Overflow-detectie in de 2's complement representatie

Jesse op den Brouw

3 oktober 2021

1 Introductie

In de digitale techniek worden rekenschakelingen ontworpen die gebruik maken van de 2's complement representatie. Hierin worden positieve en negatieve getallen uit een bepaald bereik voorgesteld op een ander niet-negatief domein [Thijssen, 1999]. Dat levert eenvoudige, compacte en snelle hardware op. De keerzijde is dat het voor mensen niet eenvoudig is om (de waarde van) van de negatieve getallen direct te herkennen.

Daarnaast levert het gebruik van de 2's complement representatie nog een probleem op: het resultaat van een bewerking kan buiten het bereik van de representatie liggen. Er is dan sprake van *overflow*.

Het detecteren van een overflow is een vereiste om de berekeningen in goede banen te laten lopen. Gelukkig is het detecteren van overflow geen moeilijke klus. Natuurlijk is de vraag: wat moet er gebeuren als er overflow geconstateerd is. Dat is aan de ontwerper van de schakeling.

In de onderstaande paragrafen wordt overflowdetectie bij optellen en aftrekken besproken.

2 De 2's complement representatie

In het dagelijkse leven gebruiken wij de zogenaamde teken-en-grootte representatie om getallen weer te geven en berekeningen uit te voeren. Het voordeel van deze representatie is dat in één oogopslag de grootte van het getal te zien is, of het nou positief of negatief is. Zo zijn de getallen $+13$ en -13 even groot maar tegengesteld.

Berekeningen uitvoeren in deze representatie is lastig. Willen we dit omzetten naar een digitale schakeling, dan levert dat erg veel hardware op. Beter is om een complement representatie te gebruiken. Complement representaties zijn gebaseerd op *modulair rekenen*. In hardware wordt dat al van nature gerealiseerd omdat er met een eindig aantal bits wordt gerekend.

Zo zijn met vier bits de getallen 0 t/m 15 ($2^4 - 1$) te realiseren, er zijn 16 (2^4) combinaties. Een optelling van twee getallen van vier bits levert een antwoord van vijf bits op, maar het vijfde bit wordt niet opgeslagen, er zijn immers maar vier bits beschikbaar. Alle berekeningen zijn dus *modulo* 2^4. Hieronder een paar voorbeelden.

Voorbeeld 2.1.

$$\begin{array}{rlcrlcrl}
(0+0) & \bmod 2^4 & = & 0 & \bmod 2^4 & = & 0 & \bmod 2^4 \\
(10+5) & \bmod 2^4 & = & 15 & \bmod 2^4 & = & 15 & \bmod 2^4 \\
 & & & & & & & \\
(13+9) & \bmod 2^4 & = & 22 & \bmod 2^4 & = & 6 & \bmod 2^4 \\
(15+15) & \bmod 2^4 & = & 30 & \bmod 2^4 & = & 14 & \bmod 2^4 \\
 & & & & & & & \\
(14+15) & \bmod 2^4 & = & 29 & \bmod 2^4 & = & 13 & \bmod 2^4 \\
(13+15) & \bmod 2^4 & = & 28 & \bmod 2^4 & = & 12 & \bmod 2^4 \\
(12+15) & \bmod 2^4 & = & 27 & \bmod 2^4 & = & 11 & \bmod 2^4
\end{array}$$

Wat bij de laatste drie voorbeelden opvalt, is dat wanneer er bij een getal 15 wordt opgeteld, het antwoord één lager is dan dat getal. We kunnen een getal dus met één verlagen door er 15 bij op te tellen. Merk op dat 15 gelijk is aan $2^4 - 1$.

We kunnen dit aantonen door de optelling van twee n-bits getallen a en b:

$$a + b \equiv a + b + 2^n \equiv a + (2^n + b) \pmod{2^n} \tag{1}$$

De operator $\equiv$ betekent dat de leden in (1) *congruent modulo* zijn. Vervangen we nu b door $-b$ dan volgt uit (1)

$$a + (-b) \equiv a + (2^n + (-b)) \equiv a + (2^n - b) \pmod{2^n} \tag{2}$$

We kunnen $-b$ dus representeren door het op te tellen bij een macht van 2:

$$-b \equiv 2^n + (-b) \equiv 2^n - b \pmod{2^n} \tag{3}$$

Merk trouwens op dat b niet willekeurig groot mag zijn. We geven nu twee voorbeelden.

Voorbeeld 2.2.
Stel $n = 4$ en $b = -3$. Dan volgt uit (3) dat -3 gerepresenteerd kan worden door $2^4 - 3 = 16 - 3 = 13$.

Stel $n = 8$ en $b = -58$. Dan volgt uit (3) dat -8 gerepresenteerd kan worden door $2^8 - 58 = 256 - 58 = 198$.

