Praktikum Entwurf digitaler Systeme
Digital Hardware Design Lab

# Exercise sheet 4

Arithmetic Pipelines

## Contents

## 1 Preparation

Prior to the laboratory afternoon for this exercise you should get familiar with the following topics:

- The complement on two representation for binary signed integer numbers
- The “shift-and-subtract” method for dividing integer numbers using their binary representation (see Section 2 and Figure 1)
- The concept of “Pipelining”. See Section 1.1 and DHL slides on “Advanced VHDL and Pipelining”.

**Task 1:** Calculate the binary integer division with the values $^{42}/_{7}$ using pen and paper.

**Question 2:** How would the steps for calculating $^{a}/_{b}$ change if a negative value was used, e.g. $a = -42$, $b = 7$?

**Question 3:** What does “pipelining” mean in the field of hardware development?

## 1.1 The Concept of Pipelining

The concept of “Pipelining” is used to divide complex computational operations into a sequence of smaller steps, which can be executed one after another. While the steps for one operation are executed sequentially, the hardware still processes multiple operations for independent data sets in parallel. One example is the well known *instruction pipeline* in microprocessor architectures. Such a pipeline could include following stages for each processor instruction:

1. Fetch
2. Decode
3. Execute
4. Memory Access
5. Write-back

When the first instruction completes the *Fetch* step and proceeds to the *Decode* step, the hardware already starts fetching the next instruction in parallel. Thus, after an initial warm-up phase, the pipeline always processes 5 instructions in parallel and can complete one instruction per clock cycle (assuming one cycle per pipeline stage). The processing latency for each individual instruction would be 5 cycles. This concept is similar to the principle of an assembly line in factories, where each product is traversing through a sequence of workstations, which are each specialized to only one specific manufacturing step.

In the Pipelining concept, it is important that every stage needs the same number of clock cycles. If one stage needs less clock cycles than the others, a chain of registers must be inserted to add the corresponding delays. An advantage of pipelines is that the clock frequency can be increased, since only the smaller pipeline steps need to be completed in one clock cycle (and not the overall computational operation). Pipelining leads to less logic operations per clock cycle and thus to lower logic gate runtimes, while there is still a throughput of one result per cycle.

In VHDL, an implementation of a pipeline typically uses the statements ”`for`” or ”`generate`”. Signals with ”`array`” types can be used to create register chains to store the intermediate results.
For further information see also the provided slides on ”Advanced VHDL and Pipelining”.

# 2 Task Description

During the course of this lab, you will develop a pipelined integer divider just knowing the requirements and test it with a simulation.

**Path:**

- Source files: "*VHDL_Ex3.data/sources*"
- ModelSim scripts: "*simulation*"

## 2.1 Implementation of a Signed Divider

In the first part of the exercise a signed divider needs to be implemented which has the following characteristics (see also the template *"div16.vhd"*):

- Use generics to parametrize the widths of the input- and output values
- Use the following widths as default values:
  - Input1: 17 Bit std_logic_vector (to be interpreted a **signed**)
  - Input2: 8 Bit std_logic_vector (to be interpreted as **unsigned**)
  - Result: 9 Bit std_logic_vector (to be interpreted a **signed**)
- The output has to be set to 0 until the result of the first division is handed out.

**Task 4:** 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.

In the **shift and subtract division algorithm**, you left-align the divisor below the dividend, and compare the divisor to the dividend digits directly above it to determine the most significant quotient digit. Once the divisor is greater than the dividend digits above it, the divisor is shifted to the right one digit. Comparing the divisor to the new dividend digits above produces the next quotient digit. This process repeats until the divisor is larger than the remaining dividend. The quotient is complete, and the remaining dividend equals the remainder.

In order to determine the steps of your pipeline for implementing this algorithm, break down the procedure into small basic steps until it becomes clear how to implement them with digital logic. Some of the steps may repeat themselves, which makes the pipeline design more efficient as such steps may be defined as a loop in VHDL.

In this task, it is of special importance to recall the **two's complement representation** and decide on a strategy how to perform the division step by step and which information to remember during the computation in order to ease the calculation in the end.

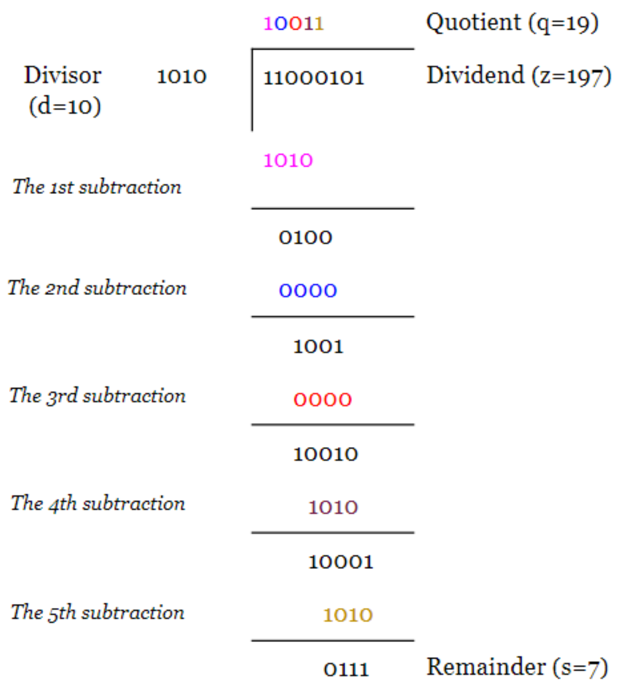

Figure 1: Shift-and-subtract binary division

**Question 5:** 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.

**Question 6:** How have you implemented the pipeline structure?

## 2.2 Divider Verification

**Task 7:** Simulate the divider with the provided **div16_tb Testbench** and verify that the results are correct.

The first data set, the testbench passes to the divider, consists of the values a=42 and b=7 which allows you to compare the divider results against the division you did by hand (see Section 1).

**Note:** During design of the pipeline, some stages in the shift register may remain unused. You are free to remove those stages or simply leave them idle.