Custom single-purpose processors

16.05.2014

Marco Faccio

Outline

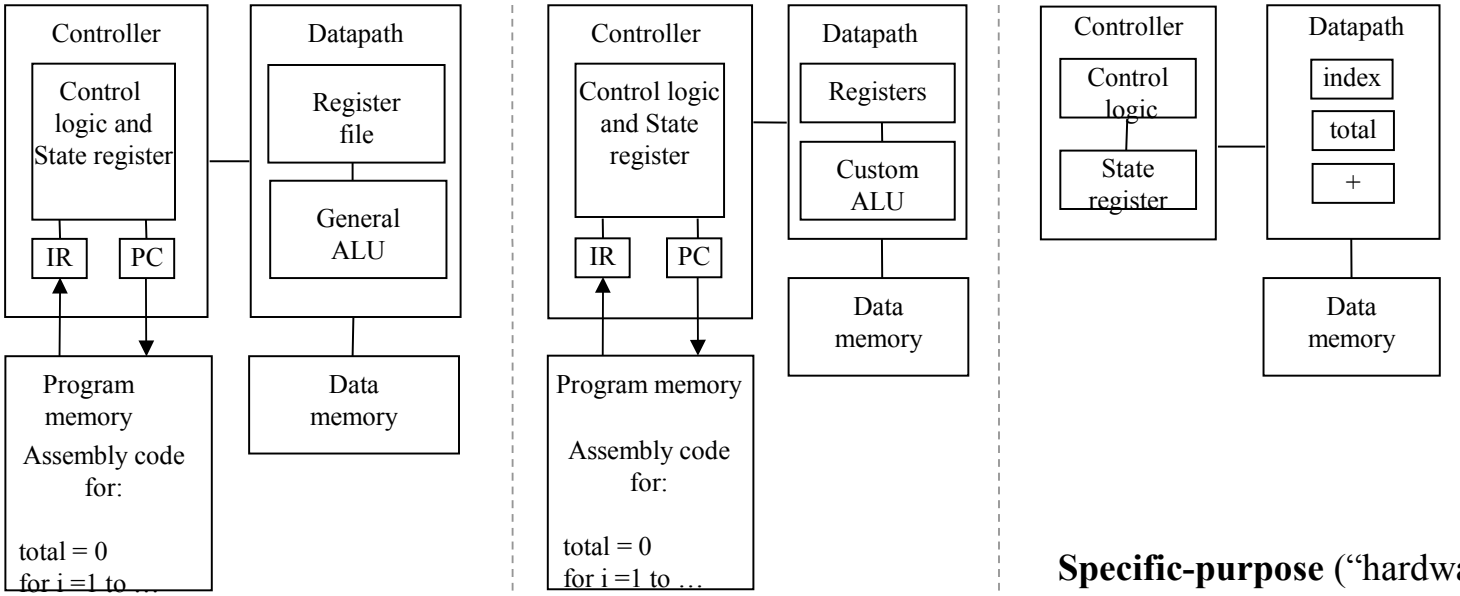
- Introduction
  - Rivisitazione di quanto definito ….
  - Ripartenza dal basso: dall'HW al sistema
- Modelli computazionali in Hw – Moore, Mealy – Automi a Stati Finiti ….
  - Combinational logic
  - Sequential logic
- Custom single-purpose processor design
- RT-level custom single-purpose processor design

Three key embedded system technologies

- Technology
  - A manner of accomplishing a task, especially using technical processes, methods, or knowledge
- Three key technologies for embedded systems
  - Processor technology
  - IC technology
  - Design technology

Processor technology

- The architecture of the *computation engine* used to implement a system's desired functionality
- ***Processor*** does not have to be ***programmable***
  - "Processor" *not* equal to general-purpose processor



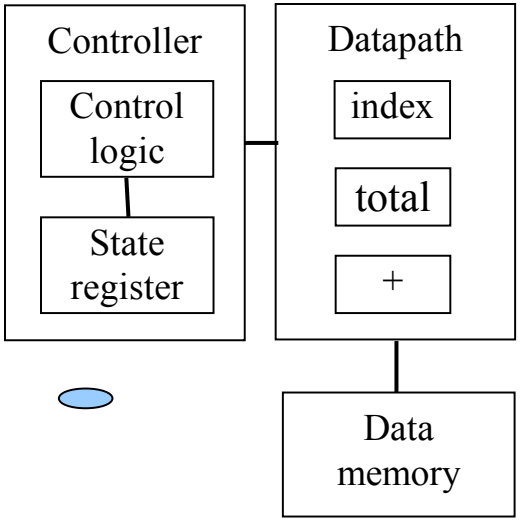
General-purpose ("software") **Application-specific** **Specific-purpose** ("hardware") **Single-purpose** ("hardware")

GP P + ASP

ASIP + SPP

Single-purpose processors

- Digital circuit designed to execute exactly one program
  - a.k.a. coprocessor, accelerator or peripheral
- Features
  - Contains only the components needed to execute a single program
  - No program memory (?!?!?)
- Benefits
  - Fast
  - Low power
  - Small size



Perchè ha un senso …
- occupazione di area ridotta;
- velocità di processamento;
- dissipazione di potenza ridotta;

LIMITI !!
- test
- parametrizzazione
- tempo sviluppo
- ….

Riflessioni preliminari *1*

Analisi dei "termini" di titolo:

- *Processor* …..

Automa *esecutore* è *digitale*

- *Single-purpose* (o anche *Specific-purpose*)
- *Custom* ….

- ***Necessarietà di ripartire dal basso*** ….. ma senza dimenticare le metodiche *topdown*, la frammentazione dei problemi in sottoproblemi, le tecniche e le metodologie proprie dell'informatica e del SW, la gestione della complessità ….. soprattutto perchè ….

Riflessioni preliminari *2*

- *... perchè ...* Un "*sistema digitale*" è un "*oggetto"* intrinsecamente molto, molto complesso … complesso da definire, complesso da *implementare* e *collaudare..*
  - Complesso per *l'overload* connesso alla codifica dell'informazione e quindi della numerosità degli elementi elaborativi elementari ….
  - Complesso nella realizzabilità fisica (circuiti integrati VLSI e ULSI),
  - Complesso perchè a fronte di definite specifiche dinamiche e di relazione *IN-OUT* esso ammetterà una molteplicità di descrizioni interne diverse *(architetture funzionali)*, per ognuna di esse ammetterà una molteplicità di implementazioni come *architetture logiche,* e ancora una molteplicità di *realizzazioni fisiche-tecnologiche* ….
- È un avvincente e straordinaria sfida ***ingegneristica*** ...

Riflessioni preliminari *3*

- Per "*esorcizzare*" le difficoltà connesse alla complessità e alla molteplicità di soluzioni implementative e realizzative, si può ricorrere e attuare una metodologia tipo *mattoncini LEGO + strumenti* e *metodologie* per sostenere contemporaneamente l'approccio *top-down* e quello *bottom-up,* capaci di *vedere* e *sintetizzare* Hw ma anche Sw ….
  - Alcune *entità base,* complete, modulari e *direttamente implementabili in HW ....*
  - Insieme di regole per comporle in *aggregati* superiori …
  - approccio *top-down* fino ad incrociare *aggregati* definiti oppure *entità di base ...*

Riflessioni preliminari *4*

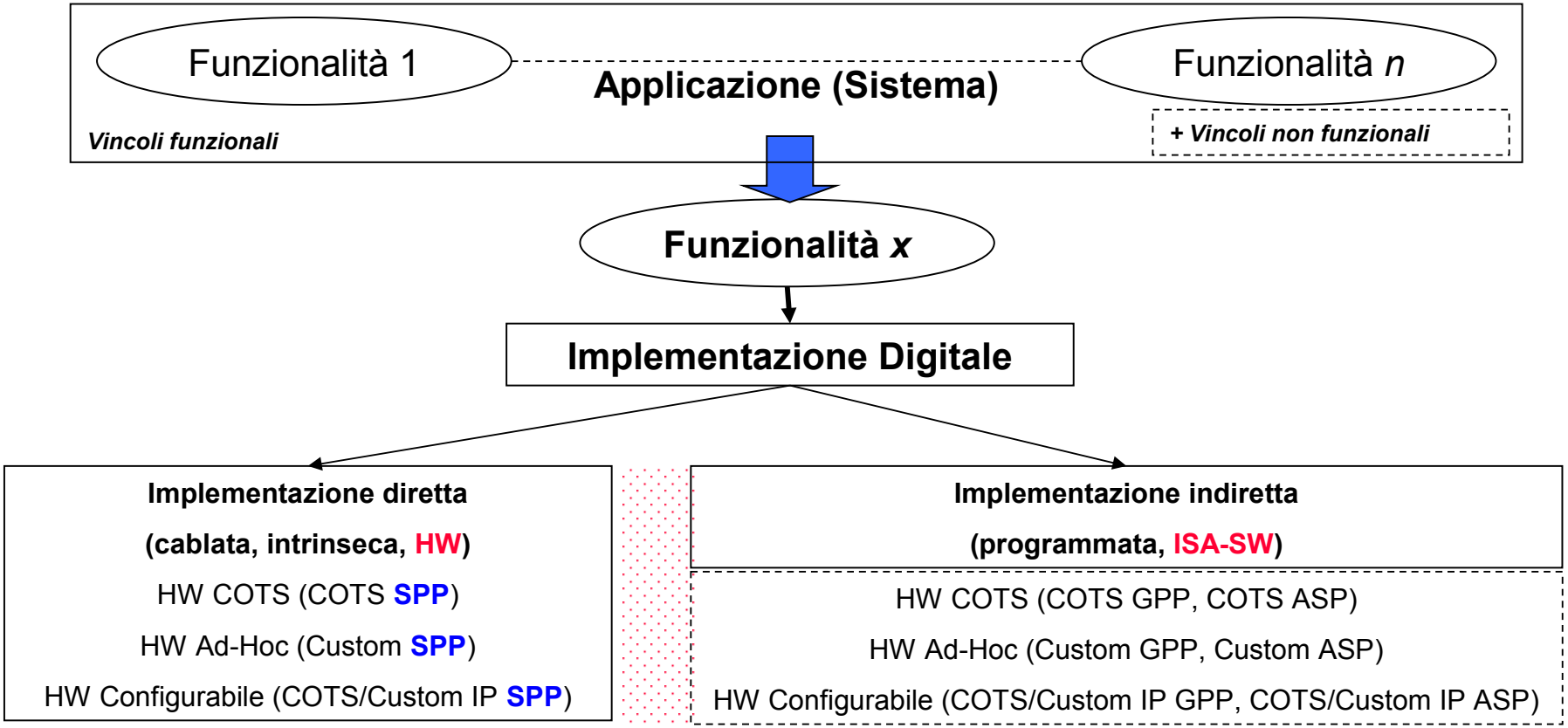
- Ci *piace* definire un ***sistema digitale*** come il "*prodotto di sintesi di definiti* ***algoritmi***" ….
  - Un *algoritmo* (comunque complesso) e/o una sub-collezione di algoritmi come ***descrizione*** della legge di trasformazione che il sistema deve esplicare ….
  - Capacità di trasformare, grazie ad una solida preparazione di base e con adeguati strumenti CAD, le descrizioni algo in soluzioni fisicamente realizzate … appartenenti all'insieme definito

Riflessioni preliminari 5

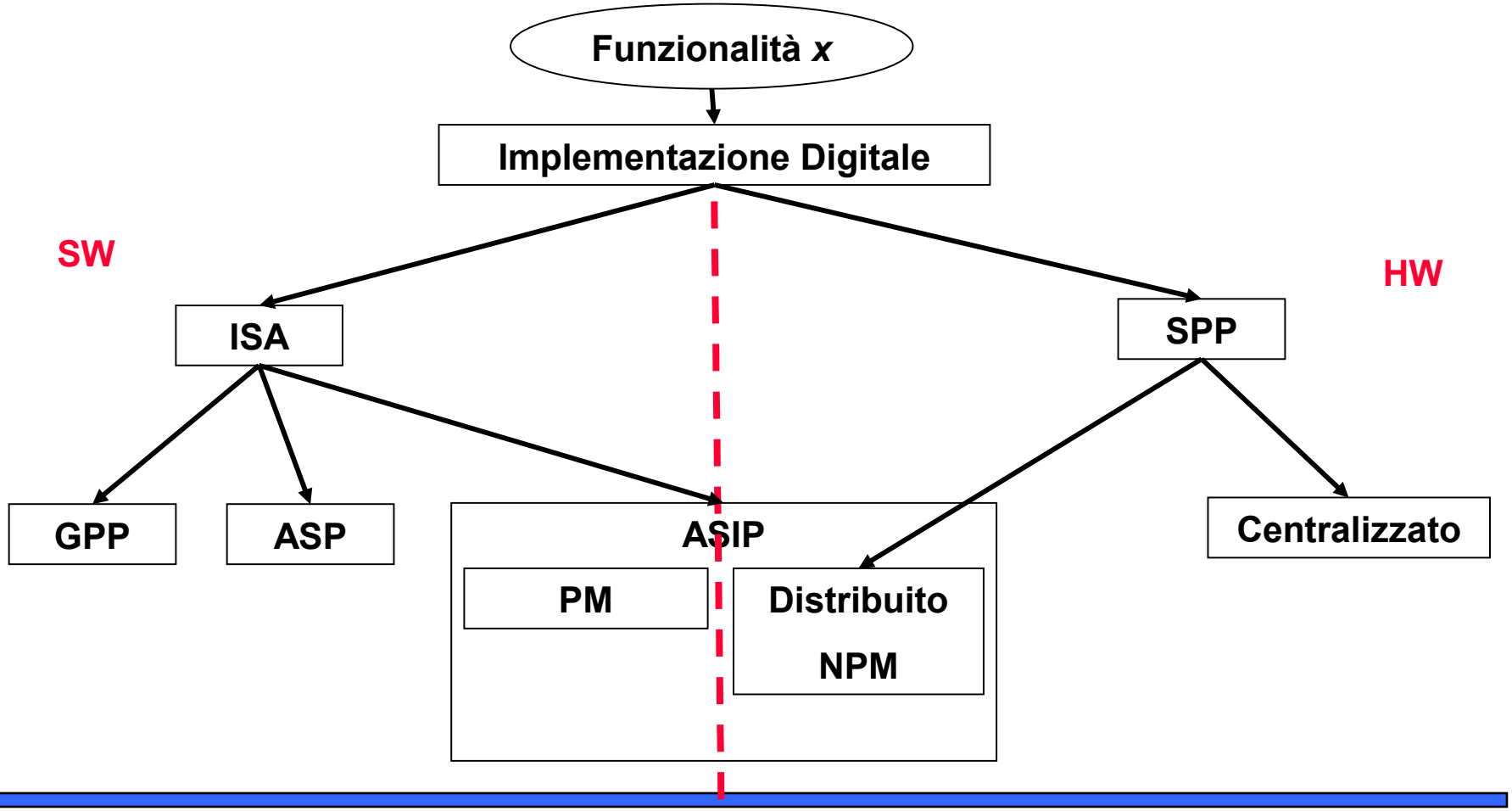
- Storicamente i “sistemi digitali” nascono dal basso come “oggetti” *full-HW*, costola dell'elaborazione analogica.
- Come per *l'analogica*, le “funzionalità” digitali implementate sono connesse alla circuiteria sintetizzata per le loro realizzazioni..
- bisognerà aspettare 1974 per “*inventare*” la *programmazione* in un HW microintegrato
- Si registrerà nel seguito una enorme crescita tecnologica (*vlsi* e *ulsi*) ma anche una enorme crescita della capacità di descrivere “algoritmi” + sviluppo di metodologie e SW CAD;
- Anni 2000: inizia il boom delle *logiche VLSI configurabili dall'UTENTE .....*

