

## Il progetto a livello RT di sistemi embedded



Frank Vahid, Tony Givargis  
*The University of California*

## Sommaro

- Panoramica sugli embedded systems
  - Cosa sono?
- Le sfide progettuali – ottimizzazione delle metriche di progetto
- Tecnologie
  - Tecnologie dei processori
  - Tecnologie degli IC
  - Tecnologie di progetto

A.A. 2006/07 02 Embedded Systems Design 2

## Panoramica sugli Embedded systems

I sistemi di elaborazione sono onnipresenti  
Pensiamo sempre ai “desktop” computer

- PC
- Laptop
- Mainframe
- Server


Esiste un altro tipo di sistemi di elaborazione


- molto più pervasivo...


A.A. 2006/07 02 Embedded Systems Design 3


## Sistemi di elaborazione embedded

- In apparecchiature elettroniche
- Difficili da definire. Quasi tutti i sistemi di elaborazione ≠ desktop computer
- Miliardi di pezzi/anno, rispetto a milioni di desktop
- Forse 50 per case e per automobile

I computer sono qui... 

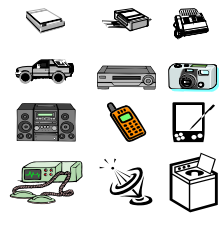
e qui... 

e persino qui... 

Molti di più di questi, anche se singolarmente costano meno. 

A.A. 2006/07 02 Embedded Systems Design 4

## Alcuni sistemi embedded

Anti-lock brakes Auto-focus cameras Automatic teller machines Automatic toll systems Automatic transmission Avionic systems Battery chargers Camcorders Cell phones Cell-phone base stations Cordless phones Cruise control Curbside check-in systems Digital cameras Disk drives Electronic card readers Electronic instruments Electronic toys/games Factory control Fax machines Fingerprint identifiers Home security systems Life-support systems Medical testing systems	Modems MPEG decoders Network cards Network switches/routers On-board navigation Pagers Photocopiers Point-of-sale systems Portable video games Printers Satellite phones Scanners Smart ovens/dishwashers Speech recognizers Stereo systems Teleconferencing systems Televisions Temperature controllers Theft tracking systems TV set-top boxes VCR's, DVD players Video game consoles Video phones Washers and dryers	
---	--	---

A.A. 2006/07 02 Embedded Systems Design 5

## Caratteristiche comuni

- Una sola funzione
  - eseguono ripetutamente una sola funzione
- Fortemente vincolati
  - basso costo, basso consumo, veloci, piccoli etc.
- Reattivi e real-time
  - reagiscono in continuazioni agli eventi nell'ambiente
  - calcolano i risultati in tempo reale senza ritardi

A.A. 2006/07 02 Embedded Systems Design 6

### Esempio

Una sola funzione -- apparecchio fotografico digitale  
 Vincoli -- basso costo, basso consumo, piccola, veloce  
 Reattiva e real-time -- in misura limitata

A.A. 2006/07 02 Embedded Systems Design 7

### Ottimizzazione metriche di progetto

Obiettivo (ovvio):  
 ■ implementazione con funzionalità desiderata

Sfida progettuale:  
 ■ ottimizzazione contemporanea di diverse metriche di progetto

Metriche di progetto  
 ■ caratteristica misurabile dell'implementazione di un sistema  
 ■ l'ottimizzazione delle metriche di progetto è una sfida chiave

A.A. 2006/07 02 Embedded Systems Design 8

### Metriche comuni:

- **Costo unitario:** costo di produrre ogni copia del sistema, esclusi il costo NRE (\$ o €)
- **Costo NRE (Non-Recurring Engineering cost):** costo una tantum di progetto del sistema (\$ o €)
- **Dimensione:** spazio fisico richiesto dal sistema
- **Prestazione:** tempo di esecuzione o throughput del sistema
- **Consumo:** potenza consumata dal sistema
- **Flessibilità:** possibilità di cambiare la funzione senza costi NRE troppo elevati

A.A. 2006/07 02 Embedded Systems Design 9

- **Time-to-prototype:** tempo per costruire una versione funzionante del sistema
- **Time-to-market:** tempo per sviluppare un sistema vendibile
- **Maintainability:** possibilità di modificare il sistema dopo la prima versione
- **Correttezza, safety, etc.**

A.A. 2006/07 02 Embedded Systems Design 10

### Competizione tra metriche

Richiesta expertise sia **software** che **hardware**

- Non solo esperti hardware o software
- Progettista esperto di varie tecnologie per scegliere la migliore in funzione di applicazione e vincoli.

A.A. 2006/07 02 Embedded Systems Design 11

### Time-to-market

Tempo necessario per sviluppare un prodotto vendibile

Finestra di mercato

- Periodo di massima vendita del prodotto

Vincolo medio di time-to-market circa 8 mesi  
 Ritardi onerosi

A.A. 2006/07 02 Embedded Systems Design 12

### Perdite per ingresso ritardato

Modello introiti semplificato

- Vita del prodotto =  $2W$ , picco a  $W$
- Il tempo di ingresso sul mercato definisce un triangolo (penetrazione di mercato)
- Area: introito
- Perdita
- Differenza di aree

A.A. 2006/07 02 Embedded Systems Design 13

Area =  $1/2 * \text{base} * \text{altezza}$

- in tempo =  $1/2 * 2W * W$
- in ritardo =  $1/2 * (W - D + W) * (W - D)$

Perdita di introiti (%) =  $(D(3W - D) / 2W^2) * 100\%$

Esempi:

- Vita  $2W=52$  sett, ritardo  $D=4$  sett
- $(4 * (3 * 26 - 4) / 2 * 26^2) = 22\%$
- Vita  $2W=52$  sett, ritardo  $D=10$  sett
- $(10 * (3 * 26 - 10) / 2 * 26^2) = 50\%$
- i ritardi costano!

