# COMPUTERGESTÜTZTES EXPERIMENTIEREN I
(Computer-Assisted Experimentation I)

P R A C T I C A L

# PART I

## Introduction to Computer Engineering

### Digital Electronics

Combinational Circuits

Sequential Circuits

Computer System Bus

Functional Model of a Computer

# 6 PRACTICAL EXERCISES

## Practical I

## Combinational Circuits

In the course of this practical you will implement small combinational circuits with NAND gates. The solutions of the exercises build on the solutions of the previous one (except exercise 5).

Proceed in the following steps for each exercise:

- Set up the function table
- Derive the algebraic expression using a canonical normal form; simplify
- Draw the logical circuit using AND, OR, and NOT gates
- Transform the circuit using solely NAND gates and build it

Hints:

Addition of two dual digits:
$0+0=0, \qquad 0+1=1, \qquad 1+0=1, \qquad 1+1=0$ carry 1
Similarly, subtraction of two dual digits:
$0-0=0, \qquad 1-0=1, \qquad 1-1=0, \qquad 0-1=1$ borrow 1

The two's complement of a dual number can be obtained by complementing each digit (one's complement) and adding 1, e.g.:

$$01010 \Longrightarrow 10101 + 1 = 10110.$$

Subtraction of a dual number B from number A can be done by adding the two's complement of B, e.g.:

$$\begin{array}{rcr} 0101 & \Longrightarrow & 0101 \\ \text{-}0011 & \Longrightarrow & +1101 \end{array}$$

## Exercises

1. Implement a circuit with two inputs whose output is "1" (true) only if one input is "1" but not if both are "1". This function is called exclusive OR (XOR) or "parity function" when generalized to three or more inputs.

2. Implement a circuit that adds the two binary digits a and b and produces the two outputs sum S and carry U. This circuit is called "Half Adder"

3. Consider now adding two dual numbes a and b. In a general situation at position i the digits $a_i$ and $b_i$ and furthermore the carry of the previous position $U_{i-1}$ have to be added resulting in the sum $S_i$ and the carry $U_i$. Implement a circuit that produces the sum $S_i$ and the carry $U_i$. This circuit is referred to as "Full Adder". For the implementation consider intermediate steps in which two binary digits of either the digits or the intermediate results are added using the half adder developed in the previous exercise.

4. Optional: Implement a circuit that calculates the difference of two dual numbers. In analogy to the full adder proceed in steps: First implement the difference of two digits (half subtractor) and then the full circuit of the full subtractor.

5. Implement the subtraction of two dual numbers represented by four digits by adding the two's complement. For implementation use the SIMULOG block containing four full adders. The one's complement shall be implemented using the block containg four NANDs.

6. Extend the circuit of exercise 5 in such a way that two four digit numbers alternatively can be added or subtracted. Implement using blocks of four fold full adders, and of four fold Exclusive OR. Is the carry $U_3$ of the most significat position sufficient to indicate an overflow situation? Implement the required "sign extension" and the signal to indicate an overflow.

7. Implement the circuit of a multiplexer. A mulltiplexer operates like a switch point whereby the two input signals A and B are routed to the output according to the switching signal $S = E_A$ and as defined in the compressed truth table:

| A | B | $S = E_A$ | Output |
|---|---|---|---|
| A | B | 0 | B |
| A | B | 1 | A |

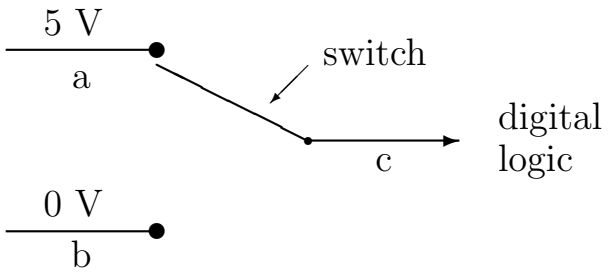
# Practical II

## Sequential Circuits

In this practical you will learn the use of the sequential circuits, called FLIP-FLOP, which can store one binary digit (bit). In the first exercise you build some basic FLIP-FLOPs with NAND gates and use them for a practcal purpose. In the subsequent exercises complete FLIP-FLOPS are used to implement larger units of storage and counters.

Hints:

A so-called SPDT switch, is a single pole double throw switch, that connects a common terminal to either of two terminals.




The SPDT switch connects either of the terminals a or b to the common terminal c. When the switch is flipped from one position to another, the springy metal switch might bounce off from the contact thus giving rise to a chattered signal on terminal c. This is referred to as chatter or contact bounce. A digital logic connected to c will see the chatter and detect it as multiple on-offs despite the switch was flipped only once.

## Exercises

1. Implement an SR FLIP-FLOP with two NAND gates.

   (a) Set up the function table that describes the behaviour of the FLIP-FLOP. How do you take into account that it is a sequential circuit?

   (b) How could you "debounce" an SPDT switch using an SR FLIP-FLOP?

   (c) Convert the FLIP-FLOP into a "gated" SR FLIP-FLOP

2. Implement a 4 bit shift register using the JK FLIP-FLOPs.

3. Implement a 4 bit binary counter (ripple counter). Add the logic to it such that upon having counted seven impulses the counter is blocked until reset.

4. Implement a 4 bit binary counter. Modify it such that it counts down.

   Extend the circuit, such that the counter alternatively counts up or down whereby a switch shall determine the direction of counting. What do you observe when you change the direction?

   Do you observe the same issue when you implement the up down counter as a synchronous counter?