We willen graag met n bits een gelijkmatige verdeling van positieve en negatieve getallen dus we zoeken naar een balans in het aantal negatieve en positieve getallen. De meest voor de hand liggende verdeling is om de heeft van de combinaties voor de positieve getallen te gebruiken en de andere helft voor de negatieve getallen. We gebruiken één bit om het teken aan te geven zodat er $n-1$ bits overblijven voor de grootte. Hieruit volgt dat voor het bereik van eerder genoemde getal b geldt

$$-2^{n-1} \leq b \leq 2^{n-1} - 1 \tag{4}$$

Te zien is dat de verdeling niet symmetrisch is. Er is één meer getal negatief dan positief. Dat is te verklaren uit het feit dat het getal 0 als positief wordt gezien. Een nadeel is dat voor het getal -2^{n-1} geen tegengesteld getal gerepresenteerd kan worden, er bestaat dus geen $+2^{n-1}$. Er kan trouwens voor elke willekeurige verdelinggekozen worden zolang er maar 2^n mogelijke combinaties te maken zijn, maar op praktische gronden is gekozen voor de verdeling in (4).

Stel we nemen twee getallen B en b zodanig dat

$$\begin{aligned} 0 \leq b \leq 2^{n-1} - 1 &\leftrightarrow B = b \\ -2^{n-1} \leq b \leq -1 &\leftrightarrow B = 2^n + b \end{aligned} \tag{5}$$

dan wordt B de *two's complement representatie* van b genoemd.

Kijken we naar een positief getal en coderen dit met bits, dan zien we dat het getal met n bits ligt in het bereik 0 – $2^{n-1} - 1$. Het meest significante bit een 0. Een negatief getal wordt geschreven als $2^n - m$ met m in het bereik van 1 – 2^{n-1} zodat het meest significante bit een 1 is. De meest significante bits worden tekenbits genoemd.

3 Overflow bij optelling

Overflow in de 2's complement representatie kan eenvoudig worden gedetecteerd aan de hand van de te bewerken getallen, het resultaat en de bewerking.

Bij een optelling van twee 2's complement getallen kan een overflow alleen maar optreden als de twee getallen hetzelfde teken hebben, met de nadruk op *kan*. Dat is te zien in de onderstaande vergelijkingen. Hierbij worden de

grenzen van de 2's complement representatie opgezocht. Deze grenzen zijn -2^{n-1} en -1 voor negatieve getallen en 0 en $2^{n-1}-1$ voor positieve getallen.

$$-2^{n-1} + -2^{n-1} = -2^n \tag{6}$$
$$(-1) + (-1) = -2 \tag{7}$$
$$0 + 0 = 0 \tag{8}$$
$$(2^{n-1} - 1) + (2^{n-1} - 1) = 2^n - 2 \tag{9}$$

De vergelijkingen 6 en 9 leveren een overflow op, de vergelijkingen 7 en 8 doen dat niet.

Als we twee getallen met verschillend teken optellen kan er nooit overflow optreden. Ook dat is eenvoudig aan te tonen. We zoeken hiervoor weer de grenzen van het bereik van de 2's complement representatie op.

$$-2^{n-1} + 0 = -2^{n-1} \tag{10}$$
$$-1 + 0 = -1 \tag{11}$$
$$-2^{n-1} + (2^{n-1} - 1) = -1 \tag{12}$$
$$-1 + (2^{n-1} - 1) = 2^{n-1} - 2 \tag{13}$$

De antwoorden van vergelijkingen (10) t/m (13) liggen allemaal in het bereik van de 2's complement representatie.

Laten er vergelijking (6) en (7) nog eens bekijken waarbij we de optelling nu modulo 2^n beschouwen, dat wil zeggen dat het antwoord in n bits moet passen met zoals is gegeven in (4). Uit deze vergelijkingen is op te maken dat de optelling van twee negatieve getallen een antwoord tussen -2 en -2^n oplevert. Bij alle antwoorden die kleiner zijn dan -2^{n-1} is er sprake van overflow. We definiëren nu een getal p zodanig dat

$$-2^{n-1} + p < -2^{n-1} \qquad (\text{met} - 2^{n-1} \leq p \leq -1) \tag{14}$$

Dan volgt daaruit

$$-2^{n-1} + p \equiv -2^{n-1} + p + 2^n \equiv 2^{n-1} + p \pmod{2^n} \tag{15}$$

Vullen we in (15) alle mogelijke waarden van p in dan volgt dat bij een overflow bij optelling van twee negatieve getallen het antwoord positief wordt.

We herhalen het bovenstaande voor de optelling van twee positieve getallen. Het antwoord van deze optelling ligt tussen 0 en $2^n - 2$. Zie hiervoor de

vergelijkingen (8) en (9). Bij alle antwoorden de groter zijn dan $2^{n-1}-1$ is er sprake van overflow. We definiëren een getal q zodanig dat

$$2^{n-1}-1+q > 2^{n-1}-1 \qquad (\text{met } 1 \leq q \leq 2^{n-1}-1) \tag{16}$$

Dan volgt hieruit

$$(2^{n-1}-1)+q \equiv (2^{n-1}-1)+q-2^n \equiv -2^{n-1}-1+q \pmod{2^n} \tag{17}$$

In (17) vullen we alle mogelijke waarden van q in dan volgt dat bij een overflow bij optelling van twee positieve getallen het antwoord negatief wordt.

