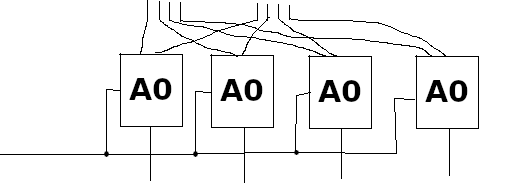
1.

(a)

ALU0 = [mfork 4, zip 4] ; zip 4 ; map 4 A0.

Or ALU0 = snd (zip 4); dstl 4; map 4 A0.

Diagram:



(b)

ALU1 = ALU0 ; fork ; snd ((apl 3)^~1 ; rdl 3 nand).

Diagram:

(c)

bitabs n = ?

A1 = snd fork ; snd (snd (bitabs 4)) ; if.

ALU2 = lsh ; snd ALU0 ; A1.

Diagram:

2.

(a)

col 0 R = <<>, x> $wire <x, <>>.

col n R = (R <|> (col (n – 1) R)) \\ snd (apl (n – 1)).

Type: <<X>\_n, Y> ~ <Y, <X>\_n> given R: <X, Y> ~ <Y, X>

col n R = [/\ n (D^~1), D^~n] ; col n (fst (D^~1) ; R ; [D, D]) ; snd (/\ n D).

Ignoring boundary anti-delays, the number of registers is n \* (1 + 1) + (0 + 1 + 2 + … + (n – 1)) = 2 \* n + 0.5 \* n \* (n – 1).

(b)

rdr 1 R = apr 1 ; R.

rdr n R = (apr (n – 1))^~1 ; lsh ; snd R ; rdr (n – 1) R.

Type: <<X>\_n, Y> ~ Y given R: <X, Y> ~ Y

rdr n R = col (R ; pi1^~1) ; pi1.

rdr n R = [/\ n (D^~1), D^~n] ; rdr n (fst (D^~1) ; R ; D).

(c)

rdr (m\*n) R = fst (group m n) ; rdr m (rdr n R).

This equation can be used for clustering R’s, allowing for partial pipelining when applied together with the equation from part (b). The values of m and n can be modified to change the level of pipelining: assuming that the total number of R’s is X, when m = 1 and n = X, we have a practically non-pipelined design, and when m = X and n = 1, we have a fully pipelined design, and when m = A and n = B such that A\*B = X, we have a partially pipelined design.

3.

(a)

DFEVar x = io.input(“x”, dfeInt(32));

DFEVar y = constant.var(dfeInt(32), 0);

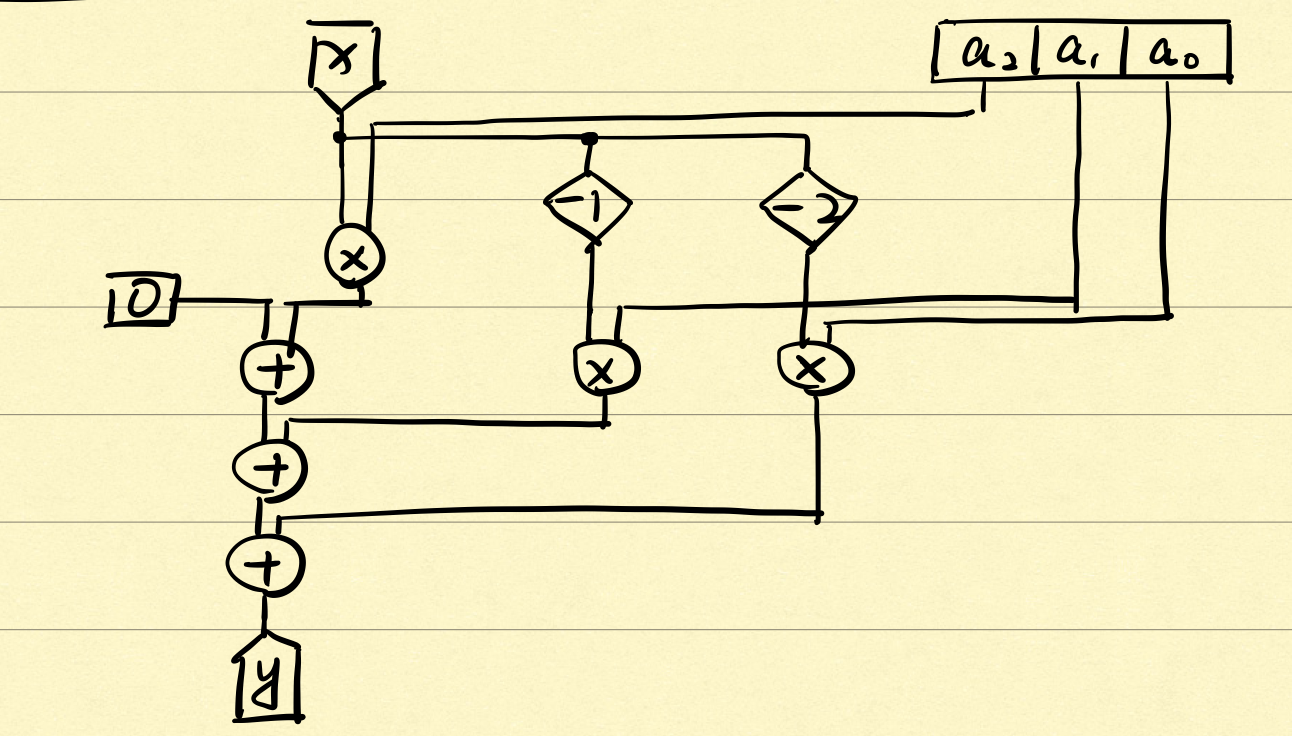
for (int j = 0; j < W; j++) {

y += a[j] \* stream.offset(x, -j);

}

Io.output(“y”, y, dfeInt(32));

(b)



(c) Tstream = max(N•Tc, 4N/B)

(d) total\_latency = M + W\*A

4.

Not examinable this year (2020/21).