Gisselquist
Technology, LLC

WB DDR3 SDRAM CONTROLLER SPECIFICATION

Dan Gisselquist, Ph.D.
dgisselq (at) opencores.org

August 1, 2016

Revision History

Rev.	Date	Author	Description
0.0	6/20/2016	D. Gisselquist	Initial Version

Contents

Figures

Figure Page

Tables

Preface

Now, just why am I building this? Because wishbone's been so good to me? Because I've never used AXI? Because I dislike not being able to see what goes on within a memory controller, and have no insight into why it's performance is as fast (or slow) as it is? Because Xilinx allows you to only open 4 banks at a tim? Or is it because, when I went to purchase my first high speed FPGA circuit board, the vendor offered me the opportunity to purchase a DMA controller with it? As a micro businessman, I really can't afford using someone else's stuff. Time is cheap, money isn't nearly so cheap.

Hence, I offer my work to you as well. I hope you find it useful. Of course, the normal caveats are available: I am available for hire, and I would be happy to modify this core or even the license it is distributed under, for an appropriate incentive.

Dan Gisselquist, Ph.D.

1.

Introduction

The purpose of this core is to provide a GPL Wishbone Core capable of commanding a DDR3 memory at full speed. A particular design goal is that consecutive reads or writes should only take one additional clock per read/write.

Since the DDR3 memory specification is dated as of August, 2009, memory chips have been built to this specification. However, since DDR3 SDRAM's are rather complex, and there is a lot of work required to manage them, controllers for DDR3 SDRAM's remain primarily in the realm of proprietary.

Currently, there are no DDR3 controllers present on OpenCores. Sure, there's a project named "DDR3 SDRAM controller", yet it has no data files present with it. This leaves the FPGA engineers with the choice of building a controller for a very complex interface, or using a proprietary core from Xilinx's Memory Interface Generator, for which there is no insight into how it works, and then retooling their bus from wishbone to AXI.

This core is designed to meet that need: it is both open (GPL), as well as wishbone compliant. Further, this core offers 32–bit granularity to an interface that would otherwise offer only 128–bit granularity. This core also offers complete pipelind performance. Because of the pipeline performance, this core is very appropriate for filling cache lines. Because the core also offers non–pipelined performance, it is also appropriate for random access from a CPU–whether by a write–through cache or a CPU working without a cache.

2.

Architecture

2.1 Strategies

2.1.1 Bank

Currently, banks are activated (opened) when needed and only precharged (closed) upon refresh request. Further, upon any read or write from one bank, the next bank is activated as well, under the assumption that the next bank will be needed soon. This is necessary to allow pipeline access with no stalls through the memory controller.

This means that, upon any bank miss, a bank precharge followed by bank activate command will be necessary.

2.1.2 Refresh

The current build will pause all operations for four subsequent refreshes, at roughly every 4 refresh intervals, and then allow operations to resume. This pause is independent of anything going on, and includes a mandatory wait for any writes to finish, followed by a precharge command—regardless of whether or not such is required.

This is non-optimal, and ripe for optimizing later. A better strategy might be to do singular refreshes after any single refresh period assuming the bus is free, to only issue a precharge if the bus is busy, and to only wait prior to that precharge if a write is busy. This will be a later optimization.

3.

Operation

When accessed from within an FPGA, this core should be simple to access: Raise the `i_wb_cyc` line at the beginning of every transaction. Set `i_wb_stb` (transaction strobe), `i_wb_we` (Write enable, true if writing or false otherwise), `i_wb_addr` (address of value), and `i_wb_data` for every transaction. You may move to the next transaction any time `i_wb_stb` is true on the same clock that `o_wb_stall` is false. Transactions will be pipelined internally. When `o_wb_ack` is true, a transaction has completed. If that transaction was a read transaction, `o_wb_data`, will also be filled with the data read from the memory device.

4.

Clocks

This design is centered around a DDR-1600 chip. In order to run this chip at speed, it requires a 200MHz clock. Xilinx recommends a 160 MHz clock for their design, so it should work at slower rates–I just don't know how much slower the design will continue to work for.

If you wish to slow down the design, adjust the parameter `CKREFI4` to be the number of clocks expected in four timse 7.8 μs.

5.

Wishbone Datasheet

Tbl. 5.1 is required by the wishbone specification, and so it is included here. The big thing to notice

Description	Specification
Revision level of wishbone	WB B4 spec
Type of interface	Slave, Read/Write, pipeline mode supported
Port size	32–bit
Port granularity	32–bit
Maximum Operand Size	32–bit
Data transfer ordering	(Irrelevant)
Clock constraints	Designed for 200MHz, DDR1600
Signal Names	Signal Name / Wishbone Equivalent: i_wb_clk — CLK_I i_wb_cyc — CYC_I i_wb_stb — STB_I i_wb_we — WE_I i_wb_addr — ADR_I i_wb_data — DAT_I o_wb_ack — ACK_O o_wb_stall — STALL_O o_wb_data — DAT_O

Table 5.1: Wishbone Datasheet

is that all accesses to the DDR3 SDRAM memory are via 32–bit reads and writes to this interface. You may also wish to note that the scope supports pipeline reading and writing, to speed up reading the results out. As a result, the memory interface speed should approach one transfer per clock once the pipeline is loaded, although there will be delays loading the pipeline.

Further, the Wishbone specification this core communicates with has been simplified in this manner: The `STB_I` signal has been constrained so that it will only be true if `CYC_I` is also true. To interface this core in an environment without this requirement, simply create the `i_wb_stb` by anding `STB_I` together with `CYC_I` before sending the strobe logic into the core.

6.

I/O Ports

The wishbone ports to this core were discussed in the last chapter, and shown in Tbl. 5.1. The rest of the I/O ports to this core are listed in Tbl. 6.1.

Port	Width	Direction	Description
i_clk_200mhz	1	Output	A 200 MHz clock input
o_ddr_reset_n	1	Output	Active low reset command to the chip
o_ddr_cke	1	Output	Clock Enable
o_ddr_cs_n	1	Output	Chip select
o_ddr_ras_n	1	Output	RAS# Command input
o_ddr_cas_n	1	Output	RAS# Command input
o_ddr_we_n	1	Output	WE# Command input
o_ddr_dqs	1	Output	True if the FPGA should drive the DQS on this clock, false otherwise. While not a DDR output, this needs to be converted to a DDR 2'b10 (if true) before it leaves the FPGA, or high impedence if false.
o_ddr_dm	3	Output	Data Mask, used to enable only those valid writes. Although a DDR output, we treat it as SDR since all transactions are 32–bits (or more).
o_ddr_odt	1	Output	On–Die–Termination bit. This will be true any time the data lines are being driven
o_ddr_bus_dir	1	Output	True if the FPGA will be driving the data bus lines during this clock, false otherwise
o_ddr_ba	3	Output	Bank Address, 0-7
o_ddr_addr	16	Output	Command address, either row or column
o_ddr_data	32	Output	The output to be sent to the chip. This will need to be bumped to DDR rates before it actually hits the chip.
i_ddr_data	32	Input	The data input from the chip. This comes in at DDR rates, and needs a Xilinx primitive to bring it from 16'bits to 32'bits.

Table 6.1: List of IO ports that are not Wishbone Related