|  |  |
| --- | --- |
|  | **Karlsruhe Institute of Technology**  Institut für Technik der Informationsverarbeitung (ITIV)  Head of Institute  Prof. Dr.-Ing. Dr. h. c. Jürgen Becker  Prof. Dr.-Ing. Eric Sax  Prof. Dr. rer. nat. Wilhelm Stork |

**Digital Hardware Design Laboratory**

**Exercise: 4**

**Group: 11**

**Names: Weikang Wu**

Task 1 Write a Divider with HW Implementation

**introduction**

The major point of this exercise stands at learning Arithmetic operations and pipelining implementaion in VHDL and using a practical divider implementation to have a deep cognition.

**Process**

1. Understand how to divide a dividend with a divisor and how to end with quotient and remainder arithmetically.

2. Understand how to implement this operation in HW: procedure will be pipelined and the divisor will be updated using a shift operation.

3. What happens when it comes to unsigned value operation? signed: +/- unsigned:+

4. Learn about the vector size reservation of operations:

output vector size(C) of a add/sub operation(A,B) is C=A+B

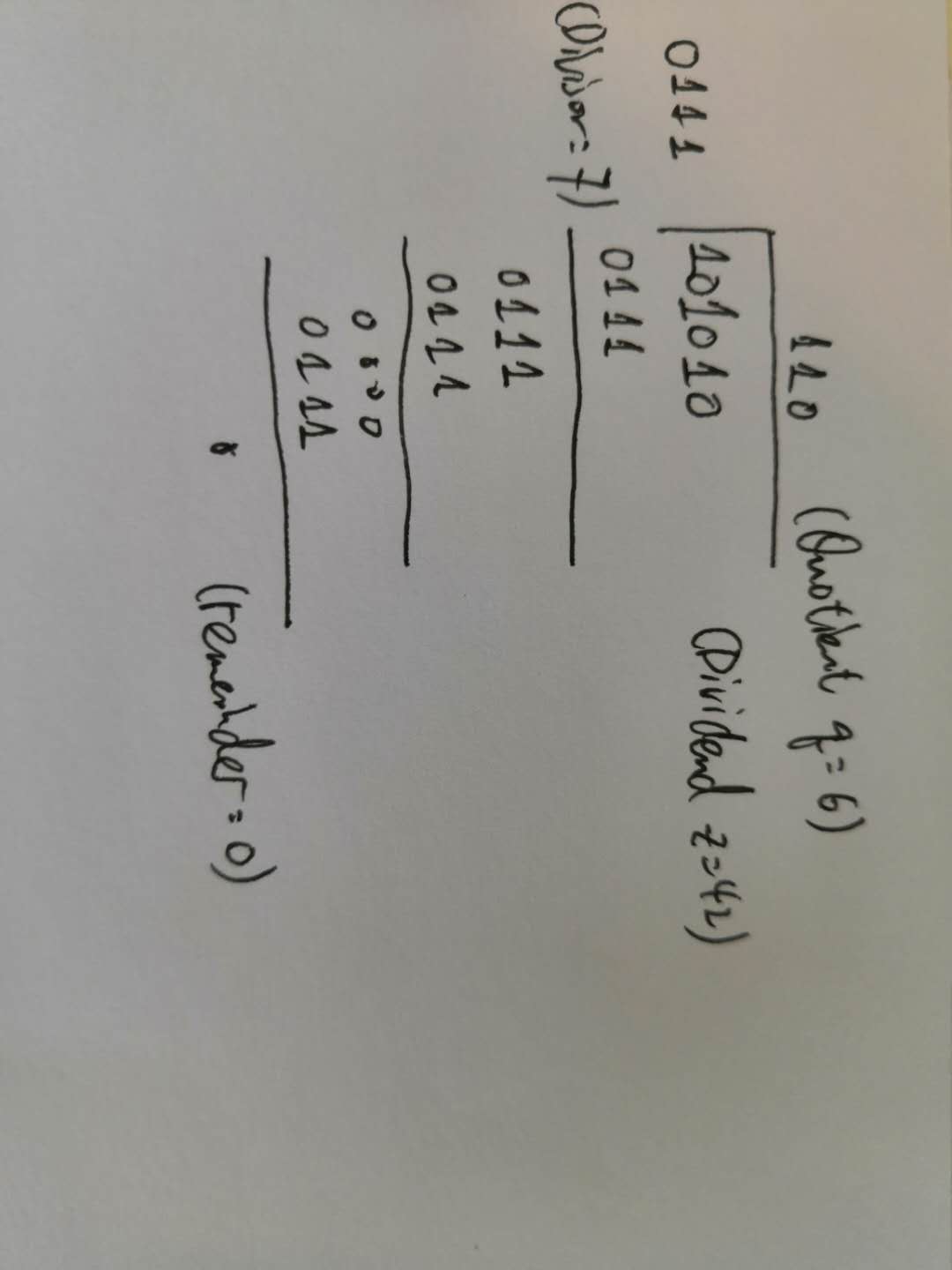
output vector size(C) of a multiplication operation(A,B) is C=A\*B

