# Novosibirsk State University Informational Technologies Faculty Computer Science and System Design

TV-Tennis

**Work Group 21215**

**Autors:**

Blaginin Artyom

Valikov Nikolai

Hlebnikov Vadim

Оглавление

[General Overview 3](#_Toc103081972)

[Part 1: Hardware: Playing Field and Videochip 4](#_Toc103081973)

[Part 2: Hardware: Kinematic Controller 5](#_Toc103081974)

[Reflect module 7](#_Toc103081975)

[Coorditanates of the ball 9](#_Toc103081976)

[Has ball hit the bat? 9](#_Toc103081977)

[Right Bat Control 10](#_Toc103081978)

[Part 3: Software: CDM-8 and Control of Right Bat 10](#_Toc103081979)

[Logisim connection 10](#_Toc103081980)

[Algorithm and Pseudocode 11](#_Toc103081981)

[Further developments of TV-Tennis 12](#_Toc103081982)

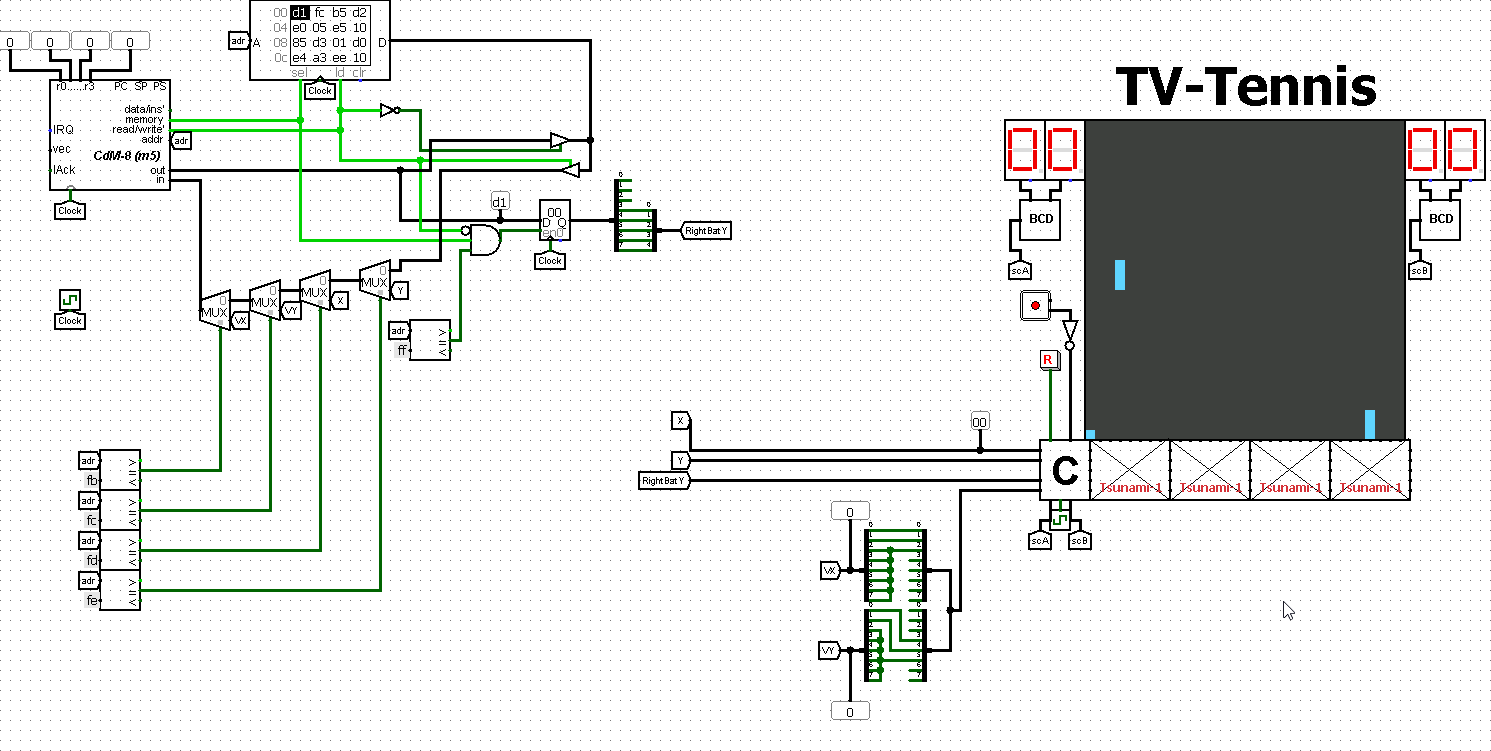
# General Overview

Добро пожаловать в руководство по игре "ТВ-Теннис", которую сделала наша группа! Данная игра берёт своё начало в 1972 году. Именно тогда компанией Atari была выпущена одна из первых коммерчески успешных видеоигр - Pong. Представляем вашему вниманию новое поколение данной игры.

