[redacted]

Assignmen[redacted]

The modified multipler_mem code is shown below, with the dual-port RAM implemented:

multiplier_mem.v

```
module multiplier_mem (
    input clr,
    input clk_wr1,
    input clk_wr2,
    input wr_en1,
    input wr_en2,
            input wr_en_bram,
            input rd_en_bram,
    input clk_rd,
    input [15:0] a,
    input [15:0] b,
            input [7:0] addr_rd,
            input [7:0] addr_wr,
    output [31:0] qout
);

wire [15:0] a_q, b_q;
wire [31:0] m;
wire full1, full2, empty1, empty2;
reg  read_req;

// instantiate FIFOs
fifo fifo_inst1 (
    .data(a),
    .rdclk(clk_rd),
    .rdreq(read_req),
    .wrclk(clk_wr1),
    .wrreq(wr_en1),
    .q(a_q),
    .rdempty(empty1),
    .wrfull(full1)
);

fifo fifo_inst2 (
    .data(b),
    .rdclk(clk_rd),
    .rdreq(read_req),
    .wrclk(clk_wr2),
    .wrreq(wr_en2),
```

```
    .q(b_q),
    .rdempty(empty2),
    .wrfull(full2)
);

// instantiate multiplier
qlpm_mult qlpm_mult_inst (
    .clock(clk_rd),
    .dataa(a_q),
    .datab(b_q),
    .result(m)
);

// instantiate ram
ram ram_inst (
    .data(m),
    .rd_aclr(clr),
    .rdaddress(addr_rd),
    .rdclock(clk_rd),
    .rden(rd_en_bram),
    .wraddress(addr_wr),
    .wrclock(clk_wr1),
    .wren(wr_en_bram),
    .q(qout)
);

// Read clock
always @(posedge clk_rd) begin
    read_req <= ~empty1 & ~empty2;
end

endmodule
```

The testbench written for this code is shown below, including the tcl file, the Verilog file, and the wave.do file:

testbench.tcl

```
quit -sim
vlib work;
vlog ../../*.v
vlog *.v
vsim work.testbench -Lf 220model_ver -Lf altera_mf_ver -Lf verilog
do wave.do
run 2 ms
```

testbench.v

```
/*
 [redacted]
 *
 * ECED 4260
 [redacted]
 * Verilog testbench
 */

`timescale 1ns / 1ps

module testbench ( );
    // Declare reg signals for inputs to the design under test
    reg clr = 1;
    reg clk_wr1 = 0;
    reg clk_wr2 = 0;
    reg wr_en1 = 0;
    reg wr_en2 = 0;
    reg wr_en_bram = 0;
    reg rd_en_bram = 0;
    reg clk_rd = 0;
    reg [15:0] a = 0;
    reg [15:0] b = 0;
    reg [7:0] addr_rd = 0;
    reg [7:0] addr_wr = 0;

    // Declare wire signals for outputs from the design under test
    wire [31:0] qout;

    // Instantiate the design under test
    multiplier_mem DUT(
        .clr(clr),
        .clk_wr1(clk_wr1),
        .clk_wr2(clk_wr2),
        .wr_en1(wr_en1),
        .wr_en2(wr_en2),
        .wr_en_bram(wr_en_bram),
        .rd_en_bram(rd_en_bram),
        .clk_rd(clk_rd),
        .a(a),
        .b(b),
        .addr_rd(addr_rd),
        .addr_wr(addr_wr),
        .qout(qout)
```

```
    );

    // Assign values to the DUT inputs at various simulation times
    always
    begin
        clk_wr1 <= 0;
        clk_wr2 <= 1;
            #5;
        clk_wr1 <= 1;
        clk_wr2 <= 0;
        #5;
    end

    always
    begin
        clk_rd <= 1;
        #2.5;
        clk_rd <= 0;
        #2.5;
    end

    always
    begin
        #10;
        clr <= 0;
        #100;
        while(1)
        begin
            wr_en_bram <= 1;
            wr_en1 <= 1;
            addr_wr <= addr_wr + 1;
            a <= a + 1;
            #10;
            wr_en1 <= 0;
            wr_en2 <= 1;
            b <= b + 1;
            #10;
            rd_en_bram <= 1;
            addr_rd <= addr_rd + 1;
            wr_en2 <= 0;
        end
    end
endmodule
```

wave.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -label clr -radix binary /testbench/DUT/clr
add wave -noupdate -divider clocks
add wave -noupdate -label clk_wr1 -radix binary /testbench/DUT/clk_wr1
add wave -noupdate -label clk_wr2 -radix binary /testbench/DUT/clk_wr2
add wave -noupdate -label clk_rd -radix binary /testbench/DUT/clk_rd
add wave -noupdate -divider enablers
add wave -noupdate -label wr_en1 -radix binary /testbench/DUT/wr_en1
add wave -noupdate -label wr_en2 -radix binary /testbench/DUT/wr_en2
add wave -noupdate -label rd_en_bram -radix binary /testbench/DUT/rd_en_bram
add wave -noupdate -label wr_en_bram -radix binary /testbench/DUT/wr_en_bram
add wave -noupdate -divider inputs
add wave -noupdate -label a -radix unsigned /testbench/DUT/a
add wave -noupdate -label b -radix unsigned /testbench/DUT/b
add wave -noupdate -label a_q -radix unsigned /testbench/DUT/a_q
add wave -noupdate -label b_q -radix unsigned /testbench/DUT/b_q
add wave -noupdate -divider outputs
add wave -noupdate -label m -radix unsigned /testbench/DUT/m
add wave -noupdate -label qout -radix unsigned /testbench/DUT/qout
add wave -noupdate -divider addresses
add wave -noupdate -label addr_rd -radix unsigned /testbench/DUT/addr_rd
add wave -noupdate -label addr_wr -radix unsigned /testbench/DUT/addr_wr
add wave -noupdate -label empty1 -radix binary /testbench/DUT/empty1
add wave -noupdate -label empty2 -radix binary /testbench/DUT/empty2
add wave -noupdate -label full1 -radix binary /testbench/DUT/full1
add wave -noupdate -label full2 -radix binary /testbench/DUT/full2
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 2} {295281 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 150
configure wave -valuecolwidth 66
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ns
update
WaveRestoreZoom {0 ps} {295599 ps}
```

Finally, a couple snapshots of the simulation results for the testbench are shown below: