DDR3 Moving Average Sample

February 6, 2013

Contents

1 Overview

The purpose of this sample is to demonstrate the proper way for a user to interact with the DDR3 systems on the M-Series modules, both from the firmware and from the host software. This sample shows the user how to write or read the DDR3 systems from software via the streaming model, and it also shows how to write to and read from the DDR3 system from a module instantiated in the FPGA fabric.

This document begins by describing the test logic implemented on the FPGA to write and read the DDR3 system. Next, it explains the test that the software runs and the simulation setup that is designed to mirror the test software. Lastly, it provides an example output of the distributed DDR3 sample.

2 Firmware

The test module that has been implemented, which is called Pico_Moving_Average.v, is designed to demonstrate the reading and writing capability of the DDR3 system on the FPGA. It implements a smoothing filter whose data starts and ends in the DDR3 memory. Before the firmware processes any data, the software should load the data to be filtered into the DDR3 memory via the WriteRam command. The firmware waits for a "Start" command from the input stream, which contains a read address, a write address, and a data size, as shown in the following table.

Bit Range	Description
95:64	Data Size (<= 4kB)
63:32	Write Address
31:0	Read Address

Table 1: Command Data Format

Note that the read address and write address are specified in terms of bytes.
The firmware reads data from the read address location, and shifts the lower 32 bits of the read data through a 4-entry shift register. The entries of the shift register are averaged, and that average data is written to the DDR3 memory at the address specified on the input stream. Note that some signals for the write and read ports are assigned values that are set in axi_defines.v, which is a header file containing definitions of the AXI protocol.

2.1 Setup

Here is what our PicoDefines.v file looks like for this sample for the M–501.

```
`define USER_MODULE_NAME Pico_Moving_Average

`define PICO_DDR3
`define PICO_1_AXI_MASTERS
`define STREAM1_IN_WIDTH 128

// We define the type of FPGA and card we're using.
`define PICO_MODEL_M501
`define PICO_FPGA_LX240T

`include "BasePicoDefines.v"
```

Notice that we have specified that this design will use the DDR3 interface by defining PICO_DDR3. On the next line, we specify the number of ports that we want to be made available to our user logic (1 in this example). Recall from the DDR3 documentation that when the DDR3 system is included in a design, 1 port is implicit for streaming data to and from the DDR3 via the PCIe.

2.2 Streaming Data

As described in the PicoDDR3 documentation, a module is instantiated in the firmware to enable streaming data between the host and each DDR3 system on the FPGA card. This sample demon-

strates the proper use of the data streaming to and from the FPGA. On the M–503, the two 4 GB DDR3 systems have independent streams for streaming data to and from the host system.

2.3 Reading

To read from the DDR3 memory, some values of the AXI protocol may remain constant, and some must vary when reading different amounts of data from varying addresses in DRAM. The signals that may remain constant are shown below. Of the signals, note that only the arsize changes between the different M-Series modules. This is because of the different AXI data bus widths.

```
assign c0_s1_axi_arid       = 0;
assign c0_s1_axi_arlock     = `NORMAL_ACCESS;
assign c0_s1_axi_arcache    = `NON_CACHE_NON_BUFFER;
assign c0_s1_axi_arprot     = `DATA_SECURE_NORMAL;
assign c0_s1_axi_arburst    = `INCREMENTING;
assign c0_s1_axi_arqos      = `NOT_QOS_PARTICIPANT;

`ifdef PICO_MODEL_M501
assign c0_s1_axi_arsize     = `SIXTEEN_BYTES;
`else
assign c0_s1_axi_arsize     = `THIRTY_TWO_BYTES;
`endif
```

