МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ"
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ"
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

15.03.06 - Мехатроника и Робототехника

Направление (профиль) : Искусственный интеллект

КУРСОВАЯ РАБОТА

Тема проекта: **ЗМЕЙКА**

Быков Антон

Голомолзин Даниил

Кириллов Евгений

Новосибирск

2025

ТАБЛИЦА СОДЕРЖАНИЯ

1 ТЕРМИНЫ И АББРЕВИАТУРЫ

CdM8	Coco de Mer 8 Mark5 Reduced – процессор, построенный на основе Logisim
CocoIDE	Среда для написания кода на низкоуровневом языке программирования
Logisim	Инструмент для разработки и моделирования логических схем

2 ВСТУПЛЕНИЕ

Тема нашего совместного проекта - это легендарная и всеми знакомая игра *“Змейка”*. Эта игра была разработана в 1997 году Танели Арманто (разработчиком из Финляндии) и некоторое время спустя она появилась в Nokia 6610.

Правила игры максимально просты. На игровом поле 14x14 ползет маленькая змейка 2 пикселя в длину. Поедая своеобразные *“Яблочки”,* змейка увеличивает свою длину пока не заполнит все игровое поле своим телом, либо пока не оступится. Закончить игру можно несколькими способами:

- перекусывание змейки самой себя
- смерть от голода змейки
- заход на границу игрового поля

Цель игрока, играя за змейку: набрать 10 яблочек.

Проект был реализован на **Logisim** с использованием процессора **CdM8** и написанием кода для него с помощью **CocoIDE**. Все вышеперечисленные инструменты проекта являются частью нашей учебной программы по Цифровым Платформам.

3 НАЗНАЧЕНИЯ И ОБЛАСТЬ ПРИМЕНЕНИЯ

Целью программы является ее реализация на низкоуровневых платформах, таких как ассемблер и логических схем. Программа предназначена для использования в игровой и развлекательных сферах.

4 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

- Управление змейкой:

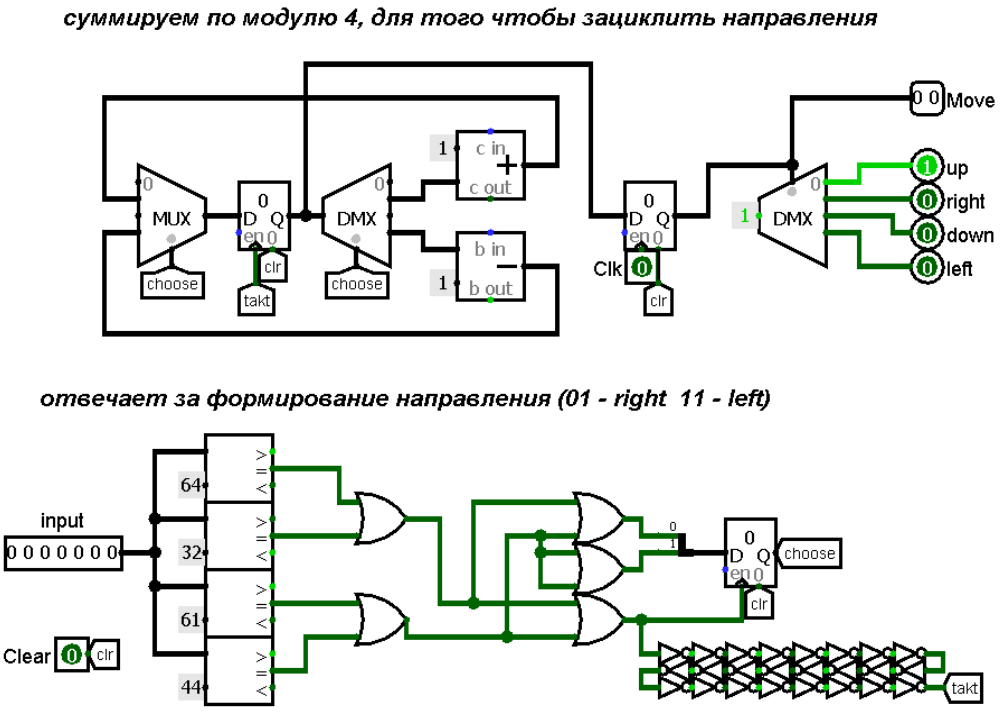

рис. 1.1

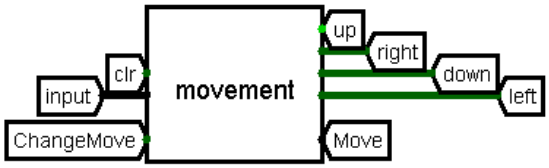

рис. 1.2

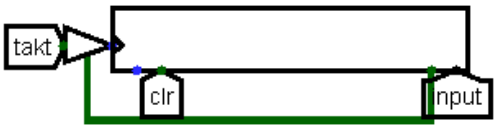

рис. 1.3

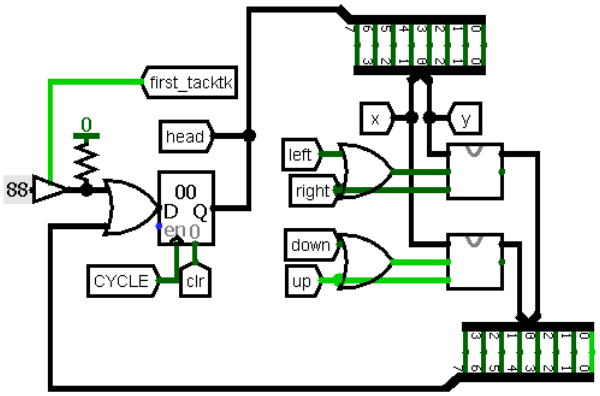

рис. 1.4

Из клавиатуры (рис. 1.3) получаем символ (a/d/ф/в) - вправо или влево (input). Изменение направления на кольце вычетов по модулю 4 (рис. 1.1) , где поворот вправо это +1, влево -1. Изменение происходит при поднятии ChangeMove (поднимается процессором). Move - текущее направление. Из-за управления с помощью двух кнопок (управление относительно головы), исключается случай разворота змейки на 180°. Сигнал из туннелей (рис. 1.2) подается в схему, изменения координат головы (рис. 1.4)

