The Speedy DDR2 Controller For FPGAs

Ray Bittner[1]
[1]Microsoft Research, Redmond, Washington, USA

Abstract – *The Speedy DDR2 controller is intended as an improvement on the Xilinx MIG controller for Virtex 5 FPGAs. Designed entirely from scratch on the ML505 development board, it achieves better performance at the same clock rate than the MIG controller while consuming comparable resources. The tight timing constraints imposed by high speed DDR2 clash with the worst case timing constraint style of FPGA design in a way that presents unique challenges. This paper discusses the primary design problems resulting from that paradox and contrasts approaches to their solution. Performance is then compared between the Speedy DDR2 controller and the MIG controller from Xilinx. The source code has been written to be more readable, maintainable and modifiable than the MIG design, and is also freely downloadable from the web.*

Keywords: FPGA, Xilinx, DDR2, Virtex 5, ML505

1 Introduction

The Xilinx MIG controller for DDR2 interfaces has existed in FPGAs for some time [1][4]. Even though the source code is provided, MIG is essentially a black box implementation that is very difficult to understand beyond a very superficial level. If the customization instructions are followed, and it still does not work, the designer is left holding the bag with a non-functional DDR2 controller. Such an experience was the motivation for the Speedy DDR2 project.

Designing a DDR2 controller in an FPGA is a non-trivial endeavor due to the problems of satisfying regimented timing relationships on a device that only provides guarantees on maximum propagation delays, with nothing promised about absolute delays. This paper discusses the major issues that arise from this dilemma and provides methods of overcoming them. While many of the methods have been seen before, Speedy DDR2 wraps them up in a package that is more easily understood than the MIG design. Speedy DDR2 is also freely downloadable from the web.

At the same time, Speedy DDR2 lives up to its name by improving on the clock for clock performance of the standard Xilinx MIG controller in all but the most ideal streaming case, where the two controllers run neck and neck. It is hoped that Speedy DDR2 can be more easily understood by the average user, will be easier to customize to a particular design environment and will provide more insight into the design of high speed DDR2 controllers in general.

2 DDR2 External Interface

The DDR2 interface consists of clock, control, and address inputs along with a bi-directional data bus. The protocol that is followed by these signals mimics earlier RAM technologies in that the user is still addressing a particular row and column of the internal RAM structure [2]. The user must issue an ACTIVATE command using the Row Address Strobe (RAS) to open a particular row of the DDR2, and then must assert the Column Address Strobe (CAS) to issue a READ or a WRITE to access a particular column within that row. The READ or WRITE will result in either a four or eight word burst, selectable by the controller at setup time. If the user accesses another column within the same row, the CAS signal may be asserted again issuing another READ or WRITE. However, if the user changes rows, the old row must be closed with a PRECHARGE command, and then the RAS and CAS steps must be performed again. Hence, it is advantageous to change rows as little as possible.

From a performance point of view, four word bursts can perform just as well as eight word bursts due to the pipelined nature of the command stream. During a transfer, data is moved on both the positive and negative edges of the clock, leading to the name Double Data Rate. This enables the completion of a four word burst in just two clock cycles.

Internally, DDR2 is actually divided into four separate banks, each with an independent notion of the open row; so that it is possible to have four different rows open at the same time. It is helpful for the controller to maintain independent control of all four rows so that four times as much memory may be accessed without the penalty of a row change.

The control signals are broken out into chip select (CS), row address strobe (RAS), column address strobe (CAS), write enable (WE), data strobe (DQS), byte write enable (DM) and a few other signals. Several data signals (DQ) are typically ganged with a single data strobe (DQS), and data timing alignment is actually maintained with respect to the strobe rather than the clock (except in a gross sense). This eases the burden on the PCB designer by allowing them the freedom of maintaining trace length matching on just the set of data signals that are ganged with a particular strobe, rather than the entire data bus.

3 Design Approach

The primary difficulties in designing a DDR2 controller in an FPGA arise from maintaining the proper timing relationships between the signals despite the fact that FPGAs generally do not offer fine grain timing control except to make guarantees about maximum propagation delays.

