Task-uri și funcții

Task-uri

- se definesc în interiorul modulelor în care se folosesc. Se poate defini un task într-un fișier separat și să se folosească `include.
- pot conține elemente de timing precum posedge, negedge sau întârzieri
- pot avea un număr de intrări și un număr de ieșiri. Ordinea în care se face declararea intrărilor și ieșirilor definește ordinea în care variabilele sunt furnizate la apelul task-ului.
- se pot declara variabile în interior și acestea vor fi locale
- pot lucra cu variabile globale, definite în modulul în care este definit task-ul.Se pot folosi variabilele globale în expresii sau li se pot da valori
- când sunt folosite variabile locale, valorile ieșirilor sunt asignate la sfârșitul execuției task-ului
- pot apela un alt task sau o funcție
- pot modela logică secvențială sau combinațională
- trebuie apelate explicit
- nu pot face parte din expresii
- sintaxa:

```
task <nume_task>;
        [declaratii_intrari;]
        [declaratii_iesiri;]
        [declaratii_variabile_locale;]
        [begin]
                [asignari;]
        [end]
endtask
```

Task care folosește variabile globale:

```
module modul_task_variabile_globale(…….);
    reg [7:0] a, b;
    reg [7:0] a_and_b, a_or_b, a_xor_b;
    always @(a or b)
    begin
        oper_bit;
    end
    ……..
    task oper_bit;
        begin
            a_and_b = a & b;
            a_or_b = a | b;
            a_xor_b = a ^ b;
        end
    endtask
    ……..
endmodule
```

Task cu intrări și ieșiri proprii:

```
module adder8(a, b, cin, s, cout);
    input [7:0] a, b;
    input cin;
    output [7:0] s;
    output cout;
        assign {cout,s} = a + b + cin;
endmodule

module test_adder8;
    reg [7:0] at, bt;
    reg cint;
```

```
        wire [7:0] st;
        wire coutt;
        adder8 adder8_inst(at, bt, cint, st, coutt);
        task stimuli;
                input [7:0] val_at, val_bt;
                input val_cint;
                begin
                at = val_at;
                bt = val_bt;
                cint = val_cint;
                // #10;
                end
        endtask
        initial
        begin
                #0 stimuli(8'haa, 8'h55, 1'b0);
                #10 stimuli(8'haa, 8'h55, 1'b1);
                #10 stimuli(8'hf0, 8'h0f, 1'b0);
                #10 stimuli(8'hf0, 8'h0f, 1'b1);
                #10 stimuli(8'h34, 8'he7, 1'b1);
                .....................
        end
endmodule
```

Funcții

- sunt definite în modulul în care se folosesc. Se pot defini funcții în fișiere separate și să se folosească `include.
- nu pot fi incluse elemente de timing. O funcție este executată în timp de simulare 0
- pot avea oricâte intrări, dar o singură ieșire
- variabilele declarate în interior sunt locale funcției. Ordinea declarării intrărilor furnizează ordinea în care variabilele sunt pasate funcției la apelarea acesteia
- se pot folosi variabile globale, declarate în modulul în care apare descrierea funcției

- atunci când se folosesc variabile locale, rezultatul este asignat la sfârșitul execuției funcției
- pot fi folosite pentru modelarea de logică combinațională
- pot apela alte funcții, dar nu pot apela task-uri
- sintaxa:

function <nume_functie>;

[declaratii_intrari;]

[declaratii_variabile_locale;]

[begin]

[asignari;]

[end]

endfunction

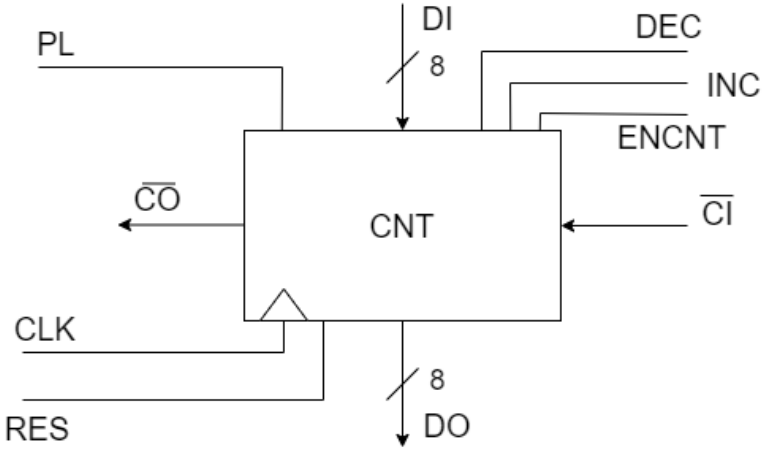
Exemplu de descriere și utilizare a unei funcții

```
module exemplu_functie(a, b, c, d, e, z);
    input a, b, c, d, e;
    output z;
    function functie_logica;
        input val_a, val_b, val_c, val_d;
        begin
            functie_logica = ((val_a & val_b) | (val_c & val_d));
        end
    endfunction
    assign z = e ? functie_logica(a, b, c, d) : 0;
endmodule
```

