NOVOSIBIRSK STATE UNIVERSITY

**Profile: Computer science and System engineering**

**TERM PAPER**

**Statsenko Ilya Romanovich**

**Fedotkin Nikita Alekseevich**

**Ermakov Alexandr Vyacheslavovich**

**TANKS**

# Description

**screenshot from the "Battle City"**

The project is a realisation of a game based on "Battle City". The aim of the game is to destroy the enemy tank and avoid being hit by an enemy shell.

The actions of the game take place on a square field of 32×32 pixels. The player's tank and the opponent's tank are located on it. The player's tank is initially located in the centre of the lower part of the field, the opponent's tank - in the centre of the upper part of the field. The enemy tank is controlled by a bot. Tanks are depicted as some set of pixels, which does not go beyond the 3×3 pixel area.

In addition to tanks, there are walls represented by green pixels on the field. They block the movements of the tank and the projectile.

The coordinate plane on the field is standard - up goes the increase of the Y coordinate, to the right - the increase of the X coordinate.

Tank movement directions: up, left, right, down. The tank can move if there are no walls in the direction of its movement. Also the tank can not go outside the field. The tank's hitbox is 3×3 pixels. There must be no walls inside this hitbox.

Every tank must be able to shoot. The projectile has a size of one pixel. Each tank can only release one projectile at a time - if there is already an active projectile, the new command to fire will be ignored. A projectile does not interact with another projectile, they fly through without colliding with each other.

The game will end if any tank is destroyed. A player wins if his shell destroys the opponent. A player will lose if the opponent's shell destroys the player. The game will be tied if the shells hit their targets at the same time.

The game status should be displayed on an additional screen above the main screen.

Note: In all of the following circuits, additional inputs and outputs and additional circuitry may be added as required.

# Project structure and description of schemes

## Screen control scheme

This scheme is responsible for displaying data on the screen. The screen must be accessed by columns. The scheme should be able to output the required screen columns on request of other schemes. Updating of the screen column should be performed in one clock cycle.

**Input Signals:**

* One 32-bit input containing the screen column to be displayed
* One 5-bit input containing the index of the column we are updating
* N 5-bit inputs, each containing the column index required by the other circuitry
* One input connected to the clock

**Output Signals:**

* 32 32-bit outputs to be connected to the matrix
* N 32-bit outputs, each containing the data of the corresponding requested column.

## Tank drawing scheme

This scheme should draw the image of the tank on the given Y coordinate. Since the direction of movement of the tank is only four, it can be encoded with two bits. Encoding in binary form will look like this: 00 - left, 01 - up, 10 - right, 11 - down. The image of the tank can be arbitrary, provided that it fits into a square of 3 × 3 pixels and allows you to visually determine the direction of movement of the tank

**Input Signals:**

* One 5-bit input containing the Y coordinate of the tank
* One 2-bit input containing the direction of the tank

**Output Signals:**

* Three 32-bit outputs containing data for a matrix with a drawn tank

## Scheme of the player's tank

This scheme is fully responsible for the operation of the player's tank. Inside it it is necessary to implement the storage of the direction of the tank and its coordinates. Signals from the player about the beginning of the tank movement must be correctly processed inside this scheme. If the player tries to pass to the area, unavailable to him, the scheme must ignore the request.

It is recommended to use auxiliary self-describing schemes to check the permissibility of a move. It is also necessary to take into account the possibility of resetting the state of the scheme. All necessary data should be stored in registers. This scheme should use the above described *scheme of tank drawing*.

One of the input pins are three 32-bit inputs, with information about the state of the screen in the area where the tank is going to move. When moving perpendicular to the columns of the screen is enough to know only one row, so it is required to use any of the three available inputs. In case of movement parallel to the columns it is necessary to use all three inputs.

**Input Signals:**

* One input connected to the clock
* One input responsible for resetting the circuit state
* One input responsible for tank movement
* One 2-bit input containing the direction of the tank
* Three 32-bit inputs with information about the state of the screen in the area where the tank is going to move.

**Output Signals:**

* Three 32-bit outputs containing data for a matrix with a drawn tank
* Two 5-bit outputs containing X and Y coordinate of the tank
* One 2-bit output containing the current tank direction
* Three 5-bit outputs, each containing the index of the required screen column

## Tank control scheme

This circuit converts the symbol from the keyboard into control signals for other circuits. The tank is controlled using the module "Keyboard" from the standard Logisim library. Control should be performed using the following keys: W - up, A - left, S - down, D - right, SPACE - shot.

Since the keyboard circuit outputs exactly the ASCII code of the pressed character and not the physical key pressed, it is necessary to take into account both Russian and English keyboard layout for ease of operation. The scheme should not use clock cycles for operation.

**Input Signals:**

* One 7-bit input containing the ASCII code of the entered character

**Output Signals:**

* One 2-bit output containing the direction of the tank
* One output contact containing information on the movement key presses
* One output contact containing information on the shot key pressed

## Screen rendering scheme

This scheme sends to the *screen control circuit the* information to be displayed. The process of rendering tanks and shells should be as follows: first the old image of the object should be erased from the screen, and then redrawn in the desired location.