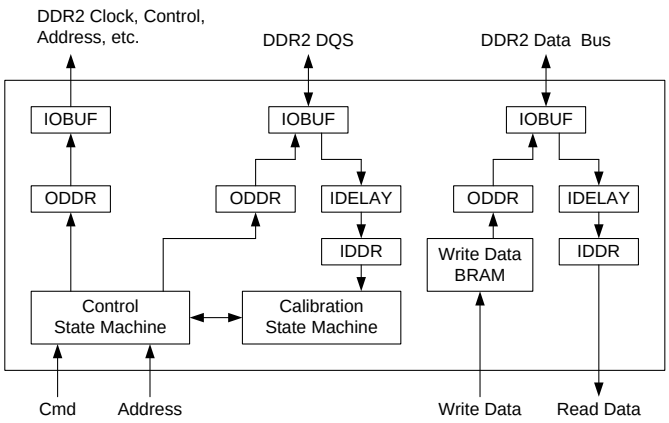

Figure 1. Speedy DDR2 Block Diagram

Figure 1 shows a high level block diagram of the Speedy DDR2 controller. The external DDR2 device control signals connect at the top of the diagram and the user connects to the bottom internal to the FPGA. The user commands are either read or write and the data buses are 256-bits each; matching the four word by 64-bit burst size on the test platform. A great number of the actual signals are not shown, but this is illustrative enough for the descriptions below.

3.1 Output Signal Timing

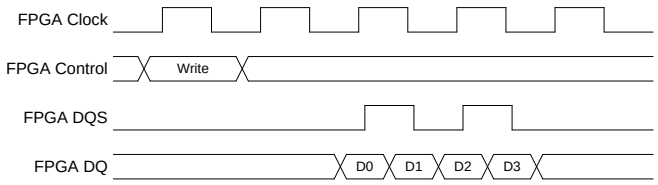

Figure 2. DDR2 Write Timing

Figure 2 illustrates the timing relationship needed for a write command as they should appear at the FPGA pin pads. All commands are issued to the DDR2 device synchronous to the positive edge of the clock. However, in the case of an external DDR device, it is necessary to use the ODDR primitive to drive the data signals in order to clock the data out on both the positive and negative edges of the clock. At the same time, the DDR2 specification demands that the edges of the data strobe signals (DQS) be center aligned with the valid data windows on the data signals (DQ). This allows the DDR2 device to use the DQS strobes as clock signals to latch the data.

The ODDR primitive introduces an unknown timing skew to the data relative to the timing path taken by a normal signal driven from the IOB output flip flop. This timing skew affects the DQS, DQ and also all control and clock signals since the clock period can be 5ns or less, which is easily within the possible skew envelop of signals leaving the chip through different device primitives. The solution to this problem is to drive all clock, DQS, DQ and other control signals from the ODDR primitive so that they all see approximately the same skew when leaving the chip and so maintain the proper timing relationship. The output data path for all of these signals becomes fabric flip flop -> ODDR -> IOBUF. Finally, the DQS strobe is center aligned with the DQ valid data window by clocking the ODDRs that drive the DQS signals from a clock that lags the normal memory clock by 90 degrees.

It is not necessary to know the exact propagation delay from the ODDR forward in the output driver chain due to the way that read capture is performed as explained later. However, it is necessary for all output signals to have approximately the same output delay. This chain of output primitives will guarantee that outcome from a logical point of view, but it is still necessary to place some timing constraints on those signal paths in the physical constraint (.UCF) file. Unlike the MIG design, the Speedy DDR2 controller does not require detailed location constraints for strict timing control. Rather, a single timing constraint was placed on all output signals of 3.8ns. This is the minimum possible output delay for that primitive chain given the Virtex 5-1 device, which effectively forces the ISE tool chain to place all output drivers in the local neighborhood of their associated pins, which turn out to have remarkably similar output propagation delays [5].

3.2 Input Signal Timing

Data returning from the DDR2 device during a read operation follows the input path. On a read, the data (DQ) and the data strobes (DQS) are both driven from the DDR2 device rather than from the FPGA. Further, the DDR2 specification changes the rules for data returning from the DDR2 device on a read so that the DQS signal is edge aligned with the transitions of the valid data windows, not center aligned as for a write. Thus the DDR2 device does not need to generate offset DQS signals and instead can simply clock them out together. It is up to the controller to locate the DQS signal relative to the clock, and from there calculate the relative position of the center of the valid data windows for sampling.