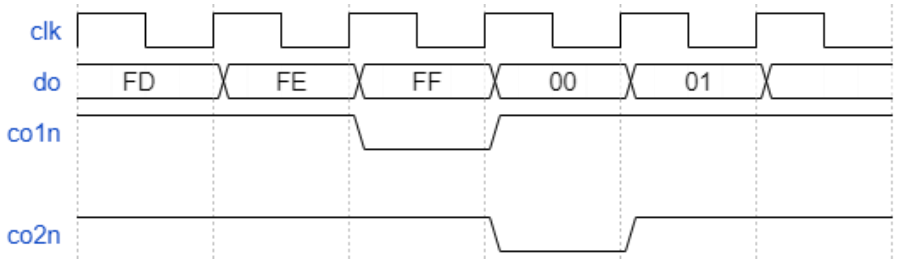
Am2940

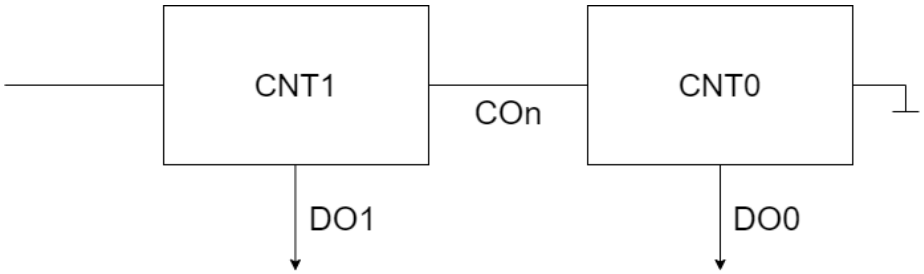
Numărătoarele

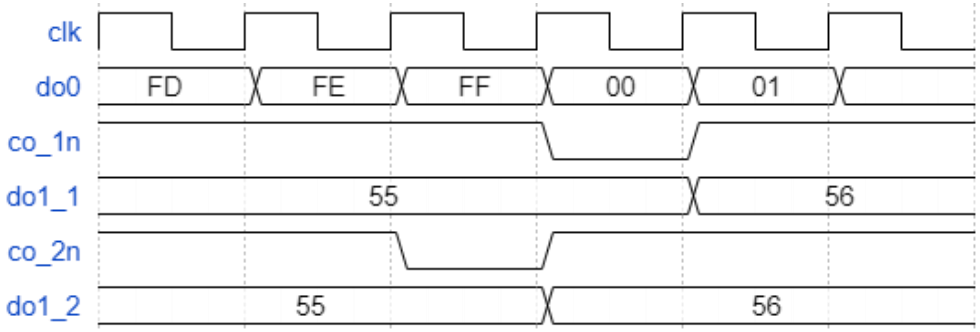
Porturi:

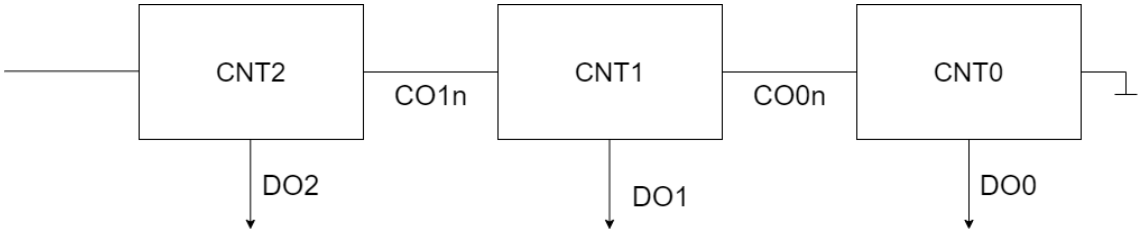


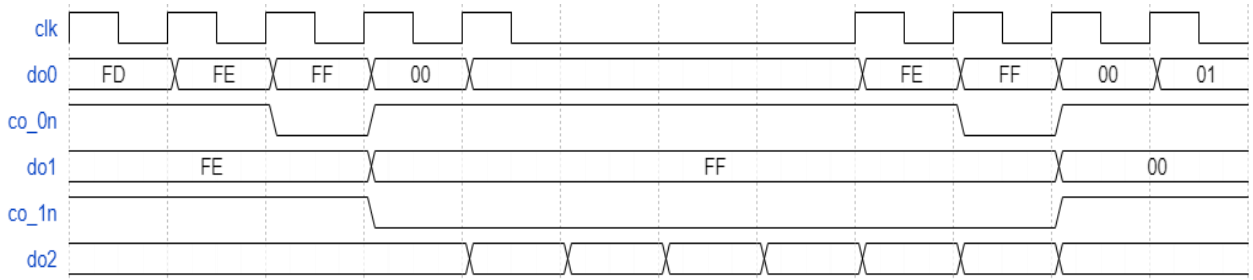
Generare COn:




CNT1
COn
CNT0
DO1
DO0


clk
do0
FD
FE
FF
00
01
co_1n
do1_1
55
56
co_2n
do1_2
55
56


CNT2
CO1n
CNT1
CO0n
CNT0
DO2
DO1
DO0



Address Reg

di(8), do(8)

clk, **plar**

Word Reg

di(8), do(8)

clk, **plwr**

Address Mux

di0(8), di1(8), do(8)

sela

sela = 0 -> do = di0 datain

sela = 1 -> do = di1 addreg

Word Mux

di0(8), di1(8), do(8)

selw

selw = 0 -> do = di0 datain

selw = 1 -> do = di1 wordreg

Ctrl Reg

di(3), do(3)

clk, **plcr**

Data Mux

di0(8), di1(8), di2(3), do(8)

seld(2)

seld =2'b00 out = di0 add cnt

seld = 2'b01 out = di1 word cnt

seld = 2'b1x out = {5'b11111,di2} di2 <- cr

```
module mux1_3(di0, di1, di2, do, sel);
        input [7:0] di0, di1;
        input [2:0] di2;
        output reg [7:0] do;
        input [1:0] sel;
        always @(di0 or di1 or di2 or sel)
                casex(sel)
                        2'b00: do = di0;
                        2'b01: do = di1;
                        2'b1x: do = {5'b11111,di2};
                endcase
endmodule
```

Address Cnt

clk, **plac, ena, inca, deca**, cina

di(8)

cona

do(8)

Word Cnt

clk, **resw, plwc, enw, incw, decw**, cinw

di(8)

conw

do(8)

Data Bus

oedata

Done Gen

cr[1:0], cinw, dowc, dowr, doac

```
module done_gen(dowc, dowr, doac, cinw, mode, done);
        input [7:0] dowc, dowr, doac;
        input [1:0] mode;
        input cinw;
        output reg done;

        always @(dowc or dowr or doac or cinw or mode)
                casex({mode,cinw})
                        3'b00_0: done = (dowc === 8'b1);
```

```
                3'b00_1: done = ~(|dowc);
                3'b01_0: done = (dowc + 1 === dowr);
                3'b01_1: done = (dowc === dowr);
                3'b10_x: done = (dowc === doac);
                3'b11_x: done = 1'b0;
            endcase
endmodule
```

I[2:0]	CR[2:0]	plar	plwr	sela	selw	plcr	seld	plac	ena	inca	deca	resw	plwc	enw	incw	decw	oedata
000	xxx	0	0	x	x	1	xx	0	0	0	0	0	0	0	0	0	0
001	xxx	0	0	x	x	0	1x	0	0	0	0	0	0	0	0	0	1
010	xxx	0	0	x	x	0	01	0	0	0	0	0	0	0	0	0	1
011	xxx	0	0	x	x	0	00	0	0	0	0	0	0	0	0	0	1
100	xx0,x11	0	0	1	1	0	xx	1	0	0	0	0	1	0	0	0	0
	x01	0	0	1	x	0	xx	1	0	0	0	1	0	0	0	0	0
101	xxx	1	0	0	x	0	xx	1	0	0	0	0	0	0	0	0	0
110	xx0,x11	0	1	x	0	0	xx	0	0	0	0	0	1	0	0	0	0
	x01	0	1	x	x	0	xx	0	0	0	0	1	0	0	0	0	0
111	000	0	0	x	x	0	xx	0	1	1	0	0	0	1	0	1	0
	0x1	0	0	x	x	0	xx	0	1	1	0	0	0	1	1	0	0
	010	0	0	x	x	0	xx	0	1	1	0	0	0	0	0	0	0
	100	0	0	x	x	0	xx	0	1	0	1	0	0	1	0	1	0
	1x1	0	0	x	x	0	xx	0	1	0	1	0	0	1	1	0	0
	110	0	0	x	x	0	xx	0	1	0	1	0	0	0	0	0	0