PROGETTAZIONE DI SISTEMI DIGITALI

[Fotocopie di Appunti PARTE 1]

A CURA DI ALESSANDRO PAGHI

PROFESSORE: Roberto Saletti (http://unimap.unipi.it/cercapersone/dettaglio.php?ri=5663&template=dettaglio.tpl)

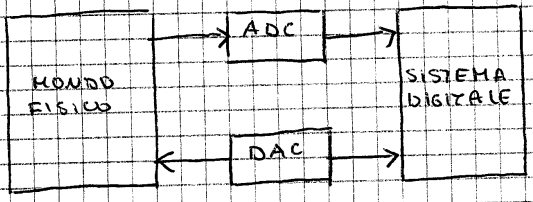
LINK AL CORSO ANNO 2017/2018: http://unimap.unipi.it/registri/dettregistriNEW.php?re=2087614::::&ri=8145

FREQUENTAZIONE: Consigliata.

Roberto Saletti 25/09/2017

Libro: Digital Design: Principle & Practice
Wakerly

Usi della Logica Binaria:
- Numerici Logiche → Applicazioni di controllo.
- Rappresentazione di numeri → Applicazioni di calcolo



Progettazione di un sistema digitale:
- Analisi delle Specifiche;
- Sintesi del Progetto;
- Implementazione del Progetto.

Il partizionamento del progetto consente di definire una gerarchia dell'architettura proposta semplificando il lavoro da svolgere.

La soluzione tecnologica scelta influenza pesantemente le scelte progettuali

Soluzioni tecnologiche per la realizzazione di un sistema digitale:
- Componentistica discreta:
  → la soluzione proposta è una soluzione estremamente rigida.
- Componentistica programmabile:
  → la soluzione proposta è estremamente flessibile
- ASIC:

→ Prestazioni elevate ma alta rigidità

- Circuiti General Purpose (μC, μP, DSP):

→ Soluzione estremamente flessibile ma non ottimizzata dal punto di vista delle prestazioni.

26/09/2017

Il progettista ha il compito di convertire le specifiche progettuali in primitive logiche.

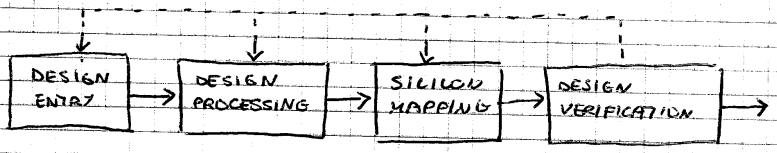
CAD

Ne abbiamo a disposizione di due tipi:

1) Strumenti di sintesi: aiuta il progettista ad implementare e spogliare la gerarchia verso il basso → Consente di sintetizzare la gerarchia

2) Strumenti di verifica: controlla la validità della sintesi effettuata. (Simulatore)

Flusso Progettuale per la realizzazione di un sistema digitale



Design Entry: Descrizione progetto in termini di funzionalità ed architettura.

→ In questa fase il progetto potrebbe essere technology independent

Design Processing: Traduzione della funzionalità in primitive e connessioni tra le stesse → Ho sintetizzato la struttura.

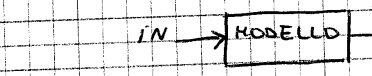
Silicon Mapping: trasferimento della sintesi sul silicio.

Design Verification: fase in cui si analizza se le specifiche sono state rispettate.

Design Entry, composta da:
- Schematic Entry: descrizione schematica.
- Text Entry: descrizione testuale.
  → consente di effettuare una descrizione comportamentale e non strutturale.

Il termine del Design Processing ottengo una netlist.

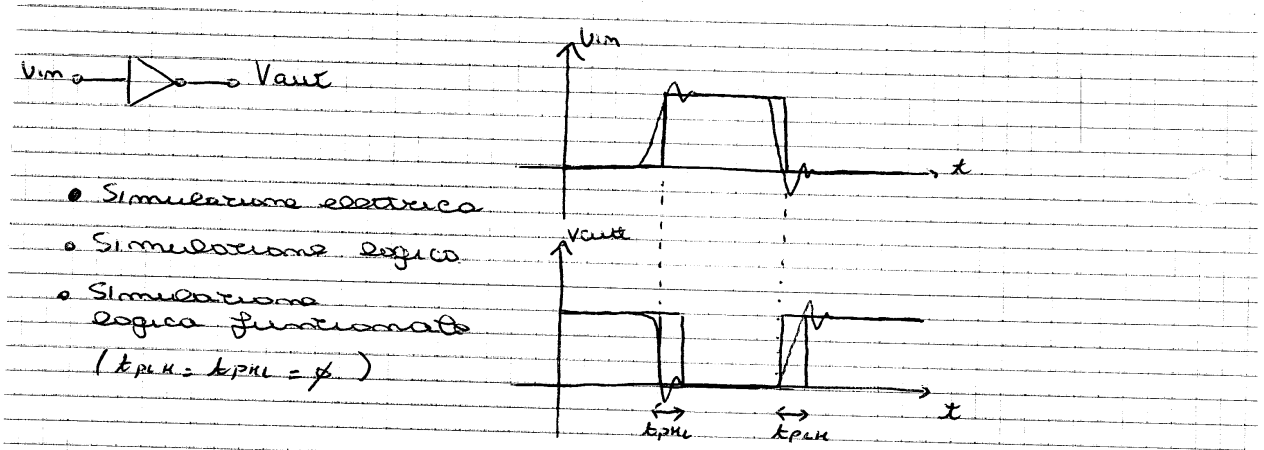
Simulatori
Traducono il sistema fisico in un modello.
Stimolano il modello con degli ingressi e mostrano le uscite.
Il progettista poi si occupa di verificare la validità di quei risultati.



- Simulatore elettrico: analizza fisicamente e risponde elettricamente al sistema
- Simulatore logico: traduce i segnali elettrici in livelli logici semplificando la simulazione.

Per circuiti complessi il simulatore logico consente di ottenere un risultato attendibile, il simulatore elettrico non è "praticamente" utilizzabile.

Simulazioni oggetto funzionali: hanno l'obiettivo di verificare il risultato funzionale del sistema trascurando ogni effetto timing.
(ritardazione)

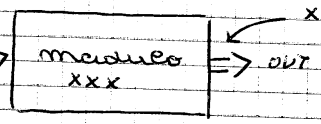


- Simulazione elettrica
- Simulazione logica
- Simulazione logica funzionale ($t_{pLH} = t_{pHL} = \emptyset$)

Aziende di interesse per questo corso:
- Xilinx
- Altera (Acquisita da Intel)

HDL - Verilog
Nasce per fare descrizione e simulazione di un circuito digitale.

Unità base di descrizione Verilog: file che descrive un modulo.



xxx.v:
Parte di dichiarazione
Parte di descrizione

I moduli possono essere istanziati gerarchicamente.

Sintassi del file xxx.v:

```
module nome_modulo (lista di nodi I/O)
input nomi_nodi_I ;
output nomi_nodi_O ;
ulteriori dichiarazioni come nodi interni (...)
[ descrizione ]
endmodule
```

// commento
/* commento */

Il linguaggio è Key sensitive

ABC diverso da abc

Che valori può assumere una variabile?

0, 1, z, x

↑ ↑

vera impedenza — insegnato

Variabili di tipo:

- Wire (nodi interni)
- reg (unica variabile mantenuta all'interno dei costrutti procedurali)
- integer

```
reg a, b, c ; // variabili interne al modulo
wire sum, prod ;

integer i, j ; // variabile a 32 bit
```

27/09/2017

```
input porta_1 ;
output porta_0 ;
inout porta_1_0 ;
```

Variabili Vettoriali:

```
reg [7 : 0] a ; // a è un vettore a 8 bit
    [MSB : LSB]
```

a[i] consente di memorizzare i-esimo componente.

```
reg [0 : 7] b ; // b è un vettore con organizzazione
                  di bit in verso opposto
reg [15 : 0] word ;
word [15 : 8] -> sono gli 8 bit più significativi
word [7 : 0] -> " " " " meno "
```

Operatore di concatenazione:

```
reg a, b, c, d ;
{a, b, c, d} è un vettore a 4 bit
 ↑        ↑
MSB      LSB
```

Rappresentazione dei numeri:

n'lxxx

n: # di bit su cui viene rappresentato il numero

l: base

xxx: cifra del numero

4'b1004 — 4bit : 9

16'hF01A

4'd9

b: bit
h: hex
d: decimal
o: ottale

Keyword per definire costanti:

parameter msb = 15;

reg [msb : 0] out;

Operatori Booleani:

& and
| or
~ not
^ xor

Nel caso di vettori, l'operazione viene fatta bit a bit

[0101] & [1100] = [0100]

Operatori aritmetici:

+ Somma
- Sottrazione
* moltiplicazione
/ divisione

-> bisogna comunque ricordare che ricadiamo sempre a livello architetturale!

(DESCRIZIONE ≠ PROGRAMMAZIONE)

N.B: divisione per due è solo uno shift verso DX.

I numeri in automatico sono:

UNSIGNED (senza segno solo positivi)

Per dichiarare numeri SIGNED:

reg signed [7:0] a; // valido la regola del complemento a 2

8'bs 1111111 vale -1 s: signed

8'b 11111111 vale 255

$A \{a_{m-1}, a_{m-2}, \dots, a_1, a_0\}$

$A_{UNS} = a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \dots + a_1 2^1 + a_0 2^0$

$A_{SIGN} = -a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \dots + a_1 2^1 + a_0 2^0$

$|A| = -A = a_{m-1}2^{m-1} - a_{m-2}2^{m-2} \dots - 2a_1 - a_0 =$

$= (2^{m-2} + 2^{m-3} + \dots + 2^1 + 1 + 1) - a_{m-2}2^{m-2} \dots - 2a_1 - a_0 =$

$= (1 - a_{m-2})2^{m-2} + (1 - a_{m-3})2^{m-3} + \dots + (1 - a_1)2 + (1 - a_0) + 1$

$1 - a_i = \bar{a_i}$

$|A| = \bar{a}_{m-2}2^{m-2} + \bar{a}_{m-3}2^{m-3} + \dots + \bar{a}_1 2 + \bar{a}_0 + 1$

$= \bar{A} + 1$

Estensione di segno:

Unsigned: estendo $\emptyset$

Signed: estendo MSB

29/09/2017

Operatori logici

&& AND

|| OR

! NOT

==

>=

<=

>

<

!=

1'b0 indica il falso

2'b00 indica il falso

⋮

1'b1 indica il vero

e qualunque altro numero diverso da $\emptyset$ indica il vero.

In una espressione multibit operatori logici non lavorano bit per bit come gli operatori bitwise.

4'b0100 && 4'b1011 → 4'b0000 FALSO

4'b0100 && 4'b1011 → VERO

~ e ! applicati su singolo bit è lo stesso ma su multibit no:

~3'b010 -> 3'b101 ~1'b0 -> 1'b1

!3'b010 -> FALSO !1'b0 -> VERO

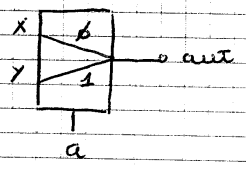
Operatore Condizionale ?

È l'equivalente di if in C.

(espressione) ? (se_vero) : (se_falso)

(a == 1) ? (x) : (y)

È chiaramente un MUX



Descrizione Logica:

Le istruzioni sono valutate in maniera concorrente e non sequenziale.

Tipologie di istruzioni all'interno della descrizione:

1) Chiamata elementi di libreria → and, or, not, buf, nand, nor, xor, xnor
and (out, in1, in2, ...)
2) Assegnazioni continue
3) Blocchi always: blocchi che descrivono il comportamento di una rete in cui è possibile assegnare valori ai reg e che lavorano in regime sequenziale.

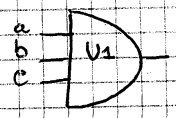
Tipologie di descrizione:

1) Descrizione Architetturale: si istanziano istanze di blocchi esistenti già provvisti e si collegano le porte.

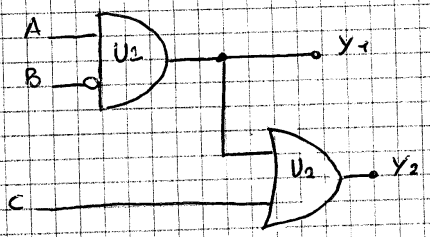
Si instanzia dalla libreria cui metodo:

tipo_elemento nome_istanza (lista porte I/O)

and pippo (out, in1, in2, ...)



and u1 (out, a, b, c);




```
module prova1 (a, b, c, y1, y2)
input a, b, c;
output y1, y2;
and u1 (y1, a, ~b);
or u2 (y2, y1, c);
end module
```

Y sono uscite e sono dichiarate automaticamente come wire.

Se y_1 non è in uscita e viene eliminato come modo di out:

```
module prova2 (a, b, c, y2)
input a, b, c;
output y2;
wire int_node;
and u1 (int_node, a, ~b);
or u2 (y2, int_node, c);
end module
```

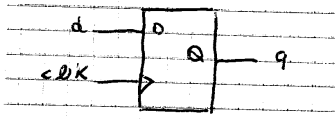
Richiamare un modulo nel momento in cui non rispetto l'ordine degli segnali da richiamare:

```
tipo del nome_istanza (.in(nome_nodo), .out(nome_nodo),
                       ..., .clk(clock));
```

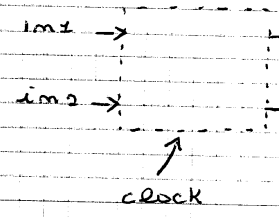
es.

```
myreg reg1 (.clk(clock), .d(in1), .q(out1));
myreg reg2 (.clk(clock), .d(in2), .q(out2));
```

Così ho specificato la parte di libreria.



my_reg. v



2) Descrizione comportamentale

Assegnazione continua:

assign nome_modo = espressione;

assign y = a & b;

assign y1 = a ^ b

assign lavora solo su wire non su reg!

lavora sempre in concorrenza

Blocchi Always:

Insieme di istruzioni procedurali eseguite sequenzialmente

Sono utili per descrivere reti combinatorie e sequenziali:

always @ (lista sensibilità) istruzione ;

↑
input
output
wire
reg
integer

always @ (lista_sensibilità)
begin
... istruzioni sequenziali
end

reg è una variabile assegnata all'interno di always

Se lista_sensibilità = * conviene di eliminare la lista:

always @ (a, b) y = a + b ;
always @ (*) y = a + b ;

All'interno di always si usa reg ad integer

All'interno dei blocchi always gli assegnamenti non hanno ritardo a cui federe con assegnazioni continue.

Tipi di assegnazioni in blocchi always:

- = blocking assignment : si vuole che la variabile assegnata assuma immediatamente il valore: y = a+b;
- <= non blocking assignment : y <= a+b; l'esecuzione di questa istruzione non blocca la sequenzialità di valuta y = a+b viene assunto al termine del blocco always.

```
a = 0;                    a = 0;
b = 0;                    b = 0;
y = a+b                   y <= a+b;
// qui y = 0              // qui y ≠ 0 ha il valore precedente
                          a = 1;
```

→ alla fine y = 0. per via che al tempo era a = 0 e b = 0.

<= viene nel progetto di macchine sequenziali. Ci troviamo funzioni di tipo sequenziali.

Lo stato è calcolato coi vecchi valori e pronto per essere utilizzato al minimo colpo di clock, ma cambia di valore con lo stato precedente.

= lo uso per blocchi combinatori.

N.B: Blocchi always sequenziali: ma più blocchi always lavorano in maniera concorrente.

Attenzione alla descrizione comportamentale:
è una descrizione non una programmazione.

= logica combinatoria
<= logica sequenziale
non mescolare le due cose all'interno dello stesso
blocco always
non utilizzare la sintassi vista in due blocchi
always diversi

Fonte di criticità:
Caso in cui una variabile non viene assegnata
durante l'esecuzione di un blocco always
Supponiamo un assegnamento condizionato:
nel caso in cui si avverte una condizione la
variabile viene assegnata mentre se avviene
un'altra condizione la variabile non viene
assegnata.

Flessofia: la variabile viene assegnata →
gli do un valore

la variabile non viene assegnata →
non c'è stai comparata e quindi non
la vedrà cambiare → non la
vedrà cambiare e quindi la deve
tenere memorizzata → la deve tenere
memorizzata e quindi inserisco un
latch nella descrizione.

