USING XILINX'S DDR3 CORE IN A VIRTEX 6 FPGA

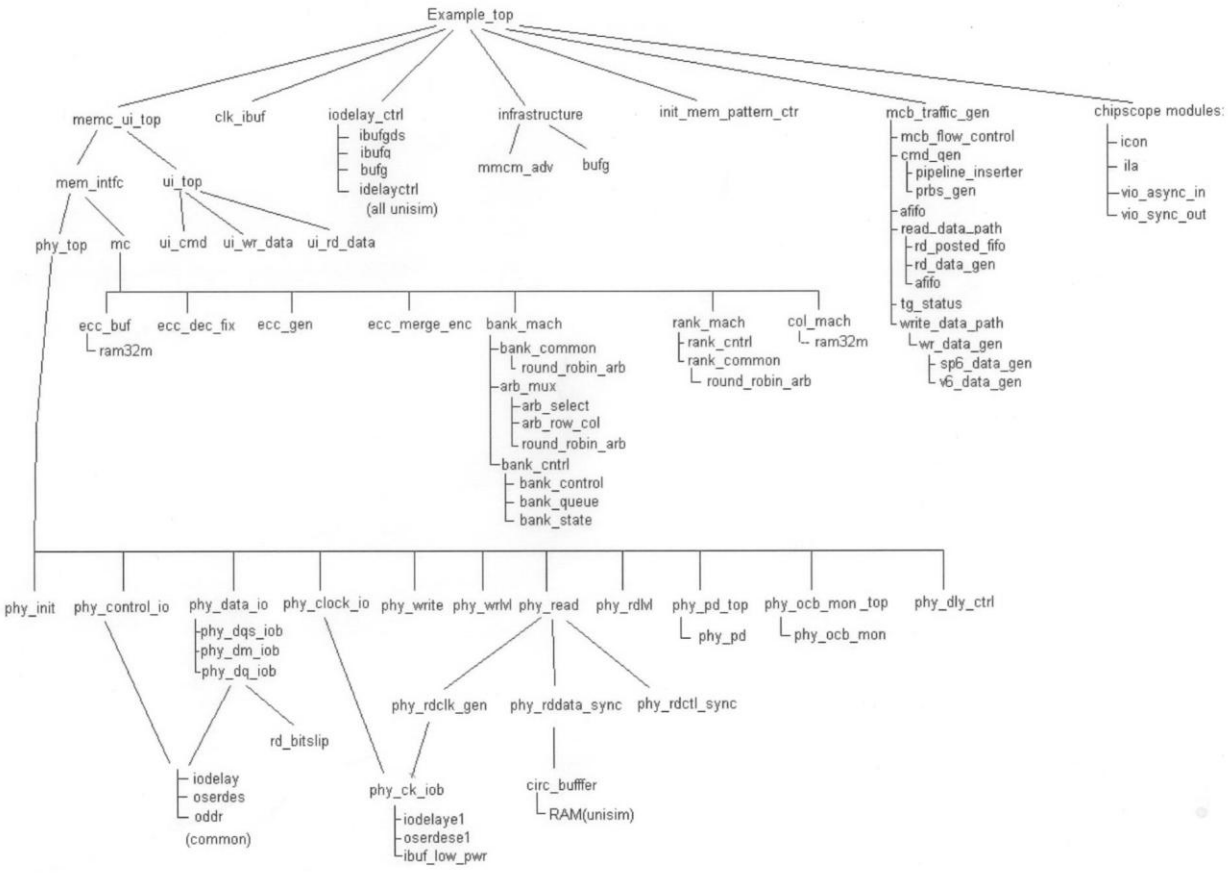
Jeffrey A. Walck

Xilinx's DDR3 core is a powerful piece of circuitry, enabling high data bandwidth from a variety of possible memory configurations. However, its sophistication can prove daunting and confusing.

Xilinx's User Guide "UG406" describes the operation of their DDR3 core in a Virtex 6. The original version 1.0 of the user guide had sketchy, and sometimes incorrect, information. Fortunately UG406 has evolved to be much more informative and useful. This tutorial does not substitute for the user guide, but instead provides some handy hints and tips that can be used in concert with the user guide.

1.0 Code Module Structure

The structure chart below shows the organization of code modules that make up the core. These are the modules that are listed in the "Example" design directory. The function of each code module is described in chapter 1 of the user guide, within the paragraph "Directory Structure and File Descriptions," and in Tables 1.1 through 1.13.



