Novosibirsk State University

4COM1042 [Computing Platforms]

Co-design Group

Project B “The Game of TV-Tennis”

Anton Pimonov 21931 & Mikhail Komarov 21931
2022

Contents

All scheme, that were mentioned in text, are highlighted by cursive and underscores.

Introduction

For our group project, we were able to choose one theme from three basic ones or come up with something original. We decided to implement TV-Tennis, because we didn't find other two basic themes interesting and because we think TV-Tennis is great, yet simple example of using everything we have passed through the program of our course.

As you can see, we successfully implemented TV-Tennis on Logisim + CdM-8 platform (Logisim for circuits, CdM-8 for code).

We divided our showcase in 3 parts: overview, hardware and software.
We'll begin with overview.

Overview

At first, we will set the rules:

It is a simple tv-tennis. Game over, when someone get 24 points. You need just hit the ball and don't let the ball hit your wall. All movements are made by joystick. Be careful: if the ball stuck between bat and wall, it can give more points to the opponent.

Also, we should set frequency to 256 Hz. This value is a good balance between speed and comfort of the game to player.

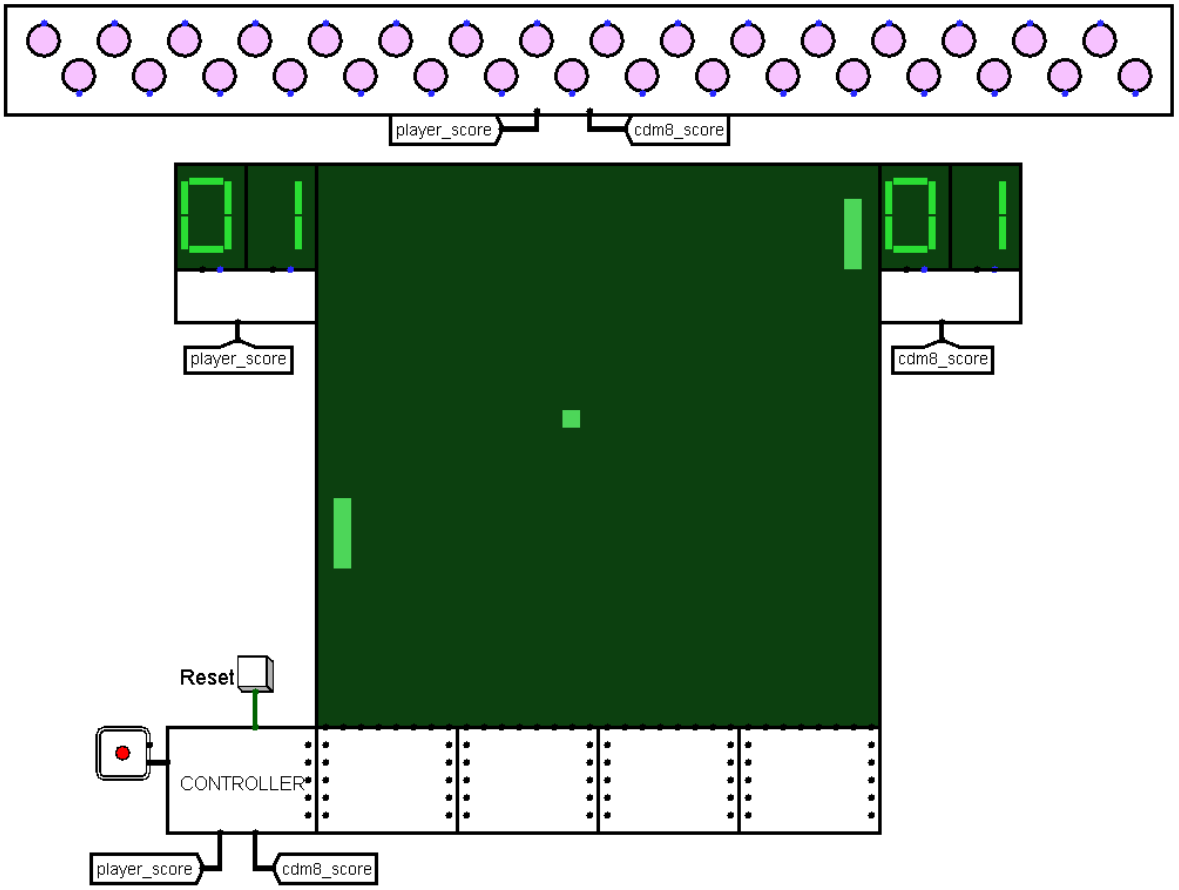
Let's split components that player is able to see on "user" and "technical".

"User" components: video display, joystick, restart button, score, led.
These pieces have meaning to player, he is able to see what happens because of display, score and led, to control the bat because of joystick, and to restart the game because of restart button.

"Technical" components: video chip and kinematic controller.
These pieces do all work, this is where everything being calculated (ball movement, bats movement etc.). It has nothing to do with the player, he just does the inputs and get the results, he has no need to see or understand what happens between those things.

All of this you can see on the next page (main circuit, actually).

Now let's move to the hardware part.



Picture 1

Hardware

Display

What do we need to see on display? The ball and the bats.
How can we display it? By deciding what the ball and the bats are.
The whole display – pixel panel, 1024 pixels. In fact, this is 32 columns of pixels, counted from 0 to 31. Each column has a 32-bit input pin. If Nth bit is 1, Nth pixel turns on.

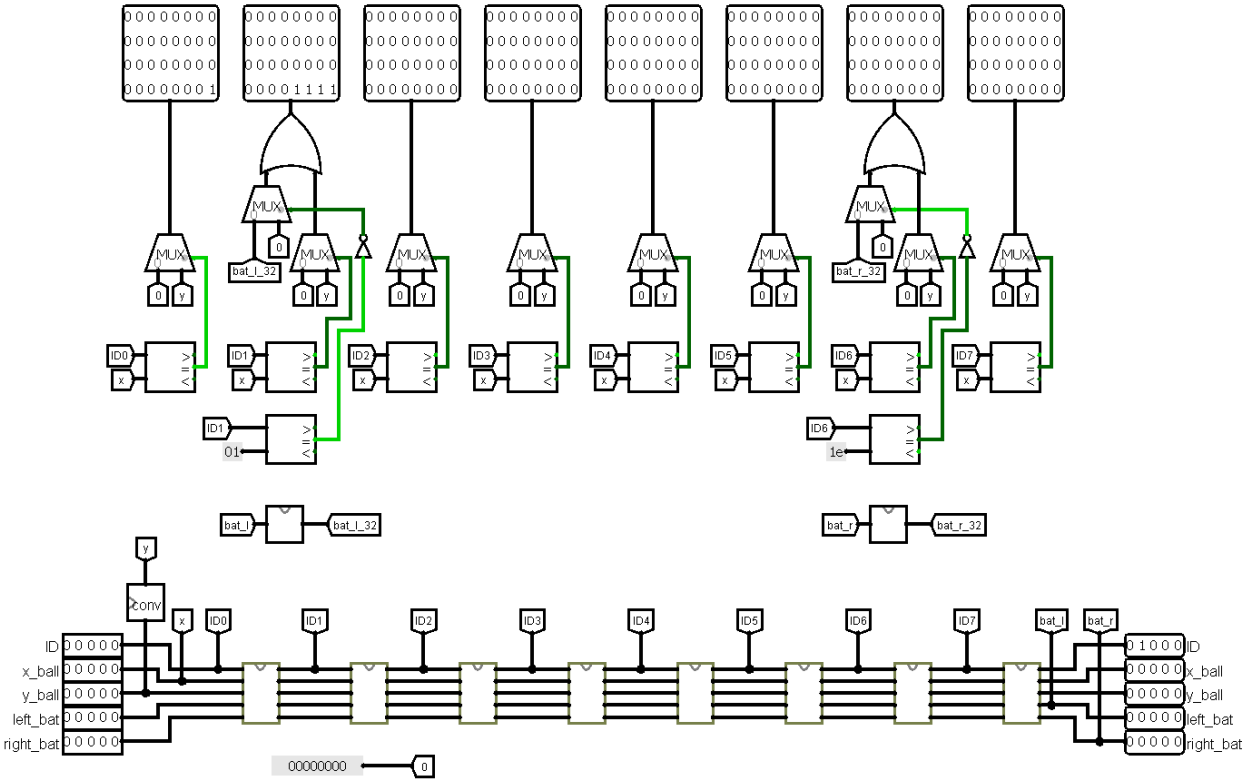
Now we can represent the ball by just turning on any pixel on the display, because the ball is just a single pixel. The player bat – three pixels in 1st column, player is able to move it up and down. The bot bat – three pixels in 30th column, same rules of movement, just like to player.
But it's not enough to display the ball, we need to make it move. For this we need to know X and Y coordinates of the ball right now and the speed of the ball (vx and vy). This information represented by 5-bit numbers. More about it will be in "*kinematic controller*" part.

