**Отображение игрового поля**

Светодиодные матрицы в Logisim поддерживают только два состояния (а соответственно и цвета) – выключено и включено (поступает сигнал 0 или 1). Но для нашего проекта этого мало, ведь игра реверси требует минимум три цвета: цвет поля, цвет фишек первого игрока, цвет фишек второго игрока. Чтобы решить эту проблему, мы решили использовать две наложенные друг на друга светодиодные матрицы (рисунок 1, синий прямоугольник). Нижняя светодиодная матрица размером 6 x 6 пикселей отвечает за отображение игрового поля и фишек компьютерного игрока. Верхняя матрица, имеющая размер 7 x 6 отображает фишки игрока (ваши) (седьмой столбец нужен для предотвращения совпадения входов светодиодных матриц). Управление отображением осуществляется через схему “registers\_to\_matrix” (рисунок 1, синяя стрелка), подключённую к входам обоих матриц. Она выполняет не только функции видео памяти, но и объединяет в себе почти все механики нужные для работы игры. “registers\_to\_matrix” мы еще подробно рассмотрим в следующих пунктах, сейчас же важно понимать, что она содержит 12 шестибитных регистров – по 6 для каждой матрицы, в которых хранится всё текущее игровое поле.

Помимо матриц есть еще несколько элементов, нужных для лучшего понимания хода игры (рисунок 1, зеленые прямоугольники). Все они также управляются схемой “registers\_to\_matrix”.

**Изображение выглядит как текст, снимок экрана, диаграмма, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.**

Рисунок ("main")

**Управление и ввод данных**

Для взаимодействия игрока с игровым полем используется 36 кнопок, которые объединены в 6 расположенных друг под другом рядов, с помощь схемы “buttons\_row” (рисунок 1, красный прямоугольник 2). Все кнопки, а именно, ряды в которых они находятся, постоянно обрабатываются схемой “baza” (рисунок 1, желтый прямоугольник), в которой происходит определение текущего ряда (как трехбитного числа, где самый верхний ряд — это “000”, а самый нижний “101) (рисунок 2, красный прямоугольник 1) и столбца (шестибитного числа, где самый правый столбец это “000001”, а самый левый “100000”) (рисунок 2, красный прямоугольник 2) нажатой на игровом поле кнопки, если при этом идет игра, состояние о которой хранится в регистре в схеме “baza” (рисунок 2, синяя стрелка), устанавливается в 1 когда была прожата кнопка старт и сбрасывается либо при нажатии кнопки “reset” либо при поданном из “registers\_to\_matrix” индикаторе “game\_over”. Помимо этого, схема имеет выход, означающий чей сейчас ход (рисунок 2, зеленая стрелка). Все данные из этой схемы поступают в “registers\_to\_matrix”, где происходит вся обработка (можно ли поставить туда фишку и последующее закрашивание).

