5.3. Контроллер динамической оперативной памяти DRAM

В данном разделе рассматривается разработка поведенческой и RTL-моделей контроллера DRAM, служащего для обеспечения связи процессоров с микросхемами динамической памяти [32]. На рис. 5.3 показано включение контроллера DRAM в цифровые электронные схемы.

Контроллер DRAM присоединяется к шине адреса и при выполнении операций записи/считывания осуществляет преобразование 8-битного адреса, передаваемого микропроцессором по шине *Addr8*, в два 4-битных адреса – номер строки и номер колонки в матрице микросхемы DRAM, которые затем в последовательном режиме передаются микросхеме памяти по шине *Addr4*. Кроме того, задача контроллера состоит в периодическом генерировании сигнала регенерации. В случае несвоевременного поступления этого сигнала память микросхемы DRAM будет стерта, что может привести к самым серьезным последствиям для обрабатываемой системой информации или управляемого оборудования.

Временная диаграмма функционирования контроллера при осуществлении операции считывания из DRAM приведена на рис. 5.4.

5.3.1. Поведенческий код

```
// ОПРЕДЕЛЕНИЯ КОНСТАНТ
`define DEL      1      // Задержка распространения сигнала в
                        // устройстве. Нулевая задержка может
```

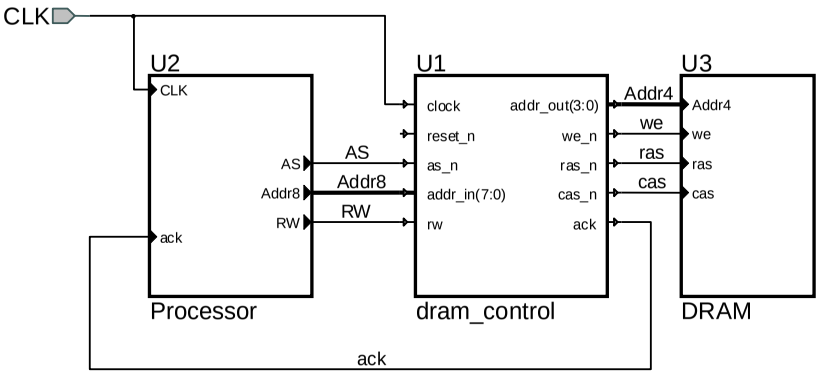

Рис. 5.3. Использование контроллера динамической памяти

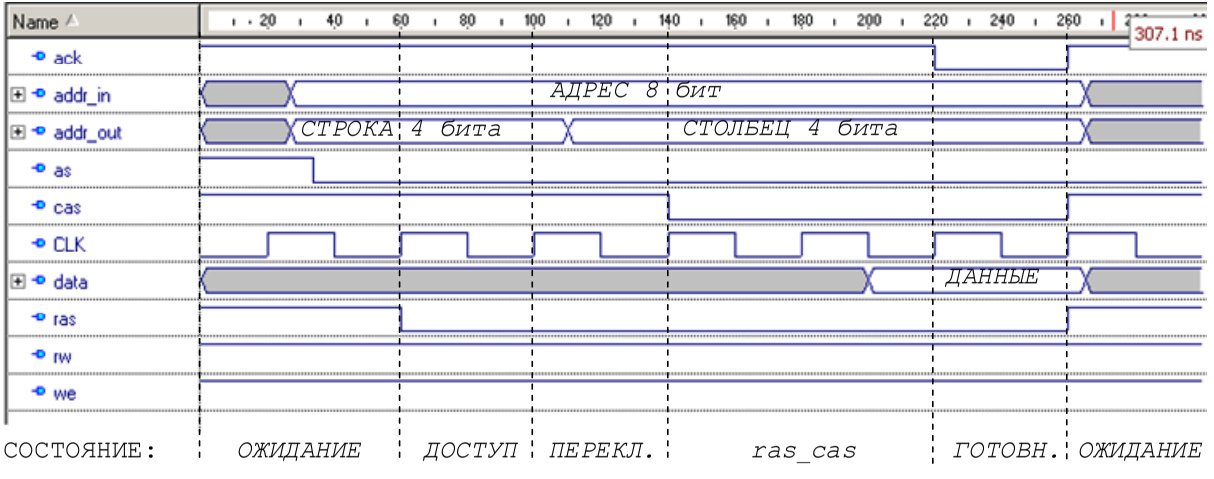

Рис. 5.4. Временная диаграмма функционирования контроллера при осуществлении операции считывания из DRAM

```
                                    // привести к проблемам
`define RBC_CYC 2          // Число тактов для удержания сигнала RAS
                           // перед установкой сигнала CAS
`define CBR_CYC 1          // Число тактов для удержания сигнала CAS
                                    // перед установкой сигнала RAS
`define RACW_CYC 1         // Число тактов для удержания сигналов
                                    // RAS и CAS вместе при записи
`define RACR_CYC 2         // Число тактов для удержания сигналов
                                    // RAS и CAS вместе при считывании
`define RACRF_CYC 1        // Число тактов для удержания сигналов
                                    // RAS и CAS вместе при регенерации
`define CNT_BITS 2         // Число тактов, необходимое счетчику
                           // для подсчитывания представленных
                                    // ниже сигналов
`define REF_cnt 24         // Число тактов между регенерациями
`define REF_BITS 5         // Число битов, необходимое счетчику
                           // для отсчитывания промежутка
                                    // между регенерациями
`define AOUT 4             // Разрядность шины адреса
                                    // со стороны DRAM
`define AIN 2*`AOUT        // Разрядность шины адреса
                                    // со стороны процессора

// ГЛАВНЫЙ МОДУЛЬ
module Dram_control(
        clock,
        reset_n,
        as_n,
        addr_in,
        addr_out,
        rw,
        we_n,
        ras_n,
        cas_n,
        ack);

// ВХОДЫ
input                      clock;  // Тактирующий сигнал для
                                          // конечных автоматов
```

```
input                    reset_n; // Срабатывающий по спаду
                                  // синхронный сигнал сброса reset
input                    as_n;    // Срабатывающий по спаду
                                  // строб адреса
input [`AIN-1:0]  addr_in;        // Адрес со стороны процессора
input                    rw;      // Сигнал выбора запись/считывания
                                  // = 1 соответствует считыванию
                                  // = 0 соответствует записи

// ВЫХОДЫ
output [`AOUT-1:0]addr_out;       // Адрес для DRAM
output                   we_n;            // Сигнал разрешения записи
output                   ras_n;   // Строб адреса строки для
                                  // памяти DRAM
output                   cas_n;   // Строб адреса столбца для
                                  // памяти DRAM
output                   ack;             // Сигнал подтверждения
                                  // для процессора

// ОБЪЯВЛЕНИЯ СИГНАЛОВ
wire                             clock;
wire                             reset_n;
wire [`AIN-1:0]   addr_in;
wire                             as_n;
wire                             rw;
wire                             we_n;
wire                             ras_n;
wire                             cas_n;
wire                             ack;
wire [`AOUT-1:0]         addr_out;

reg [3:0]          mem_state;     // Конечный автомат
wire                     col_out;         // Адрес колонки/строки
                                  // = 1 – выбор колонки
                                  // = 0 – выбор строки
reg [`CNT_BITS-1:0]      count;   // Счетчик тактов
reg        [`REF_BITS-1:0]       ref_count;       // Счетчик тактов при
                                                  // регенерации
```

```
reg                              refresh;// Запрос на регенерацию

// ПАРАМЕТРЫ
// Биты состояний представляют следующие сигналы:
//                 col_out, ras, cas, ack
parameter[3:0]                   // Состояния автомата
   IDLE          = 4'b0000,
   ACCESS        = 4'b0100,
   SWITCH        = 4'b1100,
   RAS_CAS       = 4'b1110,
   ACK           = 4'b1111,
   REF1          = 4'b0010,
   REF2          = 4'b0110;

// ОПЕРАТОРЫ ПОТОКОВ ДАННЫХ
// Пересылка сигналов на управляющие выходы из состояния автомата
assign col_out = mem_state[3];
assign ras_n = ~mem_state[2];
assign cas_n = ~mem_state[1];
assign ack = mem_state[0];

// Сброс сигнала we_n в течении регенерации
assign #`DEL we_n = rw | (mem_state == REF1) |
                                 (mem_state == REF2);

// Пересылка адресов столбца или строки на DRAM в зависимости от
// значения сигнала col_out
assign #`DEL addr_out = col_out? addr_in[`AOUT-1:0] :
                                          addr_in[`AIN-1:`AOUT];

// ОСНОВНОЙ КОД

// Отслеживание событий на сигнале reset
always @(reset_n) begin
   if (~reset_n) begin
          #`DEL assign mem_state = IDLE;
          assign count = `CNT_BITS'h0;
          assign ref_count = `REF_CNT;
```

```
            assign refresh = 1'b0;
    end
    else begin
            #`DEL;
            deassign mem_state;
            deassign count;
            deassign ref_count;
            deassign refresh;
    end
end

// Ожидание возрастания тактирующего сигнала
always @(posedge clock) begin
    // Наступило ли время запроса на регенерацию?
