EE 402 VLSI SYSTEM DESIGN II

Lab Report for Lab #2

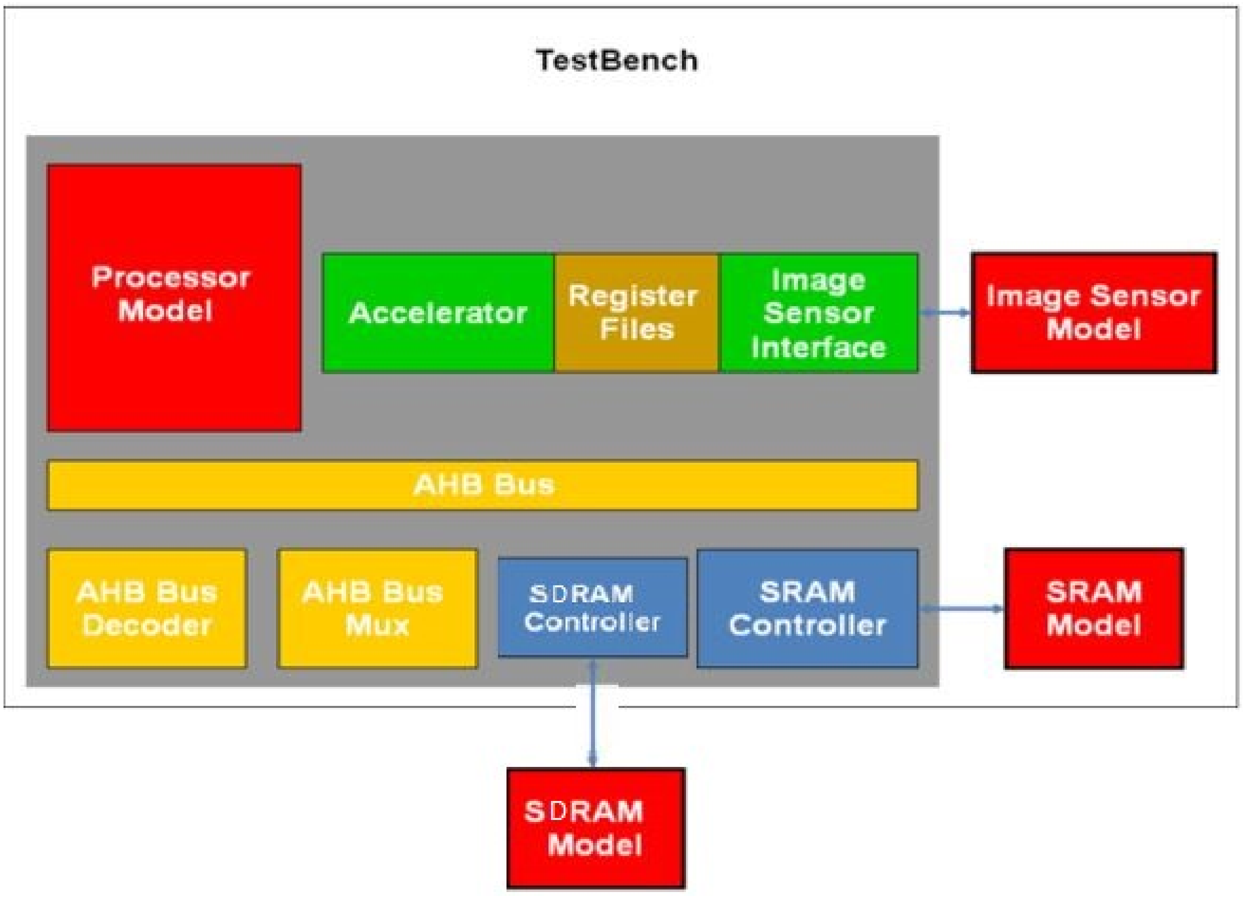
EFE ÖZTABAN – 25202
efeoztaban@sabanciuniv.edu

KAYRA BİLGİN – 25117
kayrabilgin@sabanciuniv.edu

1. Introduction

In this laboratory assignment, a System-on-Chip ASIC prototype is given as the Verilog RTL implementation. This prototype should be changed with the new requirements. These requirements are adding a SDRAM controller and SDRAM module. Processor should use the SDRAM to save the results of its calculations. It should write the results to the SDRAM and read the results from SDRAM to compare them with the results of accelerator.

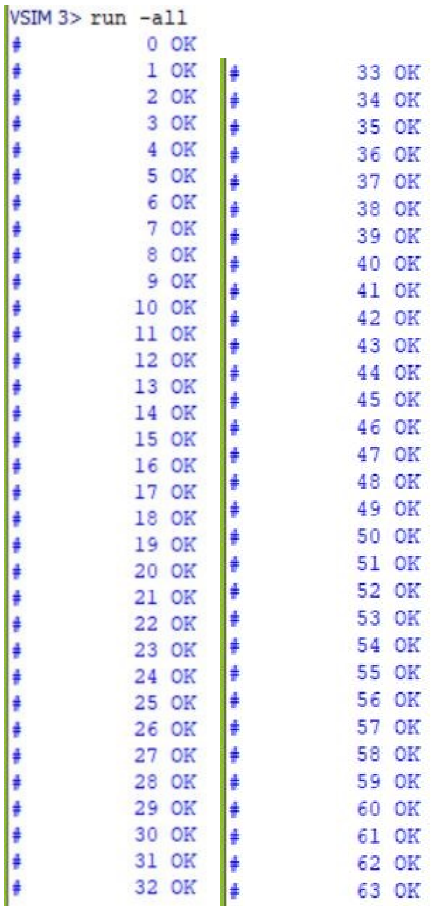
The SoC ASIC implementation is simulated with Mentor Graphics QuestaSim and Xilinx ISE in the laboratory assignment. The simulation results are given in the related section of the report. In addition to that, modified and added modules are briefly explained in the report.



2. Simulation Results

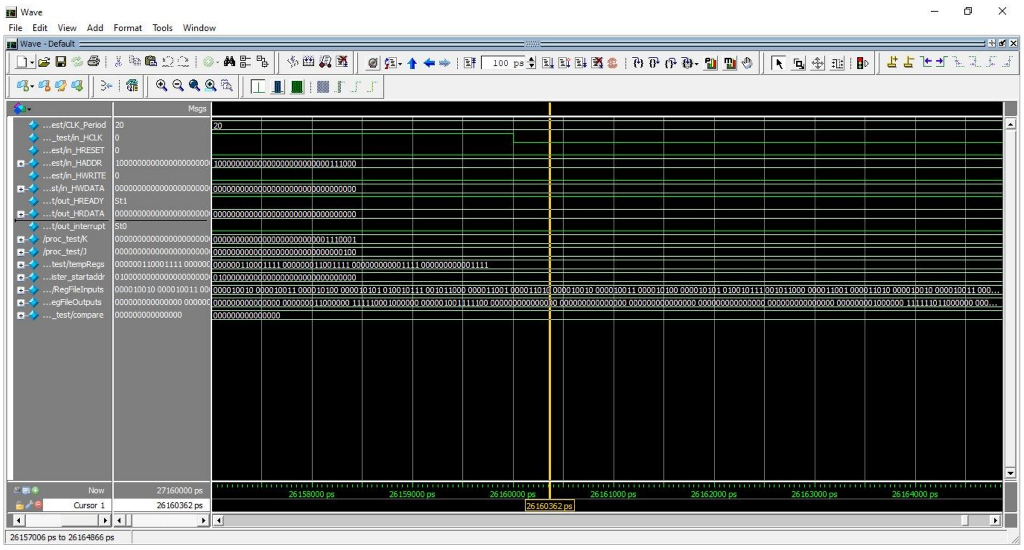
The simulations for the given SoC ASIC are performed with Mentor Graphics QuestaSim and Xilinx ISE.

