Lezioni del 19/03/2014, 20/03/2014 e 21/03/2014.

SD: Design Representation

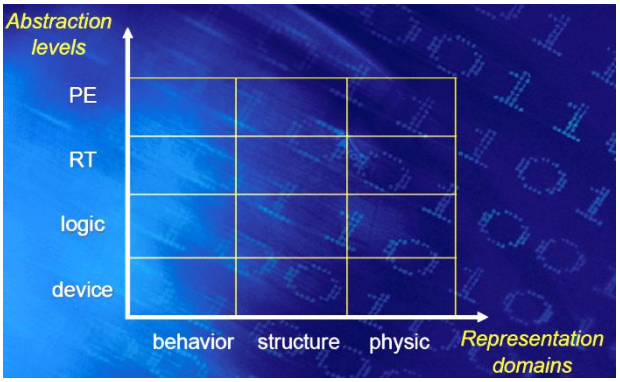
The representation matrix

Ci soffermiamo sul sottospazio dove sulle x ho il dominio di rappresentazione (=particolare aspetti che ci interessa guardare) e sulle y il livello di astrazione (per quel particolare dominio andremo a vederlo al livello di dettaglio che ci interessa).

La ratio che ci sta dietro è che quando dobbiamo trattare sistemi complessi dobbiamo necessariamente lavorare su livelli di astrazioni diversi, così è possibile riuscire a fronteggiare la complessità.

***Astrazione**: un modello semplificato di un sistema o di un altro modello che mostra solo alcuni aspetti, togliendo i dettagli che non interessano e concentrandosi su ciò che si ritiene importante.*

Storicamente anche il design hardware si è evoluto man mano che la tecnologia avanzava, all'inizio disegnavo i singoli transistor, poi ho aumentato la complessità dei blocchetti elementari-> sono passata da transistor alle porte logiche e poi ai sommatori o altre cose. Storicamente è stato fatto il bottom-up, ma noi normalmente ragioneremo a top-down.



Fonte www.testgroup.polito.it

Possibili domini di rappresentazione:

- **Dominio comportamentale** (behavior): vogliamo descrivere il comportamento del sistema senza dire com'è fatto, tipicamente descrivo le proprietà del sistema che definiscono cosa e sotto quali circostanze il sistema opera.
- **Dominio strutturale**: qui siamo interessati alla struttura del sistema, com'è fatto, qual è la sua topologia, come i blocchetti sono interconnessi fra di loro dove i blocchetti dipendono dal livello di astrazione in cui sono. Questa descrizione è fatta indipendentemente dalla tecnologia che userò per implementarlo, e.g. a me interessa che quel sistema è un sommatore, non importa come è fatto.
- **Dominio fisico**: è il dominio strutturale ma è technology dependent, descrivo fisicamente quali e come sono fatti i blocchi che utilizzerò.

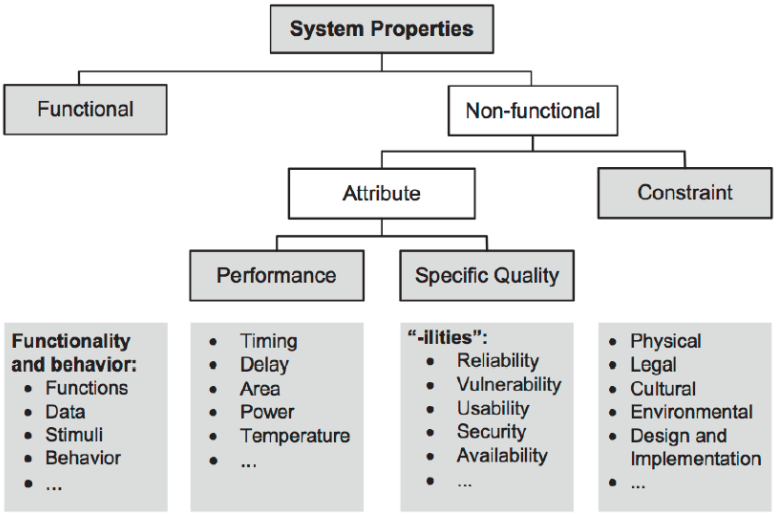
Ok, adesso li approfondiamo singolarmente:

Behavior domain

In questo dominio siamo sempre interessati a descrivere le caratteristiche funzionali.
Functional property: descrive gli ingressi e le uscite, descrive quali responses il mio sistema fornisce sotto certi stimoli.