5. Pipeline implementation:

using for-loop or for-generate

**Q1 Calculate the binary integer division with the values 42/7 using pen**

**and paper.**

****

**Q2 How would the steps for calculating a/b change if a negative**

**value was used, e.g. a = −42, b = 7?**

The sign of number will first be neglected and at the final stage will be added to the quotient and remainder. Since the rule is remainder share the same sign with dividend.

In this case we first do a normal 42/7 and get quotient 6 remainder 0, we add both negative to them which return -6 and -0(0)

**Q3 What does “pipelining” mean in the field of hardware development?**

The pipelining is a technique that was used to maximize the calculation efficiency while reducing the error possibility.

Since pipelining has its unique structure with the same method at each cycle and output one return at each cycle, latency and high efficiency are guaranteed.

**Q4 Implement the signed divider using the shift and subtract algorithm of Input1 divided by Input2. The divider should have a pipelined**

**architecture and accept a new set of input values in each clock cycle.**

See appended Codes

**Q5 How many zeros do you need to append to the divisor**

**in order to apply the shift and subtract algorithm for binary polynomial**

**division step by step in Hardware? Using binary shifts of the divisor to**

**iteratively determine the result of the division with pen and paper is the**

**most efficient solution.**

We need to add so many zeros to the divisor such that divisor has the same dimension with the dividend, which results in the same dimension of quotient and remainder. So the answer is the difference between dividend size and divisor size.

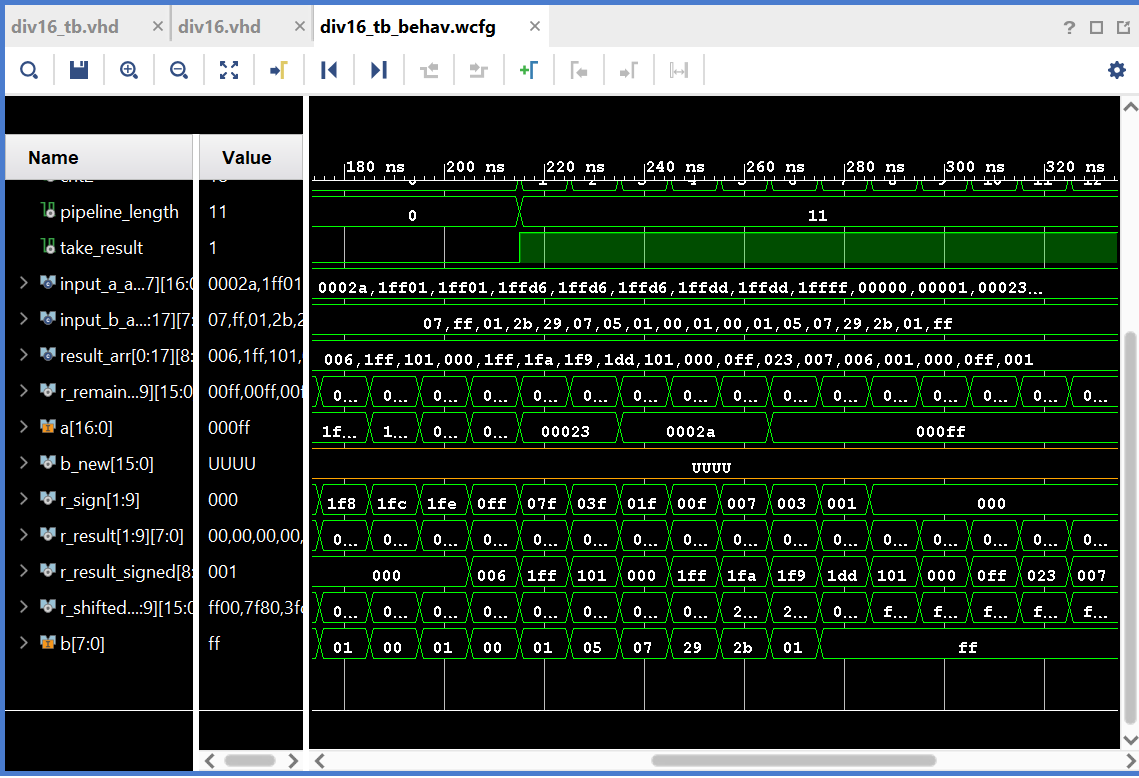
**Q6 How have you implemented the pipeline structure?**