A.A. 2006/07 02 Embedded Systems Design 14

### NRE e costo unitario

Costi:

- costo totale = costo NRE + costo unitario \* # di pezzi
- costo per prodotto = costo totale / # di pezzi
- = (costo NRE / # di pezzi) + costo unitario

Esempio

- NRE=\$2000, unitario=\$100
- Per 10 pezzi
- costo totale =  $\$2000 + 10 * \$100 = \$3000$
- costo per prodotto =  $\$2000 / 10 + \$100 = \$300$

*Ammortizzare il costo NRE sui pezzi prodotti comporta un costo aggiuntivo di \$ 200 per pezzo*

A.A. 2006/07 02 Embedded Systems Design 15

### Confronto di tecnologie in base ai costi: dipende dalle quantità prodotte

- Tecnologia A: NRE=\$2,000, unitario=\$100
- Tecnologia B: NRE=\$30,000, unitario=\$30
- Tecnologia C: NRE=\$100,000, unitario=\$2

• Bisogna anche considerare il time-to-market

A.A. 2006/07 02 Embedded Systems Design 16

### Metrica delle prestazioni

Metrica molto usata, ma molto abusata

- frequenza del clock istruzioni al secondo – metriche non buone
- Esempio: digital camera – all'utente interessa la velocità di elaborazione dell'immagine, non quella del clock o le istruzioni al secondo

Latenza (tempo di risposta)

- tempo tra inizio e fine di un task
- esempio: macchine A e B elaborano immagini in 0.25 s

A.A. 2006/07 02 Embedded Systems Design 17

### Throughput

- Tasks al secondo, esempio macchina A elabora 4 immagini al secondo
- Throughput può essere maggiore della latenza grazie alla concorrenza, esempio macchina B elabora 8 immagini/s perché acquisisce nuova immagine mentre memorizza la vecchia

Speedup di B rispetto ad A = prestazione di B / prestazione di A

- Throughput speedup =  $8 / 4 = 2$

A.A. 2006/07 02 Embedded Systems Design 18

### Tre tecnologie chiave

Tecnologia

- modo per eseguire un task, specialmente usando processi tecnici, metodi o conoscenza

Tre tecnologie-chiave per i sistemi embedded

- tecnologia dei processori
- tecnologia degli IC
- tecnologia di progetto

A.A. 2006/07 02 Embedded Systems Design 19

### Tecnologia dei processori

Architettura dell'elaboratore per ottenere la funzionalità desiderata

Il processore non è necessariamente programmabile

- "Processore" non è sinonimo di "general-purpose processor"

The diagram shows three processor architectures:

- General-purpose ("software"):** Includes a Controller (Control logic and State register), Datapath (Register file, General ALU), Program memory (Assembly code for: total = 0 for i=1 to N), and Data memory.
- Application-specific:** Includes a Controller (Control logic and State register), Datapath (Registers, Custom ALU), Program memory (Assembly code for: total = 0 for i=1 to N), and Data memory.
- Single-purpose ("hardware"):** Includes a Controller (Control logic, State register), Datapath (Index, total, +), and Data memory.

A.A. 2006/07 02 Embedded Systems Design 20

### Esempio

Desired functionality

```
total = 0
for i = 1 to N loop
total += M[i]
end loop
```

The diagram shows three processor types represented by different shapes:

- General-purpose processor (rectangle)
- Application-specific processor (cross shape)
- Single-purpose processor (plus sign)

A.A. 2006/07 02 Embedded Systems Design 21

### Processori general-purpose

Dispositivi programmabili usati per molte applicazioni

- noti anche come "microprocessori"

Caratteristiche

- Memoria dei programmi
- datapath generale con register file grande e ALU generale

Vantaggi

- costi time-to-market e NRE bassi
- notevole flessibilità

Esempio: "Pentium" ma anche molti altri

The diagram shows a general-purpose processor architecture with a Controller (Control logic and State register, IR, PC), Datapath (Register file, General ALU), Program memory (Assembly code for: total = 0 for i=1 to N), and Data memory.

A.A. 2006/07 02 Embedded Systems Design 22

### Processori single-purpose

Circuiti digitali progettati per eseguire esattamente un solo programma

- coprocessori, acceleratori etc.

Caratteristiche

- Contengono solo i componenti necessari per eseguire un singolo programma
- Non hanno memoria dei programmi

Vantaggi

- veloci
- basso consumo
- piccoli

The diagram shows a single-purpose processor architecture with a Controller (Control logic, State register), Datapath (index, total, +), and Data memory.

A.A. 2006/07 02 Embedded Systems Design 23

### Processori application-specific

Processori programmabili ottimizzati per una classe di applicazioni con caratteristiche comuni

- compromesso tra processori general-purpose e single-purpose

Caratteristiche

- Memoria dei programmi
- Datapath ottimizzato
- Unità funzionali speciali

vantaggi

- Buona flessibilità, buone prestazioni, dimensioni e consumo

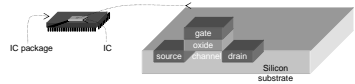
The diagram shows an application-specific processor architecture with a Controller (Control logic and State register, IR, PC), Datapath (Registers, Custom ALU), Program memory (Assembly code for: total = 0 for i=1 to N), and Data memory.

A.A. 2006/07 02 Embedded Systems Design 24

## Tecnologie degli IC

Modo in cui un'implementazione digitale a livello gate è mappata su un IC

- IC: circuito integrato o "chip"
- le tecnologie IC differiscono per come particolarizzano i progetti
- gli IC consistono di diversi layer (anche più di 10)
  - differiscono rispetto a chi costruisce ciascun layer e quando



A.A. 2006/07 02 Embedded Systems Design 25

## Tecnologie degli IC

Tre tipi:

- Full-custom/VLSI
- Semi-custom ASIC (gate array e standard cell)
- PLD (Programmable Logic Device)

A.A. 2006/07 02 Embedded Systems Design 26

## Full-custom/VLSI

Tutti i layer sono ottimizzati per una certa implementazione digitale di un sistema embedded

- Placing dei transistor
- Sizing dei transistor
- Routing dei fili

Vantaggi

- Ottime prestazioni, piccole dimensioni, consumi ridotti

Svantaggi

- alto costo NRE (es. \$300k), time-to-market lungo

A.A. 2006/07 02 Embedded Systems Design 27

## Semi-custom

I layer inferiori sono precostruiti parzialmente o totalmente

- il progettista non si occupa del routing

Vantaggi

- Buone prestazioni, buone dimensioni, costo NRE minore del full custom (tra \$10k e \$100k)

Svantaggi

- tempo di sviluppo: da settimane a mesi

A.A. 2006/07 02 Embedded Systems Design 28

## PLD

Esistono già tutti i layer:

- i progettisti possono acquisire un IC
- le connessioni sull'IC vengono create o distrutte per ottenere la funzionalità desiderata
- molto usati i Field-Programmable Gate Array (FPGA)