Video_section

Inputs:
    ID, x_ball, y_ball,
    left_bat, right_bat (5 bit)
Outputs:
    ID, x_ball, y_ball,

left_bat, right_bat (5 bit),
8 columns outputs (32 bit)

There are 4 circuits like this under display, each one has 5 5-bit input pins, 5 5-bit and 8 32-bit output pins, connected to columns. 5-bit inputs and outputs carry information about column number, X and Y coordinates of the ball and coordinates of the bats. If X coordinate of the ball and column number are equal, bit equal to Y coordinate of the ball goes 1 and turns on the pixel. This is how we display the ball. There is also bat-check: if column number equal to 1 or 30, we are turning on 4 pixels, position of lower one we get from input, other ones are just 3 pixels above.



Picture 2

Video_chip

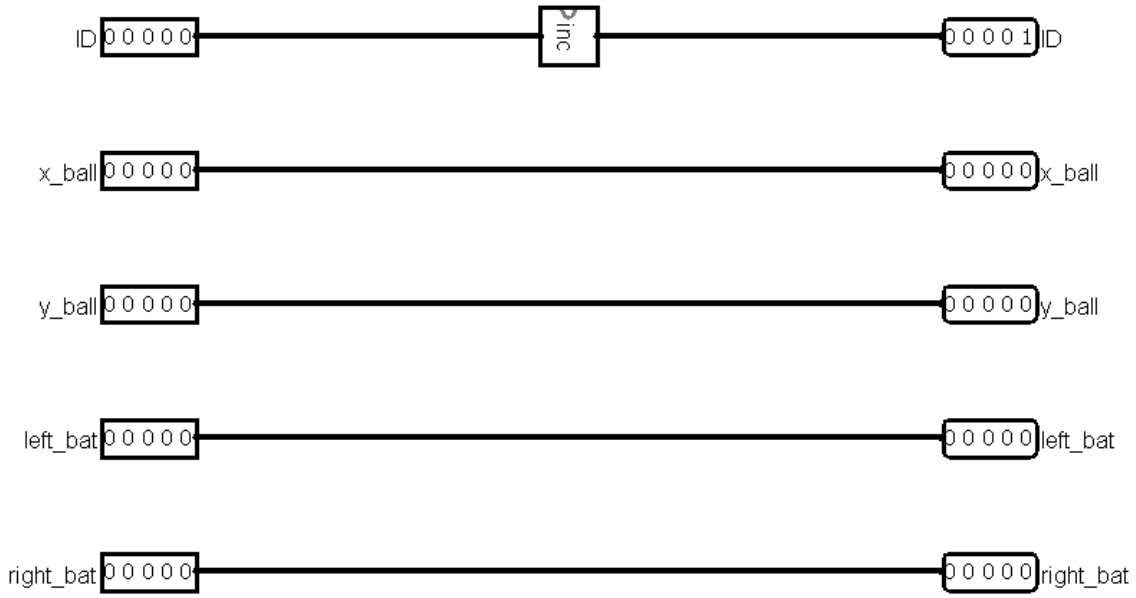
Inputs:

ID, x_ball, y_ball,
left_bat, right_bat (5 bit)

Outputs:

ID, x_ball, y_ball,
left_bat, right_bat (5 bit)

This little circuit has 5-bit inputs and outputs identical to *video section* circuit. The only task of this circuit: it increases ID (column number) by 1. By doing this, we change column number and push a new value to the next circuit. Other values remain untouchable.



Picture 3

Bat

Inputs:

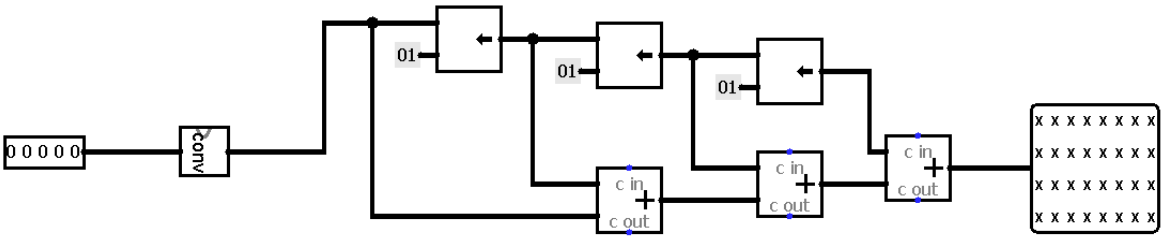
first_y (5 bit)

Outputs:

display_col (32 bit)

This circuit displays both bats.

How it works:
Both bats have one characteristic value - the coordinate of the lower pixel.
First of all, we shift '1' on n bits (n = first_y) to calculate the lower pixel.
Then, we shift this value 3 times to calculate the other three bat's pixels.
In the end, we just add all counted values to get a full bat.



Picture 4

Kinematic controller

Inputs:

reset (1 bit)

joystick (2 bit)

Outputs:

ID, ball_x, ball_y,
rightYout, leftYout , (5 bit)
player_score, cdm8_score (8 bit)

Storage:

vx, vy
x, y,
leftYreg, rightYreg (8 bit)
player_score, cdm8_score (8 bits counters)

Kinematic controller is the most important scheme in this project. It includes all sub-schemes and drives the full game.

Conditionally, we can divide *kinematic controller* on 7 parts:

1) First part is responsible for bat's move. It has "joystick" input and two schemes, in charge of the bat's moving: *bat_move* and *cdm8_bat_move*.

2) The second part is responsible for tracking ball hits. It just "check_hit" scheme. It will be described later.