It can be seen that the simulation is runed correctly from the results below:

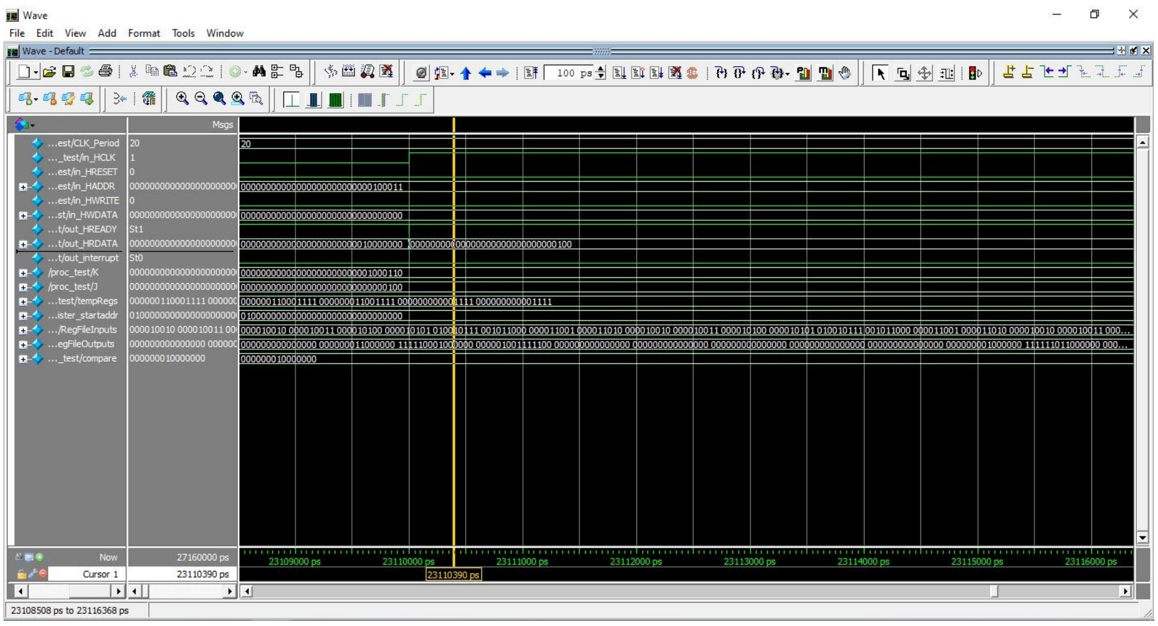


```
VSIM 3> run -all
#            0 OK
#            1 OK     #            33 OK
#            2 OK     #            34 OK
#            3 OK     #            35 OK
#            4 OK     #            36 OK
#            5 OK     #            37 OK
#            6 OK     #            38 OK
#            7 OK     #            39 OK
#            8 OK     #            40 OK
#            9 OK     #            41 OK
#           10 OK     #            42 OK
#           11 OK     #            43 OK
#           12 OK     #            44 OK
#           13 OK     #            45 OK
#           14 OK     #            46 OK
#           15 OK     #            47 OK
#           16 OK     #            48 OK
#           17 OK     #            49 OK
#           18 OK     #            50 OK
#           19 OK     #            51 OK
#           20 OK     #            52 OK
#           21 OK     #            53 OK
#           22 OK     #            54 OK
#           23 OK     #            55 OK
#           24 OK     #            56 OK
#           25 OK     #            57 OK
#           26 OK     #            58 OK
#           27 OK     #            59 OK
#           28 OK     #            60 OK
#           29 OK     #            61 OK
#           30 OK     #            62 OK
#           31 OK     #            63 OK
#           32 OK
```

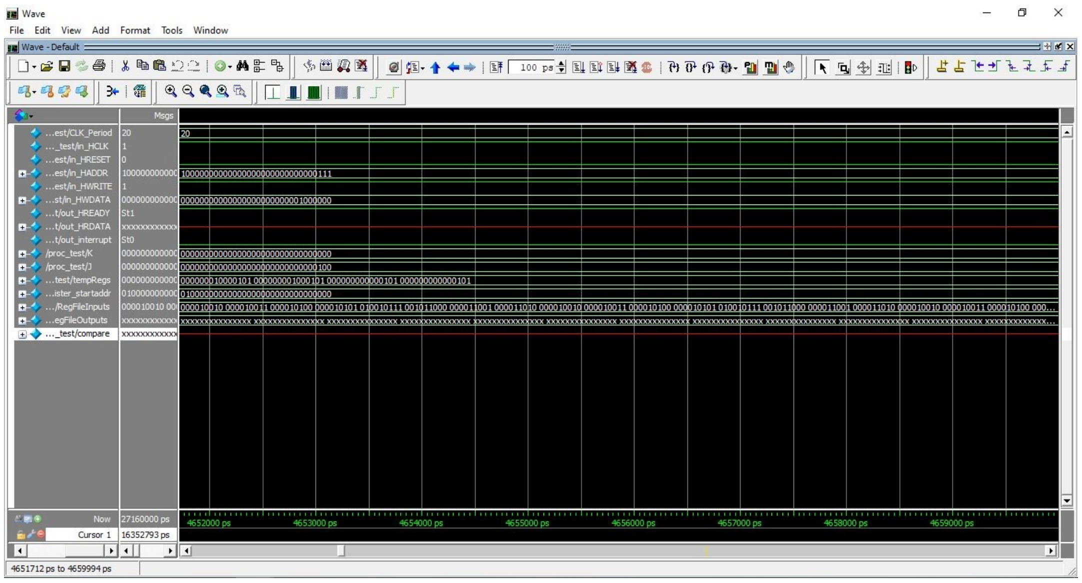
The simulation results from the Mentor Graphics QuestaSim are given with waveform below:



```
...est/CLK_Period    20
..._test/in_HCLK     0
...est/in_HRESET     0
...est/in_HADDR      100000000000000000000  10000000000000000000000000111000
...est/in_HWRITE     0
...st/in_HWDATA      000000000000000000000  00000000000000000000000000000000
...t/out_HREADY      St1
...t/out_HRDATA      000000000000000000000  00000000000000000000000000000000
...t/out_interrupt   St0
/proc_test/K         000000000000000000000  00000000000000000000000001110001
/proc_test/J         000000000000000000000  00000000000000000000000000000100
...test/tempRegs     0000001100011111 000000  0000001100011111 0000001100111 1 0000000000001111 0000000000001111
...ister_startaddr   010000000000000000000  01000000000000000000000000000000
.../RegFileInputs    000010010 000010011 00  000010010 000010011 000010100 000010101 010010111 001011000 000011001 ...
...egFileOutputs     000000000000000 000000  000000000000000 00000001100000 0 1111100010000000 000001001111100 ...
..._test/compare     000000000000000  000000000000000

Now       27160000 ps    26158000 ps  26159000 ps  26160000 ps  26161000 ps  26162000 ps  26163000 ps  26164000 ps
Cursor 1  26160362 ps    26160362 ps
26157006 ps to 26164866 ps
```