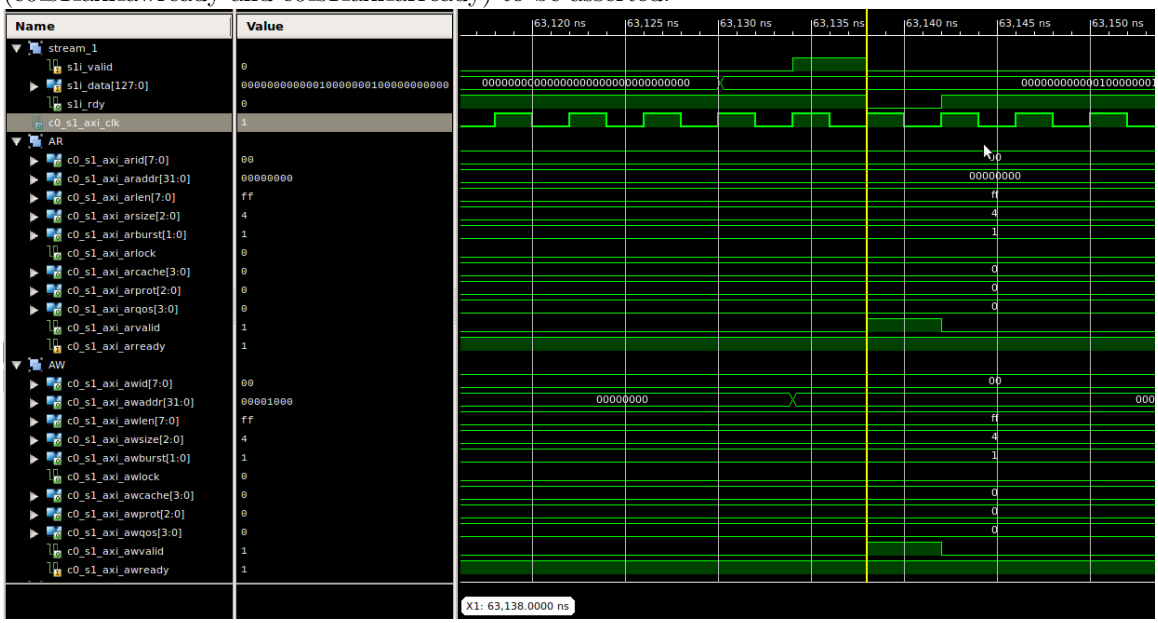
To initiate a read, the user must set the remaining variables of the AXI protocol, which may change for subsequent reads. In this case, we get the read address and length from the input stream, as shown below. Once again, notice the only difference between the M-Series modules is the length of the data to read. This is because the command specifies a number of bytes to read, and since the width of the AXI data bus is different on each module, each module requires a different number of transactions to read or write the specified number of bytes.

```
// address and length for the read from the command stream
always @ (posedge clk) begin
    if (rst_q) begin
        c0_s1_axi_araddr    <= 0;
        c0_s1_axi_arlen     <= 0;
    end else if (loadArAddr) begin
        c0_s1_axi_araddr    <= s1i_data[31:0];
        c0_s1_axi_arlen     <= (s1i_data[95:64] >> c0_s1_axi_arsize) - 1;
    end
end
```

Once the address and length have been set, the firmware asserts the valid signal on the read address port. The read is initiated when the ready signal is asserted by the AXI interconnect on the read address port.

A screen shot of the command from the input stream along with the read and write address bursts can be seen in the following figure. The command is sent on stream 1, with the read address (0x0), write address (0x1000), and the filtering length (0x1000). Once the command is received on the input stream, the address and length are loaded into the write and read address ports. These ports act independently, and they begin the write and read bursts by asserting their respective valid signals (c0_s1_axi_awvalid and c0_s1_axi_arvalid) and waiting for the respective ready signals

