```
module fulladder(a, b, cin : s, cout)
        aouexb = /a*b + a*/b
        s = /aouexb*cin + aouexb*/cin
        cout = a*/aouexb + cin*aouexb
end module

module adder8(a[7..0], b[7..0], cin : s[7..0], cout)
        fulladder(a[0], b[0], cin : s[0], c0)
        fulladder(a[1], b[1], c0 : s[1], c1)
        fulladder(a[2], b[2], c1 : s[2], c2)
        fulladder(a[3], b[3], c2 : s[3], c3)
        fulladder(a[4], b[4], c3 : s[4], c4)
        fulladder(a[5], b[5], c4 : s[5], c5)
        fulladder(a[6], b[6], c5 : s[6], c6)
        fulladder(a[7], b[7], c6 : s[7], cout)
end module

module adder32(a[31..0], b[31..0], cin : s[31..0], cout)
        adder8(a[7..0], b[7..0], cin : s[7..0], c0)
        adder8(a[15..8], b[15..8], c0 : s[15..8], c1)
        adder8(a[23..16], b[23..16], c1 : s[23..16], c2)
        adder8(a[31..24], b[31..24], c2 : s[31..24], cout)
end module

module count2(rst, clk, en : c[1..0])
        T[0] = 1
        T[1] = c[0]
        c[1..0] := /T[1..0]*c[1..0] + T[1..0]*/c[1..0] on clk reset when rst enabled when en
end module

module count4(rst, clk, en : c[3..0])
        count2(rst, clk, en : c[1..0])
        subclk := c[0]*c[1]*clk on clk reset when rst enabled when en
        count2(rst, subclk, en : c[3..2])
end module

module count_init4(rst, clk, en, init, e[3..0] : c[3..0])
        T[0] = 1*/init + init*(/e[0]*c[0] + e[0]*/c[0])
        T[1] = c[0]*/init + init*(/e[1]*c[1] + e[1]*/c[1])
        T[2] = c[1]*c[0]*/init + init*(/e[2]*c[2] + e[2]*/c[2])
        T[3] = c[2]*c[1]*c[0]*/init + init*(/e[3]*c[3] + e[3]*/c[3])
        c[3..0] := /T[3..0]*c[3..0] + T[3..0]*/c[3..0] on clk reset when rst enabled when en
end module

module count4_b1_b2(rst, clk, count, init, b1[3..0], b2[3..0] : c[3..0])
        init_b1 = init*/count
        en = /init*count

        T[0] = 1*en*/c_equal_b2 + init_b1*(/b1[0]*c[0] + b1[0]*/c[0])
        T[1] = c[0]*en*/c_equal_b2 + init_b1*(/b1[1]*c[1] + b1[1]*/c[1])
        T[2] = c[1]*c[0]*en*/c_equal_b2 + init_b1*(/b1[2]*c[2] + b1[2]*/c[2])
        T[3] = c[2]*c[1]*c[0]*en*/c_equal_b2 + init_b1*(/b1[3]*c[3] + b1[3]*/c[3])

        ucmp4(b2[3..0], c[3..0] : sup, c_equal_b2)
        c[3..0] := /T[3..0]*c[3..0] + T[3..0]*/c[3..0] on clk reset when rst
end module

module count8_b1_b2(rst, clk, count, init, b1[7..0], b2[7..0] : c[7..0])
        // compteur iniialisé avec b1 quand init=1 et count=0, compte quand init=0 et count=1 et s        'arrête quand il atteint b2

        // l'utilisation de en peut s'avérer problématique si le signal en est composé !!! bug
        c[7..0] := /t[7..0]*c[7..0] + t[7..0]*/c[7..0] on clk reset when rst

        t[0] = /init*count*sup + init*/count* (b1[0]*/c[0] + /b1[0]*c[0])