A.A. 2006/07 02 Embedded Systems Design 29

Vantaggi

- costi NRE bassi, disponibilità IC quasi immediata

Svantaggi

- Più grandi, costosi (anche \$30 per pezzo), alto consumo, più lenti

A.A. 2006/07 02 Embedded Systems Design 30

### La legge di Moore

- 1965: Gordon Moore (co-fondatore di INTEL)

**La capacità degli IC raddoppia ogni 18 mesi**

Logic transistors per chip (in millions)

Note: logarithmic scale

A.A. 2006/07 02 Embedded Systems Design 31

- Un chip del 2002 può contenere circa 15000 chip del 1981

A.A. 2006/07 02 Embedded Systems Design 32

### Tecnologia di progetto

Come si trasforma una funzionalità desiderata in un'implementazione

A.A. 2006/07 02 Embedded Systems Design 33

### La produttività di progetto

Incremento esponenziale negli ultimi decenni

A.A. 2006/07 02 Embedded Systems Design 34

### La scala del co-design

Passato:

- Tecnologie di progetto hardware e software molto diverse
- Maturità delle tecniche di sintesi: visione unificata di hardware e software

Hardware/software "codesign"

La scelta se implementare in hardware o software una funzione è un tradeoff tra metriche di progetto, quali prestazioni, consumi, dimensioni, costo NRE e flessibilità: non c'è nessuna differenza fondamentale tra quanto implementabile hardware e software.

A.A. 2006/07 02 Embedded Systems Design 35

### Indipendenza tra tecnologie dei processori e degli IC

Tradeoff

- Generale vs. custom
- rispetto alla tecnologia dei processori o degli IC
- le due tecnologie sono indipendenti

A.A. 2006/07 02 Embedded Systems Design 36

### Il gap di produttività

L'aumento di produttività di progetto non ha tenuto il passo con l'aumento della capacità dei chip

A.A. 2006/07 02 Embedded Systems Design 37

1981: chip di punta:  
 ■ 100 mesi-uomo = 10.000 transistor/100 transistor/mese  
 2002:chip di punta:  
 ■ 30000 mesi-uomo = 150 Mtransistor/5000 transistor/mese  
 Costo del progettista cresce da \$1M a \$300M

A.A. 2006/07 02 Embedded Systems Design 38

In teoria, bastano squadre più numerose di progettisti  
 In pratica, squadre numerose decrementano la produttività per problemi di gestione e di comunicazione

1M transistor, 1 designer=5000 trans/mese  
 Ogni designerin più riduce di 100 trans/month  
 Due designer producono ciascuno 4900 trans/mese

A.A. 2006/07 02 Embedded Systems Design 39

### Progetto di processori single-purpose

A.A. 2006/07 02 Embedded Systems Design 40

### Processori single-purpose

Processore

- circuito digital che esegue un'elaborazione
- datapath e unità di controllo
- General-purpose: molti tipi di elaborazioni
- Single-purpose: un tipo di elaborazione
- Custom single-purpose: elaborazione non standard

Un processore custom single-purpose può essere

- veloce, piccolo, a basso consumo
- alto costo NRE, time-to-market maggiore, meno flessibile

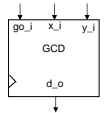
A.A. 2006/07 02 Embedded Systems Design 41

### Processore custom single-purpose algoritmico

A.A. 2006/07 02 Embedded Systems Design 42

### Esempio: massimo comun divisore

- Specifica: dati  $x_i$  e  $y_i$  su 4 bit, calcolare il loro massimo comun divisore  $d_o$  su 4 bit,  $go_i$  segnale che fa partire il calcolo



```

entity gcd is
port(
  clk:    in std_logic;
  rst:    in std_logic;
  go_i:   in std_logic;
  x_i:    in unsigned(3 downto 0);
  y_i:    in unsigned(3 downto 0);
  d_o:    out unsigned(3 downto 0)
);
end gcd;
    
```

A.A. 2006/07 02 Embedded Systems Design 43

### Algoritmo

```

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;
8:     else
9:       x = x - y;
  }
9:   d_o = x;
  }
    
```

A.A. 2006/07 02 Embedded Systems Design 44

### Architecture

```

architecture FSM of gcd is
begin
  process(rst, clk)
  -- define states using variable
  type S_Type is (ST0, ST1, ST2);
  variable State: S_Type := ST0;
  variable Data_X, Data_Y: unsigned(3 downto 0);
  begin
    if (rst='1') then -- initialization
      d_o <= "0000";
      State := ST0;
    elsif (clk'event and clk='1') then
      case State is
        when ST0 => -- starting
          when go_i='1' then
            if (go_i='1') then
              Data_X := x_i;
              Data_Y := y_i;
              State := ST1;
            else
              State := ST0;
            end if;
          end if;
        end case;
      end process;
    end if;
  end process;
end architecture FSM;
    
```

A.A. 2006/07 02 Embedded Systems Design 45

```

when ST1 => -- idle state
  State := ST2;
when ST2 => -- computation
  if (Data_X /= Data_Y) then
    if (Data_X < Data_Y) then
      Data_Y := Data_Y - Data_X;
    else
      Data_X := Data_X - Data_Y;
    end if;
    State := ST1;
  else
    d_o <= Data_X; -- done
    State := ST0;
  end if;
when others => -- go back
  d_o <= "ZZZZ";
  State := ST0;
end case;
end if;
end process;
end FSM;
    
```

A.A. 2006/07 02 Embedded Systems Design 46


### FSMD

- Dall'algoritmo al diagramma degli stati di una FSMD (finite-state machine with datapath)

Istruzione di assegnazione

```

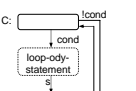
a = b
next statement
        
```



Istruzione di ciclo

```

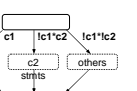
while (cond)
{
  loop-body-
  statements
}
next statement
        
```



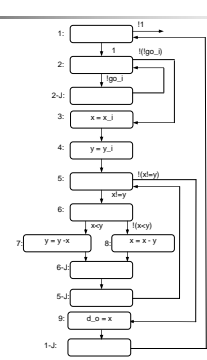
Istruzione di test

```

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



A.A. 2006/07 02 Embedded Systems Design 47

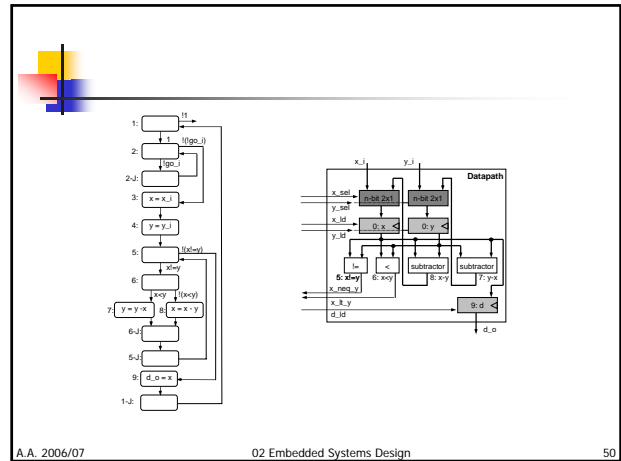


A.A. 2006/07 02 Embedded Systems Design 48

### Sintesi del datapath

- Creare un registro per ogni variabile dichiarata
- Creare un'unità funzionale per ogni operazione aritmetica
- Connettere porte, registri e unità funzionali:
  - basate su letture e scritture
  - usare multiplexer per sorgenti multiple
- Creare un identificatore unico
  - per tutti gli ingressi e le uscite di controllo dei componenti del datapath

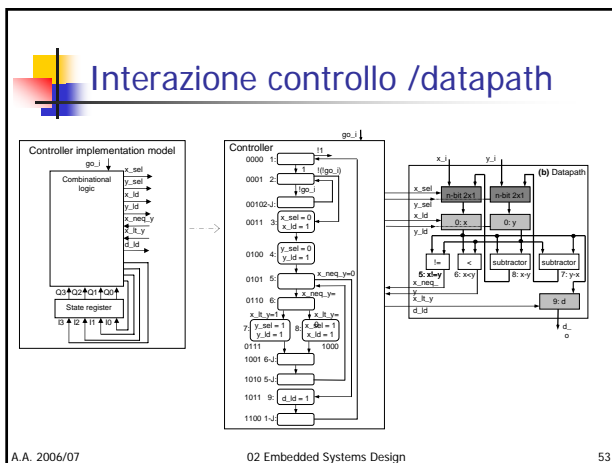
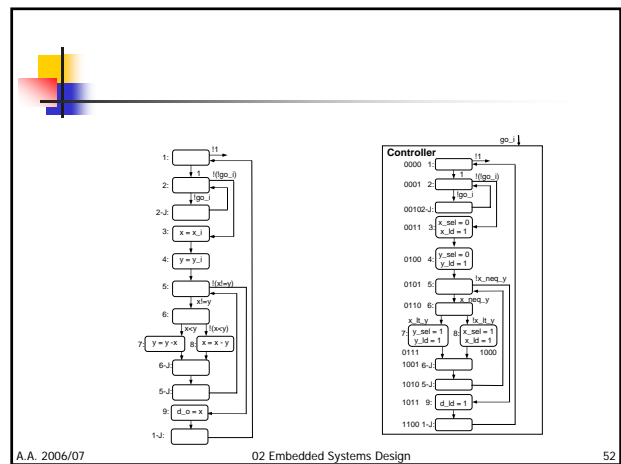
A.A. 2006/07 02 Embedded Systems Design 49



### Sintesi dell'unità di controllo

- Stessa struttura del diagramma degli stati della FSM
- Sostituire azioni o condizioni complesse con le configurazioni del datapath

A.A. 2006/07 02 Embedded Systems Design 51



### Tabella degli stati

Inputs										Outputs						
Q3	Q2	Q1	Q0	x_sel	x_sel	x_sel	go_i	I3	I2	I1	I0	x_sel	y_sel	x_sel	y_sel	d_sel
0	0	0	0	0	0	0	0	0	0	0	0	1	X	X	0	0
0	0	0	0	1	-	-	0	0	0	0	1	0	X	X	0	0
0	0	0	1	-	-	-	1	0	0	0	1	1	X	X	0	0
0	0	1	0	-	-	-	0	0	0	0	1	X	X	0	0	
0	0	1	1	-	-	-	0	1	0	0	0	0	X	X	1	0
0	1	0	0	-	-	-	0	1	0	0	1	X	0	0	1	0
0	1	0	1	0	-	-	1	0	0	1	1	X	X	0	0	0
0	1	0	1	1	-	-	0	1	0	0	1	X	X	0	0	0
0	1	1	0	-	-	-	0	1	1	0	0	X	X	0	0	0
0	1	1	1	-	-	-	0	1	1	1	1	X	X	0	0	0
1	0	0	0	-	-	-	0	0	0	0	0	0	X	X	0	0
1	0	0	1	-	-	-	0	0	0	0	1	X	0	0	1	0
1	0	1	0	-	-	-	0	0	0	0	0	X	X	0	0	0
1	0	1	1	-	-	-	0	0	0	0	0	X	X	0	0	0
1	1	0	0	-	-	-	0	0	0	0	0	X	X	0	0	0
1	1	0	1	-	-	-	0	0	0	0	0	X	X	0	0	0
1	1	1	0	-	-	-	0	0	0	0	0	X	X	0	0	0
1	1	1	1	-	-	-	0	0	0	0	0	X	X	0	0	0

A.A. 2006/07 02 Embedded Systems Design 54

Progettati:

- datapath
- tabella stati UC

Resta da fare:

- sintesi della parte combinatoria dell'UC

Il risultato non è ottimizzato.

a view inside the controller and datapath

A.A. 2006/07 02 Embedded Systems Design 55

### Modello strutturale VHDL

```

entity gcd is
  port( rst, clk, go_i: in std_logic;
        x_i, y_i: in std_logic_vector( 3 downto 0 );
        d_o: out std_logic_vector( 3 downto 0 )
  );
end gcd;

architecture gcd_arc of gcd is
  component fsm is
    port( rst, clk, proceed: in std_logic;
          comparison: in std_logic_vector( 1 downto 0 );
          enable, xsel, ysel, xld, yld: out std_logic
    );
  end component;
  component mux is
    port( rst, sLine: in std_logic;
          load, result: in std_logic_vector( 3 downto 0 );
          output: out std_logic_vector( 3 downto 0 )
    );
  end component;
end gcd_arc;
    