    if (ref_count == 0) begin
            refresh <= #`DEL 1'b1;
            ref_count <= #`DEL `REF_CNT;
    end
    else
            ref_count <= #`DEL ref_count – 1;

    // Обнуление счетчика тактов
    if (count)
            count <= #`DEL count – 1;

    case (mem_state)
            IDLE:  begin
            // Запрос на регенерацию имеет наибольший приоритет
                    if (refresh) begin
                            // Загрузка счетчика для отсчитывания времени
                            // удержания сигнала CAS
                            count <= #`DEL `CBR_CYC;
                            mem_state <= #`DEL REF1;
                    end
                    else if (~as_n) begin
                            // Загрузка счетчика для отсчитывания времени
                            // удержания сигнала RAS
                            count <= #`DEL `RBC_CYC;
                            mem_state <= #`DEL ACCESS;
```

```
                end
            end
            ACCESS:     begin
                mem_state <= #`DEL SWITCH;
            end
            SWITCH:     begin
                if (count == 0) begin
                    mem_state <= #`DEL RAS_CAS;
                    if (rw)
                        count <= #`DEL `RACR_CYC;
                    else
                        count <= #`DEL `RACW_CYC;
                end
            end
            RAS_CAS:begin
                if (count == 0) begin
                    mem_state <= #`DEL ACK;
                end
            end
            ACK:  begin
                mem_state <= #`DEL IDLE;
            end
            REF1:  begin
                if (count == 0) begin
                    mem_state <= #`DEL REF2;
                    count <= #`DEL `RACRF_CYC;
                end
            end
            REF2:  begin
                if (count == 0) begin
                    mem_state <= #`DEL IDLE;
                    refresh <= #`DEL 1'b0;
                end
            end
    endcase
end
endmodule                   // Конец модуля Dram_control
```

5.3.2. Код уровня регистровых передач

```
// ОПРЕДЕЛЕНИЯ КОНСТАНТ
`define DEL       1      // Задержка распространения сигнала в
                         // устройстве. Нулевая задержка может
                         // привести к проблемам
`define RBC_CYC 2        // Число тактов для удержания сигнала RAS
                         // перед установкой сигнала CAS
`define CBR_CYC 1        // Число тактов для удержания сигнала CAS
                         // перед установкой сигнала RAS
`define RACW_CYC 1       // Число тактов для удержания сигналов
                              // RAS и CAS вместе при записи
`define RACR_CYC 2       // Число тактов для удержания сигналов
                              // RAS и CAS вместе при считывании
`define RACRF_CYC 1      // Число тактов для удержания сигналов
                              // RAS и CAS вместе при регенерации
`define CNT_BITS 2       // Число тактов, необходимое счетчику
                         // для подсчитывания представленных
                              // ниже сигналов
`define REF_cnt 24       // Число тактов между регенерациями
`define REF_BITS 5       // Число битов, необходимое счетчику
                         // для отсчитывания промежутка
                              // между регенерациями
`define AOUT 4           // Разрядность шины адреса
                              // со стороны DRAM
