The Ministry of Education and Science of Russian Federation

Federal State Budgetary Educational Institution of Higher Professional Education

“National Research Novosibirsk State University”

The Informational Technologies Faculty

Computing Platforms Course

“The game of Noughts and Crosses”

Group project A

Project made by students of the 1st year of education:

Bukhner Mark, 20214

Bragin Mikhail, 20213

Kolomnikova Daria, 20213

February – May 2021

**Table of content**

[Overview](#_heading=h.30j0zll) **3**

[Basic system principles](#_heading=h.1fob9te) **3**

[Hardware](#_heading=h.3znysh7) **3**

[Gamepad](#_heading=h.2et92p0) 3

[Individual cells](#_heading=h.tyjcwt) 3

Software

# Short, but full Overview

Игра создана **by our own design**.

This project emulates the game of noughts and crosses, played on a 3 by 3 gamepad against an AI. The AI tries to utilize a somewhat optimal strategy. Both the processor and the circuit are emulated with Logisim, and the code is written with the CDM-8 language.

*Каждая из 9 ячеек на игровом поле имеет достаточно электроники, чтобы управлять светодиодным дисплеем 3 на 3 пикселей.*

Each of the 9 cells of the gamepad is taking control over its own 3x3 LED display.

*Состояние каждой ячейки хранится в 2 битном регистре (0\* - пустая клетка, 10 - нолик, 11 - крестик). Таких регистров 9, они заключены в game\_controller, он распознает нажатие на кнопку от игрока, он распознает ход компьютера и он же следит за состоянием игры и определяет победу, поражение или ничью.*

Cell’s status is stored in a 2-bit register, where 0\_ encodes an empty cell, 10 - nought, 11 - cross. There are 9 registers for storing cell status information, which are located in the *game\_controller*. The *game\_controller* itself recognises either moves of the computer and the player (identifying button-pushes), tracks the game status and determines the end of the game (win, lose or draw situations).

*В любой момент времени состояние игры собрано в 3 байтную шину (game\_bus), информацию откуда может читать CdM-8, используя соответствующий адрес I/O.*

*В ПЗУ компьютера загружается прошивка (алгоритм игры), у которой строго определен интерфейс, соответственно на плате есть несколько ПЗУ, между которыми можно переключаться (легкий и сложный уровень игры). Благодаря такому удобному интерфейсу, можно программировать игру как захочется, мы разработали 2 прошивки (уровни игры).*

# Basic system principles

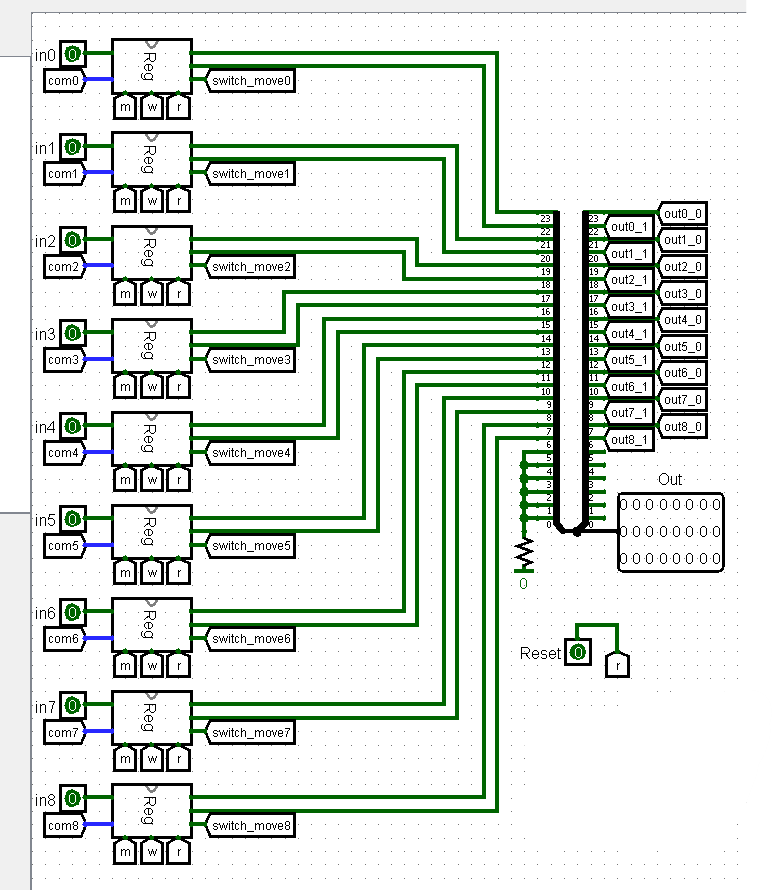
The following project implements tic-tac-toe gameplay with the common rules:

1. First move is made by the player, who plays with “crosses”.
2. A simple AI makes a move after the player, considering the fact that it can’t place a nought in the occupied cell.
3. Then the player and the AI are making their moves one by one, until a win, lose or draw situation arises. System does not allow the player to make a move in an inappropriate cell.