```

A.A. 2006/07 02 Embedded Systems Design 56

```

component comparator is
  port( rst: in std_logic;
        x, y: in std_logic_vector( 3 downto 0 );
        output: out std_logic_vector( 1 downto 0 )
  );
end component;
component subtractor is
  port( rst: in std_logic;
        cmd: in std_logic_vector( 1 downto 0 );
        x, y: in std_logic_vector( 3 downto 0 );
        xout, yout: out std_logic_vector( 3 downto 0 )
  );
end component;
component regis is
  port( rst, clk, load: in std_logic;
        input: in std_logic_vector( 3 downto 0 );
        output: out std_logic_vector( 3 downto 0 )
  );
end component;
    
```

A.A. 2006/07 02 Embedded Systems Design 57

```

signal xld, yld, xsel, ysel, enable: std_logic;
signal comparison: std_logic_vector( 1 downto 0 );
signal result: std_logic_vector( 3 downto 0 );

signal xsub, ysub, xmux, ymux, xreg, yreg: std_logic_vector( 3 downto 0 );
    
```

A.A. 2006/07 02 Embedded Systems Design 58

```

begin
  -- doing structure modeling here
  -- FSM controller
  TOFSM: fsm port map( rst, clk, go_i, comparison,
                      enable, xsel, ysel, xld, yld );

  -- Datapath
  X_MUX: mux port map( rst, xsel, x_i, xsub, xmux );
  Y_MUX: mux port map( rst, ysel, y_i, ysub, ymux );
  X_REG: regis port map( rst, clk, xld, xmux, xreg );
  Y_REG: regis port map( rst, clk, yld, ymux, yreg );
  U_COMP: comparator port map( rst, xreg, yreg, comparison );
  X_SUB: subtractor port map( rst, comparison, xreg, yreg, xsub, ysub );
  OUT_REG: regis port map( rst, clk, enable, xsub, result );
  d_o <= result;
end gcd_arc;
    
```

A.A. 2006/07 02 Embedded Systems Design 59

### Processore single-purpose RT

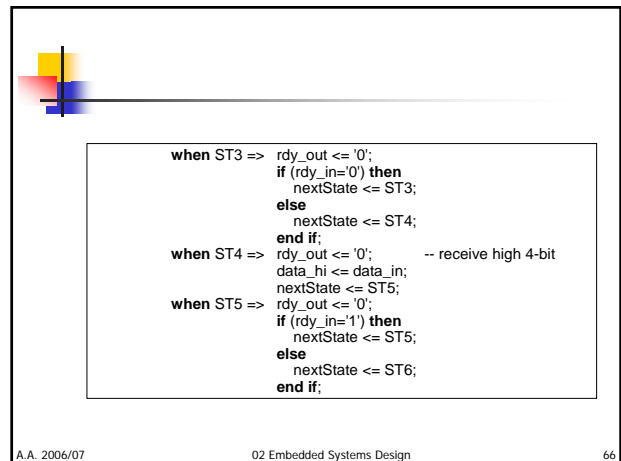
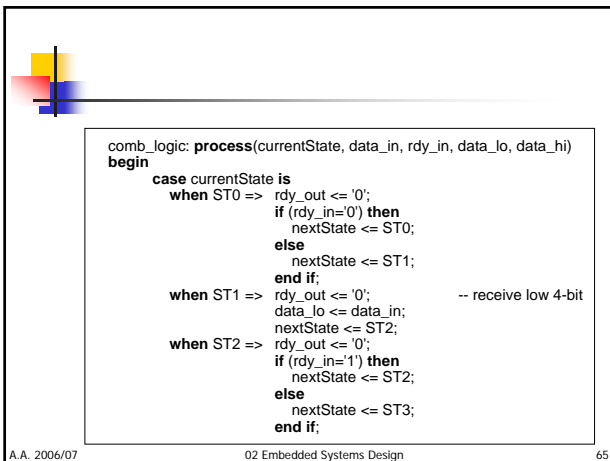
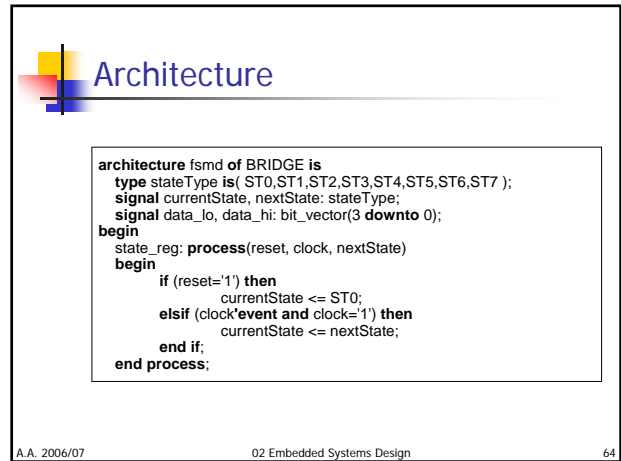
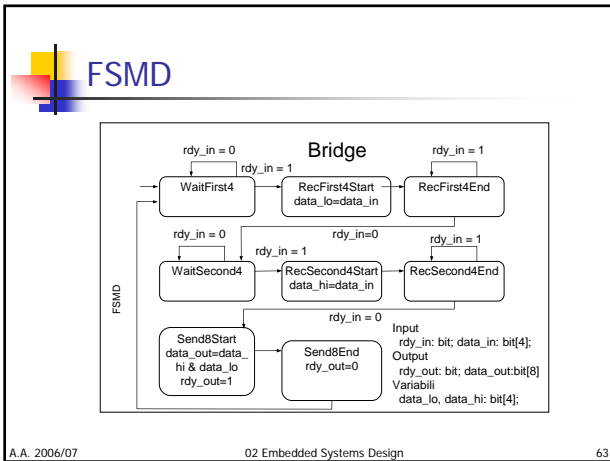
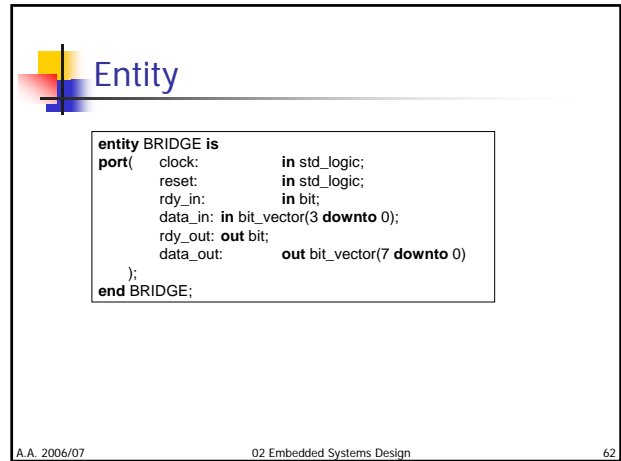
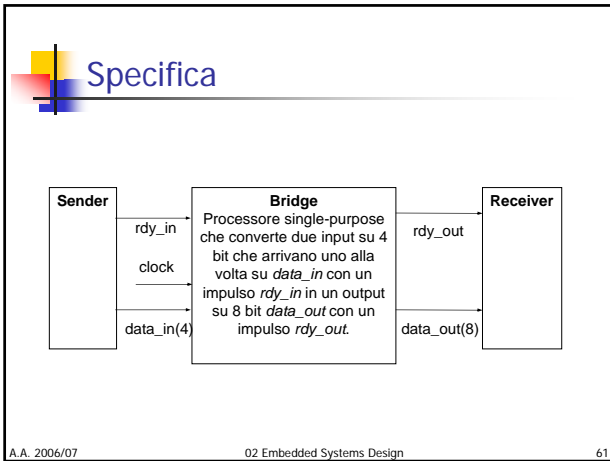
Sovente si parte da una macchina stati

- piuttosto che da un algoritmo
- la temporizzazione dei cicli è centrale rispetto alla funzionalità

Esempio

- Bus bridge che converte un bus a 4-bit in un bus a 8-bit
- Si parte da una FSM
- Il livello di astrazione è detto register-transfer (RT)

A.A. 2006/07 02 Embedded Systems Design 60