- Генерация еды

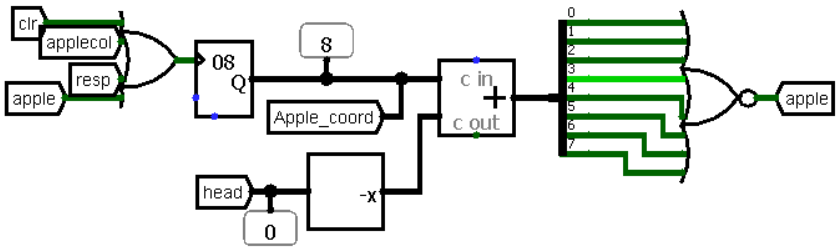

рис. 2.1

Генерация яблока происходит с помощью генератора случайных чисел, происходит сложение координаты яблока и отрицание координаты головы, если в сумме получается 0, значит голова находится на одном и том же месте, что и яблоко, следовательно яблоко съедается.

- Механика роста

Механика роста описана в Функциональных характеристиках в схеме (рис. 9.1)

- Система коллизий

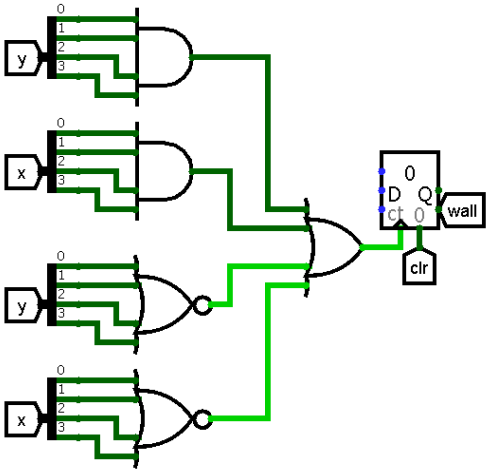

рис. 4.1

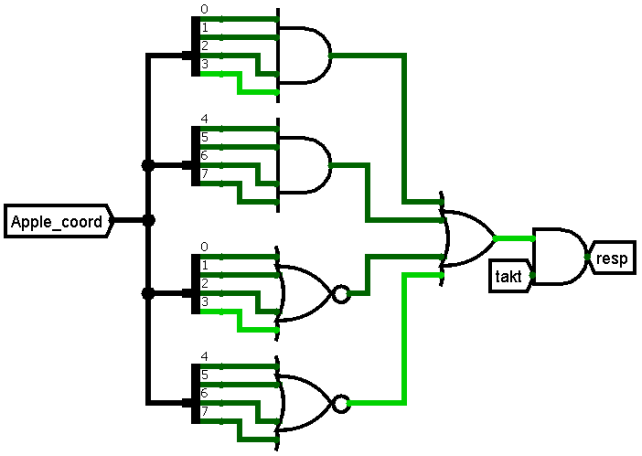

рис. 4.2

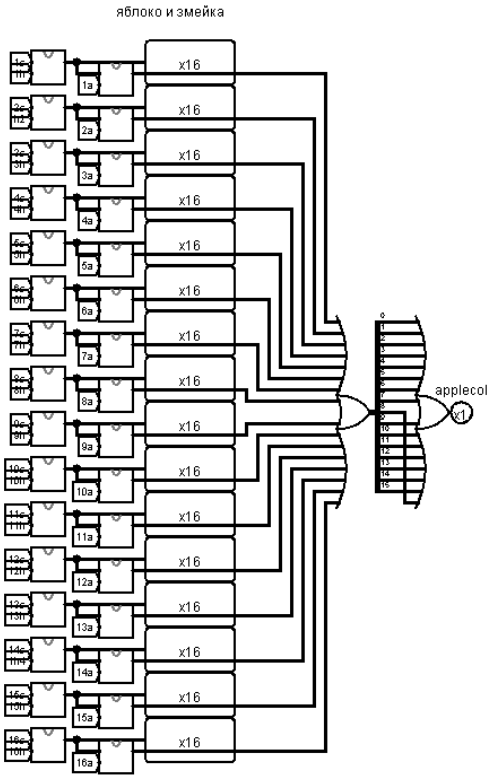

рис. 4.3

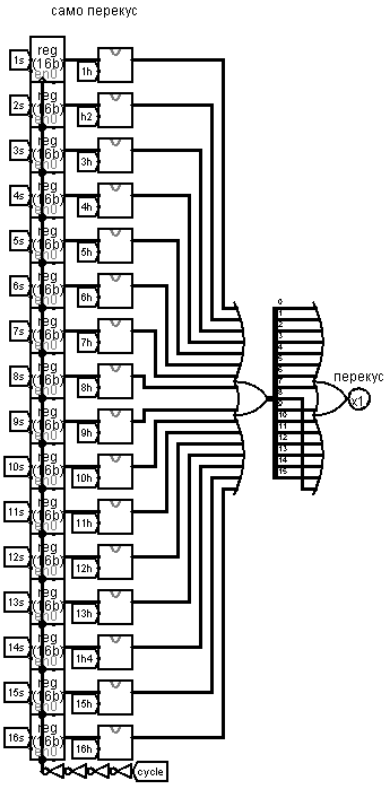

рис. 4.4

Схема (рис. 4.1) отвечает за коллизию заходе змейки на стену, подавая сигнал в туннель wall, который остановит схемы (рис. 8.4)

Схема (рис. 4.2) помогает избежать коллизии с яблоком, которое сгенерировалось на стенке игрового поля.