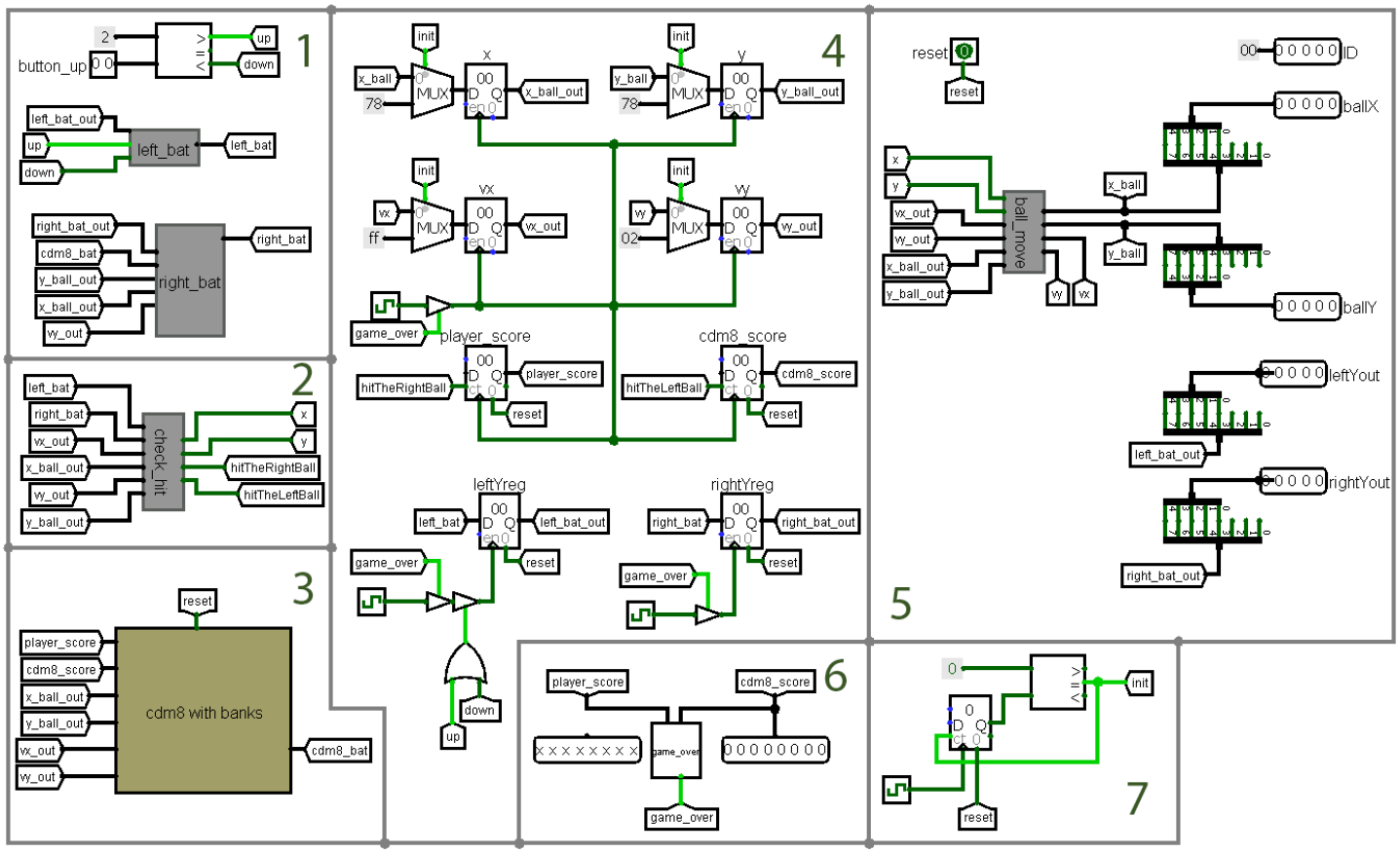
3) The third part is the brain of bot-player – *cdm8*. It counts hit point for the ball.

4) The fourth part is storage. It contains all game's values: scores, coordinates, velocity.

5) The fifth part is scheme in charge of ball moving. It will be described in more detail a bit later.

6) The sixth part is "game over" part. When someone gains the upper hand, i.e. get 24 points, this module block all values updating, that stop game.

7) The last part is initialization module. When the game starts, it loads initial values in registers.
   Initial values:
   ball_x = 120
   ball_y = 120
   vx = -1
   vy = 2
   all scores = 0
   bat's coordinates = 0



Picture 5

Cdm8 *(with memory banks)*

Inputs:

ball_x, ball_y
vx, vy
player_score,
cmd8_score (8 bits)
reset (1 bit)

Output:

cdm8_bat (8 bits)

In our project, we use cdm8 mark 5 for bot-player.

In this part, we tell you about interacting cdm8 with memory. In software part, we will talk about software implementation.

In this implementation of tv-tennis we use Harvard architecture. Scheme has two memory banks: RAM and ROM.
All read/write processes take place in RAM.

Read:
The circuit has a counter. During the counting, data is written from the kinematics controller to the memory. The counter goes from e0 to e7. It is the addresses of variables in memory. There is a multiplexer in the west of the circuit, which, depending on the state of the counter, sends the necessary data to the RAM.

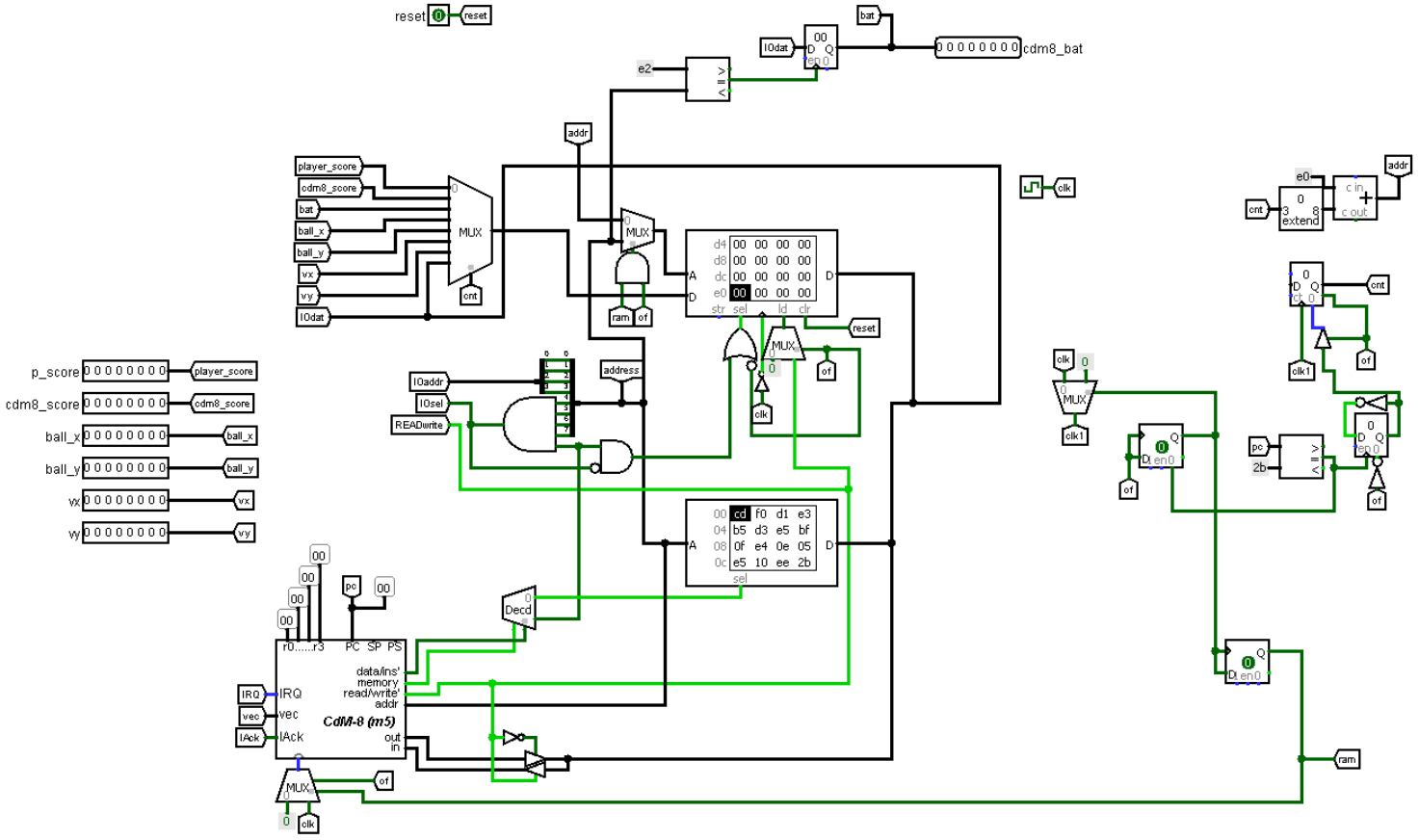
For example, "*cmd8_score*" has 0×E1 address.