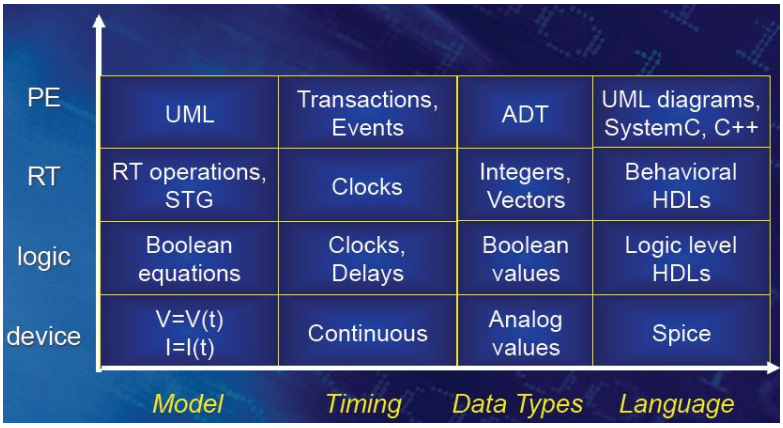
Non-functional property: in genere è un attributo (in termini di prestazioni o di specifiche qualità) o un vincolo del mio sistema.

Queste cose sono sempre più importanti perché ad esempio l'invecchiamento del mio sistema è molto importante, come riuscire in fase di progetto riuscire a capire come evolveranno le prestazioni del mio sistema.



Per descrivere il comportamento a sua volta ho bisogno di 4 sotto aspetti:

y= ***processing element***: sono a un livello d'astrazione così elevato che hardware e software sono tutti insieme. A questo livello il tempo serve solo per dare una cronologia, sono solo scambi di eventi.



Fonte www.testgroup.polito.it

Y= ***register transfer***, trasferimenti fra registri. Qui in termini di timing c'è il clock, qualcosa che temporizza la sequenza di eventi nel mio sistema, inizio ora a introdurre misure di tempo. A questo livello descrivo il mio sistema attraverso State Transition Graph (STG). I nodi rappresentano gli stati in cui il sistema può trovarsi, per ogni stato specifico i valori delle uscite. È un grafo che rappresenta come il sistema evolve in funzione degli ingressi. A questo livello tratto solo interi e vettori e in termini di linguaggio uso gli HDL (Hardware description language,e.g. VHDL o verilog, linguaggi pensati per rappresentare l'hardware).

Y=***livello logico***: il comportamento del sistema è trattato esclusivamente in termini di equazioni booleane (per lo più oggi la maggior parte dei programmi rappresentano le booleane con RDD, albero a decisione binario). A questo livello iniziano a entrare in gioco anche i delay di cammini, gli ADT's sono solo valori booleani.

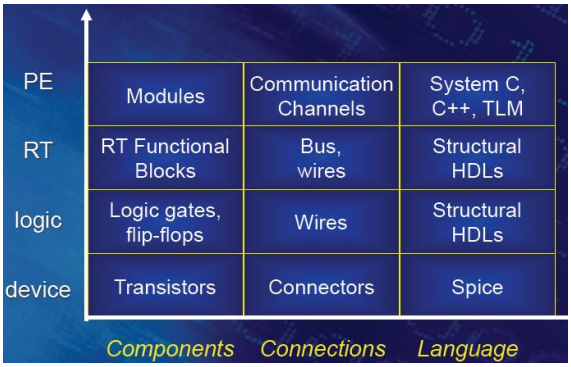
Y=***livello di dispositivo***: descrivo il comportamento in funzione di andamento di tensioni e di correnti. I valori non sono più digitali ma analogici, in genere i linguaggi d'utilizzo sono basati su spice.