`define AIN 2*`AOUT      // Разрядность шины адреса
                              // со стороны процессора

// ГЛАВНЫЙ МОДУЛЬ
module Dram_control(
        clock,
        reset_n,
        as_n,
        addr_in,
        addr_out,
        rw,
        we_n,
```

```
        ras_n,
        cas_n,
        ack);

// ВХОДЫ
input           clock;  // Тактирующий сигнал для
                        // конечных автоматов
input           reset_n;        // Срабатывающий по спаду
                        // синхронный сигнал сброса reset
input                   as_n;   // Срабатывающий по спаду
                                // строб адреса
input [`AIN-1:0]  addr_in;      // Адрес со стороны процессора
input                   rw;     // Сигнал выбора запись/считывания
                        // = 1 соответствует считыванию
                        // = 0 соответствует записи

// ВЫХОДЫ
output [`AOUT-1:0]addr_out;     // Адрес для DRAM
output                  we_n;           // Сигнал разрешения записи
output                  ras_n;  // Строб адреса строки для
                                // памяти DRAM
output                  cas_n;  // Строб адреса столбца для
                                // памяти DRAM
output                  ack;            // Сигнал подтверждения
                                // для процессора

// ОБЪЯВЛЕНИЯ СИГНАЛОВ
wire                            clock;
wire                            reset_n;
wire [`AIN-1:0]         addr_in;
wire                            as_n;
wire                            rw;
wire                            we_n;
wire                            ras_n;
wire                            cas_n;
wire                            ack;
wire [`AOUT-1:0]        addr_out;
```

```
reg [3:0]                   mem_state;      // Конечный автомат
wire                        col_out;        // Адрес колонки/строки
                                            // = 1 – выбор колонки
                                            // = 0 – выбор строки
reg [`CNT_BITS-1:0]     count;  // Счетчик тактов
reg     [`REF_BITS-1:0]     ref_count;      // Счетчик тактов при
                                            // регенерации
reg                         refresh;// Запрос на регенерацию

// ПАРАМЕТРЫ
// Биты состояний представляют следующие сигналы:
//              col_out, ras, cas, ack
parameter[3:0]              // Состояния автомата
  IDLE   = 4'b0000,
  ACCESS      = 4'b0100,
  SWITCH      = 4'b1100,
  RAS_CAS     = 4'b1110,
  ACK         = 4'b1111,
  REF1   = 4'b0010,
  REF2   = 4'b0110;

// ОПЕРАТОРЫ ПОТОКОВ ДАННЫХ
// Пересылка сигналов на управляющие выходы из состояния автомата
assign col_out = mem_state[3];
assign ras_n = ~mem_state[2];
assign cas_n = ~mem_state[1];
assign ack = mem_state[0];

// Сброс сигнала we_n в течении регенерации
assign #`DEL we_n = rw | (mem_state == REF1) |
                            (mem_state == REF2);

// Пересылка адресов столбца или строки на DRAM в зависимости от
// значения сигнала col_out
assign #`DEL addr_out = col_out? addr_in[`AOUT-1:0] :
                                    addr_in[`AIN-1:`AOUT];

// ОСНОВНОЙ КОД
```

```
// Отслеживание событий, управляющих переходом между
// состояниями
always @(posedge clock or negedge reset_n) begin
   if (~reset_n) begin
         mem_state <= #`DEL IDLE;
         count <= #`DEL `CNT_BITS'h0;
         ref_count <= #`DEL `REF_CNT;
         refresh <= #`DEL 1'b0;
   end
   else begin
   // Наступило ли время запроса на регенерацию?