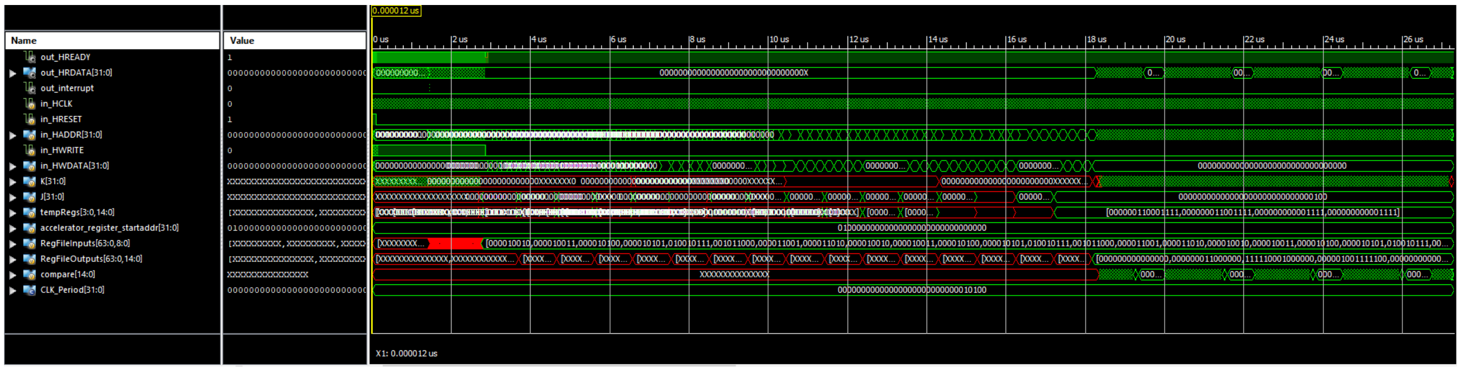


```
Wave - Default                         Msgs
...est/CLK_Period   20                      20
..._test/in_HCLK    1
...est/in_HRESET    0
...est/in_HADDR     000000000000000000000   00000000000000000000000000100011
...est/in_HWRITE    0
...st/in_HWDATA     000000000000000000000   00000000000000000000000000000000
...t/out_HREADY     St1
...t/out_HRDATA     000000000000000000000   00000000000000000000000010000000  00000000000000000000000000000100
...t/out_interrupt  St0
/proc_test/K        000000000000000000000   00000000000000000000000001000110
/proc_test/J        000000000000000000000   00000000000000000000000000000100
...test/tempRegs    000000110001111 000000  000000110001111 000000011001111 000000000001111 000000000001111
...ister_startaddr  010000000000000000000   01000000000000000000000000000000
.../RegFileInputs   000010010 000010011 00  000010010 000010011 000010100 000010101 010010111 001011000 000011001 000011010 000010010 000010011 000010100 000010101 010010111 001011000 000011001 000011010 000010010 000010011 000...
...egFileOutputs    000000000000000 000000  000000000000000 000000011000000 111110001000000 000001001111100 000000000000000 000000000000000 000000000000000 000000000000000 000000000000000 000000001000000 111110110000000 000...
..._test/compare    000000010000000         000000010000000

Now       27160000 ps    23109000 ps  23110000 ps  23111000 ps  23112000 ps  23113000 ps  23114000 ps  23115000 ps  23116000 ps
Cursor 1  23110390 ps    23110390 ps
23108508 ps to 23116368 ps
```



```
Wave - Default                         Msgs
...est/CLK_Period   20            20
..._test/in_HCLK    1
...est/in_HRESET    0
...est/in_HADDR     100000000000  10000000000000000000000000000111
...est/in_HWRITE    1
...st/in_HWDATA     000000000000  00000000000000000000000001000000
...t/out_HREADY     St1
...t/out_HRDATA     xxxxxxxxxxxx
...t/out_interrupt  St0
/proc_test/K        000000000000  00000000000000000000000000000000
/proc_test/J        000000000000  00000000000000000000000000000100
...test/tempRegs    000000000000  000000010000101 000000001000101 000000000000101 000000000000101
...ister_startaddr  010000000000  01000000000000000000000000000000
.../RegFileInputs   000010010 00  000010010 000010011 000010100 000010101 010010111 001011000 000011001 000011010 000010010 000010011 000010100 000010101 010010111 001011000 000011001 000011010 000010010 000010011 000010100 000...
...egFileOutputs    xxxxxxxxxxxx  xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxx...
..._test/compare    xxxxxxxxxxxx

Now       27160000 ps    4652000 ps  4653000 ps  4654000 ps  4655000 ps  4656000 ps  4657000 ps  4658000 ps  4659000 ps
Cursor 1  16352793 ps
4651712 ps to 4659994 ps
```

Also, the simulation result of the design in Xilinx ISE is given below:



```
Name                                      Value
out_HREADY                                1
out_HRDATA[31:0]                          0000000000000000000000000
out_interrupt                             0
in_HCLK                                   0
in_HRESET                                 1
in_HADDR[31:0]                            0000000000000000000000000
in_HWRITE                                 0
in_HWDATA[31:0]                           0000000000000000000000000
K[31:0]                                   xxxxxxxxxxxxxxxxxxxxxxxxx
J[31:0]                                   xxxxxxxxxxxxxxxxxxxxxxxxx
tempRegs[3:0,14:0]                        [xxxxxxxxxxxxxxx, xxxxxxxx
accelerator_register_startaddr[31:0]      0100000000000000000000000
RegFileInputs[63:0,8:0]                   [xxxxxxxxx, xxxxxxxxx, xxxxx
RegFileOutputs[63:0,14:0]                 [xxxxxxxxxxxxxxx, xxxxxxxx
compare[14:0]                             xxxxxxxxxxxxxxx
CLK_Period[31:0]                          0000000000000000000000000

0.000012 us    0 us  2 us  4 us  6 us  8 us  10 us  12 us  14 us  16 us  18 us  20 us  22 us  24 us  26 us
X1: 0.000012 us
```

3. Modified Modules

Processor Test

This is a test model to simulate processor. New parts are added in this module. After the calculations are done by the processor, it sends the results to the SDRAM through SDRAM controller. Then it reads the processor results from SDRAM and accelerator results from the SRAM. It compares these results and display a message about this comparison.

Project Model

In this module, new designed SDRAM model is initialized. SDRAM model is defined in this part with its interconnections. These interconnections are with forward transform SoC and processor.

```
sdram_model sdram_model(   //added
.in_clk(in_HCLK),
.in_CS(cs),
.in_WE(we),
.in_RAS(ras),
.in_CAS(cas),
.in_addr(SDRAM_addr),
.in_bank_addr(SDRAM_bank_addr),
.in_write_data(SDRAM_w_data),
.out_read_data(SDRAM_r_data)
);
```

Forward Transform SoC

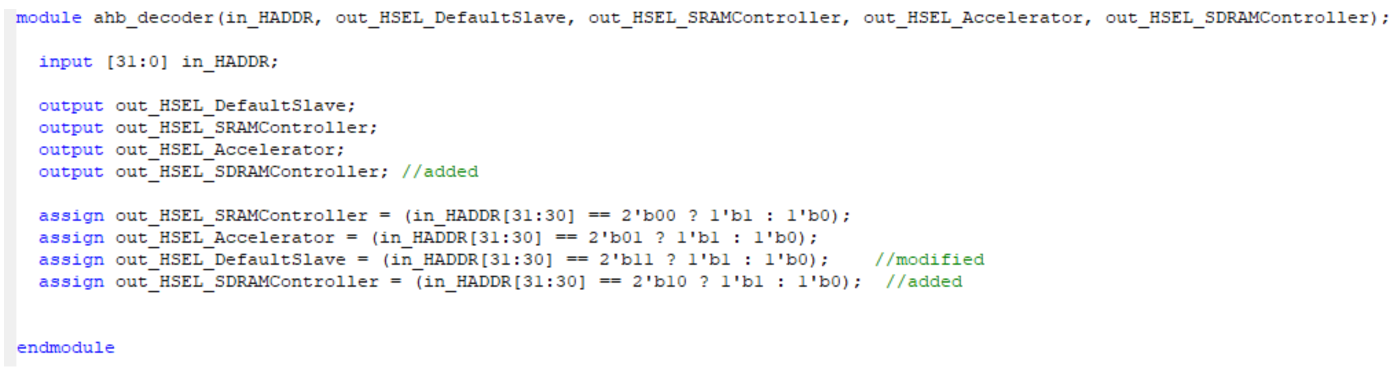
In this module, new designed SDRAM controller is initialized. SDRAM model is defined in this part with its interconnections. These interconnections are with project model, AHB decoder and AHB mux.

```
sdram_controller sdram_controller(    //added
  .in_HCLK(in_HCLK),
  .in_HRESET(in_HRESET),
  .in_HWRITE(in_HWRITE),
  .in_HSEL(SDRAM_decoder_HSEL),
  .in_HADDR(in_HADDR),
  .in_read_data(in_SDRAM_r_DATA),
  .out_HREADY(SDRAM_mux_HREADY),
  .out_HRDATA(SDRAM_mux_HRDATA),
  .in_HWDATA(in_HWDATA),
  .out_CS(out_cs),
  .out_WE(out_we),
  .out_RAS(out_ras),
  .out_CAS(out_cas),
  .out_addr(out_SDRAM_addr),
  .out_bank_addr(out_SDRAM_bank_addr),
  .out_write_data(out_SDRAM_w_DATA)
);
```