Structural domain

y=Processing element: i componenti sono i moduli, per le connessioni ho i canali di comunicazione, come linguaggio uso C, C++, etc.

y=RT, ho i blocchi funzionali a livello RT, le connessioni vengono fatte attraverso i bus, ho generici registri e generici sommatori, non ho ancora specificato che è l'elemento XXX prodotto dalla YY. Il bus è un'insieme

di fili in cui viaggia in parallelo la stessa informazione. In genere uso il bus se ho n segnali, un wire (filo) per un sono bit.



y=logic, i blocchetti sono le porte logiche e i flip flop. A questo punto non è importa cosa fa il circuito, mi dice com'è fatto il mio circuito, è una descrizione strutturale a livello logica. Poiché queste rappresentazioni sono molto comuni, si chiamano netlist (descrizione strutturale a livello logico). Ci sono standard internazionali per rappresentarlo in modo compliant. A livello di connessioni ho solo dei fili, niente più bus.

y=device, i componenti sono i transistor, le connessioni sono i connettori.

Physical domain

Come prima ma è tecnology-dependent. Specifico quali sono i componenti che uso, come sono marchiati, dove sono i piedini.

Some relevant operations

Ogni quadratino della mia matrice corrisponde a una specifica descrizione del progetto, passare e camminare sulla matrice vuol dire fare un passo di progetto. Qualsiasi spostamento in qualsiasi direzione dall'elemento in posizione [0,0] va verso la sintesi di progetto. Progettare vuol dire aggiungere dettaglio. Analisi invece vuol dire andare dal basso a destra verso qualsiasi direzione. Inoltre minimizzare vuol dire rimanere nel rettangolino ma minimizzare la funzione costo. In questo corso non si va mai fino al livello device.

SD: Typical design processes

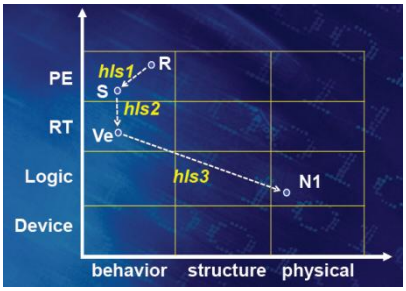
Criteri per classificare il design process:

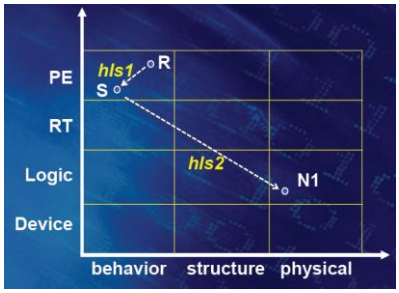
- Insieme di **building blocks** considerati 'elementari' per quella specifica implementazione.
- Il **punto di partenza**. Può essere high level synthesis (parto da alto a sx), oppure RT level synthesis se (parto da behavior -> RT), oppure logic level synthesis se parto da (behavior-> logic).
- **Sintesi automatica o sintesi manuale**.

Automated (HDL-based) design process

Prendiamo due diversi approcci, entrambi hanno lo stesso punto di partenza: i requisiti utente. Per esempio si chiede di progettare un sistema che abbia in ingresso due variabili di tipo intero e un segnale di controllo. Il sistema deve calcolare o il massimo o la somma in base al valore del segnale di controllo. Entrambi gli approcci hanno in comune che da questa descrizione a parole la esplicito in C++. A questo punto le strade si dividono.

- C-to-silicon di Cadence, prende in ingresso descrizione in C++ e scendo di livello aggiungendo dettagli arrivando a descrizione verilog. A questo punto prendo un altro tool di sintesi che prende questa descrizione e in uscita ottengo la netlist.
- Vivado di Xilinx, fornisco in ingresso in C++ e in uscita mi fornisce la netlist.





Cadence vs Xilinx, fonte www.testgroup.polito.it

Storicamente i primi layout erano fatti con i tecnigrafi, poi i tool si sono evoluti man mano. Oggi a mano a livello industriale non si fa più nulla, la progettazione è sempre fatta con tool di sintesi automatica, fino a 2-3 anni fa lo standard era che si partiva a livello RT per darlo da mangiare ai tool di sintesi. Ultimamente sono usciti altri tool che non partono da RT ma da C++, a un livello più alto. Però i tool che partono da RT sono ancora migliori e più ottimizzati di quelli che partono da C++. Non si tocca mai a mano l'output della sintesi perché si farebbero solo casini, prima devo metterlo a posto a livello RT o C++ e poi lo risintetizzo nuovamente.

Visto che oggi progettare hardware significa partire dal C++ e poi il tool fa il resto, potrei dedurre che se conosco bene il C++ allora sono anche un buon progettista hardware. Questo è sbagliato, i tool sono ottimizzati a patto che il tool riesca a comprenderlo correttamente -> bisogna conoscere l'hardware perché chi fa solo software programma in modo diverso (e.g. uso la ricorsione). Il vantaggio dei tool è che non fanno quasi mai errori e ottimizzano le prestazioni (a mano sarebbe molto più problematico).

A livello RT invece sono io che manualmente parto dalla specifica dell'utente e ottengo la descrizione RT in VHDL, poi la do al tool di sintesi e ottengo la mia netlist.

Manual design process

Il modo di progettare dipende da molte cose: building blocks (logic level or RT), se voglio ottimizzare i ritardi, l'area o la potenza, etc.

Digital network modeling

Combinational vs. Sequential networks

Modello: semplificazione di un'altra entità (oggetto fisico o a sua volta un altro modello).

È possibile distinguere un sistema combinatorio da uno sequenziale solo in termini di comportamento?

Sistema combinatorio (def. 1): iff i suoi POs sono solo ed elusivamente determinati dai PIs in quel dato istante.

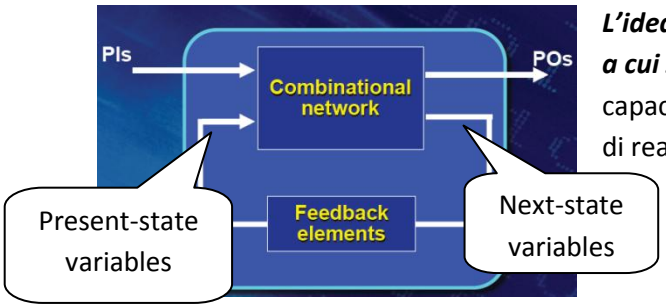
Appunti del corso "Algoritmi e calcolatori".

A livello strutturale invece:
Sistema combinatorio (def. 2): iff la netlist non ha loop element *(elementi di reazione, cioè se parto da A e attraverso un percorso chiuso ritorno in A).*

Analogamente per un circuito sequenziale. Il capire quanti anelli di reazione sono presenti è fondamentale. Un circuito sequenziale è sempre riconducibile a una rete combinatoria associata a degli elementi di feedback.

Concetto di stato

Per un circuito sequenziale, il fatto che mi devo ricordare gli ingressi precedenti vuol dire che da qualche parte li devo memorizzare, devo riuscire a memorizzare l'evoluzione-> questo viene affidato agli stati interni.



L'idea è che in ogni istante il sistema sia in un ben preciso stato a cui sono associate in modo univoco le variabili interne. La capacità di memorizzazione è direttamente legata agli elementi di reazione.

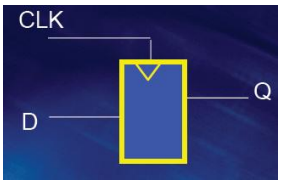
Fonte www.testgroup.polito.it

Variabili di stato*: un circuito sequenziale ha tante variabili di stato quanti sono gli anelli di reazione. La singola variabile di stato è un filo-> a livello logico assumerà solo i valori 0 o 1 -> se in un circuito ho n variabili di stato, allora potrà assumere al massimo 2n stati.*

Questi elementi di reazione si possono distinguere in asincroni e sincroni.