02/10/2017

Rete combinatoria che restituisce vero se il vettore è composto da tutti 1 o tutti 0:

```
module beee4 (in, y)
  input [3:0] in;
  output y;
  reg y;
  always @ (in)
    y = (in[3] & in[2] & in[1] & in[0]) | (~in[3] & ~in[2] &
        ~in[1] & ~in[0]);
end module

module beee4 (in, y)
  input [3:0] in;
  output y;
  reg y, wee_0, wee_1;   ← uscita di y, wee_0 o
  always @ (in)              wee_1 dentro beee4
  begin
    wee_0 = ~in[3] & ~in[2] & ~in[1] & ~in[0];
    wee_1 = in[3] & in[2] & in[1] & in[0];
    y = wee_0 | wee_1;
  end
end module

module beee4 (in, y)
  input [3:0] in;
  output y; reg y;   ← y deve essere output perché
  always @ (in)          deve uscire come out.
  begin: myblock
    reg wee_0, wee_1;   ← uscita di y, wee_0 e
    wee_0 = ...;              wee_1 solo dentro
    wee_1 = ...;              myblock
    y = wee_0 | wee_1;
  end
end module
```

Costrutti del Pascal :

if)

- if (condizione) istruzione ;
- if (condizione)
  begin
    // istruzioni
  end
- if (condizione) istruzione
  else istruzione
- if (condizione)
  begin
    // istruzioni
  end
  else
  begin
    // istruzioni
  end

N.B: Tutte le variabili in gioco devono essere assegnate in tutte le possibili condizioni.

Case)

- case (espressione)
    lista valori : istruzione ;
    lista valori : istruzione ;
    lista valori :
    begin
      istruzioni ;
    end
    default : istruzione ;
  end case

multiplexer ad 8 bit con 4 ingressi

```
module mymux ( a, b, c, d, y, see)
  input [7:0] a, b, c, d;
  output [7:0] y;
  input [1:0] see;
  reg [7:0] y;
  always @ ( a, b, c, d, see)
    case (see)
      2'b00 : y = a;
      2'b01 : y = b;
      2'b10 : y = c;
      2'b11 : y = d;
      default : y = 8'bx;
    end case;
  end module
```

N.B. Bisogna segnalare per evitare latch inferiti è assegnato tutte le variabile all'inizio del blocco always.

for)
Servono per ripetere più volte un pezzo di descrizione.

for (loop_index = 1st value, exit_condition of loop,
      next_loop_index)

```
for (i=3, i<255, i=i+2)
  istruzione i;
```

i variabile dichiarata di tipo integer.

altri costrutti : while

Come descrivere reti sequenziali :
Queste reti valutano gli ingressi in corrispondenza di un fronte di clock.
La rete si occupa a tale punto di generare nuovi stati e nuove uscite.

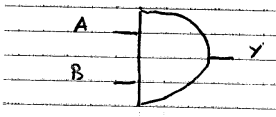
descrivere una rete sequenziale:

always @ (posedge clock)
negedge clock

03/10/2017

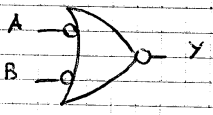
GATE ELEMENTARI

Rappresentano le funzioni logiche elementari



$Y = A \cdot B \equiv \overline{\overline{A} + \overline{B}}$

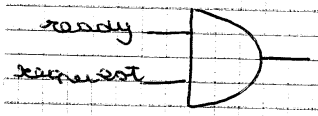
↑ Teorema di De Morgan

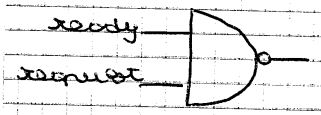


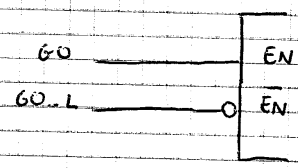
Usare nomi di segnali significativi.

Livello di attività di un segnale:
→ attivo al livello alto
→ attivo al livello basso

A livello schematico, parola importante invece a distinguere:







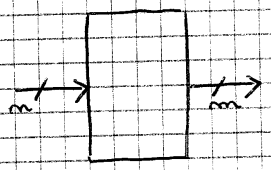
$GO\_L \neq GO$

$GO\_L = \overline{GO}$

Elementi primitivi di logica combinatoria che si possono trovare all'interno di librerie TTL:

Decoder (Decodificatore)

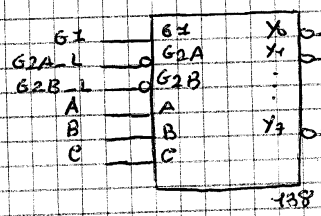
Rete combinatoria che traduce un codice di n bit in un codice a m bit, con $m > n$.



Nel decodificatore binario $m = 2^n$.
In esso viene attivata una ed una sola linea di output pesata dalle stringhe di input.

$n = 5$ → attivo output 5.

Decoder TTL 74Y38 : Dec 3 → 8 con attivo basso con 3 enable

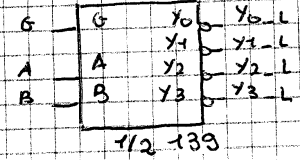


Quando anche una tra G1, G2A_L, G2B_L è disattivata le uscite sono non attive.

$Y_0 = Y_1 = \dots = Y_7 = 1$

→ Ho disattivato l'esecuzione della funzione decoder → non sto selezionando alcuna uscita

Decoder TTL 74Y139 : Dec 2 → 4 (dual) con 1 enable con attivo basso.

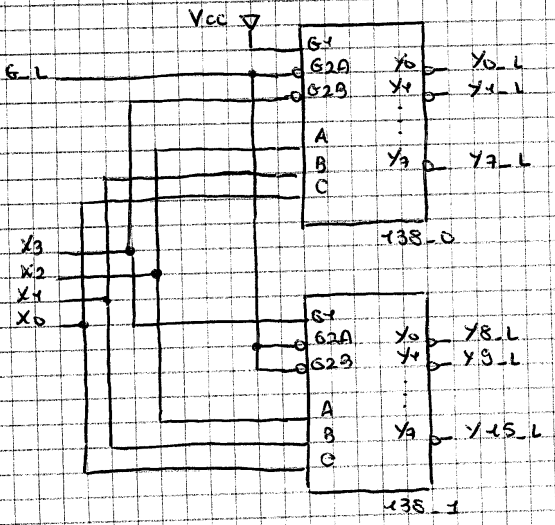


```
module 74x138 ( Y_L, G1, G2A_L, G2B_L, A, B, C )
  input G1, G2A_L, G2B_L, A, B, C ;
  output [7 : 0] Y_L ;
  wire [2 : 0] IN ;
  assign IN = {A, B, C} ;
  reg [7: 0] Y_L ;
  always @ ( G1, G2A_L, G2B_L, IN )
  begin
    if ( G1 & ~G2A_L & ~G2B_L )
      case (IN)
        3'b000 : Y_L = 8'b11111110 ;
        3'b001 : Y_L = 8'b11111101 ;
                 .
                 .
        3'b111 : Y_L = 8'b01111111 ;
        default: Y_L = 8'b11111111 ;
      endcase
    else
      Y_L = 8'b11111111 ;
  end
endmodule.

// Realizzazione con un costrutto iterativo for
.
.
  reg [7: 0] Y_L ;
  integer J ;
  always @ (*)                    ← manca contesto sugli ingressi.
  begin
    Y_L = 8'b11111111
    for ( J=0, J<=7 ; J=J+1)
      if ( J == IN)
        Y_L [J] = 1'b0 ;
  end
endmodule
```

Supponiamo di voler gestire un 4 -> 16

Utilizzo due 74138 3 -> 8



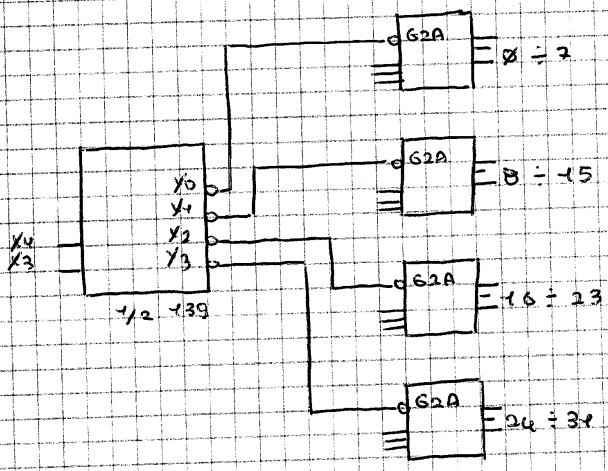
$X_3 = H$ -> 138_0 OFF, 138_1 ON

$X_3 = L$ -> 138_0 ON, 138_1 OFF

G_L -> enable generale

G_L = L -> 138_0, 138_1 attivi

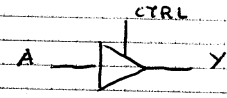
Decoder 5 -> 32

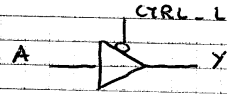


Buffer

Serve a migliorare le caratteristiche elettriche di un sistema → rigenerano le uscite.
Utilizzati per pilotare grossi carichi capacitivi.

Si usano Buffer Tristate

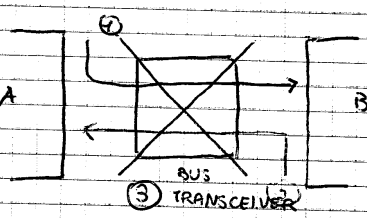





```
module 3state_buf (Y, A, CTRL)
    input A, CTRL;
    output Y;
    assign Y = (CTRL) ? A : 1'bz;
endmodule
```

L'uscita di buffer 3state è pericoloso: se usa più buffer per connettere la stessa linea che non siano mai attivi due o più insieme. Inoltre se uno non può trasmettere (tutti i buffer off) perché può assumere valore imprevedibile.

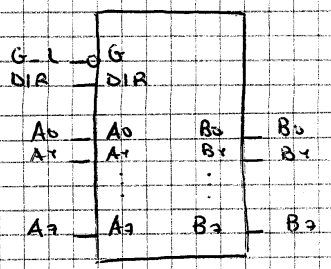
Comunicazione tra due BUS:



Bus transceiver 74245

TTL 74245

Controller Bus at 8 bit



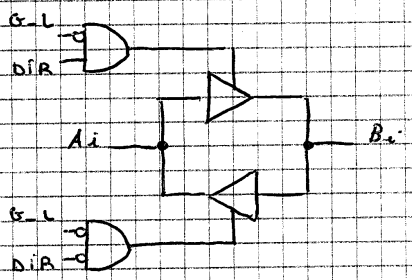
A_i, B_i sono pin bidirezionali.

G_L = H Bus separati → A e B → Z

G_L = L : a seconda del DIR A = B o B = A

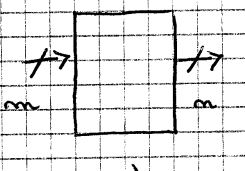
(DIR = 0) (DIR = 1)

B → A A → B




```
module 74245 ( A, B, DIR, G_L)
  input DIR, G_L ;
  inout [7:0] A, B ;
  assign A = (~G_L & ~DIR) ? B : 8'bz ;
  assign B = (~G_L & DIR) ? A : 8'bz ;
endmodule
```

Codificatore (Encoder)



$m > m$

Codificatore binario: $m = 2^m$

A seconda della linea selezionata in input l'uscita fornisce un codice.

In input ho 2^m possibili combinazioni.

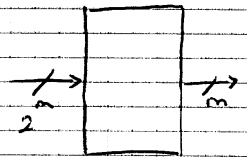
Ma queste linee sono ovviamente 2^n
(1 uscita alta e tutte le altre basse)

Combinazioni vietate: $2^{2^n} - 2^n$

Priority Encoder
Sono codificatori con priorità

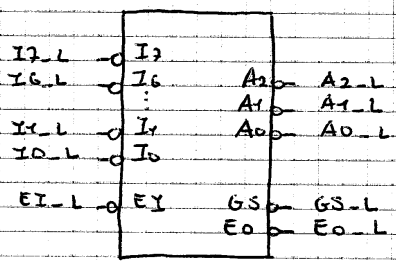
04/10/2017

Codificatore binario



abbiamo $(2^{2^n} - 2^n)$ input inutili.
soluzione: utilizzare priorità.

Priority Encoder 74148 (TTL)

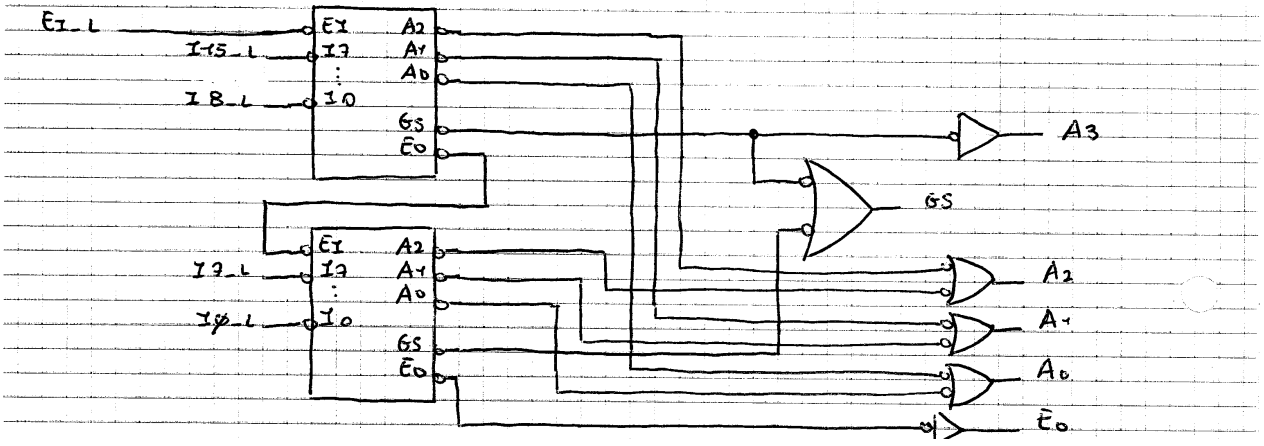


Priorità:
$P_{I7} > P_{I6} > \dots > P_{I1} > P_{I0}$

Se GS_L è vero significa che c'è almeno linea attivata.

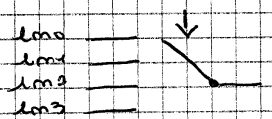
E_O: Enable out
E_I: Enable in
GS = E_I & almeno un I.
EO = E_I & nessun I.

Encoder a 16 bit



multiplexer (mux)

Converte una fra tante linee in ingressi in output



n ingressi
1 uscita
a segnali di selezione
→ $a = \lceil \log_2 n \rceil$

mux meccanico: bidirezionale

mux elettronico: unidirezionale

TTL 74-151

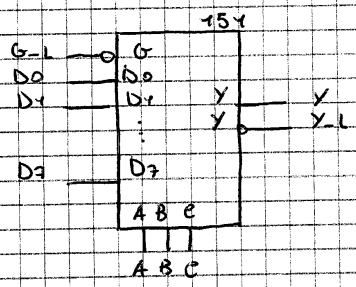
Mux ad 8 ingressi ed 1 linea con Y e $\bar{Y}$ ed EN

TTL 74-153

mux 4 a 1 duale

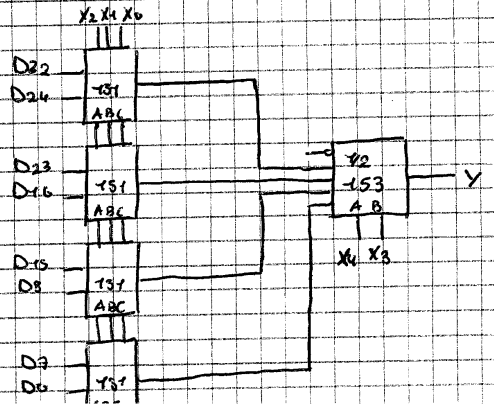
TTL 74-157

mux 2 a 1 quad

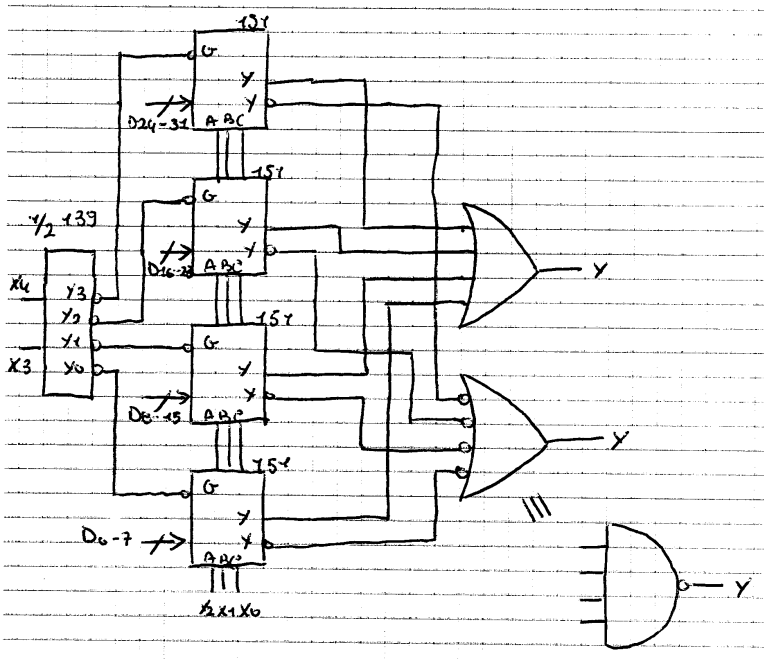


Costruzione di un 32 a 1

$a = 5$



Costruzione di un MUX 32 → 1, metodo alternativo: 09/10/2017



Demultiplexer

→ non ci sono demux nella logica TTL.

I demux sono considerati decoder.

TTL74139

G_L	A	B	Y0_L	Y1_L	Y2_L	Y3_L
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Analizziamo la configurazione Ø Ø

A = Ø , B = Ø

G_L = 1 → Y Ø_L = 1

G_L = Ø → Y Ø_L = Ø

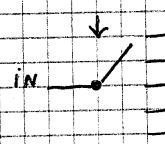
A = Ø, B = 1

G_L = 1 → Y1_L = 1

G_L = Ø → Y1_L = Ø

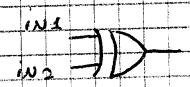
ecc...

Funzione del Demux:



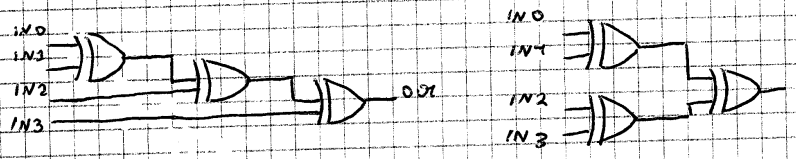
Il segnale d'ingresso del demux dell'esercizio di un decoder, l'uscita è esattamente la demultiplazione di questo ingresso. Tale ingresso smistando dell'esercizio del decoder lo ritrovo esattamente come uscita sulla linea desiderata. Decoder e demux hanno la stessa funzione logica.
Le stesse tecniche di espansione di un decoder sono tecniche di espansione di un demux.

Porta Logica XOR



IN1	IN2	OUT = IN1 ⊕ IN2
0	0	0
0	1	1
1	0	1
1	1	0

Usata in maniera ricorsiva consente di valutare il numero di 1 in una parola.
Può valutare se il numero di 1 è pari o dispari.
Una cascata di XOR esegue la stessa funzione di un albero di XOR.



Varia esclusivamente la PROPAGAZIONE.

Il bit di parità di un dato è un bit di ridondanza aggiunto durante la trasmissione di una parola. Consente di riconoscere la validità o meno di una parola.

ODD PARITY
numero di 1 dispari nella parola.

0 1 1 0 0 1
ODD

EVEN PARITY
Numero di 1 pari nella parola

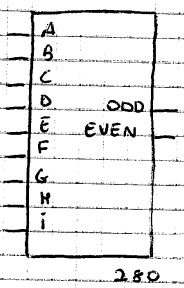
0 1 1 0 0 0
EVEN

Fornisce un primo e semplice metodo per rilevare la presenza o meno di errore.

In termini di banda non richiede aumento di performance eccessive.

Le strutture a XOR sono perfette per ricevere questo compito.

TTL74280 Parity CRecker / Gemerator



Se lo uso come generatore: IN [7÷0] = A÷H
I = 0

Trasmetto IN [7÷0] + EVEN oppure IN [7÷0] + ODD

Se lo uso come checker

IN [8 : 0] = A : 1

ODD ed EVEN mi forniscono parità a 8 bit.

Costruisco in Verilog:

```
module 74280 ( ODD, EVEN, IN )
  input [8 : 1] IN;
  output ODD, EVEN;
  reg ODD, EVEN, P;
  integer J;
  always @ ( IN )
  begin
    P = 1'b0;
    for ( J = 1 , J <= 8 , J = J + 1 )
      if( IN [J] ) : P = ~P;
        else  P = P;
    ODD = ~P;
    EVEN = P;
  end
endmodule
```

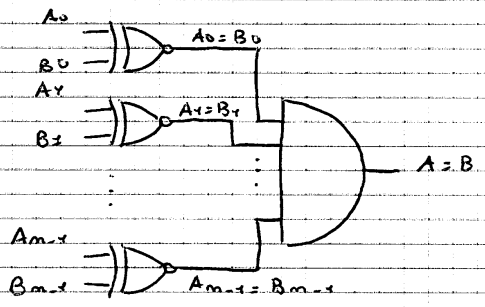
La porta XOR mi consente di capire se due bit sono uguali tra loro.

→ Funziona comparatore.

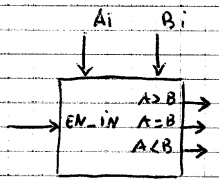
Comparatori:

- uguaglianza;
- maggioranza o minoranza;

Comparatore di uguaglianza



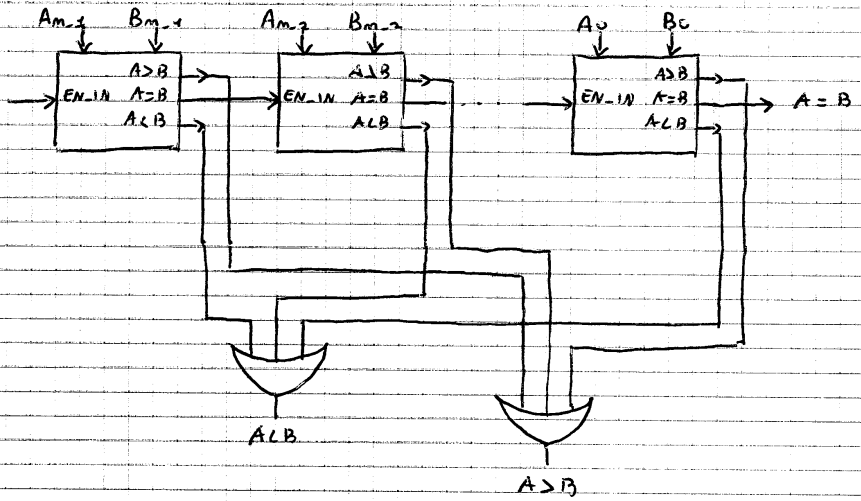
Comparatore di maggioranza / minoranza



Blocco Base

EN_IN	Ai	Bi	A>B	A=B	A<B
0	X	X	0	0	0
1	1	0	1	0	0
1	0	1	0	0	1
1	0	0	0	1	0
1	1	1	0	1	0

Per un confronto tra due parole ad n bit



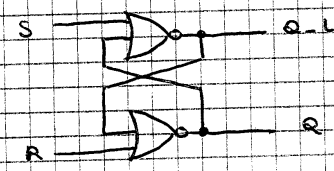
Un altro modo è effettuare A - B e controllare lo segno del risultato (valore MSB).

Attenzione all'operazione +, *, ÷ che assume un significato un poco diverso a seconda della logica utilizzata.

Elementi Basici di Logica Sequenziale:

- FLIP FLOP: elemento sensibile agli ingressi in un intervallo prossimo al fronte di un segnale di controllo. Le uscite variano solo in prossimità del fronte del segnale di controllo.
- LATCH: elemento sensibile agli ingressi in prossimità di un livello di un segnale di controllo. Le uscite variano solo durante il livello sensibile.

LATCH SR

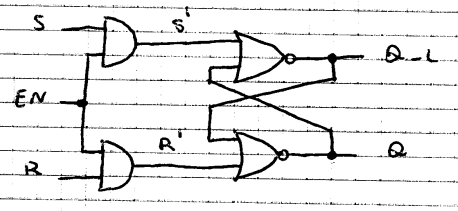


S	R	Q	Q_L	
0	0	Q	Q_L	con $Q = \overline{Q_L}$
0	1	0	1	con $Q = \overline{Q_L}$
1	0	1	0	con $Q = \overline{Q_L}$
1	1	0	0	con $Q = Q_L$

Se da S=1, R=1 passiamo a S=0, R=0 non sappiamo che valore assumerà Q e Q_L!

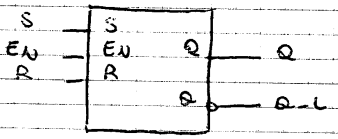
LATCH SR CON ENABLE

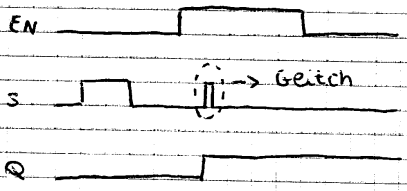
10/10/2017



$S' = S \cdot EN$

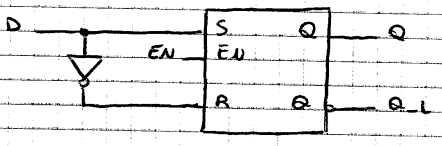
$R' = R \cdot EN$





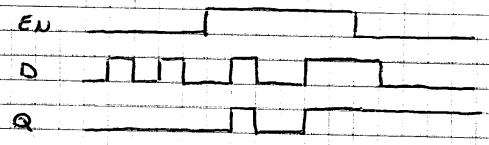
Glitch: commutazione spuria

LATCH D CON ENABLE



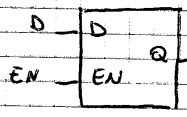
Consente di memorizzare il valore D nel momento in cui EN viene disabilitato.

→ L'istante di memorizzazione è l'istante di disabilitazione.



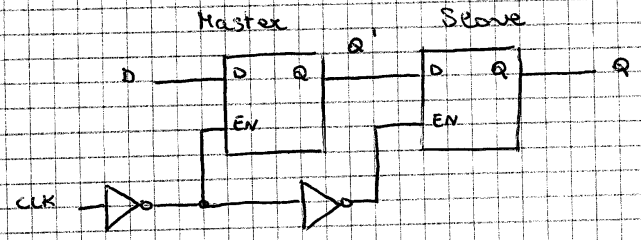
N.B: I latch quindi sono trasparenti: l'ingresso compare in uscita.

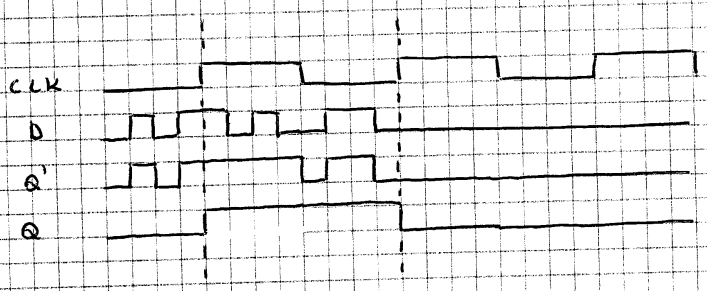
Simbolo equivalente



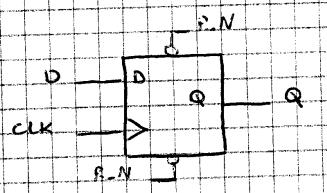
È il blocco base con cui costruire un FLIP FLOP

FLIP FLOP POSITIVE EDGE TRIGGERED DI TIPO D





È un oggetto analogo a D solo che prende ↑ del CLK. Ha uscita solo solo che prende ↑ del CLK.



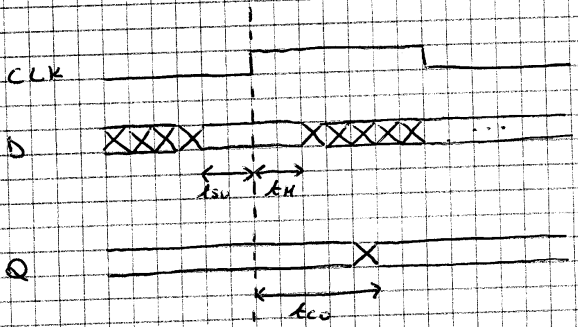
P.N : Resetta immediatamente su D (Q = 0)

P.N : Setta immediatamente su D (Q = 1)

È conosciuto anche come Flip Flop complementare.

Temporizzazione dei segnali in ingresso:

→ segnale D stabile un po' prima ed un po' dopo il fronte di salita del CLOCK.



t_{SU} : $t_{SET\ UP}$

t_H : t_{HOLD}

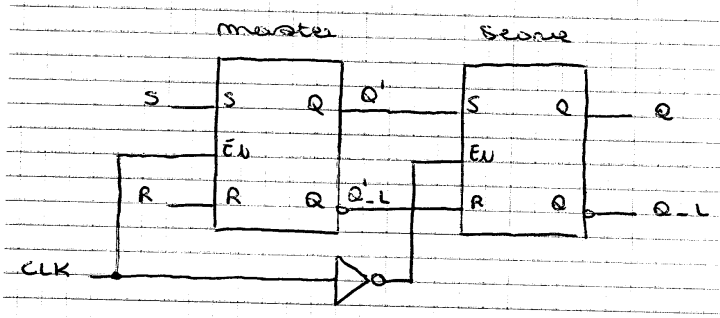
Reset e Preset:

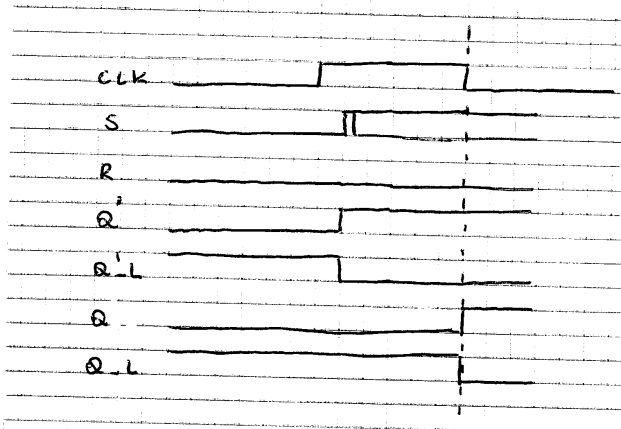
- Asincrono: effetto avviene immediatamente.
- Sincrono: effetto avviene alla presenza del prossimo fronte sensibile.

Verilog:

```
module my_dff (Q, D, CLK)
    input CLK, D;
    output Q;
    reg Q;
    always @ (posedge CLK)
        Q <= D;
endmodule
```

FLIP FLOP SR POSITIVE LEVEL SENSE

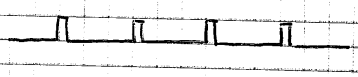




Da reset il FlipFlop non è edge triggered ma level sense.

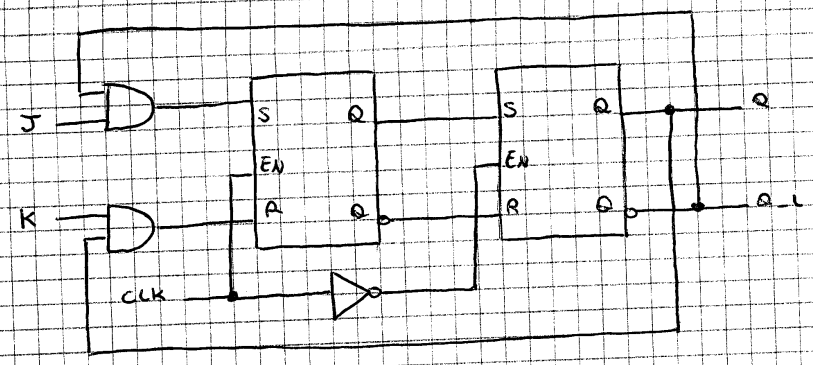
È sensibile a tutto il livello alto di CLK.

È necessario un CLK a impulsi



FLIP FLOP JK POSITIVE LEVEL SENSE

È un Flip Flop SR che risolve problemi nel caso di $S = 1$, $R = 1$



$S = J \cdot Q_L$

$R = K \cdot Q$

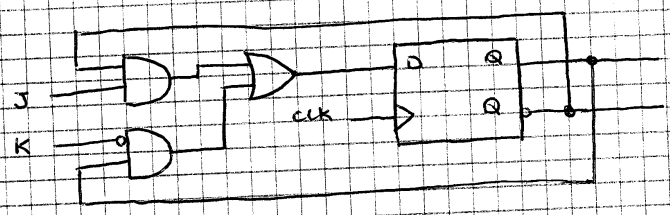
J	K	S	R	Q
0	0	0	0	Q
0	1	0	Q	0
1	0	Q_L	0	1
1	1	Q_L	Q	$\bar{Q}$

N.B: Questo è ancora un dispositivo sensibile al livello, non al fronte

→ Il dispositivo è sensibile al fronte del livello alto di CLK.

FLIP FLOP JK POSITIVE EDGE TRIGGERED

Uso dei Flip Flop D.



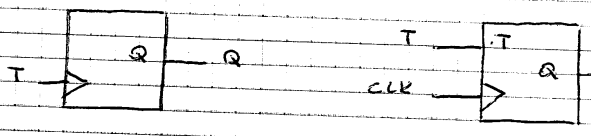
$D = J \cdot Q_L + \bar{K} \cdot Q$

	J	K	D	Q
	0	0	Q	Q
Reset:	0	1	0	0
Set:	1	0	$Q_L + Q = 1$	1
	1	1	Q_L	Q_L

Questo è un dispositivo sensibile al fronte ↑ di CLK.

FLIP FLOP TOGGLE (FLIP FLOP T)

Ha la funzione di cambiare di stato in presenza del segnale di toggle.



CASO 1

CASO 2

T = 1 Q <= ~Q

T = ϕ Q <= Q

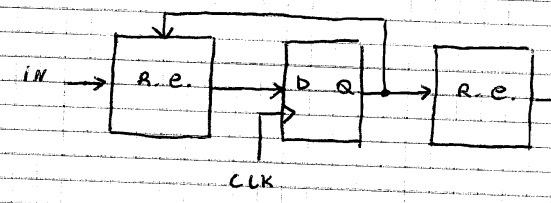
Chiamato anche FLIP FLOP contatore: è il blocco base di un contatore.

Una macchina sequenziale serve a costruire una rete sequenziale.

Utilizzare una macchina sequenziale per mantenere memorizzati gli stati di macchina a stati finiti (FSM)

FSM (FINITE STATE MACHINE)

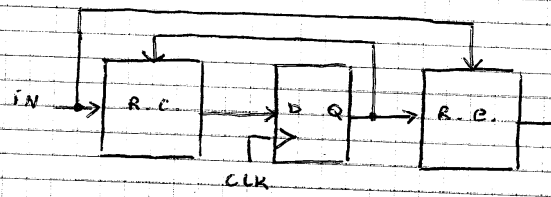
Sono macchine che generano uscite in funzione di ingressi e stati



modello di Moore

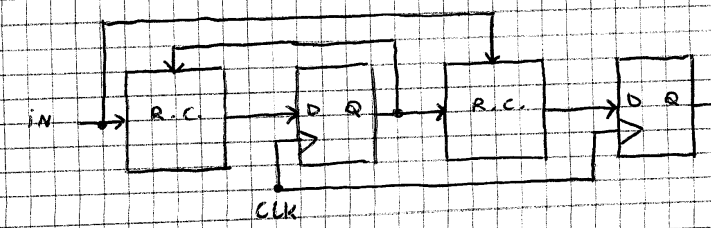
Uscite dipendono solo dagli stati correnti

R.C.: Rete Combinatoria



modello di Mealy

Qui le uscite dipendono da stati ed ingressi correnti. C'è un percorso diretto tra IN ed OUT.



modello di Mealy

Uscite determinate da stati correnti ed ingressi precedenti.

Per la realizzazione di una FSM è necessario: il numero di stati e la loro codifica, mediante reg. di stato; una rete di uscita (RC) di transizione degli stati e una di transizione delle uscite.

Una macchina a stati finiti che ha n stati, deve necessariamente avere un registro di stato e quindi s flip flop.

m stati

$s = \lceil \log_2 m \rceil$

Se $2^s > n$, ci sono $2^s - n$ stati non definiti
→ di sintesi la rete non considererà questi stati. È la codifica a minimo costo.

Se per un disturbo mi trovo in uno stato indefinito non ho il controllo della macchina
→ è possibile che nessuna combinazione degli ingressi faccia uscire la macchina da questo stato (DEAD LOCK)

Con la sintesi a minimo costo non garantisco la sicurezza.

Uso sintesi a minimo costo, invece cura di ciò che accade negli stati non definiti → è necessario definirli!
Devo prevedere un'uscita sicura degli stati non definiti.

Immaginiamo di usare codifiche ridondanti:

n stati ed n variabili di stato

Codifica one_hot:

m = 4.

		∂_3	∂_2	∂_1	∂_0
S: stati	S_0	0	0	0	1
∂: bit di codifica (var. di stato)	S_1	0	0	1	0
	S_2	0	1	0	0
	S_3	1	0	0	0

Potenziale vantaggio:

Durante la transizione stato -> stato il comportamento è veramente semplice.

es: $S_2 \rightarrow S_1$ $\quad \partial_2 = 1 \rightarrow \emptyset$ $\quad \partial_1 = \emptyset \rightarrow 1$

La logica combinatoria che varia gli stati è molto semplificata.

È però aumentata la logica necessaria per costruire le variabili di stato.

STATO INIZIALE

Solitamente si riporta tutte le variabili di stato a $\emptyset$

Codifica one_hot diventa codifica. Quasi one_hot.

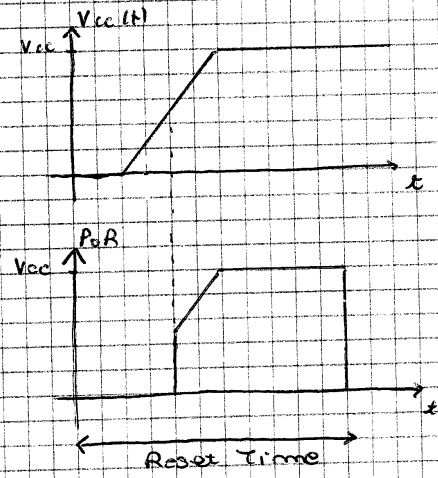
	∂_3	∂_2	∂_1	∂_0
S_0	0	0	0	0
S_1	0	0	0	1
S_2	0	0	1	0
S_3	0	1	0	0
S_4	1	0	0	0

RESET

Consente di riportare il sistema in uno stato noto, all'accensione del sistema il segnale di reset consente di definire lo stato iniziale del sistema logico progettato.

Power-on Reset (P.O.R)

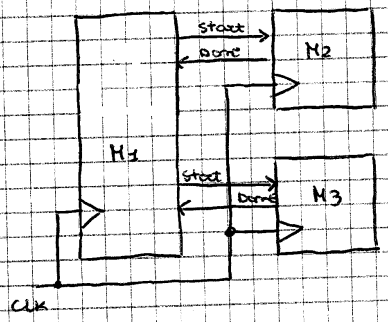
Circuito che quando $V_{cc}(t)\uparrow$ resetta gli elementi di logica sequenziale



PROGETTAZIONE DI UNA FSM

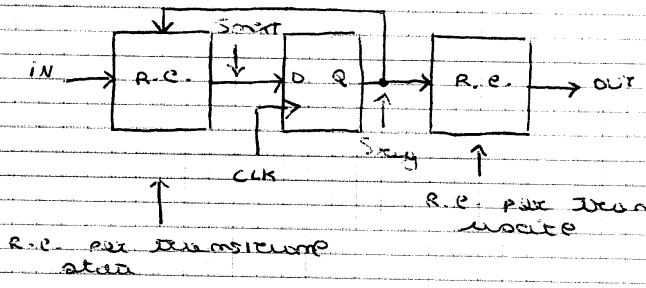
Mealy è di progettazione gerarchica.

L'attivazione di una macchina avviene attraverso un semaforo.



Descriviamo FSM in HDL Verilog 11/10/2017

Facciamo riferimento al modello di Moore.



1) Costruisci della memoria di stato: definisce tra stato futuro e stato attuale.
   1a) codifica degli stati

2) Tabella di transizione di stato: legge tra stato attuale, ingresso e stato futuro

3) Tabella di transizione delle uscite: legge tra stato attuale e uscita.

```
Stato attuale : Sreg
Stato futuro : Smxt

always (posedge clk)
  Sreg <= Smxt
```

} 1

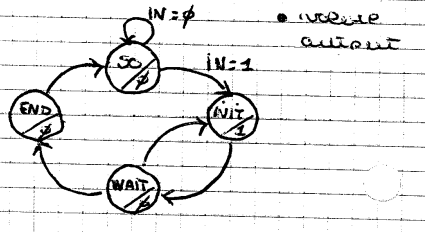
Adesso mi piacerebbe dare un nome agli stati e codificare la variabile di stato.

La codifica degli stati, in HDL, sta a monte di 1, nella parte di dichiarazione.

```
Parametex S0 = 3'b000,
          INIT = 3'b001,
          WAIT = 3'b010,
          END = 3'b100;
```

} 1a

es.



• valore output

S0 = 0 0 0
INIT = 0 0 1
WAIT = 0 1 0

Il blocco che gestisce il cambiamento di stato è un blocco combinatorio

```
always @ ( Sreg , IN )
   case ( Sreg )
      S0:  if( IN == 1) Snext = INIT;
           else Snext = S0;
      INIT: ...
```

} 2

```
always @ (Sreg)
   case (Sreg)
      S0 : out = 1'b0;
      INIT : out = 1'b1;
      WAIT: ...
```

} 3

Per una macchina di Moore:
always @ (Sreg, IN)

L'istruzione reset è necessaria di resettare la FSM:

Reset sincrono stato successivo:

```
always @ ( posedge clk)
   if ( reset ) Sreg <= reset_state;
   else         Sreg <= Snext;
```

P.S. Sreg e Snext definite come reg.

Reset asincrono stato successivo:

```
always @ ( posedge clock, posedge reset )
   if (reset) Sreg <= reset_state;
   else       Sreg <= Snext;
```

REGISTRI 13/10/2017

È un insieme di latch o flip flop sincronizzati con lo stesso segnale di sincronizzazione.

Un registro è un elemento di memoria elementare.

Caratteristiche:
- tipologia registro;
- # bit;
- uscita bassa o alta impedenza.

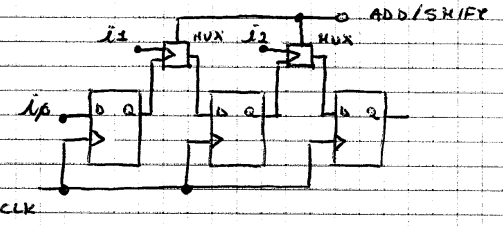
Registro 16 bit con out 3 state (FF) con clear asincrono

```
module 16reg ( Q, IN, CLK, OE_L, CLR_L )
    input CLK, OE_L, CLR_L ;
    input [15:0] IN ;
    output [15:0] Q ;
    reg [15:0] IQ ;
    always @ ( posedge CLK , negedge CLR_L )
        if ( CLR_L == 1'b0 ) IQ <= 16'b0 ;
        else IQ <= IN ;
    assign Q = (OE_L) ? 16'bZ : IQ ;
endmodule
```

Questo appena descritto è un registro PIPO

Tipologie registro:
- PIPO
- SIPO
- SISO (Chiamato anche Shift Register)
- PISO

Struttura classica di un PISO:



CONTATORE

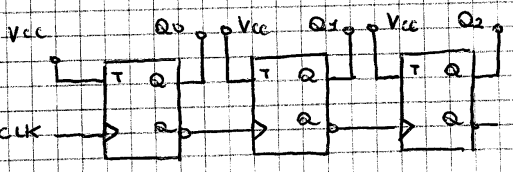
È una macchina a stati che percorre gli stati in maniera ciclica.

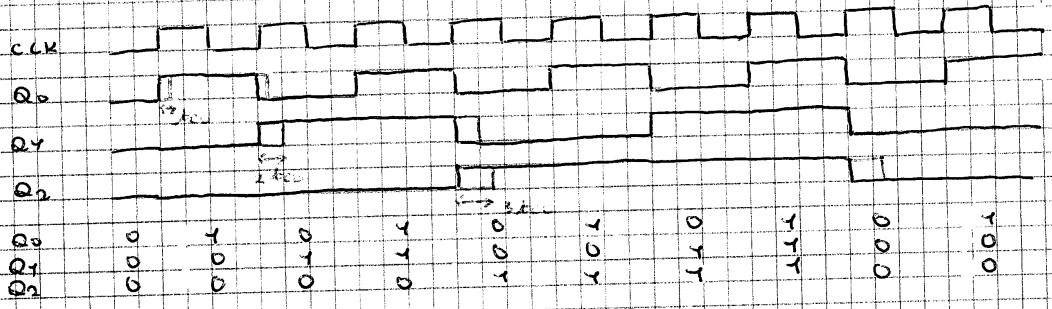
Se il numero di stati è esprimibile come 2^3 e questi stati sono percorsi in maniera binaria, si parla di CONTATORE BINARIO.

Un contatore può essere utilizzato per contare eventi.

Struttura: CONTATORE ASINCRONO

non c'è un unico segnale di clock





Q_0	0	1	0	1	0	1	0	1	0	1
Q_1	0	0	1	1	0	1	1	1	0	0
Q_2	0	0	0	0	1	1	1	1	0	0

C'è una ciclicità nel percorrere gli stati

$f_{Q_0} = \frac{1}{2} f_{CLK}$

$f_{Q_1} = \frac{1}{4} f_{CLK}$

$f_{Q_2} = \frac{1}{8} f_{CLK}$

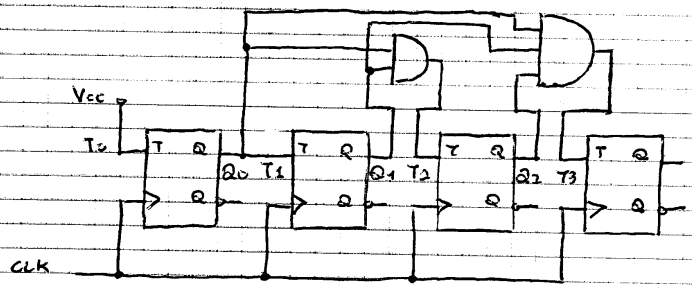
Applicazione dei contatori:

- conteggio eventi
- divisione di frequenza.

Adesso consideriamo le timing che affetto ha due uscite:

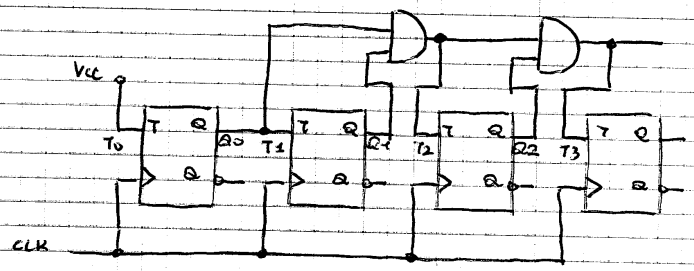
La variazione di frequenza avviene in maniera istantanea, ma il conteggio è ritardato

Per risolvere questo problema si introduce una nuova struttura: CONTATORE SINCRONO CON RIPORTO PARALLELO



$T_i = Q_{i-1} \cdot Q_{i-2} \dots Q_1 \cdot Q_0$

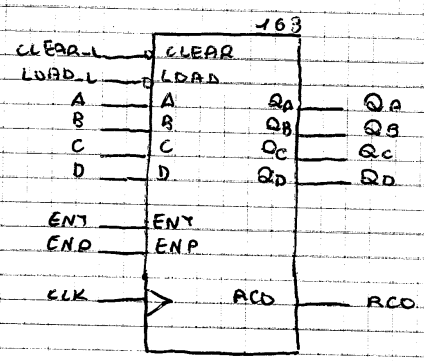
altra soluzione: CONTATORE SINCRONO CON RIPORTO SERIALE



$T_i = Q_{i-1} \cdot T_{i-1}$

TTL 74163

Contatore Sincrono reperto parallelo 2 modi (coincidenza ed asincrono) sincrono . 4 bit



