CESYS

XV2DDR
DDR SDRAM CONTROLLER

PRELIMINARY

Ver 1.0 - August 6, 2002

Netlist and Test Design Documentation

1 OVERVIEW

The DDR SDRAM is an enhancement to the traditional synchronous DRAM. It doubles the data throughput by transferring data on both edges of each clock cycle.

Unlike many other ram modules, DDR SDRAMs are harder to control because of timing issues and its banked structure. CESYS XV2DDR, DDR SDRAM Controller hides these complexities from the user and defines a simple SRAM like interface which is very easy to understand and use. It is tested with Micron DDR-SDRAM MT46V16M16 which is found on XV2DDR board.

DDR SDRAM package is synthesized with Synplicity's Synplify tool and place & routed with XILINX ISE environment. Throughout this document it is thought that you use the same environment although you can use others.

2 DDR SDRAM CONTROLLER

2.1 Features

Supported Features :

- Burst READ and burst WRITE access to the Micron DDR-SDRAM MT46V16M16.
- The initialization cycle, including MODE REGISTER and EXTENDED MODE REGISTER setting which is done automatically by the DDR-SDRAM Controller.
- The commands NOP and ACTIVE are generated by the controller.
- Supported READ and WRITE bursts are 2, 4 or 8, which can be set by one constant definition in the VHDL source code.
- READ and WRITE is done with AUTO PRECHARGE.
- The CAS latency is fixed to 2.
- Additional AUTO REFRESH is supported.

Unsupported Features:

- Power down
- CAS latency 2.5
- NOP cannot be controlled by user
- ACTIVE cannot be controlled by user
- PRECHARGE
- SELF REFRESH
- BURST TERMINATE

Initialization:

- CAS latency: 2
- Burst length: generic value
- Burst type: sequential
- Reduced drive strength

2.1 Distribution

Following is the directory and file structure of DDR SDRAM controller distribution with brief explanations of some important files.

ddr_sdram_controller - top level directory
- mt46v16m16.vhd – vhdl simulation model for MICRON MT46V16M16 ddr sdram
- **netlist** - includes controller netlist and other necessary files
  - ddr_sdr_pkg.vhd – ddr sdram controller library package file
  - **burst2**
    - ddr_sdram_ctrl.edf – ddr sdram controller netlist with burst 2
    - ddr_sdram_ctrl.vhm – ddr sdram vhdl netlist for simulation with burst 2
  - **burst4**
    - ddr_sdram_ctrl.edf – ddr sdram controller netlist with burst 4
    - ddr_sdram_ctrl.vhm – ddr sdram vhdl netlist for simulation with burst 4
  - **burst8**
    - ddr_sdram_ctrl.edf – ddr sdram controller netlist with burst 8
    - ddr_sdram_ctrl.vhm – ddr sdram vhdl netlist for simulation with burst 8
    - ddr_sdram_test.edf – ddr sdram test module netlist with burst 8
- **synplify_test** – controller test project
  - ddr_sdr_test.vhd – ddr sdram vhdl test entity
  - ddr_sdr_test.sdc – Synplify constraints file
  - **ise –** Xilinx Ise project for place & route and bit file generation
    - ddr_sdr_test.ucf – Ise constraints file
    - ddr_sdr_test.bit – Downloadable bit stream file to test the ddr sdram
    - v2test.mcs – Dummy configuration file for the flash chip XC18V04

2.2 User interface

The following table displays the signals which comprise the user interface of the DDR SDRAM controller.

clk	in	std_logic	Maximum 133 MHz system clock
rst_n	in	std_logic	Async. reset, low active
addr_q (23 downto 0)	in	std_logic_vector	Ram address
data_in_q (31 downto 0)	in	std_logic_vector	Ram input data
cmd_q (6 downto 0)	in	std_logic_vector	Command
data_out_q (31 downto 0)	out	std_logic_vector	Ram output data
data_vld_q	out	boolean	Output data valid (ready)
busy_q	out	boolean	When true, controller is in action, further commands are ignored

All Inputs (addr_q, data_in_q, cmd_q) have to be synchronous to the system clock. The outputs are synchronous to the system clock.

