M–503 QDR-II 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 three 72 Mb Quad Data Rate (QDR) II chips from the firmware on the M–503 module. This document begins by describing the test logic implemented on the FPGA to write and read the QDR systems. 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 M–503 QDR sample.

2 Firmware

First, to enable the QDR systems in the firmware, simply add the following line to the PicoDefines.v file:

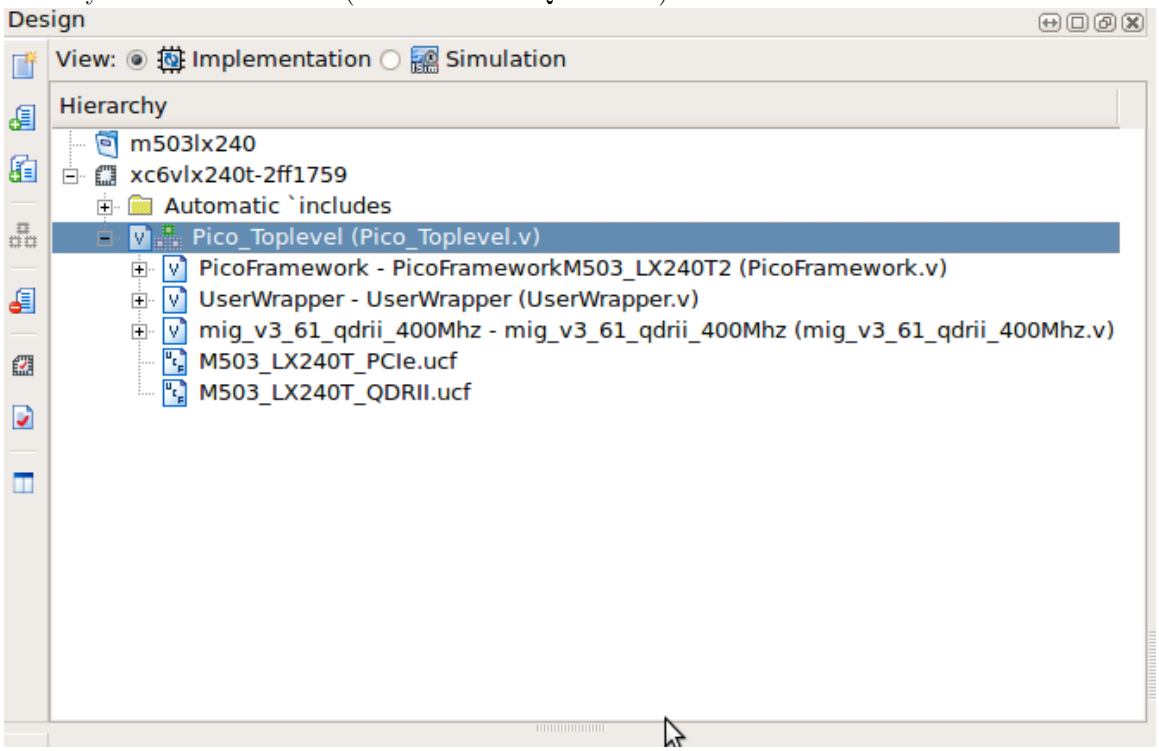
```
`define PICO_QDRII
```

The test module for the QDR system, which is called Pico_QDRII_test.v (also specified in PicoDefines.v), instantiates three copies of the Xilinx traffic generator; one for each of the QDR-II chips on the M–503. This traffic generator was generated by Coregen during the creation of the Xilinx QDR-II Memory Interface Generator (MIG). The traffic generator tests the writing and reading capability