AHB Decoder

In this module, the memory map is defined for decoder to choose the proper slave from the memory map. Because a new slave is added in the AHB, memory map should change. This new slave is the SDRAM controller and its location in the memory map is between 32’b100…0 and 32’b110…0.

New implementation of AHB decoder is given below:

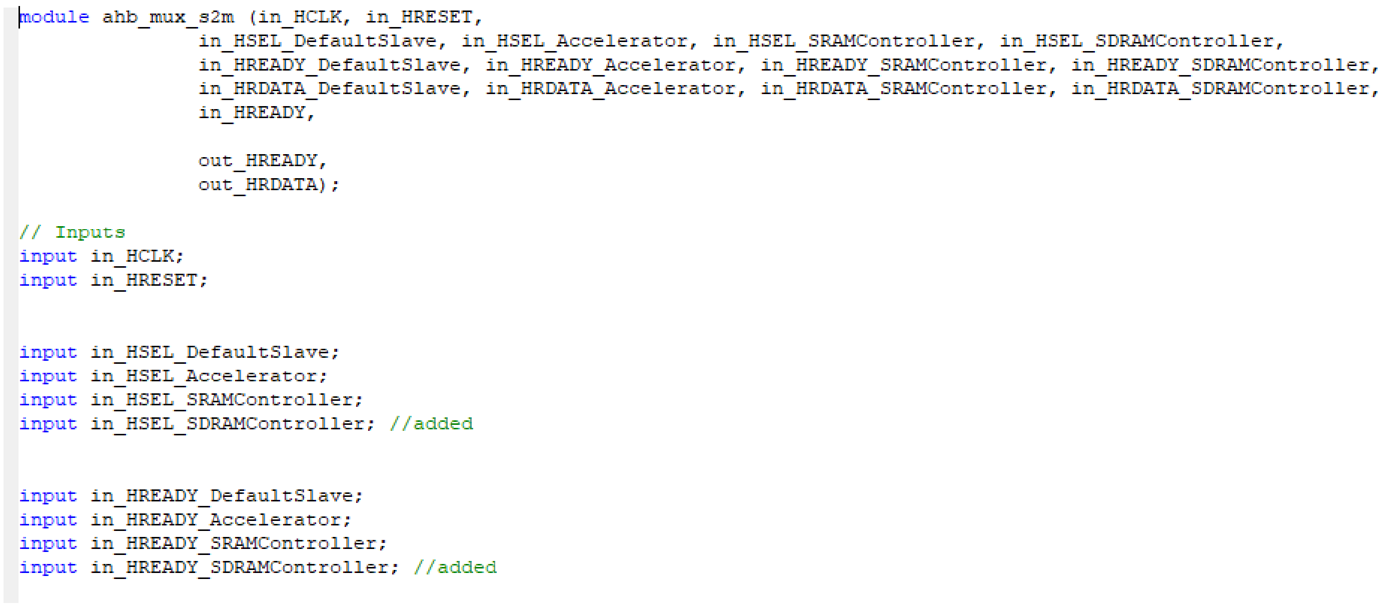


```verilog
module ahb_decoder(in_HADDR, out_HSEL_DefaultSlave, out_HSEL_SRAMController, out_HSEL_Accelerator, out_HSEL_SDRAMController);

  input [31:0] in_HADDR;

  output out_HSEL_DefaultSlave;
  output out_HSEL_SRAMController;
  output out_HSEL_Accelerator;
  output out_HSEL_SDRAMController; //added

  assign out_HSEL_SRAMController = (in_HADDR[31:30] == 2'b00 ? 1'b1 : 1'b0);
  assign out_HSEL_Accelerator = (in_HADDR[31:30] == 2'b01 ? 1'b1 : 1'b0);
  assign out_HSEL_DefaultSlave = (in_HADDR[31:30] == 2'b11 ? 1'b1 : 1'b0);    //modified
  assign out_HSEL_SDRAMController = (in_HADDR[31:30] == 2'b10 ? 1'b1 : 1'b0);  //added


endmodule
```

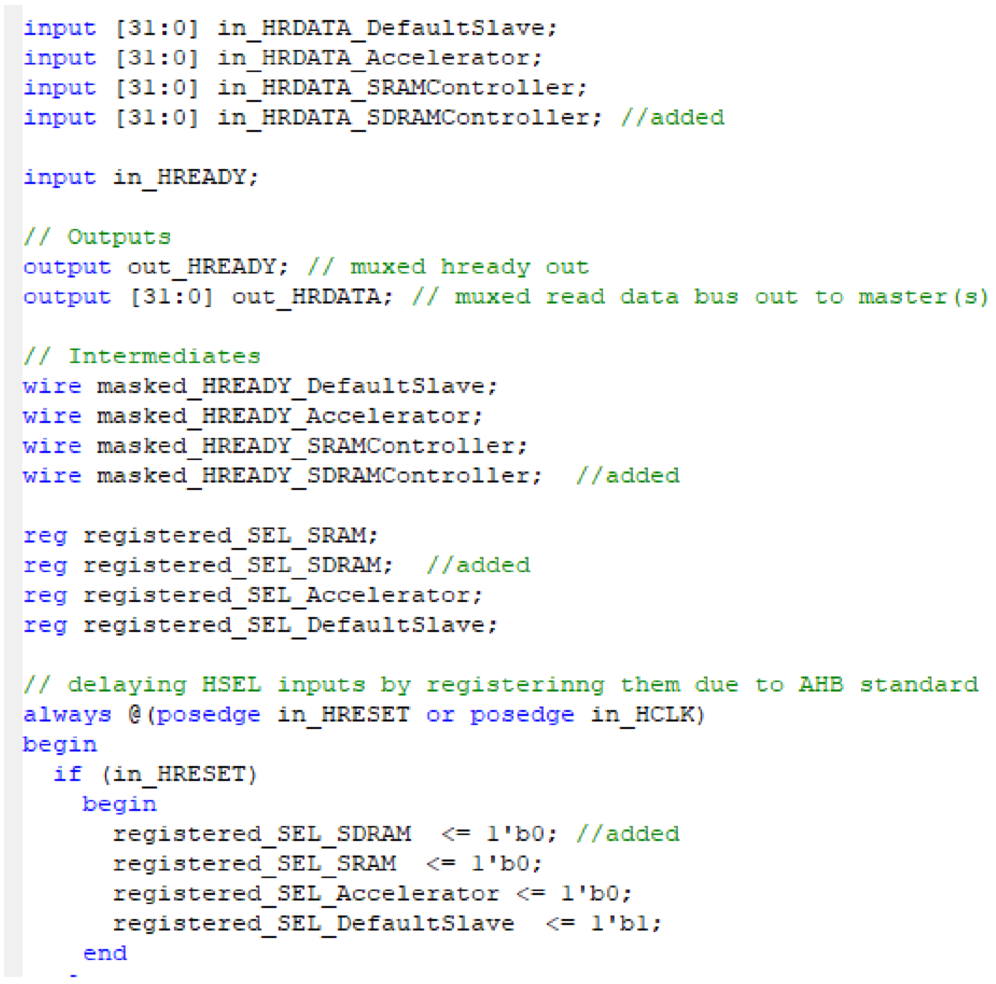
AHB Mux

In this module, a MUX is selecting the proper select, ready and data signals to be transmitted through the bus. It receives all the signals from all the slaves and gives the signals of the selected slave on the communication bus. Because a new slave is added on the bus, this module should be modified. This new slave is the SDRAM controller. In the inputs of the MUX, signals of the SDRAM controller are added. Now, it is choosing the slave to transmit information on the bus among the slaves which are Accelerator, SRAM controller, Default Slave and SDRAM controller.

