



Ausgabe: 30. Oktober 2023

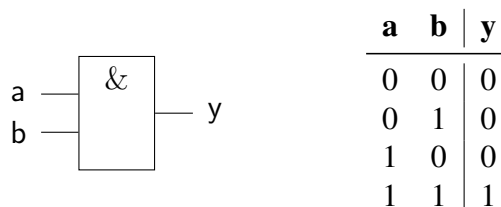
Abgaben	{		Theorie	entfällt
			Praxis	entfällt
			Rücksprache	entfällt

Aufgabe 1: Logik-Gatter

In dieser Aufgabe sollen drei grundlegende Gatter von Ihnen implementiert werden. Die Definition der Gatterfunktionalität ist jeweils durch eine Wahrheitstabelle vorgegeben.

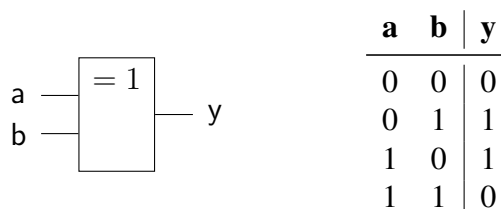
Die Funktionalität aller drei Gatter kann durch die Testbench `logic_tb` getestet werden. Beachten Sie dabei, dass in den vorgegebenen Dateien für die drei Gatter sowohl ein *entity*- als auch ein *architecture*-Abschnitt vorhanden sein muss, bevor die Testbench ausgeführt werden kann. Sollten Sie erst ein Teil der Gatter implementiert haben, können Sie die Fehler, welche die Testbench zu den noch nicht implementierten Gattern anzeigt, ignorieren. Zum Ausführen der Testbench müssen Sie in einem Terminal in den Ordner navigieren, in den Sie die Dateien aus der Vorgabe zu dieser Aufgabe entpackt haben. Führen Sie dann in dem Terminal den Befehl `make clean all` aus. Das Betrachten der Signalverläufe erfolgt dann mit dem Kommando `make view_wave`.

1. Das AND-Gatter



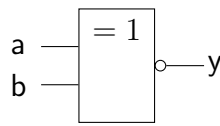
Implementieren Sie die Funktionalität des *AND*-Gatters in der Datei `and2.vhd`. Definieren Sie dafür eine neue *entity* mit dem Namen *and2* und eine *architecture* mit dem Namen *behavioral*.

2. Das XOR-Gatter



Implementieren Sie die Funktionalität des *XOR*-Gatters in der Datei `xor2.vhd`. Definieren Sie dafür eine neue *entity* mit dem Namen *xor2* und eine *architecture* mit dem Namen *behavioral*.

3. Das XNOR-Gatter



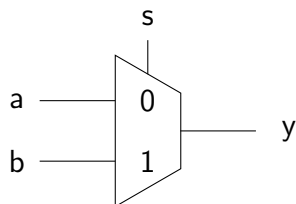
a	b	y
0	0	1
0	1	0
1	0	0
1	1	1

Implementieren Sie die Funktionalität des *XNOR*-Gatters in der Datei `xnor2.vhd`. Definieren Sie dafür eine neue *entity* mit dem Namen `xnor2` und eine *architecture* mit dem Namen `behavioral`.

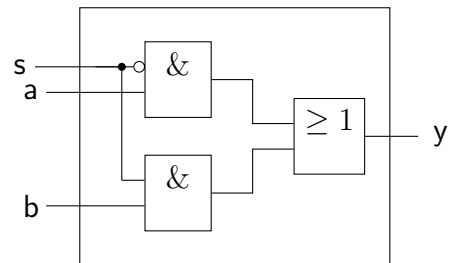
4. Simulation

Testen Sie Ihre Implementierung mit der Testbench `logic_tb` aus den Vorgaben.

Aufgabe 2: MUX2



s	a	b	y
0	0	–	0
0	1	–	1
1	–	0	0
1	–	1	1



Dieser Multiplexer schaltet abhängig von *s* die Eingänge *a* bzw. *b* nach *y* durch. Die in der Wahrheitstabelle mit – gekennzeichneten Eingänge sind für die Bestimmung Ausgabewerts nicht relevant („*don't care*“). Die Funktionalität des Bauteils soll daher für jeden mögliche Eingangswert an einem mit – gekennzeichneten Eingang sichergestellt sein.

Aus der Wahrheitstabelle kann ein Logikterm aus *AND*, *OR* und *NOT* aufgestellt werden (disjunktive Normalform): $y = a\bar{s} + bs$. Dieser kann wiederum mit einem einfachen Schaltnetz implementiert werden.

1. Legen Sie für den Multiplexer eine neue Datei `mux2.vhd` im Wurzelverzeichnis der Vorgaben an.
2. Erzeugen Sie eine *entity* "mux2" und implementieren Sie oben stehendes Schaltnetz in der *architecture* "behavioral".
3. Testen Sie Ihre Implementierung mit der Testbench `mux_tb` aus den Vorgaben.

Literatur

- [1] Jonas Tröger Max Uffke Drechsler. Gtkwavemod-repository. <https://git.tu-berlin.de/rorgpr-wise23/GTKWaveMod>.
- [2] Mentor Graphics Corporation. *ModelSim SE Reference Manual*, 6.4a edition.
- [3] Mentor Graphics Corporation. *ModelSim SE Tutorial*, 6.4a edition.

[4] Mentor Graphics Corporation. *ModelSim SE User's Manual*, 6.4a edition.