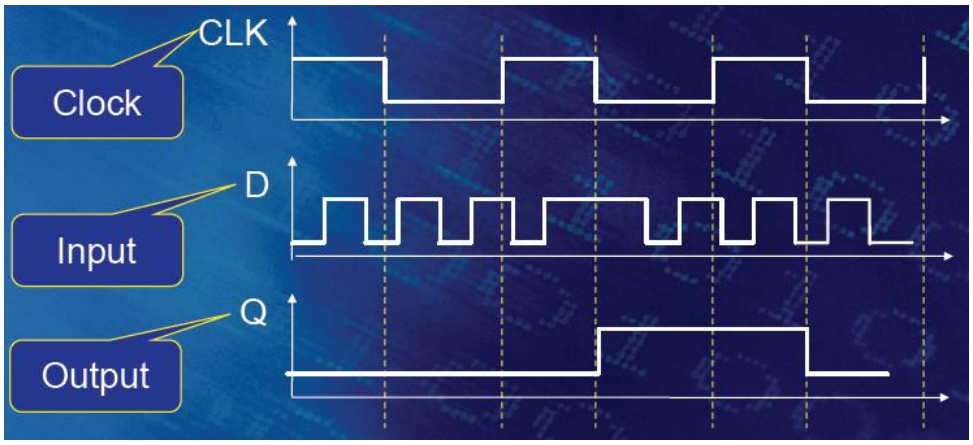
Sequenziale asincrono: iff l'anello di reazione è soltanto un filo**.** Asincrono perché il segnale che ho in uscita tornerà in ingresso dopo un po' (questo tempo dipende dal delay), non è sincrono a nulla.
Sequenziale sincrono: l'elemento di reazione è sempre eseguito attraverso un flip-flop (dispositivo in grado di memorizzare un bit). Il comportamento del flip-flop è sempre controllato dal clock, le informazioni attraverso il flip-flop fluiscono solo quando il clock è attivo, non ho più un flusso continuo di informazioni, il futuro diventa presente non grazie al delay (delle porte, del filo, etc.) ma quando il clock lo decide.

FLIP-FLOP



CLK-> segnale temporizzazione,
D->ingresso,
Q->uscita.

In questo caso il clock è attivo sui fronti di discesa, quando il clock passa da alto a basso prendo l'uscita di D in quell'istante e quindi io mantengo l'uscita a 1 indipendentemente da D, poi quando il clock passa di nuovo da alto a basso controllo il valore di D-> è basso e quindi l'uscita sarà 0. C'è un evento del clock che triggera e campiona l'uscita, gli ingressi cambiano in continuo, le uscite cambiano solo in concomitanza con l'evento di trigger del clock.



Fonte www.testgroup.polito.it

Spesso s'indica con ***state register*** (registro di stato) l'insieme di flip-flop.

FINITE STATE MACHINES

Un sistema sincrono con un numero finito di flip-flop in gergo viene definito FSM (Finite State machine).
Tutte le FSM, anche se le specifiche non le prevedono, DEVONO avere:

- ***Reset signal***, particolare segnale in ingresso caratterizzato dalla massima priorità.
- Un particolare stato, ***state reset***, nella quale la macchina si porta subito dopo essere entrato il reset signal, indipendentemente dallo stato precedente. (asserted-> quando passa il segnale di reset, indipendentemente se da alto a basso o vice versa).

Perché? Se ho un circuito con 1000 flip-flop, lo alimento, quali valori assumono all'inizio i miei flip-flop? Non posso saperlo, all'accensione non è noto a priori che valore assumono (è in pratica casuale), se ho 1000 flip-flop allora ho 2^1000 stati possibili, quando l'accendo in quale stato va il mio circuito? I do not know. L'unico modo che ho è mandarlo in uno particolare stato noto attraverso un segnale dall'esterno. Devo riuscire a forzarmi in un particolare stato, altrimenti sarei inutilizzabile. Inoltre se il mio circuito va in uno stato anomalo, allora lo forzo a uno stato noto da cui ripartire. Tutte le nostre progettazioni partiranno da un reset.

Moore vs Mealy machine

In un circuito sequenziale, posso dividere la rete combinatoria in due parti: quella che mi tira fuori le uscite e quelle che mi genera le variabili di stato future. Quindi considerando il primary output network, si possono distinguere fra:

Macchine di Moore: cambiano le uscite solo quando cambia lo stato della macchina, indipendentemente dagli ingressi. Non c'è una connessione diretta fra PI's e PO's.
Macchine di Mealy: le uscite dipendono dagli ingressi e dallo stato, è possibile che l'uscita cambi cambiando l'ingresso.

In questo corso utilizzeremo per lo più macchine di Moore.