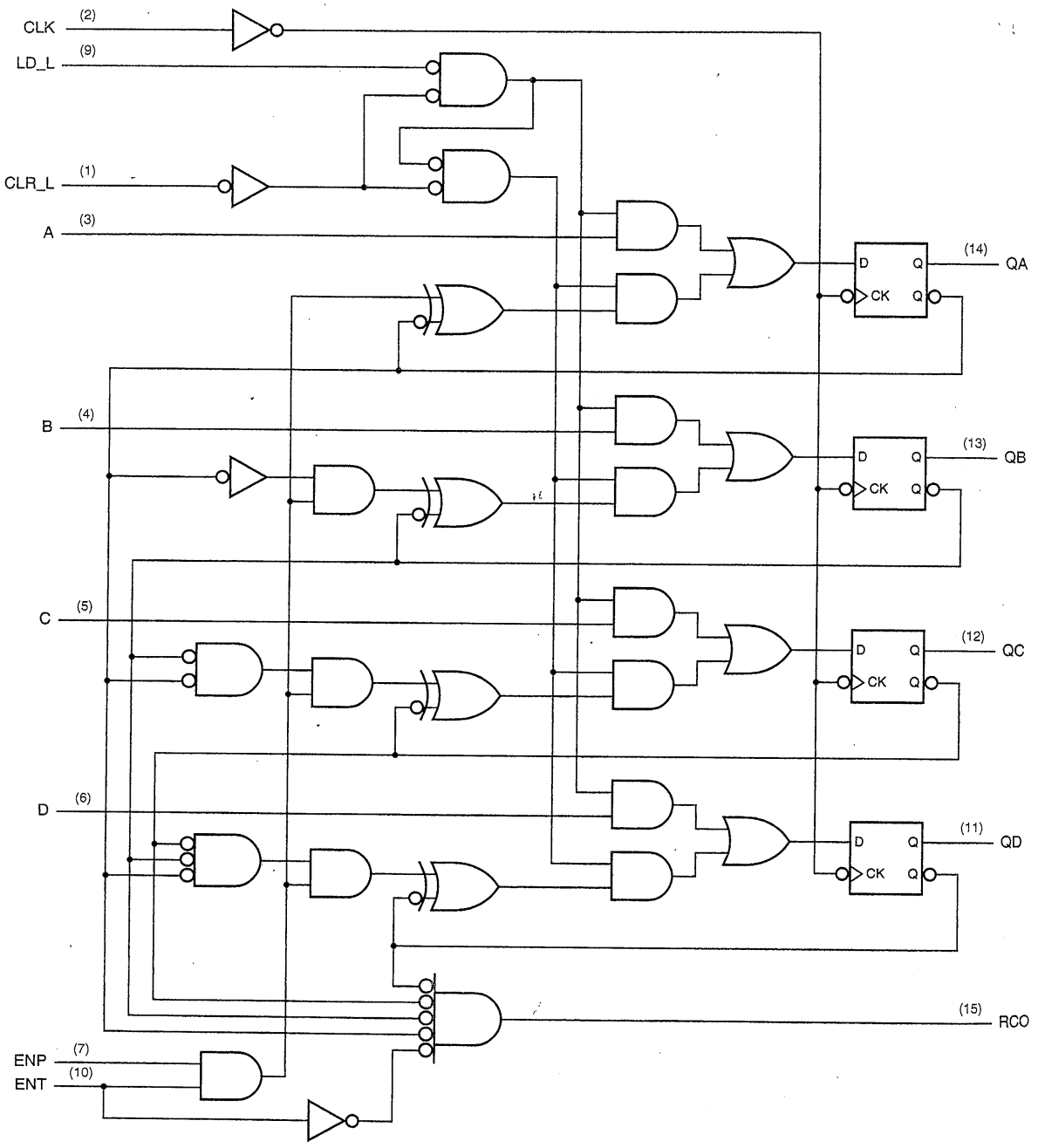
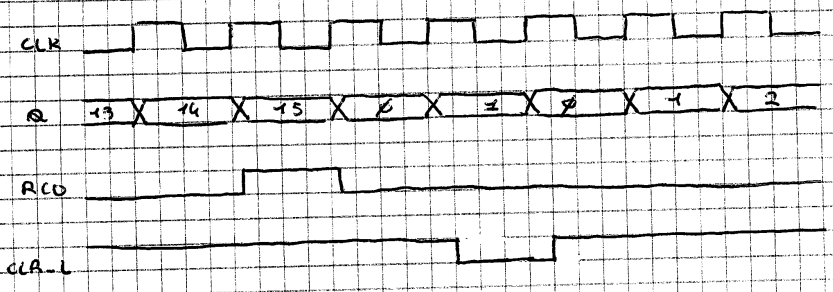
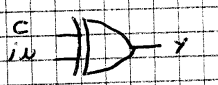


Figure 8-28 Logic diagram for the 74x163 synchronous 4-bit binary counter, including pin numbers for a standard 16-pin DIP package.

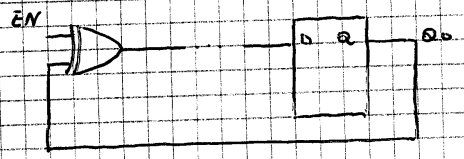


LD_L consente di caricare il contatore con un certo valore determinato da A B C D

Il contatore se non caricato mantiene il suo stato



C	IN	Y	
0	0	0	IN
0	1	1	
1	0	1	$\overline{IN}$
1	1	0	

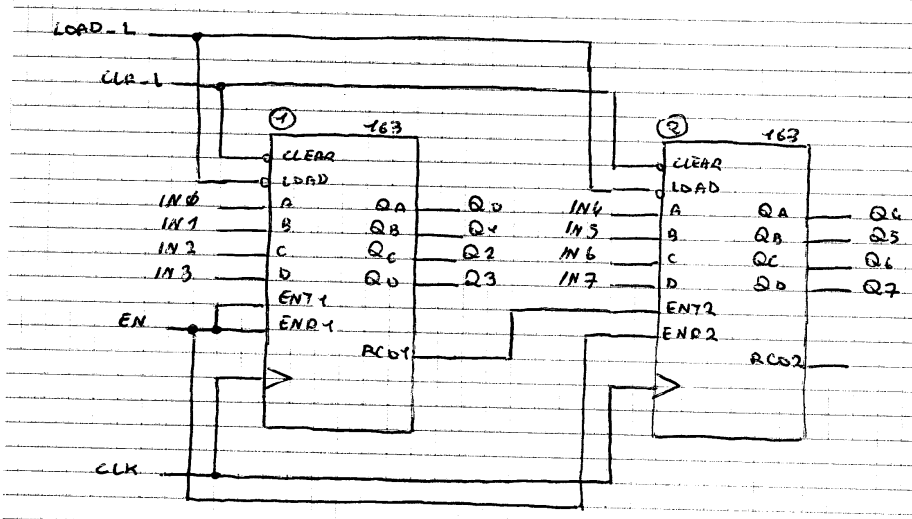


Per il resto degli stati si vede che il riporto è parallelo.

ENP ed ENT entrambi alti per attivare il funzionamento.

RCO: Ripple Carry output

Espansione a 8 bit:



Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Il contatore 2 conta una volta ogni 16 cicli del contatore 1.

ENP è un enable parallelo

RCO1 attiva ENT1 ogni 16 cicli.

$RCO_2 = Q_A \cdot Q_B \cdot Q_C \cdot Q_D \cdot ENT_2$

$= Q_4 \cdot Q_5 \cdot Q_6 \cdot Q_7 \cdot ENT_2$

$= Q_4 \cdot Q_5 \cdot Q_6 \cdot Q_7 \cdot Q_3 \cdot Q_2 \cdot Q_1 \cdot Q_0 \cdot ENT_1$

$(ENT_2 = RCO_1)$

Dal punto di vista VHDL:
la variabile di stato Sreg si traduce in un'uscita Count, da cui risulta:

Count <= Count + 1;

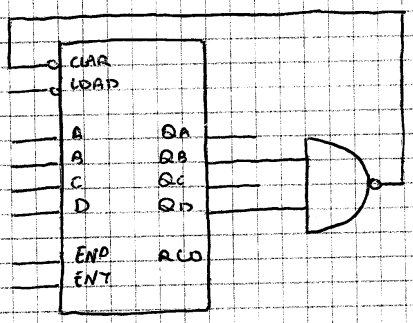
Questo sono strutture di contatori che contano modulo 2^n.

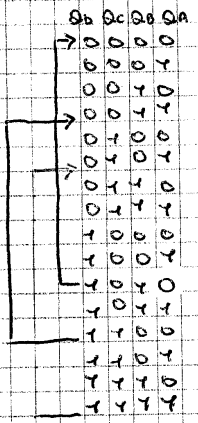
Per avere un conteggio modulo $\neq 2^n$.
Devo riconoscere uno stato di uscita e in base a quello stato attivare la CLEAR del contatore.

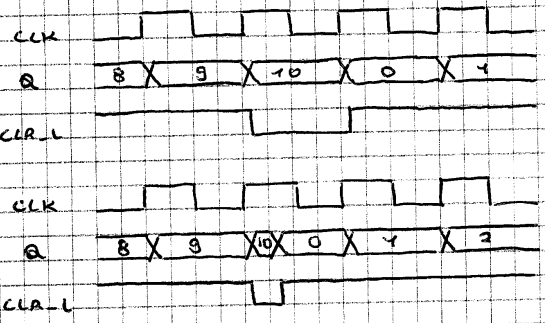
18/10/2019

Esempio contatore mod. 11 con 74x63

1)
Mi è sufficiente riconoscere
lo stato 1x1x al posto di 1010:



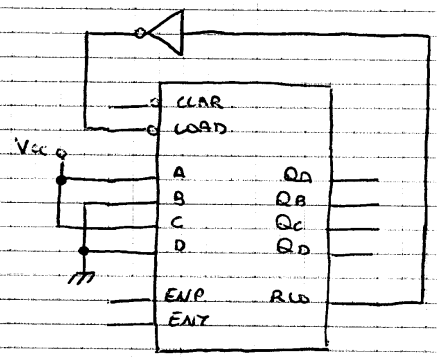




← clear sincrono

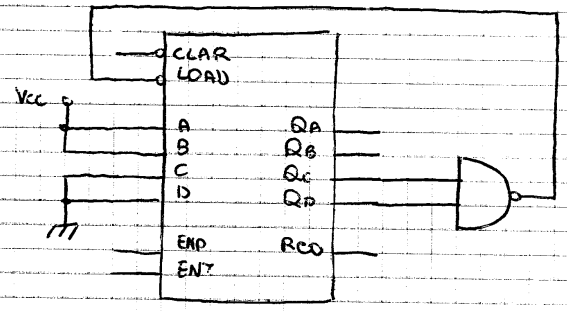
← clear asincrono

2)



3)

Contatore mod. 10 con $\delta = 50\%$ (δ: duty cicee)



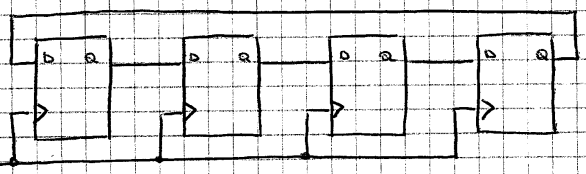
Ricordo 74xx al posto di 74LS00.

In questi casi utilizzerò integrato da 14 pin con codifica a minimo costo

CONTATORI NON BINARI

RING COUNTER

Si parla di contatori ridondanti: numero di bit della reg. di stato superiore al numero di bit minimo necessario.



Configurazione iniziale:

1000

stato successivo:

0100

0010 → n stati: n flip flop.

0001 → Ogni base ad ogni stato un'uscita è alta.

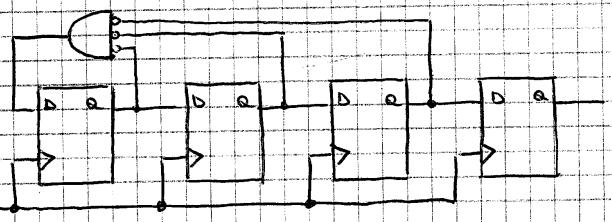
1000

...

Se la config. iniziale è 0000 o 1111 il contatore non conta.

È un contatore semplice ma poco sfruttato.

Tecniche per autocorreggere il comportamento del contatore:

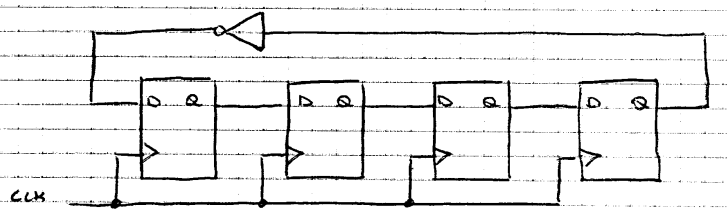


$D_0 = \overline{Q_0} \cdot \overline{Q_1} \cdot \overline{Q_2}$ → riconosce lo stato 000 x

nello stato successivo R_0: 1000

Questa soluzione autocorregge automaticamente il conteggio riportandolo a quello naturale.

JOHNSON COUNTER



Stato iniziale: 0000

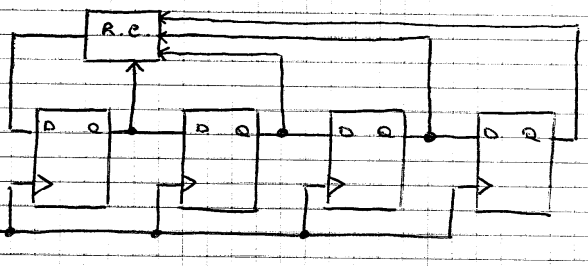
0000
1000
1100
1110
1111
0111
0011
0001

2m stati : m flip flop

Vantaggi:
- doppio numero di stati rispetto a ring counter;
- stato iniziale 0000.

Alcuni contatori, opportunamente riarrangiati riescono ad avere $2^n - 1$ stati con n flip flop.

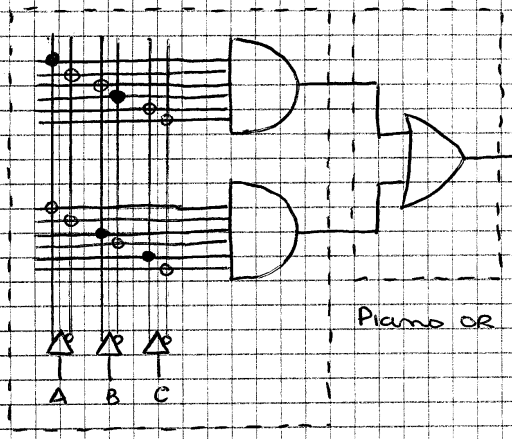
LINEAR FEEDBACK SHIFT REGISTER



Programmazione di Hardware 20/10/2017

Dispositivo digitale con funzionalità definita post fabbricazione.

Realizzare una funzione combinatoria programmabile.



⊕ : collegamento programmabile

Piano OR non programmabile.

Piano AND programmabile.

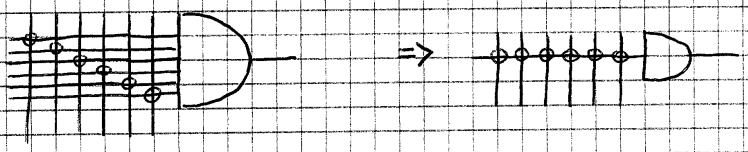
PAL : Programmable array logic

È proprio la tecnica utilizzata nel caso precedente.

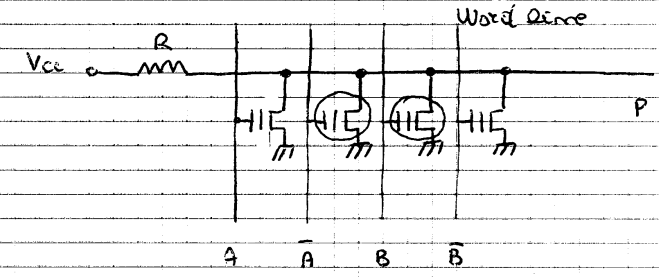
Limite:

- numero di ingressi definito
- numero di uscite esplicite definito
- numero di ingressi della parte con molti collegamenti estremamente elevato.

Simbolo di rappresentazione schematica per ridurre spazio nell'editor grafico:



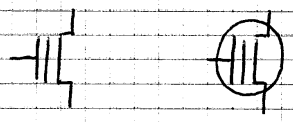
Il collegamento programmabile è costruito tramite fusibile o tramite collegamento logico (come ai fa nelle memorie)



Con questa rete $P = \overline{A+\bar{A}+B+\bar{B}} = \bar{A} \cdot A \cdot \bar{B} \cdot B$

Le word line in questo caso sono gli ingressi della rete.

Se si programmano i FAMOS:



← Gate carica di elettroni

ATTIVO $V_{th} = V_{th_L}$

INATTIVO $V_{th} = V_{th_H}$

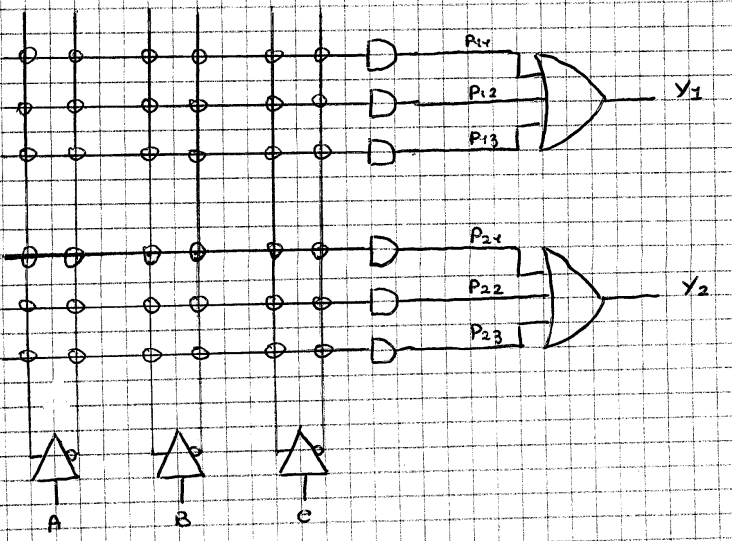
Es: $P = A \cdot \bar{B}$

È possibile aggiungere in parallelo ulteriori Bit Line.