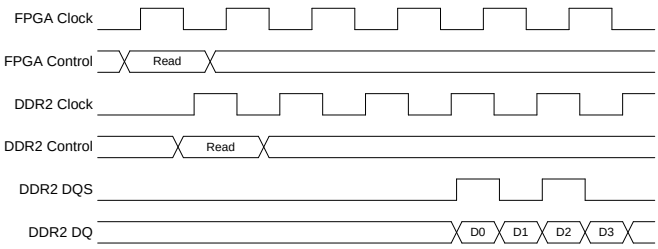

Figure 3. DDR2 Read Timing

Figure 3 shows the data transition aligned DQS signals as well as the cumulative effect of the clock/control signal output path discussed above. The FPGA signals are as seen from the internal FPGA ODDR driving flip flops, and the DDR2 signals are as seen at the FPGA IDDR inputs, after skew. The ODDR output path for the clock/control signals skews them relative to the FPGA's internal clock by an unknown amount. Additionally, the DDR2 device skews the

returned DQS/DQ signals even further; albeit by a smaller amount.

The exact delay incurred by the returning DQS/DQ signals is unknown and likely falls somewhere in between clock edges, which necessitates the use of the IDELAY primitive for accurate sub-cycle sampling of those signals. The input path for the DQS and DQ signals consists of IOBUF -> IDELAY -> IDDR -> Fabric Flip Flop. The IDELAY allows pad signals to be delayed in 78.125ps increments, up to 5ns [6]. This gives the sub-cycle resolution necessary for accurate DQS/DQ acquisition.

Similar to the output timing, the input timing paths require only a single timing constraint of 2.1ns from pad to IDDR, rather than the lengthy list of constraints that need to be customized for current MIG designs. 2.1ns is just a bit longer than the minimum possible propagation delay for that chain of primitives on the Virtex 5-1 device; forcing the tools to use the fastest local connections for all inputs and guaranteeing that they have uniform propagation delays.

Note that the possible delay range of the IDELAY puts a lower bound on the allowed DDR2 clock rate. The IDDR allows capture on both the positive and negative edges of the clock, which limits the "digital" resolution of the capture to the nearest half clock cycle. It is then up to the IDELAY to locate the transition point within that half clock cycle. Therefore, the IDELAY must be able to sweep across an entire half clock cycle in order to obtain an accurate transition location. Hence, the slowest allowable clock rate is 100MHz. In practice, the hardware attempts to place guard bands around the adjustment range so that later temperature related changes to the exact skew can be accounted for without jumping to the next positive or negative edge. For that reason, it is advisable to choose a clock rate somewhat faster than 100MHz.

3.3 Read Calibration

Since the input signal delay is unknown, the controller must search for the first rising edge of the DQS signal after a read in order to determine the exact cycle, edge and IDELAY offset where that edge is located. The search is conducted by generation of a read and then observing the outputs of the IDDR one or more cycles later while also adjusting the IDELAY offset. This process is referred to as calibration. Once the search state machine finds the correct number of cycle delays, the correct edge and the correct IDELAY offset, the controller is calibrated to receive read signals and normal operation may begin.

The calibration controller could be designed to either track the transition of the DQS signal (being representative of the delay of all of its associated DQs), or a known pattern could be written to the RAM and each individual DQ signal could be independently tracked. It was decided that only the DQS signal would be tracked, both for minimization of the hardware required, and also so that calibrations could continue to occur after initialization without requiring special patterns to be written to any part of the RAM.

Several search algorithms were experimented with in the Speedy DDR2 controller. First, several parallel state machines were used; one for each of the eight DQS signals on the test platform. Each of these employed a slow sequential search starting from the minimum expected cycle/edge/IDELAY out past the maximum expected cycle/edge/IDELAY. During initial calibration, dummy reads were generated until all eight state machines reported having a signal lock, which typically required on the order of 100 reads. This worked well, but for the sake of faster calibration both initially and at runtime, a binary search algorithm was implemented that would converge in just eight reads. The binary search worked, but it was found to add approximately 20% more resources to the entire design, and this seemed too high of a penalty for little pay back.

In practice, sequential search was fast enough and consumed far less resources. In fact, the calibration state machine was changed once again so that a single state machine was used for all eight DQS signals which are calibrated one at a time via the use of a multiplexer. That resulted in the smallest implementation and is still fast enough to keep it well calibrated.

After the initial acquisition, the calibration state machine will lock on a particular cycle/edge and will only vary the IDELAY by up to +/-0.5 clock cycle, because after that point large jumps in timing would cause complications in the read pipeline. Limiting the calibration range also avoids the problem of determining which DQS rising edge should be referenced in a long series of burst reads, as well as reducing the possible search space for the calibration controller.