Не секрет, что игр про теннис огромное множество, и с помощью игр про теннис можно проследить за эволюцией компьютерных игр – от большого игрового автомата с примитивной графикой до сложнейших 3D игр с трассировкой лучей и прочими новомодными технологиями.

Однако многие задаются вопросом, а как же игры выглядят на аппаратном уровне? Данным проектом мы хотим показать, как же всё-таки компьютер показывает картинку игры на экране, и почему это всё работает! Конечно, наша игра далека от реального положения дел, но она очень хорошо объясняет многие принципы схемотехники.

Наша группа потратила больше двух месяцев на создание логической схемы в Logisim, а также на написание искусственного интеллекта на процессоре CDM-8. В этой документации мы бы хотели представить все наши наработки, а также идеи, которые могли бы быть реализованы в будущем.



Главный вид схемы: слева процессор CDM-8, управляющий правой ракеткой, справа – игровое поле

Мы видим справа игровое поле, счёт, а также кинематический контроллер, управляющий мячом и ракетками, и 4 видео чипа, рисующих игровое поле. Процессор пытается предсказать полёт мяча и, таким образом, он перемещает правую ракетку на то место, куда, возможно, полетит мяч.

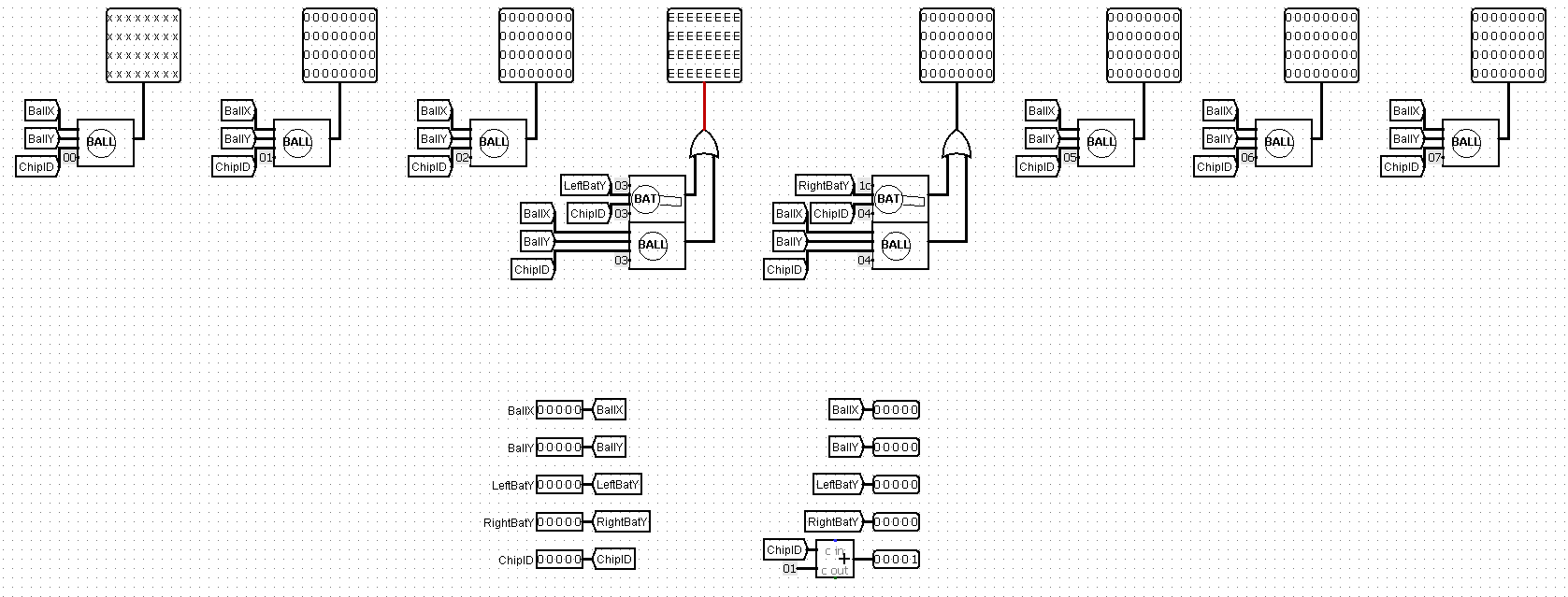
# Part 1: Hardware: Playing Field and Videochip

Игровое поле состоит из нескольких деталей: экрана 32х32, на котором отрисовываются только две ракетки и мяч, двух табло, на котором высвечивается текущий счёт игры, джойстика, кнопки сброса игры.