         if (ref_count == 0) begin
               refresh <= #`DEL 1'b1;
               ref_count <= #`DEL `REF_CNT;
         end
         else
               ref_count <= #`DEL ref_count – 1;

   // Обнуление счетчика тактов
         if (count)
               count <= #`DEL count – 1;

         case (mem_state)
               IDLE:  begin
         // Запрос на регенерацию имеет наибольший приоритет
                     if (refresh) begin
                     // Загрузка счетчика для отсчитывания
                     // времени удержания сигнала CAS
                           count <= #`DEL `CBR_CYC;
                           mem_state <= #`DEL REF1;
                     end
                     else if (~as_n) begin
                     // Загрузка счетчика для отсчитывания
                     // времени удержания сигнала RAS
                           count <= #`DEL `RBC_CYC;
                           mem_state <= #`DEL ACCESS;
                     end
               end
               ACCESS:      begin
```

```
                    mem_state <= #`DEL SWITCH;
                end
                SWITCH:     begin
                    if (count == 0) begin
                        mem_state <= #`DEL RAS_CAS;
                        if (rw)
                            count <= #`DEL `RACR_CYC;
                        else
                            count <= #`DEL `RACW_CYC;
                    end
                end
                RAS_CAS:begin
                    if (count == 0) begin
                        mem_state <= #`DEL ACK;
                    end
                end
                ACK:    begin
                    mem_state <= #`DEL IDLE;
                end
                REF1:   begin
                    if (count == 0) begin
                        mem_state <= #`DEL REF2;
                        count <= #`DEL `RACRF_CYC;
                    end
                end
                REF2:   begin
                    if (count == 0) begin
                        mem_state <= #`DEL IDLE;
                        refresh <= #`DEL 1'b0;
                    end
                end
            endcase
    end
end
endmodule               // Конец модуля Dram_control
```

5.3.3. Испытательный стенд

```
// DEFINE S
`define DEL      1      // Задержка распространения сигнала в
                        // устройстве. Нулевая задержка может
                                // привести к проблемам
`define ACC_COUNT 4  // Максимальное число тактов,
                                // необходимое для доступа к памяти
`define CLK_HPERIOD 50       // Полупериод тактового сигнала
`define CLK_PERIOD CLK_HPERIOD*2//Период тактового сигнала
`define WR_PERIOD IOD70      // Период удержания сигнала CAS,
                                        // необходимый при записи в память
`define RD_PERIOD 120        // Период удержания сигнала CAS,
                                        // необходимый при считывании
`define AOUT 4               // Разрядность шины адреса
                                        // со стороны DRAM
`define AIN 2*`AOUT          // Разрядность шины адреса
                                        // со стороны процессора
`define DWIDTH 8                        // Разрядность шины данных DRAM
`define DDEPTH 256           // Объем DRAM

// ОСНОВНОЙ КОД
module Dramcon_sim();

// ОБЪЯВЛЕНИЯ СИГНАЛОВ
reg                             clock;
reg                             reset;
reg [`AIN-1:0]    addr_in;
reg                             as;
reg                             rw;
wire                            we_n;
wire                            ras_n;
wire                            cas_n;
wire                            ack;
wire [`AOUT-1:0]          addr_out;

reg       [`DWIDTH-1:0]       data_out;       // Выход данных от
                                                // процессора
```

```
wire [`DWIDTH-1:0]    data;          // Шина данных
reg       [`AIN:0]          addr_count; // Счетчик адреса
reg [`DWIDTH-1:0]     data_patt;     // Образец данных
integer                     cyc_count;     // Число тактов,
// определяющее превышение допустимого времени доступа

// ОПЕРАТОРЫ ПОТОКОВ ДАННЫХ
assign data = (as & we_n) ? 8'hzz : data_out;

// ОСНОВНОЙ КОД

// Включение тестируемого устройства – контроллера DRAM
Dram_control Dram_control(
        .clock(clock),
        .reset_n(~reset),
        .as_n(~as),
        .addr_in(addr_in),
        .addr_out(addr_out),
        .rw(rw),
        .we_n(we_n),
