

Gerlach, Luisa	gerlaclu	599244
Heine, Tom Martin	heinetom	597978
Kühne, Marc Sebastian	kuehnese	599833
Seegert, Noah-Joël	segertno	596234

Aufgabe 1

Dezimal	Hexadezimal	Oktal	Binär	8-Bit-Zweierkomplement
-79	-4f	-117	-1001111	1011001
23	17	27	10111	01101001
212	da	324	1010100	00101100
63	3f	77	111111	01000001
-103	-67	-147	-1001111	10011001
-52	-34	-64	-1010100	11001100

Beispiel: -79

(i) Dezimal zu Binär:

$$\begin{aligned} -79 &= -(64+15) = -(64+8+7) = -(64+8+4+3) = -(64+8+4+2+1) \\ &= -(2^6 + 2^3 + 2^2 + 2^1 + 2^0) \Rightarrow \text{in Binärdarstellung: } -(1001111)_2 \end{aligned}$$

(ii) Binär zu Oktal:

Es gilt $8 = 2^3$, also lassen sich immer 3 Stellen zusammenfassen:

$$-(\underbrace{100}_{2^2=4} \underbrace{111}_{2^0=1} \underbrace{111}_{2^1=2^2+2^1+2^0=7})_2 = -(117)_8$$

(iii) Binär zu Hexadezimal:

Es gilt: $16 = 2^4$, also lassen sich immer 4 Stellen zusammenfassen:

$$-(\underbrace{100}_{2^2=4} \underbrace{111}_{2^0=1} \underbrace{111}_{2^1=2^2+2^1+2^0=7} \underbrace{11}_{2^3=8})_2 = -(4f)_{16}$$

(iv) Binär zu 8-Bit-Zweierkomplement

$$-(1001111)_2 \xrightarrow{\text{Binar}} \xrightarrow{\text{Komplementieren}} 10110001$$

komplementieren: ~~(01010000)~~

1 addieren: 10110001

mit 1 beginnend, da negative Zahl

Aufgabe 2

2. a) $(-2,83)_{10}$ in Zweierkomplementdarstellung:
Berechne Vor- und Nachkommazahl einzeln:

$$(2)_{10} = (10)_2 \quad | \quad \begin{array}{l} 0,83 \cdot 2 = 1,66 \\ 0,66 \cdot 2 = 1,32 \\ 0,32 \cdot 2 = 0,64 \\ 0,64 \cdot 2 = 1,28 \\ 0,28 \cdot 2 = 0,56 \\ 0,56 \cdot 2 = 1,12 \\ 0,12 \cdot 2 = 0,24 \\ 0,24 \cdot 2 = 0,48 \\ 0,48 \cdot 2 = 0,96 \\ 0,96 \cdot 2 = 1,92 \end{array}$$

$$\Rightarrow (2,83)_{10} = (101,11010100)_2$$

Bilde Zweierkomplement: $(101,00101100)_2$

Es gilt: $(-2,83)_{10} = (101,00101100)_2$

b) $(-3,42)_{10}$ in IEEE-754 single precision:
Für dieses Format gilt: 1 Bit Vorzeichen, 8 Bit Exponent und 23 Bit für das Mantissenbitmuster (ohne Hidden-Bit)

Umwandlung $(3,42)_{10}$ in Dualsystem:

$$(3)_{10} \Rightarrow (11)_2 \quad | \quad \begin{array}{l} 0,42 \cdot 2 = 0,84 \\ 0,84 \cdot 2 = 1,68 \\ 0,68 \cdot 2 = 1,36 \\ 0,36 \cdot 2 = 0,72 \\ 0,72 \cdot 2 = 1,44 \\ 0,44 \cdot 2 = 0,88 \\ 0,88 \cdot 2 = 1,76 \\ 0,76 \cdot 2 = 1,52 \\ 0,52 \cdot 2 = 1,04 \\ 0,04 \cdot 2 = 0,08 \end{array} \quad \begin{array}{l} 0,08 \cdot 2 = 0,16 \\ 0,16 \cdot 2 = 0,32 \\ 0,32 \cdot 2 = 0,64 \\ 0,64 \cdot 2 = 1,28 \\ 0,28 \cdot 2 = 0,56 \\ 0,56 \cdot 2 = 1,12 \\ 0,12 \cdot 2 = 0,24 \\ 0,24 \cdot 2 = 0,48 \\ 0,48 \cdot 2 = 0,96 \\ 0,96 \cdot 2 = 1,92 \\ 0,92 \cdot 2 = 1,84 \\ 0,84 \cdot 2 = 1,68 \\ 0,68 \cdot 2 = 1,36 \end{array}$$

$$\Rightarrow (3,42)_{10} = (11,0110101'111'0000101'000111)_2$$

$$= 2^1 (1,101'101'011'100'001'010'001'11)_2$$

\Rightarrow Exponent: $1+127 = 128$

128 in Dualzahl: 10000000

\Rightarrow gesuchtes Format gege durch

1	10000000	101	101	011	100	001	010	001	11
---	----------	-----	-----	-----	-----	-----	-----	-----	----