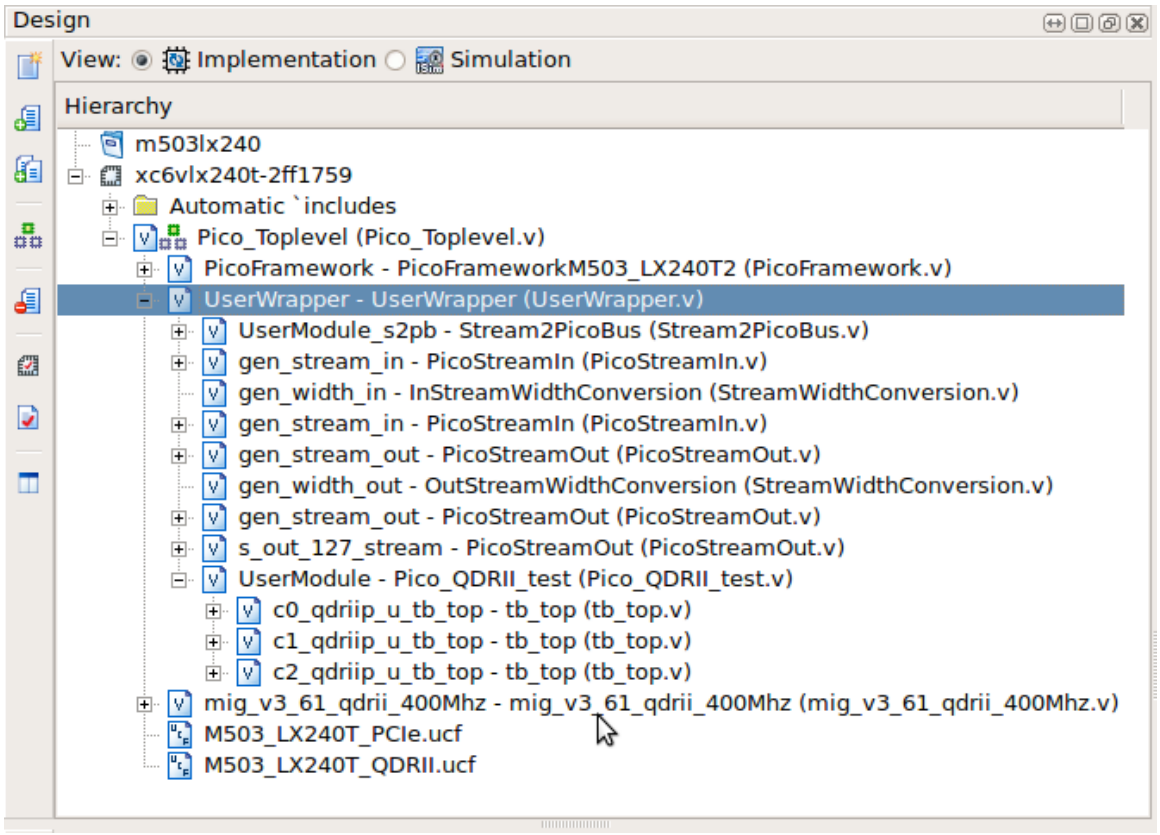
of the QDR system by writing data from a counter to sequential addresses in the SRAM. The test module compares read data to the expected data, and it reports any errors that were detected via the PicoBus.

2.1 System Hierarchy

The top level of the M–503 QDR sample is the Pico_Toplevel module. It instantiates the PicoFramework, the UserWrapper, and the mig_v3_61_qdrii_400Mhz module. The project also contains two User Constraints Files (UCF): one constraining the clock systems and the PCIe systems (M503_LX240T_PCIe.ucf) and one containing pin placements and clock constraints for the Xilinx Memory Interface Generator (M503_LX240T_QDRII.ucf).



The UserWrapper module provides an interface to user logic in order to read and write the QDR systems. The three QDR-II chips are driven by a single MIG, so the three QDR-II interfaces from the user logic are fed into the MIG. Lastly, the UserWrapper instantiates the UserModule, which in this example is the Pico_QDRII_test module.



Pico_QDRII_test instantiates one test module produced by Xilinx (tb_top.v) for each of the three QDR-II chips on the M–503. These test modules are automatically enabled after the QDR-II chips complete calibration, and they are configured to only read, only write, or simultaneously read and write from a QDR-II chip. The test modules write incrementing data to incrementing addresses, and they also report any write or read errors that occur via an error flag. The status signals for the QDR test modules can be accessed via the PicoBus at the following addresses.

Address	Description
0x12350000	control and calibration status register
0x12350020	error flags register
0x12350040	read data for QDR-II 0
0x12350060	read data for QDR-II 1
0x12350080	read data for QDR-II 2

Table 1: PicoBus QDR Addresses

2.2 Test Registers

The QDR control register is a readable and writeable register that enables the user to reset the QDR system and read the QDR state information. The only writeable bit of the register is the QDR system reset signal, but all status bits are readable. To reset the QDR system, the user should

write the reset signal to a 1 and then release the reset by writing that same bit to a 0. Once the calibration has completed, each of the QDR-II chip's calibration signals should be asserted. Users can verify that they are actually reading the QDR control register correctly by checking that the signature field matches the signature shown in the control register table.

Bit	R/W	Description
0	R	QDR-II 0 calibration completed
16	R	QDR-II 1 calibration completed
32	R	QDR-II 2 calibration completed
64	R/W	QDR-II system reset
127:96	R	QDR-II signature = 0x55AA55AA

Table 2: QDR Control Register

The QDR error flags register is a read-only register that shows the status of the error flags from each of the test modules. The error flags are reset with the QDR system reset, and they are asserted by the test modules if an error is detected. The cX_qdriip_cmd_err signal shows if the test module detects an error with the current data, whereas the cX_qdriip_error signal shows if the test module has detected an error since reset.

