**The game of "Reversi"**

Klim Sadov, Ilya Stubarev, Maxim Zaugolnikov

2025

General Information

This technical specification outlines the requirements for creating a hardware-software system designed to run the "Reversi" game on the CdM16 processor, utilizing the Logisim logic simulation environment.

"Reversi" is an abstract board game played on a 6x6 board and where both a human being and a computer take turns putting their own pieces and reversing their opponent's pieces according to rules.

The goal of the project is to make game logic for a real player in hardware, and to write a bot in C, translate it into CdM16 assembly language, and integrate it into hardware circuitry in Logisim under the Harvard architecture.

And the point that all the game logic concerning the real player's actions should be implemented in hardware is an important detail, because I think it would be too easy to do it programmatically.

**Purpose and Objectives**

*Objective*

To develop the hardware-software system implementing the Reversi game functionality running on the CDM16 processor simulated in Logisim with a possibility of external influence on the system by a user.

System Composition and Hardware Parameters

Operating System: Windows, Linux or macOS

Development Environment: code editor (VS Code, CLion, etc.) + CDM16 compiler

Hardware Part: Logisim with the model of the CDM16 processor

*Project Implementation Stages*

* Development and testing of the program in C.
* Porting the program to assembly that is compatible with CDM16.
* Creating a Logisim scheme
* Integration of the program into the Logisim circuit.
* Testing and debugging.
* Technical documentation creation.

**Rules**

Reversi is a strategy board game for two players. The objective is to have more of your colored discs on the board than your opponent when the game ends. Both players take turns dropping their discs on the board, flipping over the opponent's discs to their color. Two colors of discs are used: typically, black and white. At the start of the game, four discs are placed in the center four squares diagonally (see figure 1):

Изображение выглядит как круг, Симметрия, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

*(Figure 1)*

*Objective of the game*

To have more discs of your color on the board when the game ends, which happens when all squares are filled or neither player can make a valid move.

*Gameplay*

1. The player starts with white color. The computer uses black.
2. On your turn, you must place one disc of your color on an empty square so that you capture one or more of your opponent’s discs.
3. Capturing is where the new disc is placed so that in any of the eight directions (diagonal, vertical, horizontal) there is a line of one or more of the opponent's discs ending in a disc of your color (example on figure 2).

Изображение выглядит как круг, Симметрия

Контент, сгенерированный ИИ, может содержать ошибки.

*(Figure 2)*

1. If you have at least one valid move (a move that captures opponent discs), you must make a move. If no valid moves are available, you skip your turn.
2. After placing your disc, all captured opponent discs along all applicable lines are flipped to your color.

The game ends when:

* + All squares on the board are filled, or
  + Neither player can make the correct move.

The winner is the player with the most discs of their color on the board. If both players have the same number of discs, the game is a draw.

You can also read the rules here:

<https://www.mastersofgames.com/rules/reversi-othello-rules.htm>

**Hardware**

In hardware, several problems of the following nature will need to be solved:

*The problem of mapping three states onto a field.*

Logisim LED matrices support only two states (on/off), and for “Reversi” you need to display at least three states: empty cell, first player's chip, second player's chip.

Requirements:

* Implement a hardware circuit that allows the three states to be visually distinguished on each square of a 6×6 field.
* For this purpose, you can use two overlapping LED matrices, by the way, it is better to make them bigger for better visibility.
* Provide independent control of each matrix through the “registers\_to\_matrix” circuit, which stores the states of the cells in 12 six-bit registers (6 for each matrix).

*The problem of convenient and accurate input.*

We need to implement intuitive and accurate input of the player's move in hardware.

Requirements:

* Use 36 buttons, laid out as a 6 by 6 field.
* Implement the scheme to process signals from the buttons.
* Build a scheme that determines the coordinates of the pressed button (row & column) and the current status of the game (going / not going).
* Add processing of the “whose turn it is” signal and the end of the game signal.

*The problem of checking the possibility of a move.*

It is necessary to hardware implement the logic of checking whether it is possible to put a chip in the selected square according to the rules of “Reversi”.

Requirements:

* Implement schemes that check the permissibility of a move in all directions (horizontal, vertical, diagonal).
* For horizontal - schemes “check\_left” and “check\_right” using bitwise shifts and cell state analysis.
* For vertical - converting a column into a row and further processing as horizontal.
* For diagonals - four separate schemes implementing bit shifts on two axes and determining the possibility of stroke and shading boundaries.

*Problem of correct coloring of chips.*

After placing a chip, it is necessary to paint (flip) all the opponent's chips, caught between the new and existing chips of the player according to the rules of “Reversi”.

* Make schemes to paint in all 8 directions from the placed chip, and hardware, so much more fun than software.
* Ensure that the registers are updated correctly and only on the correct move when needed.

**Software**

The software part should implement a bot, which itself should work and compete with the player, solving the following problems:

*Problem with data reception and output*

Data will be received from the hardware level, and it must be handled somehow. Their appearance may not be convenient for working with them in the program, so it is worth to modify them to suit your needs.

Requirements:

* Make a function of decoding 12 numbers coming from the hardware. It is possible to convert them into an array of 36 elements.
* To make a function of encoding the already processed matrix received by us in the form of 12 six-bit numbers to give them to the hardware.
* Then it is also necessary to correctly represent the transmitted coordinates of the last chip (in two numbers from 0 to 5 for example), because they will be transmitted in a form not convenient for work.

*The problem of optimization*

If the bot goes through the entire field while checking where it can be put, it will be slow.

Requirements**:**

* Note that you can only go to a place on the field if it is adjacent to an already placed chip.
* Somehow to use this remark and the coordinates of the chips placed by a real player, transmitted from the hardware, to implement optimization

*The problem of the best move*

As it was already said, the basis of the bot should be a greedy algorithm. But it's boring to just calculate which move will paint the most.

Requirements:

* Make an algorithm that chooses a move by where the most chips will be painted.
* For each position on the field, we need to make some weights that will somehow influence the bot's decision (e.g. corner positions should be prioritized).

*The problem updating the state of the field*

When the bot walks, it must paint over the chips, according to the rules of the game.

Requirements:

* To make coloring in all directions from the chip placed by the bot, for example, using for loops and a matrix obtained by encoding data from the apparatus.

**Progress**

*Development of game logic algorithms in C language:*

* Decoding the input data (12 numbers, each with 6 bits) into a game field matrix (6 x 6).
* Encoding the updated state of the game board back into an array of 12 numbers.
* Realization of the function of determining possible moves of the player and bot (“possible\_moves”).
* Implementation of bot behavior logic (“new\_bot\_logic”) with the ability to flip chips in all directions.

*Creating a Logisim schema:*

* Make a basic “user interface” with implementation of stroke change.
* Make some kind of video memory for matrices.
* Make horizontal, vertical and diagonal check if a chip can be placed.
* Make coloring in all directions.

*Transfer of the game logic from C language to CDM16 assembler:*

* Analyzing the features of the CdM16 assembler.
* Adaptation of all functions to the specifics of the processor.
* Writing the main function that provides correct interaction between game functions according to the game cycle.

*Integrating the program into the Logisim scheme:*

* Connecting data memory (RAM) and instruction memory (ROM) according to Harvard architecture.
* Providing data transfer between external flags and internal memory.
* Testing the program operation on simulation.

*Testing and debugging:*

Checking the correctness of all functions during the development phase C.

* Debugging the compiled code on CDM16 assembler.
* Checking the game loop to ensure that it works in the Logisim scheme.

*Documentation:*

* Creating technical documentation on hardware and software parts.