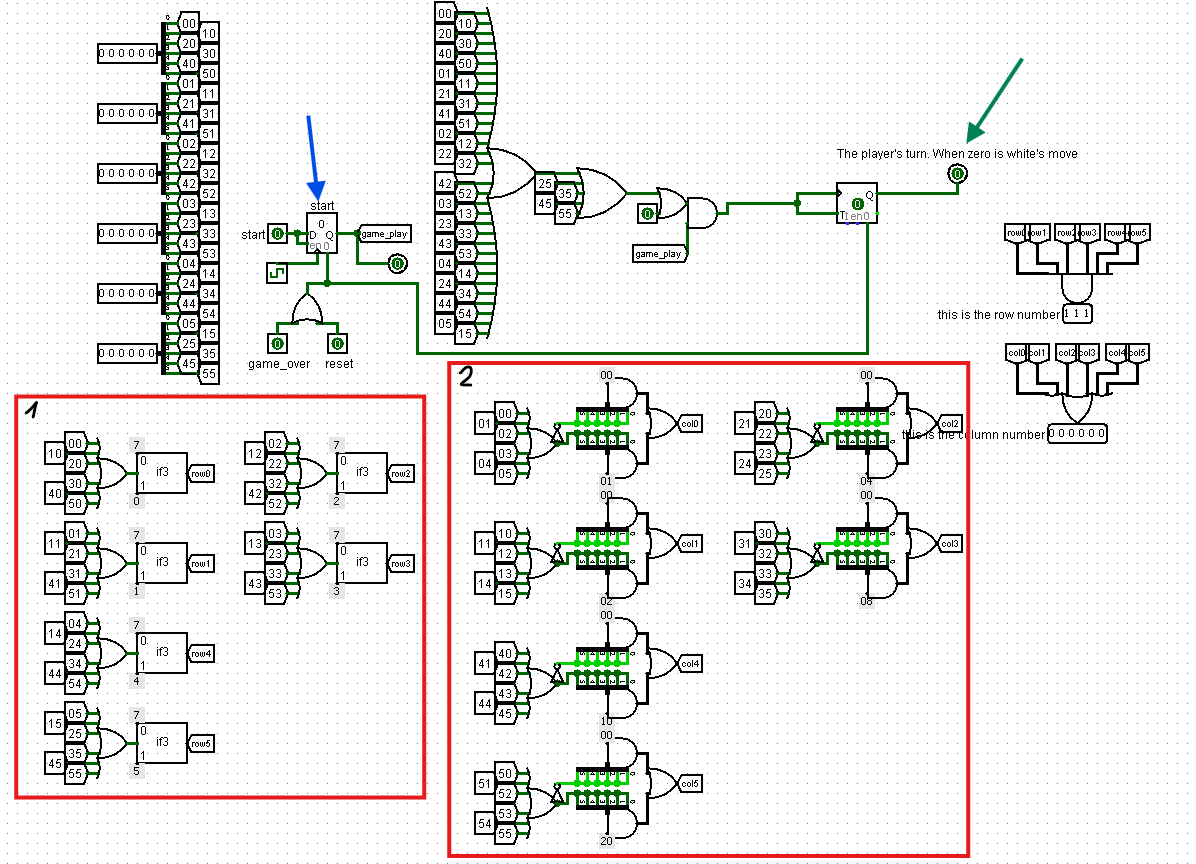


Рисунок ("baza")

**Обработка хода игрока (можно поставить/нет)**

Все схемы, отвечающие за обработку того, можно ли поставь фишку в выбранное на карте поле содержатся внутри “registers\_to\_matrix”. Для этой проверки, они используют такие данные как: “turn” – чей сейчас ход, “column” и “row”, полученные из “baza” и все текущее поле из 12 регистров (рисунок 3, желтое выделение). Значения на регистрах обновляются только в случае, если ход игрока возможен (рисунок 3, синие выделение).

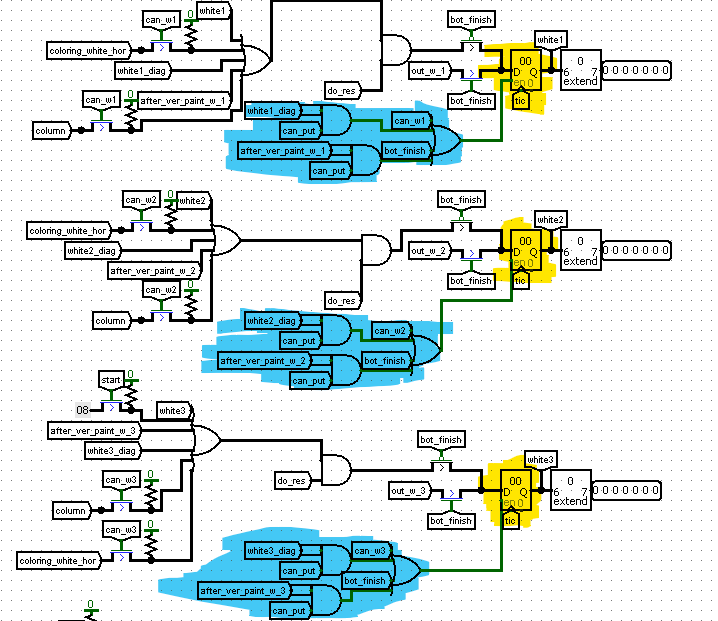


Рисунок (пример регистров из “registers\_to\_matrix”)

Когда игрок пытается поставить фишку, аппаратная часть проверяет возможно ли это по горизонтали (влево и вправо от места куда пользователь пытается поставить), вертикали (вниз, вверх) и по диагонали (вниз-влево, вниз-вправо, вверх-влево, вверх-вправо).