Bit	R/W	Description
0	R	QDR-II 0 current error state
16	R	QDR-II 0 cumulative error
32	R	QDR-II 1 current error state
48	R	QDR-II 1 cumulative error
64	R	QDR-II 2 current error state
80	R	QDR-II 2 cumulative error

Table 3: QDR Error Flags Register

3 Software

The software for the M–503 QDR-II sample is designed to enable the test modules and verify that no errors occurred during the test run. The software first resets the QDR-II systems by writing a status register via the PicoBus, which in turn causes the MIG for the three QDR-II chips to begin calibration.

```
    // reset the QDRII system
    ResetQDRII(pico);
```

The software then polls the QDR status register to see when all the QDR-II chips have been calibrated.

```
// wait for the QDRII system to finish calibration
WaitQDRIICalibration(pico);
```

After calibration has completed, the software lets the test modules run and prints the current read data to standard output for each test module every 400 milliseconds.

```
// grab the start time
time(&startTime);
time(&endTime);

// read from the QDRII for TEST_MEM_S seconds
while (difftime(endTime, startTime) < TEST_MEM_S){

    // read from each QDRII system each iteration
    for(int i=0; i<NUM_QDRII; i++){
        err = pico->ReadDeviceAbsolute(QDRII_DATA_ADDR+i*QDRII_DATA_INCR
            , cmd, 16);
        if (err < 0){
            fprintf(stderr, "ReadDeviceAbsolute error: %s\n",
                PicoErrors_FullError(err, ibuf, sizeof(ibuf)));
            exit(1);
        }else{
            printf(" : %08X_%08X_%08X", cmd[2], cmd[1], cmd[0]);
        }
    }
    printf("\n");

    // sleep for 400 ms
    Sleep(400);

    // update the elapsed time
    time(&endTime);
}
```

After a specified number of seconds, the software checks the error flags. If any of the error flags are asserted after the runtime, the test fails. Otherwise, the test is considered successful and a success message is printed to the standard output.

```
// check the error flags //
err = pico->ReadDeviceAbsolute(QDRII_ERR_ADDR, cmd, 16);
if (err < 0){
    fprintf(stderr, "ReadDeviceAbsolute error: %s\n",
        PicoErrors_FullError(err, ibuf, sizeof(ibuf)));
    exit(1);
}
for(int i=0; i<NUM_QDRII; i++){
    if(cmd[i]!=0){
        fprintf(stderr, "Error found when testing QDRII %i\n", i);
        exit(1);
    }
}
```

4 Simulation

The simulation testbench, called PicoTestbench.v, is written to mirror the function of the distributed software test. The simulation begins by resetting the QDR-II system and waits for the calibration to complete, which should occur after approximately 35 microseconds of simulation time.

```
    // calibration completed signals are 16-bit aligned
    reg [15:0]  cal_done [0:NUM_QDRII];

    // only exit the calibration phase when all QDRII chips have been
        calibrated
    assign all_cal_complete = cal_done[2][0] & cal_done[1][0] & cal_done
        [0][0];

    initial begin

        // reset all QDRII systems
        PicoSim.WritePicoBus(QDRII_CTRL_ADDR, {64'h1, 64'h0});
        PicoSim.WritePicoBus(QDRII_CTRL_ADDR, 128'h0);

        // wait for calibration to complete on all QDRII systems
        for (i=0; all_cal_complete!==1; i=i+1) begin
            PicoSim.ReadPicoBus(QDRII_CTRL_ADDR, 16);
            {junk, cal_done[2], cal_done[1], cal_done[0]} = PicoSim.
                PicoReadBuffer128(0);
        end
```

Instead of continually reading data from the QDR registers, the simulation lets the test modules run for the specified runtime without any reading via the PicoBus. At the end of the test, the simulation reads random data from each of the QDR-II chips, and then it checks the error flags in the QDR status register.

```
        // let the test run for a specified number of nanoseconds
        if (NUM_QDRII > 0) begin
            #TEST_RUNTIME_NS;
        end

        // read some data from each QDRII system
        for (i=0; i<NUM_QDRII; i=i+1) begin
            PicoSim.ReadPicoBus(QDRII_DATA_ADDR + i*QDRII_DATA_INCR, 16);
        end

        // read the error flags
        PicoSim.ReadPicoBus(QDRII_ERR_ADDR, 16);
        if (PicoSim.PicoReadBuffer128(0) != 128'h0) begin
            $display("Error in traffic generation module! Error flags: %h\n
                ", PicoSim.PicoReadBuffer128(0));
            $stop;
        end
```

