Novosibirsk State University
*THE REAL SCIENCE

Group Project B
“The Game of TV-Tennis”

N. Korneich 20931
Y. Timofeeva 20930
A. Karmanova 20930

2021

Content

Introduction……………………………………………………………………3
Project description……..……..……………………………3
Our main goals………...…………………………………….3
Tools we used……………………………………………....4
Relevance of our project…………………………………..4
Hardware description……………………………………………………....5
The display panel, the bats and the ball……………………....5
The video subsystem……………………………………….....6
The video chip………………………………………………....7
Kinematic controller……………………………………………..8
Position of the ball……………………………………………….9
Calculation of the ball movement……………………………10
Reflecting the ball……………………………………………...10
Changing the ball's velocity…………………………………...11
Counting scores……………………………………………….11
Interfacing the game with the CdM-8 platform……………...12
Software description………………………………………………………14
Conclusion………………………………………………………………....15
Sources and developing applications……………………………………...15

Introduction

Project description

Our project is an implementation of the TV-Tennis game using tools such as Logisim (a program for modeling electronic circuits) and the CdM-8 assembly language. TV-Tennis is one of the names of the first ever created video game (the game of tennis), which was first invented in 1969 by Ralph Bering and was called "The Game of pong", it was for two players. In our project, there can only be one person who opposes artificial intelligence.
In the hardware part of the project, we implemented a 32x32 video display and a kinematic controller chip that moves the "rackets" and the ball around the display, and also contains a counter (which determines the end of the round) and scoring. We used a set of hexadecimal indicators to display the current points for two players and a timer.
In the software part, there is an AI for the opponent player, which predicts where the ball will fly, and positions the robot player's racket to hit it.
In this project, we are trying to reproduce some of the features of TV tennis, as well as make the computer play it with a person as much fun as possible

Our main goals

Our main task was to create the most optimized game and at the same time make it look good and work stably at any values. At the beginning of the work, we made the following plan:
1) Distribute the roles in the team, choose a commander
2) Get acquainted with the features of the Logisim program
3) Write a video chip and connect it to the display
4) Make a chip (kinematics controller) that calculates the position of the ball and rackets
4) Connect the kinematics controller to the display
5) Connect the cdm8 platform
6) Write a program for AI and connect all the data
7) Make a scoring system and win position
8) Make the scheme pleasant to the eye and write the documentation

Tools we used

Logisim

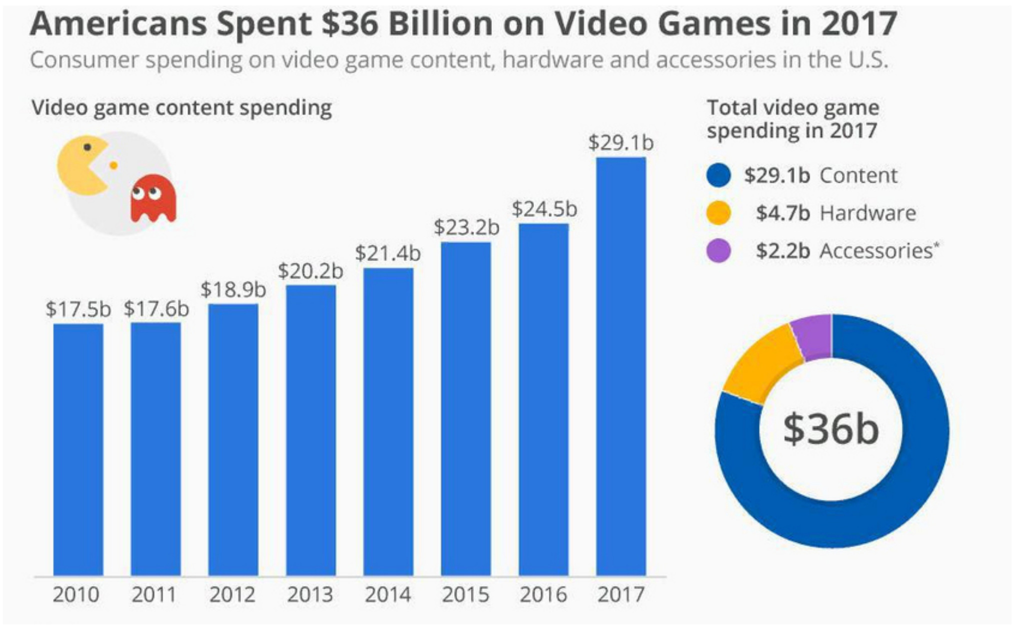
A tool that allows you to design and simulate digital electronic circuits using a graphic user interface. Logisim is free software released under the GNU GPL; can run on Microsoft Windows, Mac OS X, and Linux. The code is written entirely in Java using the Swing GUI library. The main developer, Carl Burch, has been working on Logisim since its inception in 2001.

Assembler (CDM-8)

A low-level machine-oriented programming language we use for writing AI for a robot player. It is a notation system used to represent programs written in machine code in an easy-to-read form. Its commands correspond directly to the individual commands of the machine or their sequences. It is substantially platform dependent.

Relevance of our project

In today's world, video games have become one of the largest segments of the economy and entertainment industry.



In terms of the degree of influence on consumers and their involvement in the interactive environment, video games have long stood out from other forms of entertainment.
The industry is constantly evolving: e-sports is spreading, the field of VR and augmented reality is expanding, research is emerging on the positive impact of video games in the field of education.
The gaming industry is a promising educational, entertainment and economic sphere of human activity.