Considerazioni

- Riassumendo…

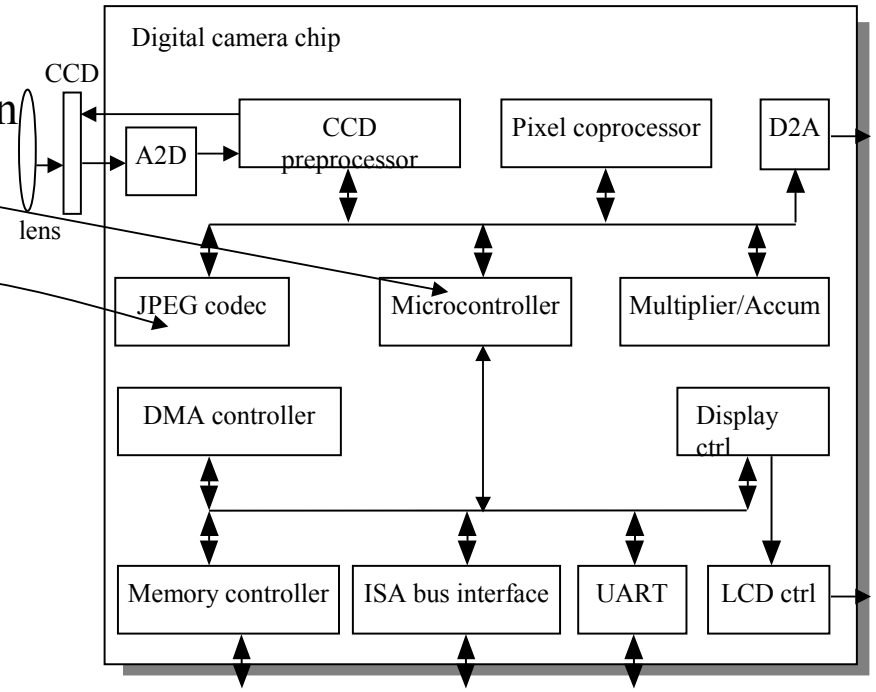


Riassumendo



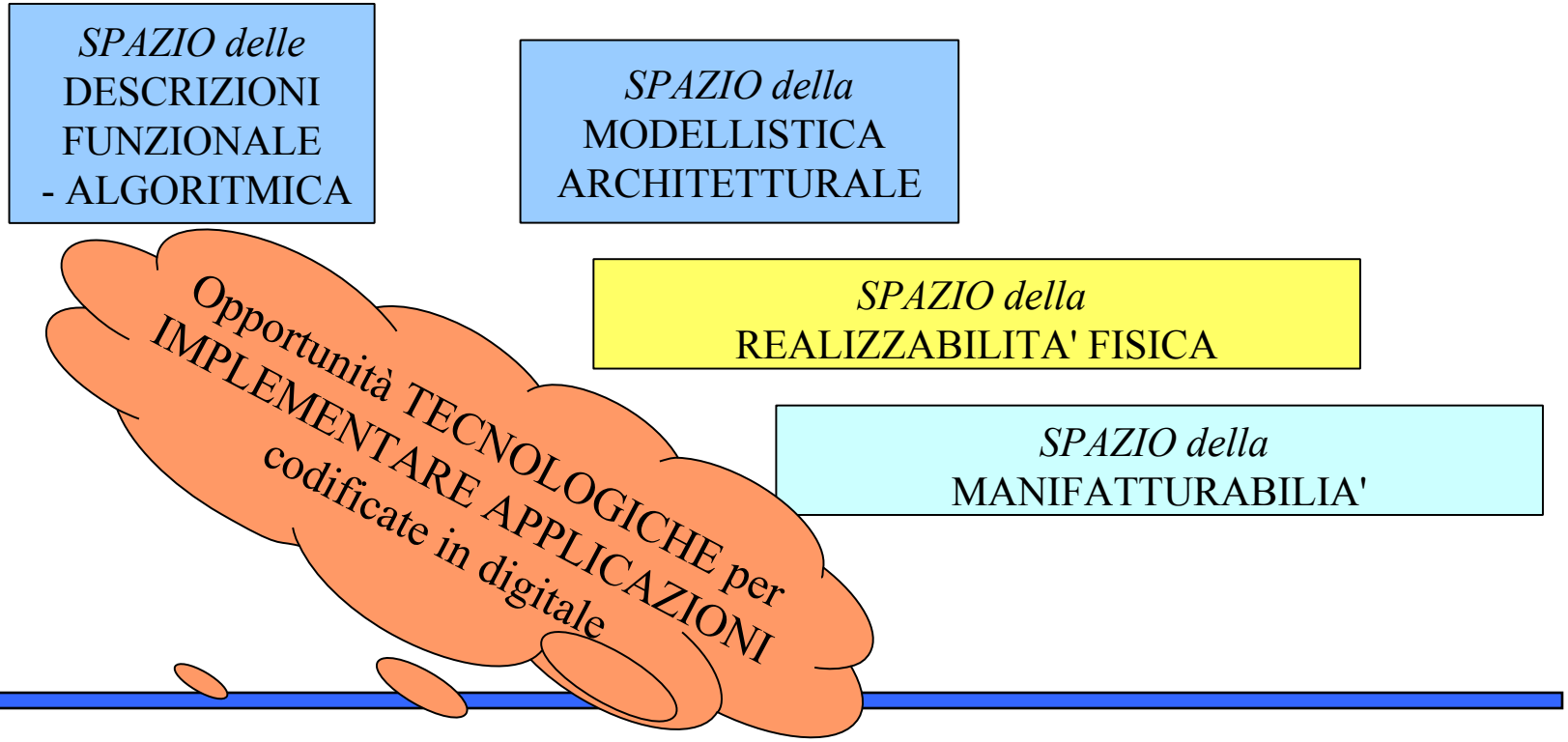
Esempio di SoC

- Processor
  - Digital circuit that performs a computation tasks
  - Controller and datapath
  - General-purpose: variety of computation tasks
  - Single-purpose: one particular computation task
  - Custom single-purpose: non-standard task
- A custom single-purpose processor may be
  - Fast, small, low power
  - But, high NRE, longer time-to-market, less flexible



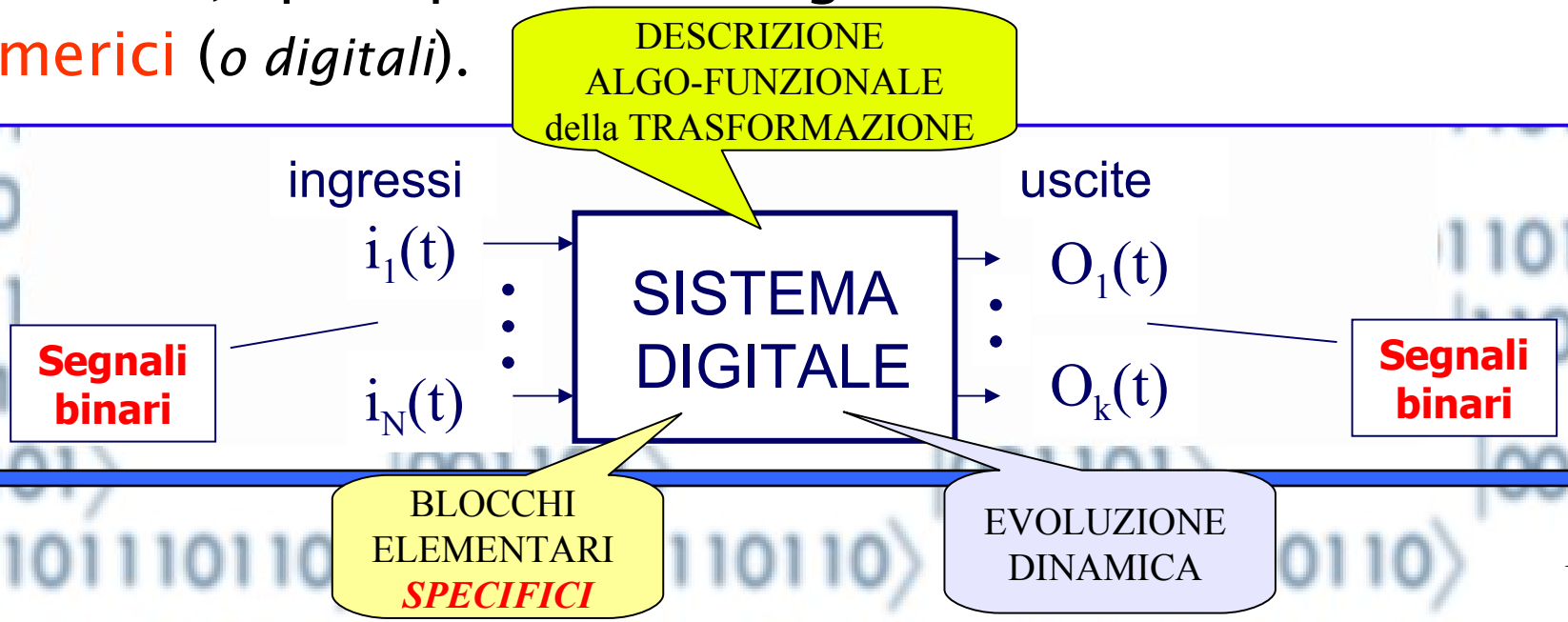
Ripartiamo dal basso

- riDefiniamo il "*sistema digitale*"
- Guardiamo le sue opportunità di sintesi e realizzazione



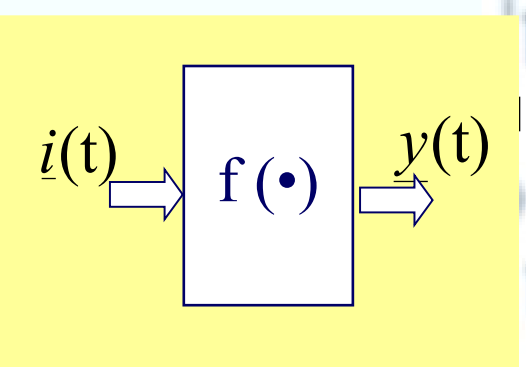
Che cos'e' un *sistema digitale*

Un sistema (o circuito) digitale e' un **sistema *dinamico*** (*che evolve nel tempo*), **artificiale** (*progettato dall'uomo*) che manipola (*elabora, trasforma, processa*) in modo **numerico** (o digitale), **segnali numerici** (o digitali) d'ingresso che ***rappresentano*** (*opportunamente*) l'informazione da *elaborare*, per produrre segnali di uscita anch'essi numerici (*o digitali*).



Il sistema

- Il ***SISTEMA*** e' rappresentato da un ***blocco funzionale*** che realizza la ***trasformazione dinamica*** degli ***ingressi*** nelle ***uscite***.



- **Modalità di evoluzione dinamica**
  - **SINCRONA**
  - **Asincrona**

- **PROBLEMA dell'ISOMORFISMO architetturale interno**
  - **??? come si sintetizzera' il suo interno??**
    - **Logica Programmata ??**
    - **Logica a Sintesi Diretta???**
    - ***Altro???***

Il blocco funzionale (interno) *2*

- nel passare dal ***problema*** al ***circuito finale*** si possono riconoscere almeno 3 livelli gerarchici descrittivi fondamentali:

1. il ***FUNZIONALE*** LOGICO

   descrizione ***formale*** del sistema e ***partizioni top-down*** in moduli e sottomoduli formalmente autoreferenti

2. l'***ARCHITETTURALE*** CIRCUITALE LOGICO (***strutturale***)

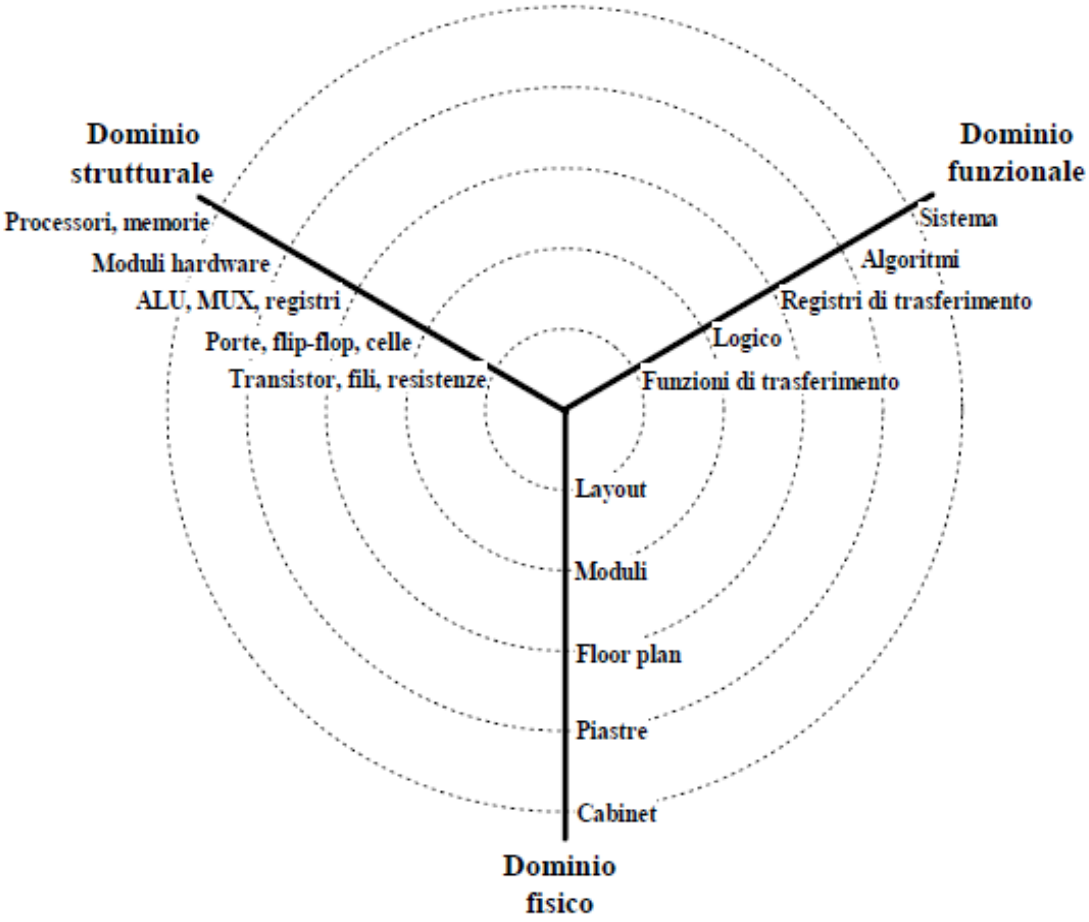
   l'insieme delle porte logiche elementari e le connessioni tra esse capaci di soddisfare il vincolo imposto dal modulo

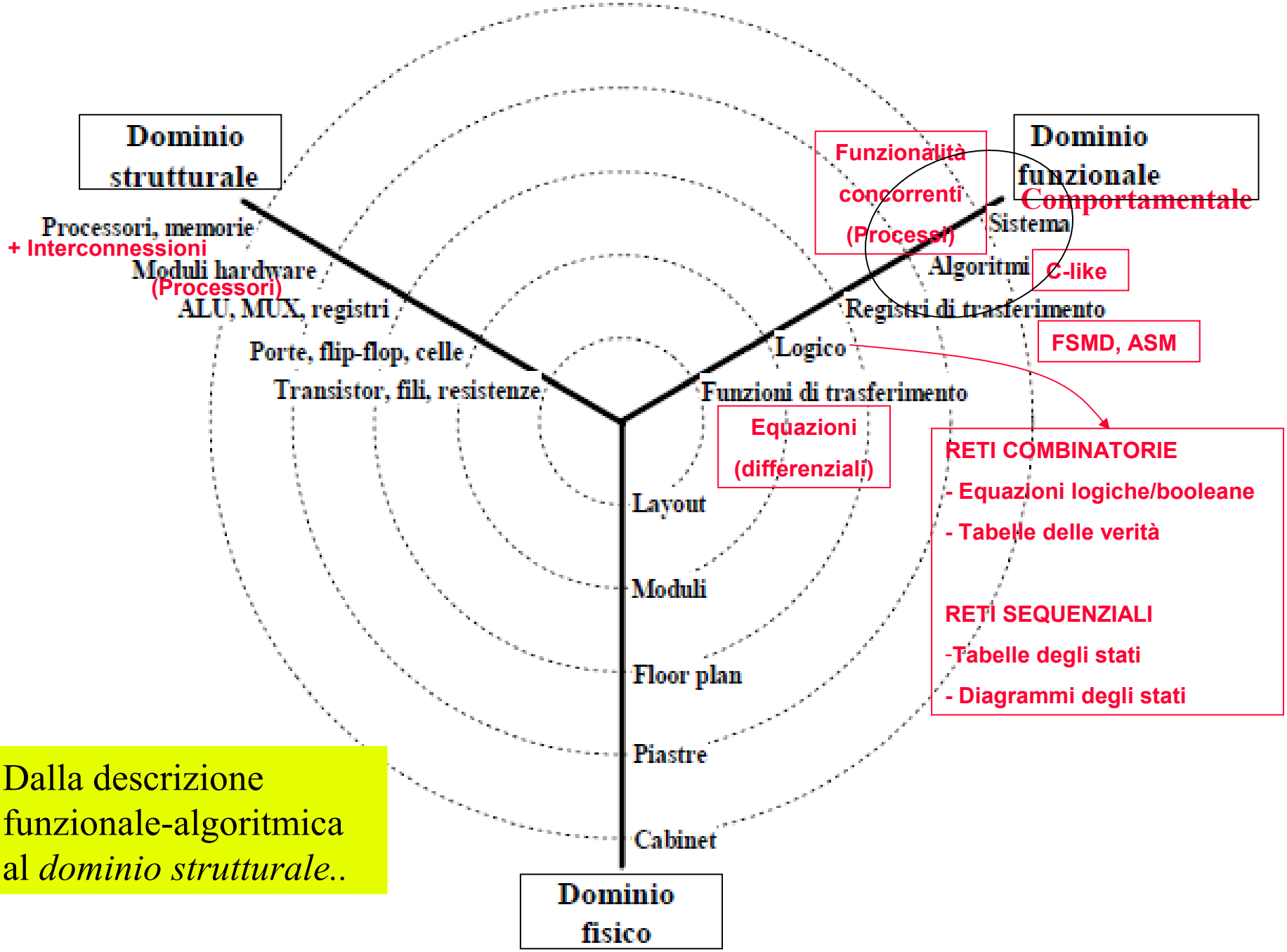
3. lo ***STRUTTURALE*** REALIZZATIVO (***fisico***)

   organizzato in sottolivelli, in base alla tecnica realizzativa scelta e alla tecnologia (CI , transistors ecc……)

Considerazioni

- Diagramma a Y





Dalla descrizione funzionale-algoritmica al *dominio strutturale*..

I modelli *1*

Riassumendo:

- In un **sistema digitale complesso**, in generale,

  l'uscita ad certo istante di tempo t_x dipende **sia** dagli ***ingressi*** presenti in quel ***medesimo istante di tempo*** che dalla **successione** degli ***ingressi nei tempi antecedenti*** a quello considerato. **È un sistema *dinamico***

  E' "algoritmico" ….