5 Sample Output

Below is an example of the expected output when running the distributed M–503 QDR-II sample software.

```
user@Pico:~/pico/samples/M503_QDRII/software$ ./qdrii ../firmware/
    M503_LX240T_QDRII.bit
Loading FPGA with '../firmware/M503_LX240T_QDRII.bit' ...

Resetting the QDRII system...
Waiting for QDRII system to finish calibration...

Testing QDRII for 10 seconds...
Elapsed time :        QDRII 0 Data        :        QDRII 1 Data        :
          QDRII 2 Data
    0.000000 : 00000032_994C8603_2190CA61 : 000000E6_F3799CCE_67339BCD :
        0000006D_B6DB4DA6_D369B6DB
    0.000000 : 00000078_BC5E0F07_83C1E2F1 : 0000002B_95CAC562_B158AE57 :
        000000B3_D9ECD66B_359ACF67
    0.000000 : 000000ED_F6FB5DAE_D76BB7DB : 000000A2_D168944A_25128B45 :
        000000C3_E1F0D86C_361B0F87
    1.000000 : 000000CA_E572994C_A6532B91 : 00000019_8CC64321_90C86633 :
        000000D3_E9F4DA6D_369B4FA7
    1.000000 : 000000DB_EDF6DB6D_B6DB6FB7 : 00000028_944A0502_8140A251 :
        000000E3_F1F8DC6E_371B8FC7
    2.000000 : 0000008B_C5E2D168_B45A2F17 : 000000D3_E974DA4D_369B4FA5 :
        000000BF_DFEFD7EB_F5FAFF7F
    2.000000 : 000000F9_FCFE5F2F_97CBE7F3 : 000000AF_D7EBD5EA_F57ABF5F :
        000000D9_ECF65B2D_96CB67B3
    2.000000 : 0000003C_9ECF47A3_D1E8F67B : 000000F0_F87C1E0F_0783C3E1 :
        00000012_89448241_20904A25
    3.000000 : 000000B4_DA6D168B_45A2D369 : 00000067_B3D9CCE6_73399ECF :
        000000BC_DE6F178B_C5E2F379
    3.000000 : 000000F6_FB7D9ECF_67B3DBED : 000000AB_D5EAD56A_B55AAF57 :
        000000FE_FF7F9FCF_E7F3FBFD
    4.000000 : 000000D6_EB759ACD_66B35BAD : 000000EF_F7FADDEF_F77BBFDF :
        00000042_A1508844_22110A85
    4.000000 : 000000E7_F3F9DCEE_773B9FCF : 0000000B_85C2C160_B0582E17 :
        000000ED_F6FB5DAE_D76BB7DB
    4.000000 : 000000C5_E2F158AC_562B178B : 000000E0_F0781C0E_070383C1 :
        00000096_CAE492E9_74BA5F2F
    5.000000 : 00000009_84420100_80402211 : 00000023_91C8C462_31188E47 :
        000000A8_D46A150A_8542A351
    5.000000 : 0000007E_BF5F8FC7_E3F1FAFD : 000000CC_E673198C_C6633399 :
        00000021_90C84422_11088643
    6.000000 : 00000028_94CA4522_9148A653 : 0000007C_BCDE0F07_83E1E2F1 :
        00000063_B158CC46_23118AC5
    6.000000 : 00000038_9C4E0703_81C0E271 : 000000B8_DCEE976B_85D2E371 :
        0000000E_874381C0_E0703A1D
    6.000000 : 000000E2_F1789C4E_27138BC5 : 00000098_CC661309_84C26331 :
        000000B8_DC6E170B_85C2E371
    7.000000 : 0000005E_AF578BC5_E2F17ABD : 0000000F_87C3C1E0_F0783E1F :
        00000062_B1588C46_23118AC5
    7.000000 : 00000038_9C4E0703_81C0E271 : 000000BB_DDEED76B_B5DAEF77 :
        0000000E_874381C0_E0703A1D
```

```
    8.000000 : 0000004C_A6530984_C2613299 : 000000FF_FFFFDFEF_F7FBFFFF :
        000000B7_DBEDD6EB_75BADF6F
    8.000000 : 0000005B_AD568B45_A2D16AB5 : 0000000F_87C3C1E0_F0783E1F :
        000000FA_FD7E9F4F_A7D3EBF5
    8.000000 : 000000CF_E7F3D9EC_F67B3F9F : 000000B8_DC6E170B_85C2E371 :
        000000A3_D1E8D46A_351A8F47
    9.000000 : 00000012_89448201_20904A21 : 00000060_B0580C06_030182C1 :
        0000004C_A6530984_C2613299
    9.000000 : 000000EE_F77B9DCE_E773BBDD : 000000D7_EBF5DAED_76BB5FAF :
        00000028_944A0502_8140A251
Checking QDRII error flags...

QDRII test successful
```