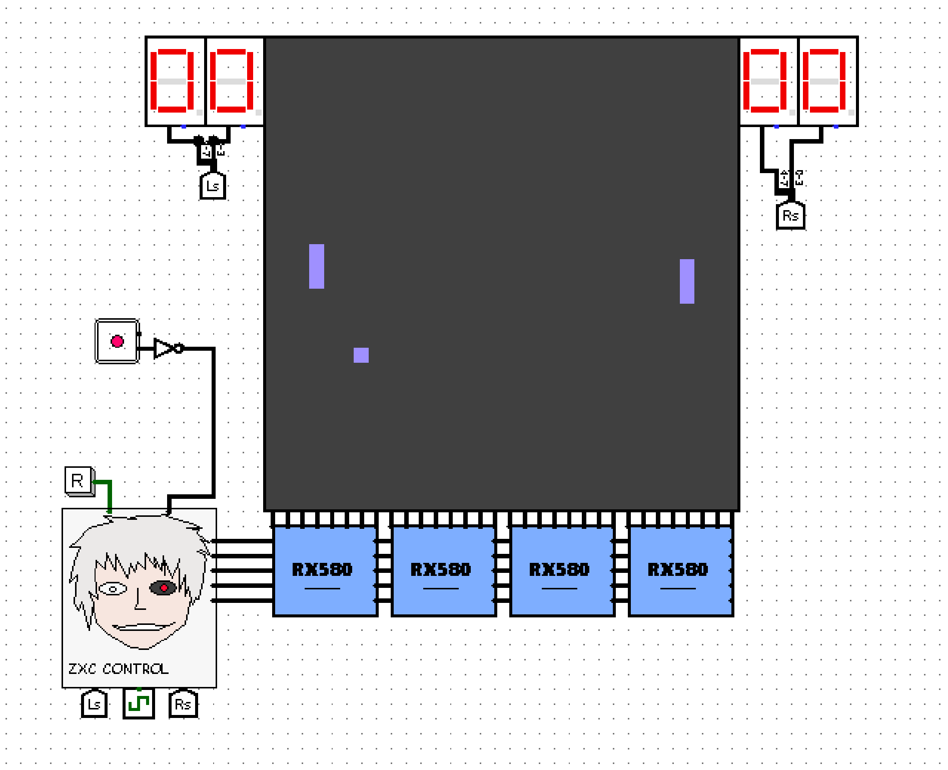
Description
Hardware

The display panel, the bats and the ball
We have a general-purpose display panel(pic.1), 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 pin, with one bit per pixel. The pattern displayed by a column is dictated by the 32-bit pattern that is asserted on this pin (the highs in the pattern switch pixels on and the lows switch pixels off). Given the right input signals this panel could display any 32x32 pixel pattern.

For TV-tennis we need to display 2 bats and a ball against a plain background. The ball is drawn as a single pixel, and each bat is drawn as a vertical line made up of 3 adjacent pixels in a single column. The ball can appear anywhere on the screen, but the bats can only appear in fixed columns (left bat in column 3 and right bat in column 28).

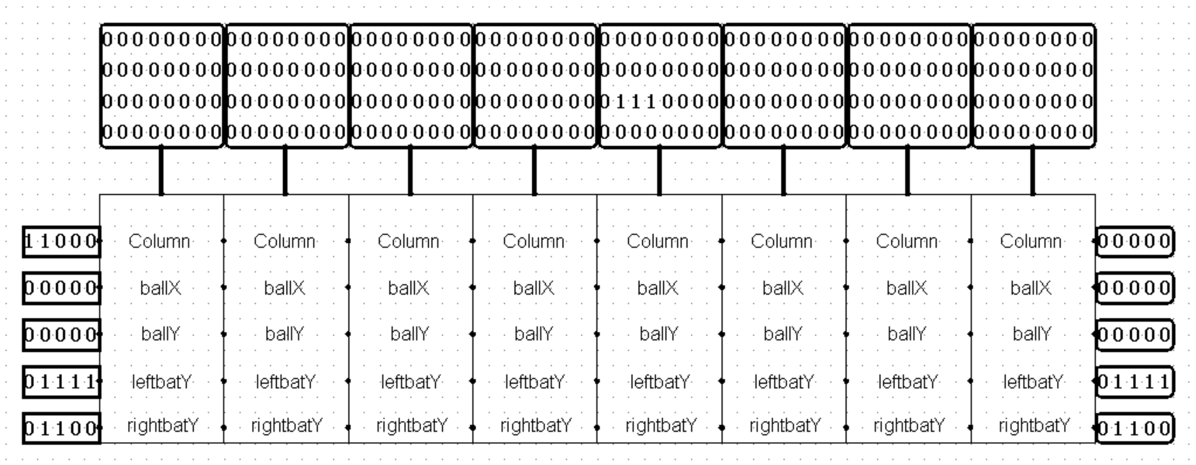
To display the ball we need to switch on 1 pixel out of 32 in a specific column on the screen. We need to know the column (x coordinate) and row (y coordinate) in which the ball should appear. To display one bat we switch on 3 adjacent pixels out of 32 in a fixed column, so we just need to specify the row (y coordinate) in which the bottom end of the bat should appear. So we need four 5-bit numbers to specify the positions of the ball (ballX, ballY) and the two bats (leftbatY, rightbatY), as well as an input pin labeled ColumnID for chip numbering. These will come from the kinematic controller.