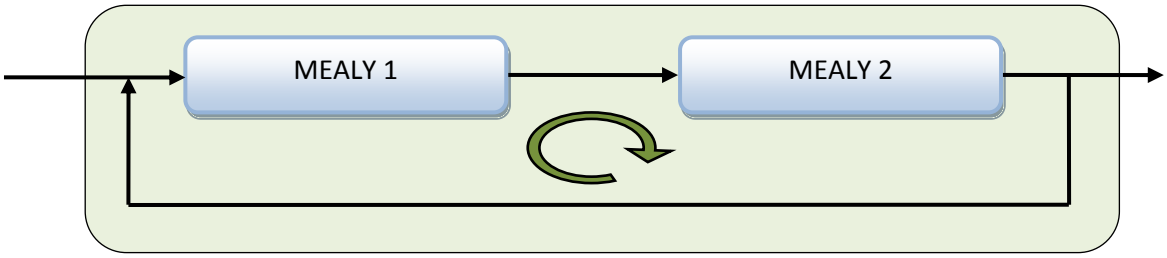
NOTE:

In questo corso non progetteremo circuiti asincroni, ma solo sincroni con macchine di Moore, sia perché questo è il primo corso, sia perché le design rules prevedono che i circuiti siano TUTTI sincroni. Perché in un

circuito asincrono tutto dipende da quanto velocemente il segnale attraversa il filo, non so quanto velocemente cambierà di stato, non è predicibile a priori. Per cui eviteremo di progettare circuiti asincroni.

Ok, ma in termini di velocità un circuito asincrono è più veloce di quello sincrono, allora perché non usarlo? È più veloce ma non so misurare di quanto, inoltre riuscire a fare i test di fine produzione di un circuito asincrono è quasi del tutto impossibile, tutte le regole di collaudabilità prevedono dei circuiti sincroni. Inoltre progettare manualmente circuiti asincroni è molto più difficile di quelli sincroni.

Se prendo due macchine di Mealy in cascata con un filo di feedback, sequenziale, è perfettamente sincrono?

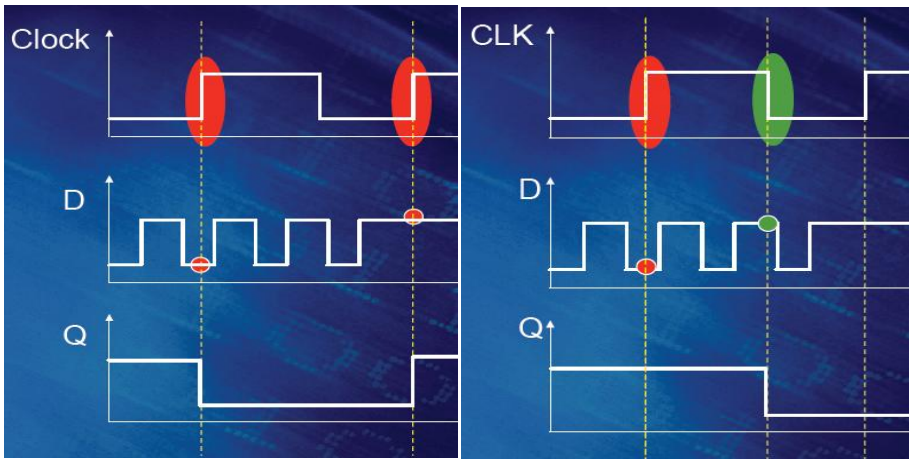


Questo è un circuito asincrono, perché cambiando l'ingresso cambio tutto quanto e faccio un loop senza trovare neanche un flip-flop. Invece se collego in cascata delle macchine di Moore non ho questo problema.

I/O clustering

È fondamentale distinguere fra i tipi di segnali. Quando si progetta bisogna essere ordinati, un segnale di data rimane solo segnale di data, non può diventare un clock o un control signal.

Come evento di trigger si può prendere il fronte (di salita o di discesa) (edge trigger) oppure l'impulso alto o basso (pulse), dove un fronte campiona e l'altro modifica l'uscita. Noi progetteremo indipendentemente da questo, non importa a livello strutturale.