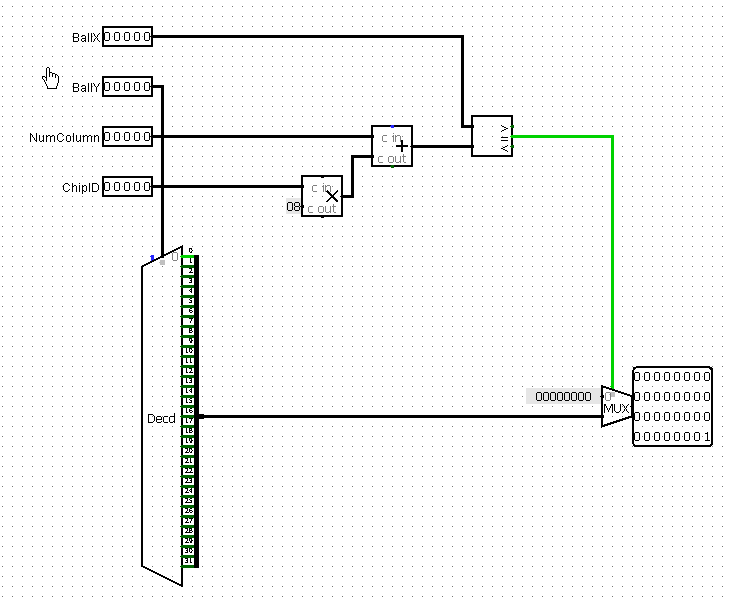
Джойстиком игрок управлят левой ракеткой. Деталь BCD переводит число из шестнадцатиричной системы счисления в десятичную, из-за чего мы может видит счёт в привычном для нас понимании.

Одной из важных компонент игры является видеочип Tsunami-1, отрисовывающий всю игру. Он состоит из 4 одинаковых ядер, каждый из которых отрисовывает картинку в своих столбцах (0-7, 8-15, 16-23 и 24-31). У каждого такого ядра есть свой **ChipID**, причём номер первого является нулём, а остальные чипы получают инкрементированный номер из предыдущего чипа.

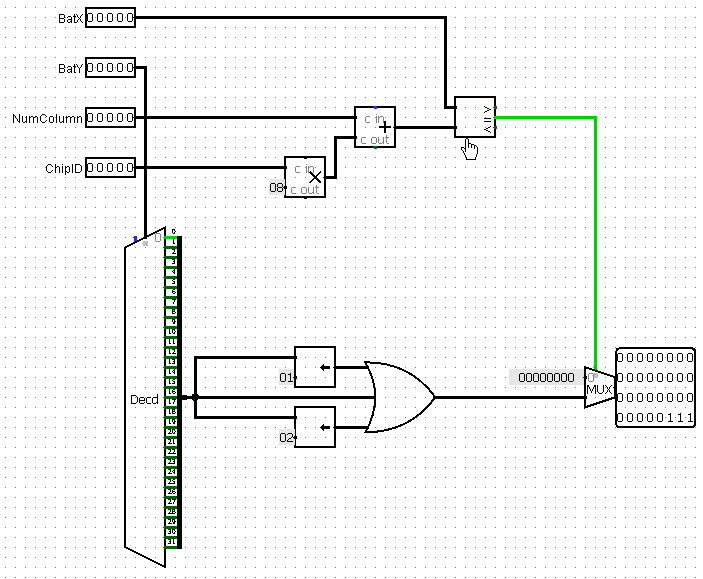
В само ядро подаётся 5 пятибитных параметров: положение мяча по **X** и **Y**, положение **Y** для первой и для второй ракетки, номер чипа. Положения левой и правой ракетки по **X** – 3 и 28 соответственно.

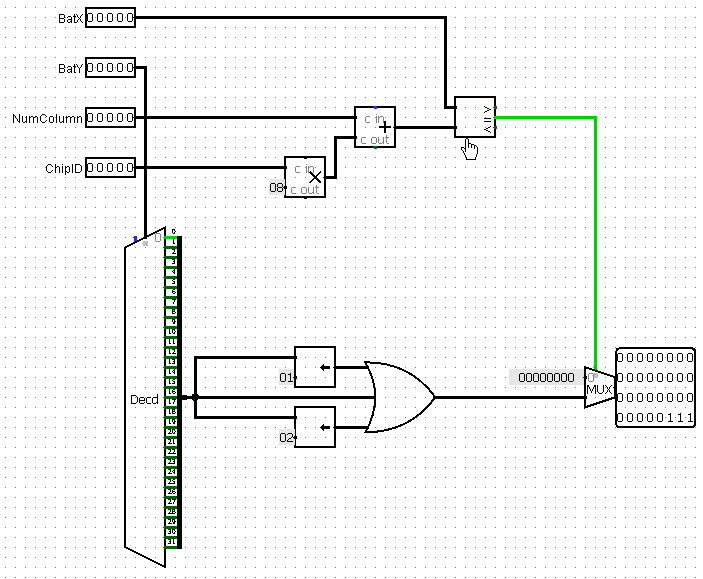


Рассмотрим подробнее детали BALL и BAT.