(pic. 1)

The video subsystem

The video subsystem(pic. 2) has five 5-bit inputs: Column, ballX, ballY, leftbatY, rightbatY. The display panel is controlled by a set of 32 video chips, and all five of these inputs are connected to each video chip. However, to reduce the number of wired connections, the video chips are chained together. The chips are arranged in a chain from West to East; the West-most chip receives all the input data and passes it to the next chip in the chain.



(pic. 2)

(pic. 3)

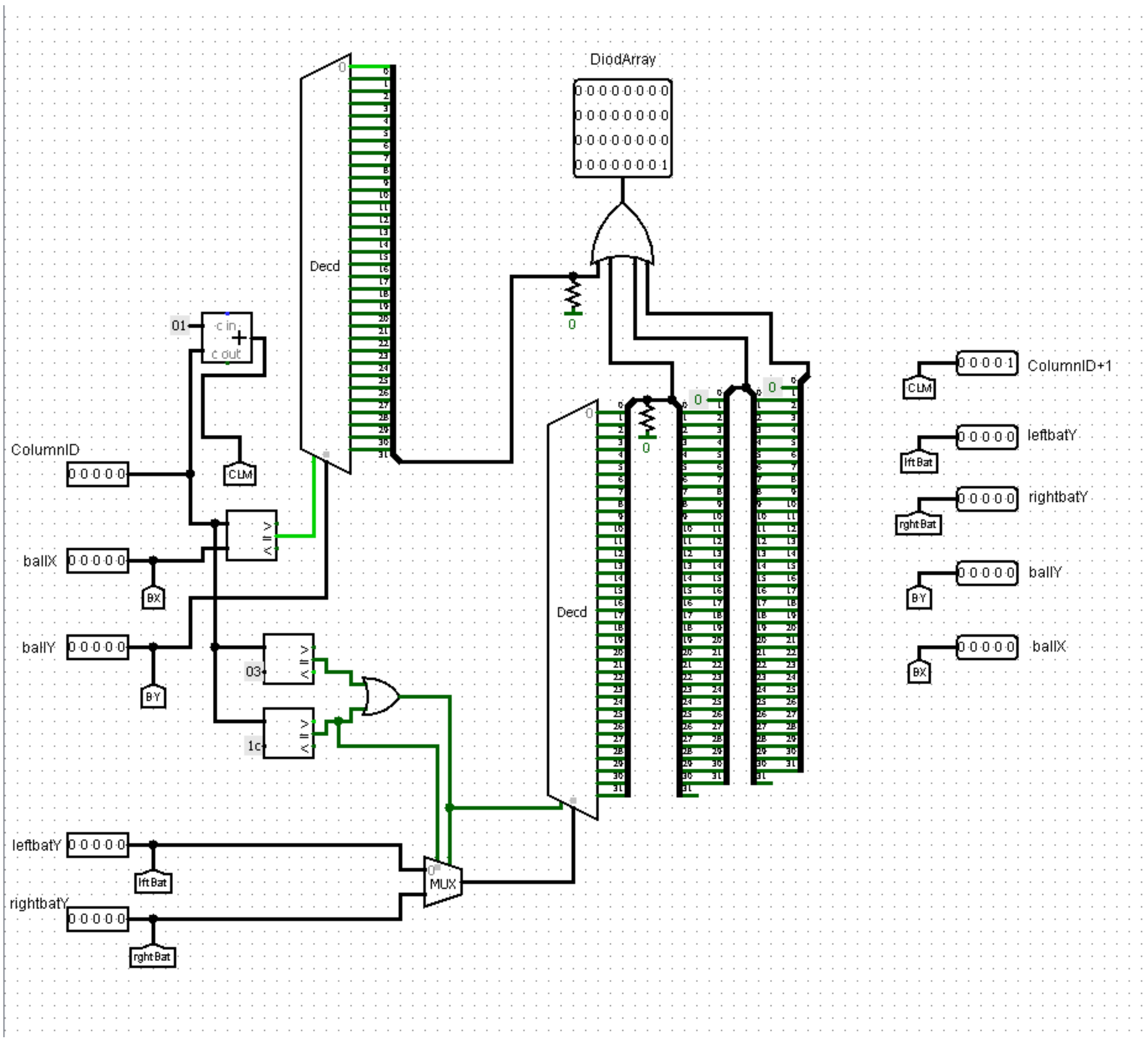
Instead of combining 32 separate video chips, we use a cellular structure. 8 chips are connected in a chain, forming a video section(pic. 3). Then 4 of these sections are connected in a chain. So we designed a single cell, added it to the schema library, and then created a large schema by joining together several identical cells in a regular pattern. Each section is indicated on the schematic diagram by a blue rectangle.

The video chip

Every video chip(pic. 4) has ColumnID, ballX, ballY, leftbatY, rightbatY input pins on its West side and ColumnID+1, ballX, ballY, leftbatY, rightbatY output pins on its East side, and the bit patterns on each of them pass through each video chip unchanged.

Each video chip drives a specific column of the display panel, and each chip needs to ‘know’ which column it is driving, so the chips are numbered 0..31 like the columns. Rather than store a different column number in each chip individually we give each video chip an additional 5-bit input pin labelled ColumnID on its West side, and add a corresponding output pin on its East side, but this time we add 1 to the number before outputting it again.