I ***sistemi digitali*** (generali comunque complessi sono ***SISTEMI SEQUENZIALI***

Modello di riferimento per la sua descrizione

AUTOMA a STATI FINITI

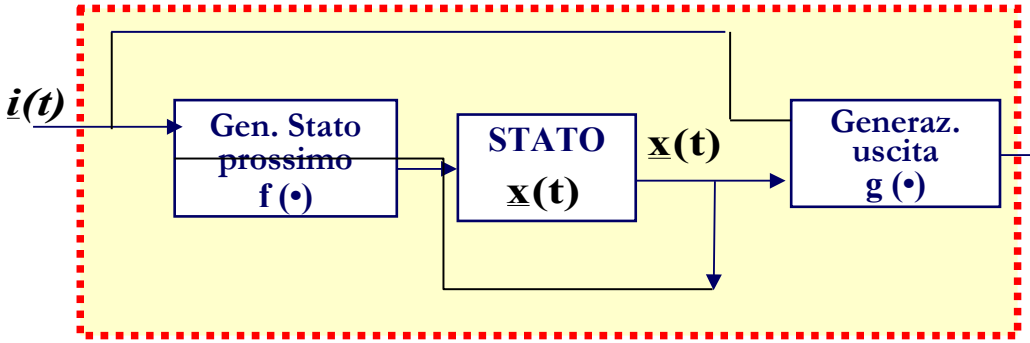
MACCHINA A STATI FINITI

- E' un *modello* che, recependo la teoria dei **sistemi dinamici**, descrive l'evoluzione temporale e funzionale del *sistema sequenziale*
- E' detto **modello** *MACCHINA A STATI FINITI o AUTOMA a STATI FINITI*
- *La sua scrittura esplicita puo' essere:*
  - ***direttamente tradotto in HW;***
  - *Ammettere impl. Sw su gpphw*

$$\begin{cases} \underline{x}(t+1) = f(\underline{i}(t), \underline{x}(t)) \\ \underline{y}(t) = g(\underline{i}_{\underline{y}}(t), \underline{x}(t)) \end{cases}$$

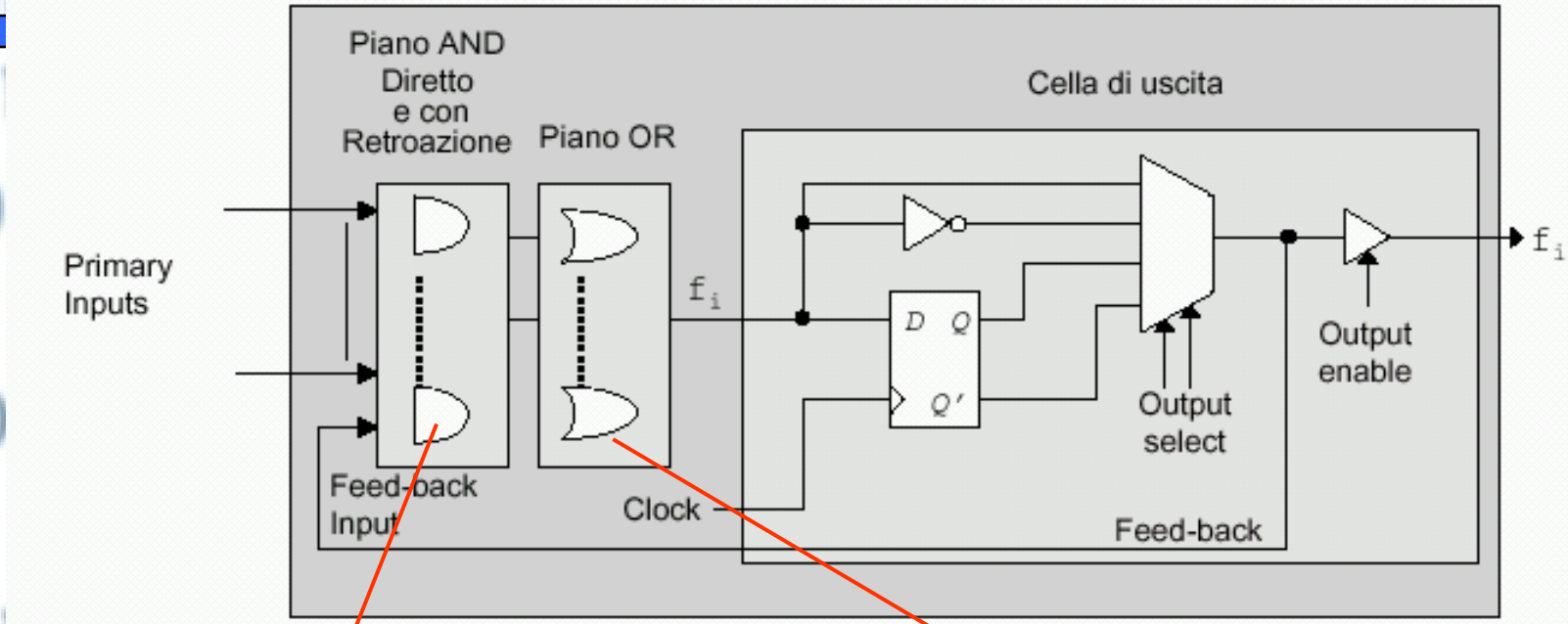
Eq. implicite generali

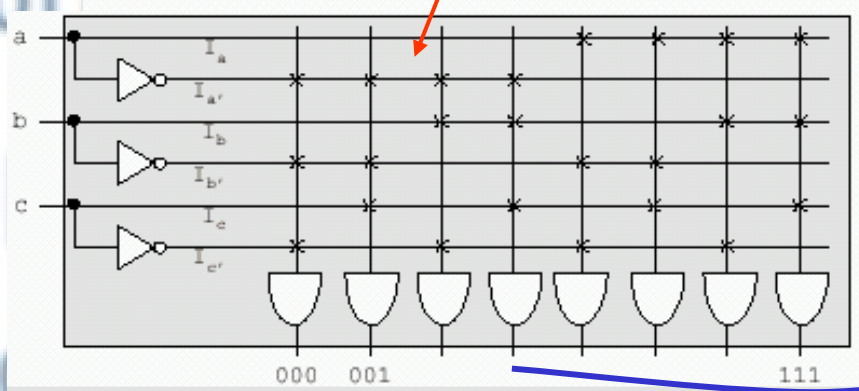
Tutte le logiche CONFIGURABILI implementano in HW questo modello

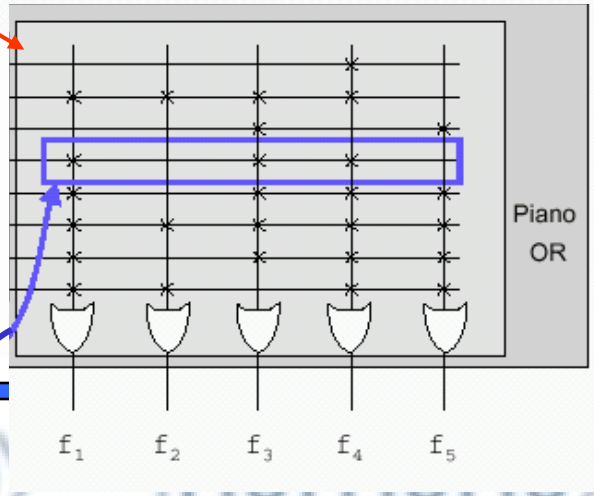


Modello di MEALY

Esempi di realizzazione (*PLD*)





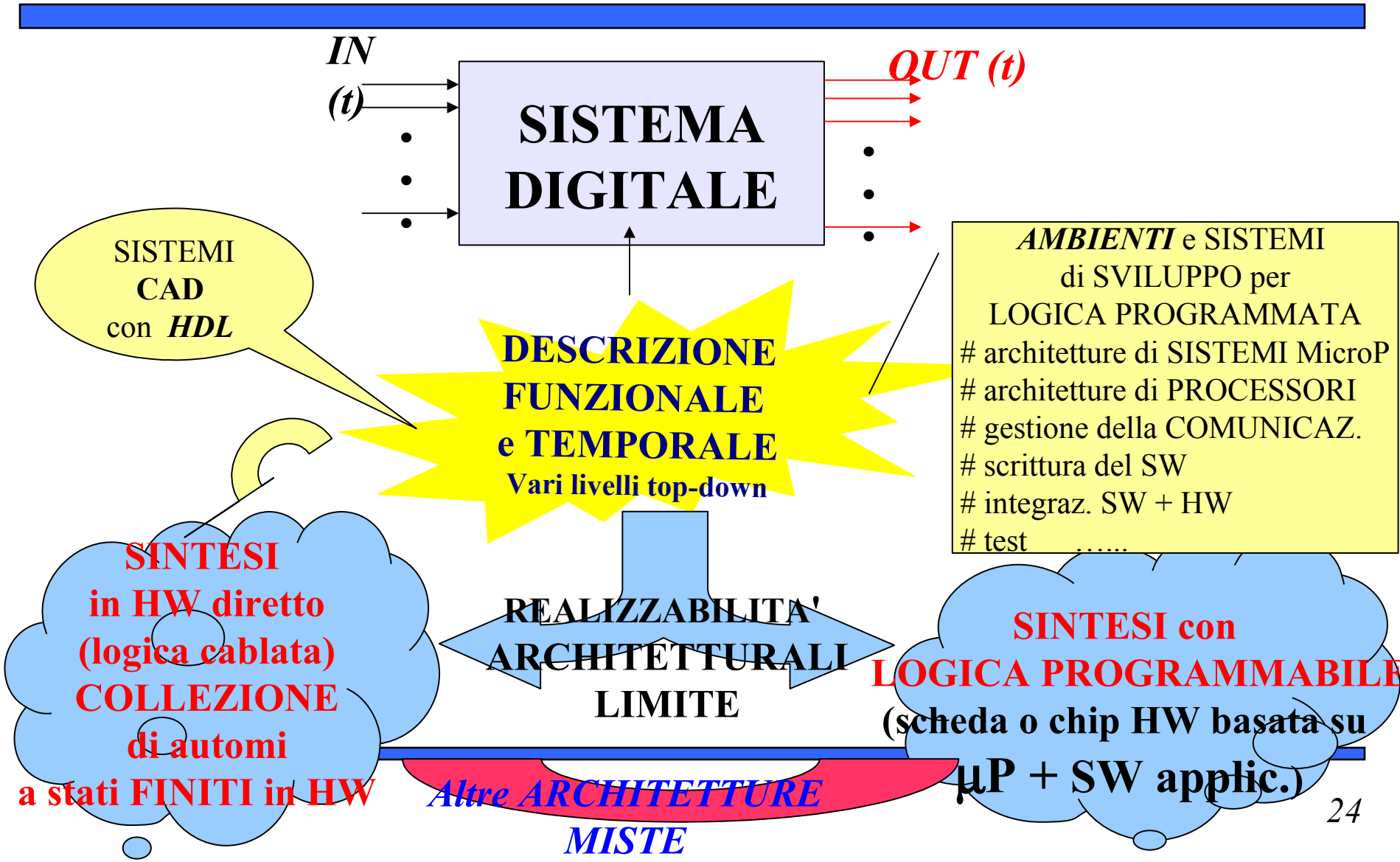


Significato: ogni crocetta indica quali variabili sono coinvolte, il tipo di porta (es: AND) indica schematicamente come tali variabili collegate tra loro.

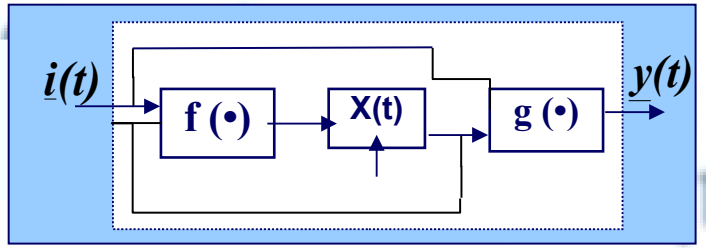
Quali elementi condizionano la *realizzabilità* del sistema fisico ....

- Il quadro delle possibilità architetturali-tecnologiche circa la realizzabilità (anche queste strutturate, partendo dall'alto) sono:
  - ***Logica CABLATA*** (o meglio ***ad HW diretto***) con adeguati *tool* di sviluppo e gestione basati su linguaggi HDL (Hardware Description Linguage)
  - ***Logica PROGRAMMATA*** (basata sull'utilizzo di un HW standard (microprocessori) + SW appl. finalizzati all'applicazione tecnica)
- Dovrebbe essere chiaro come la prima discriminante sia proprio ***la complessità HW :*** un semplice *microP* di per se è costituito da diverse ***decine di migliaia*** di porte logiche, a parità di grado di integrazione, non avrebbe gran senso utilizzare un *microP* per realizzare una funzione digitale che richiede 50 o 500-1000 porte logiche !!??....

PUNTO SOSTANZIALE

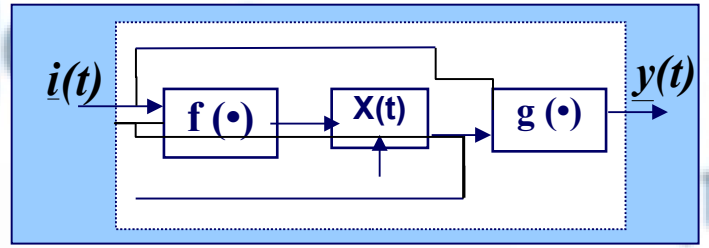


Classi e sottoclassi



- nel modello generale è possibile introdurre delle **partizioni**. si generano delle **sottoclassi** che, pur mancando di completa generalità, trovano **moltissime applicazioni**.
- esse sono:
  - classe 0 - macchina combinatoria
  - classe 1 - macchina semplicemente ritardata
  - classe 2 - macchina senza ingresso (evoluzione libera)
  - classe 3 - macchina con uscita sullo stato
  - **classe 4 - macchina a stati modello di MOORE**
  - classe 5 - macchina a stati completa (modello MEALY)

CLASSE 0



MACCHINA **COMPLETAMENTE ISTANTANEA**

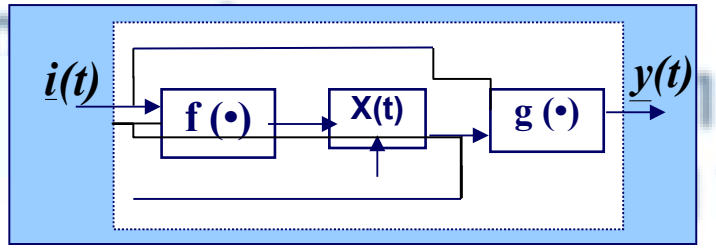
SISTEMA COMBINATORIO

$$\underline{i}(t) \Rightarrow \boxed{f(\bullet)} \Rightarrow \underline{y}(t) = g(\underline{i}(t))$$

manca la FUNZIONE DI STATO

l'uscita dipende esclusivamente dagli ingressi presenti in quel momento (a meno di ritardi propri della rete)

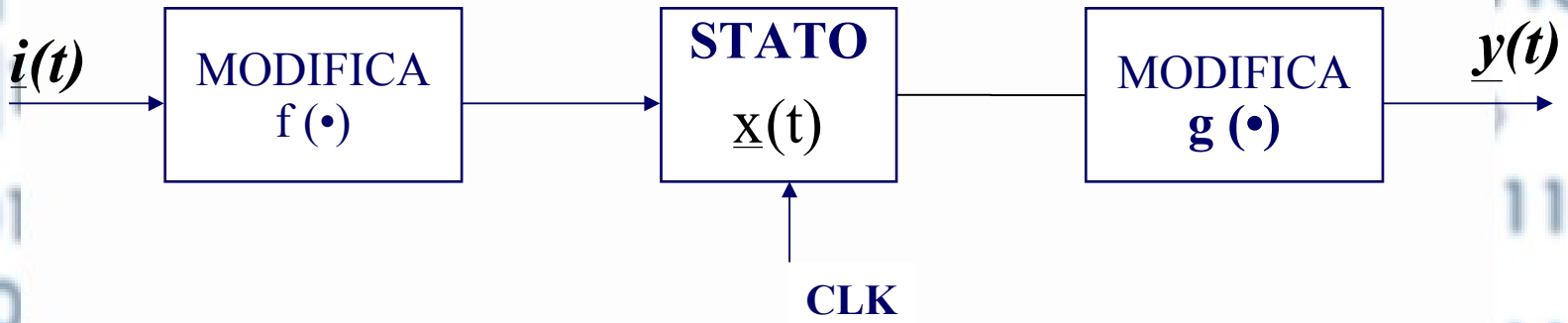
CLASSE 1



Mancano:

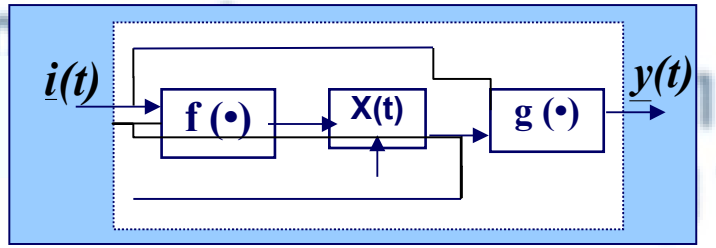
- **LA REAZIONE dello STATO** E LA **PARTE ISTANTANEA**

$$\underline{y}(t) = g[f(\underline{i}(t-1))]$$



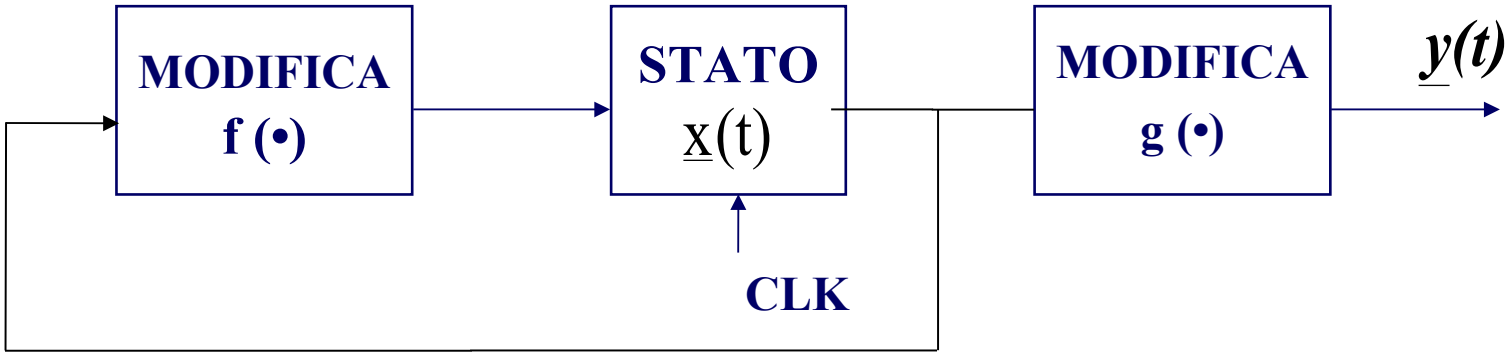
la rete si presenta come un *ritardo temporale finito* tra ingresso e uscita, tale ritardo è stabilito dal tempo clock (es. modulo di filtro digitale FIR)

CLASSE 2



MANCA L'INGRESSO

lo stato prossimo è stabilito, tramite la f (•) esclusivamente dallo stato ATTUALE.