        .ras_n(ras_n),
        .cas_n(cas_n),
        .ack(ack));

// Включение модели DRAM
dram Dram(
        .ras(~ras_n),
        .cas(~cas_n),
        .we(~we_n),
        .address(addr_out),
        .data(data));

// Генератор тактирующего сигнала
always #`CLK_HPERIOD clock = ~clock;

// Моделирование
initial begin
```

```
// Инициализация входов
   clock = 1;
   reset = 0;
   as = 0;

   // Проверка работы сигнала reset
   #`DEL;
   // Установка сигнала reset
   reset = 1;

   // Ожидание асинхронного изменения выхода
   #`DEL
   #`DEL
   // Тестирование выходов
   if ((ras_n === 1'b1) && (cas_n === 1'b1) &&
            (ack === 1'b0))
            $display ("Reset работает");
   else begin
   $display("\nОШИБКА в момент времени %0t:", $time);
            $display("Reset не работает");
            $display(" ras_n = %b", ras_n);
            $display(" cas_n = %b", cas_n);
            $display(" ack = %b\n", ack);

                     // Использование $stop для отладки
            $stop;
   end

   // Сброс сигнала reset
   reset = 0;

   // Инициализация счетчика адресов
   addr_count = 0;

   // Инициализация образца данных для записи
   data_patt = `DWIDTH'h1;

   // Засылка последовательности значений в память
```

```
while (addr_count[`AIN] === 0) begin
        // Запись в память
        writetem(addr_count[`AIN-1:0], data_patt);

        // Увеличение счетчика адресов
        addr_count <= addr_count + 1;

        // Сдвиг образца данных
        data_patt <= (data_patt << 1);
        data_patt[0] <= data_patt[`DWIDTH-1];

        // Ожидание изменения образца данных
        #`DEL;
  end

  // Инициализация счетчика адресов
  addr_count = 0;

  // Инициализация образца данных для считывания
  data_patt = `DWIDTH'h1;

  // Проверка значений, которые были записаны
  while (addr_count[`AIN] === 0) begin
        // Чтение из памяти
        readmem(addr_count[`AIN-1:0], data_patt);

        // Увеличение счетчика адресов
        addr_count <= addr_count + 1;

        // Сдвиг образца данных
        data_patt <= (data_patt << 1);
        data_patt[0] <= data_patt[`DWIDTH-1];

        // Ожидание изменения образца данных
        #`DEL;
  end
  $display("\nМоделирование завершено без ошибок\n");
  $finish;
end
```

```
// Проверка числа тактов, необходимых для всех
// видов доступа к памяти
always @(posedge clock) begin
   // Проверка, не было ли время доступа слишком длинным
   if (cyc_count > 3*`ACC_COUNT) begin
   $display("\nОШИБКА в момент времени %0t:", $time);
            if (rw)
            $display("Время считывания слишком большое\n");
            else
                     $display("Время записи слишком большое\n");

                     // Использование $stop для отладки
            $stop;
   end
end

// СЦЕНАРИИ
// Запись данных в память
task writemem;

// INPUTS
input [`AIN-1:0]  write_addr;      // Адрес в памяти
input [`DWIDTH-1:0]     write_data;// Записываемые данные

// КОД СЦЕНАРИЯ
begin
   cyc_count = 0;            // Инициализация счетчика тактов

   // Ожидание нарастания тактирующего сигнала
   @(posedge clock);
   rw <= #`DEL 0;                              // Команда на запись
   addr_in <= #`DEL write_addr;          // Установка адреса
data_out <= #`DEL write_data; // Установка
                                               // записываемых данных
   as <= #`DEL2 1;                    // Установка строба адреса

   // Ожидание подтверждения
   @(posedge clock);
```

```
while (~ack) begin
        // Увеличение счетчика тактов
        cyc_count = cyc_count + 1;

        @(posedge clock);
  end

  as <= #`DEL 0;                    // Сброс строба адреса
  @(posedge clock);          // Ожидание 1 такта

end
endtask          // Конец сценария writetemt

// Считывание данных из памяти и проверка их правильности
task readmem;

// INPUTS
input [`AIN-1:0]  read_addr;          // Адрес в памяти
input [`DWIDTH-1:0]    exp_data;      // Ожидаемый ответ

// КОД СЦЕНАРИЯ
begin
  cyc_count = 0;          // Инициализация счетчика тактов

  // Ожидание нарастания тактирующего сигнала
  @(posedge clock);
  rw <= #`DEL 1;                    // Команда на считывание
  addr_in <= #`DEL read_addr; // Установка адреса
  as <= #`DEL2 1;                   // Установка строба адреса

  // Ожидание подтверждения
  @(posedge clock);
  while (~ack) begin
        // Увеличение счетчика тактов
        cyc_count = cyc_count + 1;

        @(posedge clock);
  end
```

```
// Удалось ли получить ожидаемые данные?
