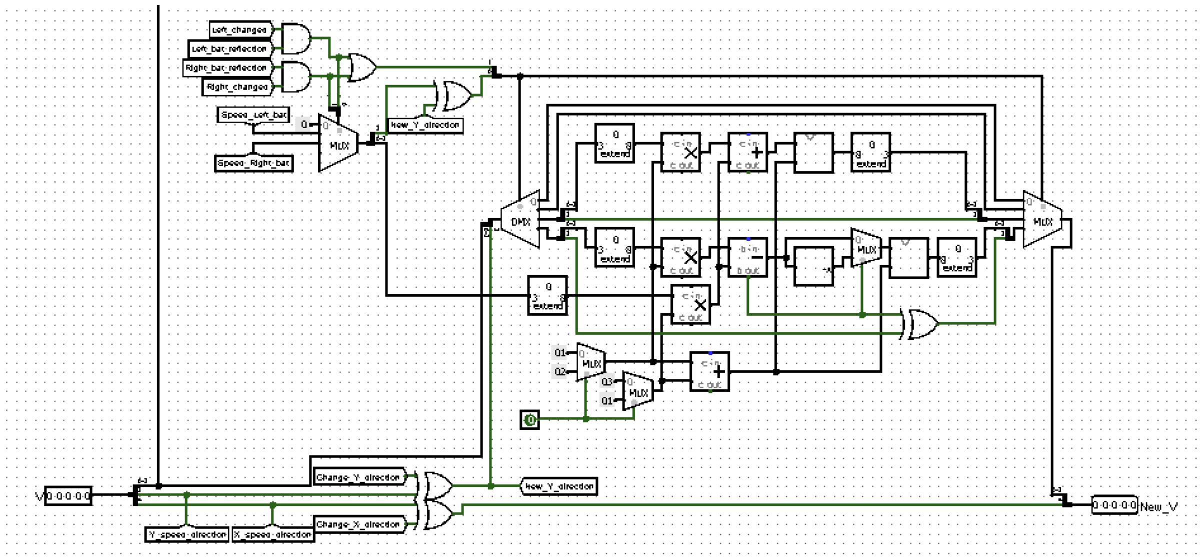
To simplify understanding we decided to make the settings less flexible and more understandable. That is why the user has only a few coefficient configurations to choose from.



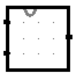
light weight -> 1 c_ball and 3 c_bat
heavy weight -> 2 c_ball and 1 c_bat

(C_bat * Speed_bat + C_ball * Speed_ball) / (C_bat + C_ball)

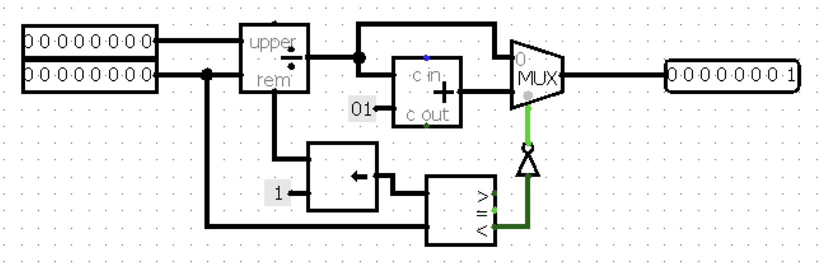
This calculation is made on the next scheme.