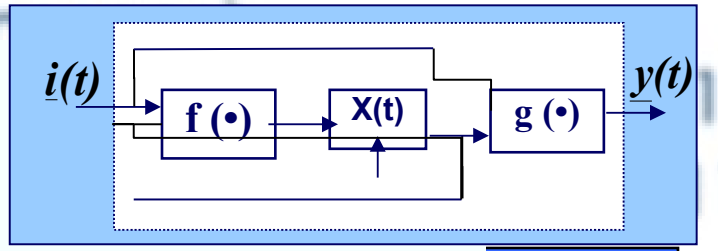


ES: SEQUENZIATORI
CONTATORI di stati

$$\begin{cases} \underline{x}(t+1) = f(\underline{x}(t)) \\ \underline{y}(t) = g(\underline{x}(t)) \end{cases}$$

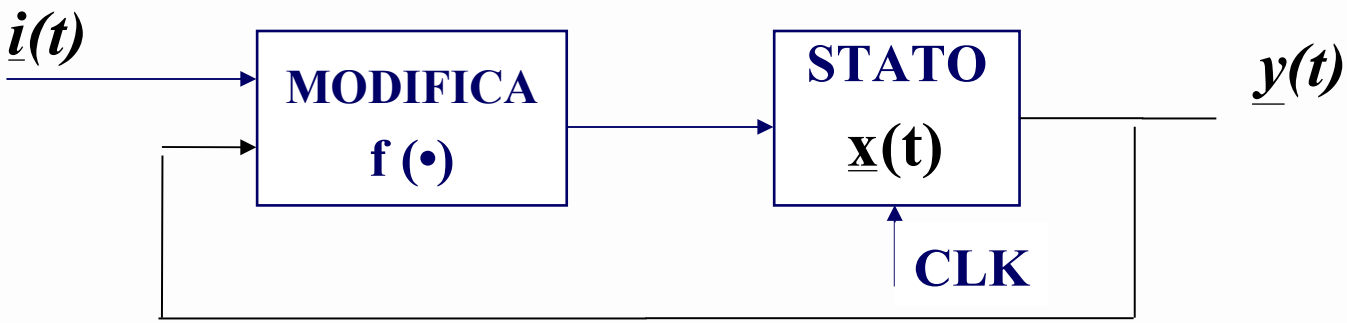
Classe 3

uscita diretta nello stato



MANCA LA **PARTE ISTANTANEA** SULL'USCITA.

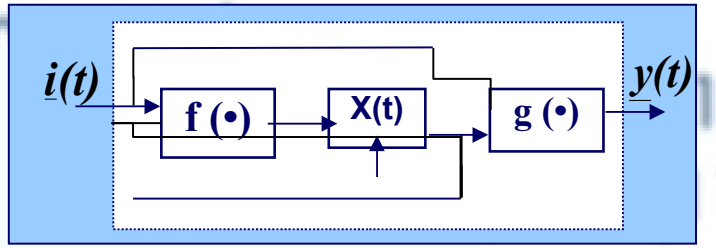
l'uscita è' completamente determinata dallo *stato*.



$$\begin{cases} \underline{x}(t+1) = f\left(\underline{i}(t), \underline{x}(t)\right) \\ y(t) = \underline{x}(t) \end{cases}$$

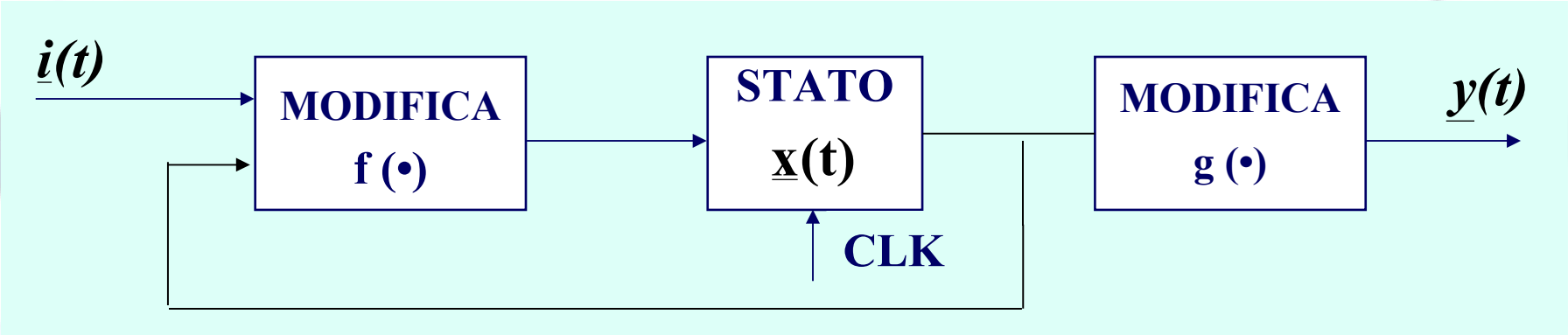
Classe 4

(Il nostro AUTOMA)



MANCA LA **PARTE ISTANTANEA** SULL'USCITA.

L'USCITA È' DETERMINATA solo dallo STATO.

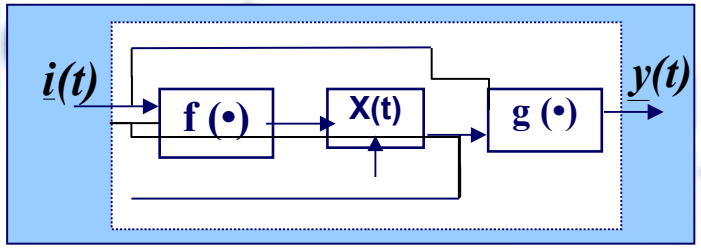


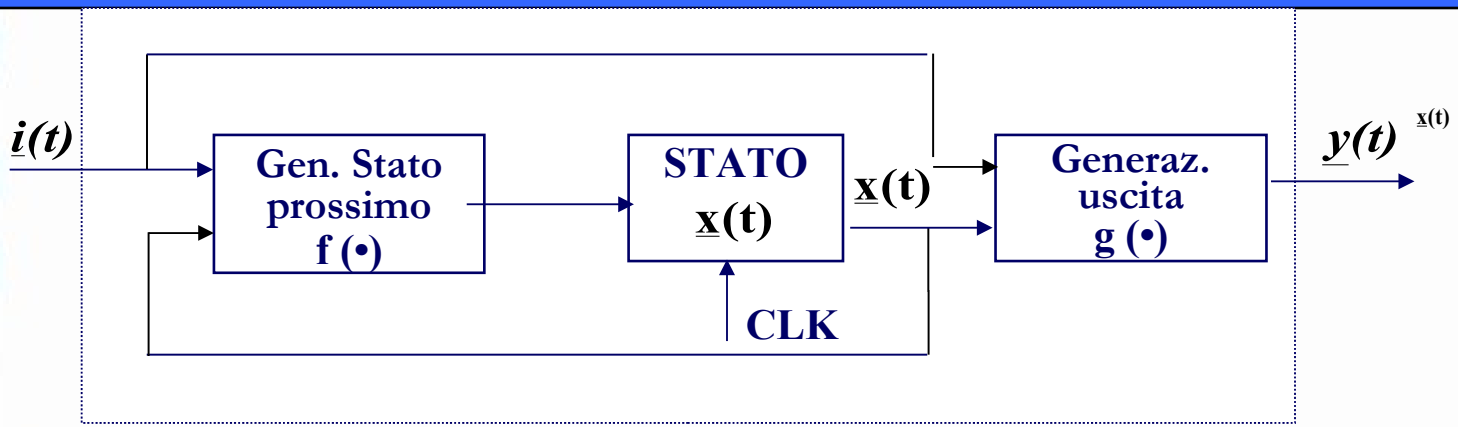
MODELLO di MOORE

$$\begin{cases} \underline{x}(t+1) = f(\underline{i}(t), \underline{x}(t)) \\ \underline{y}(t) = g(\underline{x}(t)) \end{cases}$$

Rappresenta per noi il **modello base**, il piu' utilizzato. (***Sistemi artificiali***)

CLASSE 5





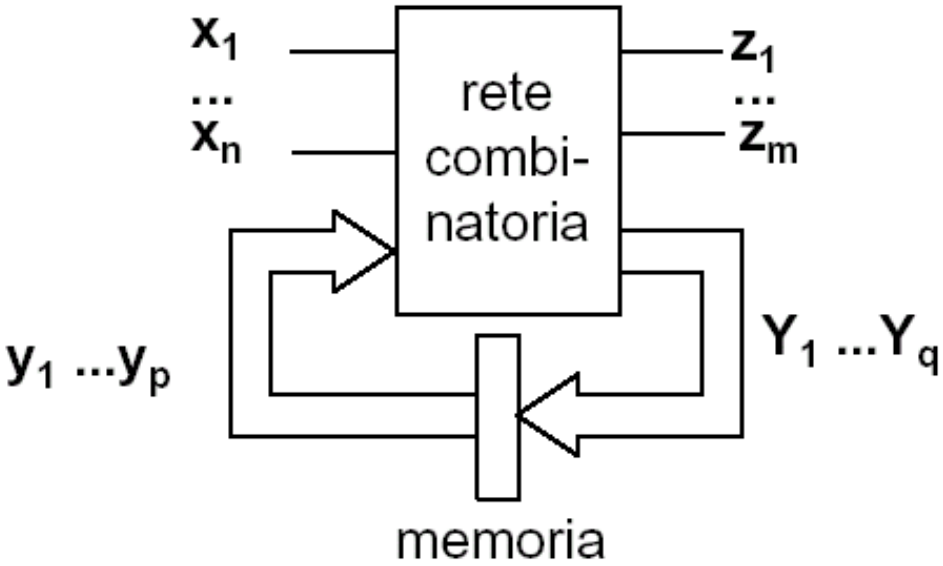
- E' il modello di MEALY
- E' "teoricamente" il modello di macchina piu' completo, utile nei processi di "identificazione"
- in generale ***nelle realizzazione pratiche*** (lato *design)* si preferisce evitare *la parte istantanea* (si usa il modello di MOORE)

Osservazione ….

- E' possibile dimostrare (vedi teoria dei sistemi) come, assegnato un modello di Mealy, si possa sempre definire una *trasformazione congrua* che generi un *EQUIVALENTE* sistema con ***modello di MOORE***.
- I due modelli sono equivalenti,
- Nell'approccio progettuale si preferisce il modello di Moore, perchè intrinsecamente più ***TESTABILE***

variante

- In letteratura a volte si fa riferimento, come modello generale, anche al modello di *HUFFMAN*



- Con alcune trasformazione conformi e partizioni semplici si puo' immediatamente dimostrare che esso coincide esattamente con **i modelli di Mealy o di MOORE** (la memoria coincide con lo stato)

Modello di Huffman

Sintesi finale *il nostro obiettivo*

- una **<u>macchina a stati finiti</u>** (MSF) è composta quindi
  - da blocchi **<u>puramente combinatori</u>** *(insieme connesso di porte NOT, AND, OR) e*
  - dalla **<u>funzione di stato</u>** opportunamente reazionata.
    - la funzione di stato è "*semplice*" ed è formata da un **<u>ritardo temporale</u>** (**FF di tipo D** o collezione di FF D – *registro*)
    - Spesso si parla di ***REGISTRO di STATO***
- riferimento pratico: ***il modello di MOORE***
- **ogni modulo *MSF* e' traducibile direttamente in HW** (vedi le realizzazioni)
- **Un *sistema digitale* può essere scomposto in più *AUTOMI***
- **La composizione (serie e/o parallelo) di MSF è ancora una MSF**

Metodi di descrizione

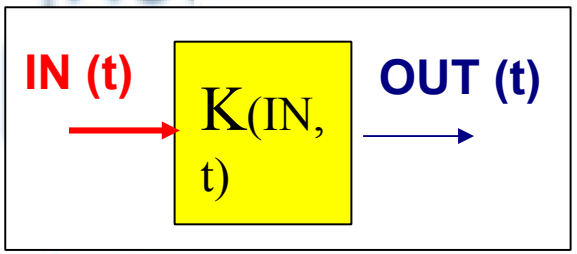
- Gli Automi o Macchine a Stati Finiti **consistono in una collezione di azioni** che devono essere attivate e realizzate nei ***tempi propri dell'evoluzione dinamica*** del sistema
- Si necessita quindi di
  - **Metodi Formali** per descrivere in *modo univoco* le azioni da realizzare ….
  - **tool** che ci consentano di simulare e verificare la correttezza logica e temporale di quanto descritto;
  - **tool** che ci aiutino a *SCRIVERE* i circuiti fisici.

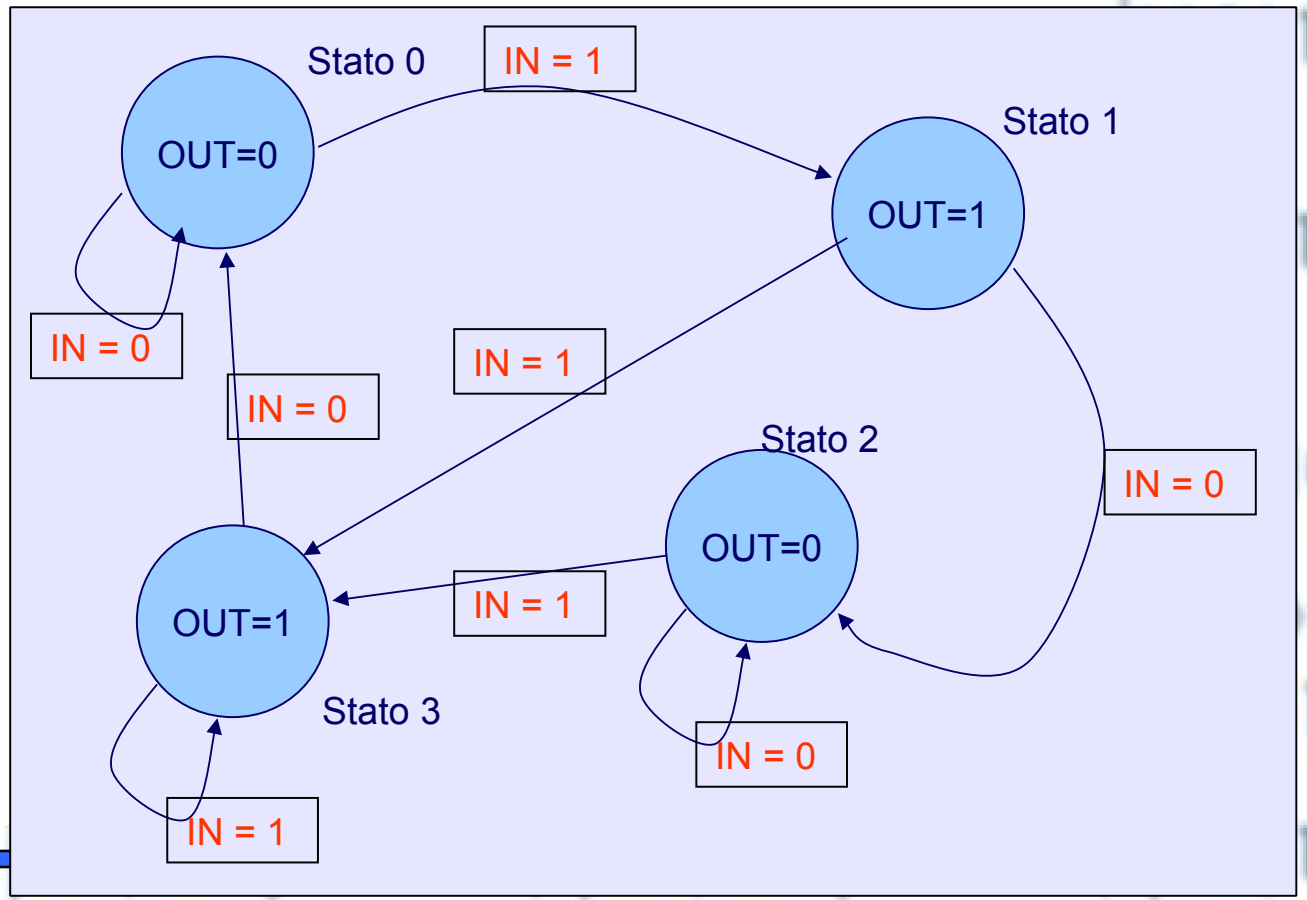
Metodi Formali

- approccio iniziale (comprendere le funzioni base):
  - diagrammi *stato/transizione*
  - ASM (ok)
- approccio ***complesso***
  - VHDL
  - verilog
  - system C
  - e altri (es. *ULM*)

Esempi base: diagrammi stato-transizione

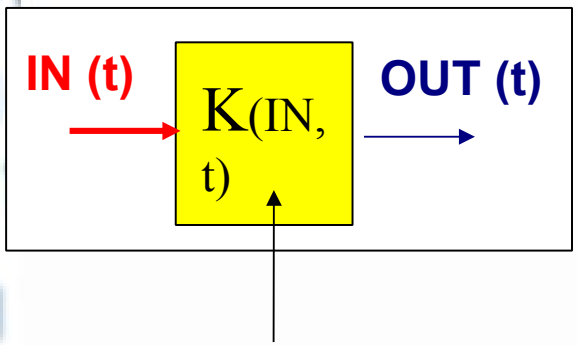
Descrizione funzionale con <diagrammi stato/transizione>