Схема (рис. 4.3) совершает логическое AND строчки матрицы и строчки с яблоком, если на выходе не пусто, то поднимается applecol. Аналогичным способом работает схема (рис. 4.4), которая проверяет само перекус змейки. Далее передавая сигналы в схемы под номерами 10.6 и 10.7

- Счетчики и игровая статистика

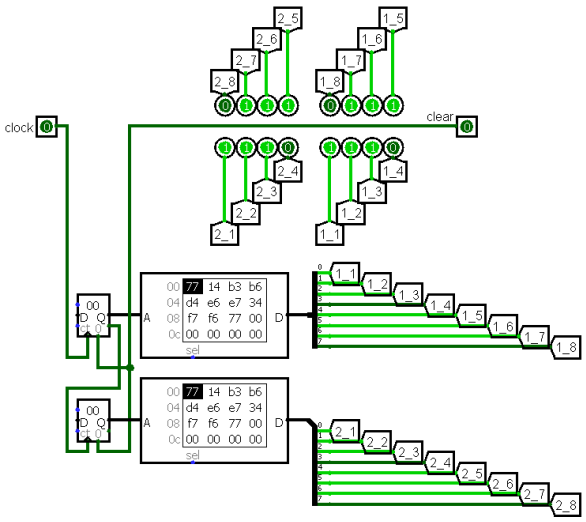

рис. 5.1

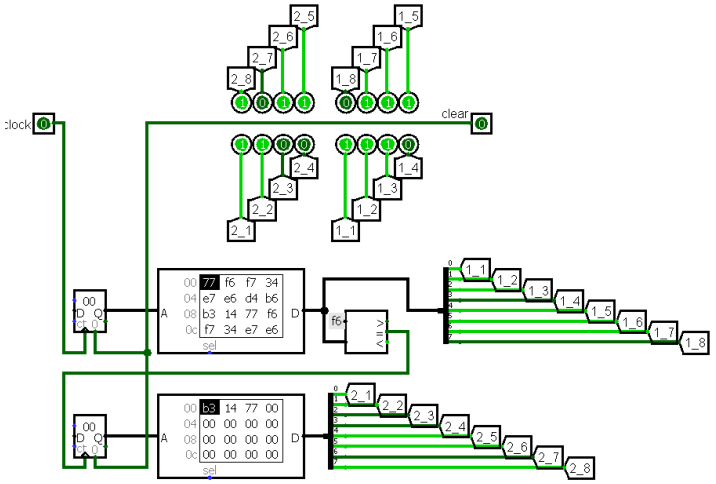

рис. 5.2

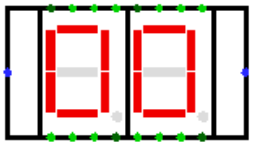

рис. 5.3

Счетчики организованы с использованием нескольких связанных ПЗУ.

Прямой, увеличивающий значение, счетчик очков (рис. 5.1) последовательно переходит в памяти на след ячейку памяти, сменяя значение индикатора на след цифру. При достижении цифры 9 подается сигнал на след память, увеличивая цифру десятков. Обратный, уменьшающий значение, счетчик голода, ведет обратный отсчет от 20 до 0 похожим способом, при подаче сигнала, происходит переход на ячейку памяти с цифрой меньше текущей.

Вывод будет происходить на индикаторы показанные на (рис. 5.3)