This limited automatic re-calibration runs continuously during normal operation, recalibrating the IDELAY timing on each DQS signal at a user specified rate. The goal is to adjust for small timings changes that may occur due to temperature variations. Since only the DQS signal is used to perform calibration, the data is irrelevant and can remain untouched. Thus under most circumstances, any user read may also serve as a calibration reference point. As normal user reads occur, the calibration controller measures the exact position of the rising edge of each DQS signal in turn. Using sequential IDELAY adjustment, this may require as many as 64 reads per DQS signal. Once the DQS signal is located, its position is stored and new adjustments are made to the IDELAYs attached to its associated DQ signals as discussed below.

In order to guarantee that automated re-calibrations occur at a known rate, the user passes a parameter to the Speedy DDR2 controller indicating the maximum amount of time that may elapse between re-calibrations. If this time elapses and all DQS signals have not yet been re-calibrated, the Speedy DDR2 controller will pause normal user activity and insert dummy reads until a complete re-calibration is achieved on all DQS signals.

3.4 Read Data Capture

Once the timing of the DQS/DQ signals has been determined, there are two generally used methods of latching the data. The first, used by the current Xilinx MIG design, is

to delay the DQS signal by ¼ clock period and use that as a clock signal to the associated DQ IDDR primitives to latch the data [3]. The latched data is then in the "clock domain" of the DQS signal and must be latched again into the FPGA's internal clock domain, which represents extra latency in the read data path. Another problem with this method is that further calibrations of the DQS signal require complete interruption of normal user commands while the IDELAY is tweaked by the search algorithm. Lastly, this method relies on the propagation delay from the delayed DQS to all of the associated DQ IDDR flip flops to be approximately the same; since the exact capture time for each DQ IDDR will be skewed by the propagation delay difference of its clock signal.

The other method, which is used by the Speedy DDR2 controller, and which has been used by MIG in the past, is to locate the DQS signal relative to one of the edges of the FPGA's internal clock, and then delay all of the associated DQ IDDRs by and additional ¼ clock cycle. Using that method, the data is latched directly into the FPGA's internal clock domain and no retiming is necessary. Further, since the DQS signal is not actually used to latch the DQ data, the calibration state machine is free to run in parallel with user commands without disrupting the flow of user data. Lastly, there are no clock skew problems with the DQ IDDRs because they are being driven by the global clock driver used for the internal FPGA clock.

3.5 DDR2 Timing Parameters

There are a large number of timing parameters that must be satisfied in issuing commands to the DDR2 device. When designing the controller, tradeoffs could be made with regards to obtaining the absolute minimal timings in all cases, or providing a more general "catch all" timing structure that works in a worst case fashion.

The Speedy DDR2 controller maintains minimal DDR2 device timings in order to ensure the best possible performance at a given clock rate. This is achieved through the use of a set of 13 timers arranged in a functionally inverted fashion. Rather than maintain elapsed time since a given event, they track the time that must elapse before a new event of a given type may occur. This means that multiple timers must be updated when a given command is issued, but that only a single timer needs to be consulted in order to determine when the next command may proceed. Since the determination of the next command is in the critical path, and the loading of the timers is not, this turns out to be a good tradeoff.

Most interestingly, the timers are implemented as long shift registers rather than binary countdown timers. After trying both implementations, it was found that the long shift register approach resulted in the consumption of fewer resources, likely due to the register rich architecture of the Virtex 5 FPGA. The shift register implementation also resulted in better timing since the last register in the chain could be reference directly as the ready/not ready flag, rather than having to derive this from the value of a counter.

4 Design Performance/Evaluation

4.1 Speed of Speedy DDR2

The Xilinx MIG controller version 2.0 was used as a standard of reference for performance [1]. Several different tests were applied to both the MIG and Speedy controllers for comparison as shown in Table 1.

Table 1. Performance In Terms Of Millions Of Bursts Per Second At 198MHz.