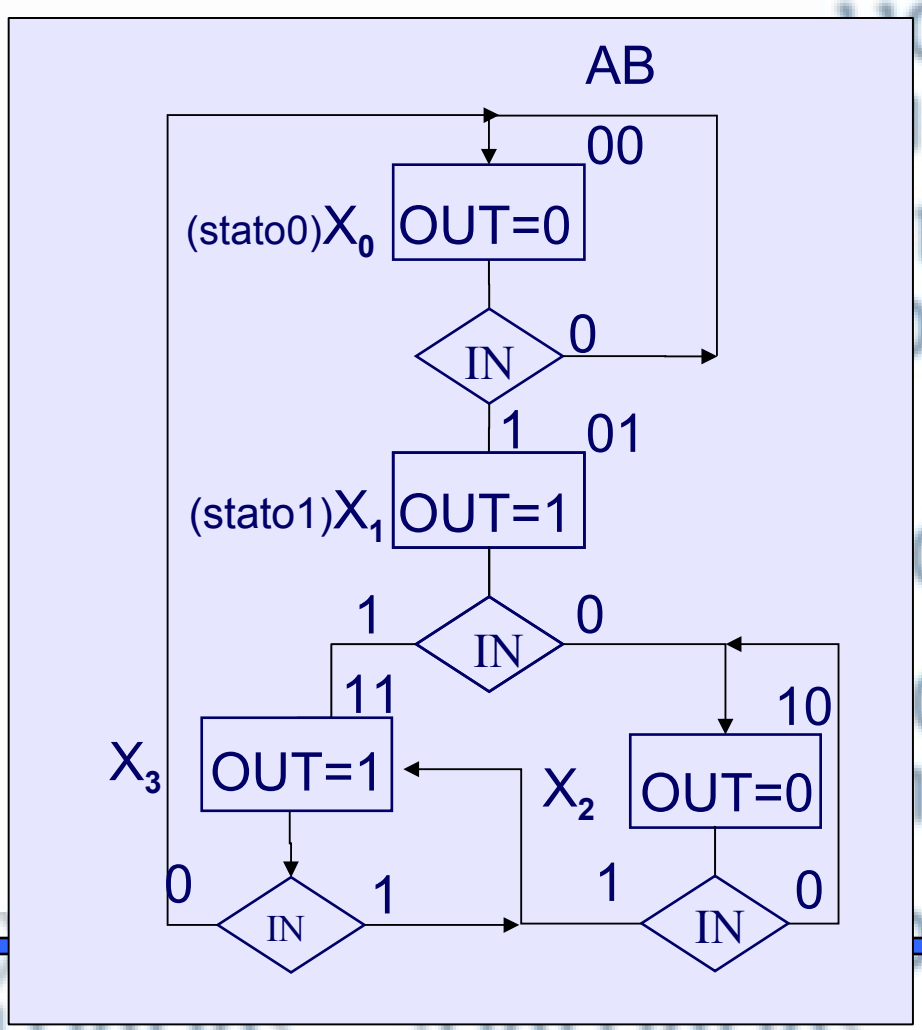




Esempi base: ASM

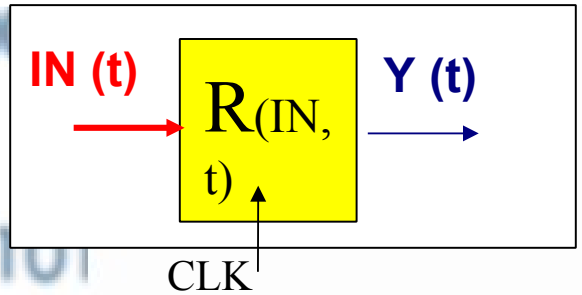
Descrizione funzionale con <Algorithm State Machine >





A.S.M. esercizio S1.

PROBLEMA: considerato un ingresso seriale a un bit (IN) sincrono con il CLK di sistema, si fornisca in uscita un segnale che viene posto ad 1 quando vengono ricevuti tre 1 consecutivi, 0 altrimenti.

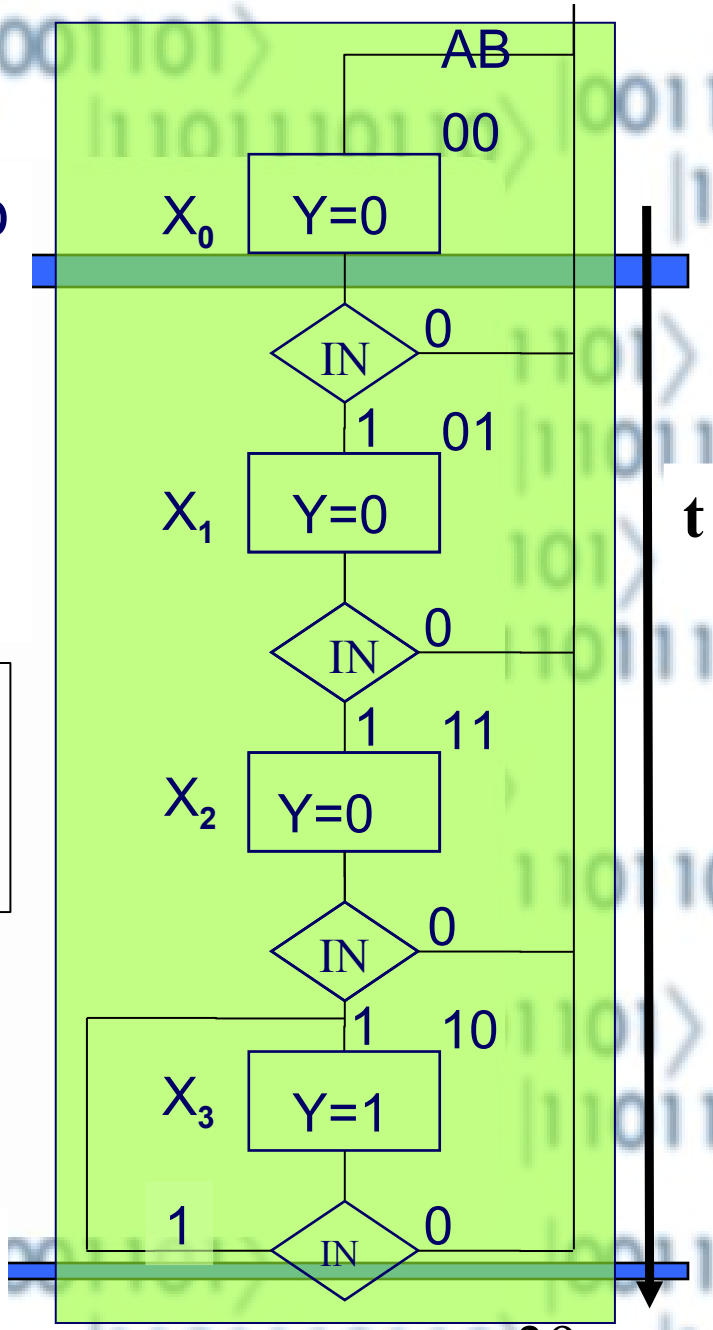


SIST.*SEQUEZ*

↓

Mod.*MOORE*

Essendo sincrono CLK e bit ingresso (IN), si può ad ogni tempo di stato acquisire (leggere) un bit



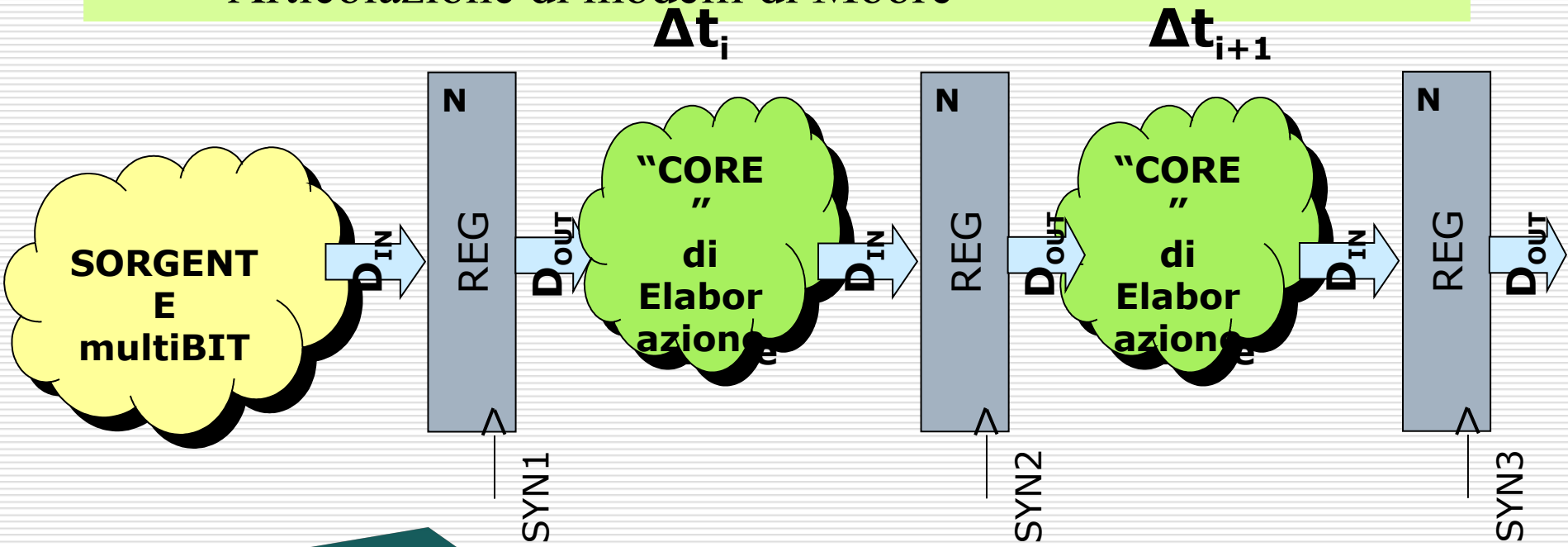
- L'A.S.M. presenta 4 stati sono quindi necessari $\log_2(N)$ variabili di stato cioè 2 {A,B}.

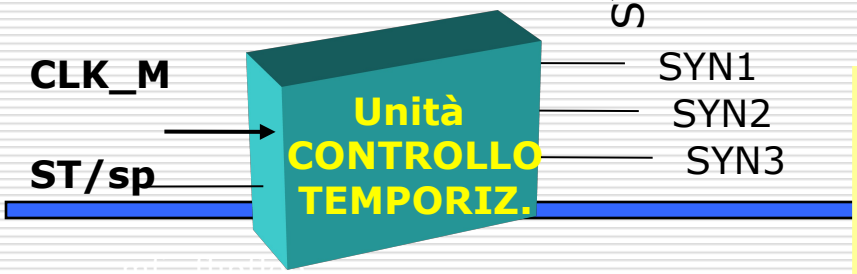
Es. generico di formalizzazione in VHDL

```
--
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all ;
--
--
entity decoder is
    generic(
            c_encoded_dimension : integer := 4;
            c_decoded_dimension : integer := 16
           );
    port(
        data_encoded_i  : in  std_logic_vector((c_encoded_dimension-1) downto 0);
        data_decoded_o  : out std_logic_vector((c_decoded_dimension-1) downto 0)
        );
end decoder;
--
--
architecture rtl of decoder is
--
--
begin
--
--
modular_encoder : process (data_encoded_i)
begin
   for i in (c_decoded_dimension-1) downto 0 loop
      if (i = (conv_integer(data_encoded_i))) then
         data_decoded_o(i) <= '1';
      else
         data_decoded_o(i) <= '0';
      end if;
   end loop;
end process;
--
--
end rtl;
--
--
```

Estensione dei modelli hw processamento *1*

- Register Transfer model
  - Articolazione di modelli di Moore





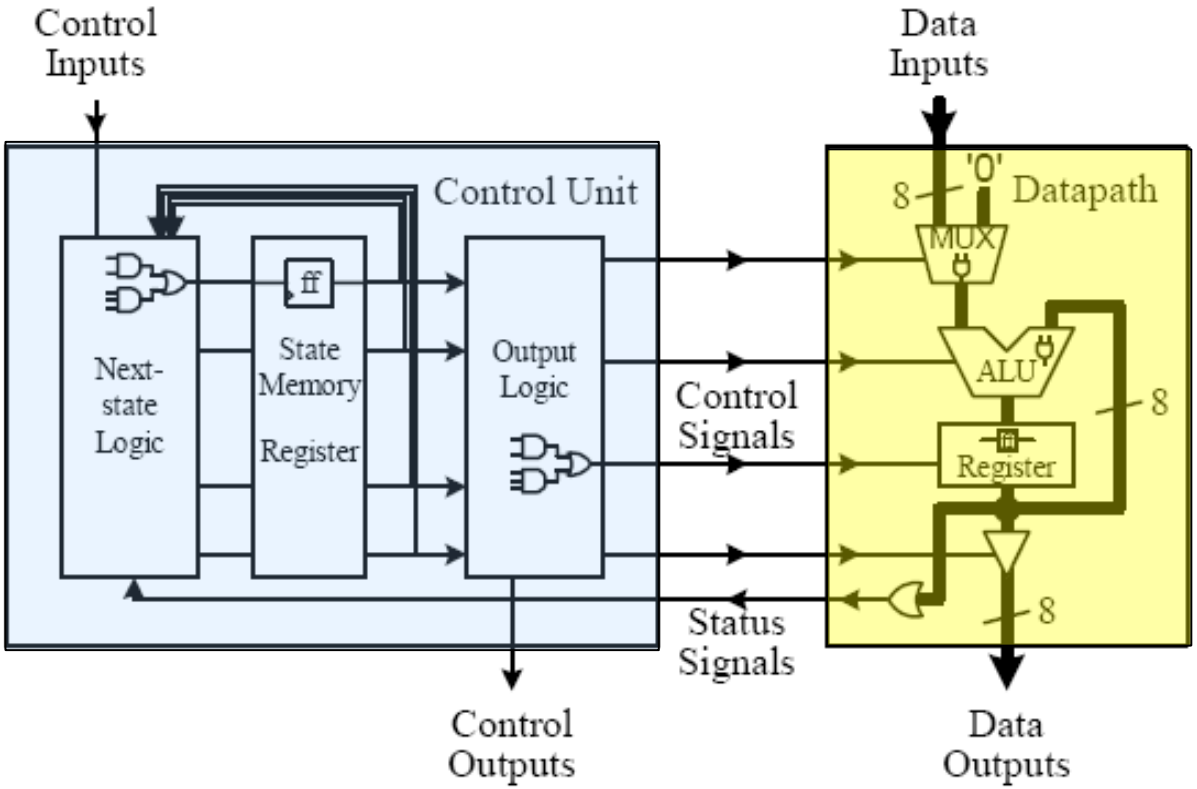
DIAGRAMMI temporali per la SINCRONIZZAZIONE dei REGISTRI

Estensione dei modelli *hw* processamento *2*

Trasformazione in: Modello

DATA-path +

CONTROL-path ***(****moore stretto, possibile il partizionamento* ***)***



Visione di principio di un *GPP*…..

- Riferimento a ***datapath*** + ***UC*** (*controlpath* partizionato

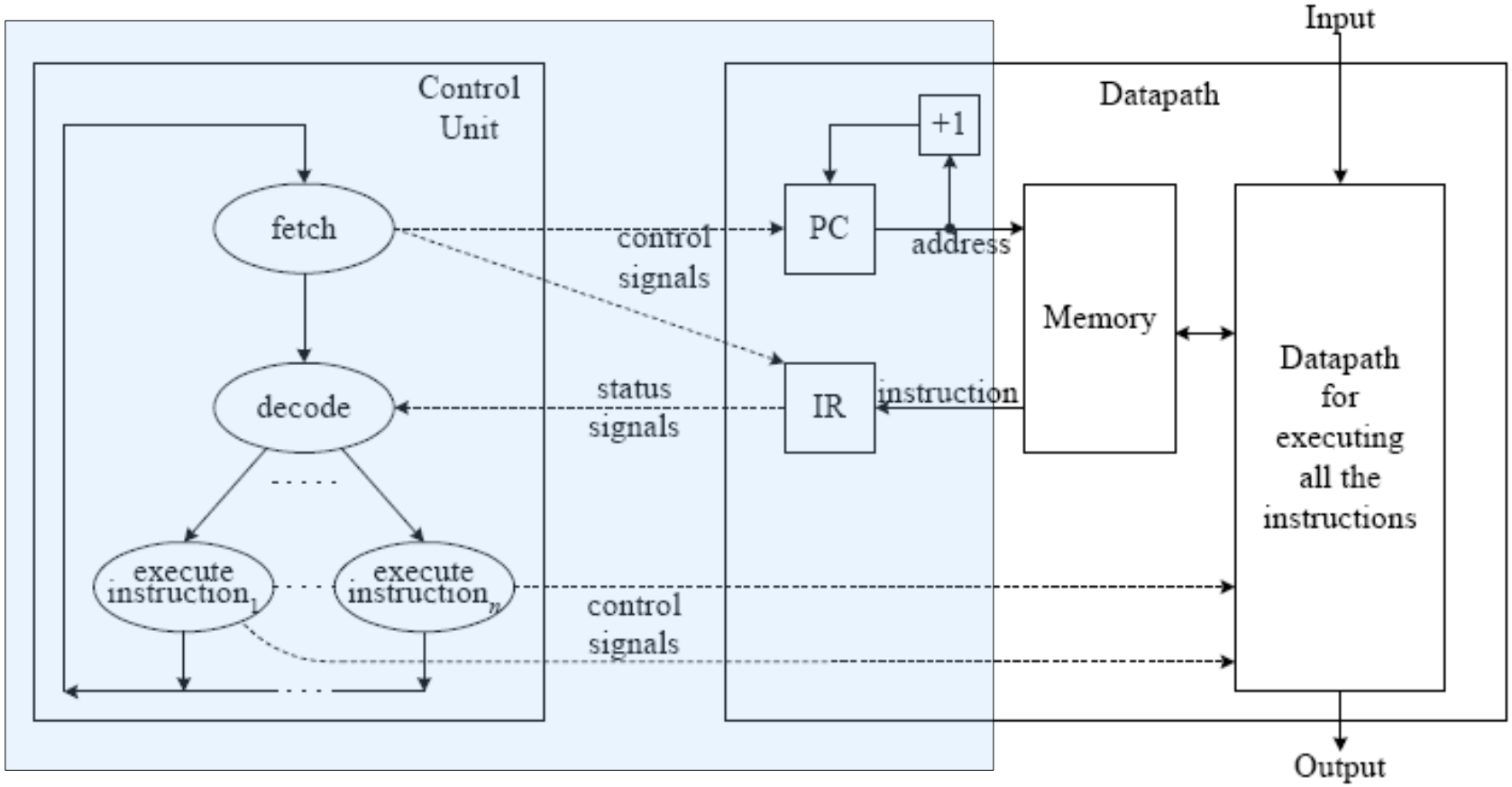


Figure 12.1. Overview of a general-purpose microprocessor.