Obviously, 0×E1 - 0×E0 = 1, so, counter state is 1. Then, we send 1 to multiplexer. It sends data from 1-port to RAM by 0xE1 address.

Write:
When the counter stops, the circuit turns off the RAM and turns on the ROM with compiled program. During the running, program change some variables, which located in RAM. Cdm8 load calculated point to “cdm8_bat” output pin.

There is a comparator on the west part of the scheme, that compares the current value of pc-counter with 0×2B. In this case, 0×2B is the address of a branch command, which transfers the control to the beginning of program. When pc-counter's value is equal to 0×2B, we switch memory banks and the process starts again.

The last one, “reset” is used for RAM cleaning.



Picture 6

Ball_move

Inputs:

crash_x, crash_y (1 bit),
vx, vy,
ball_x, ball_y (8 bit)

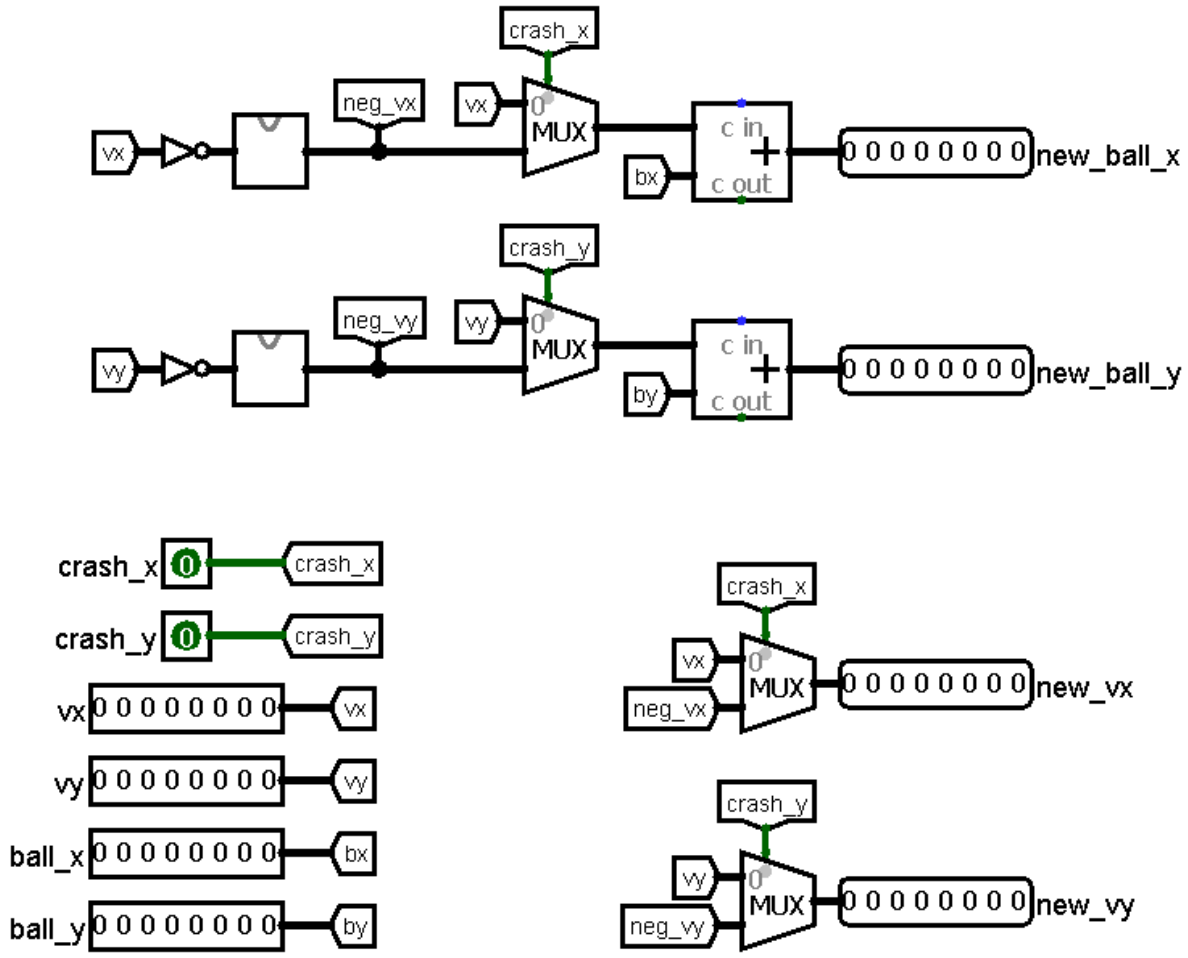
Outputs:

new_ball_x, new_ball_y,
new_vx, new_vy (8 bit)

Ball move is pretty simple. Every tick current velocity is added to the ball’s coordinate, i.g.

new_ball_y = ball_x + vx
new_ball_y = ball_y + vy.

When crash_x or crash_y is high, we take the corresponding velocity with the opposite sign.

