Computer Architecture

Lab 1: Integer Multiply/Divide Unit

Author:313551151 吳承鉞

Introduction:

We conducted 32bits iterative multiplier/divider in verilog, communicating with test source and test sink module through val/rdy interfaces. Each of the computation is done in 33 cycles, with 32 cycles for computation and 1 cycle for val/rdy interfaces.

Designs:

```
module imuldiv_IntDivIterativeCtrl
(
 input        clk,        // Clock signal
 input        reset,       // Reset signal
 input        divreq_val,  // Request valid signal
 input        divresp_rdy, // Response ready signal
 output reg    divreq_rdy,
 output reg    divresp_val,
 output reg    a_mux_sel,   // Select for A register mux
 output reg    b_mux_sel,   // Select for B register mux
 output reg    result_mux_sel, // Select for result register mux
 output       sign_en
);
 assign sign_en = ((counter_reg==5'd31) && (counter_en==1'b0));
 reg counter_en;
 reg [4:0] counter_reg;
 always @(posedge clk or posedge reset) begin
  if (counter_en) begin
   // Decrement the counter during the iterative process
   counter_reg <= counter_reg - 1;
  end
  if (reset) begin
   // Reset counter to default value
   counter_reg <= 5'd31;
   a_mux_sel     <= 1'b0; // Default to loading unsigned A
   b_mux_sel     <= 1'b0; // Default to loading unsigned B
   result_mux_sel <= 1'b0; // Default to initializing result
   counter_en     <= 1'b0;
   divreq_rdy     <= 1'b1;
   divresp_val    <= 1'b0;
  end
  else if ((counter_en == 1'b0) && (counter_reg == 5'd31)) begin
    // During the first cycle, load initial values
    if(divreq_val) begin
      a_mux_sel     <= 1'b1;
      b_mux_sel     <= 1'b1;
      result_mux_sel <= 1'b1;
      counter_en    <= 1'b1; // Enable counter decrementing
      divreq_rdy    <= 1'b0;
     end
    else begin
      result_mux_sel <= 1'b0;
      divreq_rdy    <= 1'b1;//about to end
      a_mux_sel     <= 1'b0; // Default to loading unsigned A
      b_mux_sel     <= 1'b0; // Default to loading unsigned B
     end
     divresp_val   <= 1'b0;
  end
  else if ( (counter_reg == 5'b0)) begin
    a_mux_sel     <= 1'b0; // Select the shifted A value
    b_mux_sel     <= 1'b0; // Select the shifted B value
    result_mux_sel <= 1'b0; // Select the adder output for result
    counter_en    <= 1'b0; // Disable counter decrementing
    divreq_rdy    <= 1'b1;
    divresp_val<= 1'b1;
  end
 end
endmodule
```

The operation is done in 33 cycles. There would be one additional clock 31 in front of clock 31:0. The first 31 clock is for hand-shaking (both req and resp) and loading data. So the clock before the first 31 clock (for the first request it would be “reset-status”, and for the later requests, it would be clock 0) would set up req_rdy, resp_val(but not the reset_status), and be ready to load (set a/b_mux_sel and result_mux_sel to 0.)

Reset:
When reset is true, imuldiv is ready for the request (req_rdy<=1). The response is not valid yet(divresp_val<= 1'b0;).Counter is stopped at 31(counter_reg <= 5'd31; counter_en<=1'b0; a/b_reg is ready for loading new data.(a/b_mux_sel<=0).result_reg is ready to be cleared.(result_mux_sel <= 1'b0;)

Not reset, but the timer is stopped:

if requests arrives (divreq_val==1’b1):

The imuldiv is doing hand-shaking and loading, and it is about to start the calculation.

Stop hand-shaking (req_rdy<=0;resp_val<=0;)

Stop loading(set a/b_mux_sel and result_mux_sel to 1.)

Otherwise, back to reset-status(else part)

Timer is at last step(0):

Ready to handshake, load, data, and temporarily stop the timer because it is not calculating (counter_en<=0;)

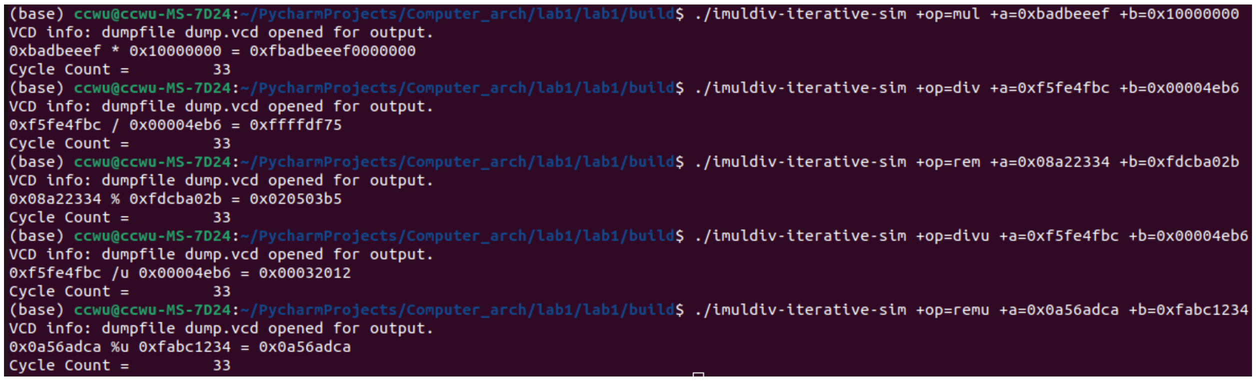
Test Methodology:

Newly appended divu/remu tests:

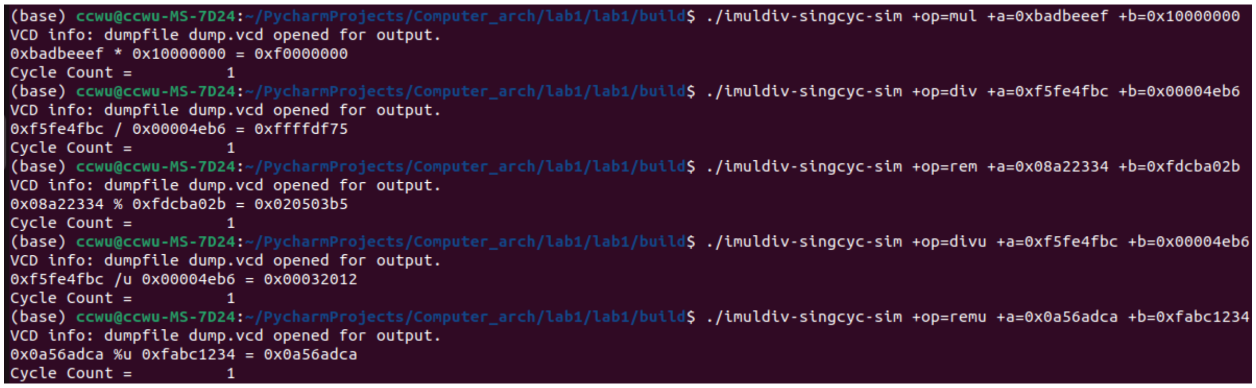
```
t0.src.m[ 12] = 65'h0_00000000_ffffffff; t0.sink.m[ 12] = 64'h00000000_00000000;
t0.src.m[ 13] = 65'h0_ffffffff_ffffffff; t0.sink.m[ 13] = 64'h00000000_00000001;
t0.src.m[ 14] = 65'h0_0a01b044_ffffb14a; t0.sink.m[ 14] = 64'h0a01b044_00000000;
t0.src.m[ 15] = 65'h0_deadbeef_0000beef; t0.sink.m[ 15] = 64'h0000227f_00012a90;
t0.src.m[ 16] = 65'h0_f5fe4fbc_00004eb6; t0.sink.m[ 16] = 64'h000006f0_00032012;
t0.src.m[ 17] = 65'h0_f5fe4fbc_ffffb14a; t0.sink.m[ 17] = 64'hf5fe4fbc_00000000;
```

We use a unit test framework that communicates with imuldiv with val/rdy interface. The framework is consisted of TestSource and TestSink. The TestSource is stored with tests data and would fire requests sequentially upon receiving req_rdy, and TestSink stores and verify the output with correct answer. If the answer is wrong, it displays “[failed]” and correct anser. When TestSink wait too long not finishing the verifying, it displays “[sink failed]”.

Evaluation:



```
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-iterative-sim +op=mul +a=0xbadbeeef +b=0x10000000
VCD info: dumpfile dump.vcd opened for output.
0xbadbeeef * 0x10000000 = 0xfbadbeeef0000000
Cycle Count =          33
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-iterative-sim +op=div +a=0xf5fe4fbc +b=0x00004eb6
VCD info: dumpfile dump.vcd opened for output.
0xf5fe4fbc / 0x00004eb6 = 0xffffdf75
Cycle Count =          33
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-iterative-sim +op=rem +a=0x08a22334 +b=0xfdcba02b
VCD info: dumpfile dump.vcd opened for output.
0x08a22334 % 0xfdcba02b = 0x020503b5
Cycle Count =          33
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-iterative-sim +op=divu +a=0xf5fe4fbc +b=0x00004eb6
VCD info: dumpfile dump.vcd opened for output.
0xf5fe4fbc /u 0x00004eb6 = 0x00032012
Cycle Count =          33
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-iterative-sim +op=remu +a=0x0a56adca +b=0xfabc1234
VCD info: dumpfile dump.vcd opened for output.
0x0a56adca %u 0xfabc1234 = 0x0a56adca
Cycle Count =          33
```



```
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-singcyc-sim +op=mul +a=0xbadbeeef +b=0x10000000
VCD info: dumpfile dump.vcd opened for output.
0xbadbeeef * 0x10000000 = 0xf0000000
Cycle Count =           1
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-singcyc-sim +op=div +a=0xf5fe4fbc +b=0x00004eb6
VCD info: dumpfile dump.vcd opened for output.
0xf5fe4fbc / 0x00004eb6 = 0xffffdf75
Cycle Count =           1
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-singcyc-sim +op=rem +a=0x08a22334 +b=0xfdcba02b
VCD info: dumpfile dump.vcd opened for output.
0x08a22334 % 0xfdcba02b = 0x020503b5
Cycle Count =           1
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-singcyc-sim +op=divu +a=0xf5fe4fbc +b=0x00004eb6
VCD info: dumpfile dump.vcd opened for output.
0xf5fe4fbc /u 0x00004eb6 = 0x00032012
Cycle Count =           1
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/lab1/build$ ./imuldiv-singcyc-sim +op=remu +a=0x0a56adca +b=0xfabc1234
VCD info: dumpfile dump.vcd opened for output.
0x0a56adca %u 0xfabc1234 = 0x0a56adca
Cycle Count =           1
```

Conclusion:

This newly-built imuldiv is stable and not depending on other situations (reset is set up until half clock before the requests arrive, requests is continuous) but only the val/rdy interfaces, and is capable of operating 3 operations (mul,div/rem,divu/remu).