2.3 Operations

DDR SDRAM controller's functionality is controlled with cmd_q vector. The upper 5 bits of this vector is not used by the user. Following is the meanings of the lower 2 bits:

cmd_q="xxxxx00" NOP = no command, idle
cmd_q="xxxxx01" Burst READ
cmd_q="xxxxx10" Burst WRITE

The READ and WRITE commands must be active for one clock cycle and they must be followed by a NOP command. With the command the start address is latched and in case of WRITE the first double word is latched too.
The subsequent double words have to be provided with the next clock cycle(s).
One clock cycle after the command is issued; the busy flag becomes true and stays true since the controller is ready for the next command. Commands are ignored by the controller when busy is true.

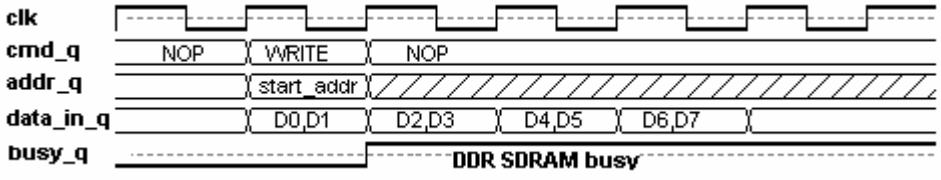

Figure 1: Write Cycle Timing Diagram (Burst 8)

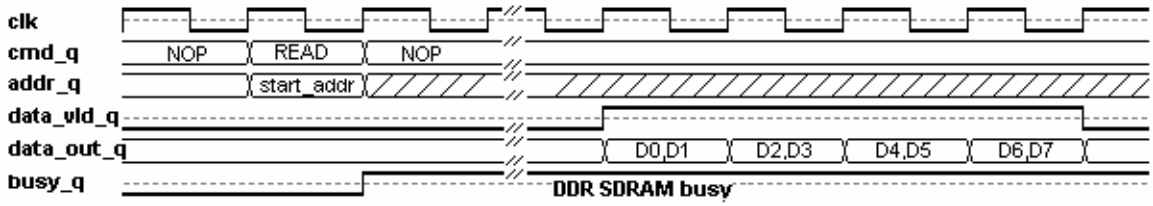

Figure 2: Read Cycle Timing Diagram (Burst 8)

The number of clock cycles between a read command and the first valid data is dependent on implementation and cannot be fixed yet.

Controller's address line, addr_q, is organized in a way that it reflects the banked structure of a DDR SDRAM. Micron MT46V16M16 device is organized in 4 banks. Each bank consists of 8192 rows and each row contains 512 address locations of words. Overall it can address:

4 x 8192 x 512 x 16 (bits) = 256 Kbits = 32 Kbytes

Following is the address assignment of user interface:

addr_q : address line, 24 bits
addr_q(23..22) : bank select, 2 bits, 4 banks
addr_q(21…9) : row, 13 bits, 0…8191
addr_q(8…..0) : column, 9 bits, 0…511

3 TEST AND REDESIGN

3.1 Testing the design

Connect the download cable to XV2DDR board and switch the power on. Connect test outputs to your logic analyzer. Test signals are connected to the north connector as shown in the table below.

Pin Name	North connector pin name	Signal name & function
Test(0)	N11	writing signal – Active when test data is written to the ddr sdram
Test(1)	N13	reading signal - Active when test data is read from the ddr sdram
Test(2)	N15	error signal – Active when there is a data comparison error
Test(3)	N17	done signal – Active when test is completed successfully
Test(4)	N19	busy signal – Active when ddr sdram module is busy
Test(5)	N21	data valid signal – Active when there is a valid data to read from the ddr sdram

Download the design ddr_sdram_controller\synplify_test\ise\ddr_sdr_test.bit on the Virtex II chip and v2test.mcs file on the flash chip XC18V04 using a tool like IMPACT from XILINX which supports JTAG interface. Since test design produces internal reset signal, the design starts running immediately after download. If you want to produce a manual reset just switch the user switch 1 ON and OFF. Now you can observe the test signals by triggering on them. Below is a sample signal diagram of test pins, during a successful test:

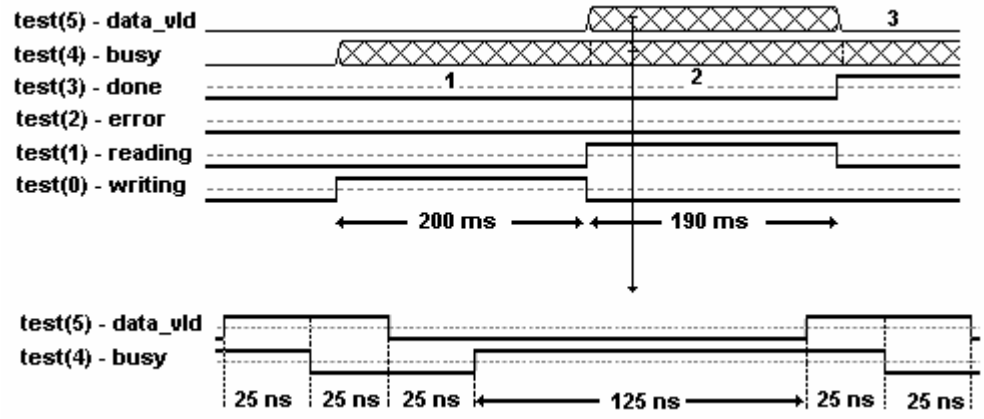

Figure 3 – DDR SDRAM Controller test diagram

The clock frequency for this design is 80 Mhz, and the values below will change with other frequencies.

In region 1 busy signal is active because of auto refreshing of the ddr sdram and reading.

In region 2 it is active additionally for writing. The lower diagram shows the relationship between the busy and data valid signals during a write transaction to the ddr sdram. Notice that the values written are nominal and may differ slightly from the real ones.

In region 3 ddr sdram is idle but refreshing is still active. However, you don't have to care about refreshing since the controller state machine is intelligent enough to handle it automatically. During auto refreshing busy signal is active for 5 clock cycles (112.5 ns) with a frequency of 130 kHz.

3.2 Rebuilding the design

Open ddr_sdram_controller\synplify_test\ddr_sdr_test.prj project file with Synplify. Choose frequency as 133 Mhz (maximum that ddr sdram can achieve), select Symbolic FSM Compiler and Resource sharing options. Choose Xilinx Virtex II XC2V1000 with speed -5 and package fg456 as the target device. Change the output edif netlist path to ddr_sdram_controller\netlist\burst8\ddr_sdr_test.edf. Be sure that you did not check Disable I/O insertion option and run the compiler.

After you have created the netlist open the project file ddr_sdram_controller\synplify_test\ise\ise.npl with Xilinx Ise Project Navigator tool. This project includes two netlist files: ddr_sdr_ctrl.edf and ddr_sdr_test.edf together with ddr_sdr_test.ucf constraints file which includes all timing, connection and other constraints related features of the signals used in the test design. The second netlist is the one that you just created.

In the module view pane highlight the ddr_sdr_ctrl.vhd file, than right click on *Implement design entry* in process view pane and select *Properties*. On this applet choose *Map Properties* tab. Select *For Inputs and Outputs* option for *Pack I/O Registers/Latches into IOBs* entry. Than choose *Place & Route properties* pan and select *Normal* option for *Place & Route Effort Level (Overall)* entry.

Right click on *Generate programming file* entry and choose *Properties*. In *Startup options* tab select *JTAG clock* for *Startup clock* entry. Run the place & route tool and generate the bit stream file ddr_sdr_test.bit. Check the section *3.1 – Testing the design* for a description of the usage & test of this file.

3.3 Using the netlist in your own design

Include ddr_sdr_pkg.vhd library package file into your project. Open it in a vhdl viewer and scroll down to the following code:

```
-- *** BURST Constants ***
-- these constants can be set by the user to determine burst length and data generation for test application
-- constant BURST_LENGTH : BURST_LENGTH_TYPE := TWO;
-- constant BURST_LENGTH : BURST_LENGTH_TYPE := FOUR;
constant BURST_LENGTH : BURST_LENGTH_TYPE := EIGHT;
```

There are three constant definitions which define the burst lengths used by the ddr sdram controller. Uncomment only the one that will use. Others must be commented. Be sure that you use the right version of the ddr sdram controller netlist, since there are three versions of it for burst lengths 2, 4 and 8. Each of them are located in different directories under ddr_sdram_controller\netlist\burstX.