PAL BIPOLARI: Selezione con fusibili.

PAL MOS: Selezione con Famos.

Architettura Basica di un dispositivo PAL



Limiti architettura:
- numero ingressi (es. 3)
- numero output (es. 2)
- numero di termini di prodotto che fanno parte di ciascuna funzione OR (es. 3)

Un PAL è un piano AND programmabile ed un successivo piano fisso OR.

Nell'esempio considerato:

A B \ C	0 0	0 1	1 1	1 0
0	(1)	0	(1)	0
1	0	(1)	0	(1)

Questa funzione è da maggiore. Per realizzarla avrei necessità di 4 termini di prodotto ma nella cella considerata ne ho solo 3.

Soluzione:
- Se ho a disposizione un ingresso D, collego l'uscita parziale su D e userò un microchip solo per il calcolo totale.
- Calcolo una parte su una fpga e una parte su un'altra fpga e poi sommo esternamente.

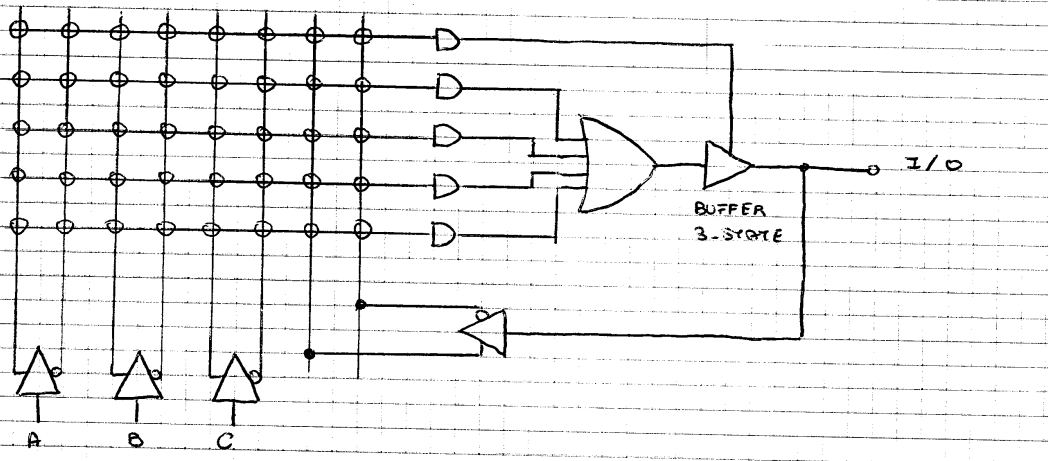
Se invece di un piano OR fisso un piano NOR, questo può essere utile.

es: $Y = \underset{OR}{A + B + C + D}$ deve avere 4 somme di prodotti non disponibile in quest'architettura

$\bar{Y} = \underset{NOR}{\overline{A+B+C+D}}$

$= \bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$ deve avere solo 1 somma di prodotti!

Inserire flessibilità in un'architettura PAL integrata



Quando uscita è in mod. I:

- rinuncio ad out per avere un ulteriore input ed in questo caso il buffer deve essere in alta impedenza.

Sfrutto l'ulteriore ingresso in un'altra sezione.

PAL XX □ YY

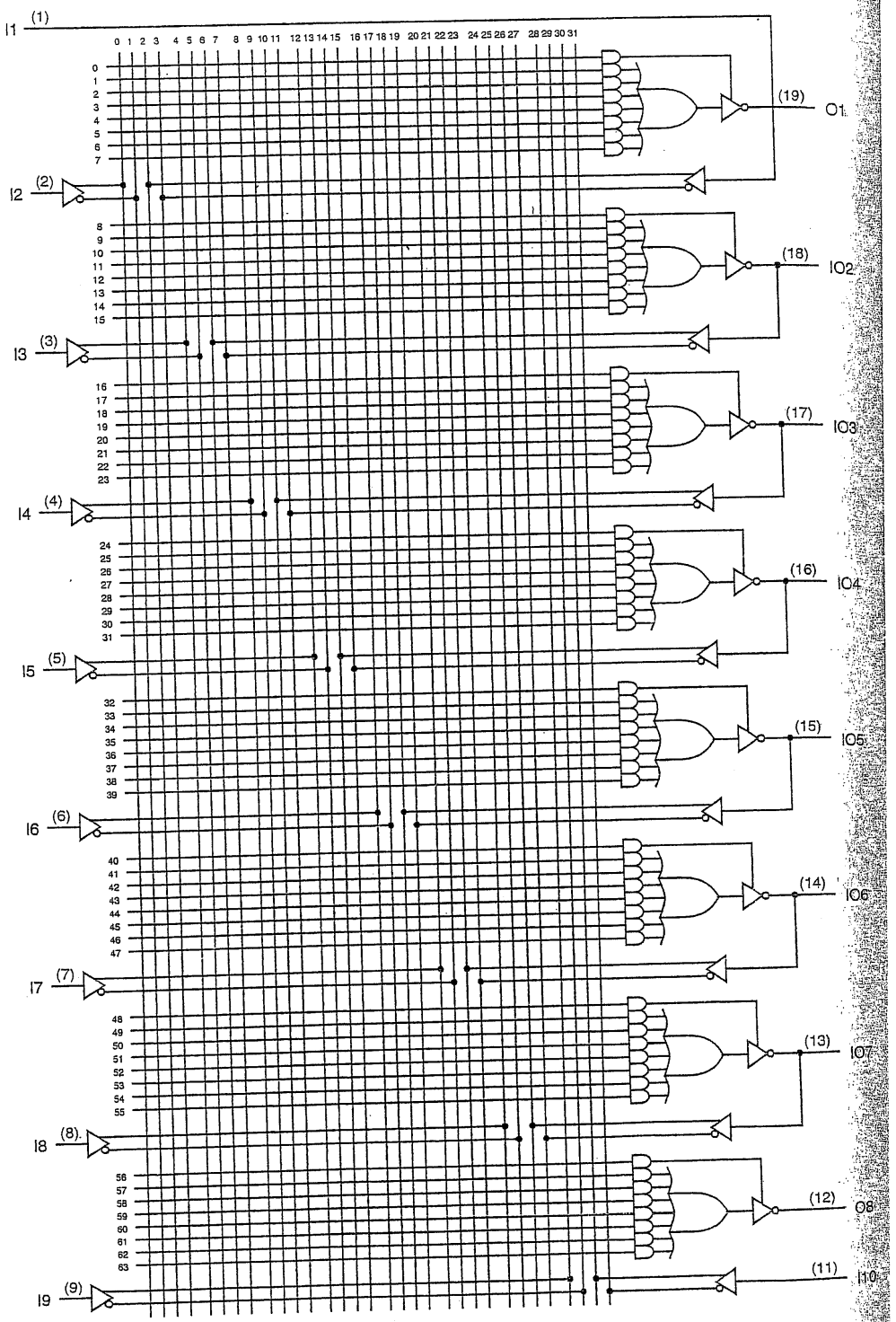
□: H: Sintetizzata con OR

L: Sintetizzata con NOR

XX: max input

YY: max output

6-8
liagram of
L16L8.



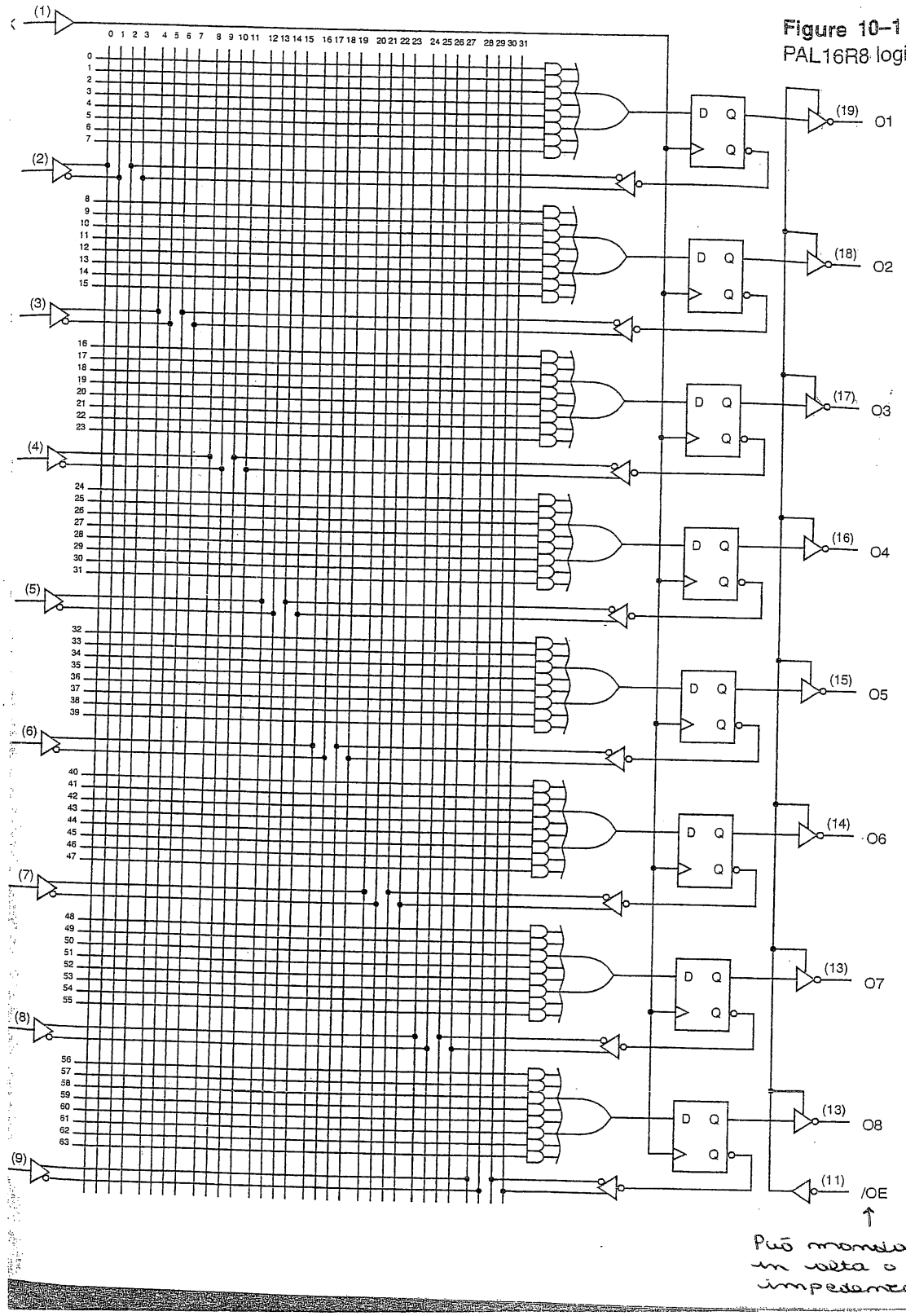


Figure 10–1
PAL16R8 logic diagram

Può mandare out in alta o bassa impedenza.

PAL 16 L 8

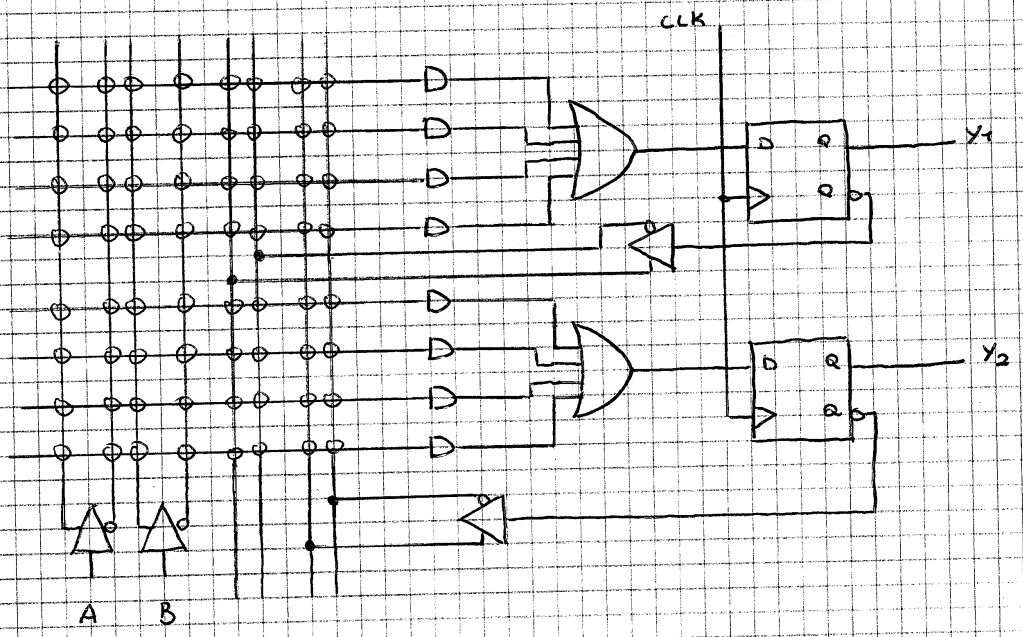
- 2 out
- 6 in / out
- 10 in

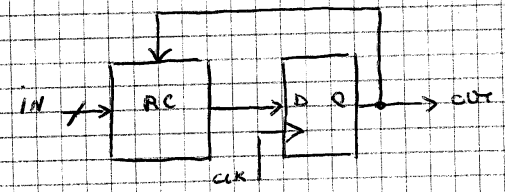
Numero di collegamenti programmabili:

8 x 8 = 64 righe

16 x 2 = 32 colonne → 2048 collegamenti programmabili

PAL per funzioni logiche sequenziali





Coincide con una macchina di Moore

→ Mealy coi mintermini con la var. di stato.

PAL 16 R 8

8 uscite : 2^8 stati possibili

8 ingressi (+ 8 ingressi in reazione)

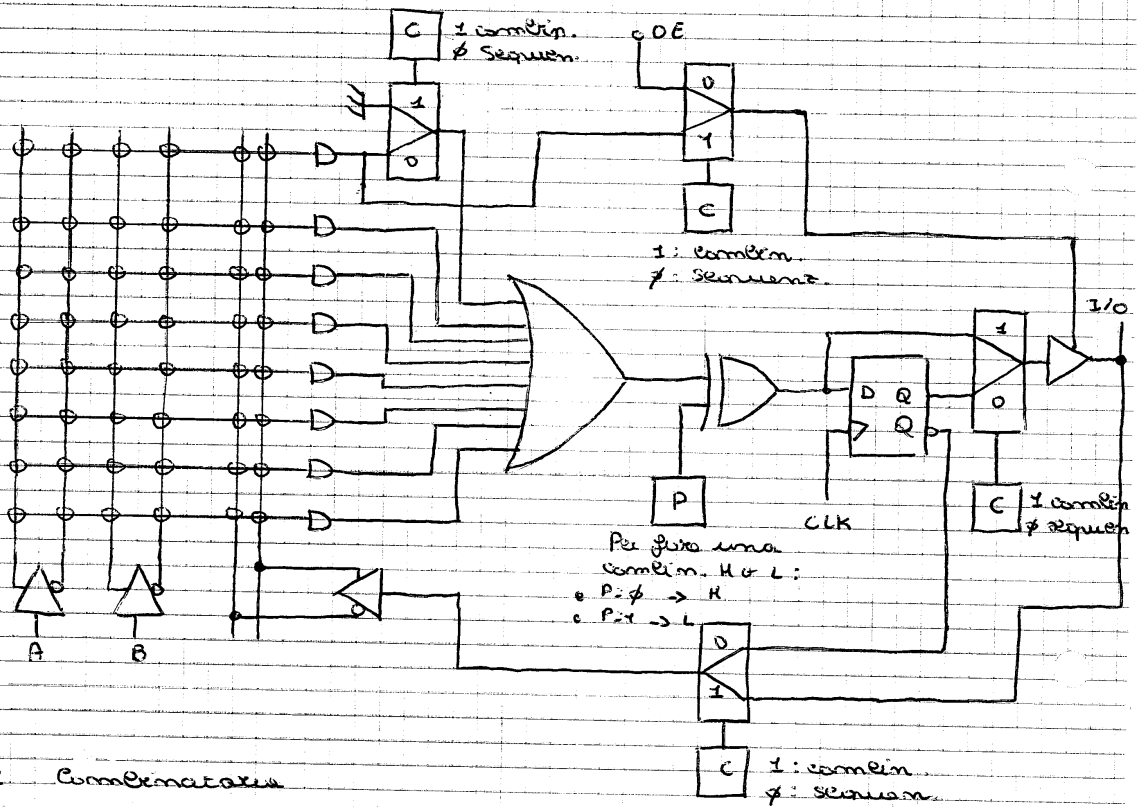
(8 + 8) x 2 = 32 colonne

8 x 8 = 64 righe → 2048 collegamenti programmabili

PAL combinatorie o sequenziali

→ collegamento programmabile definisce che logica che tipo di architettura.

