# Computer Architecture

# Lab 1: Integer Multiply/Divide Unit

Author:313551151 吳承鋮

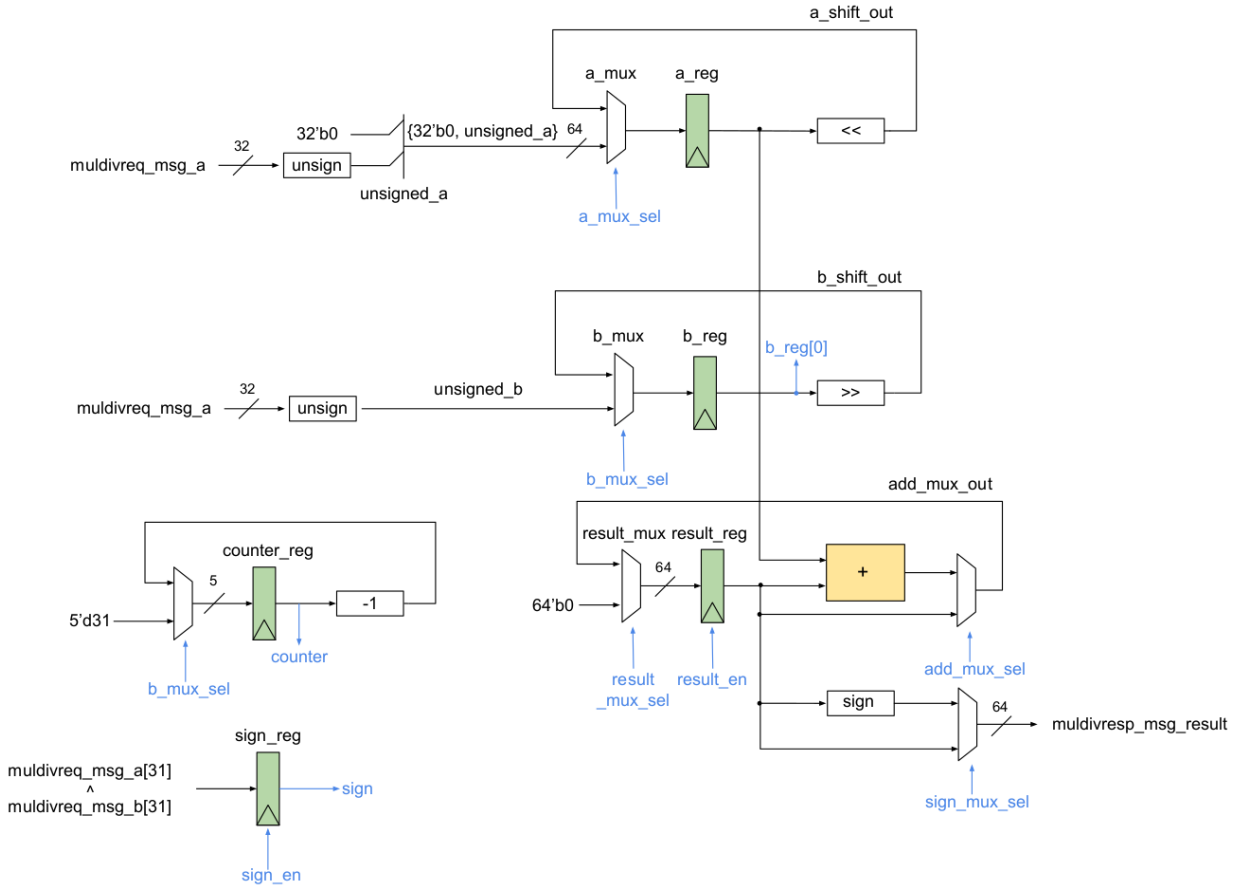
Introduction:

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:

1. imuldiv-IntMulIterative.v

The design diagram is largely based on the appendix diagram:



With some supplement and modifications:

A.  Renaming (muldivreq_msg_a-> mulreq_msg_a,

muldivresp_msg_result-> mulresp_msg_result)

B. Control unit

a. counter:

counter_en(register) is added to control the counter.

b.Replacing sign_en

In the revised design, the sign_en signal, which originally managed the sign of the final result, is now implicitly handled through the control of the counter. Specifically:

The sign calculation is enabled when the counter reaches 5'd31, meaning the sign is only calculated at the start of the iterative process. The XOR of the sign bits (sign_bit_a ^ sign_bit_b) is stored in the sign_reg and applied to the final result.

