GTI HS 20 Lösung Serie 10

Michael Baur, Tatjana Meier, Sophie Pfister

Die 10. Serie ist bis Montag, den 7. Dezember um 12:00 Uhr zu lösen und als PDF-Dokument via ILIAS abzugeben. Für Fragen steht im ILIAS jederzeit ein Forum zur Verfügung. Zu jeder Frage wird, falls nicht anders deklariert, der Lösungsweg erwartet. **Lösungen ohne Lösungsweg werden nicht akzeptiert**. Allfällige unlösbare Probleme sind uns so früh wie möglich mitzuteilen, wir werden gerne helfen.
Viel Spass!

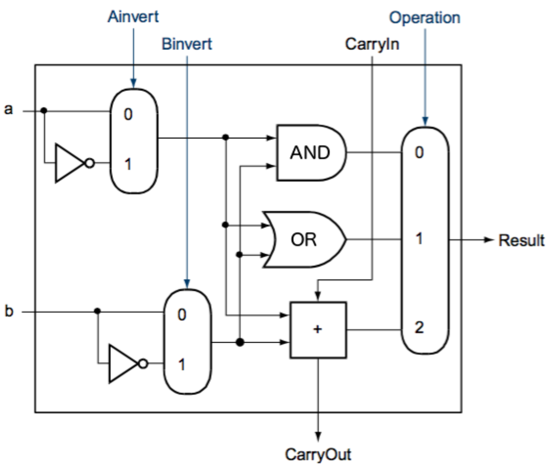
1 Operationen einer 1-Bit ALU (4 Punkte)

Für die gegebene 1-Bit ALU (siehe Abbildung) und die Funktion f, welche Werte müssen die Inputs `Ainvert`, `Binvert`, `CarryIn` und `Operation` haben, damit f(a,b) berechnet wird?

(a) (2 Punkte) f(a,b) = (a NAND b)

(b) (2 Punkte) f(a,b) = (a XOR b)

Tipp zu (a): (a NAND b) = $\neg(a \wedge b) \overset{\text{de Morgan}}{=} \ldots$



Lösung

(a) Es sind keine Modifikationen nötig, da (a NAND b) = $\neg(a \wedge b) \overset{\text{de Morgan}}{=} \neg a \vee \neg b$. Somit reicht es, nur die Kontroll-Inputs anzupassen:
`Ainvert` = 1, `Binvert` = 1, `CarryIn` = Don't Care, `Operation` = 01

(b) Es kann der Resultat-Ausgang des Addierers benutzt werden (dabei muss der `CarryIn` auf 0 gesetzt sein). `Ainvert` = 0, `Binvert` = 0, `CarryIn = 0`, `Operation` = 10

2 Overflow bei der Addition (2 Punkte)

Ein Überlauf (Overflow) entsteht, wenn das Ergebnis einer Operation nicht mit der verfügbaren Hardware dargestellt werden kann.

(a) (1 Punkt) Welcher Wertebereich (ganze Zahlen) kann durch ein 16-Bit Wort dargestellt werden, wenn die Zahlen

   (i) im Zweierkomplement

   (ii) unsigned (ohne Vorzeichen)

   dargestellt werden?

(b) (1 Punkt) Ein n-Bit Addierer nimmt als Inputs zwei signierte n-Bit Zahlen (signiert = mit Vorzeichen, Darstellung im Zweierkomplement). Der Output hat auch die Länge n – ein allenfalls entstehender Übertrag aus dem Addierer kann wegen der fixen Wortlänge nicht vom Prozessor verarbeitet werden.

   Beschreibe, wie man (ohne das Übertrags-Bit zu benutzen) erkennen kann, dass eine Addition zu einem Overflow geführt hat. Begründe deine Antwort.