        t[1] = /init*count*sup*c[0] + init*/count* (b1[1]*/c[1] + /b1[1]*c[1])
        t[2] = /init*count*sup*c[0]*c[1] + init*/count* (b1[2]*/c[2] + /b1[2]*c[2])
        t[3] = /init*count*sup*c[0]*c[1]*c[2] + init*/count* (b1[3]*/c[3] + /b1[3]*c[3])
        t[4] = /init*count*sup*c[0]*c[1]*c[2]*c[3] + init*/count* (b1[4]*/c[4] + /b1[4]*c[4])
        t[5] = /init*count*sup*c[0]*c[1]*c[2]*c[3]*c[4] + init*/count* (b1[5]*/c[5] + /b1[5]*c[5])
        t[6] = /init*count*sup*c[0]*c[1]*c[2]*c[3]*c[4]*c[5] + init*/count* (b1[6]*/c[6] +      /b1[6]*c[6])
        t[7] = /init*count*sup*c[0]*c[1]*c[2]*c[3]*c[4]*c[5]*c[6] + init*/count* (b1[7]*/c[7] +          /b1[7]*c[7])

        ucmp8(b2[7..0], c[7..0] : sup, eq)
end module
```

```
module count_012789(rst, clk, en : c[3..0])
        init7 = /c[3]*/c[2]*c[1]*/c[0] // quand on est à 2 pour passer à 7
        init0 = c[3]*/c[2]*/c[1]*c[0] // quand on est à 9 pour passer à 0
        init = init7 + init0
        e[3..0] = "0000"*init0 + "0111"*init7
        count_init4(rst, clk, en, init, e[3..0] : c[3..0])
end module

module decoder2to4(e[1..0] : s[3..0])
        s[0] = /e[1] * /e[0]
        s[1] = /e[1] * e[0]
        s[2] = e[1] * /e[0]
        s[3] = e[1] * e[0]
end module

module decoder3to8(e[2..0] : s[7..0])
        decoder2to4(e[1..0] : s_temp[3..0])
        s[3..0] = /e[2]*s_temp[3..0]
        s[7..4] = e[2]*s_temp[3..0]
end module

module decoder4to16(e[3..0] : s[15..0])
        decoder3to8(e[2..0] : s_temp[7..0])
        s[7..0] = /e[3]*s_temp[7..0]
        s[15..8] = e[3]*s_temp[7..0]
end module

module ucmp1(a, b : sup, equ)
        sup = a * /b
        equ = a * b + /a * /b
end module

module ucmp2(a[1..0], b[1..0] : sup, equ)
        ucmp1(a[0], b[0] : su0, eq0)
        ucmp1(a[1], b[1] : su1, eq1)
        sup = su1 + eq1*su0
        equ = eq0*eq1
end module

module ucmp4(a[3..0], b[3..0] : sup, equ)
        ucmp2(a[1..0], b[1..0] : su0, eq0)
        ucmp2(a[3..2], b[3..2] : su1, eq1)
        sup = su1 + eq1*su0
        equ = eq0*eq1
end module

module reg8_D(rst, clk, en, e[7..0] : sr[7..0])
        sr[7..0] := e[7..0] on clk reset when rst enabled when en
end module

module reg8_T(rst, clk, en, e[7..0] : sr[7..0])
        T[7..0] = /e[7..0]*sr[7..0] + e[7..0]*/sr[7..0]
        sr[7..0] := /T[7..0]*sr[7..0] + T[7..0]*/sr[7..0] on clk reset when rst enabled when en
end module

module incr1(x, inv_i : y, inv_iplus1)
        y = x*/inv_i + /x*inv_i
        inv_iplus1 = inv_i * x
end module

module incr8(x[7..0], inv0 : y[7..0])
        incr1(x[0], inv0 : y[0], inv1)
        incr1(x[1], inv1 : y[1], inv2)
        incr1(x[2], inv2 : y[2], inv3)