Considerazione finale ...

L'implementazione, globale oppure in singoli moduli, di un modello come quello di pag.42 porta alla sintesi di ***SPP***.

Considerato il modello generale di pag.42, è sempre possibile ri-descrivere internamente UC con l'approccio ***fetch/decode/execute + mem*** (pag.43).

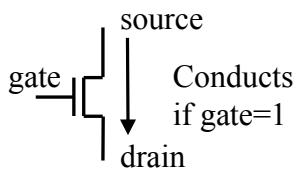
In questo caso si può qualificare il sistema sia un ***GPP*** ma anche, configurando un *set-istruz ad hoc* per *applicazione specifica,* come un ***ASIP***.

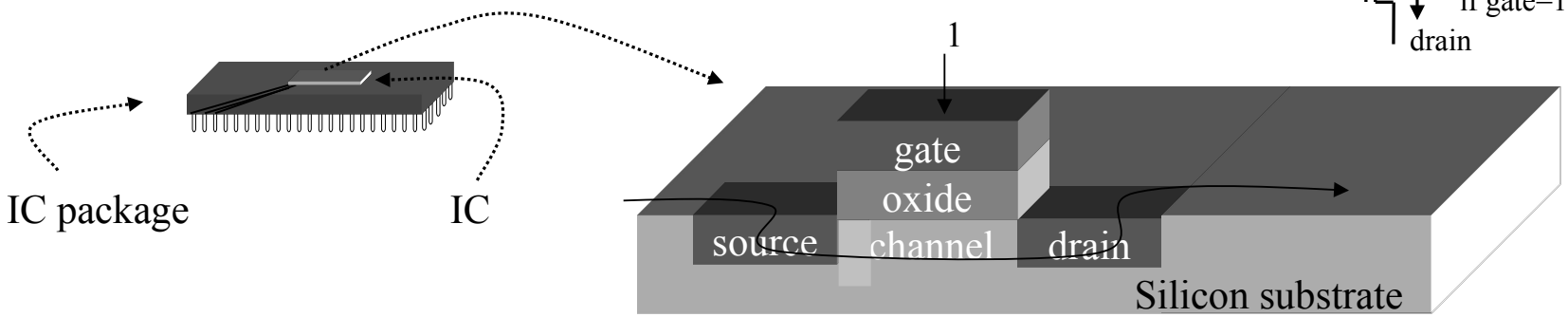
Esempi - esercizi

- ***Mediatore***
- *Prodotto di matrice (riga – colonna)*
- Implementazione di filtro FIR …
- ……..

CMOS transistor on silicon

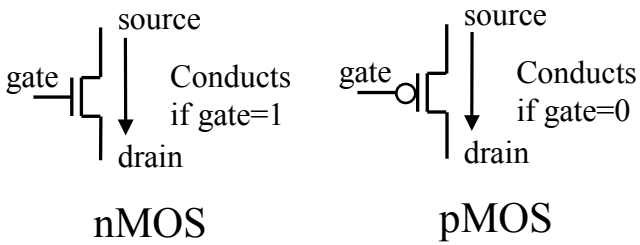
- Transistor
  - The basic electrical component in digital systems
  - Acts as an on/off switch
  - Voltage at “gate” controls whether current flows from source to drain
  - Don’t confuse this “gate” with a logic gate

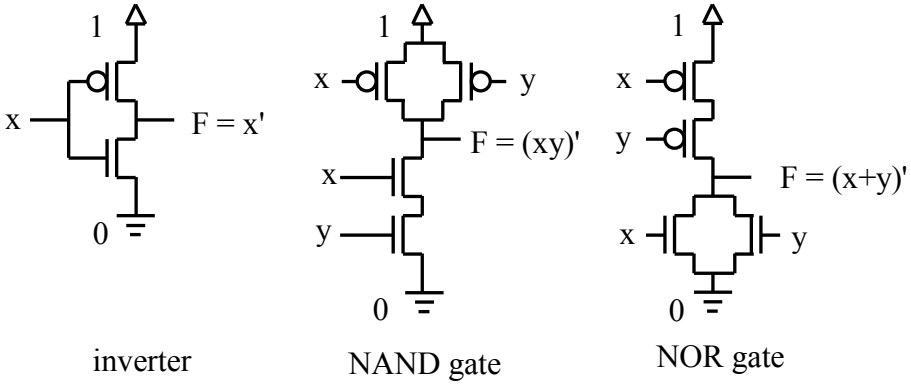




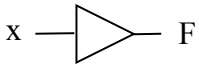
CMOS transistor implementations

- Complementary Metal Oxide Semiconductor
- We refer to logic levels
  - Typically 0 is 0V, 1 is 5V
- Two basic CMOS types
  - nMOS conducts if gate=1
  - pMOS conducts if gate=0
  - Hence "complementary"
- Basic gates
  - Inverter, NAND, NOR



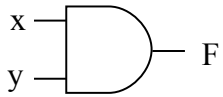


Basic logic gates



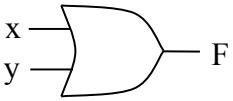
x	F
0	0
1	1

F = x
Driver



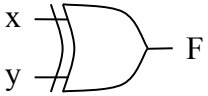
x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

F = x y
AND



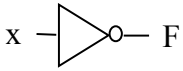
x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

F = x + y
OR



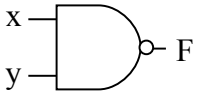
x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

F = x ⊕ y
XOR



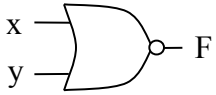
x	F
0	1
1	0

F = x’
Inverter



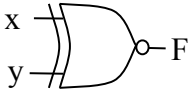
x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

F = (x y)’
NAND



x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

F = (x+y)’
NOR



x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

F = x ⊙ y
XNOR

Combinational logic design

A) Problem description

y is 1 if a is to 1, or b and c are 1. z is 1 if b or c is to 1, but not both, or if all are 1.

B) Truth table

Inputs			Outputs	
a	b	c	y	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

C) Output equations

y = a'bc + ab'c' + ab'c + abc' + abc

z = a'b'c + a'bc' + ab'c + abc' + abc

D) Minimized output equations

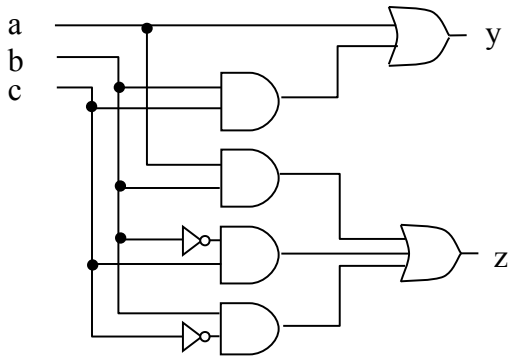
y: a \ bc	00	01	11	10
0	0	0	1	0
1	1	1	1	1

y = a + bc

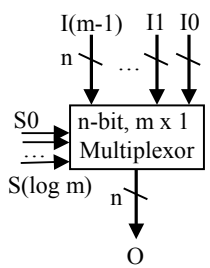
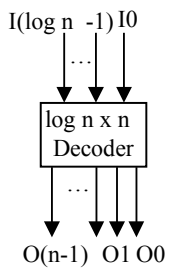
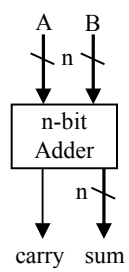
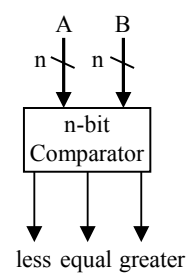
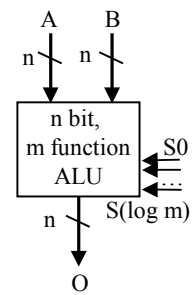
z: a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

z = ab + b'c + bc'

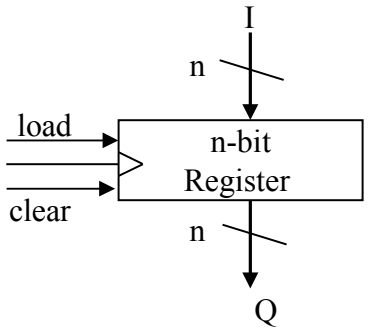
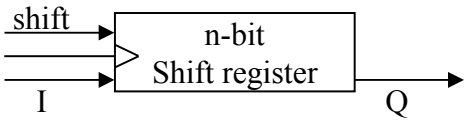
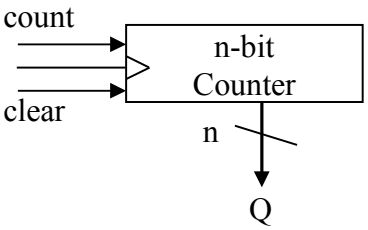
E) Logic Gates



(RT-Level) Combinational components

				
O = I0 if S=0..00 I1 if S=0..01 … I(m-1) if S=1..11	O0 =1 if I=0..00 O1 =1 if I=0..01 … O(n-1) =1 if I=1..11	sum = A+B (first n bits) carry = (n+1)'th bit of A+B	less = 1 if A<B equal =1 if A=B greater=1 if A>B	O = A *op* B *op* determined by S.
	With enable input e → all O's are 0 if e=0	With carry-in input Ci→ sum = A + B + Ci		May have status outputs carry, zero, etc.

(RT-Level) Sequential components

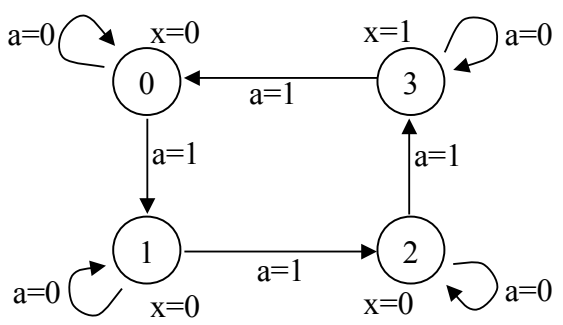
		
Q = 0 if clear=1, I if load=1 and clock=1, Q(previous) otherwise.	Q = lsb - Content shifted - I stored in msb	Q = 0 if clear=1, Q(prev)+1 if count=1 and clock=1.

Sequential logic design

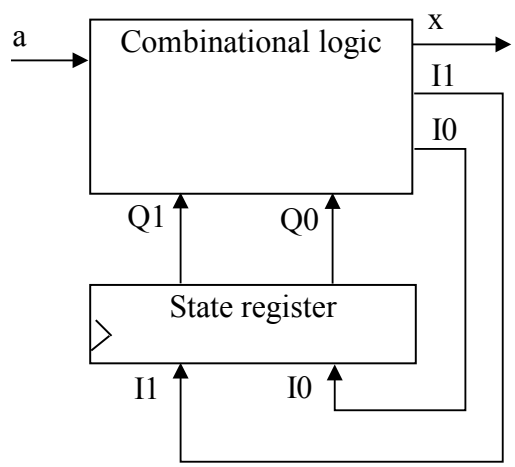
A) Problem Description

You want to construct a clock divider. Slow down your pre-existing clock so that you output a 1 for every four clock cycles

B) State Diagram



C) Implementation Model



D) State Table (Moore-type)

Inputs			Outputs		
Q1	Q0	a	I1	I0	x
0	0	0	0	0	0
0	0	1	0	1	
0	1	0	0	1	0
0	1	1	1	0	
1	0	0	1	0	0
1	0	1	1	1	
1	1	0	1	1	1
1	1	1	0	0	

- Given this implementation model
  - Sequential logic design quickly reduces to combinational logic design

Sequential logic design (cont.)

E) Minimized Output Equations

I1 Q1Q0

a \ Q1Q0	00	01	11	10
0	0	0	1	1
1	0	1	0	1

I1 = Q1'Q0a + Q1a' + Q1Q0'

I0 Q1Q0

a \ Q1Q0	00	01	11	10
0	0	1	1	0
1	1	0	0	1

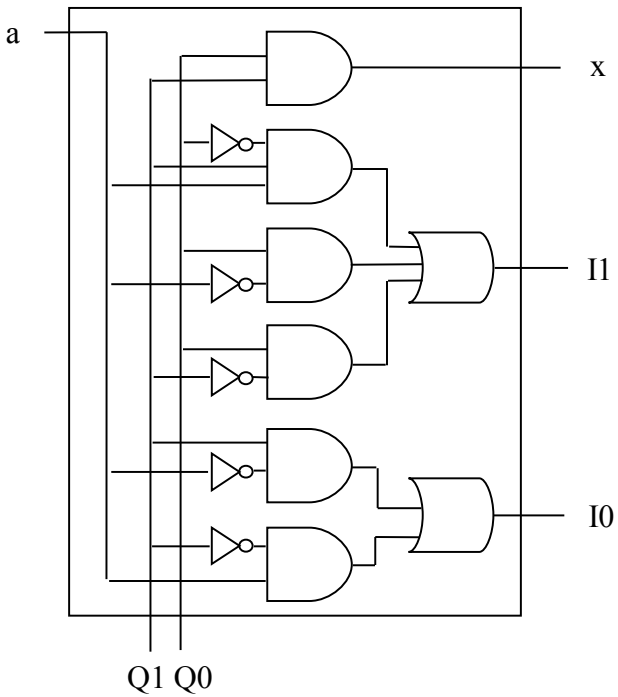
I0 = Q0a' + Q0'a

x Q1Q0

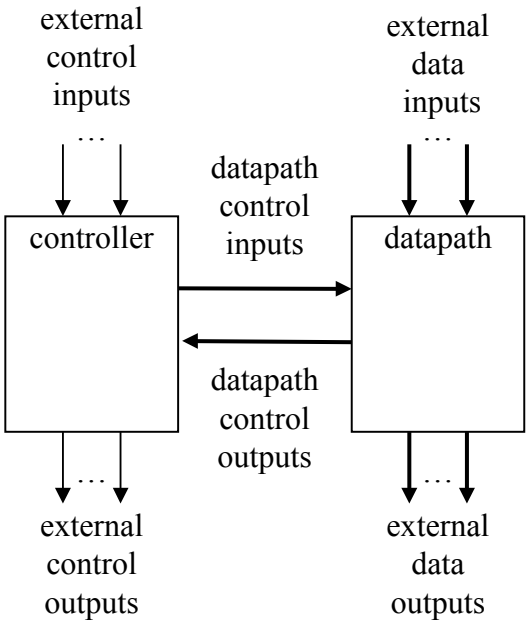
a \ Q1Q0	00	01	11	10
0	0	0	1	0
1	0	0	1	0

x = Q1Q0

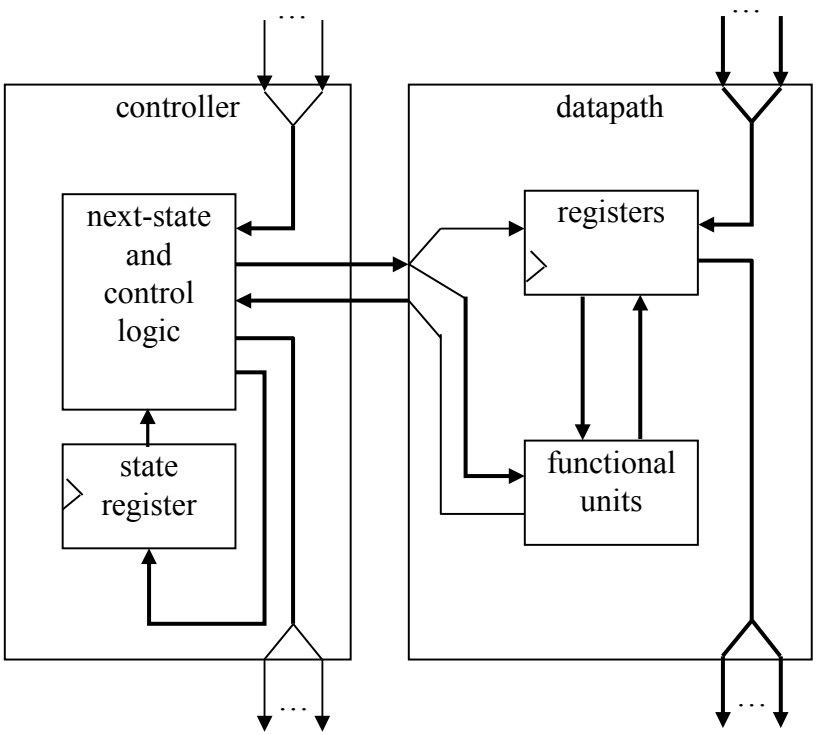
F) Combinational Logic



Custom single-purpose processor basic model



controller and datapath

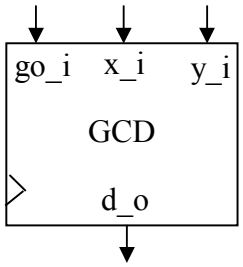


a view inside the controller and datapath

Example: greatest common divisor

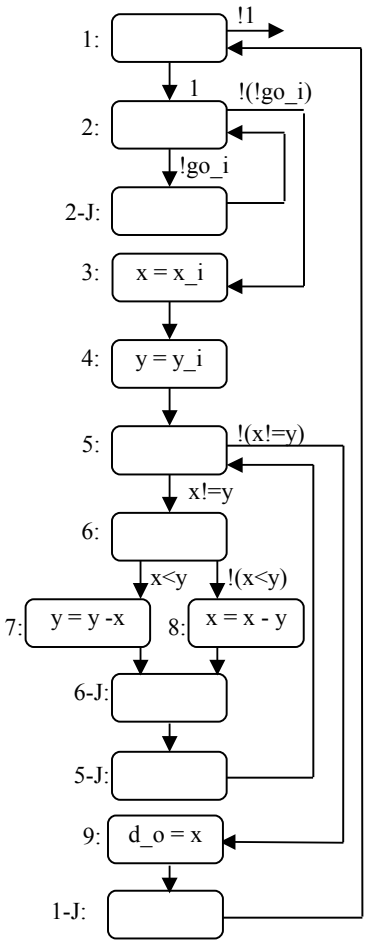
- First create algorithm
- Convert algorithm to "complex" state machine
  - Known as FSMD: finite-state machine with data
  - Can use templates to perform such conversion