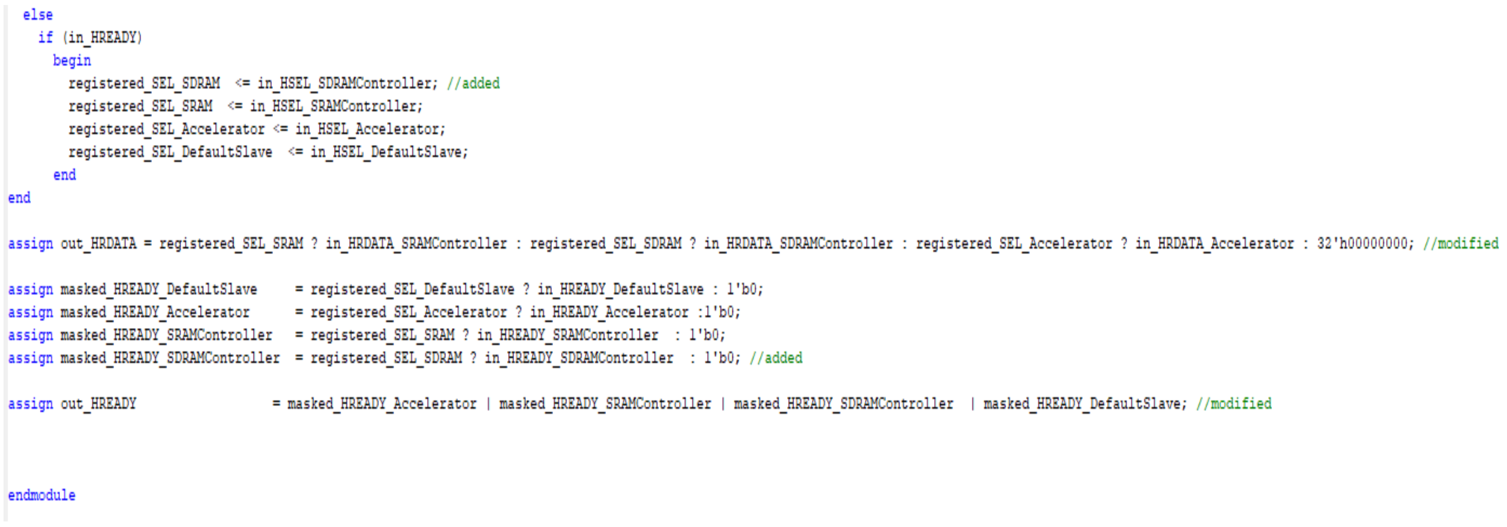
New implementation of AHB mux is given below:



```verilog
module ahb_mux_s2m (in_HCLK, in_HRESET,
               in_HSEL_DefaultSlave, in_HSEL_Accelerator, in_HSEL_SRAMController, in_HSEL_SDRAMController,
               in_HREADY_DefaultSlave, in_HREADY_Accelerator, in_HREADY_SRAMController, in_HREADY_SDRAMController,
               in_HRDATA_DefaultSlave, in_HRDATA_Accelerator, in_HRDATA_SRAMController, in_HRDATA_SDRAMController,
               in_HREADY,

               out_HREADY,
               out_HRDATA);

// Inputs
input in_HCLK;
input in_HRESET;


input in_HSEL_DefaultSlave;
input in_HSEL_Accelerator;
input in_HSEL_SRAMController;
input in_HSEL_SDRAMController; //added


input in_HREADY_DefaultSlave;
input in_HREADY_Accelerator;
input in_HREADY_SRAMController;
input in_HREADY_SDRAMController; //added
```



```
input [31:0] in_HRDATA_DefaultSlave;
input [31:0] in_HRDATA_Accelerator;
input [31:0] in_HRDATA_SRAMController;
input [31:0] in_HRDATA_SDRAMController; //added

input in_HREADY;

// Outputs
output out_HREADY; // muxed hready out
output [31:0] out_HRDATA; // muxed read data bus out to master(s)

// Intermediates
wire masked_HREADY_DefaultSlave;
wire masked_HREADY_Accelerator;
wire masked_HREADY_SRAMController;
wire masked_HREADY_SDRAMController;  //added

reg registered_SEL_SRAM;
reg registered_SEL_SDRAM;  //added
reg registered_SEL_Accelerator;
reg registered_SEL_DefaultSlave;

// delaying HSEL inputs by registerinng them due to AHB standard
always @(posedge in_HRESET or posedge in_HCLK)
begin
  if (in_HRESET)
    begin
      registered_SEL_SDRAM  <= 1'b0; //added
      registered_SEL_SRAM  <= 1'b0;
      registered_SEL_Accelerator <= 1'b0;
      registered_SEL_DefaultSlave  <= 1'b1;
    end
```



```
  else
    if (in_HREADY)
      begin
        registered_SEL_SDRAM  <= in_HSEL_SDRAMController; //added
        registered_SEL_SRAM  <= in_HSEL_SRAMController;
        registered_SEL_Accelerator <= in_HSEL_Accelerator;
        registered_SEL_DefaultSlave  <= in_HSEL_DefaultSlave;
      end
end

assign out_HRDATA = registered_SEL_SRAM ? in_HRDATA_SRAMController : registered_SEL_SDRAM ? in_HRDATA_SDRAMController : registered_SEL_Accelerator ? in_HRDATA_Accelerator : 32'h00000000; //modified

assign masked_HREADY_DefaultSlave     = registered_SEL_DefaultSlave ? in_HREADY_DefaultSlave : 1'b0;
assign masked_HREADY_Accelerator      = registered_SEL_Accelerator ? in_HREADY_Accelerator :1'b0;
assign masked_HREADY_SRAMController   = registered_SEL_SRAM ? in_HREADY_SRAMController  : 1'b0;
assign masked_HREADY_SDRAMController  = registered_SEL_SDRAM ? in_HREADY_SDRAMController  : 1'b0; //added

assign out_HREADY                  = masked_HREADY_Accelerator | masked_HREADY_SRAMController | masked_HREADY_SDRAMController  | masked_HREADY_DefaultSlave; //modified



endmodule
```

4. Added Modules

SDRAM Controller

This module is a controller module for SDRAM. It allows master to communicate with SDRAM without knowing its internal communication procedures and commands. It receives address, write data, write/read select and controller select signals to perform. It uses these signals to communicate with SDRAM. It has a finite state machine in it. It starts with IDLE state. When the controller receives controller select signal, it gets into ACTIVATE state. It also waits 2 clock cycles in the ACTIVATE state because the RAS-CAS delay is 2 clock cycles. In this state it sends the bank and row address to the SDRAM. These addresses are selected as:

Bank address = address [29:28]

Row address = address [27:14]

Then it goes into the READ or WRITE states according to the write/read select signal. In these states it sends the column address as:

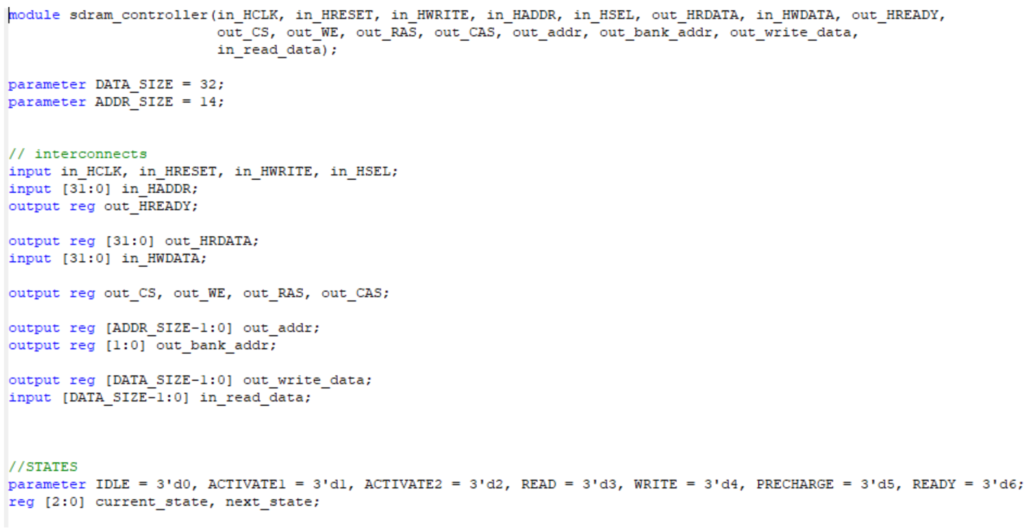
Column address = address [13:0]

In the READ state, it receives the read data and in the WRITE state, it sends the data to be written. After these states, it does not close the bank and row which are opened to be read or write.

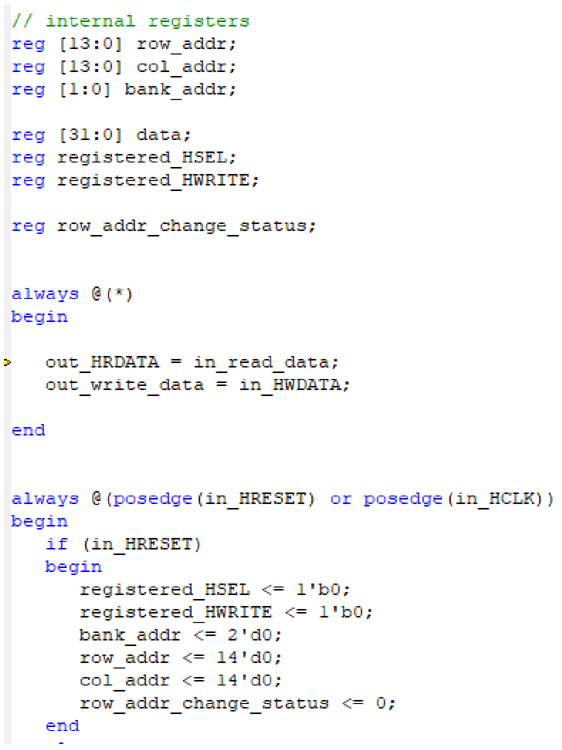
If a new row address is requested to be read or write, it gets into the PRECHARGE state. In this state, it closes the opened bank and row.

All these processes are done by sending commands to the SDRAM module. These commands are sent with command bits as CS, WE, CAS and RAS.

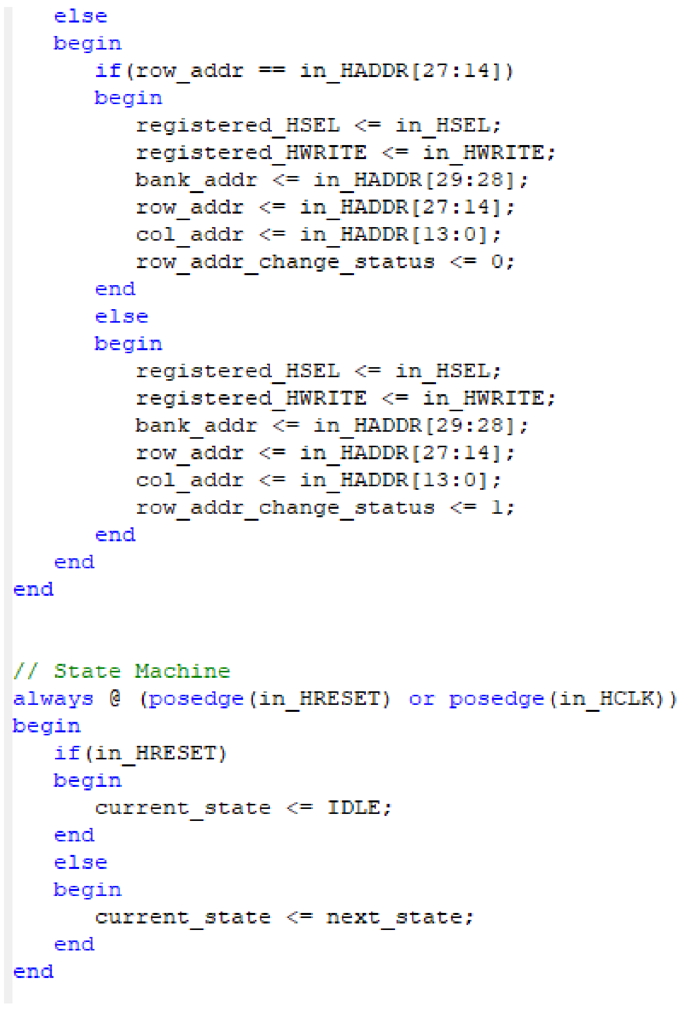
The implementation of SDRAM controller is given below:



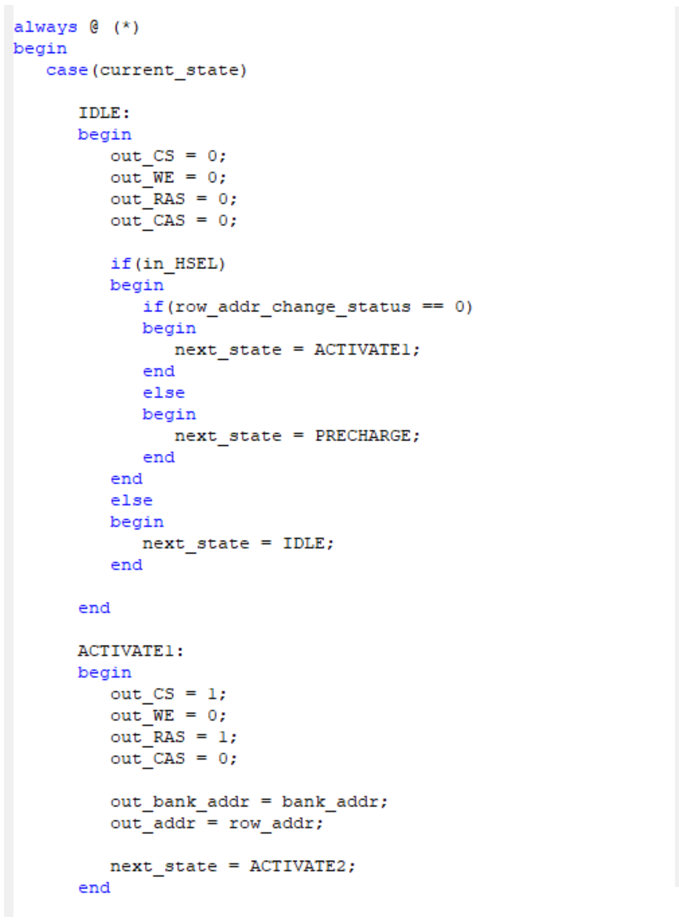
```
module sdram_controller(in_HCLK, in_HRESET, in_HWRITE, in_HADDR, in_HSEL, out_HRDATA, in_HWDATA, out_HREADY,
                        out_CS, out_WE, out_RAS, out_CAS, out_addr, out_bank_addr, out_write_data,
                        in_read_data);

parameter DATA_SIZE = 32;
parameter ADDR_SIZE = 14;


// interconnects
input in_HCLK, in_HRESET, in_HWRITE, in_HSEL;
input [31:0] in_HADDR;
output reg out_HREADY;

output reg [31:0] out_HRDATA;
input [31:0] in_HWDATA;

output reg out_CS, out_WE, out_RAS, out_CAS;

output reg [ADDR_SIZE-1:0] out_addr;
output reg [1:0] out_bank_addr;

output reg [DATA_SIZE-1:0] out_write_data;
input [DATA_SIZE-1:0] in_read_data;



//STATES
parameter IDLE = 3'd0, ACTIVATE1 = 3'd1, ACTIVATE2 = 3'd2, READ = 3'd3, WRITE = 3'd4, PRECHARGE = 3'd5, READY = 3'd6;
reg [2:0] current_state, next_state;
```