GAL (Generic Array Logic) } PAL combinatorie o sequenziali.
PAL XX V 8 (V: Variabile)

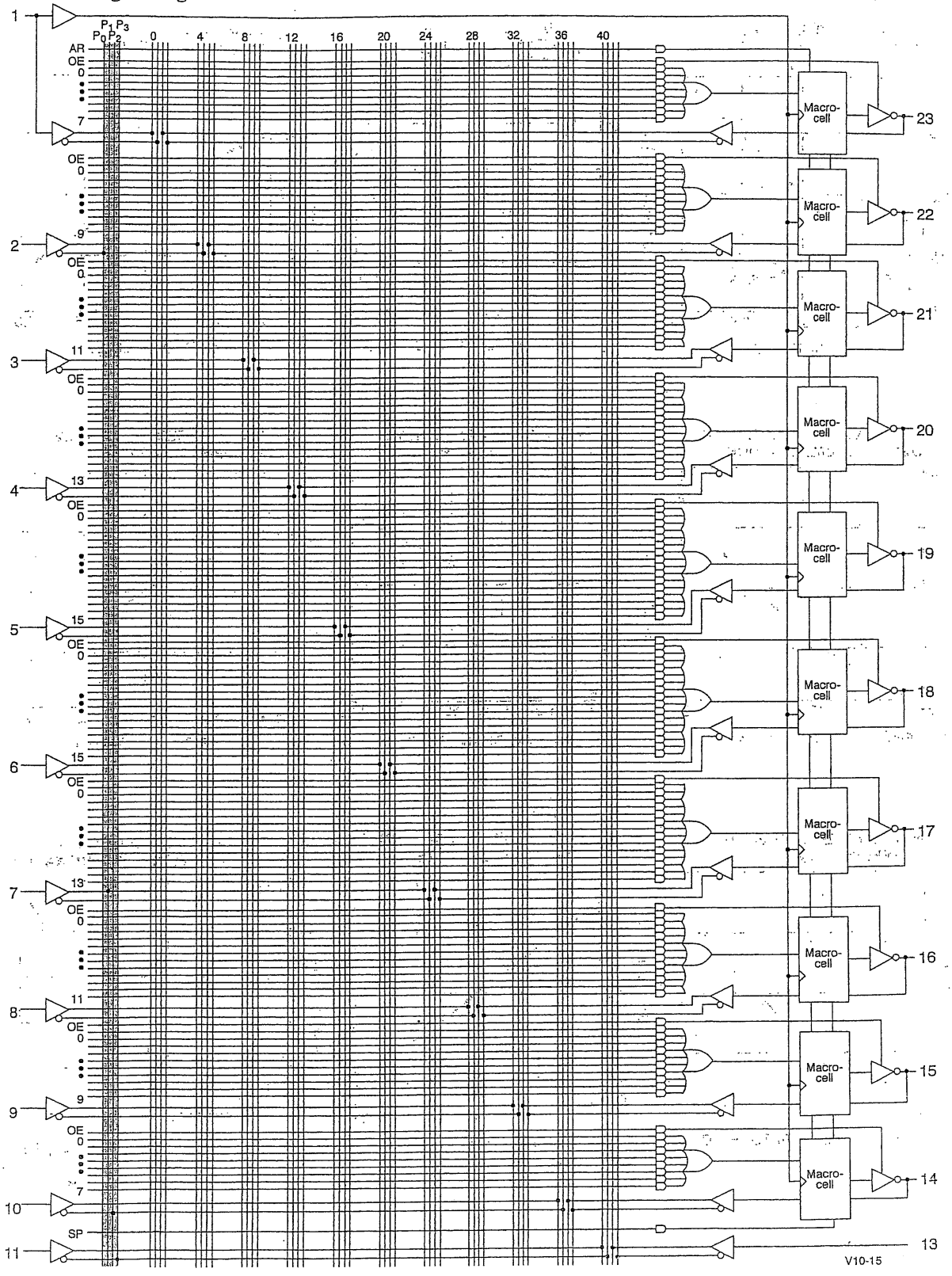


c = 1 Combinatorie
c = 0 Sequenziale

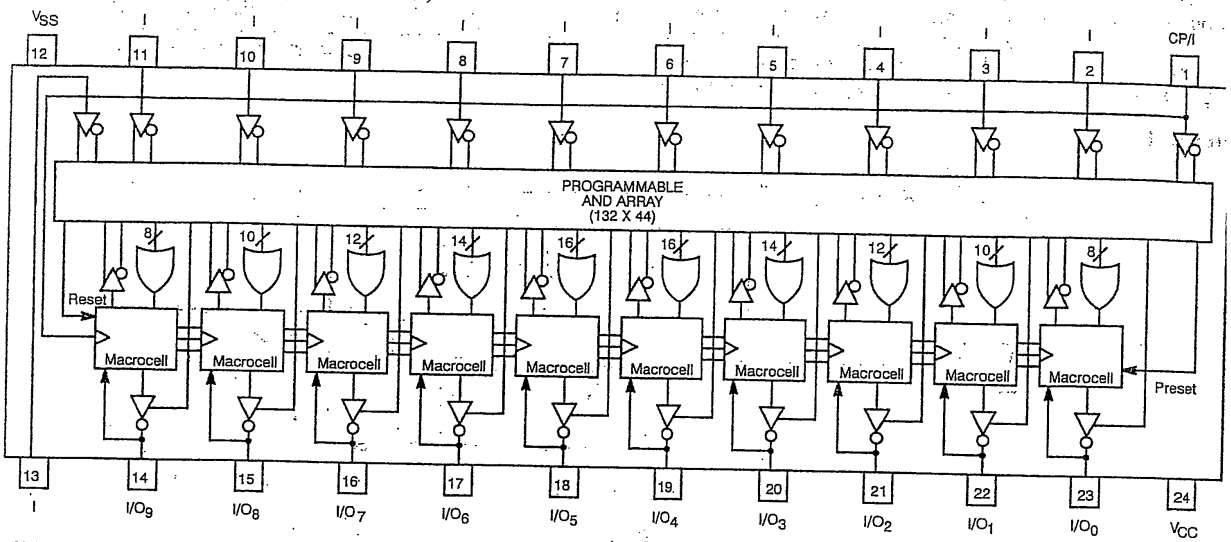
P = 1 L
P = 0 H

Eventualmente potrei pensare di configurare esternamente da singola cella in modo da attivare selettivamente combinatorie o della sequenziali nella stessa PAL.

Functional Logic Diagram for PALC22V10



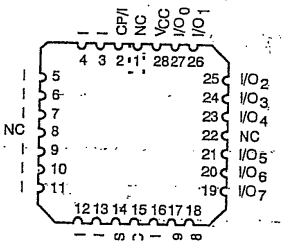
Logic Block Diagram (PDIP/CDIP)



V10-1

Pin Configuration

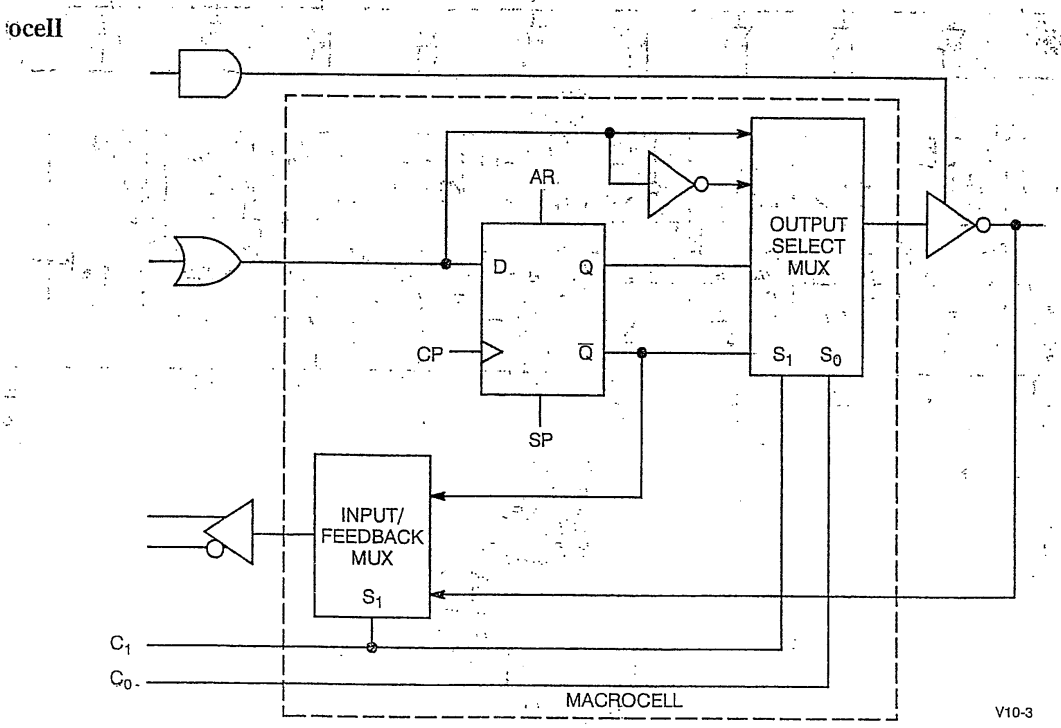
LCC/PLCC
Top View



21 inputs and only a single output and down to 12 inputs outputs are possible. The 10 potential outputs are enabled product terms. Any output pin may be permanently se-

1	1	Combinatorial/Active HIGH

ocell



V10-3

27/10/2017

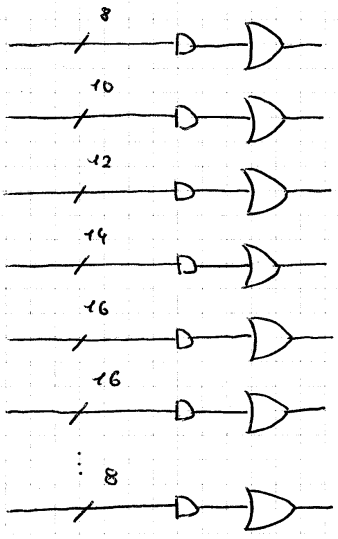
PAL 22V10

10 I/O

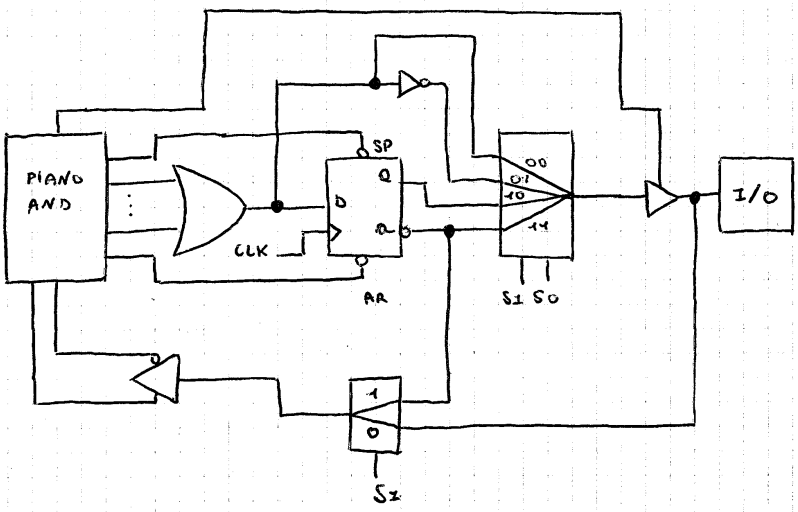
12 I (10 sono ingressi retroazionati da I/O)

Questo dispositivo consente di configurarsi in maniera individuale ogni macrocella.

Qui per ogni funzione i termini di prodotto sono variabili: da 8 a 16.



10 celle.



CLK è un segnale globale.

(8 + 16 + 32 + 64 + 16) . 2 : termini di prodotto complessivi = 120

10 termini di prodotto per il controllo di impedenza, 1 per reset asincrono (AR) e 1 per preset sincrono (SP)

120 + 10 + 1 + 1 = 132 righe.

22 . 2 colonne (segnale + segnale negato) = 44

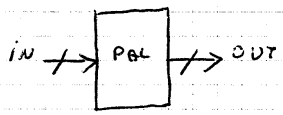
132 . 44 = 5808. elementi programmabili nel piano AND

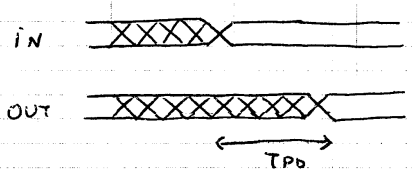
2 . 10 colonne . righe per il controllo dei MUX

tot : 5828 elementi programmabili.

Analizziamo l'architettura in termini di tempo di propagazione :
Le PAL, per via della loro architettura, hanno ingressi ed uscite funzionalmente equivalenti, ovvero dovunque entro e dovunque esco la prestazione non cambia significativamente. Il trasferimento su silicio non pone vincoli alle prestazioni ottenibili.

PAL COMBINATORIA può essere vista come :



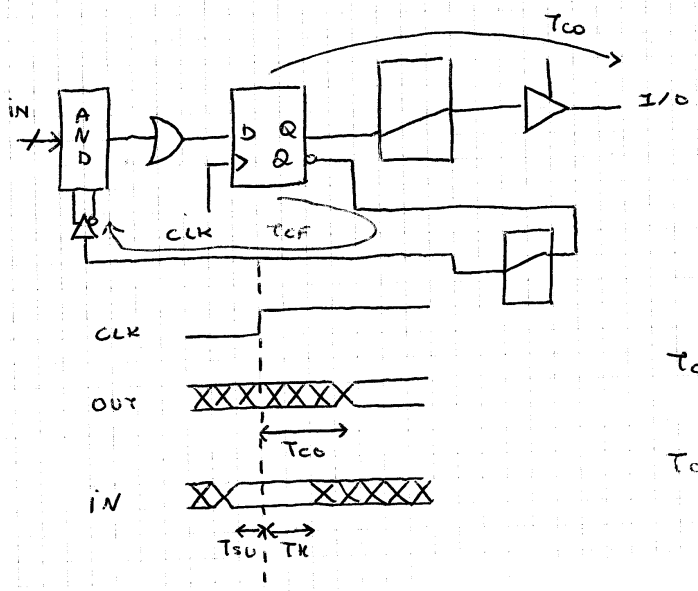


T_{PD} : tempo tra variazione ingresso ed uscita stabile.

T_{PD} ha significato se la PAL è realizzata in una singola passata : ovvero se sono sufficienti i termini di prodotto disponibili nel piano AND.

Altrimenti il tempo di propagazione aumenta rispetto a quello minimo (che è appunto T_{PD}

PAL SEQUENZIALI più spesso viste come:



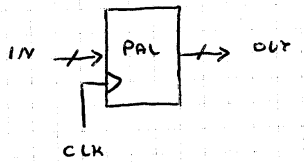
T_{SU} : tempo mantenimento IN stabile prima della variazione di CLK.

T_H : tempo mantenimento IN stabile dopo la variazione di CLK.

T_{co} : ritardo tra variazione clock ed uscita stabile.

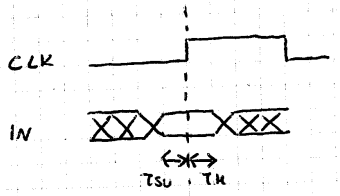
T_{CF} : ritardo tra variazione clock e ricezione sugli ingressi stabili

Prestazioni di una PAL sequenziale definite a priori.



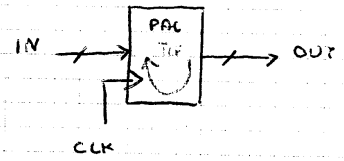
PAL in modalità no feedback (campionatore)

Funzionamento corretto di un sistema digitale: ingressi stabili nell'intervallo di SET UP e di HOLD.