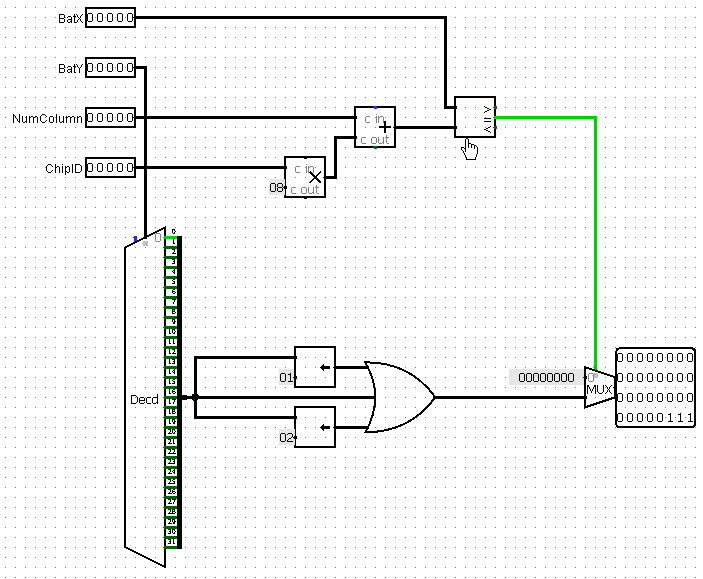
BALL отрисовывает мяч на экране. Для этого он Вычисляет текущий столбец, в котором нужно отрисовать мяч, а затем сравнивает номер столбца с реальным положением мяча. Если находится совпадение, то n-ый BALL отрисует мяч в n-столбце и в строке с номером **BallY** (нумерация идёт от 0 снизу вверх).

Если же столбец, в котором находится мяч, отличается от текущего столбца, в котором рисует BALL, то чип просто ничего не рисует.

Chip ‘BALL’’



Аналогичным образом работает деталь BAT. Она отрисовывает две ракетки в одних и тех же соответствующих столбцах, а также добавляет ещё 2 пикселя сверху к координате BatY.

Chip ‘BAT’’

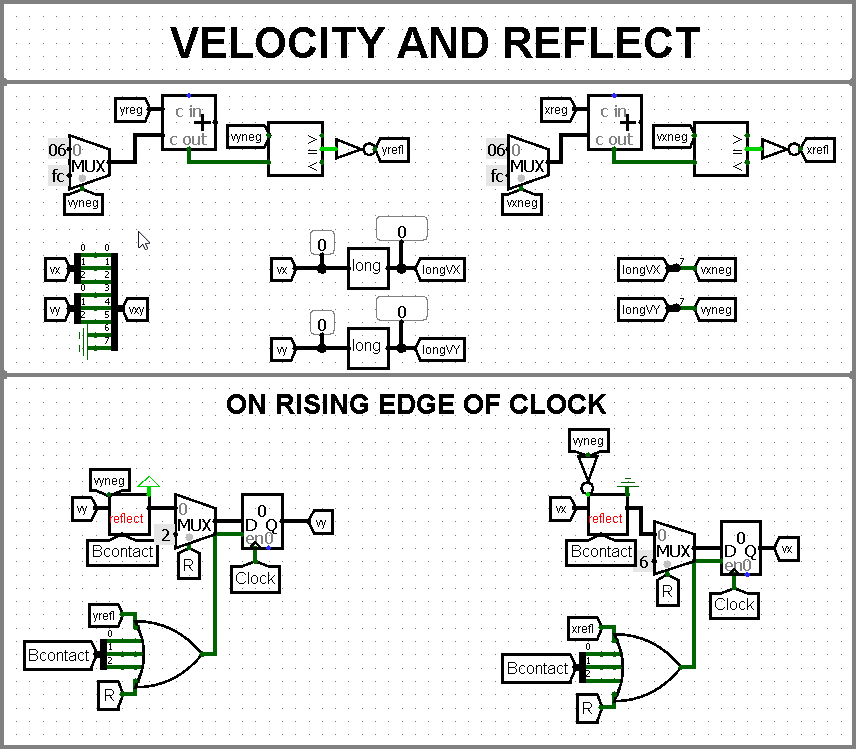
# Part 2: Hardware: Kinematic Controller

Итак, главная часть управления всей игрой, а именно координатами мяча, его скоростью и ракетками. Помимо этого, контроллер отслеживает, когда мяч отразится от стены и от ракетки, а также ведёт счёт игры. Давайте рассмотрим подробнее этот контроллер и его составные части.

Здесь находится 4 входа: clock, тактовая частота, подаваемая контроллеру, Reset, LeftYIn, координата левой ракетки, управляемой с джойстика, и RightYIn, координата правой ракетки, управляемой искусственным интеллектом.

Выходов у контроллера – десять. Пять из них (BallX, BallY, LeftBatY, RightBatY, ChipID) подаются в видеочип для дальнейшей отрисовки и имеют разрядность 5. Ещё три слева (BallX\_8, BallY\_8, VXY) имеют восьмибитное представление, они нужны процессору, чтобы он мог предсказать дальнейший полёт мяча, однако об этом будет позже. На последние два выхода подаётся счёт первого и второго игрока (scA and scB).

*Ball’s velocity*

In order for the speed of the ball to appear, let's start small. When you press the Reset button (tunnel with the name R) on the rising edge of the clock, the speed is reset to the default values, namely **vx** = -2, **vy** = 2 (five-bit tunnels).

Let's add a few more tunnels:

* **vxy**(8 bit) = **vx**(0-2) : **vy**(3-5) : 00;
* **longVX**(8 bit) = **vx**-arithmetically;
* **longVY**(8 bit) = **vy**-arithmetically[[1]](#footnote-1);
* **vxneg** = **longVX**(7th bit);
* **vyneg** = **longVY**(7th bit);

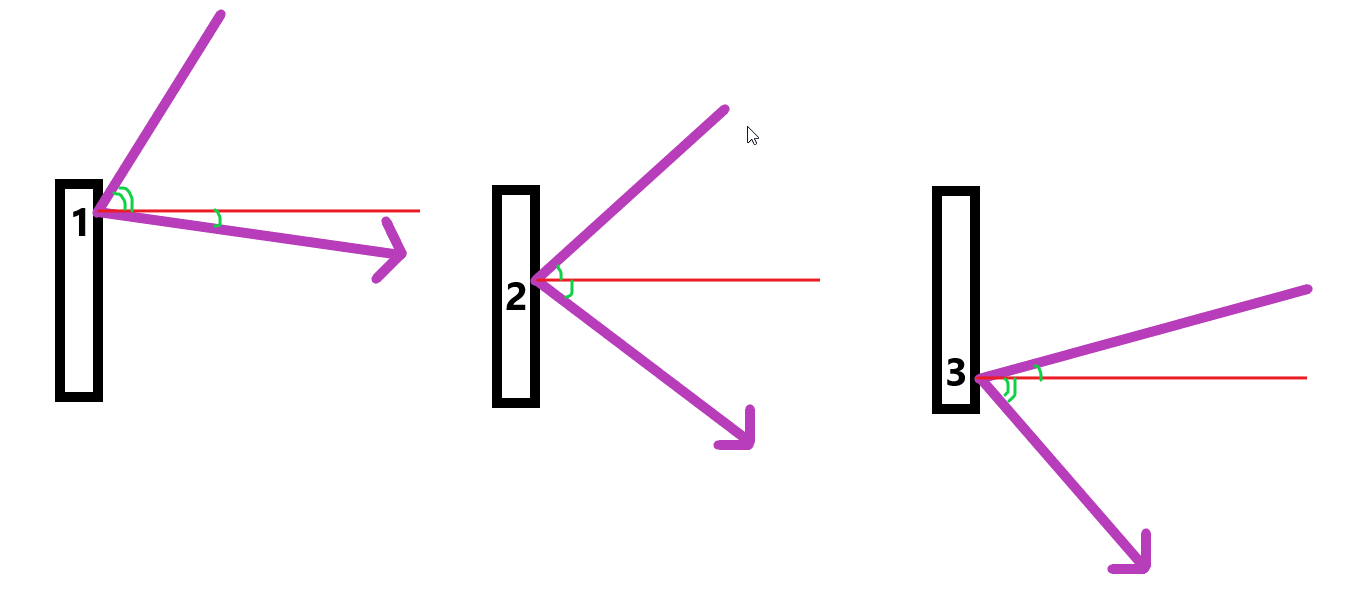
Значение vxy, longVX и longVY подаём на выход. Теперь поймём, в какой момент мяч сталкивается со стеной. Понятно, что если при добавлении к координате значение скорости мы получаем беззнаковое переполнение, то мяч примкнул к стене, и его скорость должна смениться на противоположную. Таким образом, получаем значения тоннелей xrefl и yrefl.

Изменение скорости происходит таким образом:

* если мяч отскакивает от вертикальной стенки, скорость по X меняет знак, причём vy не меняется.
* Аналогично, если шарик отскакивает от горизонтальной стенки, vy меняет знак.