There is one unidentified circuit in this scheme ->

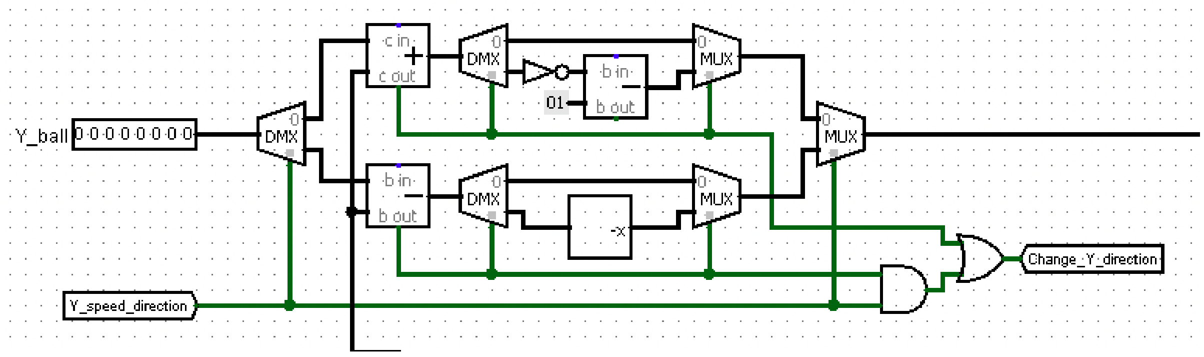


This is a rounding division scheme.
How it works?



The first number is divided by the second, and if it happens that the remainder of the division multiplied by two is greater or equal to the divisor, then it means that the fractional part of the number is greater than or equal to 0.5, consequently, the number must be rounded up by adding 1.

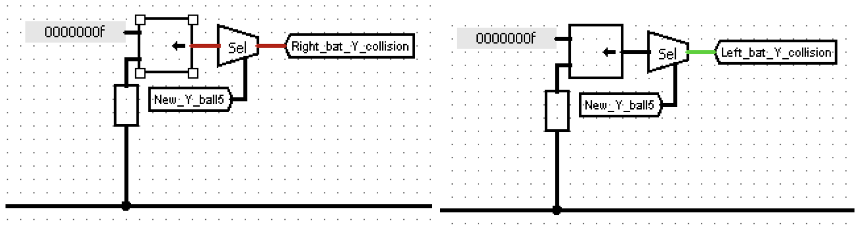
After calculating the velocity at Y, the ball coordinate at Y changes similarly to the ball coordinate at X.



Collisions

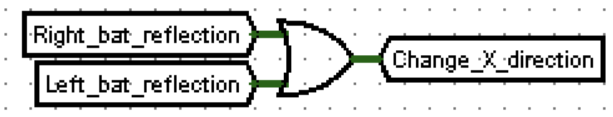
bats reflections

The input of the bit shifter is the bat location, after that we get a 32-bit representation of the bat and use the bit selector to check for a matching of the bat bit and the location of the ball.



Change X direction

If there was a collision with one of the bats, the X direction of the velocity should be changed.

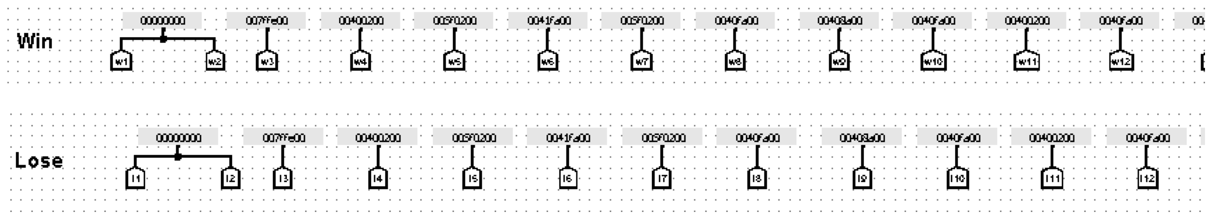


Change Y direction

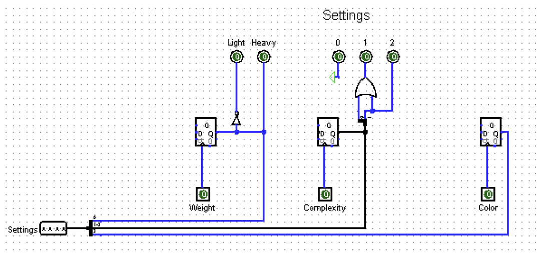
If an overflow happens when you add or subtract speed to the ball's Y position, it means that there was a collision with one of the horizontal walls and the Y direction of speed must be changed.

Game end screen and settings (Supplement)

To display the victory and defeat screens we calculated and wrote the values of each screen line into 32 bit constants, after the end of the game multiplexers pass to the output to the display not the state of the game, but these constants. This is how it works.