For one clock the scheme must send one column of the screen. Since the image of the tank fits on 3 columns, it takes 6 clock cycles to process it. Similarly, it takes 2 clock cycles to process a projectile. In total, 16 clock cycles are needed to update the image of all objects.

The scheme must support state reset capability. When resetting the state, it is necessary to draw the game map previously written in ROM. It is recommended to use self-written auxiliary schemes for erasing and drawing the image.

**Input Signals:**

* One input connected to the clock
* One input responsible for resetting the circuit state
* Four 5-bit inputs, with coordinate information for player, player's projectile, enemy, enemy's projectile
* Three 32-bit inputs containing data for the matrix with the player's drawn tank
* Three 32-bit inputs containing data for the matrix with the drawn enemy tank
* One 32-bit input containing data for a matrix with the player's drawn projectile
* One 32-bit input containing data for a matrix with a rendered enemy projectile

**Output pins:**

* One 32-bit output containing the screen column to be displayed
* One 5-bit output containing the index of the column we are updating

## Control scheme for the auxiliary screen with game status information

This scheme controls a 31×7 pixel display (31 columns, 7 rows) that shows the current game state. The following text should be displayed for each game state:

* **PLAY** - the game continues;
* **YOU WIN** - the player wins;
* **YOU LOSE** - defeating the enemy;
* **DRAW** is a draw.

The matrix should be accessed row by row. The scheme should independently determine whether the game state has changed or not. In case of change it is necessary to refresh the screen. It should be taken into account that at the moment of the first start of the game, the screen is completely black and no signals about the game state change are received. Therefore, it is necessary to add an additional contact that will forcibly start the rendering process. The font of the text can be arbitrary. All these texts must be loaded in advance into ROM.

**Input Signals:**

* One input connected to the clock
* One input with information about the player's victory
* One input with information about the opponent's victory
* One input responsible for starting the circuit operation

**Output Signals:**

* Seven 31-bit outputs containing the screen line to be displayed

## Scheme projectile

This scheme is responsible for moving, drawing and checking for projectile hits. When the scheme receives a signal on a certain input, it must launch a projectile in a given direction. The direction is encoded with two bits (the encoding is similar to the direction of the tank). If there is already an active projectile, the trigger signal should be ignored.

The projectile must be drawn as a single pixel. Once launched, the projectile must not change its direction. The projectile is destroyed only in the following cases: leaving the playing field, collision with a wall or with a target.

To determine a hit, you should compare the coordinates of the projectile and the coordinates of the tank, which is considered to be the target. This approach will allow to make the scheme universal and use it for both the player and the enemy. The scheme should support the ability to reset the state

**Input Signals:**

* One input connected to the clock
* One input with shot information
* One input responsible for resetting the circuit
* One 2-bit input with information about the direction of the tank launching the projectile
* Two 5-bit inputs containing X and Y coordinate of the tank launching the projectile
* Two 5-bit inputs containing the X and Y coordinate of the target

**Output Signals:**

* One 32-bit output containing the screen column to be displayed
* One 5-bit output containing the index of the column we are displaying
* One output with target destruction information

## Enemy control scheme

This scheme is fully responsible for the operation of the enemy tank. Inside it it is necessary to realise the storage of the tank direction and its coordinates. It should be based on the scheme of the player's tank and modify it so that it can receive data from the processor and convert them into the necessary control signals.

**Input Signals:**

* One input connected to the clock
* One input responsible for resetting the circuit state
* One 16 bit input containing data from the processor

**Output Signals:**

* Three 32-bit outputs containing data for a matrix with a drawn tank
* Two 5-bit outputs containing x and y coordinate of the tank
* One 2-bit output containing the current tank direction

**Processor schematic**

The scheme implements the creation of control signals for movement and shot.

The processor will have a Harvard architecture, which will allow us to apply algorithms that use a lot of memory.

The processor will use the BFS graph traversal algorithm (traversal in width) to calculate the shortest route to the player, taking into account the walls. For this purpose, a map of possible bot positions will be created, taking into account its size, and route calculations will be performed on its basis.

The algorithm works on the principle of step-by-step expansion of the search area from the starting point: from the current position of the bot, all valid neighbouring cells are sequentially explored until the player's position is reached. All cells are processed in the order of increasing distance from the starting point, which guarantees finding the shortest path in terms of the number of steps. When the algorithm is finished, the path to the player is reconstructed using information about the previous steps.

Initially, the algorithm will calculate the path from the initial position of the bot to the initial position of the player. Recalculation of the route will be performed with a specified periodicity, which will allow to take into account the player's movements on the map. In the future, the algorithm of calculating the path can be changed, if technical difficulties or its long running time are identified.

The implementation of the shot mechanics is done by comparing the coordinates of the enemy and the player. To check the visibility between the enemy and the player, arrays of prefix sums for each row and each column of the map are used to check the presence of walls in the line of fire in one mathematical operation.

Data intercept circuitry is used for input to the processor. The CDM-16 general purpose register is used to output data from the processor.

**Input Signals:**

* One input connected to the clock
* Two 16-bit inputs containing the x and y coordinates of the player's tank.

**Output Signals:**

* One 16-bit output containing the bot's direction code or shot code.