- Реализация игрового поля

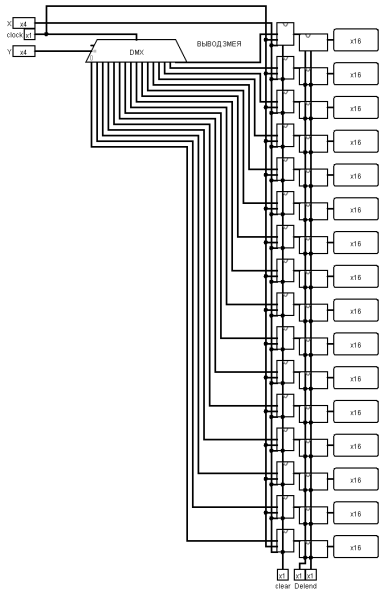

рис. 6.1

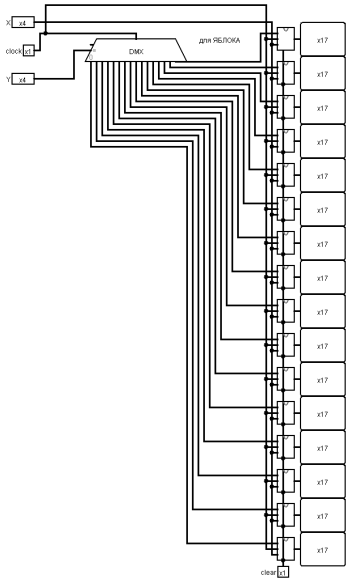

рис. 6.2

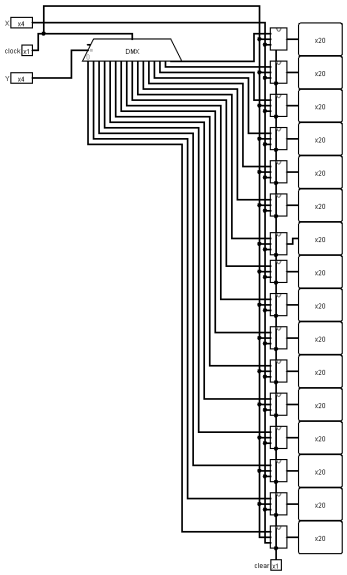

рис. 6.3

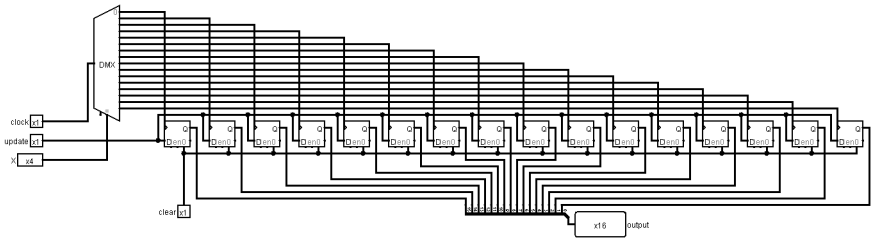

рис. 6.4

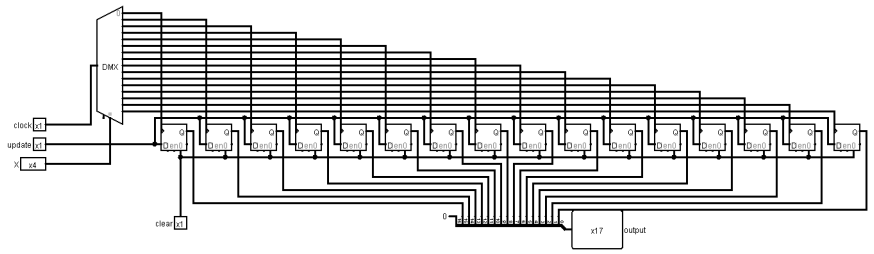

рис. 6.5

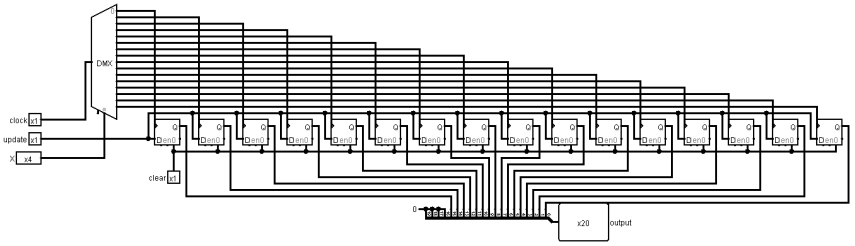

рис. 6.6

Вывод на игровое поле был реализован путем создания регистров. 16-битные регистры используют D-триггер для отображения змейки на дисплей. 17-битные регистры используются для отображения яблока. 20-битные регистры используются для вывода головы. Схемы (рис. 6.1), (рис. 6.2) (рис. 6.3) используют схемы (рис. 6.4), (рис. 6.5) (рис. 6.6) соответственно.

- Дополнительные схемы
    1. Голод змейки

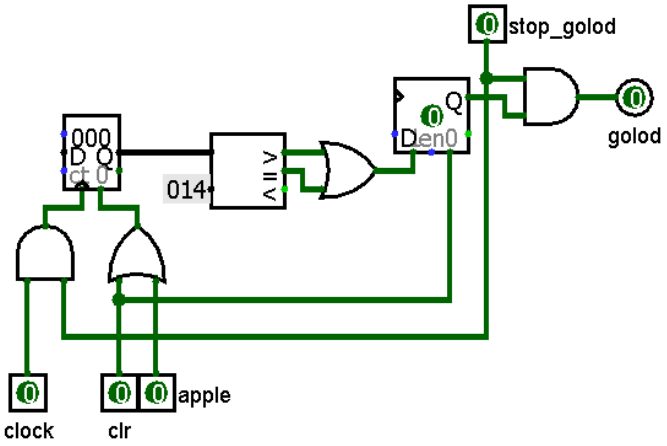

рис. 7.1

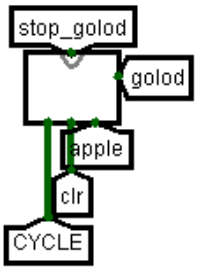

рис. 7.2

Голод змейки основан на подсчете кол-ва тактов генератора, каждый такт увеличивает счетчик, далее значение сравнивается с константой (20) и при совпадении / переполнении срабатывает триггер, поднимая сигнал Голода, сообщая что змейка умерла от Голода.

2. Общие схемы

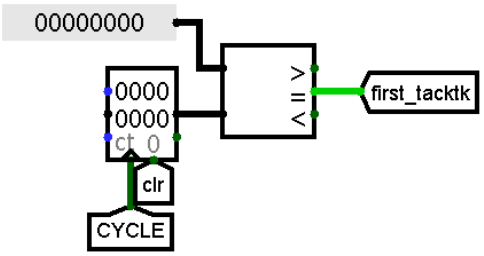

рис. 8.1

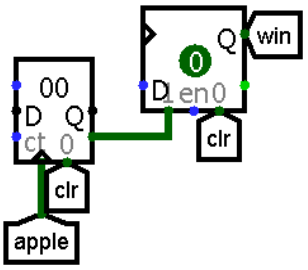

рис. 8.2

Схема (рис. 8.1) отвечает за первый такт, который подает сигнал схеме (рис. 1.4) и определяет положение змейки в начале игры. По умолчанию это центр игрового поля.

Схема (рис. 8.2) считает кол-во съеденных яблок. При достижении максимального значения счетчика (значение 10) срабатывает триггер и подается сигнал в схему (рис. 8.3), который перекрывает срабатывание тактового генератора.

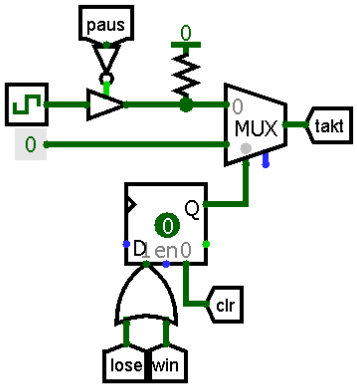

рис. 8.3

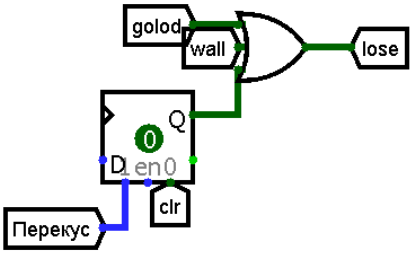

рис. 8.4

Схема (рис. 8.3) отвечает за перекрытия передачи сигнала генератора на схемы проекта при неблагоприятных условиях, с помощью триггера. Так же организована пауза.

Схема (рис. 8.4) отвечает за условия, которые ведут к проигрышу игрока: Само Перекусывание, истечение времени голода, стена.