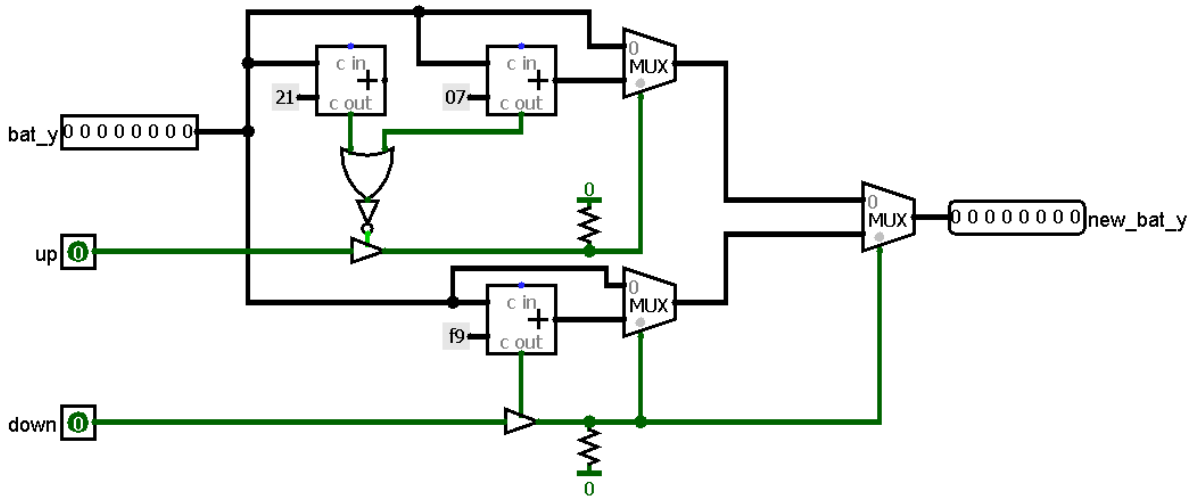


Picture 7

Bat_move

Inputs:
bat_y (8 bit),
up, down (1 bit)
Outputs:
new_bat_y (8 bit)

Bat_y pin refers to low y coordinate.
When “up” pin is high (i.e., player pulls the joystick up), we add 7 to the current bat’s y coordinate. But we should pay attention to the size of the bat. At the same time, we are adding 21 to the current bat’s coordinate to check a carry bit. If a carry bit is appearing, the bat meets the ceiling, and we should stop the bat moving.
When “down” pin is high, we do similar work, but we should check the disappearance of the carry bit, because adding positive y-coordinate to negative velocity always produces carry.



Picture 8

Cdm8_bat_move

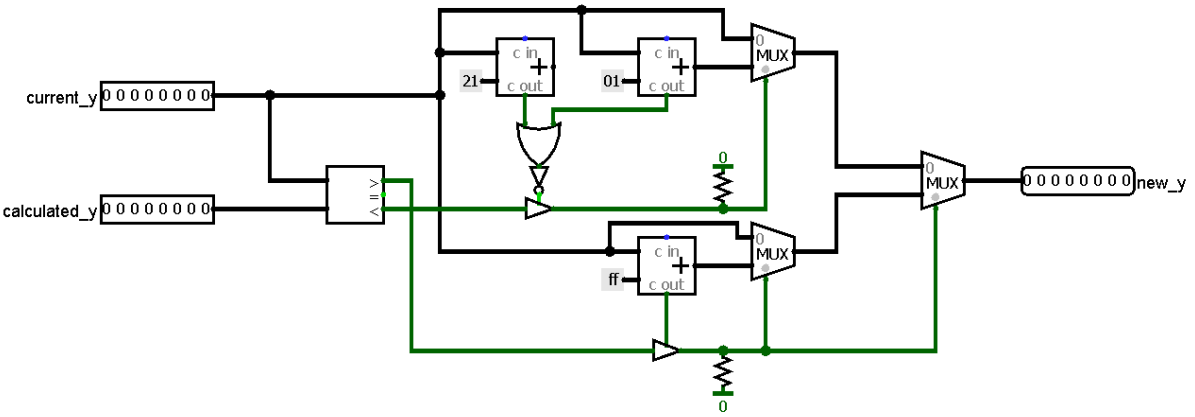
Inputs:

current_ball_x, new_ball_x (8 bit)

Outputs:

calculated_y (8 bit)

This circuit works like *player_bat*, but instead of “up” and “down” pins, it has the y-coordinate of cross point calculated by the cdm8. The principle of work is simple: we just add 1/-1 to the current bat coordinate, until it becomes equal to the calculated value.



Picture 9

Check_hit

Inputs:

left_bat, right_bat,

vx, vy,

ball_x, ball_y (5 bit)

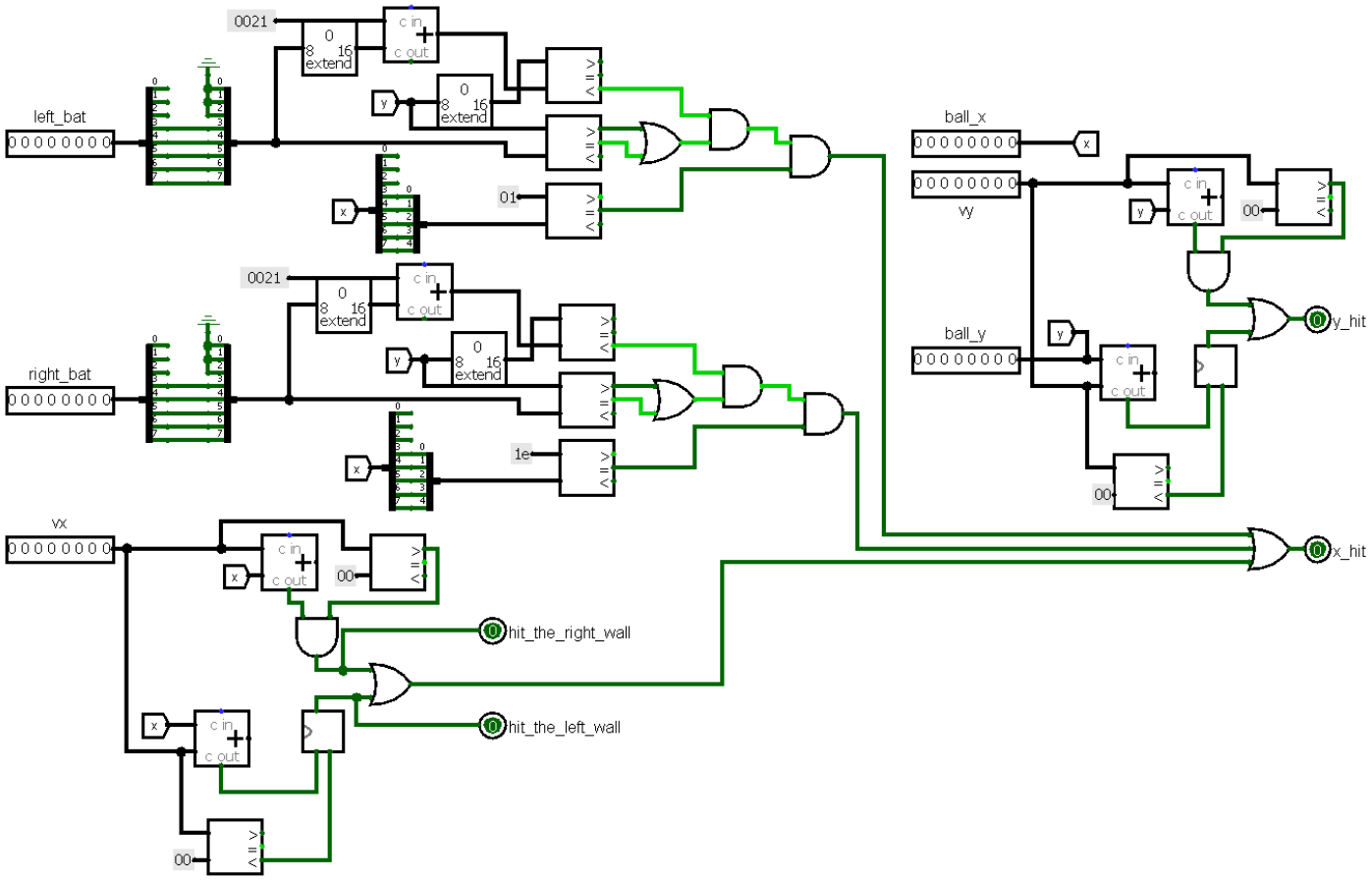
Outputs:

x_hit, y_hit, right_wall_hit,
left_wall_hit (1 bit)

How it works:

With some adders and comparators, we check the situation, is there any kind of hit right now. Actually, this circuit is some kind of predicate: is there hitting the up/down walls, left/right walls or maybe hitting the left/right bat? It may look like something difficult to understand, what is happening on this circuit, but it's not that difficult at all, to be honest there are 4 very similar algorithms, they work almost identical.