Then when we connect the chips in a chain we make sure to input 0b00000 on the ColumnID pin of the West-most chip (= column 0), which adds 1 and so sends the signal 0b00001 to the next chip in the chain. This way all 32 chips number themselves automatically.



(pic. 4)

A video chip has no internal state and produces a 32-bit output pattern that drives one column of the display panel. That pattern is output on the single North pin DiodArray. Each chip includes circuitry to display a ball and circuitry to display a single bat. The ball's position is received on input pins ballX and ballY. Each chip compares its ColumnID to ballX, and if they are equal it lights up pixel number ballY in its column of the display.
This is a straightforward piece of combinational logic.

Because the bats are in fixed columns we can 'hard-wire' their column numbers (3 and 28) into the video subsystem. Every video chip contains two 5-bit constants: 0b00011 and 0b11100. These two constants are compared with the ColumnID, so the video chip that has 00011 as its ColumnID displays the left bat with its bottom pixel in row leftbatY, and the video chip that has 11100 as its ColumnID displays the right bat with its bottom pixel in row rightbatY. This too is achieved by combinational logic.

Kinematic controller

This controller (pic. 5) is one of the most important parts of this project, its tasks are to move the ball by updating its coordinates, to block the right racket when the ball is on the right side of the screen, and to pass the positions of the ball and the two rackets to other parts of the system (the program and the video subsystem).

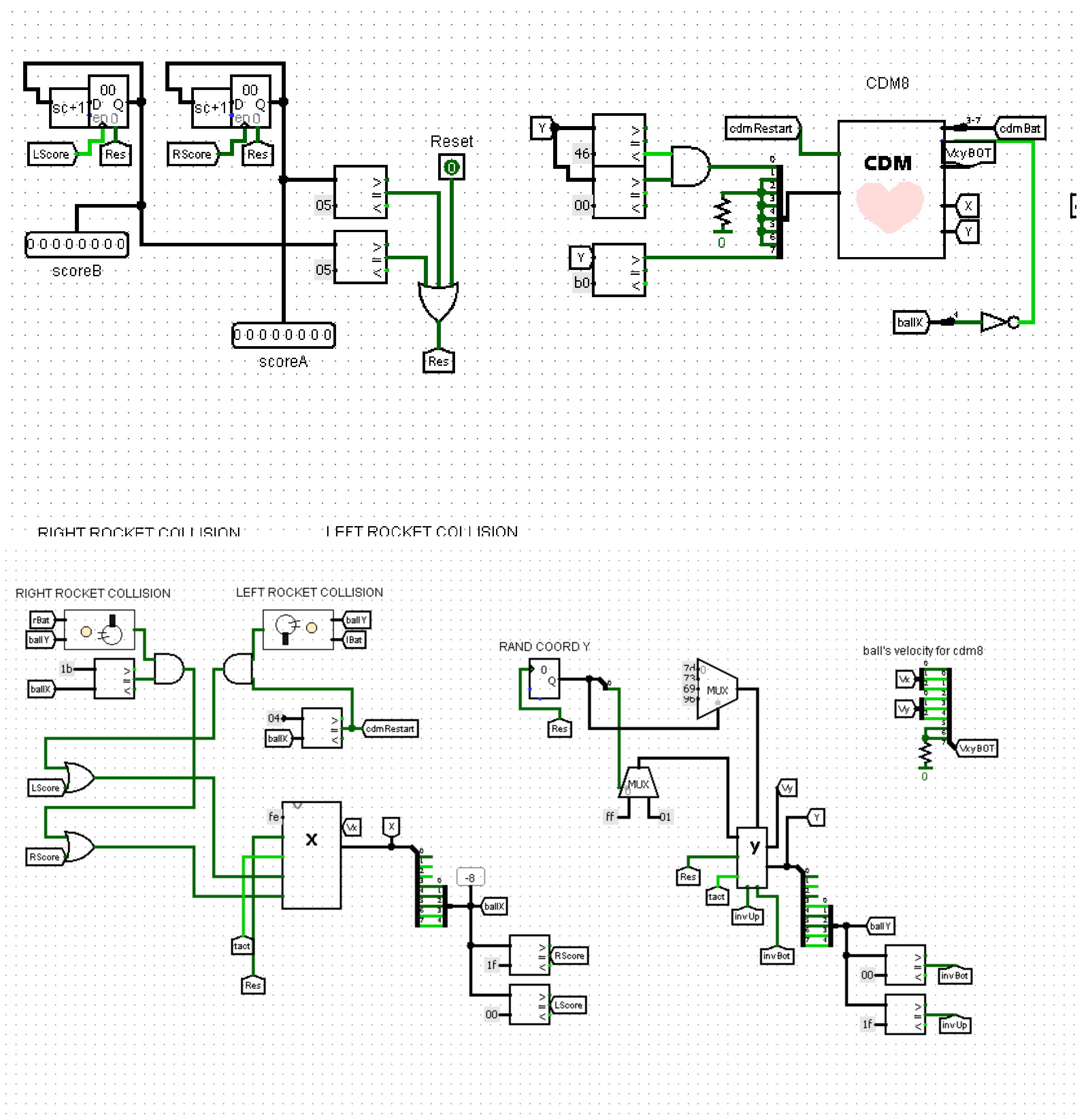
Kinematic controller returns:

- Numbers ballX, ballY – ball coordinates for video system (5-bit length).
- Bat's Y-positions (leftYout and rightYout).
- Scores of both players as scoreA and scoreB
- A five-bit number Zero for numbering columns and working with them in the video chip

The rest is done by other pieces of hardware and software: the left racket is controlled directly by the joystick, and the right racket is controlled (via the I/O interface) by the CdM-8 core, which runs the robot player program.