See appended Codes

**Q7 Simulate the divider with the provided div16\_tb Testbench**

**and verify that the results are correct.**

According to the result picture, all answers are matched, no warnings appear.



Appended Codes:

if en = '1' then

-- if neg, abs(A) neglect MSB => r\_remainder

if a(A\_WIDTH-1) = '1' then

a\_signed := abs(signed(a));--convert the 2nd Complement to origin

r\_remainder(1) <= unsigned(a\_signed(15 downto 0));

r\_sign(1) <= '1';

else -- if pos, neglect MSB => r\_remainder

r\_remainder(1) <= unsigned(a(15 downto 0));

r\_sign(1) <= '0';

end if;

-- assign r\_shifted\_b with value of b and appended 0s

r\_shifted\_b(1) <= unsigned(b & "00000000");

--pipeline construction

for i in 2 to RESULT\_WIDTH loop

--move sign pipeline forward 1 step

r\_sign(i) <= r\_sign(i-1);

--if current remainder bigger than divisor

--new remainder is the substraction

--result append 1 and forward 1 step

if r\_remainder(i-1) >= r\_shifted\_b(i-1) then

r\_remainder(i) <= r\_remainder(i-1) - r\_shifted\_b(i-1);

r\_result(i) <= shift\_left(r\_result(i-1),1) + 1;

--if current remainder smaller than divisor

--new remainder stay the same

--result append 0 and forward 1 step

else

r\_remainder(i) <= r\_remainder(i-1);

r\_result(i) <= shift\_left(r\_result(i-1),1);

end if;

-- shift b to the right, update the divisor

r\_shifted\_b(i) <= '0' & r\_shifted\_b(i-1)(A\_WIDTH-2 downto 1);

end loop;

--final result manipulation

--we have dividend of size 16(neglecting MSB) and divisor of size (8+8),

--we shift r\_shifted\_b(b+00000000) for 8 times inside the pipeline so the final r\_shifted\_b is b its self, we need to make one more comparison of b and remainder.

v\_result := shift\_left(r\_result(9),1);

if r\_remainder(9) >= r\_shifted\_b(9) then

v\_result := v\_result + 1;

end if;

--output result according to the sign and result

if r\_sign(9) = '1' and v\_result /= 0 then

--if dividend is neg and not 0, we need to construct the 2nd complement

r\_result\_signed <= signed('1' & (not v\_result + 1));

else

--if dividend is pos or 0, just output result

r\_result\_signed <= signed('0' & v\_result);

end if;

--if not enabled output 0

else

r\_result\_signed <= "000000000";

end if;