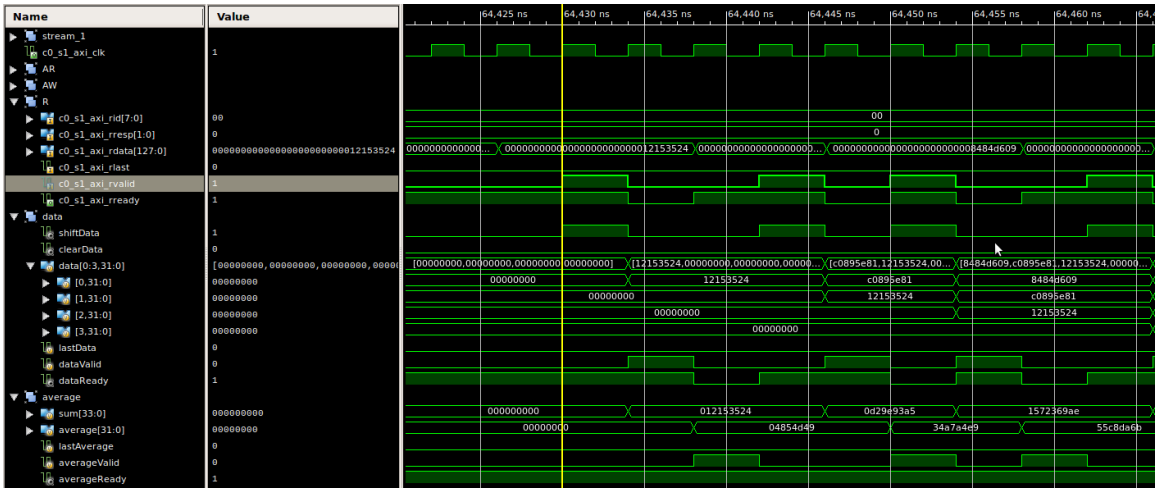
(c0_s1_axi_awready and c0_s1_axi_arready) to be asserted.



2.4 Averaging

The smoothing filter is implemented by averaging four consecutive data values. The values are shifted through a shift register. Every cycle, the average of the contents of the shift register is computed, but it is only written to DRAM if the first shift register entries contains a valid entry. To clarify, the first value written to the DRAM will be the one-fourth the value of the first entry, since the shift register is cleared between sets of data to be filtered. The size of filtered data written to DRAM is equal to the size of input data read from DRAM.

A screen shot of data being read from the read data port and shifted into the shift register can be seen in the following figure. Data is valid on the read data port when c0_s1_axi_rvalid is asserted, but the transaction occurs when c0_s1_axi_rready is also asserted by the user logic. When data is read from the read data port, it is placed into the shift register, which is under the *data* heading in the following image. Data is only valid and ready to be written into the DDR memory when the dataValid signal is asserted (which means the first entry of the shift register holds valid data that has not yet been written to memory).



2.5 Writing

To write to DRAM, the firmware first sends the address and length (along with some constants) to the DDR3 on the write address port. The contant signals are shown below. Once again, the only difference between the M-Series modules is the width of the AXI data bus.

```
assign c0_s1_axi_awid       = 0;
assign c0_s1_axi_awlock     = `NORMAL_ACCESS;
assign c0_s1_axi_awcache    = `NON_CACHE_NON_BUFFER;
assign c0_s1_axi_awprot     = `DATA_SECURE_NORMAL;
assign c0_s1_axi_awburst    = `INCREMENTING;
assign c0_s1_axi_awqos      = `NOT_QOS_PARTICIPANT;

`ifdef PICO_MODEL_M501
assign c0_s1_axi_awsize     = `SIXTEEN_BYTES;
`else
assign c0_s1_axi_awsize     = `THIRTY_TWO_BYTES;
`endif
```

Before initiating the write, the firmware must set the write address and length, which are loaded into registers from the input stream. Once again, notice the only difference between the M-Series modules is the length of the data to write, which is due to the different size in AXI data bus.

```
// address and length for the write from the command stream
always @ (posedge clk) begin
    if (rst_q) begin
        c0_s1_axi_awaddr    <= 0;
        c0_s1_axi_awlen     <= 0;
    end else if (loadAwAddr) begin
        c0_s1_axi_awaddr    <= s1i_data[63:32];
        c0_s1_axi_awlen     <= (s1i_data[95:64] >> c0_s1_axi_awsize) - 1;
    end
end
```

Once the address and length have been loaded from the input stream, the firmware asserts the write address valid signal to initiate the write burst. The write address and length (along with the

other constant signals) are received by the AXI interconnect when the write address ready signal is asserted by the interconnect.