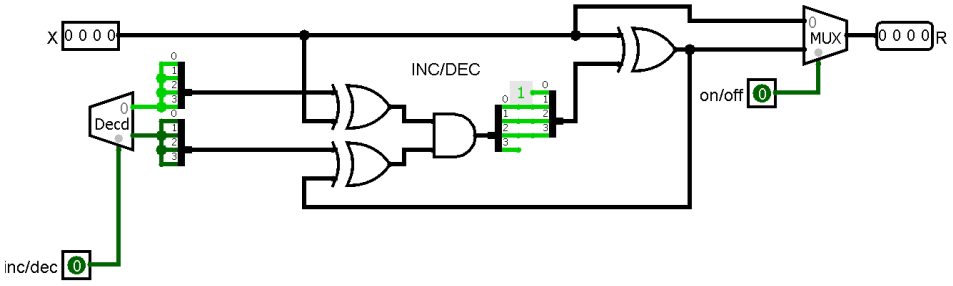

рис. 8.5

Схема (рис. 8.5) - INC-ементор ИЛИ DEC-ементор. Используется для увеличения / уменьшения на единицу 4 битного числа в схеме (рис. 1.4) для просчитывания координат головы.

5 ФУНКЦИОНАЛЬНЫЕ ХАРАКТЕРИСТИКИ

Особенность нашего проекта заключается в процессоре. Именно на него возложена большая часть обработки данных. Он отвечает за просчитывание координат змейки и за правильность вывода на игровой дисплей.

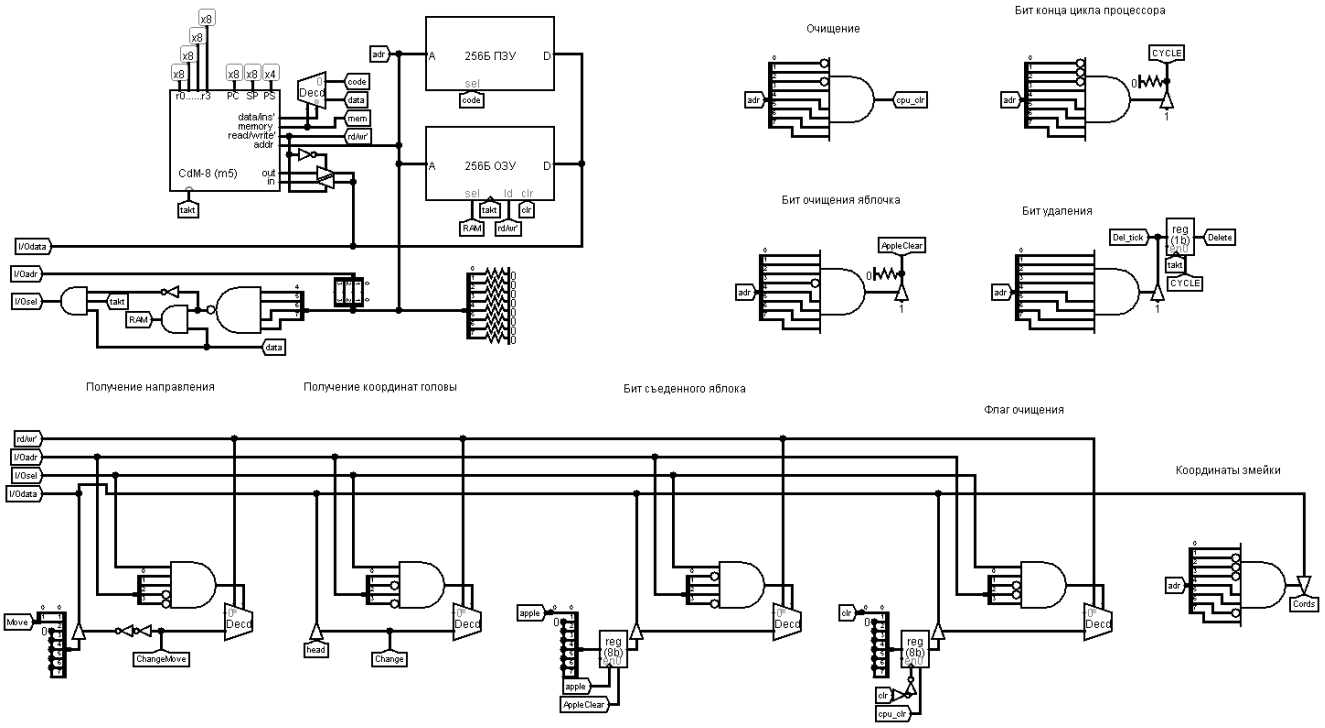

рис. 9.1

Основная идея, используемая в коде для процессора:

- Вводы в процессор:
    1. Head – координаты головы
    2. Move – направление движения
    3. Apple - съедено ли яблоко
    4. Clr – нажатие кнопки очищения
- Вывод:
    1. Через Cords – два пикселя: один на добавление, и один удаление
    2. Cycle - бит окончания цикла.
    3. AppleClear – для очищения регистра входных данных Apple.
    4. Delete – для дисплея, говорит, что пиксель должен удаляться.
    5. Del_tick – очищения регистров (иначе, так как до этого мы добавляли пиксель, то добавленный пиксель сразу удалится) ((на схеме output_matrix вход для Delt_tick обозначен vivod)).
    6. Cpu_clr – поднимается процессором для повторного очищения, т.к. между clr с кнопки и очищением процессора может пройти цикл.

Для ввода в памяти процессора используются ячейки 0xf0 – 0xff, при обращении к ним мы получаем данные из Logisim.