Самое интересное происходит, когда мяч отскакивает от ракетки. Было бы, наверно, скучно, если бы мяч имел постоянную по модулю скорости, а изначально скорости по модулю равны двум.

Чтобы мяч летел под различными углами, мы ввели такую физику мяча.



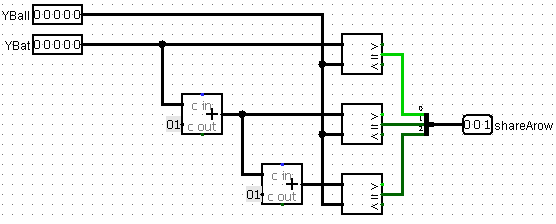
Мяч может прилететь к ракетке либо снизу от неё, либо сверху. Рассмотрим изменение скорости на примере рисунка, когда мяч прилетает сверху. Имеем, что ракетка состоит из трёх пикселей, пронумеруем их от 1 до 3 сверху вниз.

|  |  |  |
| --- | --- | --- |
| Sector | VX | VY |
| 1 | |VX| + 1 | |VY| – 1 |
| 2 | |VX| | |VY| |
| 3 | |VX| – 1 | |VY| + 1 |

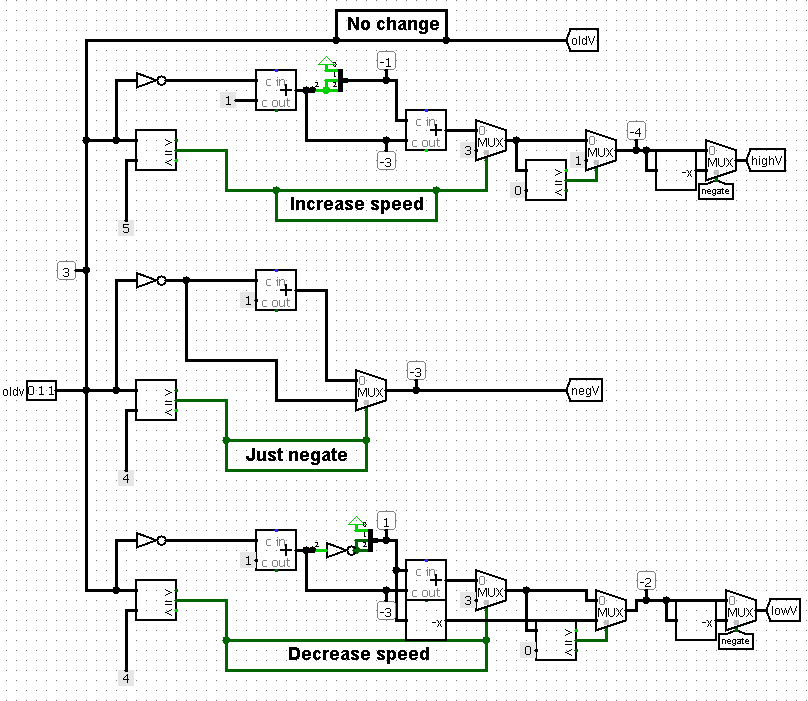
* *При попадании в сектор 1*, vx по модулю увеличивается, а vy уменьшается.
* *При попадании в сектор 2*, vx и vy не меняются.
* *При попадании в сектор 3*, vy по модулю увеличивается, а vx уменьшается.

Это работает как с левой, так и с правой ракеткой. Если мяч летит снизу, то ситуация аналогично предыдущей, но нумерация от 1 до 3 происходит снизу вверх.

## Reflect module

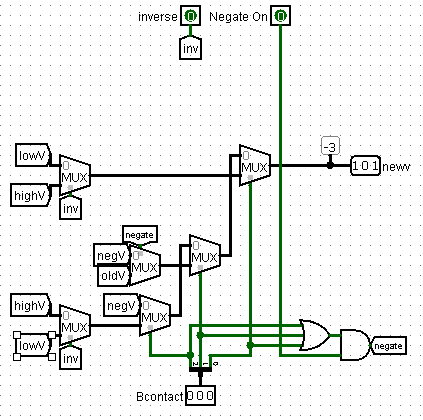
Поговорим о том, как контроллер отслеживает столкновение мяча с ракеткой. Чтобы понять, находится ли мяча на одной строке с каким-либо сектором ракетки, мы используем модуль inrange.

На вход подаётся координата мяча по Y и координата первого пикселя у ракетки. На выход подаётся трёхбитное число, каждый бит обозначает, в каком секторе ракетки находится мяч. В нашем контроллере такое значение хранит тоннель Bcontact, о нём чуть подробнее расскажем дальше.

 Теперь рассмотрим довольно громоздкий модуль reflect.