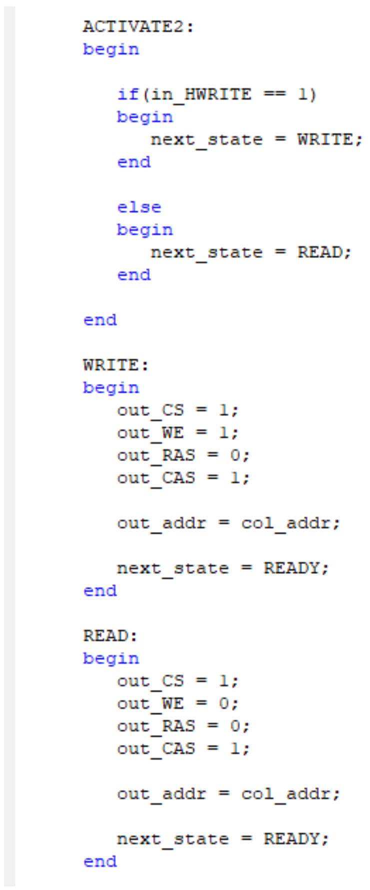
```
// internal registers
reg [13:0] row_addr;
reg [13:0] col_addr;
reg [1:0] bank_addr;

reg [31:0] data;
reg registered_HSEL;
reg registered_HWRITE;

reg row_addr_change_status;


always @(*)
begin

   out_HRDATA = in_read_data;
   out_write_data = in_HWDATA;

end


always @(posedge(in_HRESET) or posedge(in_HCLK))
begin
   if (in_HRESET)
   begin
      registered_HSEL <= 1'b0;
      registered_HWRITE <= 1'b0;
      bank_addr <= 2'd0;
      row_addr <= 14'd0;
      col_addr <= 14'd0;
      row_addr_change_status <= 0;
   end
```



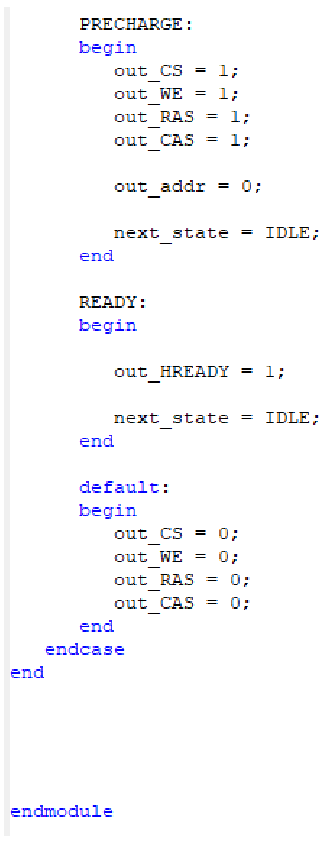
```
   else
   begin
      if(row_addr == in_HADDR[27:14])
      begin
         registered_HSEL <= in_HSEL;
         registered_HWRITE <= in_HWRITE;
         bank_addr <= in_HADDR[29:28];
         row_addr <= in_HADDR[27:14];
         col_addr <= in_HADDR[13:0];
         row_addr_change_status <= 0;
      end
      else
      begin
         registered_HSEL <= in_HSEL;
         registered_HWRITE <= in_HWRITE;
         bank_addr <= in_HADDR[29:28];
         row_addr <= in_HADDR[27:14];
         col_addr <= in_HADDR[13:0];
         row_addr_change_status <= 1;
      end
   end
end


// State Machine
always @ (posedge(in_HRESET) or posedge(in_HCLK))
begin
   if(in_HRESET)
   begin
      current_state <= IDLE;
   end
   else
   begin
      current_state <= next_state;
   end
end
```



```
always @ (*)
begin
   case(current_state)

      IDLE:
      begin
         out_CS = 0;
         out_WE = 0;
         out_RAS = 0;
         out_CAS = 0;

         if(in_HSEL)
         begin
            if(row_addr_change_status == 0)
            begin
               next_state = ACTIVATE1;
            end
            else
            begin
               next_state = PRECHARGE;
            end
         end
         else
         begin
            next_state = IDLE;
         end

      end

      ACTIVATE1:
      begin
         out_CS = 1;
         out_WE = 0;
         out_RAS = 1;
         out_CAS = 0;

         out_bank_addr = bank_addr;
         out_addr = row_addr;

         next_state = ACTIVATE2;
      end
```



```
      ACTIVATE2:
      begin

         if(in_HWRITE == 1)
         begin
            next_state = WRITE;
         end

         else
         begin
            next_state = READ;
         end

      end

      WRITE:
      begin
         out_CS = 1;
         out_WE = 1;
         out_RAS = 0;
         out_CAS = 1;

         out_addr = col_addr;

         next_state = READY;
      end

      READ:
      begin
         out_CS = 1;
         out_WE = 0;
         out_RAS = 0;
         out_CAS = 1;

         out_addr = col_addr;

         next_state = READY;
      end
```