	Speedy	MIG (AL=0)		MIG (AL=2)	
	Perf.	Perf.	Rel.	Perf.	Rel.
Streaming Read	95.98	95.64	0.36%	95.46	0.54%
Streaming Write	94.47	94.69	-0.23%	94.28	0.20%
Alternating Rd/Wr	17.67	13.90	27.12%	13.08	35.09%
Row Change Read	17.75	14.88	19.29%	14.87	19.37%
Row Change Write	14.97	14.02	6.78%	12.28	21.91%
Random Read	17.86	15.26	17.04%	15.26	17.04%
Random Write	17.01	14.15	20.21%	12.44	36.74%

Both the Speedy and MIG controllers were implemented on a Xilinx ML505 development board running at 198MHz using ISE 10.1 SP2. This board used a Xilinx XC5VLX50T-1FF1136 part with the standard supplied 256 MegaByte unbuffered Micron SODIMM (MT4HTF3264HY-53E). The DDR2 was programmed for bursts of four 64-bit words for each read/write operation, resulting in 32 bytes per burst transfer. The results in the table are expressed in terms of millions of bursts per second, so these could be converted to bytes per second by multiplying by 32. Four tests were applied:

- Streaming Read/Write – Increments through all possible memory addresses linearly and either reads or writes each depending on the test. This results in the minimum number of DDR2 row changes for full memory coverage and should produce the best possible performance.
- Alternating Read/Write – Increments through all possible memory addresses linearly, first writing and then reading each location back before proceeding to the next address.
- Row Change Read/Write – Skips from one DDR2 row to the next either reading or writing each in succession depending on the test. This produces the worst possible DDR2 row change behavior.
- Random Read/Write – Uses an LFSR to generate random 23-bit burst addresses in order to read or write each, depending on the test. The LFSR guarantees that all addresses except zero are visited exactly once before cycling around again. This could be a first approximation to normal microprocessor behavior where the software makes no attempt to minimize DDR2 row changes.

The columns of Table 1 show the performance of the Speedy controller for each test, followed by the performance of the MIG controller in two different configurations along with the performance of Speedy relative to MIG. The two MIG configurations are with Additive Latency (AL) set to 0

and 2, respectively. In all cases but one, the Speedy DDR2 controller exceeds the performance of the MIG controller, with the remaining case being only marginally worse.

The streaming tests represent the best possible performance because they require the fewest number of row changes in order to address all memory locations. If there were no row changes, no refresh cycles (tRFI), and no row closing timeouts (tRAS max), the ideal performance would be 198MH/2 = 99 million bursts per second, since a burst requires two clock cycles to complete. Neither controller can reach that performance due to the presence of those effects.

The streaming tests show the smallest difference in performance between the controllers, since both designs are well pipelined and differences in command latency are hidden by the sequential access pattern. The Speedy controller is slightly slower on the streaming write test because of the automated recalibration mechanism built into Speedy. All tests were run with a guaranteed read timing recalibration interval of not more than 1 millisecond, meaning that the Speedy controller will recalibrate the read timings at least 1,000 times per second. During the streaming write test, this meant that the Speedy controller periodically forced a series of dummy reads, and possibly a DDR2 row change, in order to guarantee that requirement; thus affecting performance slightly. Running the steaming write test with guaranteed recalibration set for 100 microseconds, the streaming write performance lagged behind the MIG controller by approximately 5%. It is believed that the MIG controller does not perform these recalibrations, and so no disruption of the write stream occurred for MIG. This was not an issue with the streaming read tests, since the Speedy controller is capable of opportunistically performing recalibrations on normal user read operations.

The remaining tests show a much larger gap in performance between the Speedy and MIG controllers. For greater insight as to why this is the case, a behavioral simulation was run with each of the controllers in order to measure some of the relevant timing latencies.

Table 2 shows the ideal, Speedy and MIG command latencies in terms of clock cycles at 198MHz. The ideal latency represents the minimum possible number of clock cycles that the controller would need to wait before issuing the read or write command according to the DDR2 timing constraints. For example, a read to the same row could theoretically be issued on the same clock cycle that it was received from the user, but it would need to wait a minimum of 4 clock cycles before returning any data due to the tCL parameter of the DDR2. A row changing read on the other hand must also satisfy the precharge time (tRP) and the RAS to CAS delay (tRCD) in addition to tCL before returning data. Ideal write numbers are calculated similarly, except that the write is considered complete on the cycle that the write is issued on the external DDR2 bus control signals. Hence, the ideal write latency is 0.

Of course, no controller can meet the ideal timings because of the practical limitations of propagation delays and the design of the DDR interface circuits of the FPGA. The Speedy and two MIG columns of Table 2 show these realities. Command latencies were measured from the cycle that the read or write command was accepted from the internal interface until the result appears on the external DDR2 bus. In the case of writes, this would be the write command being asserted on the external bus. In the case of reads, this would be when all data has been returned for a burst.