(a) black-box view



(b) desired functionality

```
0: int  x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while  (x != y)  {
6:      if  (x < y)
7:         y = y - x;
        else
8:         x = x - y;
     }
9:   d_o = x;
   }
```

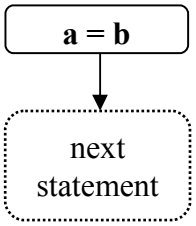


(c) state diagram

State diagram templates

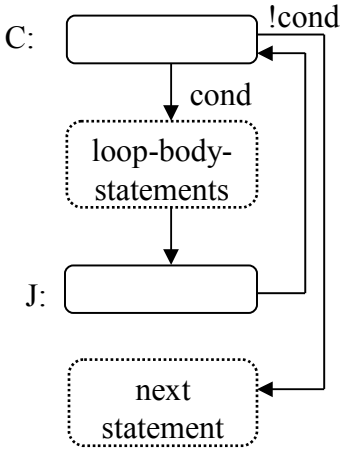
Assignment statement

a = b
next statement



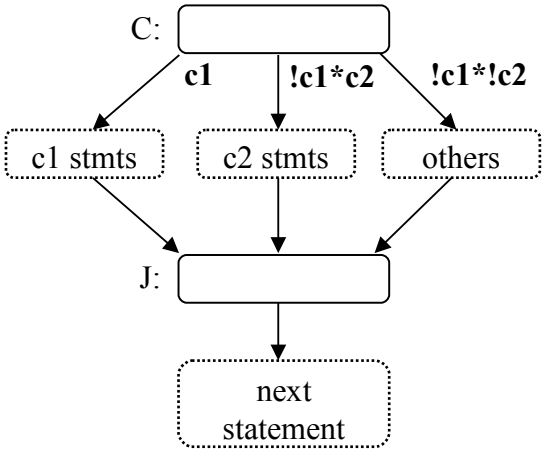
Loop statement

while (cond) {
loop-body-
statements
}
next statement



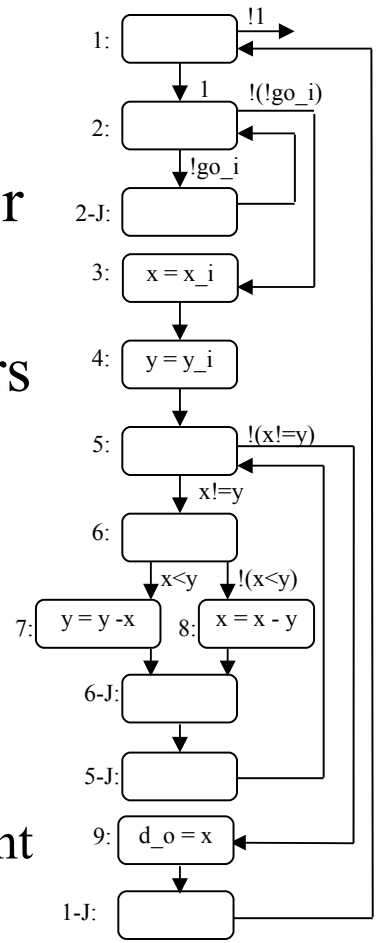
Branch statement

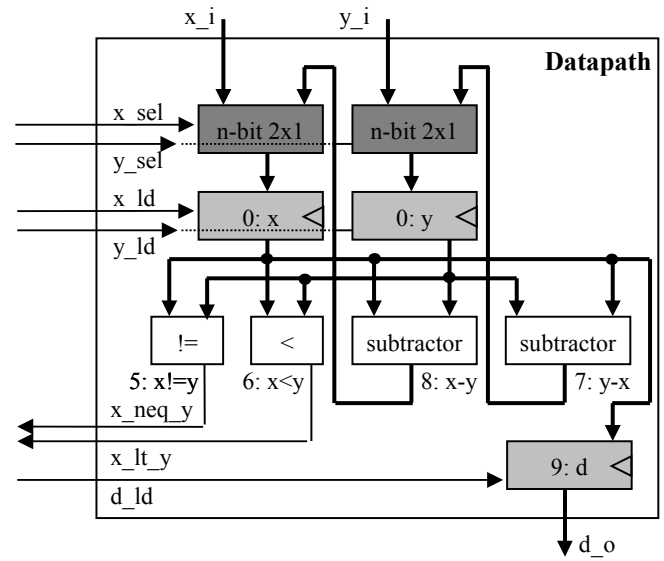
if (c1)
c1 stmts
else if c2
c2 stmts
else
other stmts
next statement



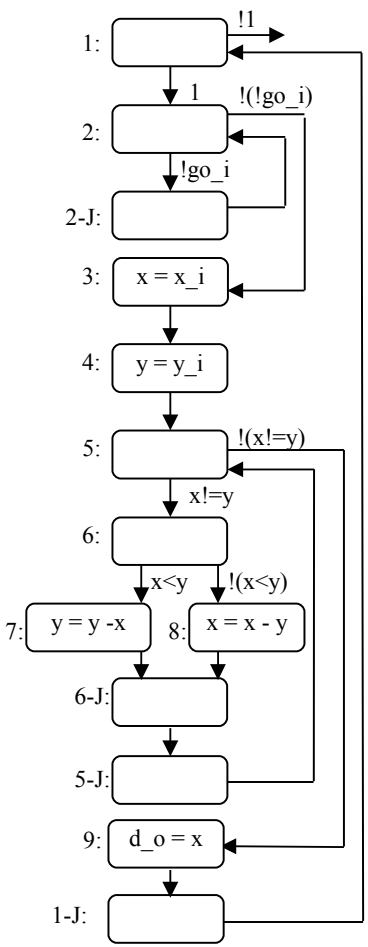
Creating the datapath

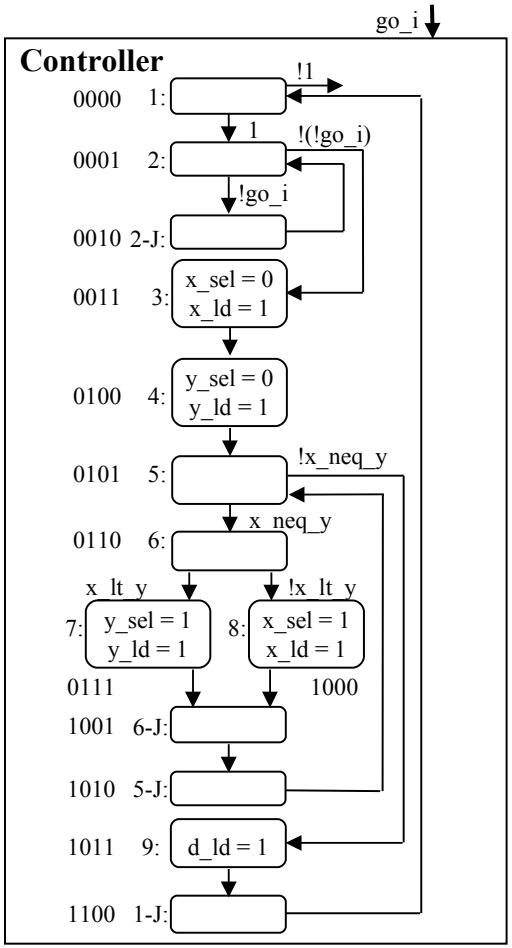
- Create a register for any declared variable
- Create a functional unit for each arithmetic operation
- Connect the ports, registers and functional units
  - Based on reads and writes
  - Use multiplexors for multiple sources
- Create unique identifier
  - for each datapath component control input and output



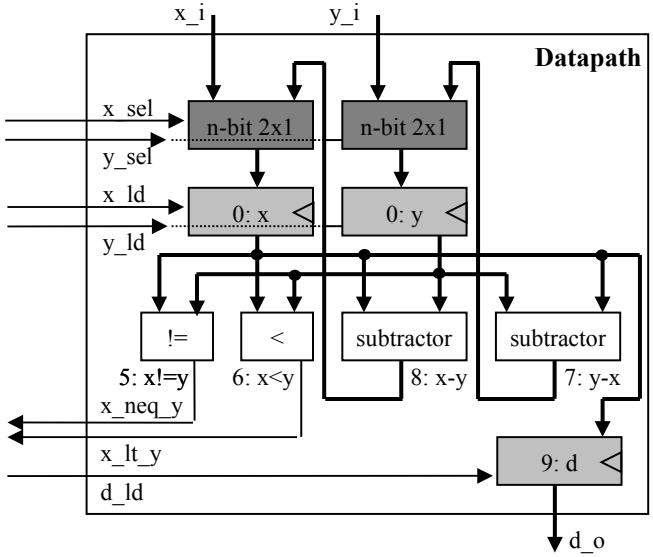


Creating the controller's FSM

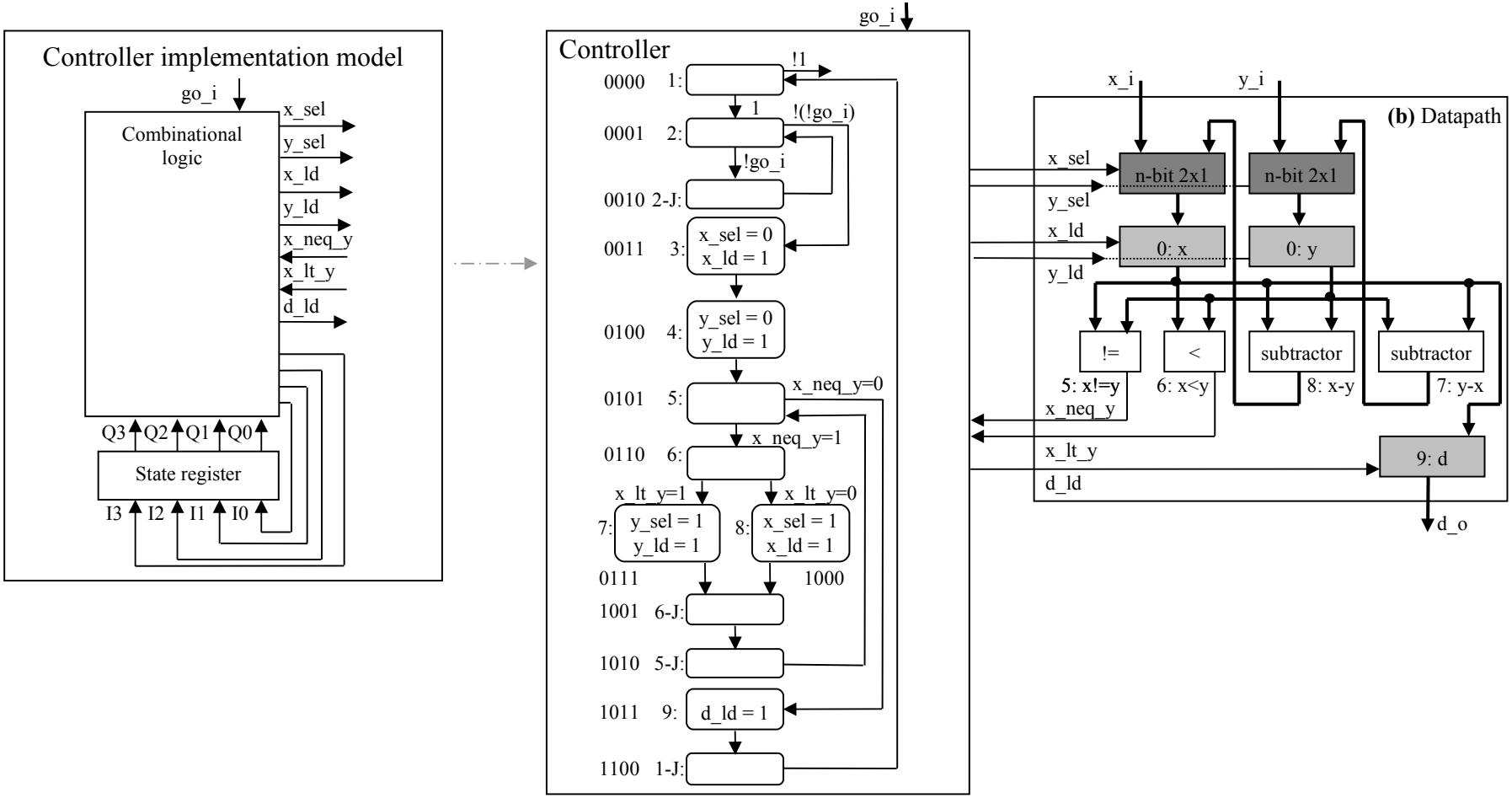




- Same structure as FSMD
- Replace complex actions/conditions with datapath configurations



Splitting into a controller and datapath

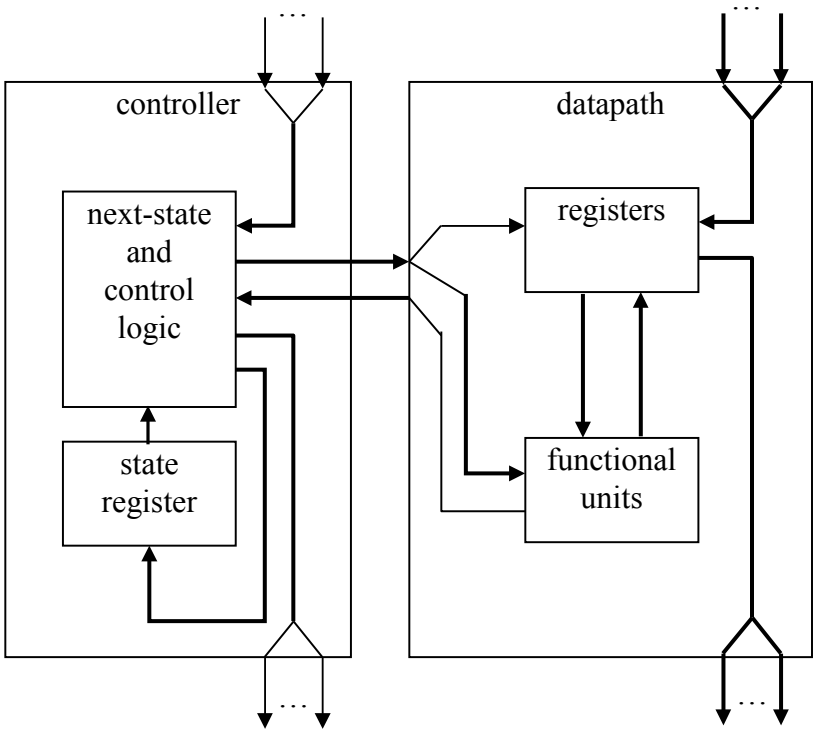


Controller state table for the GCD example

Inputs							Outputs								
Q3	Q2	Q1	Q0	x_neq_y	x_lt_y	go_i	I3	I2	I1	I0	x_sel	y_sel	x_ld	y_ld	d_ld
0	0	0	0	*	*	*	0	0	0	1	X	X	0	0	0
0	0	0	1	*	*	0	0	0	1	0	X	X	0	0	0
0	0	0	1	*	*	1	0	0	1	1	X	X	0	0	0
0	0	1	0	*	*	*	0	0	0	1	X	X	0	0	0
0	0	1	1	*	*	*	0	1	0	0	0	X	1	0	0
0	1	0	0	*	*	*	0	1	0	1	X	0	0	1	0
0	1	0	1	0	*	*	1	0	1	1	X	X	0	0	0
0	1	0	1	1	*	*	0	1	1	0	X	X	0	0	0
0	1	1	0	*	0	*	1	0	0	0	X	X	0	0	0
0	1	1	0	*	1	*	0	1	1	1	X	X	0	0	0
0	1	1	1	*	*	*	1	0	0	1	X	1	0	1	0
1	0	0	0	*	*	*	1	0	0	1	1	X	1	0	0
1	0	0	1	*	*	*	1	0	1	0	X	X	0	0	0
1	0	1	0	*	*	*	0	1	0	1	X	X	0	0	0
1	0	1	1	*	*	*	1	1	0	0	X	X	0	0	1
1	1	0	0	*	*	*	0	0	0	0	X	X	0	0	0
1	1	0	1	*	*	*	0	0	0	0	X	X	0	0	0
1	1	1	0	*	*	*	0	0	0	0	X	X	0	0	0
1	1	1	1	*	*	*	0	0	0	0	X	X	0	0	0