The settings panel gets data about the player's button presses and sends them to the kinematic controller.



Interfacing the game with the CdM-8 platform

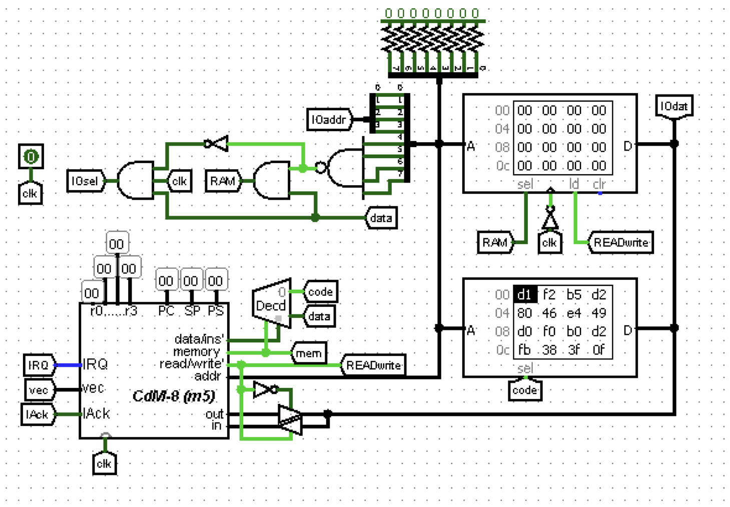
Inputs

- **ball position**
- **ball speed**

Outputs

- **new bat position**

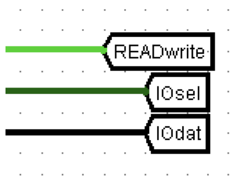
In our project we use a CDM8 to define the position in which the bot must be in so that it can kick the ball.

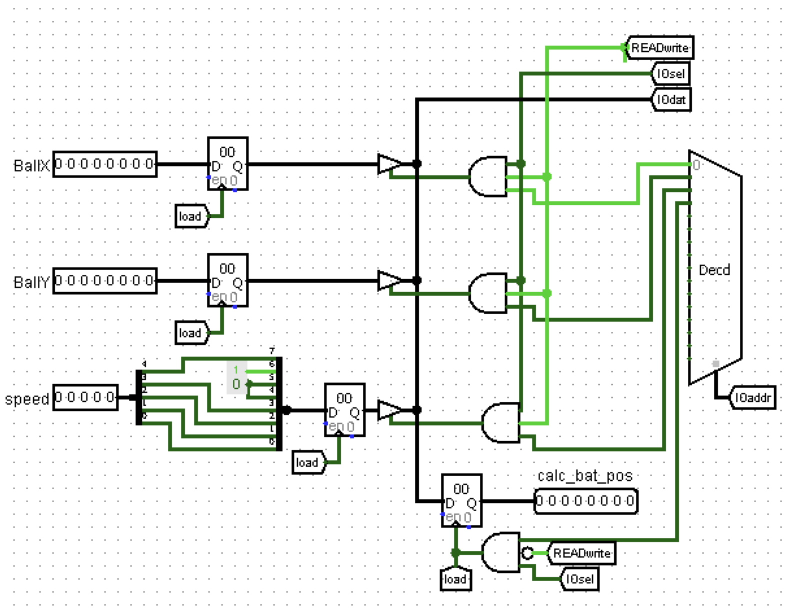


This connection includes 3 input values (location in memory F0 - F2) and one output value (F3). All I/O data goes on one bus which blocks or allows the path depending on the signals. In order to make all the data belong to the same point in the game, registers are used, which is updated when a new mouse value is calculated.

How it works?

In order to understand the connection it is necessary to understand the meaning of each of the signals. IOsel is made to recognize the type of commands that are currently running (code work or input, output). readwrite is for recognizing the type of command - read or write. IOaddr is the cell number in the input/output line. IOdat is the data bus where data is written to RAM as well as taken from it.