**Проверка по горизонтали**

Схемами для проверки по горизонтали являются схемы “check\_left” и “check\_right”. Они имеют одинаковый принцип работы, так что достаточно рассказать только про одну из них. На вход “check\_right” получает три шестибитных контакта (рисунок 4, желтое выделение): текущие белый и черный ряды из регистров (только тот ряд, в который мы пытаемся поставить нашу фишку) и горизонтальная координата места куда мы пробуем поставить фишку (можно сказать текущий выбранный столбец). При помощи побитовых сдвигов, выполняемых на контакте “current”, осуществляется проверка, справа от того места, которое нас интересует. На первых двух сдвигах проверяется только не дошла ли единичка до края и не является ли клетка на текущем сдвиге черной (потому что если в соседней клетке не черная фишка, то и закраска в ту сторону невозможна), потом же добавляются проверки уже на присутствие белой фишки. Если на каком-то из сдвигов в ходе проверки единица была послана на туннель “break”, то проверка останавливается и на выходной контакт передается 0/1, в зависимости от возможности хода. Такая реализация позволяет за один такт проверить возможность хода по горизонтали (рисунок 4, красные прямоугольники).

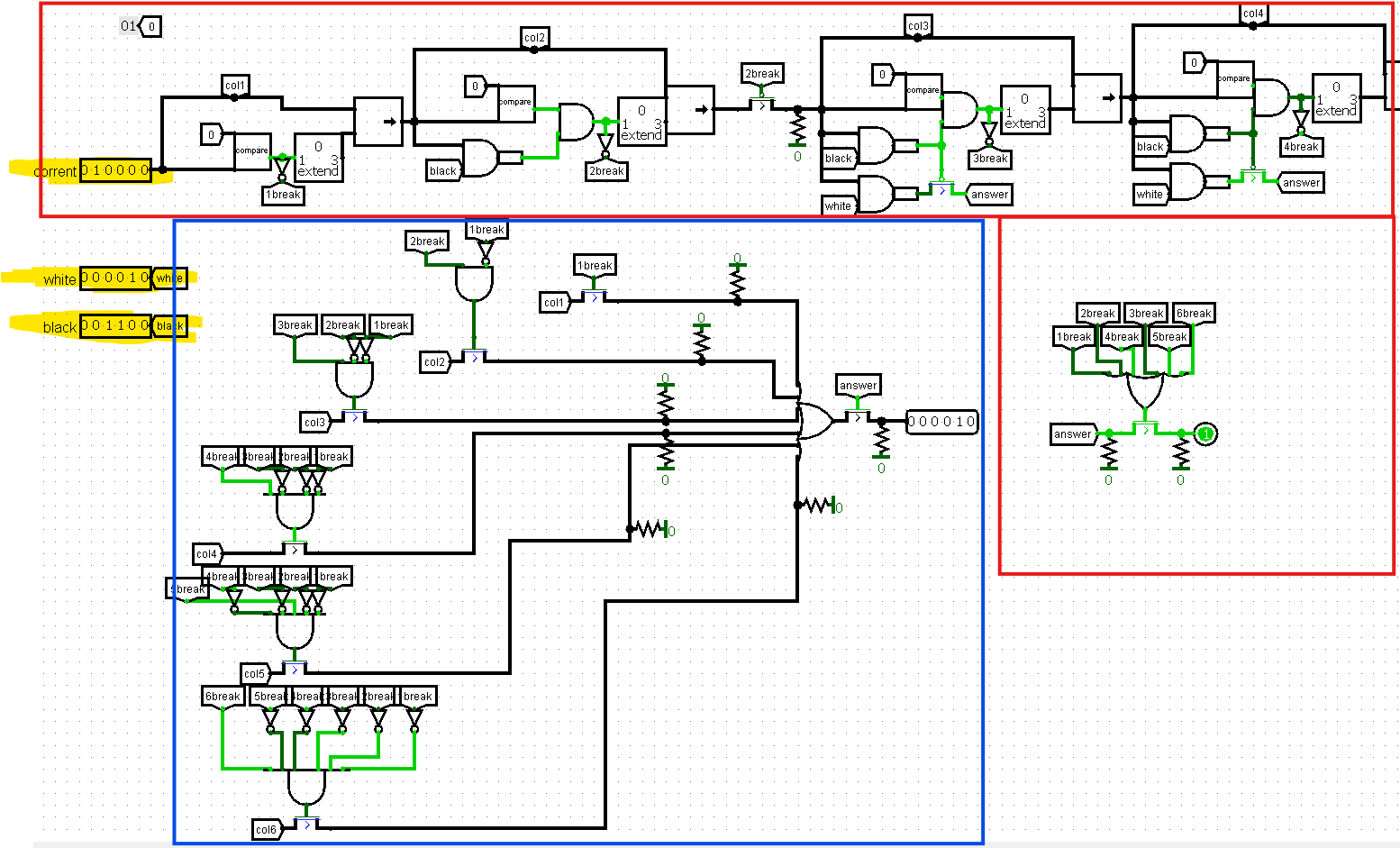


Рисунок ("check\_right")