After sending the write address and length, the firmware sends the filtered data to the DRAM. When filtered data is ready to be written, the firmware asserts the write data ready signal, which is transferred to the AXI interconnect when the write data ready signal is also asserted (by the AXI interconnect). The firmware asserts the "wlast" signal along with the last piece of write data for a data set.

The following figure is a simulation screenshot showing the write data transactions, when valid data is being written into the DDR memory. Under the *average* heading in the image, the sum is the combinational summation of the values currently in the shift register. Average (under the same heading) is the registered value of sum divided by 4. The average is ready to be written to the DDR when averageValid is asserted. On the write data port, to write to the DDR, c0_s1_axi_wvalid is asserted when averageValid is asserted, and the data is sent to the DDR for the write when c0_s1_axi_wready is also asserted.

3 Software

The software for the DDR3 sample is designed to test the smoothing filter system. It creates a random data set and streams that data to the DDR3 memory via the WriteRam command.

```
// only LS word of each 128-bit entry is included in the data to be filtered
for(int i=0; i<4096/16; i++){
    inputData[4*i+3] = inputData[4*i+2] = inputData[4*i+1] = 0;
    inputData[4*i] = rand();
}

// write to DDR address 0
printf("Writing 4096B of data to the DDR\n");
err = pico->WriteRam(0, inputData, 4096, PICO_DDR3_0);
```

The software then streams the read address, write address, and data size to stream 1 of the FPGA.

```
// filter 4096 B from address 0 to address 4096
printf("Sending command to begin moving average computation\n");
uint32_t cmd[4];
```

```
cmd[3] = cmd[0] = 0;
cmd[2] = cmd[1] = 4096;
err = pico->WriteStream(stream, cmd, 16);
```

After allowing the smoothing filter to complete, the software reads the result out of the DDR3 memory via the ReadRam command and compares the filtered data from the FPGA to the expected filtered data.

```
// read from DDR address 4096
printf("Reading 4096B of averaged data from the DDR\n");
err = pico->ReadRam(4096, actualAverage, 4096, PICO_DDR3_0);
```

4 Simulation

The simulation testbench, called PicoTestbench.v, is written to mirror the function of the distributed software. The simulation begins by creating 256 bytes of random data and streaming that to the FPGA, which occurs once the DRAM has completed its calibration phase (∼60us).

```
// stream 4kB of data to the memory at address 0
for(i=0; i<4096; i=i+16) begin
    inputData[i/16] = $random & 32'hFFFFFFFF;
    PicoSim.PicoLoadBuffer128(i, inputData[i/16]);
end
PicoSim.WriteRam(0, 4096, `PICO_DDR3_0);
```

Next the simulation sends a command to stream 1 on the FPGA to begin the smoothing filter.

```
// send command to compute moving average of 4kB of data from address 0 to
   address 4kB
PicoSim.PicoLoadBuffer128(0, {32'h0, 32'd4096, 32'd4096, 32'h0});
PicoSim.WriteStream(1, 0, 16);
```

After the filter has completed, the simulation reads the contents of the DDR3 memory and compares the computed data set to the expected filtered data.

```
// read moving average data from DDR
PicoSim.ReadRam(4096, 4096, `PICO_DDR3_0);
for(i=0; i<4096; i=i+16) begin
    actualAverage[i/16] = PicoSim.PicoReadBuffer128(i);
end
```

5 Sample Output

Below is an example of the expected output when running the distributed DDR3_MovingAverage sample software with the M–501 bitfile.

```
user@pico:~/pico/samples/DDR3_MovingAverage/software$ ./ddr3_moving_average
    ../firmware/M501_LX240T_DDR3.bit
Loading FPGA with '../firmware/M501_LX240T_DDR3.bit' ...
Writing 4096B of data to the DDR
```

```
Sending command to begin moving average computation
Reading 4096B of averaged data from the DDR
Comparing the computed moving average data to the expected moving average
All tests successful!
```