2.0 Master State Machine State Listing

The table below is excerpted from the code module phy_init.vhd and lists the states that the core's internal state machine progresses through during initialization. These states can be monitored using Xilinx's ChipScope™ internal logic analyzer. They can also be watched during simulation as further described in paragraph 3.0.

```
-- Master state machine encoding
constant INIT_IDLE                  : std_logic_vector(5 downto 0) := "000000";       --0
constant INIT_WAIT_CKE_EXIT         : std_logic_vector(5 downto 0) := "000001";       --1
constant INIT_LOAD_MR               : std_logic_vector(5 downto 0) := "000010";       --2
constant INIT_LOAD_MR_WAIT          : std_logic_vector(5 downto 0) := "000011";       --3
constant INIT_ZQCL                  : std_logic_vector(5 downto 0) := "000100";       --4
constant INIT_WAIT_DLLK_ZQINIT      : std_logic_vector(5 downto 0) := "000101";       --5
constant INIT_WRLVL_START           : std_logic_vector(5 downto 0) := "000110";       --6
constant INIT_WRLVL_WAIT            : std_logic_vector(5 downto 0) := "000111";       --7
constant INIT_WRLVL_LOAD_MR         : std_logic_vector(5 downto 0) := "001000";       --8
constant INIT_WRLVL_LOAD_MR_WAIT    : std_logic_vector(5 downto 0) := "001001";       --9
constant INIT_WRLVL_LOAD_MR2        : std_logic_vector(5 downto 0) := "001010";       --A
constant INIT_WRLVL_LOAD_MR2_WAIT   : std_logic_vector(5 downto 0) := "001011";       --B
constant INIT_RDLVL_ACT             : std_logic_vector(5 downto 0) := "001100";       --C
constant INIT_RDLVL_ACT_WAIT        : std_logic_vector(5 downto 0) := "001101";       --D
constant INIT_RDLVL_STG1_WRITE      : std_logic_vector(5 downto 0) := "001110";       --E
constant INIT_RDLVL_STG1_WRITE_READ : std_logic_vector(5 downto 0) := "001111";       --F
constant INIT_RDLVL_STG1_READ       : std_logic_vector(5 downto 0) := "010000";       --10
constant INIT_RDLVL_STG2_WRITE      : std_logic_vector(5 downto 0) := "010001";       --11
constant INIT_RDLVL_STG2_WRITE_READ : std_logic_vector(5 downto 0) := "010010";       --12
constant INIT_RDLVL_STG2_READ       : std_logic_vector(5 downto 0) := "010011";       --13
constant INIT_RDLVL_STG2_READ_WAIT  : std_logic_vector(5 downto 0) := "010100";       --14
constant INIT_PRECHARGE_PREWAIT     : std_logic_vector(5 downto 0) := "010101";       --15
constant INIT_PRECHARGE             : std_logic_vector(5 downto 0) := "010110";       --16
constant INIT_PRECHARGE_WAIT        : std_logic_vector(5 downto 0) := "010111";       --17
constant INIT_DONE                  : std_logic_vector(5 downto 0) := "011000";       --18
constant INIT_IOCONFIG_WR           : std_logic_vector(5 downto 0) := "011001";       --19
constant INIT_IOCONFIG_RD           : std_logic_vector(5 downto 0) := "011010";       --1A
constant INIT_IOCONFIG_WR_WAIT      : std_logic_vector(5 downto 0) := "011011";       --1B
constant INIT_IOCONFIG_RD_WAIT      : std_logic_vector(5 downto 0) := "011100";       --1C
constant INIT_DDR2_PRECHARGE        : std_logic_vector(5 downto 0) := "011101";       --1D
constant INIT_DDR2_PRECHARGE_WAIT   : std_logic_vector(5 downto 0) := "011110";       --1E
constant INIT_REFRESH               : std_logic_vector(5 downto 0) := "011111";       --1F
constant INIT_REFRESH_WAIT          : std_logic_vector(5 downto 0) := "100000";       --20
constant INIT_PD_ACT                : std_logic_vector(5 downto 0) := "100001";       --21
constant INIT_PD_ACT_WAIT           : std_logic_vector(5 downto 0) := "100010";       --22
constant INIT_PD_READ               : std_logic_vector(5 downto 0) := "100011";       --23
constant INIT_REG_WRITE             : std_logic_vector(5 downto 0) := "100100";       --24
constant INIT_REG_WRITE_WAIT        : std_logic_vector(5 downto 0) := "100101";       --25
constant INIT_DDR2_MULTI_RANK       : std_logic_vector(5 downto 0) := "100110";       --26
constant INIT_DDR2_MULTI_RANK_WAIT  : std_logic_vector(5 downto 0) := "100111";       --27
```

excerpted from phy_init.vhd (MIG 3.4)

3.0 Speeding up Simulation of the Core

The core, together with behavioral models of the DDR3 SDRAM makes for a very complex piece of logic. Attempting to simulate it using typical software-only simulation (such as Modelsim) will result in such long run times as to be impractical even if storage of the resulting enormous files was not an issue. To get around this problem, select VHDL generic options that skip over parts of the core initialization.