The controller updates the coordinates and velocity of the ball and reports them to the program, which uses these values to predict the trajectory of the ball and the corresponding movement of its racket. The kinematic controller needs to know the position of both rackets to determine whether the ball has hit one of them.

Two values are fed to the controller: leftYin (the coordinates of the left racket, determined by the joystick) and clock cycles, while the controller outputs a constant value of zero (for subsequent numbering of ColumnID in the video subsystem), ballX, ballY, leftbatY and rightbatY. The controller also contains subcircuits for scoring points, predicting the trajectory of the ball for the robot player, schemes for the collision of the ball and changing its velocity.



(pic. 5)

Position of the ball

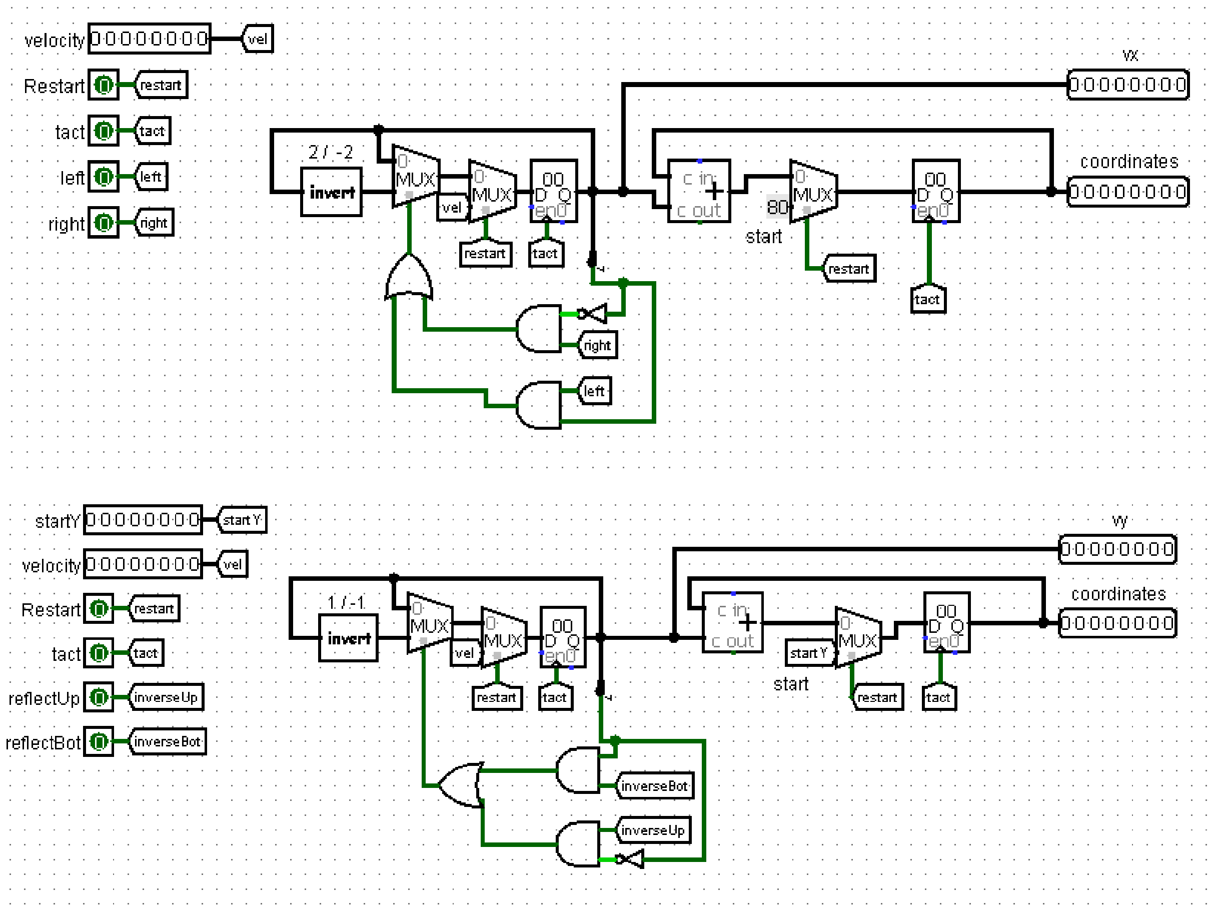
The ball moves around a square space of LED matrix that is divided up into cells by a 256x256 grid, with the cells numbered 0..255 from left to right and 0..255 from bottom to top, so the bottom left cell has the coordinates 0,0. The x and y coordinates of the ball are held as 8-bit unsigned numbers, and all movement calculations and position updates are performed on those 8-bit values.

Calculation of the ball movement

The movement of the ball on the display is determined by the ball Movement X and ball Movement Y schemes(pic.6).