Positive edge triggering vs. pulse triggering (fonte: testgroup.polito.it)

Se prendo come evento trigger il pulse e il fronte di campionamento avviene contemporaneamente con una modifica del valore di data, quale campione prendo? Il costruttore di dice che affinché funzioni il dato deve essere stabile quando arriva al clock (a 1 o 0 da sufficiente tempo), inoltre il dato deve essere stabile

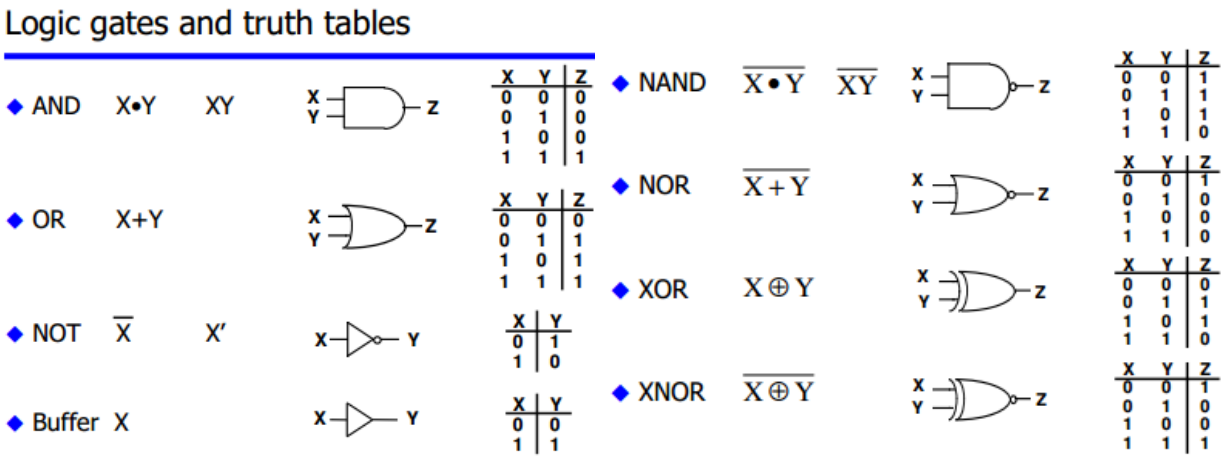
per abbastanza tempo anche dopo l'evento di trigger del clock (set-up e hold times). Il costruttore mi garantisce il funzionamento se rispetto queste tempistiche.

Control inputs: dall'esterno forniscono informazioni su come il sistema si deve comportare. I segnali di controllo vengono divisi fra quelli sincroni e asincroni (rispetto al clock).

Segnale di controllo asincrono: non appena diventa attivo,indipendentemente dal clock, il suo effetto è immediato (e.g. segnale di reset).

Segnale di controllo sincrono: diventano attivi al prossimo evento trigger del clock (edge o pulse). Ci sono anche i reset sincroni (il reset asincrono serve e non si discute, perché serve anche uno sincrono? Perché se il mio sistema deve essere perfettamente sincrono (esclusa inizializzazione e fault), allora anche il reset "normale" deve essere sincrono).

Cenni di algebra booleana



Fonte: https://courses.cs.washington.edu/courses/cse370/09au/pdfs/lectures/04-Logic%20gates.pdf

EXOR (or XOR)

Torna 1⇔ ho un numero dispari di 1 in ingresso. (se ho 3 ingressi, torna 1 se ho 001, 010, 100, 111). In particolare se x=0, allora l'uscita=y; se x=1, uscita=y'. Posso vedere l'exor come un complementare controllato.
Oppure posso leggerla dicendo che se x=y-> uscita=0, else uscita=1 -> è un comparatore fra due bit.

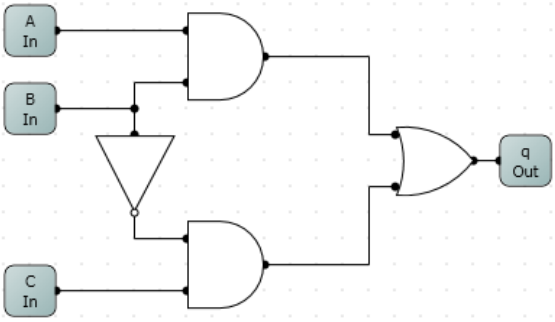
Si chiama or esclusivo, la differenza è nel caso in cui i due ingressi sono entrambi 1, è la differenza fra vel (O normale, almeno uno dei due), e aut (or esclusivo, solo uno dei due).