Unfortunately, knowing which options to select is not clear-cut. Some of the generic settings described in the comments are outright wrong, or do nothing. For example, in the listing below, for "SIM_INIT_OPTION," the skip-power-up-delay generic value "SKIP_PU_DELAY" does not work. That is, the power up delay takes place even with this option selected. For each of generic options below, note

the selection that was made along with the comments describing what actually works. These are the options needed to minimize simulation times during the RAM's initialization. These "SKIP" options should only be used when compiling for simulation. For operation in hardware, the "NONE" options should be selected.

Similarly, WRLVL = "OFF" turns off the time-consuming write leveling operation for simulation. WRLVL should be set to "ON" for operation in actual hardware.

In example design, some of the top level generics are wrong. The following addition/changes to the generics are needed to skip DRAM initialization, calibration, and write leveling during simulation:

```
SIM_INIT_OPTION     = "SKIP_INIT"        ( string )
-- # = "SKIP_PU_DLY" - Skip the memory
--                     initilization sequence,
-- # = "SKIP_INIT" - Skip the memory                 -- JW this actually skips the sequence
--                     initilization sequence,
--   = "NONE" - Complete the memory
--              initilization sequence.
SIM_CAL_OPTION      = "SKIP_CAL"         ( string )
-- # = "FAST_CAL" - Skip the delay
--                  Calibration process,
-- # = "SKIP_CAL" - Skip the delay                   -- JW this actually skips calibration
--                  Calibration process,
--   = "NONE" - Complete the delay
--              Calibration process.

WRLVL               = "OFF"              ( string )   -- JW changed to OFF for simulation
-- # = "ON" - DDR3 SDRAM
--   = "OFF" - DDR2 SDRAM.
```

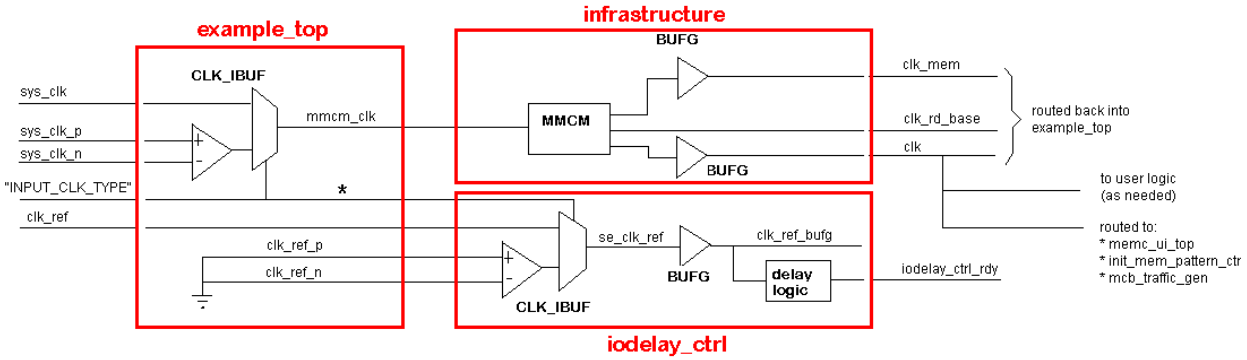
-- This was needed to make the reset work correctly at the top level:

```
RST_ACT_LOW         = 0                  ( integer )  -- JW change
-- =1 for active low reset,
-- =0 for active high.
```

4.0 Propagation of Clocks through the Core

Following the progression of multiple clocks through the core can be a chore. Some of the clocks are used within the core itself for write and read operations. Some must be routed back out of the core to enable user logic to synchronize with the core's user interface (UI), and some are routed to the RAM devices themselves.

The handy diagram below shows the major clocks and how their names evolve as they are routed through various code modules.



* it's assumed INPUT_CLK_TYPE must select "SINGLE_ENDED" to produce iodelay_ctrl_rdy signal.