# Logic Synthesis

**Hardware for Artificial Intelligence Fundamentals - Lab**

**Hardware for Artificial Intelligence Group**

TECHNISCHE UNIVERSITÄT DARMSTADT

Winter 2024
Sheet 5

In this exercise we are going to create an accelerator for a single neuron. We will use industry standard electronic design automation tools, which are installed on the lab PCs. For guidance on using the lab PCs please read the provided document “How to use the lab PCs”.

## Task 5.1: Describing Adders and Multipliers in Hardware

As discussed during the lecture, there are benefits to embedding weights directly into to the hardware. We will used signed integers for this exercise, you do not need to worry about shifting the zero point.

### 5.1a) Adder

Create a verilog module for an adder. The bit-with for inputs (a, b) and output (y) should be 16 bit[1]. Use the template below.

Listing 1: add.v

```verilog
module ADD (
    input wire signed [15:0] a,
    input wire signed [15:0] b,
    output wire signed [15:0] y
);
    // your code here
endmodule
```

### 5.1b) Multiplier

Create a verilog module for a multiplier. The bit-with for the inputs (a, b) should be 8 bit and therefore the result (y) is 16 bit wide.

Listing 2: mul.v

```verilog
module MUL (
    input wire signed [7:0] a,
    input wire signed [7:0] b,
    output wire signed [15:0] y
);
    // your code here
endmodule
```

[1]Theoretically, the addition of to 16 bit numbers could result in a 17 bit number. We simply ignore this for now in our design.

## 5.1c) Synthesis

To create an ASIC from our modules we need a technology library, which describes the available gates. The library is available on moodle, the following script describes to Design Compiler what it is suppsed to do: compile our module.

Listing 3: synthesis.tcl

```
set target_library NangateOpenCellLibrary_typical.db ; # tell synopsis which library to use
set link_library NangateOpenCellLibrary_typical.db

read_file add.v ; # read specified input file
link ; # link design with library
compile_ultra ; # compile design with extra effort
write -f verilog -hierarchy -output netlist_add.v ; # save result as verilog netlist
quit
```

Use `module load syn/2017.09-SP3` to make Synopsys Design Compiler available in your shell. You can then execute the compilation script with `dc_shell -f synthesis.tcl`.

Have a look at the netlist. There is no instance of the adder anymore, instead everything is replaced by the modules from the library we used. Go ahead and compile the netlist for the multiplier as well.

## 5.1d) Simulation

Testing a hardware design is a difficult process. It is not feasible to test all input combinations and transitions, instead we provide you with a testvector. Download the testvector and testbench from moodle and place it into the same directory as your netlists.

Listing 4: testbench.tcl

```
vlib work ; # create library work
vlog testbench_add.v ; # add file to library
vlog add.v
vsim work.testbench_add ; # simulate module testbench_add in lib work
run 10 us ; # run simulation for 10 us
quit -f
```

Use `module load modelsim` to load the modelsim package. You can now execute the testbench `vsim -c -do "do testbench.tcl"`. It will tell you if your adder is working as expected or not. Modify the testbench for the multiplier and its testvector and test it as well.

## Task 5.2: Creating a Neuron

Using our multiplier and adder circuits we want to create a neuron. It will have 4 inputs and the given weights ($w_i$) and bias ($b$) in table 1. The neuron should also have a ReLU function.

Table 1: Neuron parameters

| Param | value |
|---|---|
| $w_0$ | -5 |
| $w_1$ | 10 |
| $w_2$ | 27 |
| $w_3$ | -13 |
| $b$ | 7 |

### 5.2a) Implementation

Extend the code below to implement the complete neuron. Use your multiplication and addition module for math operations. Do not forget to add code for the ReLU.

Listing 5: neuron.v

```
module NEURON (
    input wire signed [7:0] inp0,
    input wire signed [7:0] inp1,
    input wire signed [7:0] inp2,
    input wire signed [7:0] inp3,
    output wire signed [15:0] out
);
    parameter [7:0] w0 = -8'sd5;  // s: signed, d: decimal

    // your code here

endmodule
```

### 5.2b) Testing

Copy and modify the testbench/testvector[2] from the previous task to test your neuron implementation.

[2]Tip: you can use python to generate your testvector. `https://numpy.org/doc/stable/reference/generated/numpy.binary_repr.html`