Resumerend kunnen we zeggen dat

Overflow treedt op als de tekens van de op te tellen getallen gelijk zijn en ongelijk zijn aan het teken van het antwoord

We geven twee voorbeelden waarbij overflow optreedt. We gebruiken hiervoor getallen van vier bits.

Voorbeeld 3.1.
In dit voorbeeld worden getallen met gelijk teken opgeteld met als resultaat dat het teken van het antwoord verandert.

1001		-7			0101		$+5$
1101	$+$	-3			0110	$+$	$+6$
1) 0110		-10?			0) 1011		$+11$?

Uit bovenstaande voorbeelden is eenvoudig te bepalen of er overflow is opgetreden door de tekenbits te vergelijken.

Stel dat we de op te telken getallen a en b noemen met als antwoord getal s (met a, b en s alle n bits getallen), dan kunnen we de detectie van overflow als volgt in een functie beschrijven:

$$V_{add} = \{(a_{n-1} = b_{n-1}) \neq s_{n-1}\} \tag{18}$$

Zetten we deze functie om in een schakelfunctie dan volgt:

$$V_{add} = \overline{a_{n-1}} \cdot \overline{b_{n-1}} \cdot s_{n-1} + a_{n-1} \cdot b_{n-1} \cdot \overline{s_{n-1}} \tag{19}$$

We bekijken nu de optelling van alleen de tekenbits van de getallen en nemen daar in de ingaande en uitgaande carry's mee. We kunnen de optelling dan uitschrijven als:

$C_{n-1}...$		ingaande carry's
$a_{n-1}...$		getal A
$b_{n-1}...$	$+$	getal B
$C_n)\ s_{n-1}...$		uitkomst S en uitgaande carry

Hierin zijn C_{n-1} en C_n resp. de ingaande en uitgaande carry's. We vullen nu voor a_{n-1}, b_{n-1} en s_{n-1} alle mogelijke combinaties is en berekenen daaruit de waarden voor C_n en C_{n-1}.

$a_{n-1} = 0$, $b_{n-1} = 0$, $s_{n-1} = 0$. Uit de optelling van a_{n-1} en b_{n-1} volgt dat $s_{n-1} = 0$. Om deze optelling kloppend te maken moet $C_{n-1} = 0$ zijn en dan volgt automatisch dat $C_n = 0$.

$a_{n-1} = 0$, $b_{n-1} = 0$, $s_{n-1} = 1$. De optelling van de twee sombits levert een 1 in het antwoord. Dat kan alleen als $C_{n-1} = 1$ en dan volgt dat $C_n = 0$. Tevens is er overflow opgetreden.

$a_{n-1} = 0$, $b_{n-1} = 1$, $s_{n-1} = 0$. Om uit de optelling van $0 + 1$ een 0 in het sombit te krijgen, moet $C_{n-1} = 1$ zijn. Dan volgt dat $C_n = 1$, immers $1 + 0 + 1 = 10$.

$a_{n-1} = 0$, $b_{n-1} = 1$, $s_{n-1} = 1$. Hieruit volgt dan $C_{n-1} = 0$ en $C_n = 0$.

$a_{n-1} = 1$, $b_{n-1} = 0$, $s_{n-1} = 0$. Deze situatie is vergelijkbaar met de derde situatie.

$a_{n-1} = 1$, $b_{n-1} = 0$, $s_{n-1} = 1$. Deze situatie is vergelijkbaar met de vierde situatie.

$a_{n-1} = 1$, $b_{n-1} = 1$, $s_{n-1} = 0$. Het optellen van $1 + 1$ levert als sombit een 0 op, dat kan alleen als $C_{n-1} = 0$. Verder volgt dat $C_n = 1$. Tevens is er overflow opgetreden.

$a_{n-1} = 1$, $b_{n-1} = 1$, $s_{n-1} = 1$. Het optellen van $1 + 1$ levert als sombit een 1 op, dat kan alleen als $C_{n-1} = 1$. Verder volgt dat $C_n = 1$.

We stellen voor alle combinaties van a_{n-1}, b_{n-1} en s_{n-1} een waarheidstabel op en tonen hierin de carry's en of er overflow optreedt.

Tabel 1: *Relaties tussen overflow en de carry's bij optelling van twee getallen*

a_{n-1}	b_{n-1}	s_{n-1}	C_n	C_{n-1}	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Uit tabel 1 blijkt dat overflow optreedt als $C_nC_{n-1} = 01$ of $C_nC_{n-1} = 10$.

$$\boxed{V_{add} = C_n \oplus C_{n-1}} \tag{20}$$

Overflow treedt op als bij de op als bij optelling van de tekenbits de ingaande en uitgaande carry's ongelijk zijn.