The schemes receives the input:

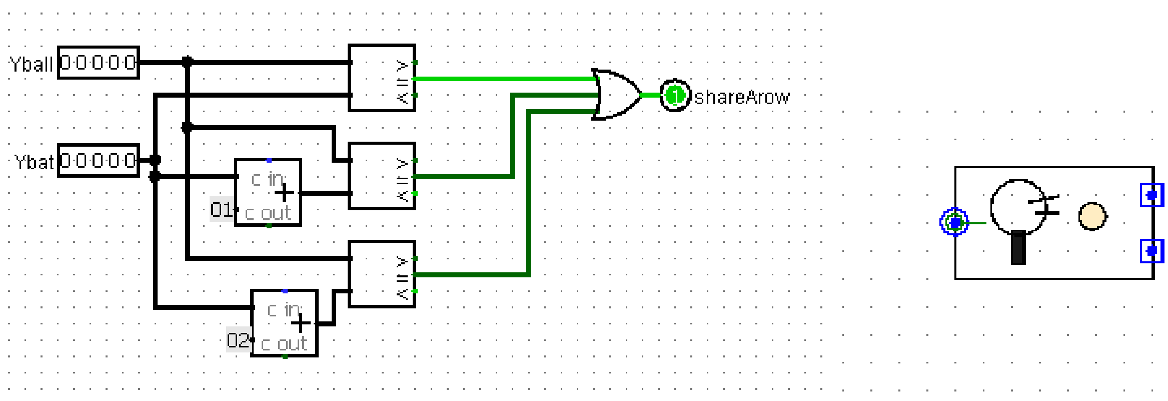
- the starting value of the ball's position, which is the center of the display
- flags of collision with walls or rackets for subsequent changes in its direction
- the initial velocity when starting the program for changing the coordinates during operation
- restart flag for resetting values



(pic. 6)

Reflecting the ball

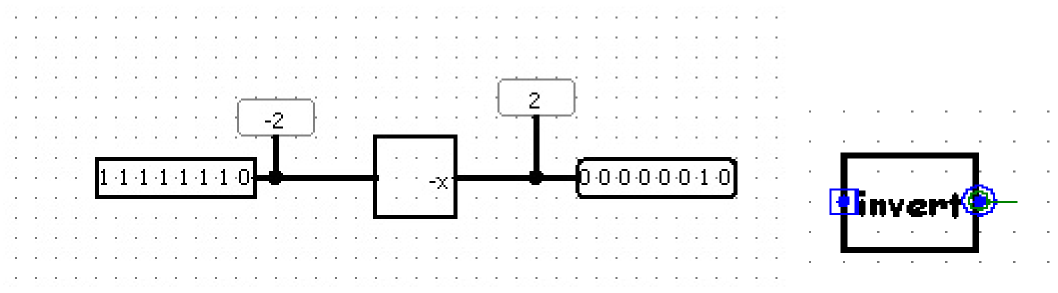
The inrange scheme(pic.7) receives the input 5-bit values YBall, YBat, denoting, respectively, the Y coordinate of the ball and the bat's coordinate (the coordinate of the lower pixel). Since the bat consists of 3 adjacent pixels, we provide for a situation where the ball can touch any of them, for this we compare its coordinate with each single pixel of the bat. In case of touching the bat, scheme passes 1 to the output contact, if the ball doesn't hit the bat, output stays with 0.



(pic. 7)

Changing the ball’s velocity

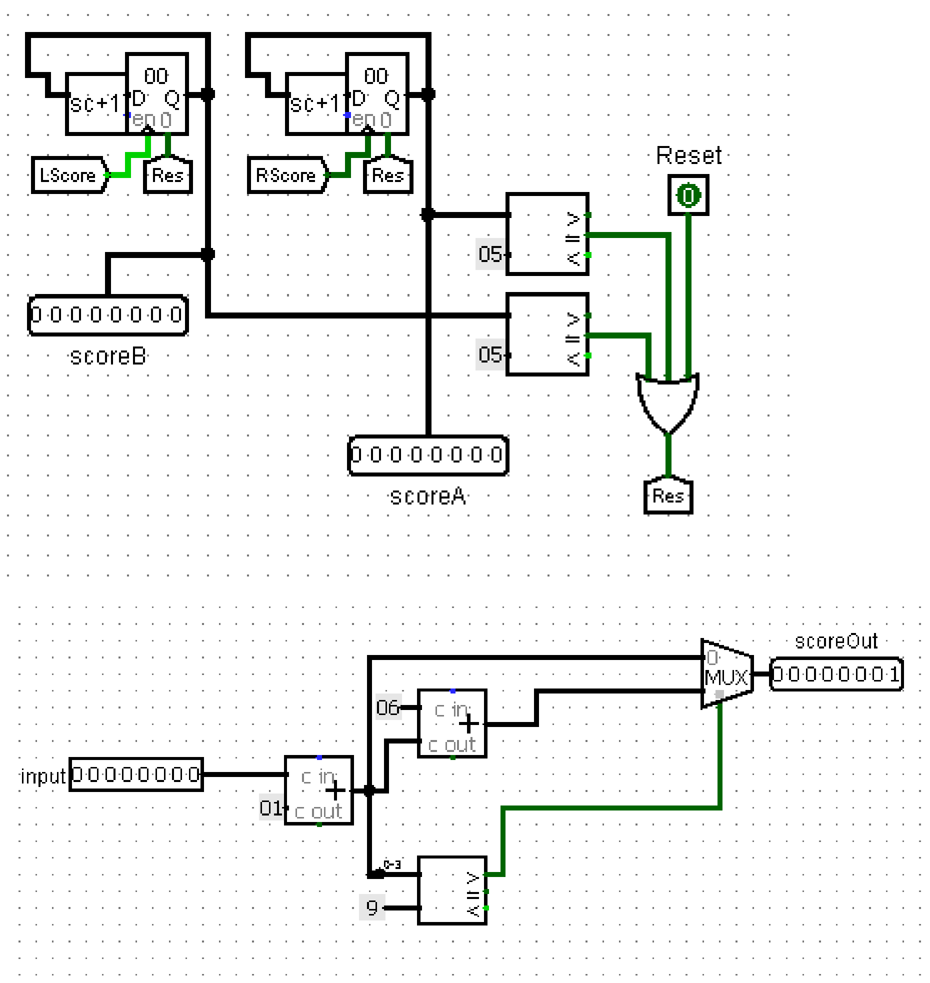
The essence of the invert scheme (pic.8) is to change the direction of the ball movement using the negator. The input is the speed of the ball(positive or negative, depending on direction), and its inverted value is returned. For the x coordinate, the direction of movement can change to the right or left, for y - up or down, respectively.