Первая часть модуля возвращает значения скоростей, которые будут после отскока от ракетки. На вход подаётся изначальная скорость oldV, а в тоннели подаются такие скорости:

* oldV = oldv;
* |highV| = |oldv| + 1;
* negV = -sign(oldv)\*|oldv|;
* |lowV| = |oldv| - 1;

Важно отметить, что скорость меняется только в промежутке от -4 до 3 и за исключением нуля! Поэтому в схеме важно было рассмотреть частные случаи (нельзя уменьшать по модулю 1 и нельзя увеличить по модулю -4).

Осталось понять, какую скорость необходимо подать на выход. Поскольку скорость по X меняет свой знак, а скорость по Y не меняет её, пустим на вход бит Negate On, обозначающий, противоположная ли по знаку скорость, или нет.

Как мы поняли ранее, нужно различать, что мы делаем со скоростью, когда мяч налетает на ракетку сверху или снизу. Если бит inverse равен нулю, то мяч налетает сверху – все значения по умолчанию, иначе увеличение скорости и уменьшение меняются местами (смотри таблицу выше).

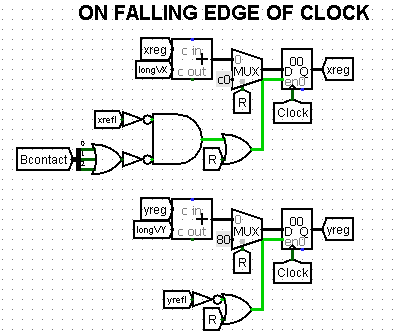
Данный модуль удовлетворяет физике, которую мы хотели получить выше. Этот модулю так же обрабатывает случай, если мяч оттолкнулся от стены – в этом случае Bcontact просто равен нулю.

Таким образом, получаем принцип изменения скорости:

1. Кнопкой Reset задаём начальную скорость;
2. При столкновении с вертикальной стеной и ракеткой скорость по модулю могу меняться, причём знак vx меняется, а у vy – нет;
3. При столкновении с горизонтальной стеной знак vy меняется, а у vx – нет.
4. Изменение скорости происходит при Clock = 1, и если было столкновение со стеной, ракеткой, или была нажата кнопка Reset.

## Coorditanates of the ball

Когда мы имеем скорость мяча, теперь можно говорить о его координатах на игровом поле. Важно заметить, что изменение координаты возможно только на нисходящем фронте часов. Это сделано для того, чтобы сначала установить правильные скорости мяча перед изменением координат.

Итак, у нас есть скорость мяча, на каждый нисходящий фронт часов мы добавляем к координатам X и Y мяча значения его скоростей longVX и longVY, обновляем значения координат в регистрах и записываем эти значения в тоннели xreg и yreg (8 bit).

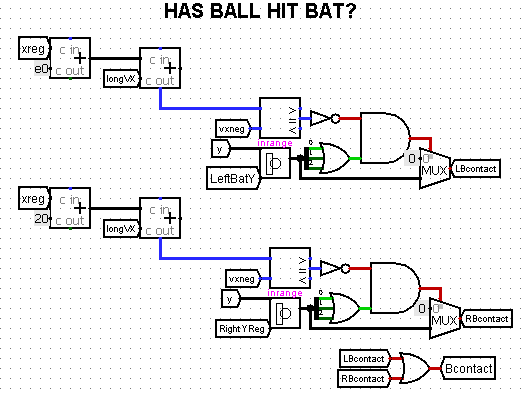
Однако координата по X не прибавляется, если мы нажали кнопку Reset, или мяч столкнулся с вертикальной стенкой или ракеткой. Координата по Y не обновляется, если так же зажата кнопка Reset, или мяч столкнулся с горизонтальной стеной.

При зажатой кнопке Reset в xreg и yreg записывается изначальное положение мяча: 0xc0 и 0x80 соответственно.

Значения xreg и yreg переводим в 5-битные строки, и получаем значения на выход BallX и BallY.

## Has ball hit the bat?

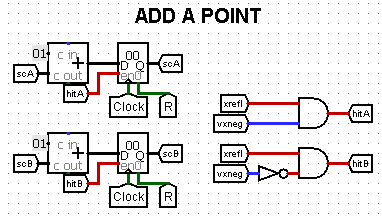
Проверку, прилетел ли мяч в ракетку, осуществляют две детали, возвращающие значения LBcontact и RBcontact – попадание в левую и правую ракетку соотвественно.

Схема довольно простая: ракетки находится на координатах 24-31 и 240-247, поэтому, если мяч находится в зоне действия ракетки (то есть срабатывает inrange) и его координата при прибавлении скорости становится меньше 32 (0x20), то мяч коснулся левой ракетки, иначе если координата становится больше 240 (0xe0), то мяч коснулся правой ракетки.

LBcontact и RBcontact хранят в себе сектор, в который попал мяч, либо содержат ноль, который означает, что мяч не задевал ракетки.