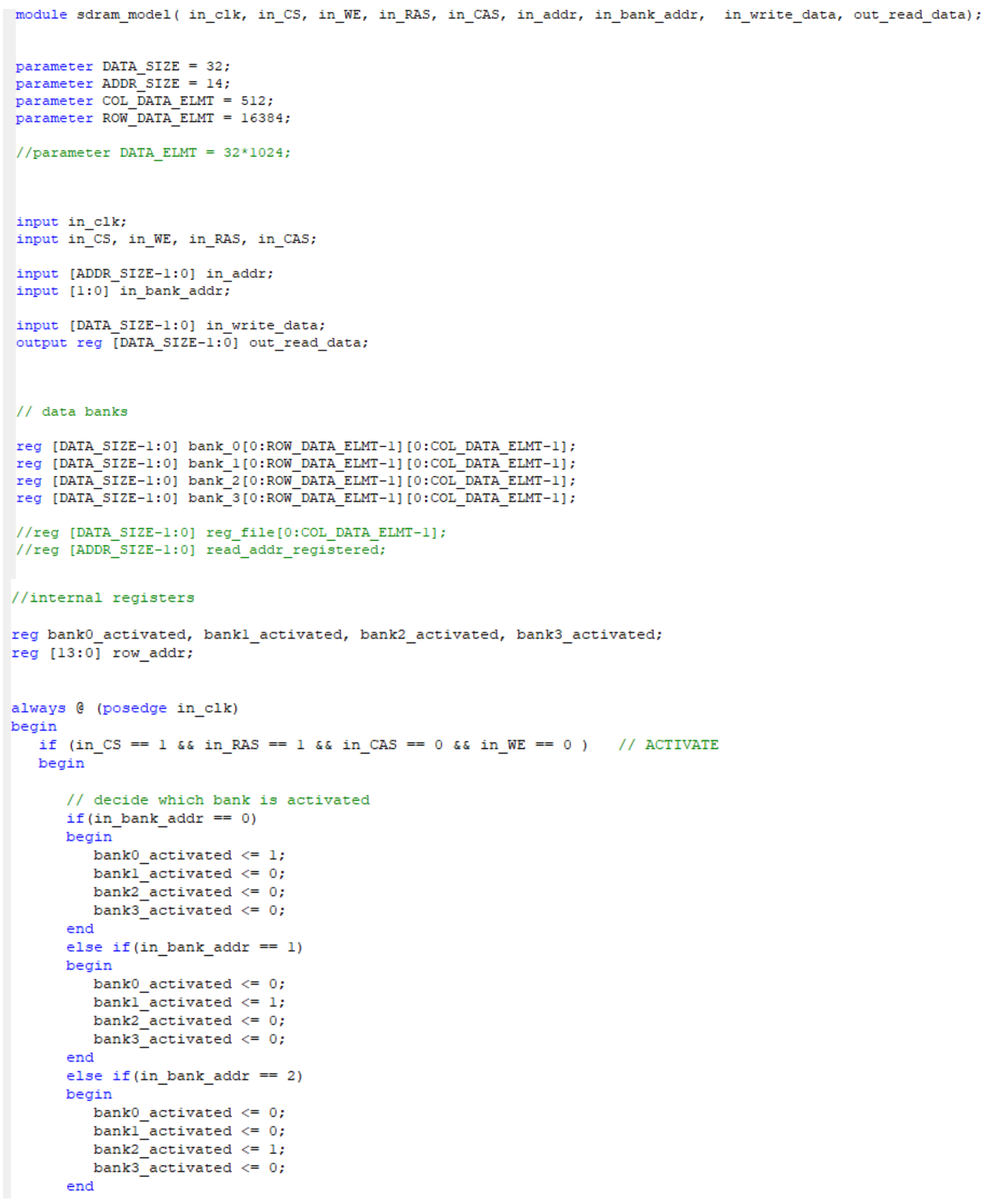
```
      PRECHARGE:
      begin
         out_CS = 1;
         out_WE = 1;
         out_RAS = 1;
         out_CAS = 1;

         out_addr = 0;

         next_state = IDLE;
      end

      READY:
      begin

         out_HREADY = 1;

         next_state = IDLE;
      end

      default:
      begin
         out_CS = 0;
         out_WE = 0;
         out_RAS = 0;
         out_CAS = 0;
      end
   endcase
end



endmodule
```

SDRAM Model

This module includes a model to simulate a SDRAM. This SDRAM has 4 banks. It also has 14 column and 14 row select bits. It receives 4 control pins to receive commands from the SDRAM controller. These bits are CS, WE, CAS, RAS. With this pin, it receives commands such as ACTIVATE, READ, WRITE and PRECHARGE. With ACTIVATE command, it receives the bank and row address to be opened. With READ command, it receives column address and reads the bits in the opened row. It also sends these bits which is read to the controller. With WRITE command, it receives column address and write data. It writes this data to the address in the opened row. With the PRECHARGE command, it closes the opened row and bank.

In the banks data is saved in words with 32 bits. There are 14 column and 14 row select bits. Therefore, there are 2^{14} = 16384 rows and 2^{14}/32 = 512 columns with words with 32 bits.

The implementation of SDRAM is given below:



```
module sdram_model( in_clk, in_CS, in_WE, in_RAS, in_CAS, in_addr, in_bank_addr,  in_write_data, out_read_data);


parameter DATA_SIZE = 32;
parameter ADDR_SIZE = 14;
parameter COL_DATA_ELMT = 512;
parameter ROW_DATA_ELMT = 16384;

//parameter DATA_ELMT = 32*1024;



input in_clk;
input in_CS, in_WE, in_RAS, in_CAS;

input [ADDR_SIZE-1:0] in_addr;
input [1:0] in_bank_addr;

input [DATA_SIZE-1:0] in_write_data;
output reg [DATA_SIZE-1:0] out_read_data;



// data banks

reg [DATA_SIZE-1:0] bank_0[0:ROW_DATA_ELMT-1][0:COL_DATA_ELMT-1];
reg [DATA_SIZE-1:0] bank_1[0:ROW_DATA_ELMT-1][0:COL_DATA_ELMT-1];
reg [DATA_SIZE-1:0] bank_2[0:ROW_DATA_ELMT-1][0:COL_DATA_ELMT-1];
reg [DATA_SIZE-1:0] bank_3[0:ROW_DATA_ELMT-1][0:COL_DATA_ELMT-1];

//reg [DATA_SIZE-1:0] reg_file[0:COL_DATA_ELMT-1];
//reg [ADDR_SIZE-1:0] read_addr_registered;

//internal registers

reg bank0_activated, bank1_activated, bank2_activated, bank3_activated;
reg [13:0] row_addr;


always @ (posedge in_clk)
begin
   if (in_CS == 1 && in_RAS == 1 && in_CAS == 0 && in_WE == 0 )    // ACTIVATE
   begin

      // decide which bank is activated
      if(in_bank_addr == 0)
      begin
         bank0_activated <= 1;
         bank1_activated <= 0;
         bank2_activated <= 0;
         bank3_activated <= 0;
      end
      else if(in_bank_addr == 1)
      begin
         bank0_activated <= 0;
         bank1_activated <= 1;
         bank2_activated <= 0;
         bank3_activated <= 0;
      end
      else if(in_bank_addr == 2)
      begin
         bank0_activated <= 0;
         bank1_activated <= 0;
         bank2_activated <= 1;
         bank3_activated <= 0;
      end
      else if(in_bank_addr == 3)
      begin
         bank0_activated <= 0;
         bank1_activated <= 0;
         bank2_activated <= 0;
         bank3_activated <= 1;
      end


      //keep the row address
      row_addr <= in_addr;


   end

   else if (in_CS == 1 && in_RAS == 0 && in_CAS == 1 && in_WE == 0 )   // READ
   begin

      if(bank0_activated == 1)
      begin
         out_read_data <= bank_0[row_addr][in_addr];
      end
      else if(bank1_activated == 1)
      begin
         out_read_data <= bank_1[row_addr][in_addr];
      end
      else if(bank2_activated == 1)
      begin
         out_read_data <= bank_2[row_addr][in_addr];
      end
      else if(bank3_activated == 1)
      begin
         out_read_data <= bank_3[row_addr][in_addr];
      end

   else if (in_CS == 1 && in_RAS == 0 && in_CAS == 1 && in_WE == 1 )   // WRITE
   begin

      if(bank0_activated == 1)
      begin
         bank_0[row_addr][in_addr] <= in_write_data;
      end
      else if(bank1_activated == 1)
      begin
         bank_1[row_addr][in_addr] <= in_write_data;
      end
      else if(bank2_activated == 1)
      begin
         bank_2[row_addr][in_addr] <= in_write_data;
      end
      else if(bank3_activated == 1)
      begin
         bank_3[row_addr][in_addr] <= in_write_data;
      end

   end

   else if (in_CS == 1 && in_RAS == 1 && in_CAS == 1 && in_WE == 1 )   // PRECHARGE
   begin
      bank0_activated <= 0;
      bank1_activated <= 0;
      bank2_activated <= 0;
      bank3_activated <= 0;

      row_addr <= 0;

   end

   else
   begin
      bank0_activated <= 0;
      bank1_activated <= 0;
      bank2_activated <= 0;
      bank3_activated <= 0;

      row_addr <= 0;

   end
end




endmodule
```