# Practical III

## Computer System Bus

In this practical you will recapitulate and deepen your knowlege of a computer's system bus in order to understand its principal functionality.

A bus is a collectively used system of lines (bus is a contraction of the Latin omnibus meaning together) which is used to connect the multiple outputs and inputs of compound logic components such as registers. On such a commonly used path the outputs of many sources will be routed to the inputs of many destinations. However, in the digital electronics we may not connect the outputs of gates directly as it would produce a short circuit. Hence, combinational circuits such as multiplexers would be required to join the outputs, which, however, would be expensive.

Therefore,the outputs of the components are connected to busses using tristate buffers. These buffers propagate the logical state of the input unchanged to the output unless an inhibit signal is set. In this event the output goes into a high impedence state, i.e. the input is electrically disconnected and has no impact on the output.

The individual lines of a bus can be put into groups of same function:

| | |
|---|---|
| DATA BUS | The lines carry the information from one component connected to the bus to another. |
| ADDRESS BUS | The lines determine where the information should be sent to. |
| CONTROL BUS | The lines control the operations and the timing of the transfer of data and addresses. |

## Exercises

In this exercise we set up a simplified version of the model computer discussed in the lecture. The system bus consists mainly of the four lines of the data bus. Address bus and control lines are only implict. Implement the following components:

- A block of 4 bit inputs with toggle switches connected to the bus via a tristate buffer.
- A 4 bit shift register A whose inputs are connected directly to the bus and whose outputs are connected via tristate buffer to the bus.
- A 4 bit shift register B whose inputs are connected directly to the bus
- A 4 bit output register, that stores and shows the state of the buses lines.
- And a serial adder that adds the numbers stored in registers A and B and stores the result in register A. The intermediate carry is stored in a flip-flop.

First, make yourself familiar with the individual components and then implement the whole system.

# Practical IV

## Functional Model of a Computer

In the previous practical you made yourself familiar with the main functional units of a stored-program computer. In this practical you will design and implement a simplfied model of a computer in order to understand its functionality.

The generic architecture of a digital computer was originally described by John von Neumann. According to this architecture any computer consists of the following units:

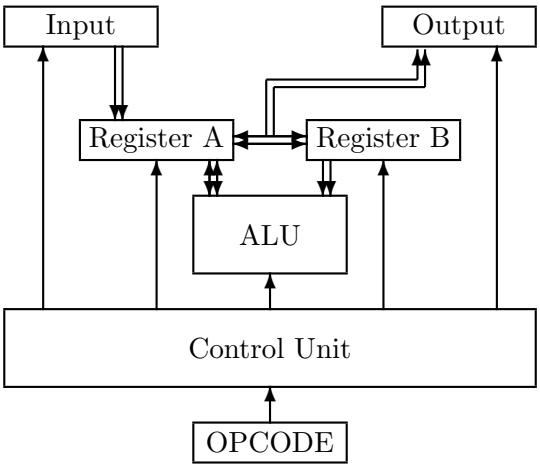
| | |
|---|---|
| Central Processing Unit | consisting of |
| Processing Unit (ALU) | performing arithmetic and logic operations on the registers |
| Registers | storing the operands |
| Control Unit | controls the execution of the instructions using instruction register and program counter |
| Memory Unit | stores data **and** instructions |
| Input Unit | devices for input of data (and instructions) also from secondary mass storage |
| Output Unit | devices for output of data (and instructions) also to secondary mass storage |

In our functional model we will not use memory; data will come from the input, results of the processing will be put to the output, and the instructions (OPCODE) will be entered via a switch register, representing the instruction register. The data paths will be 4 bits wide. All instructions will be executed in a single clock cycle.

Apart from designing the architecture of the computer, the major task will be the design of the combinational circuit that implements the control unit.

## Exercises

The functionality of our computer is defined by the diagram below and the operations listed in the table. There is not a unique design that would implement the requested funtions. However, we will be limited by the available SIMULOG hardware. After you have done your own design, you will be guided to a feasible set up. The computer shall be a 4 bit computer, i.e. the data paths represented by double arrows in the diagram will be four lines wide and we will use SIMULOG blocks containing four of the respective components.




The control unit shall generate the control signals represented in the diagram by single arrows according to the operations given in the table. The operations are defined by a 3 bit number, the operation code:

| OPCODE | FUNCTION |
|---|---|
| 000 | $\mathrm{A} + \mathrm{B} \Longrightarrow \mathrm{A}$ |
| 001 | $\mathrm{A} - \mathrm{B} \Longrightarrow \mathrm{A}$ |
| 010 | $\mathrm{A} + 1 \Longrightarrow \mathrm{A}$ |
| 011 | $\mathrm{A} - 1 \Longrightarrow \mathrm{A}$ |
| 100 | $\mathrm{Input} \Longrightarrow \mathrm{A}$ |
| 101 | $\mathrm{A} \Longrightarrow \mathrm{Output}$ |
| 110 | $\mathrm{A} \Longleftrightarrow \mathrm{B}$ |
| 111 | $\mathrm{B}_i \Longrightarrow B_{i-1}$ |

## Questions and Suggestions for Programs

1. Which classes of operations are not implemented in our model computer?
2. As we use two's complement, in which range can numbers be stored in the registers?
3. Write a program that calculates the absolut value of $3 - 5$.
4. Write a program that runs through the whole range of numbers, i.e. $-8$ through $+7$.
5. Calculate $5 + 3$. Calculate $-2 - 7$. How could we detect an overflow? See practical I.