```

when ST6 => data_out <= data_lo & data_hi;
             rdy_out <= '1';
             nextState <= ST7;           -- data out ready
when ST7 => rdy_out <= '0';
             nextState <= ST0;           -- go back original
when others => rdy_out <= '0';
              nextState <= ST0;

end case;
end process;
end fsmd;
    
```

A.A. 2006/07 02 Embedded Systems Design 67

### Interazione controllo-datapath

A.A. 2006/07 02 Embedded Systems Design 68

### Modello strutturale VHDL

```

entity BRIDGE is
port(
    clock:         in std_logic;
    reset:         in std_logic;
    rdy_in:        in std_logic;
    data_in:       in std_logic_vector(3 downto 0);
    rdy_out:       out std_logic;
    data_out:      out std_logic_vector(7 downto 0)
);
end BRIDGE;

architecture struct of BRIDGE is
    component registers
    generic(n: positive);
    port(
        clk:         in std_logic;
        rst:         in std_logic;
        ld:          in std_logic;
        reg_in:      in std_logic_vector(n-1 downto 0);
        reg_out:     out std_logic_vector(n-1 downto 0)
    );
    end component;
    
```

A.A. 2006/07 02 Embedded Systems Design 69

```

component controller
port(
    ctrl_clk:      in std_logic;
    ctrl_rst:      in std_logic;
    sig_in:        in std_logic;
    data_lo_ld:    out std_logic;
    data_hi_ld:    out std_logic;
    data_out_ld:   out std_logic;
    sig_out:       out std_logic
);
end component;

signal wire0, wire1, wire2: std_logic;
signal connection: std_logic_vector(7 downto 0);
    
```

A.A. 2006/07 02 Embedded Systems Design 70

```

begin
CTRL: controller port map (
    clock, reset, rdy_in, wire0,
    wire1, wire2, rdy_out);
SREG_1: registers generic map(4)
port map (clock, reset, wire1, data_in, connection(7 downto
4));
SREG_2: registers generic map(4)
port map (clock, reset, wire0, data_in, connection(3 downto
0));
BREG: registers generic map(8)
port map (clock, reset, wire2, connection, data_out);
end struct;
    
```

A.A. 2006/07 02 Embedded Systems Design 71

### Ottimizzazione

Ottimizzazione: ricerca dei migliori valori per le metriche di progetto

Campi di applicazione dell'ottimizzazione

- algoritmo originale
- FSMD
- datapath
- FSM

A.A. 2006/07 02 Embedded Systems Design 72

## Ottimizzazione: algoritmo

**Analisi**

- complessità spazio-temporale
  - numero di operazioni di calcolo
  - dimensioni delle variabili
- costo delle singole operazioni
  - le moltiplicazioni e le divisioni sono costose

A.A. 2006/07 02 Embedded Systems Design 73

## Esempio

**algoritmo originale**

```

0: int x, y, r;
1: while (1) {
2:   while (!go_);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
12: }
```

GCD(42, 8) - 9 iterazioni richieste  
valori di x e y : (42, 8), (34, 8), (26, 8), (18, 8), (10, 8), (2, 8), (2, 8), (2, 4), (2, 2).

sostituire la sottrazione con il calcolo del resto della divisione (modulo)

**algoritmo ottimizzato**