(pic. 8)

Counting scores

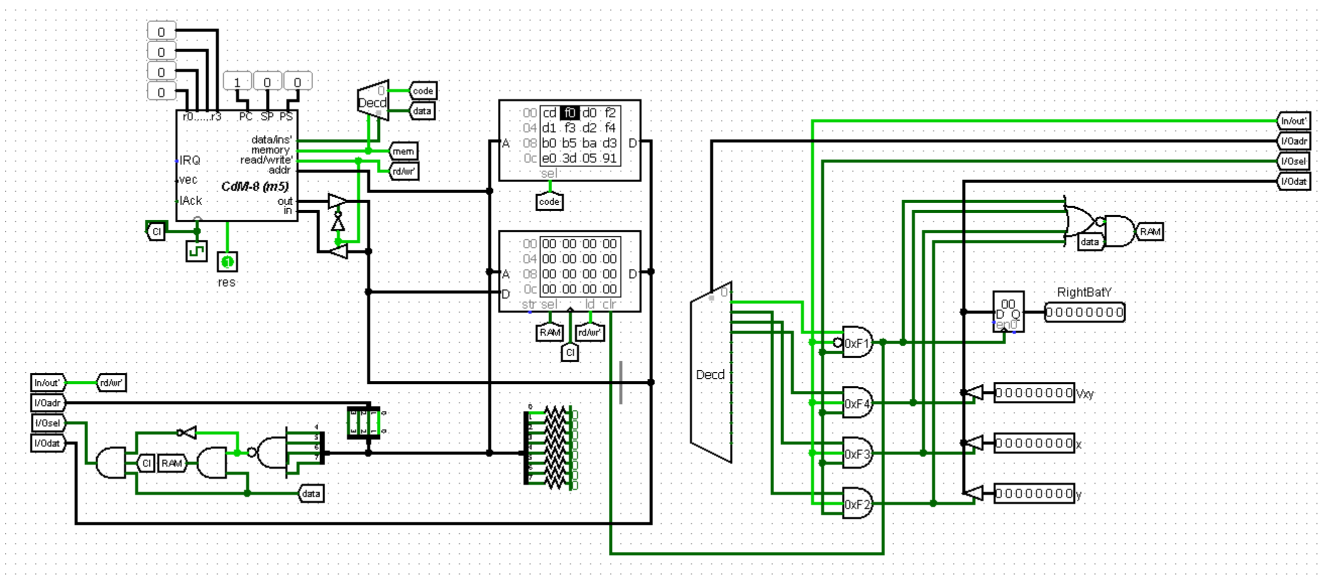
The player's score in the game is the number of reflections of the ball from the vertical wall in half of the other player's movement zone. The controller includes two counters for keeping a score. The trigger for the score counters comes from hitting a wall. Each time a collision with the wall is detected (this is done by comparing ballX and the extreme columns with coordinates 0x00 and 0x1f). As a result, we get a collision signal, which is fed to the register shown below (pic.9) and add one to the previous result. Since we operate with 8 bits of numbers, we need to add 6 to the count, because we should not see the letters on the screen. At a certain point value, the game will be reset to zero.



(pic.9)

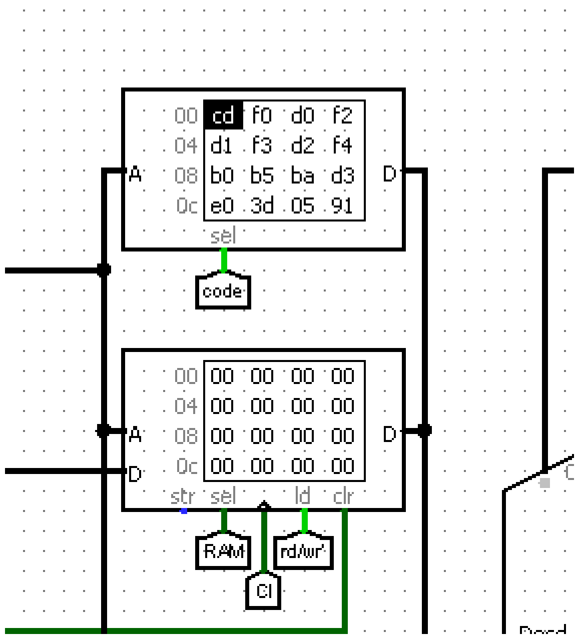
Interfacing the game with the CdM-8 platform

To implement a robot player that controls one of the rackets, we use the CDM-8 platform, implemented in hardware of Logisim.(pic.10) The interface includes one output register, into which the program will write the Y coordinate of the right bat. This is an 8-bit register (in our case, its address is 0xF1), which should use the most significant bits of the I/Odat group. It is also necessary to use two more registers so that the program can read the 8-bit x and y coordinates of the ball, these are 0xF3 and 0xF2 in our project, respectively. All the interface needs to do is decode the I/O address / direction of transmission and confirm these values in the I/Odat group, then execute all instructions specified in the memory bank.

