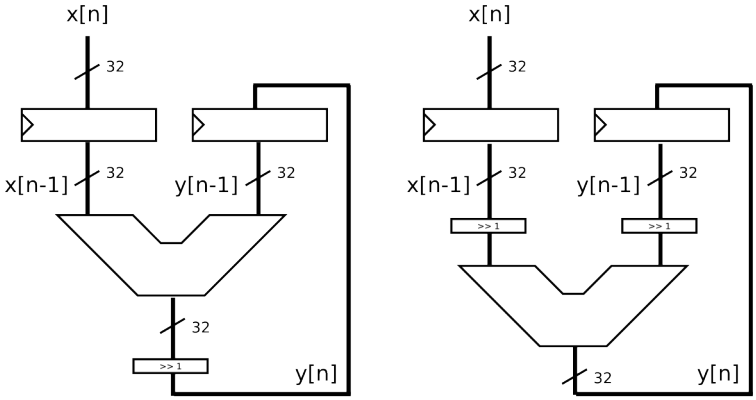
David Guillen Fandos

Exercicis previs DCISE sessió 3

Apartat 1:

En el cas enter no hi ha diferència d'eficiència. Això es deu a que una divisió per 2 és un desplaçament aritmètic a la dreta, que no és res més que connectar cables (cap porta lògica) de manera que el bit i passa a ser el bit i-1 i el bit de major pes es replica.



En el cas de fer ús de reals en coma flotant IEEE754 aleshores és més complicat donat que segueixen el format:

s eeeeeeee mmmmmmmmmmmmmmmmmmmmmmm

On *s* és el signe, *e* l'exponent i *m* la mantissa de manera que el nombre és en general 1,mmm..mm · 2^ee..e. És a dir, la mantissa amb un 1 implícit per dos elevat a l'exponent. L'exponent està codificat en excés 127.

Així si volem dividir un nombre per 2 en coma flotant necessitem un restador de 6 bits i una lògica per a detectar els casos de codificació especials com NaN, inifinits, etc. Així doncs serà millor fer ús d'una sola divisió que no pas de dues divisions, ja que així ens estalviem molts transistors. Per altra banda el retard serà aproximadament igual donat que els divisors treballaran en paral·lel.

Apartat 2:

En aquest cas dividir primer i sumar després provoca una pèrdua de precisió que apreciem a la taula. Per altra banda el fet que el sumador sigui de 32 bits provoca que hi pugui haver overflow i, per tant, el nombre doni la volta en el cos finit.

x[n-1]	y[n-1]	y[n] exacte	y[n] sol A	y[n] sol B
6	12	9	9	9
11	8	9.5	9	9
5	15	10	9	10
22	18	20	20	4
25	16	20.5	20	4
21	23	22	21	6

Marcats els errors per overflow (només en el cas B) i els errors per pèrdua de precisió (cas A).

Apartat 3:

Per a solucionar el cas B, el que podem fer és fer ús del bit d'overflow del sumador com a bit 32 del bus de sortida. Així la sortida té 33 bits i, com que ho desplacem, ens acabem quedant amb el bit d'overflow i els 31 bits de major pes de la sortida del sumador.

Per a solucionar el cas A cal veure que l'error es produeix al dividir abans de sumar, cosa que només succeeix quan depreciem el bit 0 que està a 1. Això vol dir que si el nombre és imparell cometem un error de 0.5. Aleshores dos errors de 0.5 produeixen un error de 1. Per tant en cas de tenir dos nombres imparells a l'entrada caldrà afegir 1 al resultat. Podem fer servir una AND entre els dos bits de menor pes de x i y com a entrada del carry 0 del sumador. Així sumarà x+y+1 si x i y son imparells.

Apartat 4:

El camí crític es defineix com el major retard possible entre la sortida d'un registre i l'entrada d'un altre registre o bé el mateix. Així tenim que els dos possibles camins són idèntics i passen per divisor i sumador en el cas A i B. El retard bé determinat pel retard del sumador, el temps de propagació del registre, el temps de hold i el temps del divisor (que en enters és 0, com ja s'ha dit).

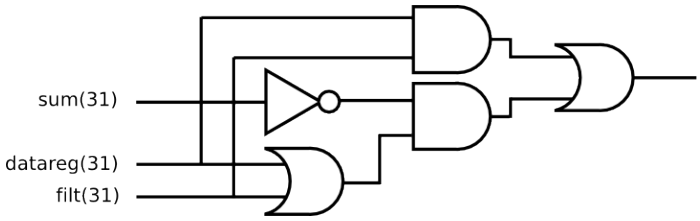
Apartat 5:

Carry look-ahead adder: Es basa en calcular els bits de carry fent ús d'un circuit combinacional de 2 nivells (AND,OR; NAND,NAND) de manera que el carry sigui molt ràpid, ja que aquest determina el camí crític. Típicament no es calculen tots els carry de cop, si no a blocs de N bits. Així el retard dels carry és el retard de dues portes entre el nombre de grups de carry. La sortida dels bits de resultat es calculen en paral·lel. És el més gran i el que té més retard comparativament però consumeix relativament poc.

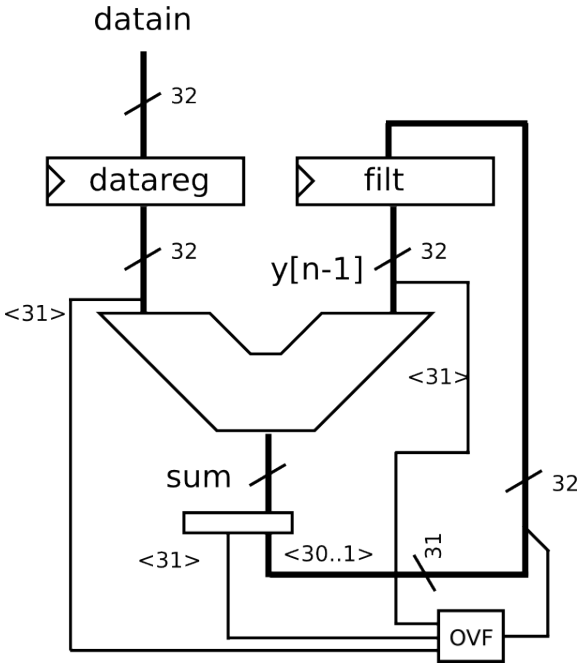
Sklansky adder: Es divideix en tres etapes: un preprocessat on es calcula el que d'alguna forma és el pre-carry (el carry sense conèixer el carry anterior), una etapa en arbre (amb alçada $\log_2 n$) que calcula els carry de veritat fent ús de dades de les etapes anteriors del bits de menor per i una última etapa que realitza la suma sabent els carry. És molt eficient ja que el retard és proporcional al logaritme del nombre de bits però té motes connexions, pel que el fan-out és gran i, per tant, el retard creix.

Redundant arithmetic adder: Es basa en fer ús de codificacions redundants (més d'una codificació per a un mateix nombre) de forma que no hi pugui haver carry en una operació de suma de 1 bit (el carry sempre és 0). Així totes les operacions s'efectuen en paral·lel. És dels més petits però el que més consumeix.

Apartat 6:



Apartat 7:



El circuit OVF és l'esmentat a l'apartat 6. Es veu clarament que és la solució que només té un divisor modificada per tal de corregir l'overflow.