```

0: int x, y, r;
1: while (1) {
2:   while (!go_);
3:   // x must be the larger number
4:   if (x_i >= y_i) {
5:     x = x_i;
6:   } else {
7:     y = y_i;
8:   }
9:   while (y != 0) {
10:    r = x % y;
11:    x = y;
12:    y = r;
13:  }
14:  d_o = x;
15: }
```

GCD(42,8) - 3 iterazioni richieste  
valori di x e y : (42, 8), (8, 2), (2, 0)

A.A. 2006/07 02 Embedded Systems Design 74

## Ottimizzazione: FSM

**Possibili miglioramenti:**

- fusione di stati
  - stati con costanti sulle transizioni sono eliminabili, la transizione è già nota
  - stati con operazioni indipendenti possono essere fusi
- separazione di stati
  - stati con operazioni complicate (es.  $a*b*c*d$ ) possono essere divisi in più stati per minimizzare la dimensione dello hardware
- scheduling

A.A. 2006/07 02 Embedded Systems Design 75

## Esempio

**FSMD originale**

**FSMD ottimizzata**

A.A. 2006/07 02 Embedded Systems Design 76

## Ottimizzazione: datapath

**Condivisione di unità funzionali**

- la corrispondenza 1:1 operazione : unità non è necessaria
- se la stessa operazione avviene in stati diversi, l'unità funzionale può essere condivisa

**Unità multi-funzionali**

- ALU: realizza molte operazioni, può essere condivisa tra operazioni che avvengono in stati diversi

A.A. 2006/07 02 Embedded Systems Design 77

## Ottimizzazione: FSM

- Codifica degli stati
- Minimizzazione degli stati
- fusione di stati equivalenti
  - due stati sono equivalenti se e solo se per tutte le combinazioni di ingressi generano le stesse uscite e evolvono verso stati equivalenti

A.A. 2006/07 02 Embedded Systems Design 78

## Ottimizzazione di FSM

Cap. 9 *Contemporary Logic Design*

Randy H. Katz  
University of California, Berkeley  
May 1993

A.A. 2006/07 02 Embedded Systems Design 79

## Procedura di progetto

*Procedura base di progetto di una FSM:*

- (1) Capire il problema
- (2) Derivare una specifica formale
- (3) Minimizzare gli stati
- (4) Codificare gli stati
- (5) Scegliere i FF per realizzare lo stato
- (6) Realizzare la logica combinatoria

A.A. 2006/07 02 Embedded Systems Design 80

## Esempio

Controllo di parità dispari: due diversi diagrammi degli stati

- stesso comportamento in uscita per qualsiasi sequenza di ingresso
- le FSM sono *equivalenti*, ma hanno realizzazioni diverse
- prima definire il diagramma degli stati senza preoccuparsi del numero degli stati, poi minimizzarli

A.A. 2006/07 02 Embedded Systems Design 81

## Motivazione

Perché realizzare una FSM con il numero minimo di stati?

- Richiede (di solito) meno flipflop
- I limiti importanti sono dati dalle potenze di 2
- Meno stati di solito forniscono più don't care
- Minimizza (di solito) il numero di porte nella realizzazione

A.A. 2006/07 02 Embedded Systems Design 82

## Minimizzazione degli stati

Obiettivo:  
Identificare e combinare stati con lo stesso comportamento

*Stati equivalenti:* per qualsiasi ingresso due stati equivalenti

- producono le stesse uscite
- hanno transizioni verso stati equivalenti (o uguali)

*Controllore di parità* S0, S2 sono equivalenti  
entrambi producono l'uscita 0  
entrambi passano ad S1 con 1 in ingresso e tornano su se stessi con 0 in ingresso

A.A. 2006/07 02 Embedded Systems Design 83

## Algoritmo

- Iniziare dalla tabella delle transizioni tra gli stati
- Identificare stati con lo stesso comportamento in uscita
- Se gli stati hanno lo stesso stato futuro, sono equivalenti
- Combinare insieme coppie di stati equivalenti
- Ripetere finché non ci sono più stati equivalenti

A.A. 2006/07 02 Embedded Systems Design 84

### Metodo della corrispondenza tra righe

Specifica:

- 1 ingresso X, 1 uscita Z
- Considerando gli ingressi a gruppi di quattro, generare 1 in uscita se l'ultimo gruppo era 1010 o 0110

Esempio di comportamento ingresso/uscita:

X = 0010 0110 1100 1010 0011 ...  
 Z = 0000 0001 0000 0001 0000 ...

Limite superiore alla complessità di questa FSM:

- quindici stati (1 + 2 + 4 + 8)
  - trenta transizioni (2 + 4 + 8 + 16)
- bastano per riconoscere qualsiasi stringa di 4 bit!

Diagramma degli stati:

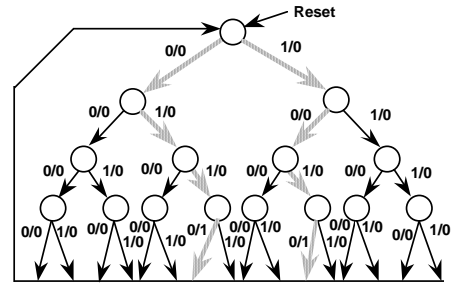


Tabella delle transizioni iniziale

Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	0	0
0	S <sub>1</sub>	S <sub>3</sub>	S <sub>4</sub>	0	0
1	S <sub>2</sub>	S <sub>5</sub>	S <sub>6</sub>	0	0
00	S <sub>3</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
01	S <sub>4</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
10	S <sub>5</sub>	S <sub>11</sub>	S <sub>12</sub>	0	0
11	S <sub>6</sub>	S <sub>13</sub>	S <sub>14</sub>	0	0
000	S <sub>7</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
001	S <sub>8</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
010	S <sub>9</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
011	S <sub>10</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0
100	S <sub>11</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
101	S <sub>12</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0
110	S <sub>13</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
111	S <sub>14</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0

Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	0	0
0	S <sub>1</sub>	S <sub>3</sub>	S <sub>4</sub>	0	0
1	S <sub>2</sub>	S <sub>5</sub>	S <sub>6</sub>	0	0
00	S <sub>3</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
01	S <sub>4</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
10	S <sub>5</sub>	S <sub>11</sub>	S <sub>12</sub>	0	0
11	S <sub>6</sub>	S <sub>13</sub>	S <sub>14</sub>	0	0
000	S <sub>7</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
001	S <sub>8</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
010	S <sub>9</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
011	S <sub>10</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0
100	S <sub>11</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
101	S <sub>12</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0
110	S <sub>13</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
111	S <sub>14</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0

Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	0	0
0	S <sub>1</sub>	S <sub>3</sub>	S <sub>4</sub>	0	0