Для внутренних переменных (Tale, PrevMove) и для выполнения вывода (Res), используются ячейки на b строке памяти.

```
# Data
  asect 0xf3
```

```
Move:
  asect 0xb4
PrevMove:
  asect 0xb1
Res:
  asect 0xb5
Tale:
  asect 0xf5
Head:
  asect 0xf6
Apple:
  asect 0xf7
AppleClear:
  asect 0xf9
clr:
  asect 0xfa
cpu_clr:
  asect 0xff
Delete:
# End of data
  asect 0xf8
Ending:
```

В начале запуска процессора (метка start), мы сохраняем в памяти начальные координаты хвоста 0x87, и закидываем в стек начальный сегмент змейки (направление: вверх, длина: 2). Если поднимается бит очищения, то процессор возвращается на метку start.

```
  asect 0x00
start:
# инициализация змейки
  ldi r0, cpu_clr
  ld r0, r0              # вызываем clr в логизиме (для стабильности)
  setsp 0xf0              # устанавливаем SP значение 0xf0

  ldi r1, 0b00000011
  push r1                # закидываем в память начальный сегмент змейки

  ldi r1, Tale           # сохраняем начальные координаты хвоста
  ldi r0, 0x87
  st r1, r0

  ldi r0, Ending         # поднимаем бит Cycle
```

```
ld r0, r0
```

Змея хранится сегментами в стеке в формате 1 байта: 2 бита на направление, 6 бит на длину сегмента (с запасом). Длина первого сегмента увеличивается на 1, если не съедено яблоко длина последнего уменьшается на 1. Если последний сегмент становится нулевой длины, то это приводит к сдвигу стека. Считывается движение змейки и сравнивается с предыдущим, если не совпадают, то создаем новый сегмент. Перезаписываем предыдущее направление.

В памяти процессора хранятся координаты хвоста, если яблоко не съедено, то мы удаляем предыдущий хвост и, в соответствии с последним сегментом, изменяем координаты хвоста и сохраняем их в памяти.

```
Main:
        # проверка на clear
        ldi r0, clr
        ld r0 , r0
        if
          tst r0
        is nz
          br start
        fi

        # считывание пикселя для добавления
        ldi r0, Head
        ld r0, r0
        ldi r1, Res       # сохраняем новое положение головы
        st r1, r0                 # и выводим в логзим

        # проверка на поворот
        ldi r0, Move
        ld  r0, r0                # получаем текущее направление
        ldi r1, PrevMove
        ld r1, r2                 # получаем предыдущее направление
        st r1, r0               # на место предыдущего направление - новое

        if
          cmp r2, r0              # двигаемся в том же направление
        is z
          ldsp r3               # адрес первого сегмента
          ld r3, r1             # значение сегмента
          inc r1                # увеличиваем длину сегмента на 1
          st r3, r1             # перезаписываем
```

```
else                    # создаем новый изгиб
# r0: 000000xx -> xx000000
        shl r0
        shl r0
        shl r0
        shl r0
        shl r0
        shl r0
        inc r0            # xx000001 – новый сегмент
        push r0         # сохраняем в стеке
      fi

      # проверка на яблоко
      ldi r0, Apple
      ld r0, r0         # получаем значение бита яблока
      if
        tst r0
      is z              # если не съели
        ldi r0, 0xef    # адрес последнего сегмента
        ld r0, r1       # значение сегмента
        dec r1          # уменьшаем длину сегмента на 1
        st r0, r1               # перезаписываем

        # проверка на нулевую длину сегмента
        # если 0, то сдвигаем стек
        if
          ldi r0, 0b00111111          # маска для получения длины
          and r0, r1                  # получаем длину
        is z                          # если 0
          ldsp r0                     # адрес первого сегмента + указатель на стек
          ldi r1, 0xf0                # адрес "конца" стека

          sub r0, r1                  # получаем количество сегментов
          neg r1                      # результат получается отрицательный

          pop r2                      # получаем первый сегмент + изменяем SP
          inc r0                      # увеличиваем указатель на стек

          # цикл для сдвига стека
          tst r1
          while
          stays nz              # пока r1 /= 0
            ld r0, r3           # получаю новый сегмент
```

```
    st r0, r2           # на его место записываю предыдущий
    inc r0              # увеличиваю указатель
    move r3, r2         # перекидываю новый сегмент в r2
    dec r1              # уменьшаю счётчик
   wend
   st r0, r2            # сохраняю последний сегмент
  fi

  # изменяю координаты хвоста, в соответсвии с последнем сегментом
  ldi r3, 0xef
  ld  r3, r3                    # получаю последний сегмент
  ldi r2, 0b11000000            # маска для получения направления
  ldi r0, 0b01000000            # маска для получения длины
 # выбираю число для изменения хвоста
  if
   and r3, r2                   # направление в r2
  is  z             # up
   ldi r3, 1
  else
   if
    cmp r2, r0
   is z             # right
    ldi r3, 0b00010000
   else
    if
     shl r0
     cmp r2, r0
    is z            # down
     ldi r3, 0b11111111
    else            # left
     ldi r3, 0b11110000
    fi
   fi
  fi
  ldi r0, Tale
  ld r0, r1             # получаю старые координаты хвоста

  ldi r2, Delete
  ld r2, r2             # поднимаю бит для удаления
  ldi r2, Res
  st r2, r1             # вывожу старый хвост на удаление
  add r1, r3            # изменяю координаты хвоста
  st r0, r3                     # перезаписываю хвост
```

```
        fi

        ldi r0, AppleClear     # clear внешнего регистра с Яблочком
        ld r0, r0              # иначе, если использовать Ending, то сразу обнуляется
        ldi r0, Ending
        ld r0, r0          # поднятие флажка окончания цикла
br Main             # go back to the start of the main loop
end
```

6 ВИЗУАЛЬНЫЕ ХАРАКТЕРИСТИКИ

Начальное состояние:

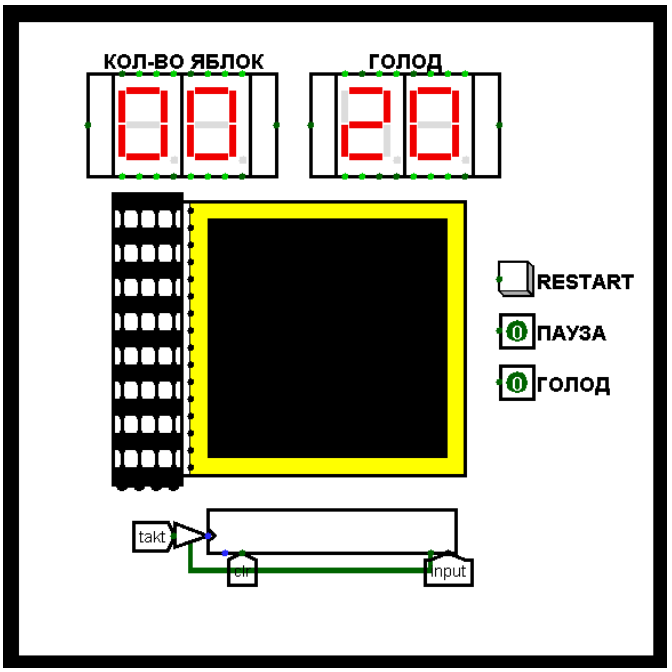

рис. 10.0

Схема (рис. 10.0) является начальным экраном. В нем содержится: счетчик очков КОЛ-ВО ЯБЛОК, оставшееся время до голода змейки ГОЛОД, перезапуск игры через кнопку RESTART, остановка/запуск игры через контакт ПАУЗА, остановка/запуск игры через контакт ГОЛОД и управление организованное через клавиатуру.

Возможные выводы на экран в разных случаях:

рис. 10.1

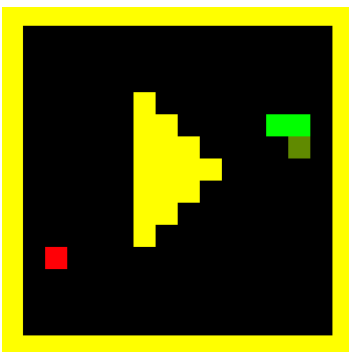

рис. 10.2

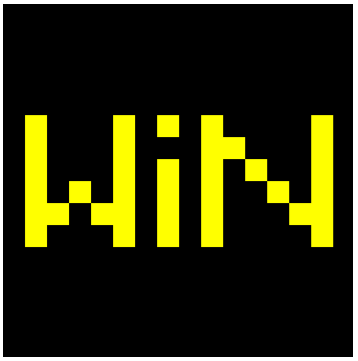

рис. 10.3

Надпись (рис. 10.1) высвечивается при само перекусывании змейки, истечении времени голода, столкновение со стеной, при этом очищая игровое поле от змейки и яблока. Надпись (рис. 10.2) высвечивается при нажатии паузы, сохраняя местоположения змейки и яблока. Надпись (рис. 10.3) высвечивается при выигрыше - достижении 10 яблок

Цвета используемые в проекте:

- Светло-зеленый - 00FF00
- Черный - 000000
- Темно-зеленый - 608A00
- Красный - FF0006
- Желтый - FFFF00

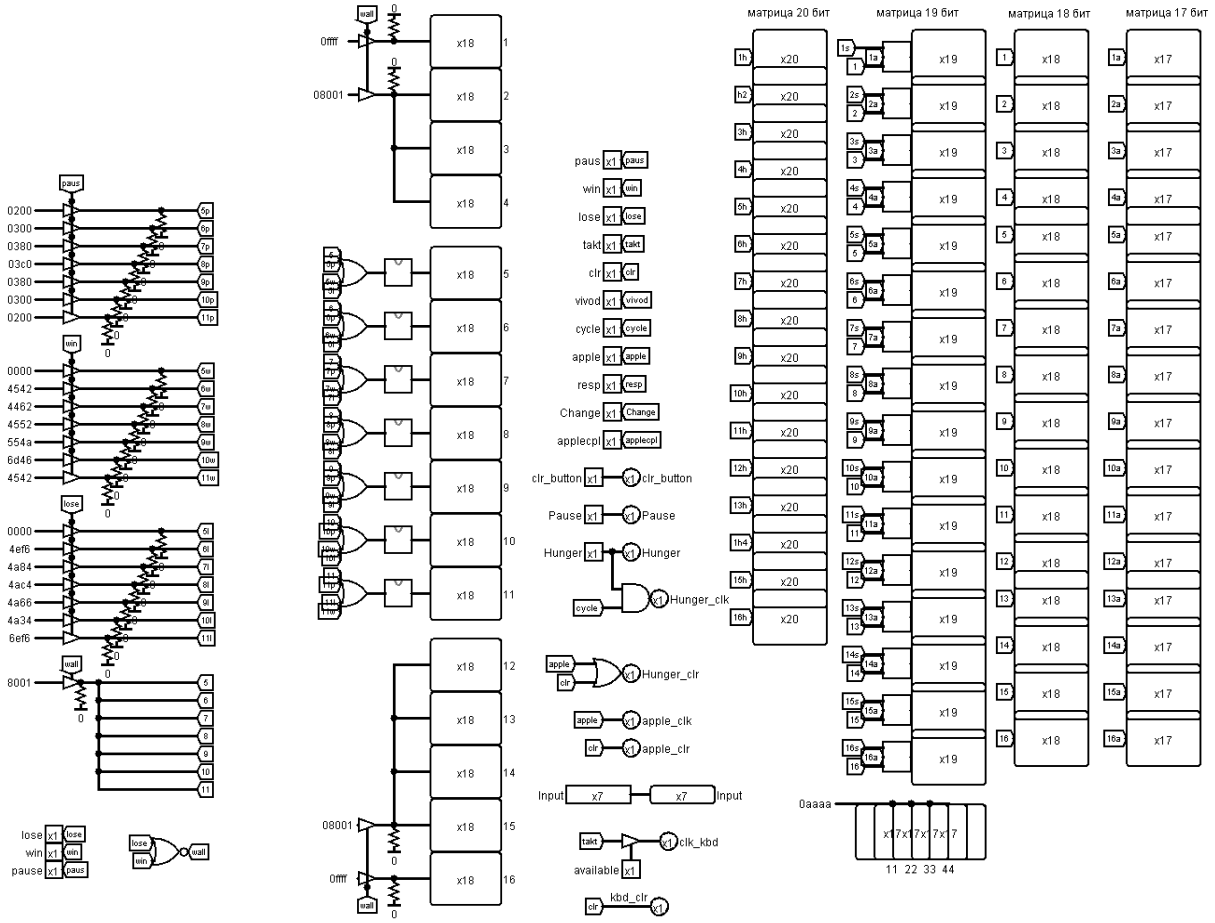

рис. 10.4 рис. 10.5

Вывод заставок на дисплей происходит исходя из схемы (рис. 10.4). В зависимости от того какой момент игры, по соответствующим каналам будут подаваться данные для вывода. Если игра на пузе - будет (рис. 10.2). Если игрок проиграл - будет (рис. 10.1). Если игрок выиграл - будет (рис. 10.3).

