Universität Bremen AGRA Fachbereich 3 Mathematik / Informatik

Prof. Dr. Rolf Drechsler, drechsler@uni-bremen.de, MZH 4330
Dr. Muhammad Hassan, hassan@uni-bremen.de, MZH 4280
Milan Funck, mfunck@uni-bremen.de, MZH 4260 Jan Zielasko, zielasko@uni-bremen.de, MZH 4184

1. Exercise sheet of the lecture

Computer architecture and embedded systems

Rechnerarchitektur und Eingebettete Systeme

For all tasks, unless specified otherwise, holds:

- Numbers are 8-bit wide.
- Only basic components are allowed to be used in the Digital[1] simulator (The drop-down elements Logic, IO, and Wires).
  - Other components may can be constructed and used through building *embedded circuit*s. This usage is encouraged if it helps readability of your submission.
- Multi-bit inputs and outputs shall be single elements.
- Implementation is limited to combinational logic.
- *Adequate* documentation of the circuits in the submitted `.dig` circuit files necessary for passing the exercise.

Submission: Every exercise is to be submitted in an individual `.dig` file with labeled inputs and outputs (they should reference each other using embedded circuits, however).

All relevant files, including a PDF of the textual answers, are to be archived using `zip` or `gzip` formats and sent no later than **26. November 2024** to `mfunck@uni-bremen.de` with the subject "*Submission Exercise 1 Group X*", where X is your group number.

Exercise 1

Build a non-rotating right shift and left shift function. The to-be-shifted number shall be labeled **A**, the length of the shift **B**, and the result shall be labeled **Y**. The shift direction should be determined based on a one-bit **Operand**:

Operand	Y
0	A << B
1	A >> B

Would this be easier if you were allowed to use sequential logic? What would be the runtime and cost of a real implementation?

Tip: Building a *Decoder* for the length will probably help.

Exercise 2

Build a carry-select adder with two 5-bit blocks and one 3-bit block of carry-ripple adders in such a way that it accepts two two's complement numbers (**A** and **B**) and outputs one number (**Y**). The carry-bit over the 8-bit width of Y can be discarded.

[1] See tutorial slides *Installing Digisim* or `https://github.com/hneemann/Digital`

Name some of the benefits and tradeoffs for carry-select adders.

Tip: Building a *Multiplexer* first will probably help.

Exercise 3

Build a seven-segment display[2] with as many digits as needed for a **16 bit** two's complement number. The display shall show negative and positive **decimal** numbers. The number shall be constructed by the input **A** as the low 8 bit and the input **B** as the high 8 bit.

Why is "add 6" to every (hexadecimal) digit with value > 9 a valid strategy?

Tip: Start "in reverse" with BCD-to-sevensegment, then hex-to-x-digit-BCD, and then *two's-complement*-to-*sign-value*.

Exercise 4

Combine all three logic functions into one ALU with the inputs **A**, **B**, **Operand** and the output **Y**. The function shall be determined by the following table:

Operand	Y
`000`	A << B
`001`	A >> B
`010`	A << 1
`011`	–*
`100`	A + B
`101`	A + 1
`110`	A − 1
`111`	1337_{10}

*: Display `(B << 4) | A` on sevensegments

[2] By *Seven-Segment Display* we mean the actual seven segment display under Components→IO→Displays→Seven-Segment Display. Not the Seven-Segment *Hex* Display. Where would be the fun in that?