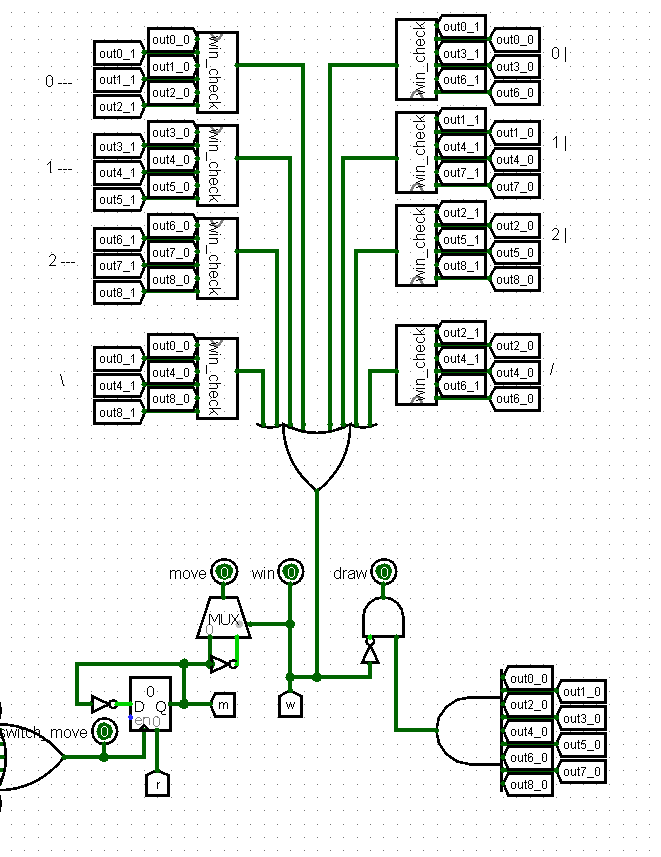
# Hardware

## Game controller

Game controller состоит из 2 частей: он хранит состояние игры, состояние каждой клетки хранится в 2 битном регистре, который обвязан by gates, которые получают состояние игры (чей ход, закончилась ли игра и reset), а также 2 инпута (1 от человека, 1 от компьютера), и изменяют состояние клетки, если это возможно



при изменении состояния подают на выход сигнал switch\_move, который делает reset процессора, и он заново вычисляет свой ход. Также все выходы этих регистров связаны в game\_bus, доступ к которой имеют I/O устройства.

Вторая часть game\_controller - устройство распознавания конца игры 

Набор gates определяет состояниы игры, эти значения имеют выход во внешний мир, их видит scoreboard, а также регистры, описанные выше.

## Gamepad

Геймпад представляет собой 9 экранов и 9 кнопок, сигналы с которых уходят в game\_controller и там обрабатываются, за поведение каждого из экранов отвечает display\_controller, каждый такой чип получает 2битное значение состояния клетки из game\_controller.

## 

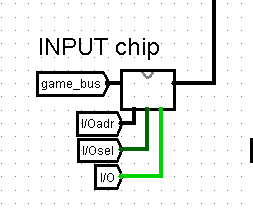
## Scoreboard

Так как game\_controller выводит наружу состояние победы или ничьи, а также чей сейчас ход, то к нему подключена бегущая строка 32x6, внутри которой закодированы слова “YOU WIN”, “YOU LOSE” и “DRAW”. Внутри находится 32 регистра, которые обеспечивают движение строки

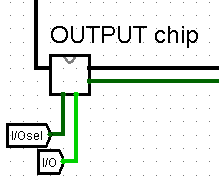
## I/O chips (firmware interface)

I/O доступен из кода прошивки в строке 0xE, для получения состояния поля необходимо обращать к первым 9 байтам (0xE0 - 0xE8), считывая значения оттуда, INPUT chip будет доставать из game\_bus нужное значение.

Каждый такой чип устроен просто. Например, INPUT chip вычленяет из game\_bus нужные 2 бита, и выводит их как 8 битное число с нулями в конце.



Когда компьютер принимает решение о том, в какую клетку необходимо поставить нолик, то нужно вывести индекс клетки (от 0 до 8) в любую из ячеек 0xE. OUTPUT chip распознает вывод, и отправит решение компьютера в game\_controller.



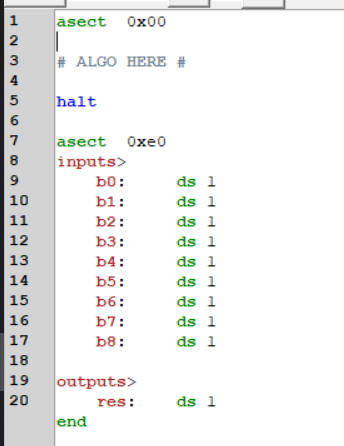
В остальном схема крайне простая, остальные неописанные чипы являются разветвителями game\_bus шины.