Checking the hit on Y coordinate easier because on X coordinate we should consider existence of the bats.



Picture 10

Check_carry

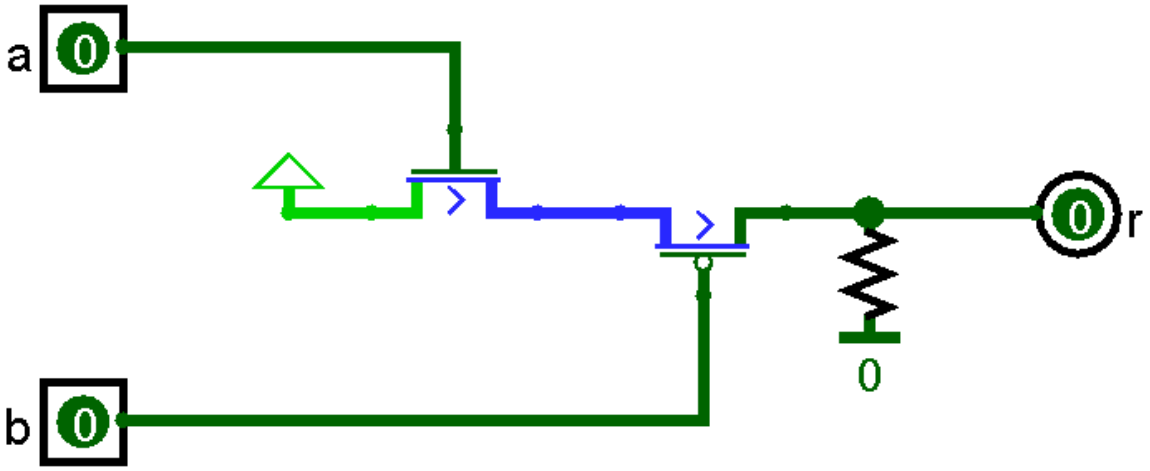
Inputs:

a, b (1 bit)

Outputs:

r (1 bit)

Very little circuit, it helps us to get the moment of hitting the wall, because hitting leads to carrying. We detect this carrying and sending 1 in output signal.



Picture 11

Game_over

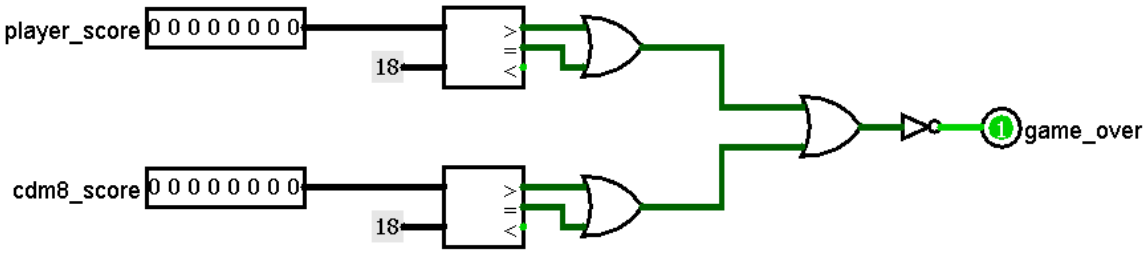
Inputs:

player_score, cdm8_score (8 bit)

Outputs:

game_over (1 bit)

This circuit stops the game when player or bot get 24 points. We always check it by compare player_score and cdm8_score to 24. If output is 0, game stops.



Picture 12

Dial

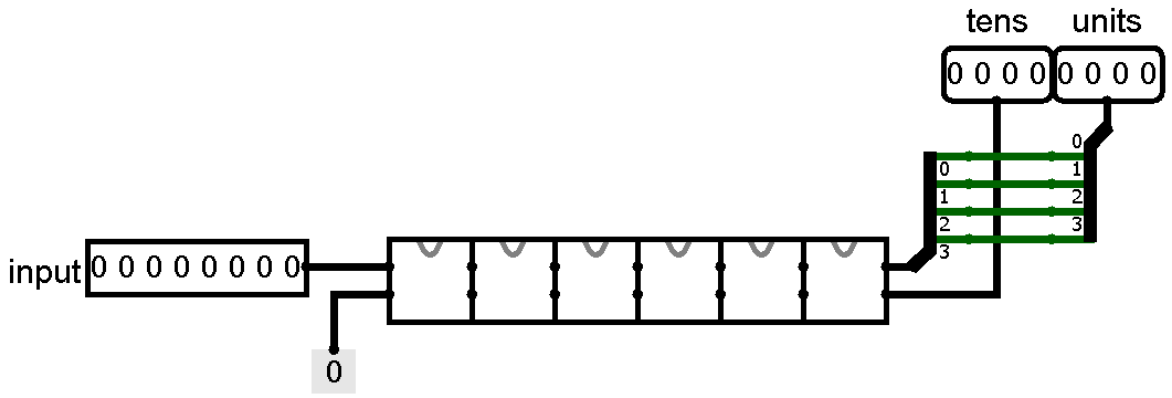
Inputs:

input (8 bit)

Outputs:

tens, digits (4 bit)

Pretty simple circuit, it consists of six *tens-units* circuits. Input value being dragged through them, and then we get tens and units on separate outputs to dials.



Picture 13

Tens_units

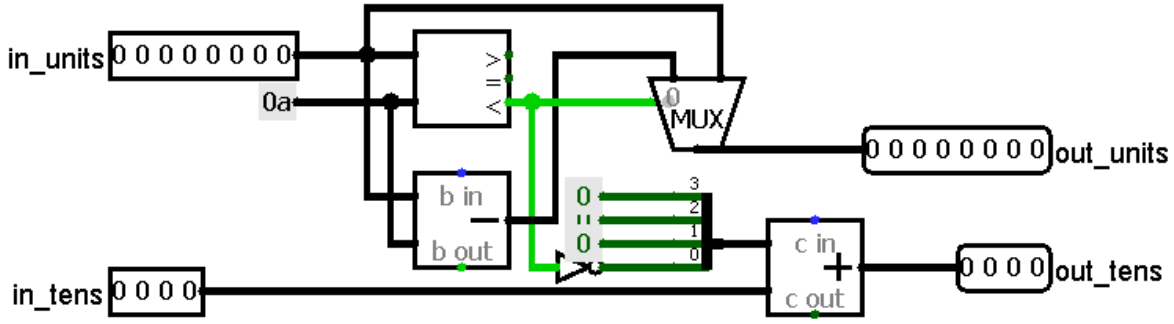
Inputs:

in_units (8 bit),
in_tens (4 bit)

Outputs:

out_units (8 bit),
out_tens (4 bit)

This circuit a bit more difficult, but yet still simple. Here we are trying to subtract ten from in_units. If it is possible, we increase in_tens by one, out_units become (in_units - 10). Else, we do not change anything, outputs equal to inputs. There are six of this circuits in *dial*, but 3 could be enough, because the game ends at 24 points. To perform all what we described above, we use comparator, subtractor, adder, and multiplexer.