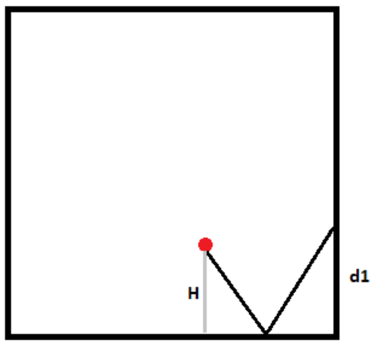
Then these signals are checked by the logic elements and if the cell address matches, the buffer passes the data on the bus and data is read or written depending on the signals.

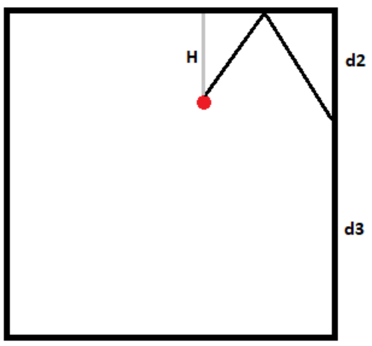
Software

The software part is responsible for calculating the point at which the ball will be when it reaches 31 verticals (X_ball ≥ 248).

Algorithm

We need to calculate this point using the current position of the ball and its speed (X_ball, Y_ball, V). The velocity along the Y-axis can be directed both positively and negatively (the fourth bit is responsible for the direction along the Y axis). Let's imagine that we are able to solve the problem in the case of positive velocity.





In the case of a negative direction, we can reduce the problem to a problem with a positive direction using a reverse.

A reverse is a change of direction of speed and

position of the ball to a symmetrical one relative to the axis passing horizontally through the center of the field.
So X_ball_reversed = X_ball, Y_ball_reversed = 255 - Y_ball, X_speed_reversed = X_speed, Y_speed_reversed = -Y_speed, d1 = d2 = 255 - d3. We can get the answer to the original problem by reversing the answer (d3).
Now let's consider a problem with only a positive velocity direction. At first calculate k = number of steps through which the ball will be on the 31 vertical, the minimum k, such that X_ball + k * X_speed ≥ 248. X_speed = 4, consequently k = (251 - X_ball) / 4. Then we need to add Y_speed to Y_ball for k-times using the cycle. In case of overflow at some step of the cycle
 (Y_ ball_new = Y_ball + Y_speed > 255, reflection from the upper side), you need to calculate the correct position of the ball and reverse the problem since the direction of the ball has become negative. For example Y_ball = 253, Y_speed = 7 then Y_ball new = 250, Y_speed_new = -7, make the reverse (Y_ball_reversed = 5, Y_speed_reversed = 7) and continue to add Y_ball to Y_ball_reversed in the loop. During the algorithm it is necessary to count the number of reverses that we have made. If this number is odd we reverse the answer.
Note, we can add k to Y_ball for Y_speed-times instead. This can greatly increase performance because of the fact that k can be an order of magnitude more than Y_speed (0 ≤ Y_speed ≤ 7, 1 ≤ k ≤ 62). We will use it in our code.
Code
Due to the fact that the player can direct the ball to any point, the bot cannot predict where the ball will be until the player has hit it.Therefore, when the ball flies to the player (significant bit of V is 1), the program returns the value 128 (center of the field), this position is the most advantageous. If the ball flies to the bot first we count k. After that, we reverse if the Y_speed direction is negative and start counting the number of reverses. Then add k to the current Y_ball for Y_speed-times. In case of overflow, we easily calculate Y_ball_reversed by adding 1 and increment the counter of reverses. At the end we reverse the answer if this counter is odd.

Conclusion

We have created a game where the user is comfortable playing against a bot. You can choose the level of difficulty of the game, the speed of the ball, as well as change the color of the ball – all this allows the player to customize the game to suit himself. And also worked on the visual part of the project: chose balanced colors and added the display of “You lose” and “You win” at the end of the game, depending on the loss or win respectively.

Thanks to this project, we have learned to work in a team, to properly share responsibilities. We have consolidated our knowledge of working with Logisim and CdM-8. And we also realized that we had received useful information on the “Digital Platforms” course, which we are able to apply in practice.