Так же на схеме “check\_right” (как в общем то и на других схемах, которые проверяют, можно ли поставить фишку), есть выходной контакт, который служит для определение места до которого можно закрашивать, от места куда мы пытаемся поставить фишку (рисунок 4, синий прямоугольник). Этот контакт будет играть большую роль в схемах, созданных для закрашивания.

**Проверка по вертикали**

Для проверки по вертикали используются те же схемы что и для проверки по горизонтали, отличие заключается лишь в том, что нужно как-то получить весь черный и белый столбец и интерпретировать номер ряда, в который хотим поставить фишку, как шестибитное число. С этими задачами справляется схема “column\_to\_row” и блок из схемы “registers\_to\_matrix”.

Изображение выглядит как диаграмма, снимок экрана, текст, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как диаграмма, План, Технический чертеж, схематичный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок (блок перевода) Рисунок ("column\_to\_row")

**Проверка по диагонали**

Для проверки по диагонали у нас реализовано 4 схемы. Они работают по одному принципу, но проверяют в разных направлениях, от теоретической фишки. Разберем как работает схема “check\_diag\_ru” (проверка справа сверху от фишки). Ее работа похожа на работу “check\_right” присутствием битовых сдвигов, однако тут передается абсолютно вся текущая матрица, как белых, так и черных фишек и сдвиги происходят не только по горизонтали, но и по вертикали (т. е. движется не только по колонкам, но и параллельно этому по рядам). В сдвигах по горизонтали здесь мы просто проверяем, что не дошли до края, если дошли, то вызываем “break”, так как дальше идти нельзя (рисунок 7, синий прямоугольник). В сдвигах же по вертикали мы так же, как это было в схеме “check\_right”, делаем проверки на наличие черной или белой фишки на проверяемой клетке поля, прерываясь в случае чего (рисунок 7, красный прямоугольник). Схема также выводит горизонтальную координату клетки, до которой будет происходить закрашивание. Все работает в один такт, кстати говоря.

