Lab 1　Report

313553054 陳冠霖

1. Introduction

This lab focuses on designing and implementing an iterative multiplier and divider in Verilog. The multiplier supports 32-bit unsigned and signed multiplication, and the divider supports both unsigned and signed division. This report will describe the design choices, testing methodology, and evaluation results for both modules.

2. Design

2-1 Implementation

In this section, I will present the design approach of the multiplier, including the composition of the datapath and control logic, as well as the communication between the two through various signals.

a. Control Logic

The control logic of the multiplier unit is based on a finite state machine (FSM) with three states: **IDLE**, **RUN**, and **DONE**. The FSM is responsible for managing the flow of the multiplication process and ensuring proper sequencing of operations.

1. **IDLE State**:
   The system starts in the IDLE state, waiting for valid input. When both mulreq_val and mulreq_rdy are high, indicating that the multiplication request is ready and valid, the FSM transitions to the RUN state. This ensures that the inputs are correctly loaded into the datapath registers.
2. **RUN State**:
   In this state, a 6-bit counter is initialized to 32, representing the number of cycles required for a 32-bit multiplication. Each clock cycle, the counter decrements by 1, while the datapath iteratively performs a partial addition or shifting operation to compute the product. When the counter reaches 0, the FSM transitions to the DONE state.

3. **DONE State**:
   In the DONE state, the multiplication result is ready to be returned. The FSM waits for both mulresp_val and mulresp_rdy to be high, signaling that the result has been accepted by the downstream logic. Once acknowledged, the FSM transitions back to the IDLE state, ready for the next multiplication request.

This FSM design organizes the multiplication process with clear state transitions and ensures that the multiplication is completed in 32 cycles using the iterative approach.

b. Datapath Design

The datapath is designed to efficiently perform the multiplication operation using an iterative approach. Four registers (a_reg, b_reg, unsigned_result, and sign_bit_reg) and the combination logic manage the data flow between the registers and the control unit.

1. **a_reg & b_reg**:
   These registers store the two operands (mulreq_msg_a and mulreq_msg_b) for multiplication. In the IDLE state, a_reg is loaded with the value of operand A extended to 64 bits ({32'b0, unsigned_a}), while b_reg is loaded with the 32-bit value of operand B. In the RUN state, a_reg shifts left by one bit on each clock cycle, and b_regshifts right by one bit, aligning the operands for partial additions. The shift operations are controlled by the FSM.
2. **unsigned_result**:
   This register accumulates the partial sums during the multiplication process. If the least significant bit (LSB) of b_reg is 1, the current value of a_reg is added to the accumulator. Otherwise, the accumulator remains unchanged. This conditional addition is performed iteratively over 32 cycles, following the standard binary multiplication algorithm.
3. **sign_bit_reg**:
   This register tracks the sign of the final result. The result's sign is determined by the XOR of the sign bits of the two operands (sign_bit_a ^ sign_bit_b). If the XOR evaluates to 1, the result is signed, and the accumulated product is

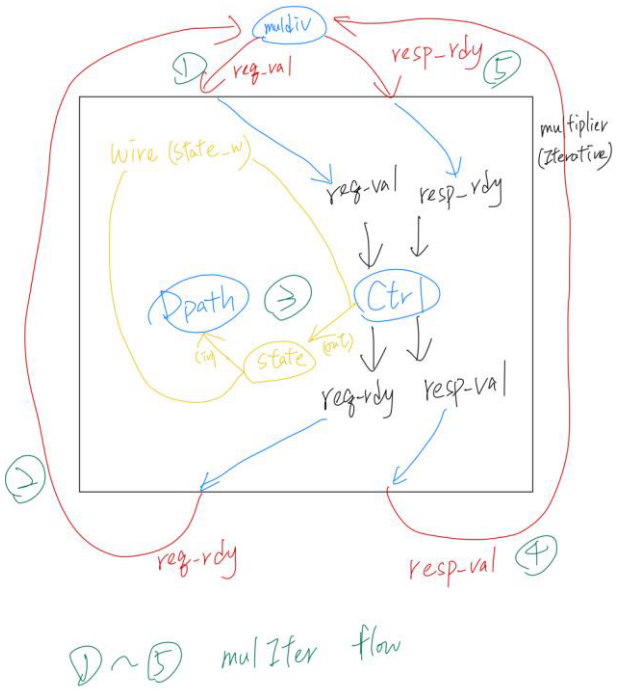
negated at the end of the computation to reflect the correct sign.

The principles of the divider and multiplier are fundamentally very similar, with the primary difference being the computational algorithms. Therefore, they will not be further elaborated here.

2-2 Design Decisions and Justifications

- **Deviations from the prescribed datapath**

I have separated the control and datapath into two independent components. The former is used to control the handshake between val and rdy, while the latter handles the logical operations for the data. Communication between the two is managed through the previously mentioned finite state machine (FSM).



3. Testing Methodology

To ensure the robustness and correctness of the Mul/Div Unit, a comprehensive

testing methodology was employed. The test cases cover a wide range of input values. Each test case includes 64-bit operations, ensuring the full range of bit-width is utilized. This allows for accurate validation of both multiplication and division functionalities. Additionally, the test bench is automated, and the results are verified using a test sink to check the expected outcomes against the actual results, confirming that the module performs correctly across various inputs.

Additionally, I also added several test cases in the three .t.v files.

4. Evaluation

Simulation results show that both the multiplier and divider perform as expected:

1. 0xbadbeeef * 0x10000000 = 0xfbadbeeef0000000; Cycle Count = 33;
2. 0xf5fe4fbc / 0x00004eb6 = 0xffffdf75; Cycle Count = 33;
3. 0x08a22334 % 0xfdcba02b = 0x020503b5; Cycle Count = 33;
4. 0xf5fe4fbc /u 0x00004eb6 = 0x00032012; Cycle Count = 33;
5. 0x0a56adca %u 0xfabc1234 = 0x0a56adca; Cycle Count = 33;

5. Conclusion

In this lab, I successfully implemented and tested an iterative multiplier and divider in Verilog. The multiplier handles both signed and unsigned 32-bit multiplication, and the divider supports both signed and unsigned division. Testing confirmed the correctness of the design, and simulation results matched the expected cycle counts. This lab provided practical insights into the design and verification of arithmetic units using hardware description languages.