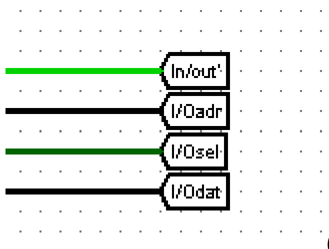

(pic.10)

Read/Write block

One of the most important parts of the cdm8 platform is the read / write block (pic. 11). These are the parts of the chip that work in pairs, the ROM (top memory bank) contains a set of 8-bit instructions, our program, and the RAM (bottom) stores temporary data while executing the instructions above.
The flags (pic. 12) are needed to determine the type of instruction, In/Out is needed for a read or write signal, and I/Osel is used to select which of the commands is being executed - read or write. The four-bit I/Order and I/Dat signals store the address of the memory cell that we are currently working with and its value, respectively.



(pic. 11)

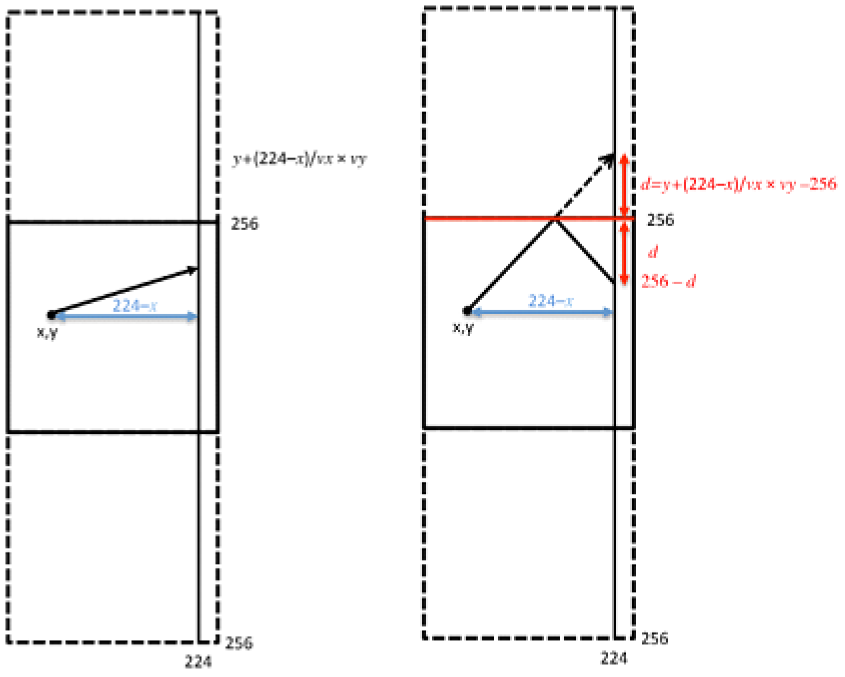


(pic. 12)

Software

The prediction and calculation of the ball trajectory are calculated by a program running on the CDM-8 processor, which determines the intersection point of the 28th vertical of the display to determine the position of the right racket. To do this, we supply the coordinates of the ball and its velocity to the program input. Based on this data, the algorithm calculates the Y coordinate for the right bat to successfully hit the ball.
Basic formulas for trajectory prediction: $d = by + (228 - bx) * vx/vy - (0/256)$

- by, bx – current Y and X positions of ball
- 224 – movement line of right bat (X-coordinate, a current distance between ball and bat)
- vy, vx – current ball velocities.



The ball will need (224 - x) / vx bars to reach the racket line. Its vertical offset during this time will be (224 - x) / vx * vy. This offset, when added to the current y coordinate, determines the intersection point on the racket line that the ball will hit (formula: $d = y + (224 - x) / vx * vy$), if there is no collision with the top or bottom of the screen. In case of a collision, we do negation (~ d), to move the ball in the opposite direction.

Basic algorithm:
- Read the input data
- Subtract from 224 the coordinate of the ball on the Move
- Check the sign of the speed by, in the case of negative, we invert the value (- (224-x)/vx*vy)
- Division by the speed
- The sum of the offset of the racket and the old Y coordinate of the ball
- Write to the register(0xF1) and terminate the program

Conclusion

This project seemed to us very exciting, but at the same time difficult. But you can use this documentation as an instruction for writing your own project on the Logisim platform. You can expand your project with your own ideas and possibly optimize it, however, our project is completely ready to work. Thanks!

Sources and developing applications

Books:

- Co-design Group Project B “The Game of TV-Tennis” Notes written by Alex Shafarenko & Steve Hunt, 2016
- “Computing platforms” A.Shafarenko and S.P.Hunt

Tools:

- CocoIDE v.1.91
- Logisim v.2.7.1