The actual sign is applied to the result in the datapath when the multiplication is complete by using the sign_reg to adjust the final result based on whether the result should be negative.

c.Replacing val_reg

Similarly, the valid signal val_reg is no longer explicitly needed because the process is now tied to the state of the counter:

The result is considered valid (mulresp_val) when the counter reaches 5'b0, which indicates the multiplication process has finished.

**d.Mimicking rdy/val interface of single cycle version**

To make it easier to understand, I will name the test period of testbench into 3 periods:

Preparing stage: Half clock before reset vanish, message arrives.

(a cycle that is not in any stages)

Testing stage: First message arrives, Evaluating  the result

Ending stage: after evaluating  the result

In single cycle version, req_rdy =1 in Preparing stage and resp_val = 0 in Ending stage.

To test whether stages it is in, I use 2 regs: program_start (Testing stage & Ending stage)and starting(Preparing stage).

And a condition that only satisfied in Preparing stage and Ending stage:
(~mulreq_val)&&(counter_en == 1'b0)

So the setup would be:

```
assign mulreq_rdy = ((counter_reg==5'b0)||((program_start||starting)&&(~mulreq_val)&&(counter_en == 1'b0)));
assign mulresp_val = ((counter_reg==5'b0) && (~((program_start)&&(~mulreq_val)&&(counter_en == 1'b0))));
```

2. imuldiv-IntDivIterative.v

Which is very similar as above.

But I reduce all the 32 bit to 31 bit by ommiting the sign bit of unsign variable to make it acheive in 33 cycles.

**Test Methodology:**

A unit test framework simplifies creating and managing unit tests by providing a standardized approach that improves understanding, execution, and logging. This course uses unit testing to evaluate each module in isolation, with every source file requiring a matching test file. The provided test framework includes a TestSource and TestSink to store input messages and expected outputs, respectively, and automatically manage test progression. Tests for mul, div, and rem instructions are included, but you must add cases for divu and remu. It's also advisable to expand tests for the full 64-bit range for the mul operation.