Picture 14

Led

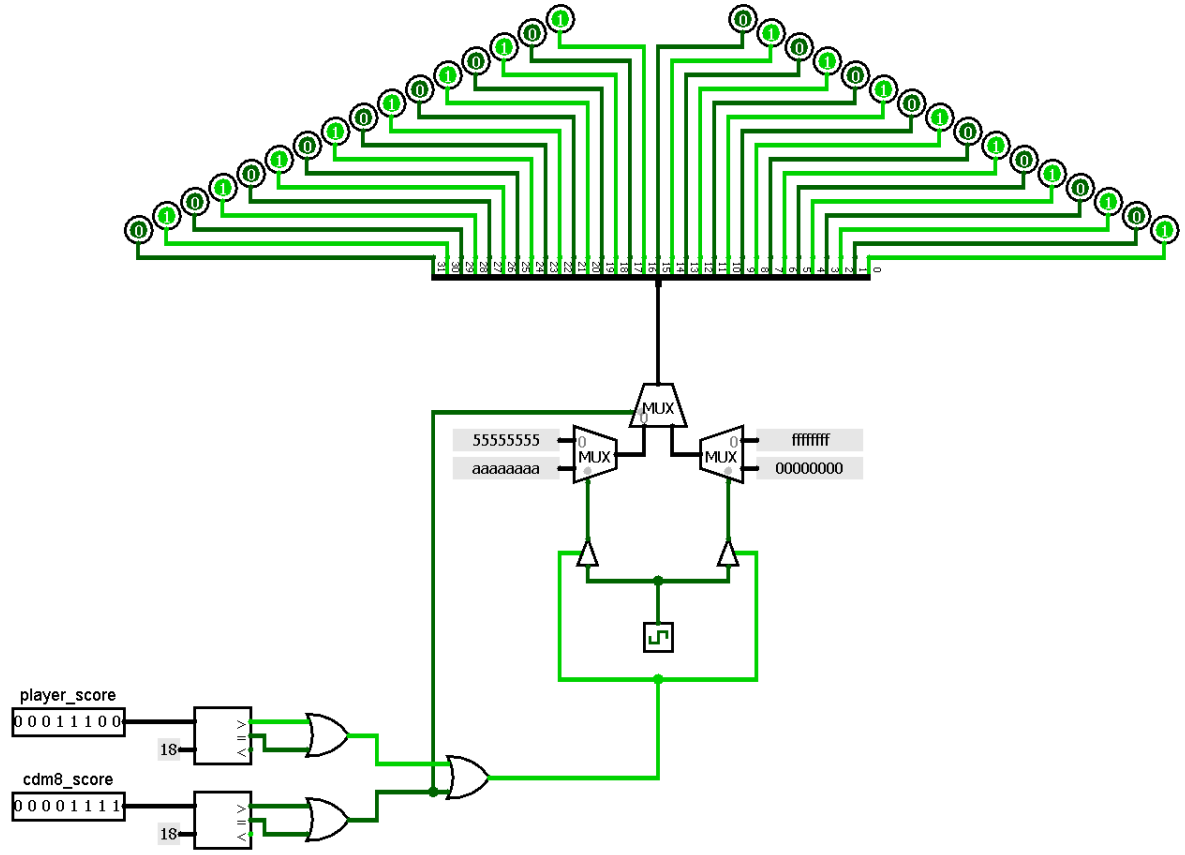
Inputs:

player_score, cdm8_score (8 bit)

Outputs:

32 output signals (1 bit)

This circuit display some kind of animation when player or bot wins, pattern of “animation” depends on who get 24 points. The player_score and cdm8_score constantly being checked and compared to 24, then with some multiplexers we create pattern of animation by changing signals on each output.
Each output connected to led.



Picture 15

Increments

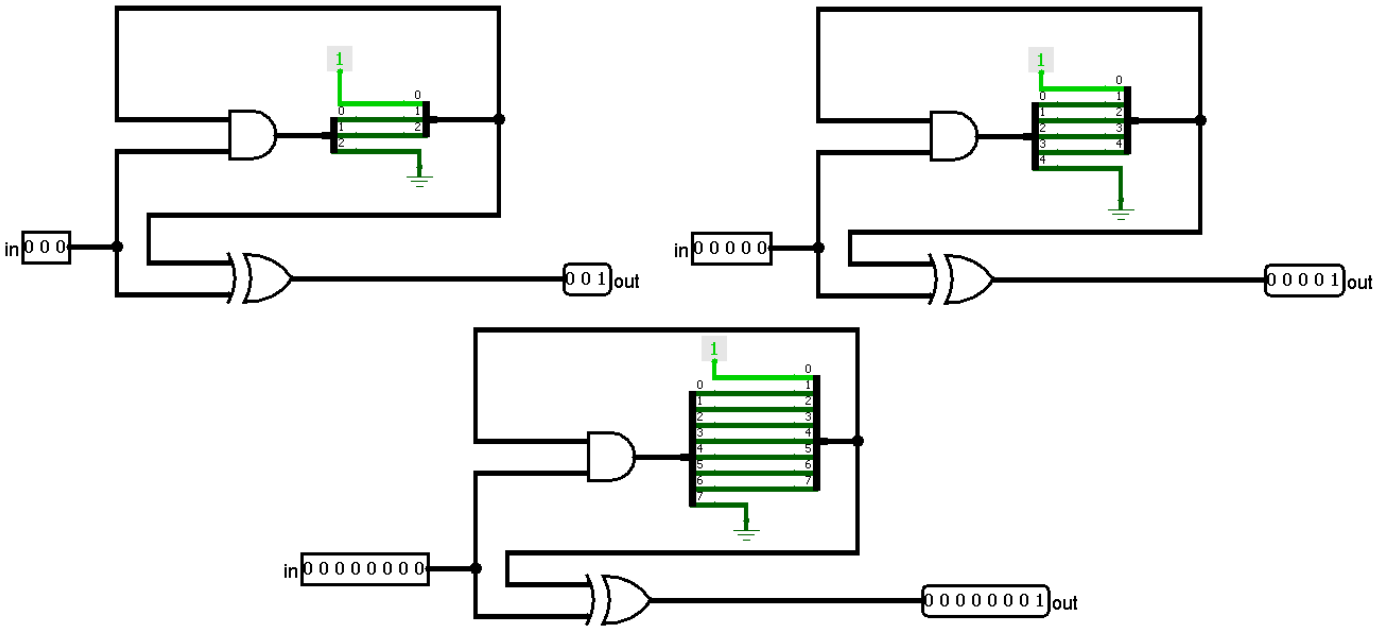
Inputs:

In (3 / 5 / 8 bit)

Outputs:

Out (3 / 5 / 8 bit)

This little circuits just increase value by one. There is three of them: inc_3_bit, inc_5_bit, inc_8_bit. They are no different in design, only number of bits of information, so we describe them in one paragraph.