Lösung

(a) (i) $\left[-2^{15}, 2^{15}-1\right] = [-32'768, 32'767]$ (generell: $\left[-2^{n-1}, 2^{n-1}-1\right]$)

   (ii) $[0, 2^n - 1] = [0, 65'535]$

(b) Wenn die MSB's der beiden Inputs verschieden sind, kann kein Overflow entstehen, denn wenn $x \in [-2^n, -1]$ und $y \in [0, 2^n - 1]$, dann ist $x + y \in [-2^n, 2^n - 2]$. Bei gleichem Vorzeichen bemerkt man einen Overflow daran, dass das Vorzeichen-Bit nach der Addition anders ist als bei den beiden Input-Zahlen.

3 4-Bit ALU (4 Punkte)

Eine 4-Bit ALU kann realisiert werden, indem man vier 1-Bit ALUs zusammenschaltet (siehe Abbildung). Diese 4-Bit ALU nimmt als Input eine 16-Bit lange Instruktion, welche wie folgt aufgebaut ist:

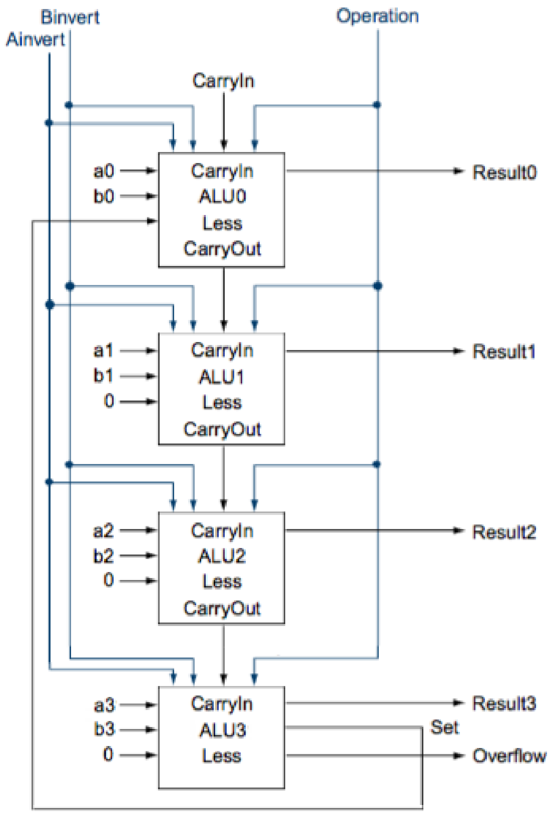
0–2	3	4	5	6	7	8	9	10	11	12	13	14–15
unused	a0	a1	a2	a3	b0	b1	b2	b3	AInvert	BInvert	CarryIn	Operation

(a) (1 Punkt) Welche Boolesche Funktion wird mit folgender Instruktion berechnet?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1

(b) (3 Punkte) Bestimme die Werte von `Result0`, `Result1`, `Result2`, `Result3`, `Set` und `Overflow` nachdem die Instruktion aus (a) fertig berechnet wurde.

1. Tipp zu (b): Der `CarryOut` jeder 1-Bit ALU wird immer berechnet, egal welchen Wert `Operation` annimmt.
2. Tipp zu (b): Die unterste 1-Bit ALU hat eine Overflow-detection (siehe Vorlesung).
3. Tipp zu (b): Die **MSBs** (Most Significant Bits) sind `a3`, `b3` und `Result3`. Die **LSBs** (Least Significant Bits) sind `a0`, `b0` und `Result0`.



Lösung

(a) a = 1001, b = 0101
(a OR ¬b)

(b)

Result = 1001||¬0101 = 1001||1010 = 1011
Result0 = 1
Result1 = 1
Result2 = 0
Result3 = 1

ADD: a+¬b = 1001+1010+1(CarryIn) = (1)0100
Set = 0
Overflow = 1

Note to self: Je nach dem, ob der Overflow erklärt wird oder nicht, ist dieses Bit lösbar oder dann halt eben nicht