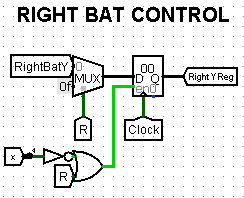
Объединяя LBcontact и RBcontact, мы получаем значение Bcontact, означающий, что мяч коснулся какой-либо ракетки, а также сектор, в который мяч попал.

Add a point

Если игрок промахивается ракеткой по мячу, то мяч летит в левый или правый край экрана, xrefl = 1. Таким образом, если мяч попал в левый край, т. е. vxneg = 0, очко засчитывается компьютеру, hitA = 1, иначе очко начисляется на счёт человека, hitB = 1.

На каждый такт восходящего фронта, если очко в чью-то пользу было засчитано, в регистр сохраняется счёт игрока, увеличенный на 1. Счета хранятся в тоннелях scA и scB, счёт человека и компьютера соответственно. Они подаются на выход контроллера.

## Right Bat Control

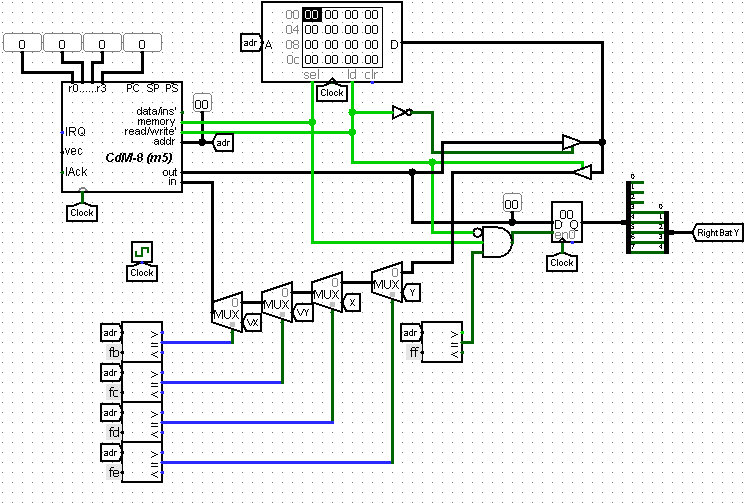
Значение RightBatY подаётся на вход процессором CDM-8. Данная деталь следит за тем, чтобы ракетка останавливалась на правой половине экрана. Так игра становится интереснее, ведь компьютер может не успеть переместить ракетку в конечное положение. Так же схема устанавливает значение по умолчанию для правой ракетки, если была нажата кнопка Reset.

# Part 3: Software: CDM-8 and Control of Right Bat

Перейдём к последнему элементу нашей видеоигры – искусственному интеллекту. Интеллект сделан с помощью процессора CDM-8, работающего с одним банком оперативной памяти на 256 байт.

## Logisim connection

Процессор получает четыре восьмибитных значения: VX, VY, BallX (X), BallY (Y)и помещает их в память по адресам 0xFB, 0xFC, 0xFD, 0xFE. CDM-8 работает с этими данными, а затем выводит в регистр Logisim положение правой ракетки, хранящееся по адресу 0xFF. Значение записывается в пятибитный тоннель RightBatY.

Полная схема соединения представлена ниже на картинке.

## Algorithm and Pseudocode

Покажем псевдокод алгоритма вычисления конечной координаты правой ракетки процессором. Алгоритм приближен к коду, который может быть написан на ассемблере CDM-8.

*Pseudocode of the algorithm*



Данный код свободно переносится на ассемблер CDM-8, памяти в 256 байт полностью хвататет.

# Further developments of TV-Tennis

Благодарим за прочтение документации к нашей игре! Мы постарались сделать игру так, чтобы она была достаточно интересной. Однако наша команда на этом не останавливается, ведь у нас есть некоторые другие идеи, которые можно было бы реализовать в будущем.

1. *Оптимизация вычислений координат.* К сожалению, процессор вычисляет конечную координату не за столь быстрое время, как нам хотелось бы. Но у нас уже есть идеи по оптимизации кода, например, быстрое деление на 3 позволит не отнимать координату, а вычислять значение вне зависимости от числа, то есть за определённое количество тактов.
2. *Кручёные удары.* Если вы играли в настольный теннис, то знаете, что, ударив по мячу движущейся ракеткой, вы заставляете мяч крутиться, из-за этого он может неожиданно для второго игрока резко улететь в сторону. Мы хотим ввести эту идею в наш TV-Tennis, для того чтобы играть было ещё интереснее.
3. *Мультимяч.* Безумный режим, в котором вам придётся играть не одним, а сразу тремя мячами! Нужно будет лишь успевать отбивать мячи, но следить за ними будет гораздо труднее.

1. Данным преобразованием в longVX и longVY занимается модуль newVX. [↑](#footnote-ref-1)