Picture 16

5_to_32

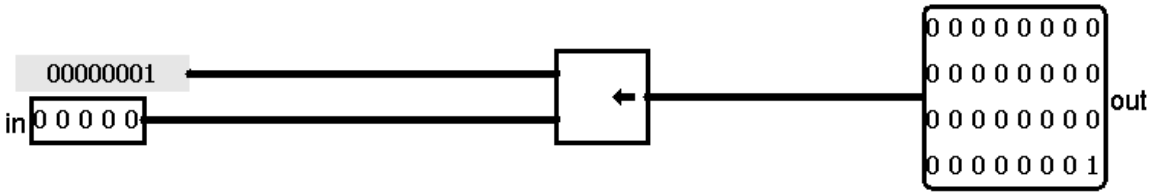
Inputs:

In (5 bit)

Outputs:

Out (32 bit)

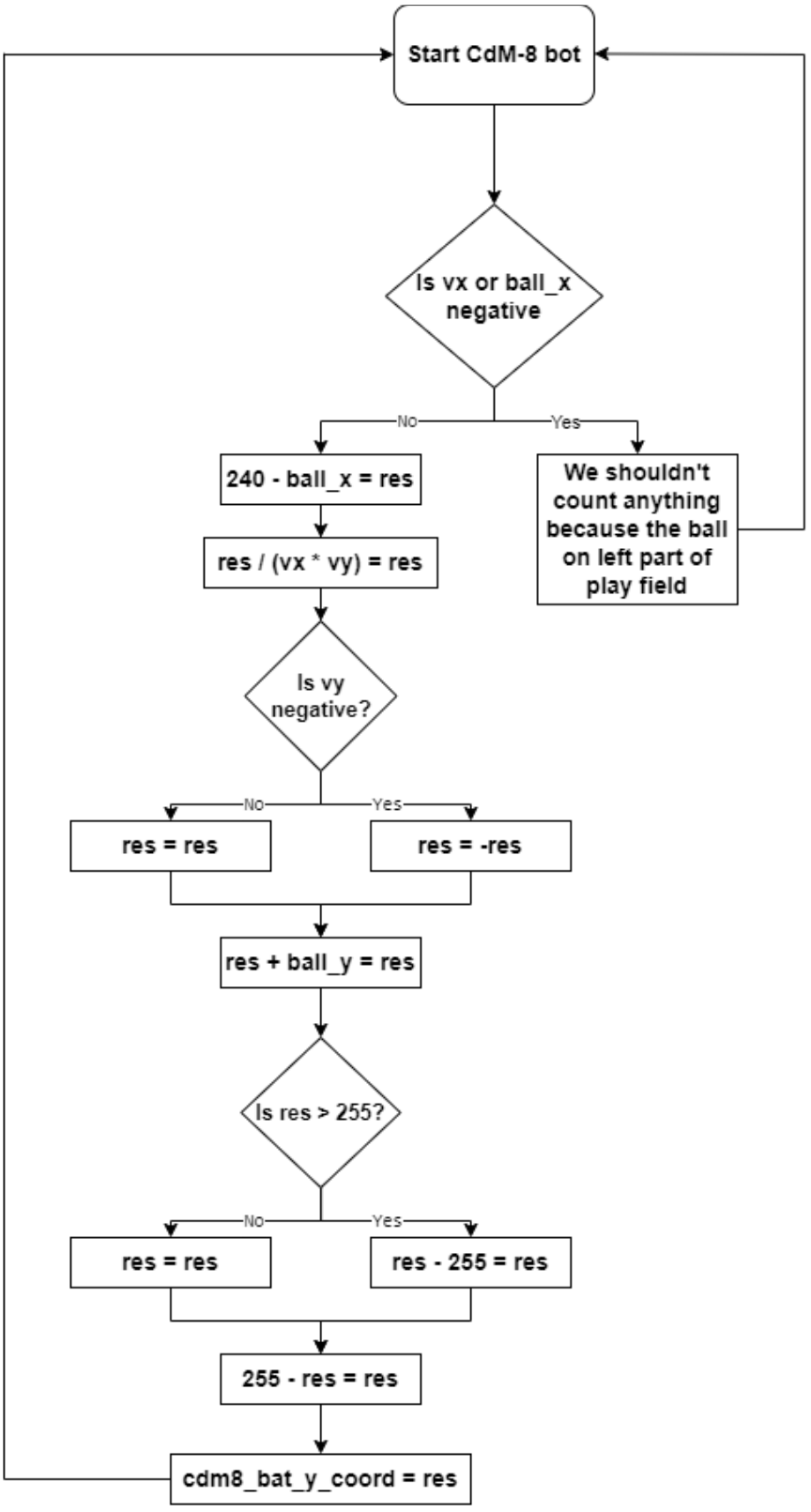
This circuit just converts 5 bit number to 32 bit number by moving bits. Pretty simple circuit.



Picture 17

Software

In this game, bot is very simple. All it should do is just calculating the hit point by the formula
Formula: 255 - (y + (240 - x) / vx * vy - 255)
The principle of work is explained in block-scheme below.



Picture 18

Conclusion

As was mentioned at the beginning, we used almost everything that we have been taught throughout whole course “Digital platform”. While working on this project we successfully implemented TV-Tennis, trained our skill in work with assembler code and creating circuits. We also found this project fun and interesting, but importantly, this project has shown us how important setting and following a plan and thoughtful separation of work between groupmates.

Attachments

1) GitHub repository:
   https://github.com/AsphodelRem/tv-tennis.git

2) Code:

```
###############################################
# 0xE0, 0xE1 - game score
# 0xE2 - coordinates of cdm8's bat
# 0xE3, 0xE4 - ball's coordinates
# 0xE5, 0xE6 - ball's velocity by x and y
###############################################

#load some value into register from an address (ldi and ld together)

macro idv/2
   ldi $1, $2
   ld $1, $1
mend

#data
asect 0xE0
   player_score:        ds 1
   cdm8_score:          ds 1
   cdm8_bat_y_coord:    ds 1
   ball_x:              ds 1
   ball_y:              ds 1
   vx:                          ds 1
   vy:                          ds 1

asect 0x00

#set a pointer on stack
setsp 0xf0

#start bot
cdm8_player:
#**********************************************
#formula for counting
#d = y + (240 - x) / vx * vy - 255
#**********************************************
```

```
#load values
idv r1, ball_x
idv r3, vx

#we shouldn't count anything, if the ball on the left side
if
	tst r3
is mi, or
	tst r1
is mi
then
	br 0x2b
fi

#(240 - x)
ldi r0, 240
sub r0, r1

#reset all flags
tst r1

#division
shra r1

idv r3, vy

#if vy was negative, take the inverse number
if
	tst r3
is mi
	neg r1
fi

idv r2, ball_y

#if there is carry bit, sustract 255 from our value
if
	add r2, r1
is cs
	ldi r2, 255
	sub r1, r2
fi

#count final cross point
ldi r3, 255
sub r3, r2

#save value to the memory
ldi r0, cdm8_bat_y_coord
st r0, r2

#start again
br cdm8_player

rts
```

```
halt
end
```