Overall, it can be seen that the MIG controller adds several more clock cycles of latency to these crucial timings, which are ultimately reflected in terms of performance for the non-linear addressing tests shown in Table 1. Row Change Writes, for example, would ideally require 6 clock cycles of latency, but in reality Speedy has a latency of 12 cycles for perform this operation and MIG with AL=0 requires 16 cycles. In other words, if the addressing pattern demands that a row change is required for a write, MIG will need 4 (AL=0) or 5 (AL=2) additional clock cycles to perform that operation. When the requested user operations use a linear addressing pattern this difference in latencies does not matter because it is all pipelined, but when row changes are required the extra latency shows itself in poorer performance. For some applications, one clock cycle may be subtracted from the MIG read latencies because the MIG returns the lower 16 bytes one cycle earlier than shown. The upper 16 bytes of a 32 byte burst are returned one cycle after that, matching the latencies shown in Table 2.

Interestingly, the use of Additive Latency did not help the MIG controller at all in the performance tests, and in fact this feature seems to be implemented by inserting additional pipeline stages into the MIG controller, which actually hurts it in some tests. Likely, the additional cycles of latency were needed in order to meet timing at 333MHz, as advertised by the MIG controller in the fastest Virtex 5-3 parts. The Speedy controller will pass timing analysis on the fastest Virtex 5-3 parts at speeds of up to approximately 290MHz. However, it

Table 2. DDR2 Controller Command Latencies

	Same Row Read	Same Row Write	Row Change Read	Row Change Write
Relevant Timing Parameters	tCL=4		tRP=3 tRCD=3 tCL=4	tRP=3 tRCD=3
Total Ideal	4	0	10	6
Speedy	13	6	19	12
MIG (AL=0)	18	9	25	16
MIG (AL=2)	20	9	27	17

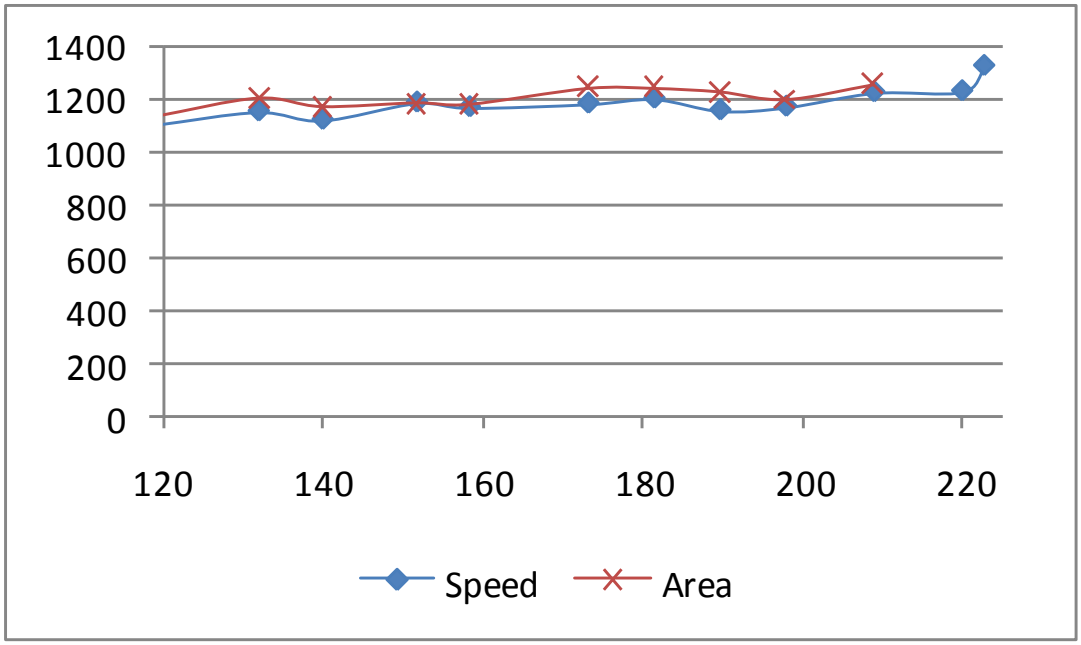

Figure 4. Speedy Slices Used vs. Clock Rate (MHz)

could not be tested at this speed given the Virtex 5-1 part on the ML505. Assuming the 20% relative clock for clock speed increase for Speedy, as shown in some of the tests, Speedy would attain an effective clock rate of 290*1.2 = 348MHz relative to the MIG, depending on the data access pattern. For pure streaming performance, MIG would still win at 333MHz compared to Speedy's 290MHz.