# Software

Схема спроектирована так, что прошивку компьютера можно менять как угодно, и даже во время игры переключаться между разными. А создать собственную не трудно из-за очень просто I/O интерфейса.

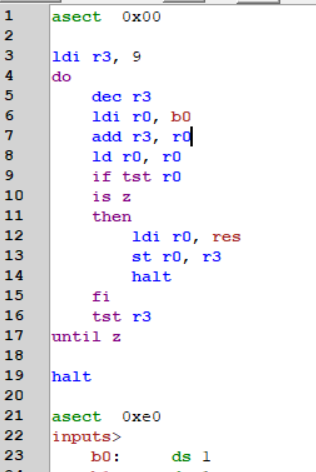
Задача, которую решает программа - в какую клетку сделать ход.

После каждого хода игрока в CdM-8 поступает сигнал reset. И программа вычисляет очередной свой ход. Паттерн для написания кода выглядит вот так:



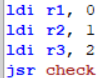
Нами создано 2 прошивки:

Первая - самая простая и банальная



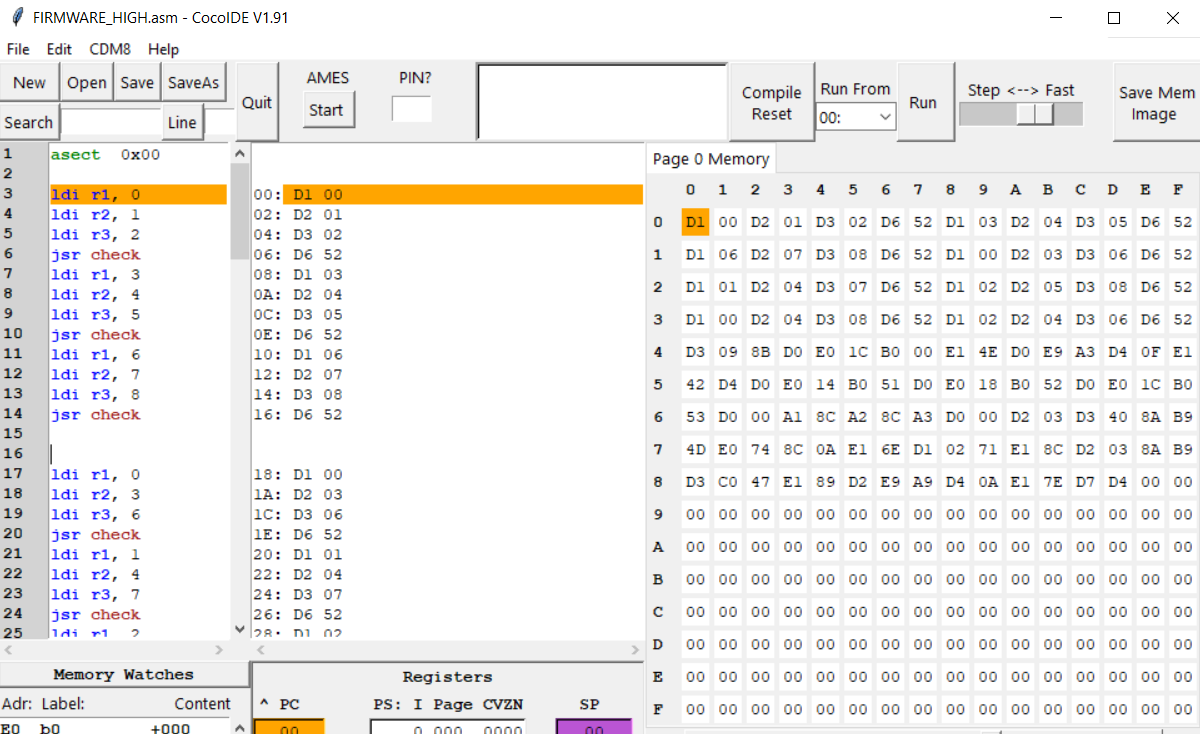
Данный код находит первую свободную ячейку с конца поля, и выводит индекс первой попавшейся свободной. Обыграть такой алгоритм невероятно легко.

Поэтому доработаем его, напишем routine “check”, которой на “вход” будет поступать 3 индекса в регистрах r1, r2, r3 (линия, которую надо проверить на предмет того, что соперник может выиграть поставив туда третий крестик следующим ходом). Эта routine загружает состояния необходимых клеток, и, если необходимо сделать ход, то выводит нужный индекс, иначе достигает команды ***rts*** и возвращается в исходную точку. Таким образом можно проверять каждую линию (которых 8) за 4 команды:



Переиграть данный алгоритм уже труднее, однако все еще возможно.

Самый интересный момент - такая программа занимает чуть больше половины доступной памяти



А значит, ее вполне реально доработать до беспроигрышного алгоритма, и при этом уместиться в 256 байт памяти!

Оптимальная частота работы схемы - 1KHz для любой программы.