1	S <sub>2</sub>	S <sub>5</sub>	S <sub>6</sub>	0	0
00	S <sub>3</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
01	S <sub>4</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
10	S <sub>5</sub>	S <sub>11</sub>	S <sub>12</sub>	0	0
11	S <sub>6</sub>	S <sub>13</sub>	S <sub>14</sub>	0	0
000	S <sub>7</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
001	S <sub>8</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
010	S <sub>9</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
011 or 101	S <sub>10</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0
100	S <sub>11</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
110	S <sub>13</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
111	S <sub>14</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0

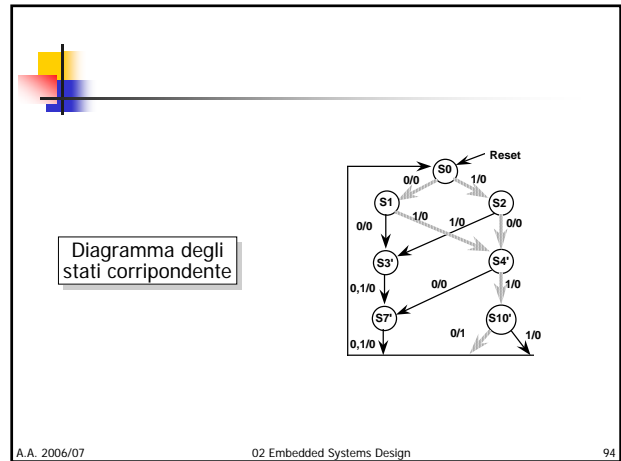
Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	0	0
0	S <sub>1</sub>	S <sub>3</sub>	S <sub>4</sub>	0	0
1	S <sub>2</sub>	S <sub>5</sub>	S <sub>6</sub>	0	0
00	S <sub>3</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
01	S <sub>4</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
10	S <sub>5</sub>	S <sub>11</sub>	S <sub>12</sub>	0	0
11	S <sub>6</sub>	S <sub>13</sub>	S <sub>14</sub>	0	0
000	S <sub>7</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
001	S <sub>8</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
010	S <sub>9</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
011 or 101	S <sub>10</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0
100	S <sub>11</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
110	S <sub>13</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
111	S <sub>14</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0

Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	0	0
0	S <sub>1</sub>	S <sub>3</sub>	S <sub>4</sub>	0	0
1	S <sub>2</sub>	S <sub>5</sub>	S <sub>6</sub>	0	0
00	S <sub>3</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
01	S <sub>4</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
10	S <sub>5</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
11	S <sub>6</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
not (011 or 101)	S <sub>7</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
011 or 101	S <sub>10</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0

Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	0	0
0	S <sub>1</sub>	S <sub>3</sub>	S <sub>4</sub>	0	0
1	S <sub>2</sub>	S <sub>5</sub>	S <sub>6</sub>	0	0
00	S <sub>3</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
01	S <sub>4</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
10	S <sub>5</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
11	S <sub>6</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
not (011 or 101)	S <sub>7</sub>	S <sub>9</sub>	S <sub>10</sub>	0	0
011 or 101	S <sub>10</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0

Tabella finale minimizzata

Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	0	0
0	S <sub>1</sub>	S <sub>3</sub>	S <sub>4</sub>	0	0
1	S <sub>2</sub>	S <sub>5</sub>	S <sub>6</sub>	0	0
00 or 11	S <sub>3</sub>	S <sub>7</sub>	S <sub>8</sub>	0	0
01 or 10	S <sub>4</sub>	S <sub>7</sub>	S <sub>10</sub>	0	0
not (011 or 101)	S <sub>7</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
011 or 101	S <sub>10</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0



Conclusioni:

- facile da capire e da realizzare
- limite: non sempre ottiene il minimo numero di stati!

Esempio: controllore di parità a 3 stati

Stato presente	Stato futuro		Uscita
	X=0	X=1	
S <sub>0</sub>	S <sub>0</sub>	S <sub>1</sub>	0
S <sub>1</sub>	S <sub>2</sub>	S <sub>2</sub>	1
S <sub>2</sub>	S <sub>2</sub>	S <sub>1</sub>	0

Non c'è modo di combinare S0 ed S2 basandosi sul criterio dello stato futuro!

Metodo della tabella delle implicazioni

Specifica:

- 1 ingresso X, 1 uscita Z
- generare 1 in uscita quando si è riconosciuta la sequenza 010 o 110 in ingresso

Tabella delle transizioni iniziale:

Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	0	0
0	S <sub>1</sub>	S <sub>3</sub>	S <sub>4</sub>	0	0
1	S <sub>2</sub>	S <sub>5</sub>	S <sub>6</sub>	0	0
00	S <sub>3</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
01	S <sub>4</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0
10	S <sub>5</sub>	S <sub>0</sub>	S <sub>0</sub>	0	0
11	S <sub>6</sub>	S <sub>0</sub>	S <sub>0</sub>	1	0

Enumerazione di tutte le possibili coppie di stati

Stati futuri per ogni combinazione di ingresso

Struttura dati semplicistica:  
 $X_{ij}$  e' sempre uguale ad  $X_{ji}$  (relazione simmetrica)  
 Inoltre, possiamo eliminare la diagonale (relazione riflessiva)

Tabella delle implicazioni

A.A. 2006/07 02 Embedded Systems Design 97

Come riempire la tabella:

Elemento  $X_{ij}$ : riga è  $S_i$ , colonna è  $S_j$

$S_i$  è equivalente ad  $S_j$  se le uscite sono le stesse e gli stati futuri sono equivalenti

$X_{ij}$  contiene gli stati futuri di  $S_i$  ed  $S_j$  che devono essere equivalenti se  $S_i$  ed  $S_j$  sono equivalenti

Se  $S_i, S_j$  hanno uscite diverse, si cancella  $X_{ij}$

A.A. 2006/07 02 Embedded Systems Design 98

Esempio:  
 $S_0$  va ad  $S_1$  con 0, ad  $S_2$  con 1;  
 $S_1$  va ad  $S_3$  con 0, ad  $S_4$  con 1;

Quindi  $X<0,1>$  contiene  $S_1-S_3$  (ingresso a 0)  
 $S_2-S_4$  (ingresso ad 1)

$S_0$       $S_1-S_3$   
            $S_2-S_4$   
            $S_1$

A.A. 2006/07 02 Embedded Systems Design 99

$S_2$  ed  $S_4$  hanno un diverso comportamento in uscita  
 Questo implica che  $S_1$  ed  $S_0$  non possono essere combinati

Tabella delle implicazioni iniziale

A.A. 2006/07 02 Embedded Systems Design 100

Primo passo di marcatura

Il secondo passo non aggiunge nulla  
 $S_3$  ed  $S_5$  sono equivalenti  
 $S_4$  ed  $S_6$  sono equivalenti  
 Questo implica che  $S_1$  ed  $S_2$  sono anche equivalenti!

A.A. 2006/07 02 Embedded Systems Design 101

Tabella delle transizioni minimizzata

Sequenza ingr.	Stato presente	Stato futuro		Uscita	
		X=0	X=1	X=0	X=1
Reset	$S_0$	$S_1$	$S_1$	0	0
0 or 1	$S_1$	$S_3$	$S_4$	0	0
00 or 10	$S_3$	$S_0$	$S_0$	0	0
01 or 11	$S_4$	$S_0$	$S_0$	1	0

A.A. 2006/07 02 Embedded Systems Design 102