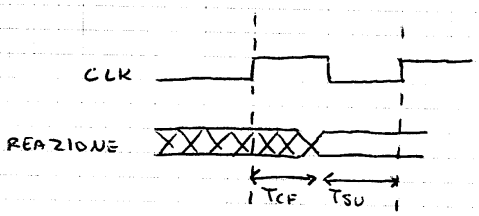


$T_{CLK} \geq T_{SU} + T_H$

$$f_{MAX\ \text{no feedback}} = \frac{1}{T_{SU} + T_H}$$

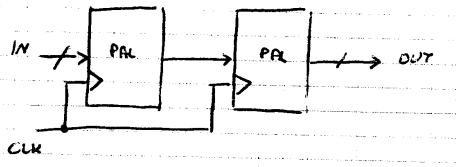


PAL in modalità feedback interno



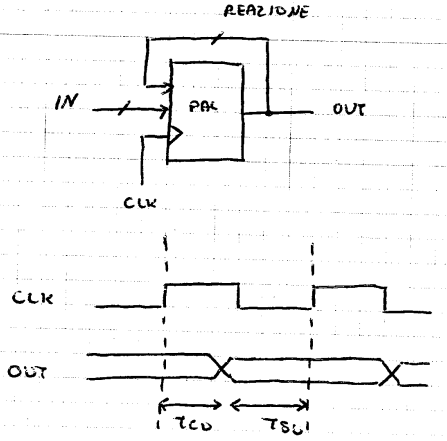
$T_{CLK} \geq T_{CF} + T_{SU}$

$$f_{CLK\ feedback\ interno} = \frac{1}{T_{CF} + T_{SU}}$$



PAL in modalità feedback esterno.

Corrisponde a:



$T_{CLK} \geq T_{CO} + T_{SU}$

$$f_{CLK\ feedback\ esterno} = \frac{1}{T_{CO} + T_{SU}}$$

Commercial Switching Characteristics PALCE22V10[2,7]

Parameter	Description	22V10-5		22V10-7		22V10-10		22V10-15		22V10-25		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t_{PD}	Input to Output Propagation Delay[8]	3	5	3	7.5	3	10	3	15	3	25	ns
t_{EA}	Input to Output Enable Delay[9]		6		8		10		15		25	ns
t_{ER}	Input to Output Disable Delay[10]		6		8		10		15		25	ns
t_{CO}	Clock to Output Delay[8]	2	4	2	5	2	7	2	8	2	15	ns
t_{S1}	Input or Feedback Set-Up Time	3		5		6		10		15		ns
t_{S2}	Synchronous Preset Set-Up Time	4		6		7		10		15		ns
t_H	Input Hold Time	0		0		0		0		0		ns
t_P	External Clock Period ($t_{CO} + t_S$)	7		10		12		20		30		ns
t_{WH}	Clock Width HIGH[6]	2.5		3		3		6		13		ns
t_{WL}	Clock Width LOW[6]	2.5		3		3		6		13		ns
f_{MAX1}	External Maximum Frequency ($1/(t_{CO} + t_S)$)[11]	143		100		76.9		55.5		33.3		MHz
f_{MAX2}	Data Path Maximum Frequency ($1/(t_{WH} + t_{WL})$)[6, 12]	200		166		142		83.3		35.7		MHz
f_{MAX3}	Internal Feedback Maximum Frequency ($1/(t_{CF} + t_S)$)[6,13]	181		133		111		68.9		38.5		MHz
t_{CF}	Register Clock to Feedback Input[6,14]		2.5		2.5		3		4.5		13	ns
t_{AW}	Asynchronous Reset Width	8		8		10		15		25		ns
t_{AR}	Asynchronous Reset Recovery Time	4		5		6		10		25		ns
t_{AP}	Asynchronous Reset to Registered Output Delay		7.5		12		13		20		25	ns
t_{SPR}	Synchronous Preset Recovery Time	4		6		8		10		15		ns
t_{PR}	Power-Up Reset Time[6,15]	1		1		1		1		1		µs

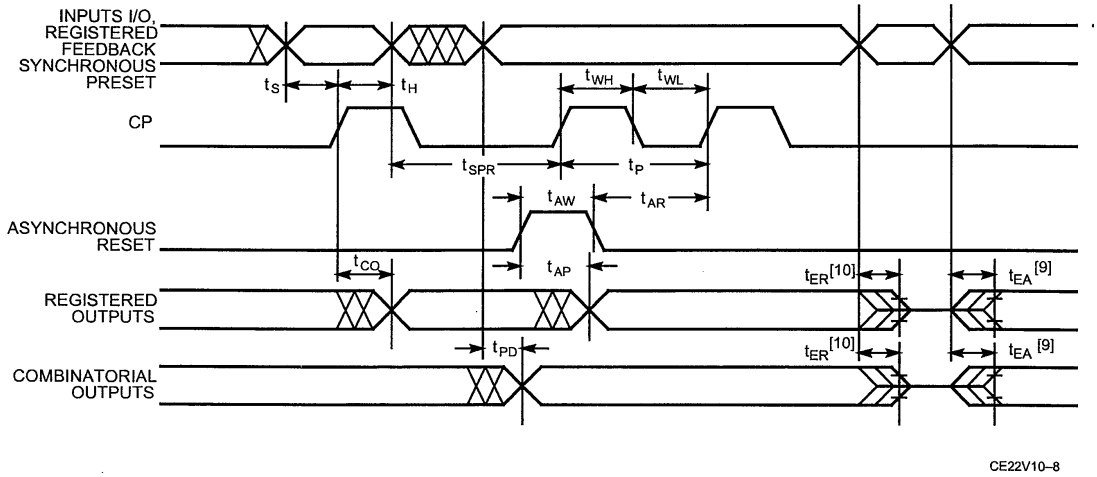
Notes:

7. Part (a) of AC Test Loads and Waveforms is used for all parameters except t_{ER} and $t_{EA(+)}$. Part (b) of AC Test Loads and Waveforms is used for t_{ER}. Part (c) of AC Test Loads and Waveforms is used for $t_{EA(+)}$.
8. Min. times are tested initially and after any design or process changes that may affect these parameters.
9. The test load of part (a) of AC Test Loads and Waveforms is used for measuring $t_{EA(-)}$. The test load of part (c) of AC Test Loads and Waveforms is used for measuring $t_{EA(+)}$ only. Please see part (e) of AC Test Loads and Waveforms for enable and disable test waveforms and measurement reference levels.
10. This parameter is measured as the time after output disable input that the previous output data state remains stable on the output. This delay is measured to the point at which a previous HIGH level has fallen to 0.5 volts below V_{OH} min. or a previous LOW level has risen to 0.5 volts above V_{OL} max. Please see part (e) of AC Test Loads and Waveforms for enable and disable test waveforms and measurement reference levels.
11. This specification indicates the guaranteed maximum frequency at which a state machine configuration with external feedback can operate.
12. This specification indicates the guaranteed maximum frequency at which the device can operate in data path mode.
13. This specification indicates the guaranteed maximum frequency at which a state machine configuration with internal only feedback can operate.
14. This parameter is calculated from the clock period at f_{MAX} internal ($1/f_{MAX3}$) as measured (see Note above) minus t_S.
15. The registers in the PALCE22V10 have been designed with the capability to reset during system power-up. Following power-up, all registers will be reset to a logic LOW state. The output state will depend on the polarity of the output buffer. This feature is useful in establishing state machine initialization. To insure proper operation, the rise in V_{CC} must be monotonic and the timing constraints depicted in Power-Up Reset Waveform must be satisfied

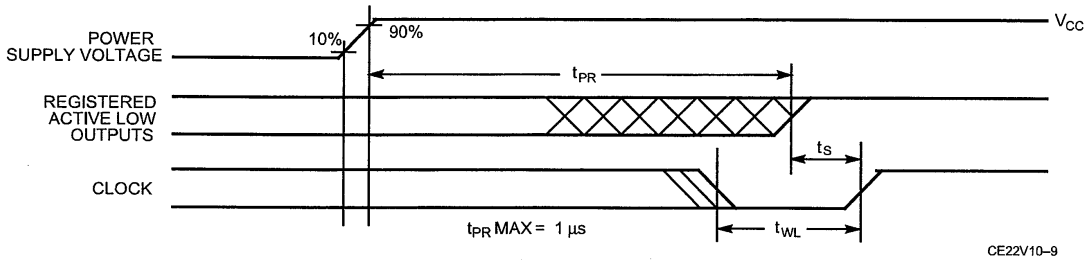
Se considero $f_{MAX} = 1/t_{H1}\,t_{S1} = 333$ MHz > $f_{MAX\,2} = 200$ MHz

$f_{MAX\,2} = 1/T_{WH} + T_{WL}$: max frequenza alla quale gli elementi logici interni lavorano a funzionare.

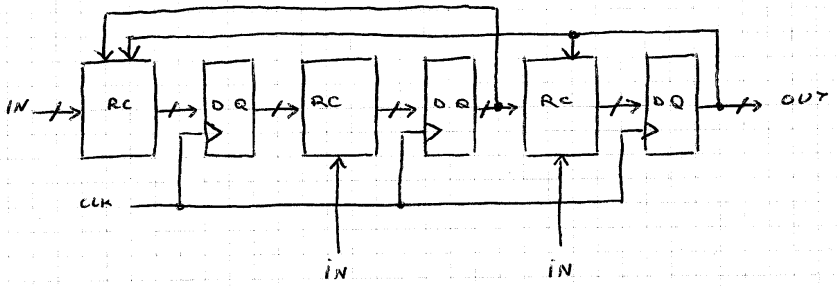
Switching Waveforms



Power-Up Reset Waveform[15]



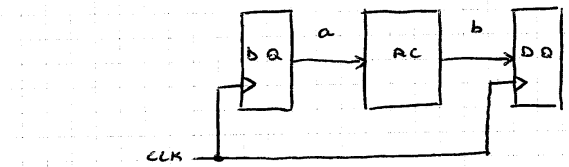
Architettura di un sistema digitale generico:
Sistema digitale sincrono di esempio.

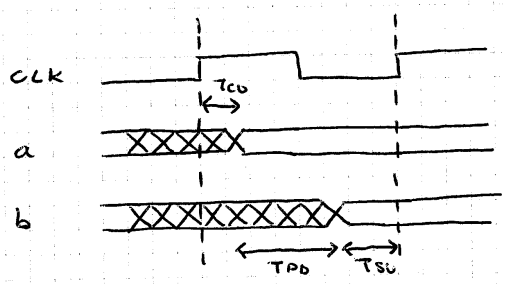


Valutare le prestazioni di questo sistema:
è necessario valutare il percorso critico

Esamino tutti i percorsi che uniscono ingresso di un flip flop con uscita di un altro flip flop precedente.

es:





1) $T_{CLK} \geq T_{CO} + T_{PD} + T_{SU}$

T_{PD}: tempo di propagazione del segnale nella logica → tempo tra variazione a e b stabile.

$$f_{CLK\ RLR} = \frac{1}{T_{CO} + T_{PD} + T_{SU}}$$ CRITICAL PATH

Valuto tutti i percorsi e la velocità massima di esecuzione è data dalla logica RLR che ha il T_{PD} maggiore se tutti i flip-flop sono uguali.

In una PAL il T_{PD} è noto a priori in quanto l'architettura è già costruita; per un generico sistema digitale combinatorio T_{PD} va valutato.

03/11/2017

la seconda condizione da rispettare è dovuta al tempo di hold:

2) $T_{CO} + T_{PD} \geq T_H$

-> il dato in ingresso può cominciare a variare solo successivamente al termine del tempo di hold T_H.

Dunque le due condizioni da rispettare per una RLR:

1) $T_{CLK} \geq T_{CO} + T_{PD} + T_{SU}$

2) $T_{CO} + T_{PD} \geq T_H$

Torniamo a discutere le architetture PAL:

La complessità circuitale di una PAL aviene quadraticamente al numero di fasce -> aumentano righe ed aumentano colonne.

Questo rappresenta in termine di carichi introdotti e ritardi di propagazione.

L'aumento degli I/O produce un incremento esponenziale delle dimensioni dell'architettura.

L'evoluzione tecnologica si divide in architetture PAL-LIKE ed architettura ASIC-LIKE.

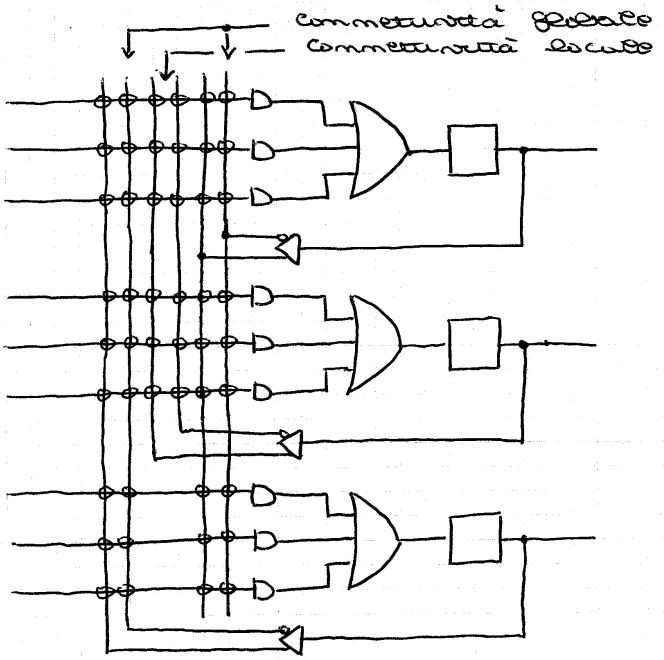
Componenti PAL-LIKE sono denominati come dispositivi PLD (programmable logic device) e CLD (Complex logic device).

Componenti ASIC LIKE sono denominati come dispositivi FPGA (Field Programmable Gate Array)

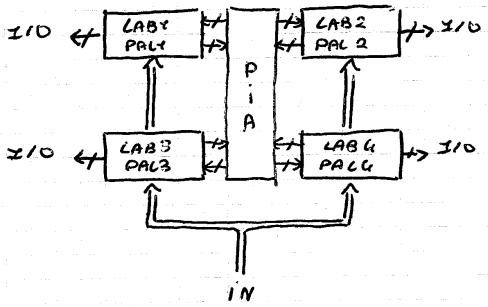
L'idea di queste architetture è di aggirare i limiti delle PAL nell'espansione in termini di I/O, dovute alla complessità circuitale eccessiva.

La prima idea è che non tutti i nodi debbano essere obbligatoriamente interconnessi.

Connettività globale della PAL lascia il posto ad una connettività locale.



Questa idea rappresenta concretamente ciò che facciamo MAX (Multiple Array Matrix)
All'interno di questa architettura convivono tante piccole PAL autonome con una risorsa fra loro condivisa.



- Risorsa di comunicazione programmabile.

PIA : PROGRAMMABLE INTERCONNECT ARRAY

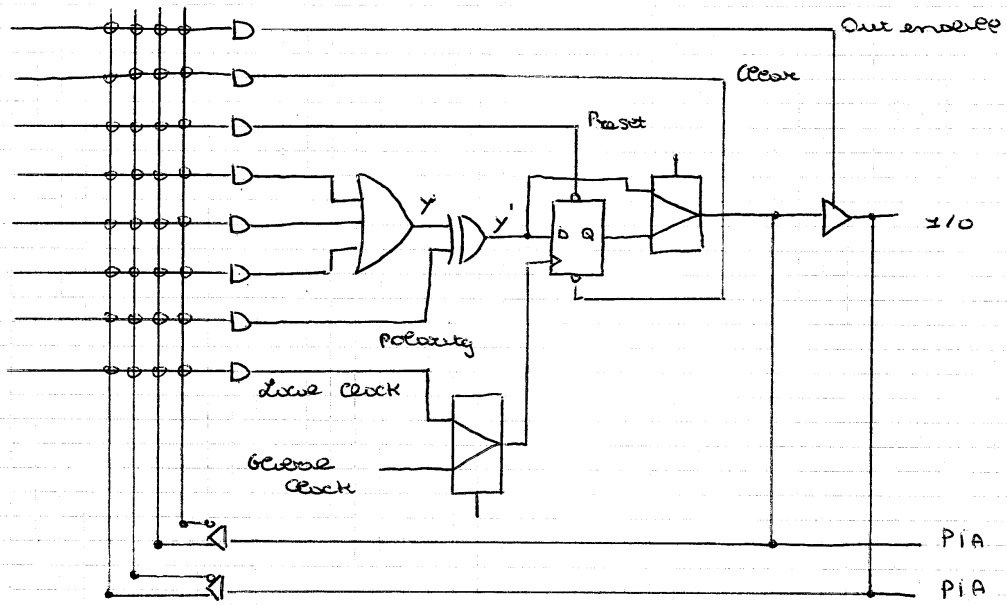
PAL indipendenti interconnesse da connessioni programmabili.

Ci sono elementi programmabili per fare logica ed elementi programmabili per fare interconnessione.

-> CONNETTIVITA' E LOGICA PROGRAMMABILE

FAMIGLIA MAX 5000

Rappresentazione di una SLICE all'interno di un LAB.



3 termini di prodotto logici, 1 termine di polarità, 1 termine di preset ed 1 di clear, 1 termine di abilitazione dell'uscita

Vantaggio della doppia reazione: a differenza della singola reazione nella PAL, volendo impostando I/O = 0 posso usare il pin come output ma comunque utilizzare la logica per elaborare l'informazione.

Questo grazie alle due reazioni separate.

Ricco a realizzare Buried functions: funzioni segrete.