Examen Architecture des ordinateurs – janvier 2020

Corrigé

1- max(A, B, C)

résultat = A si A >= B et A >= C
B si B > A et B >= C
C si C > A et C > B

Module max (A[7..0], B[7..0], C[7..0] : MAX[7..0])

Ucmp8 (B[7..0], A[7..0] : supBA, eqBA)
Ucmp8 (C[7..0], A[7..0] : supCA, eqCA)
Ucmp8 (C[7..0], B[7..0] : supCB, eqCB)
MAX[7..0] = A[7..0] * (/supBA * /supCA)
+ B[7..0] * (supBA * /supCB)
+ C[7..0] * (supCA * supCB

// il faut s'assurer que un et un seul terme est vrai

End module

Une autre solution plus simple : **max3(A, B, C) = max2(max2(A, B), C)**
Voir corrigé exam 20_21

2- Incrémenteur

Module Inc1 (Xi, Invi : Yi, Invip1)
Yi = Xi * /Invi + /Xi * Invi //inversion si invi=1
Invip1 = Invi * Xi
End module

Module Inc8 (X[7..0], inv0 : Y[7..0])
Inc1 (X[0], inv0 : Y[0], Inv1)
Inc1 (X[1], Inv1 : Y[1], Inv2)
…
Inc1 (X[6], Inv6 : Y[6], Inv7)
Inc1 (X[7], Inv7 : Y[7], Inv8)
End module

3- Divisible

Séquencement de l'algorithme (exécution des actions dans le temps)
Fini <- 0 et x <- A une transition
X <- x –B une transition qui sé répète dans la boucle Tantque
Divisible <- 1
Divisible <- 0 2 transitions alternatives entre 2 mêmes états
Donc 3 états : INIT, TANTQUE, FINI

Transition	condition	actions
INIT -> TANTQUE	go = 1 et B/=0	x <- A (FINI état déjà initialisé à 0)
INIT -> FINI	go = 1 et B=0	divisible <- 0
TANTQUE -> TANTQUE	go = 1 & x > 0	x <- x – B
TANTQUE -> FINI	go = 1 & x = 0	divisible <- 1 (erreur dans l'énoncé)
	go = 1 & x < 0	divisible <- 0
FINI -> FINI	go = 1	

Pour simplifier (comme au TD3), on assigne une bascule D à chaque état :

```
Module divisible (rst, clk, A[7..0], B[7..0], go : FINI, divisible)
        // transitions
        INIT := /go             on clk set when rst
        TANTQUE := INIT*go*/Bnul + TANTQUE*Xsup0*go      on clk reset when rst
        FINI := TANTQUE*/Xsup0*go + FINI*go + Bnul*go    on clk reset when rst

        Bnul = /B[7]*/B[6]*/B[5]*/B[4]*/B[3]*/B[2]*/B[1]*/B[0]
        // on peut aussi faire un module zero8 ou utiliser ucmp8

        //Xsup0
        ucmp8 (X[7..0], "00000000", XUsup0, Xeq0)
        Xsup0 = /X[7]*XUsup0

        // X : un registre
        reg8 (rst, clk, enX, eX[7..0] : X[7..0])
        enX = INIT + TANTQUE*Xsup0
        eX[7..0] = A[7..0]*INIT + XmoinsB[7..0]*TANTQUE*Xsup0

        // XmoinsB : un adder8
        NonB[7..0] = /B[7..0]
        adder8 (X[7..0], NonB[7..0], 1 : XmoinsB[7..0], cout)

        divisible = /X[7]*/X[6]*/X[5]*/X[4]*/X[3]*/X[2]*/X[1]*/X[0]*FINI*/Bnul
        // on peut aussi peut faire un module zero8 ou utiliser ucmp8
End module
```