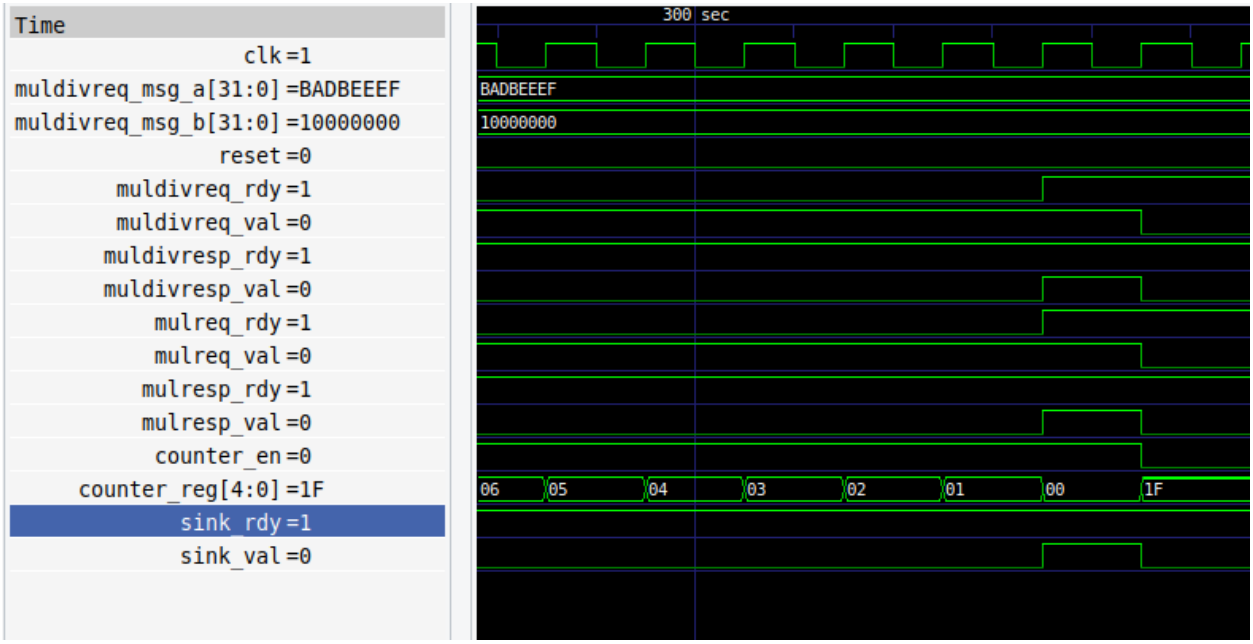
**Evaluation:**

make check passed

```
(base) ccwu@ccwu-MS-7D24:~/PycharmProjects/Computer_arch/lab1/build$ make check
iverilog -g2005 -Wall -Wno-sensitivity-entire-vector -Wno-sensitivity-entire-array -o imuldiv-DivReqMsg-utst -I ../vc -I ../imuldiv -I ../vc ../imuldiv/imuldiv-DivReqMsg.t.v
./imuldiv-DivReqMsg-utst +verbose=2 > imuldiv-DivReqMsg-utst.out
iverilog -g2005 -Wall -Wno-sensitivity-entire-vector -Wno-sensitivity-entire-array -o imuldiv-MulDivReqMsg-utst -I ../vc -I ../imuldiv -I ../vc ../imuldiv/imuldiv-MulDivReqMsg.t.v
./imuldiv-MulDivReqMsg-utst +verbose=2 > imuldiv-MulDivReqMsg-utst.out
iverilog -g2005 -Wall -Wno-sensitivity-entire-vector -Wno-sensitivity-entire-array -o imuldiv-IntMulDivSingleCycle-utst -I ../vc -I ../imuldiv -I ../vc ../imuldiv/imuldiv-IntMulDivSingleCycle.t.v
./imuldiv-IntMulDivSingleCycle-utst +verbose=2 > imuldiv-IntMulDivSingleCycle-utst.out
iverilog -g2005 -Wall -Wno-sensitivity-entire-vector -Wno-sensitivity-entire-array -o imuldiv-IntMulIterative-utst -I ../vc -I ../imuldiv -I ../vc ../imuldiv/imuldiv-IntMulIterative.t.v
./imuldiv-IntMulIterative-utst +verbose=2 > imuldiv-IntMulIterative-utst.out
iverilog -g2005 -Wall -Wno-sensitivity-entire-vector -Wno-sensitivity-entire-array -o imuldiv-IntDivIterative-utst -I ../vc -I ../imuldiv -I ../vc ../imuldiv/imuldiv-IntDivIterative.t.v
./imuldiv-IntDivIterative-utst +verbose=2 > imuldiv-IntDivIterative-utst.out
iverilog -g2005 -Wall -Wno-sensitivity-entire-vector -Wno-sensitivity-entire-array -o imuldiv-IntMulDivIterative-utst -I ../vc -I ../imuldiv -I ../vc ../imuldiv/imuldiv-IntMulDivIterative.t.v
./imuldiv-IntMulDivIterative-utst +verbose=2 > imuldiv-IntMulDivIterative-utst.out

 Entering Test Suite: vc-Misc
 Entering Test Suite: vc-Muxes
 Entering Test Suite: vc-Arith
 Entering Test Suite: vc-StateElements
 Entering Test Suite: vc-Regfiles
 Entering Test Suite: vc-RAMs
 Entering Test Suite: vc-Queues
 Entering Test Suite: vc-Arbiters
 Entering Test Suite: vc-Memories
 Entering Test Suite: imuldiv-DivReqMsg
 Entering Test Suite: imuldiv-MulDivReqMsg
 Entering Test Suite: imuldiv-IntMulDivSingleCycle
 Entering Test Suite: imuldiv-IntMulIterative
 Entering Test Suite: imuldiv-IntDivIterative
 Entering Test Suite: imuldiv-IntMulDivIterative
```

But the imuldiv-iterative-sim fails for longing infinitive.




Even though I have set up the resp_val flag.

**Conclusion:**

In conclusion, several modifications were made to the iterative multiplier unit to streamline the control logic and align it with the single-cycle version's interface. Renaming inputs and outputs like muldivreq_msg_a to mulreq_msg_a improved clarity. The control unit was enhanced by adding counter_en to manage the counter and replacing the sign_en and val_reg signals. The sign calculation now triggers when the counter reaches 5'd31, while mulresp_val is asserted when the counter hits 5'b0, indicating a completed process. Additionally, the interface was refined to mimic the ready/valid interface by managing

stages using two registers (program_start and starting), ensuring smoother transitions between test phases.