Так как нам нужен разный цвет вывода мы использовали 5 матриц. 1 матрица отвечала за светло-зеленое тело змеи. 2 матрица - красное яблоко. 3 матрица - желтые границы и надписи. 4 матрица - для черного фона. 5 матрица использовалась для темно-зеленой головы. Именно для

вывода используется схема (рис. 10.5)

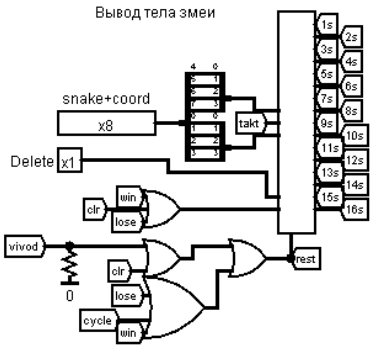

рис. 10.6

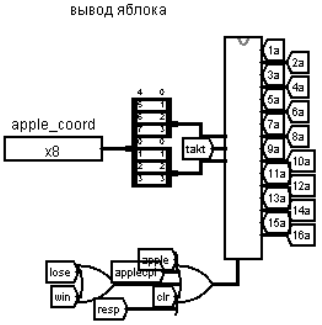

рис. 10.7

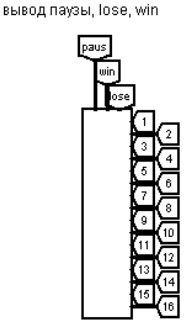

рис. 10.8

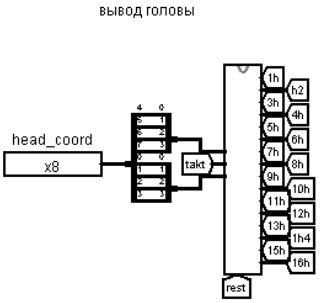

рис. 10.9

Схемы указанные выше отвечают за вывод в соответствующие выходы указанные в схеме (рис. 10.5).

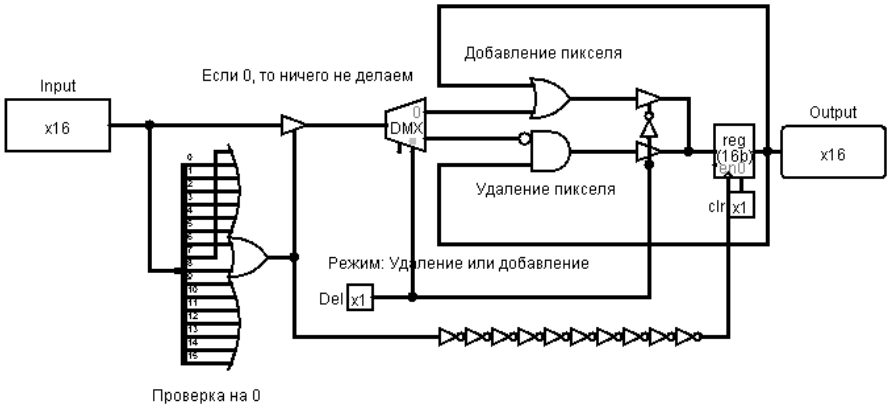

рис. 10.10

Схема (рис. 10.10) отвечает за срабатывание D-триггера. Как следствие за отображение пикселя змейки на дисплее. В зависимости от режима, будет происходить добавление или удаление пикселя.

7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Перед игроком есть:

- Дисплей, где будет происходить игровой процесс
- Левый счетчик “КОЛ-ВО ЯБЛОК” считает кол-во съеденных яблок
- Правый счетчик “ГОЛОД” ведет обратный отсчет до момента смерти змейки от голода
- Кнопка “RESTART”, отвечающая за перезапуск игры
- Контакт “ПАУЗА”, который ставит игру на паузу
- Контакт “ГОЛОД”, позволяющий активировать способность змейки голодать
- В нижней части находится клавиатура, предназначенная для управления змейкой

Запуск игры осуществляется с помощью запуска тактового генератора. После запуска в центре поля отобразится змейка длиной 2 пикселя и цель (красный пиксель). Пользователь

должен направить змейку в сторону цели, и *“съесть”* ее. После того как цель достигнута, счетчик Голода сбрасывается до изначального состояния и на поле появляется новая цель.

Для того чтобы управлять змейкой, нужно использовать 2 клавиши левого (a/ф) и правого направления (d/в). Стоит заметить, что движение змейки происходит относительно ее головы.

Чтобы победить, пользователь должен собрать 10 яблочек.

Пользователь может закончить игру проигрышем в нескольких случаях, а именно: при само перекусывании змейки, если змейка врезалась в границу игрового поля, если кончился обратный отсчет Голода.

При победе будет высвечиваться надпись WIN, при проигрыше LOSE. Чтобы перезапустить игру, нужно нажать на кнопку RESTART.

8 ВЫВОД

С помощью инструмента Logisim, процессора Cdm 8, и языка программирования CocoIDE мы разработали игру оригинальную игру *“Змейка”* с дополнением в виде *“Голода”*. Цель показать все возможности схемотехники и низкоуровневого языка программирования на деле - достигнута. Проект подходит для развлекательных целей.

9 ИСТОЧНИКИ

- *"Computing platforms", A. Shafarenko and S.P. Hunt, School of Computer Science University of Hertfordshire. 2015*
- *Random-access memory - Wikipedia*
- *Read-only memory - Wikipedia*