Create your own vhdl file & entity which will use ddr sdram controller component. You can find the component declaration of the ddr sdram controller at the very beginning of ddr_sdr_pkg.vhd file as shown below. Copy this code into your source file and uncomment it. Than instantiate the component with respect to your logic.

```
--
--component DDR_SDR_CTRL
--      port (
--      clk             : in std_logic;     -- Max 133 MHz system clock, from fpga pad
--      clk_fb          : in std_logic;     -- Feedback clock
```

```
--         rst_n             : in std_logic;     -- Asyncronous reset, low active
--       sys_rst_n         : out std_logic;  -- Asyncronous reset, low active
                                                -- Released after DCMs locked.
--                                              -- Use this reset signal in your design
--       sys_clk_out      : out std_logic;  -- Max 133 MHz system clock after DCM.
--                                              --  Use this clock in your design.
--
--
--       -- user interface
--       cmd_q              : in std_logic_vector(U_CMD_WIDTH -1 downto 0);
--                                              -- Command: read, write, nop
--       addr_q             : in std_logic_vector(U_ADDR_WIDTH -1 downto 0);  -- Address
--       data_in_q          : in std_logic_vector(U_DATA_WIDTH -1 downto 0);  -- Input data
--       data_vld_q         : out boolean;                          -- Is there a valid data to read?
--       data_out_q         : out std_logic_vector(U_DATA_WIDTH -1 downto 0); -- read data
--       busy_q             : out boolean;                          -- Is ddr sdram controller busy?
--                                                                 --When true all commands are ignored.
--
--         -- ddr_sdr interface
--       sdr_clk            : out std_logic;    -- ddr_sdram_clock
--       sdr_clk_n          : out std_logic;    -- ddr_sdram_clock inverted
--       cke_q              : out std_logic;    -- clock enable
--       cs_qn              : out std_logic;    -- chip select
--       ras_qn             : out std_logic;    -- ras
--       cas_qn             : out std_logic;    -- cas
--       we_qn              : out std_logic;    -- write enable
--       ldm_q              : out std_logic;    -- data mask, lower byte
--       udm_q              : out std_logic;    -- data mask, upper byte
--       dqs                : out std_logic_vector(1 downto 0);
--       dqsz_q             : out std_logic_vector(1 downto 0);
--       ba0_q              : out std_logic;    -- bank select
--       ba1_q              : out std_logic;    -- bank select
--       a_q                : out std_logic_vector(SDR_ADDR_WIDTH -1 downto 0);
                                                -- address bus
--       data_in            : in std_logic_vector(SDR_DATA_WIDTH -1 downto 0);  -- data bus
--       d2sdr              : out std_logic_vector(SDR_DATA_WIDTH -1 downto 0);  -- data bus
--       tristate_q         : out std_logic_vector(15 downto 0);
--
--         --test pins
--         dcms_locked         : out std_logic_vector(1 downto 0)  -- DCMs locked when "11"
--       );
--end component;

-- Attributes specific to Synplify. If you use another tool for synthesis pleace replace them
-- with the corresponding ones.
attribute syn_black_box : boolean;
attribute syn_black_box of DDR_SDR_CTRL: component is true;
attribute black_box_tri_pins : string;
attribute black_box_tri_pins of DDR_SDR_CTRL : component is "tristate_q";
```

There are some attributes that affect the behaviour of the ddr_sdram_controller netlist which you can see at the end of the previous code. As indicated in the commented part, these attributes are specific to Synplify synthesizer. Therefore, if you use another tool, you should check the documentation of your Vendor's documentation and replace them with their counterparts. If not just copy and paste this code after your component declaration.

You should also copy the following code into your vhdl file. It is necessary to put the tristate flip-flops (FF) into IOBs (input output blocks). One tristate FF is needed for every data-bit and strobe, to get IOB-FFs. If you use only one tristate-FF it won't be mapped to IOBs and the interface timing may not match.

```
--signal definitions
signal data_in, d2sdr : std_logic_vector(SDR_DATA_WIDTH -1 downto 0);
signal tristate_q : std_logic_vector(15 downto 0);
signal dqs, dqsz_q : std_logic_vector(1 downto 0);
…..
…..
…..
gen3: for n in 0 to 15 generate
        data(n) <= 'Z' when tristate_q(n)='1' else d2sdr(n);
        data_in(n)  <= data(n);
end generate;
ldqs_q <= 'Z' when dqsz_q(0)='1' else dqs(0);
udqs_q <= 'Z' when dqsz_q(1)='1' else dqs(1);

-- ddr sdram controller component instantiation
u_ddr : DDR_SDR_CTRL
        port map (
        clk             => clk_b,
        clk_fb          => clk_fb,
        rst_n            => rst_n_g,


       …
        …
        …

        dqs             => dqs,
        dqsz_q           => dqsz_q,
        data_in          => data_in,
        d2sdr            => d2sdr,
        tristate_q       => tristate_q,

        --test pins
        dcms_locked        => dcms_locked);
```