        incr1(x[3], inv3 : y[3], inv4)
        incr1(x[4], inv4 : y[4], inv5)
        incr1(x[5], inv5 : y[5], inv6)
        incr1(x[6], inv6 : y[6], inv7)
        incr1(x[7], inv7 : y[7], inv8)
end module

module divisible(rst, clk, a[7..0], b[7..0], go : FINI, divisible)
        //divisible = 1 si a divise b, 0 sinon
```

```
	Bnul = /b[7]*/b[6]*/b[5]*/b[4]*/b[3]*/b[2]*/b[1]*/b[1]*/b[0]

	// Les états du graphe
	INIT := /go on clk set when rst
	DECRB := INIT*go*/Bnul + DECRB*go*xsup0 on clk reset when rst
	FINI := FINI*go + Bnul*go + DECRB*go*/xsup0 on clk reset when rst

	//xsup0
	ucmp8(x[7..0], "00000000" : xUsup0, xeq0)
	xsup0 = /x[7]*xUsup0

	//X stocké dans un registre
	reg8_D(rst, clk, enx, ex[7..0] : x[7..0])
	enx = INIT + DECRB*xsup0
	ex[7..0] = a[7..0]*INIT + xmoinsb[7..0]*DECRB*xsup0

	//xmoinsb : complément à deux
	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
end module

module max3(a[7..0], b[7..0], c[7..0] : max[7..0])
	// max = max(a,b,c)

	ucmp8(a[7..0], b[7..0] : asupb, aeqb)
	maxab[7..0] = a[7..0]*asupb + b[7..0]*/asupb

	ucmp8(maxab[7..0], c[7..0] : maxabsupc, maxabeqc)
	max[7..0] = maxab[7..0]*maxabsupc + c[7..0]*/maxabsupc
end module

module min3(a[7..0], b[7..0], c[7..0] : min[7..0])
	ucmp8(a[7..0], b[7..0] : asupb, aeqb)
	minab[7..0] = a[7..0]*/asupb + b[7..0]*asupb

	ucmp8(minab[7..0], c[7..0] : minabsupc, minabeqc)
	min[7..0] = minab[7..0]*/minabsupc + c[7..0]*minabsupc
end module

module etats_cal_max(rst, clk, cal, finTab : INITAD, INITMAX, MAJMAX, FINI)
	INITAD := /cal on clk set when rst
	INITMAX := cal*INITAD on clk reset when rst
	MAJMAX := cal*INITMAX + cal*MAJMAX*/finTab on clk reset when rst
	FINI := cal*MAJMAX*finTab + cal*FINI on clk reset when rst
end module

module cal_max(rst, clk, cal, ad1[7..0], ad2[7..0], elemCour[31..0] : adCour[7..0], max[31..0],  adMax[7..0], INITAD,
INITMAX, MAJMAX, FINI)
	etats_cal_max(rst, clk, cal, finTab : INITAD, INITMAX, MAJMAX, FINI)
	ucmp8(adCour[7..0], ad2[7..0], sup, finTab)

	count8_b1_b2(rst, clk, enAdC, initAdC, ad1[7..0], ad2[7..0] : adCour[7..0])
	initAdC = cal*INITAD
	enAdC = cal*INITMAX + cal*MAJMAX*/finTab

	reg8_D(rst, clk, enAdMax, adCour[7..0] : adMax[7..0])
	enAdMax = cal*INITMAX + cal*MAJMAX*supMax

	ucmp32(elemCour[31..0], max[31..0] : supMax, eqMax)

	reg32_D(rst, clk, enAdMax, elemCour[31..0] : max[31..0])
end module

module max_tab(rst, clk, cal, ad1[7..0], ad2[7..0] : max[31..0])
	$ram_aread_swrite(clk, 0, adCour[7..0], "00000000000000000000000000000000" :  elemCour[31..0])

	cal_max(rst, clk, cal, ad1[7..0], ad2[7..0], elemCour[31..0] : adCour[7..0], max[31..0],  adMax[7..0], INITAD, INITMAX,
MAJMAX, FINI)
end module
```