Completing the GCD custom single-purpose processor design

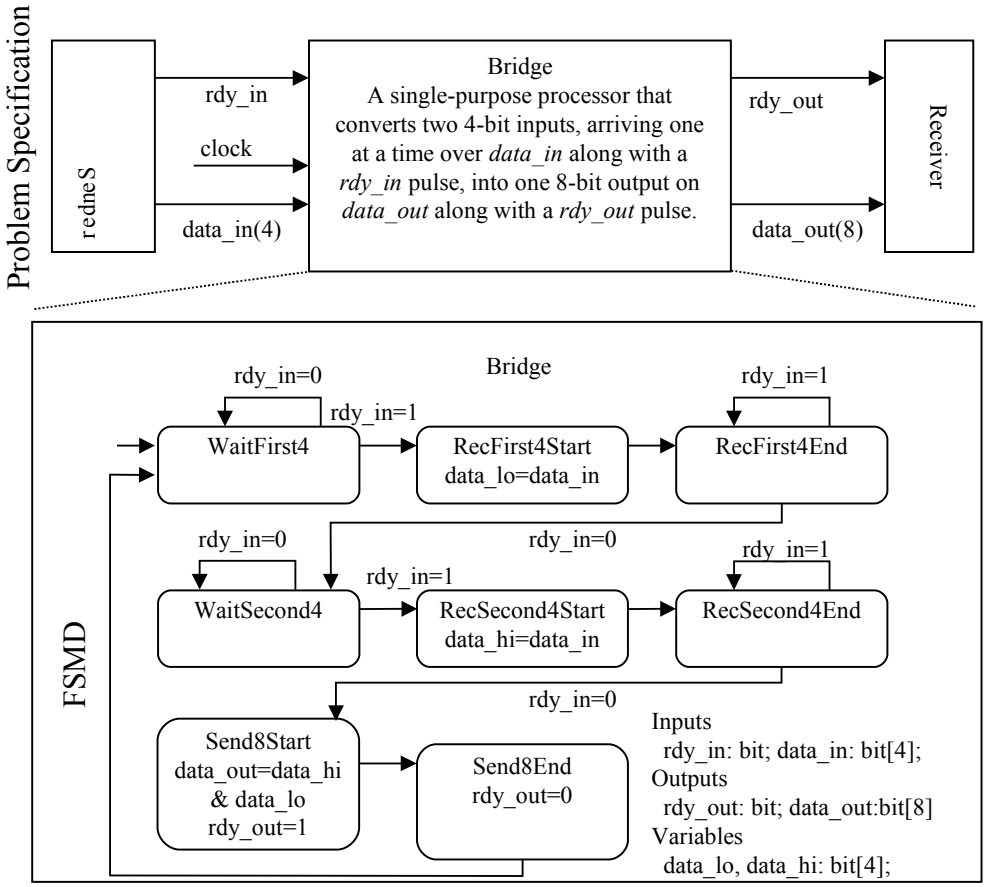
- We finished the datapath
- We have a state table for the next state and control logic
  - All that's left is combinational logic design
- This is *not* an optimized design, but we see the basic steps



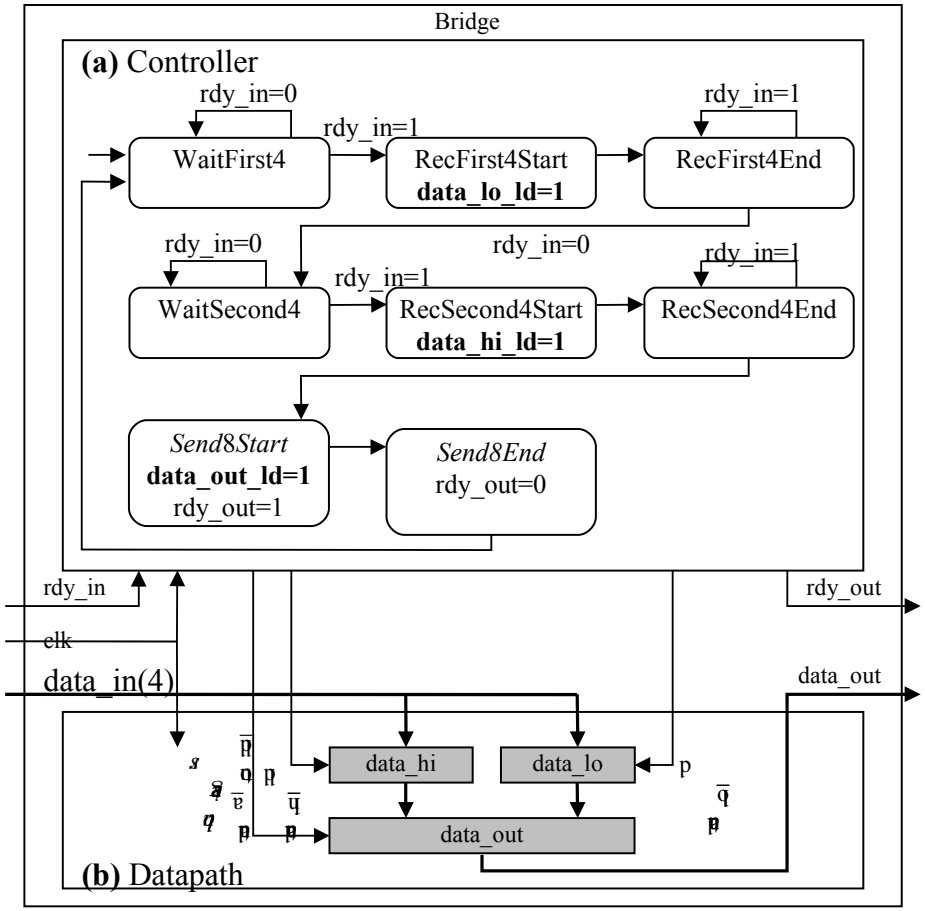
a view inside the controller and datapath

RT-level custom single-purpose processor design

- We often start with a state machine
  - Rather than algorithm
  - Cycle timing often too central to functionality
- Example
  - Bus bridge that converts 4-bit bus to 8-bit bus
  - Start with FSMD
  - Known as register-transfer (RT) level
  - Exercise: complete the design



RT-level custom single-purpose processor design



Optimizing single-purpose processors

- Optimization is the task of making design metric values the best possible
- Optimization opportunities
  - original program
  - FSMD
  - datapath
  - FSM

Optimizing the original program

- Analyze program attributes and look for areas of possible improvement
  - number of computations
  - size of variable
  - time and space complexity
  - operations used
    - multiplication and division very expensive

Optimizing the original program (cont')

original program

```
int  x, y;
while (1) {
  while (!go_i);
  x = x_i;
  y = y_i;
  while  (x != y)  {
     if  (x < y)
        y = y - x;
     else
        x = x - y;
  }
  d_o = x;
}
```

replace the subtraction operation(s) with modulo operation in order to speed up program

optimized program

```
int  x, y, r;
while (1) {
  while (!go_i);
  // x must be the larger number
  if (x_i >= y_i)  {
     x=x_i;
     y=y_i;
  }
  else {
     x=y_i;
     y=x_i;
  }
  while  (y != 0)  {
     r = x % y;
     x = y;
     y = r;
  }
  d_o = x;
}
```

GCD(42, 8) - 9 iterations to complete the loop

x and y values evaluated as follows : (42, 8), (43, 8), (26,8), (18,8), (10, 8), (2,8), (2,6), (2,4), (2,2).

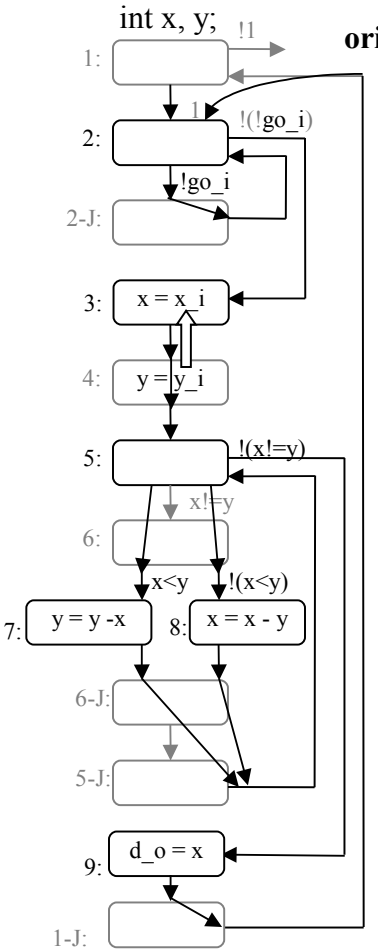
GCD(42,8) - 3 iterations to complete the loop

x and y values evaluated as follows: (42, 8), (8,2), (2,0)

Optimizing the FSMD

- Areas of possible improvements
  - merge states
    - states with constants on transitions can be eliminated, transition taken is already known
    - states with independent operations can be merged
  - separate states
    - states which require complex operations (a*b*c*d) can be broken into smaller states to reduce hardware size
  - scheduling

Optimizing the FSMD (cont.)



original FSMD

eliminate state 1 – transitions have constant values

merge state 2 and state 2J – no loop operation in between them

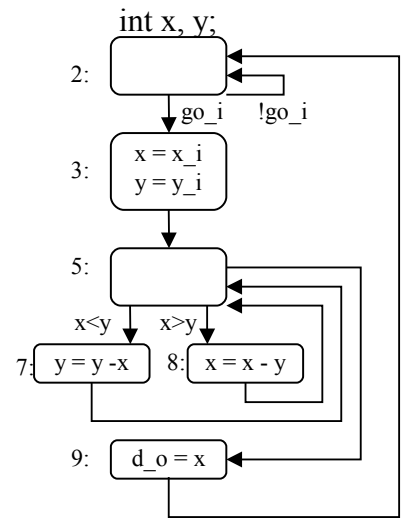
merge state 3 and state 4 – assignment operations are independent of one another

merge state 5 and state 6 – transitions from state 6 can be done in state 5

eliminate state 5J and 6J – transitions from each state can be done from state 7 and state 8, respectively

eliminate state 1-J – transition from state 1-J can be done directly from state 9

optimized FSMD



Optimizing the datapath

- Sharing of functional units
  - one-to-one mapping, as done previously, is not necessary
  - if same operation occurs in different states, they can share a single functional unit
- Multi-functional units
  - ALUs support a variety of operations, it can be shared among operations occurring in different states

Optimizing the FSM

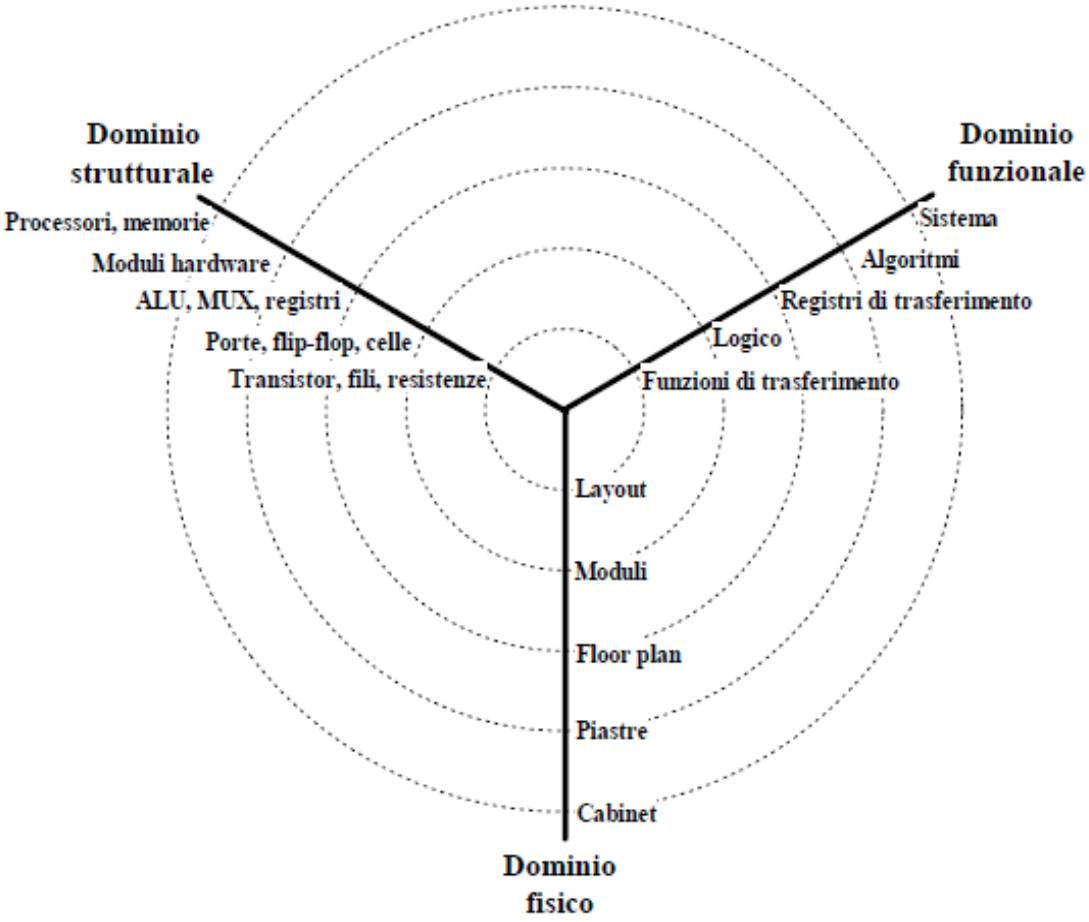
- State encoding
  - task of assigning a unique bit pattern to each state in an FSM
  - size of state register and combinational logic vary
  - can be treated as an ordering problem
- State minimization
  - task of merging equivalent states into a single state
    - state equivalent if for all possible input combinations the two states generate the same outputs and transitions to the next same state

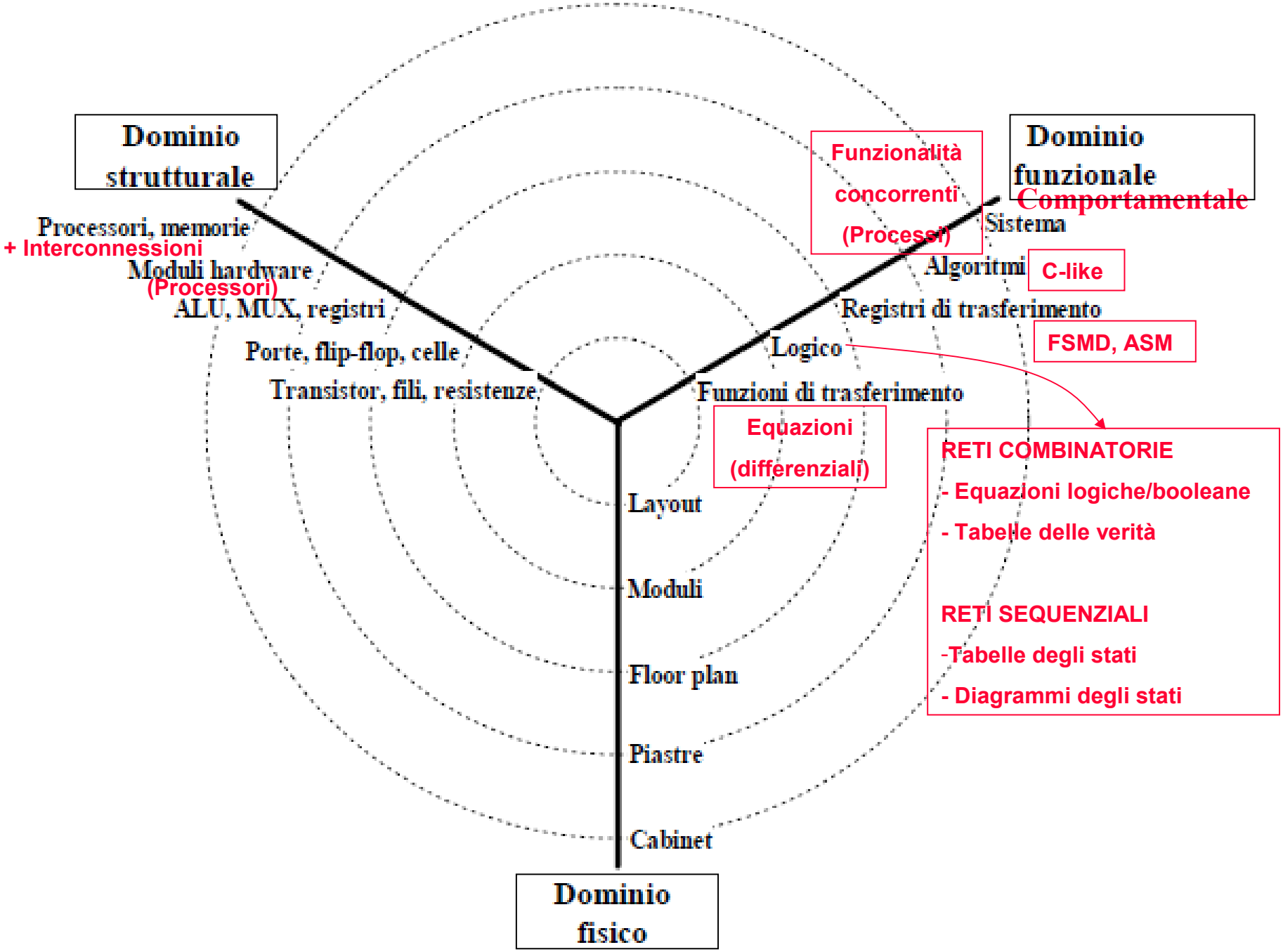
Summary

- Custom single-purpose processors
  - Straightforward design techniques
  - Can be built to execute algorithms
  - Typically start with FSMD
  - CAD tools can be of great assistance

Considerazioni

- Diagramma a Y




Dominio strutturale
Processori, memorie
+ Interconnessioni
Moduli hardware
(Processori)
ALU, MUX, registri
Porte, flip-flop, celle
Transistor, fili, resistenze
Dominio funzionale
Comportamentale
Funzionalità concorrenti (Processi)
Sistema
Algoritmi
C-like
Registri di trasferimento
FSMD, ASM
Logico
Funzioni di trasferimento
Equazioni (differenziali)
RETI COMBINATORIE
- Equazioni logiche/booleane
- Tabelle delle verità
RETI SEQUENZIALI
-Tabelle degli stati
- Diagrammi degli stati
Layout
Moduli
Floor plan
Piastre
Cabinet
Dominio fisico