   if (data !== exp_data) begin
   $display("\nОШИБКА в момент времени %0t:", $time);
            $display("Контроллер не работает");
            $display("записанные данные = %h", exp_data);
            $display("считанные данные = %h\n", data);

                     // Использование $stop для отладки
            $stop;
   end

   as <= #`DEL 0;                  // Сброс строба адреса
   @(posedge clock);               // Ожидание 1 такта

end
endtask            // Конец сценария readmem

endmodule                          // Конец модуля Dramcon_sim

// ПОДЧИНЕННЫЙ МОДУЛЬ
// Модель памяти DRAM
module Dram(
            ras,
            cas,
            we,
            address,
            data);

// ВХОДЫ
input                              ras;          // Строб строки адреса
input                              cas;          // Строб столбца адреса
input                              we;           // Разрешение записи
input [`AOUT-1:0]          address;              // Шина адреса DRAM

// ДВУНАПРАВЛЕННЫЕ ПОРТЫ
inout [`DWIDTH-1:0]        data;          // Шина данных

// ОБЪЯВЛЕНИЯ СИГНАЛОВ
```

```
wire                    ras;
wire                    cas;
wire                    we;
wire [`DWIDTH-1:0] data;
reg [`DWIDTH-1:0] mem[`DDEPTH-1:0];// Хранимые данные
reg      [`AIN-1:0]     mem_addr;          // Адрес памяти

time            rd_time;        // Время последнего изменения входа rd
wire [63:0]     rd_dtime; // Время rd_time,
        // задержанное на `RD_PERIOD шагов моделирования

time            wr_time;        // Время последнего изменения входа wr
wire [63:0]     wr_dtime;       // Время wr_dtime,
        // задержанное на `RD_PERIOD шагов моделирования

reg             oen;            // Внутреннее разрешение вывода –
// этот сигнал устанавливается в 1 после минимальной задержки CAS
// при чтении
reg                     wen;    // Внутреннее разрешение записи –
// этот сигнал устанавливается в 1 после минимальной задержки CAS
// при записи


// ОПЕРАТОРЫ ПОТОКОВ ДАННЫХ
assign #`RD_PERIOD rd_dtime = rd_time;
assign #`WR_PERIOD IODwr_dtime = wr_time;
assign data = (oen & ~we) ? mem[address] : 8 'hzz;

// ОСНОВНОЙ КОД
initial begin
 oen = 0;
 wen = 0;
 wr_time = 0;
 rd_time = 0;
end

// Ожидание возрастания сигнала RAS (поступления адреса строки)
always @(posedge ras) begin
```

```
    mem_addr[`AIN-1:`AOUT] <= #`DEL address;
  end

  // Ожидание возрастания сигнала RAS (поступления адреса столбца)
  always @(posedge cas) begin
    mem_addr[`AOUT-1:0] <= #`DEL address;
  end

  // Ожидание начала процесса записи/считывания
  always @(posedge cas) begin
    if (we) begin
                                        // Запись
        wr_time = $time;        // Сохранение времени
                                // последнего возрастания сигнала wr
        wen = 0;                        // Сброс внутреннего
                                        // разрешения записи
    end
    else begin
                                        // Чтение
        rd_time = $time;        // Сохранение времени
                        // последнего возрастания сигнала rd

        oen = 0;          // Сброс внутреннего разрешения считывания
    end
  end

  always @(wr_dtime) begin
    if (wr_time === wr_dtime) begin
        wen = 1;
    end
  end

  always @(rd_dtime) begin
    if (rd_time === rd_dtime) begin
        oen = 1;
    end
  end

  always @(negedge cas) begin
```

```
    if (wen & we) begin
            mem[mem_addr] <= #`DEL data;
    end
    wen = 0;        // Отключение внутреннего разрешения записи
    oen = 0;        // Отключение внутреннего разрешения считывания
end

endmodule                   // Конец модуля Dram
```