While it may be possible to redesign the Speedy controller to run faster, for the time being the designer is left with the trade off of higher clock rate with MIG, or the benefits of lower command latencies with the Speedy controller. Using the slowest Virtex 5-1 parts, the design process starts to become tedious at around 200MHz, when multiple extra pipeline stages are needed in order to maintain timing closure. So, this was considered to be a good tradeoff.

4.2 Size of Speedy DDR2

The relative sizes of the MIG and Speedy controllers were determined by examining section 14 of the map report in ISE.

Table 3. DDR2 Controller Resource Comparison At 198MHz

	Speedy	MIG (AL=0)	MIG (AL=2)
Slices	1175	1337	1190
Slice Regs	1530	2097	2108
LUTs	2198	1736	1720
LUTRAM	2	20	19
BRAM	4	3	3

Table 3 shows these results, although it should be mentioned that these numbers varying depending on compiler settings and target clock rate. The numbers shown reflect compilation for operation at 198MHz, and seem to indicate that the designs are comparable in size overall, depending on the metric of interest.

In order to determine the effect of clock frequency on the area of the Speedy controller, a study was performed showing the resources used in terms of slices versus clock rate in Figure 4. The compilation target for this study was the Virtex 5-1 part, which results in a maximum frequency of approximately 209MHz when compiling for area, and 222MHz when compiling for speed. As can be seen, speed/area settings and clock rate had very little effect on the overall size as measured in slices. The one notable exception was at the very high end of the compilation for speed numbers, where the size of the design grew from 1175 at 198MHz to 1330 (+13.2%) at 222MHz. Likely this was the result of the tools using replication in order to obtain that last bit of performance.

4.3 Shortcomings of Speedy DDR2

While the performance of the Speedy controller has advantages, there are some features that MIG implements that Speedy does not.

- DDR2 Only – The MIG controller supports DDR as well as DDR3.
- Additive Latency Not Supported – The MIG controller does support this feature, although it seems to be more of a penalty in the performance tests.
- Dual Rank RAMs Not Supported – These are DIMMS or SODIMMS that use more than one chip select. Again, the MIG does support this feature, and while some of the plumbing to support it is there in the Speedy controller, it is not fully implemented and so is not supported.
- No ECC Support – This is something that could be added to Speedy DDR2, but ECC is not commonly supported by

commercial DIMMs and SODIMMs and was deemed unnecessary.

- No Buffered DDR2 Support – Like ECC, buffered RAMs are not the norm, and unbuffered RAMs are much more prevalent.

5 Conclusions

The Speedy DDR2 controller is a high speed implementation of DDR2 running with minimal DDR2 timing parameters and a minimum of internal latency in the processing pipeline. It has been shown to be competitive with or better than the Xilinx standard MIG design at a clock rate of 198MHz, and the ISE tools indicate that it can run as fast as 290MHz on a Virtex 5-3 device, while remaining approximately equal to the MIG design in device resource requirements.

A detailed discussion of the primary design challenges arising during the implementation of the Speedy DDR2 controller has been given, along with practical solutions. It is hoped that these insights and the design itself can be useful to others facing similar problems, and/or who would like an alternative to the MIG controller. The associated ISE project and source code may be downloaded from:

http://research.microsoft.com/people/raybit

6 References

[1] http://www.xilinx.com/memorycorner, Xilinx Corporation.

[2] Bruce Jacob, Spencer W. Ng, David T. Wang. "Memory Systems: Cache, DRAM, Disk". Morgan Kaufmann, 2008.

[3] Adrian Cosoroaba. "Memory Interfaces Made Easy with Xilinx FPGAs and the Memory Interface Generator". Xilinx Corporation, white paper 260, February 16, 2007.

[4] "Xilinx Memory Interface Generator (MIG) User Guide". Xilinx Corporation, user guide 86, October 2, 2008.

[5] "Virtex-5 FPGA Data Sheet: DC and Switching Characteristics". Xilinx Corporation, data sheet 202, February 6, 2009.

[6] "Virtex-5 FPGA User Guide". Xilinx Corporation, user guide 190, January 9, 2009.