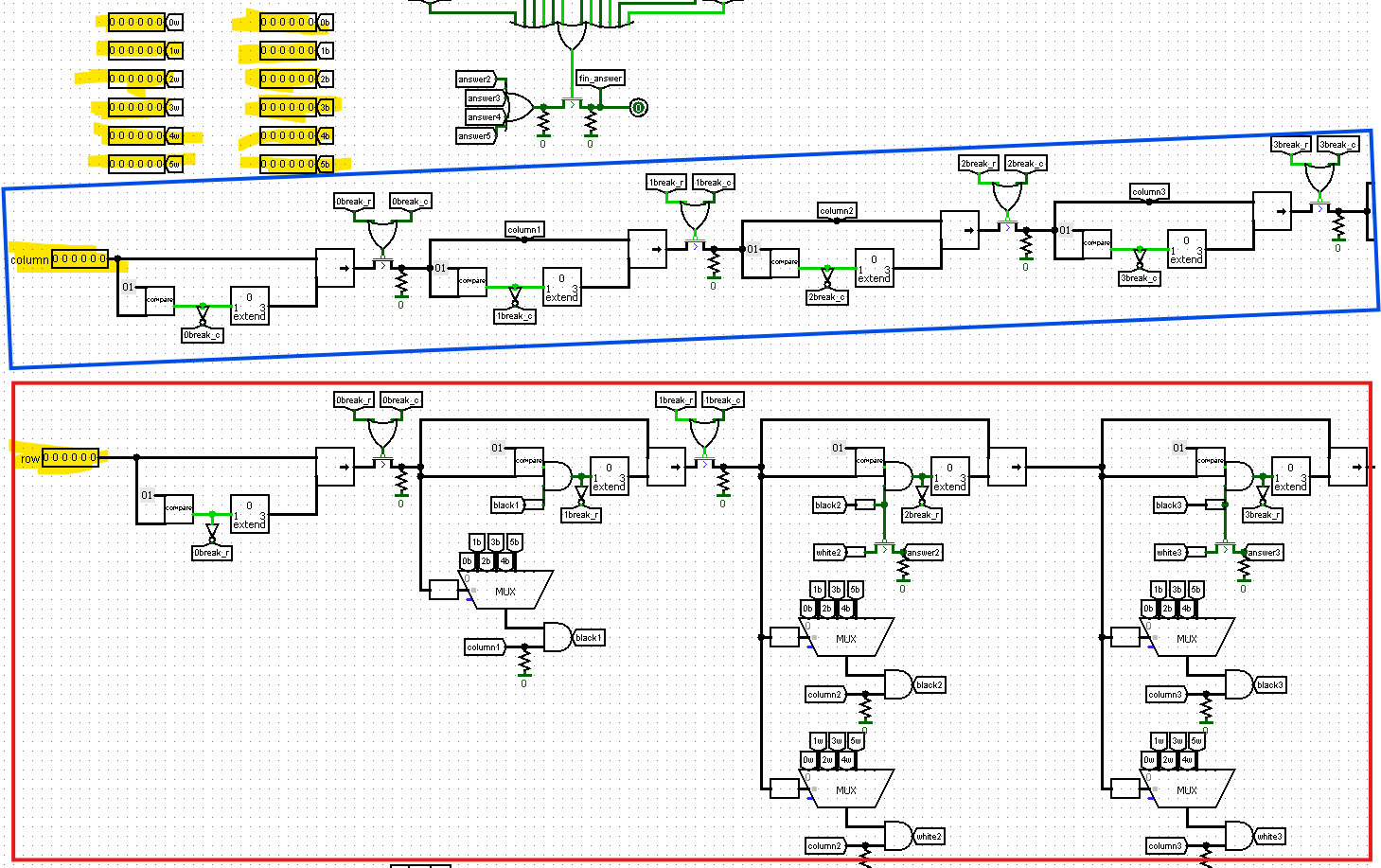


Рисунок (“check\_diag\_ru”)

**Закрашивание фишек**

Все схемы для закрашивания фишек так же хранятся внутри уже почти изученной нами “registers\_to\_matrix”. Их всего две: одна для закрашивания по горизонтали и по вертикали “coloring\_hor” (рисунок 8), другая для закрашивания по диагонали “coloring\_diag” (рисунок 8.1).

Изображение выглядит как текст, диаграмма, снимок экрана, символ

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 8 (appearance "coloring\_hor") Рисунок 8.1 (appearance “coloring\_diag”)

**Закрашивание по горизонтали**

Как уже было сказано, схема для закраски по горизонтали, только одна – “coloring\_hor”. Потому что в ней закрашивается весь ряд и справа и слева (если можно) от фишки. Какие входы есть у этой схемы? Во-первых, горизонтальная координата места куда мы пытаемся поставить фишку, во-вторых, текущий белый и черный ряд, в которые мы хотим поставить фишку, ну и те самые горизонтальные координаты клетки до который нужно производить закрашивание, причем координаты как левее от текущей попытки, так и правее, чтобы можно было закрашивать в обе стороны. Закраска немного похожа, на проверку, только здесь мы проверяем не дошла ли наша сдвинутая координата не до края поля, а до фишки, до которой надо закрашивать. Так же мы здесь не просто сдвигаем и ждем пока получим результат, а на каждом сдвиге делаем побитовое or для белого ряда и xor для черного ряда (рисунок 9).

Изображение выглядит как диаграмма, План, текст, схематичный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок (“coloring\_hor”)

В результате схема выводит обновлённые белый и черный ряды, которые потом запишутся на регистры в схеме “registers\_to\_matrix” (рисунок 10).

Изображение выглядит как диаграмма, текст, План, Технический чертеж

Контент, сгенерированный ИИ, может содержать ошибки.

**Закрашивание по вертикали**

Закрашивания по вертикале реализует та же схема, что и по горизонтали “coloring\_hor”. Работает это так же, как и с вертикальной проверкой, а именно с помощью уже известных нам схем (рисунок 5 и рисунок 6) столбец преобразуется в ряд и с ним идет работа как с горизонталью.

**Закрашивание по диагонали**

Закрашивание по диагонали, а именно “coloring\_diag”– пожалуй одна из самых сложных и объемных схем.