Aufgabe 3

3.1

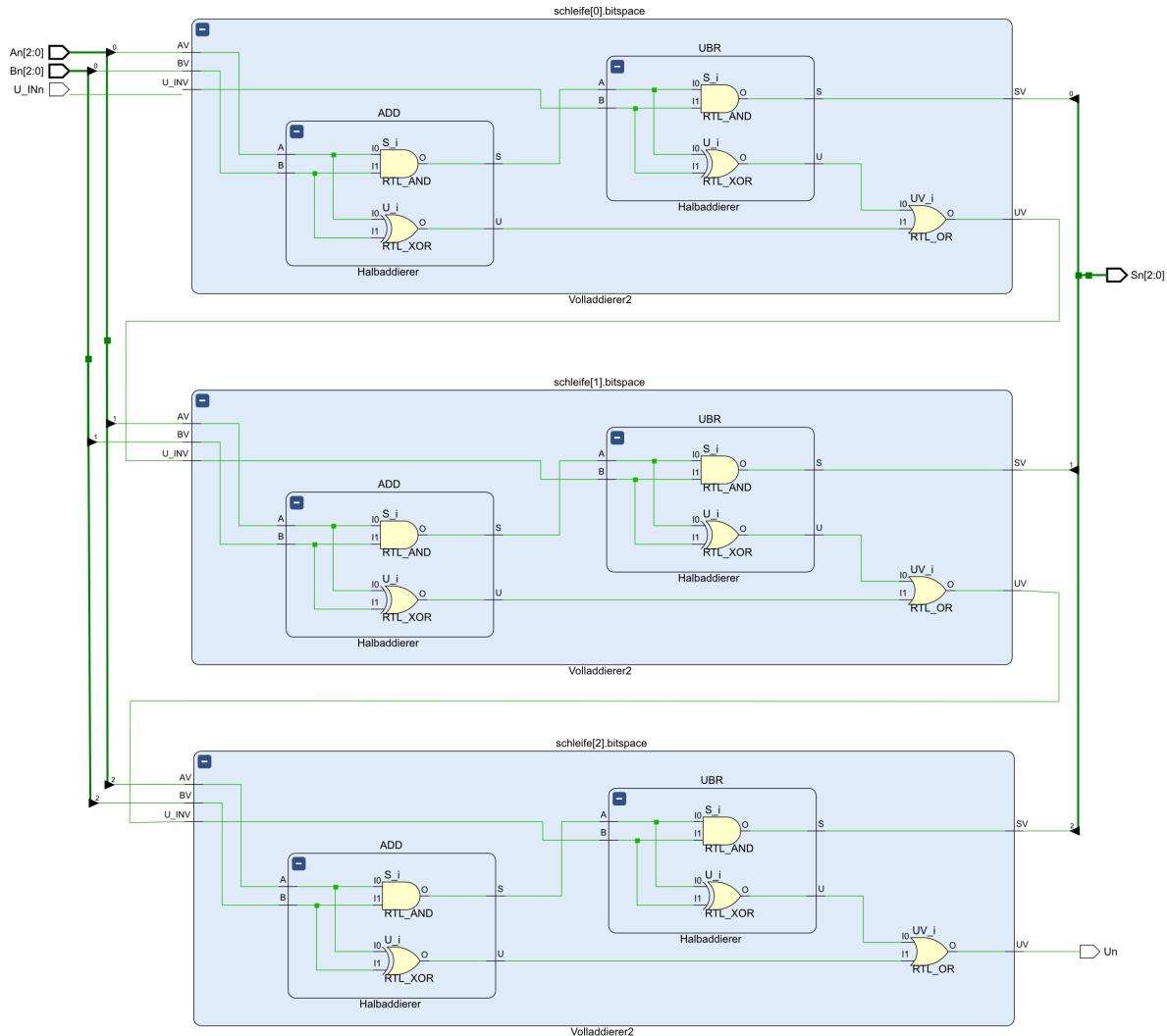


Abbildung 1: 3-bit Addierers auf Basis von Volladdierern auf Gatterebene

3.2

Für Gatterlaufzeit eines 4-bit-Addierers kann folgende berechnet werden.

(Die folgende Beschreibung orientiert sich an der in 3.1 verwendeten Notation. Für ein XOR-Gatter wird ebenfalls 12 ns angenommen, trotz dessen das in einigen Architekturen das XOR-Gatter durch 3 Gatter realisiert wird.)

Jeder Volladdierer kann parallel die ADD Halbaddierer berechnen (innerhalb dessen laufen die Gatter ebenfalls parallel ab), somit kostet das nur einmal 12 ns.

Ferner kostet jeder UBR Halbaddierer (innerhalb dessen laufen die Gatter ebenfalls parallel ab) 12ns. Zu guter Letzt muss um den Eingangsübergang zu berechnen noch das

OR-Gatter der Volladdierer berechnet werden. Es Resultiert die folgende Formel:

$$t(\text{bits}) = 12ns + 2 \cdot 12ns \cdot \text{bits} \quad (1)$$

Somit Resultiert für 4 bits eine Gatterlaufzeit von 108ns.

3.3

$s = a + b$, wobei sich das Ergebnis s für die Addition von zwei 4-Bit Zahlen a, b sich wie gefolgt in Bitschreibweise zusammensetzt: $s = c_4s_3s_2s_1s_0$

$s_i = a_i \oplus b_i \oplus c_i$ für $i \in \{0, 1, 2, 3\}$ mit c_0 als c_{in} und c_4 als letzten Übertrag

$$\begin{aligned}s_0 &= a_0 \oplus b_0 \oplus c_0 \\s_1 &= a_1 \oplus b_1 \oplus c_1 \\s_2 &= a_2 \oplus b_2 \oplus c_2 \\s_3 &= a_3 \oplus b_3 \oplus c_3 \\c_4 &\end{aligned}$$

Die Carries c_i lassen mit den folgenden Schaltfunktionen vorberechnen

$c_i = g_{i-1} \vee p_{i-1}c_{i-1}$ mit $g_i = a_i b_i$ und $p_i = a_i \vee b_i$, sodass

$$c_1 = g_0 \vee p_0 c_0 \quad (\text{Einmal ausführlich: } c_1 = a_0 b_0 \vee (a_0 \vee b_0) c_0)$$

$$c_2 = g_1 \vee p_1 g_0 \vee p_1 p_0 c_0$$

$$c_3 = g_2 \vee p_2 g_1 \vee p_2 p_1 g_0 \vee p_2 p_1 p_0 c_0$$

$$c_4 = g_3 \vee p_3 g_2 \vee p_3 p_2 g_1 \vee p_3 p_2 p_1 g_0 \vee p_3 p_2 p_1 p_0 c_0$$