Immaginiamo di dover sommare due numeri binari:
0101 +
1100=
1001 Senza contare il riporto, di fatto è come un exor -> è un sommatore in CA2 senza riporto.

La tabella di verità è un modo esaustivo per rappresentare la mia funzione, la mappa di Karnaugh è anche lei una rappresentazione esaustiva ma non è più in 1D ma sue 2D ->diventa molto utile quando progetto a

mano. Il loro trucco è che passando riga o colonna allora cambia un solo bit per volta (cioè hanno distanza unitaria).

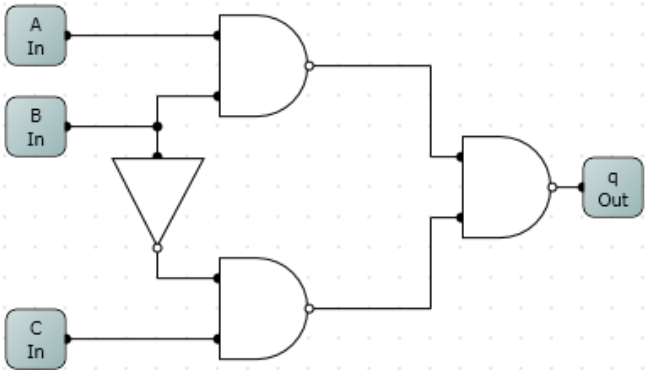
Domanda: se devo rappresentare un qualsiasi funzione booleana, devo avere per forza tutte quelle basi? Qual è l'insieme minimo di funzioni booleane che mi serve? Il punto è che la funzione nand da sola è sufficiente per rappresentare qualsiasi funzione booleana, oppure posso usare il set not, and, or. Supponiamo di dover implementare una cosa del genere:



A	B	C	q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Circuito 1 con rispettiva tabella di verità (tool: Logical Circuit)

Adesso nessun problema, ma negli anni 70 avevo dei circuiti integrati con magari 4 porte or o and e uno con 6 inverter. Quindi mi servivano 3 circuiti integrati di cui utilizzavo pochissime porte, però quel circuito è analogo a ques'altro-> stessa funzione ma ho bisogno di un solo componente.



A	B	C	q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Circuito 2 con rispettiva tabella di verità (tool: Logical Circuit)

Boolean functions in circuit design

Geometric interpretation of Boolean algebras

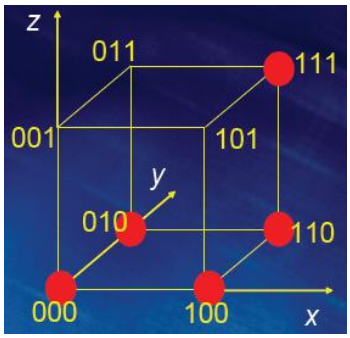
Una funzione booleana che opera su n variabili la posso rappresentare come appartenente a uno spazio a n dimensioni. Se ho un solo ingresso mi basta una dimensione (1 o 0), se ho due ingressi ho bisogno di 2 dimensioni (2 valori per parte), se ho tre ingressi ho un cubo dove ai vertici ci sono le possibili combinazioni di valori che assumono tre variabili booleane. Per 4 dimensioni immagino di avere 2 cubi uno dentro l'altro.

Values taken by functions

Una funzione booleana può assumere 0 o 1 o non essere specificato. Per convenzione i vertici nei quali la funzione vale zero compongo l'***off-set*** (valore nullo), dove vale 1 è l'***on-set***; dove non è specificato ***don't-care-set***. Perché m'interessano i don't care set? L'obiettivo finale è arrivare a progettare un circuito a costo minimo. Ci possono essere combinazioni di variabili le quali so a priori che non capiteranno mai, e.g. 4 segnali in ingresso e uno in uscita. Ogni segnale è un bit, poi so che dall'esterno arrivano numeri decimali <=9, quindi non m'interessa progettare un sistema che riesca anche a operare con 11-> se devo minimizzare il costo allora devo implementare solo ciò che mi serve.
Se il don't care set non contiene elementi allora la funzione è completamente specificata, altrimenti no.

How representing functions?

In che modo posso rappresentare I valori che la mia funzione assume in ciascun vertice?



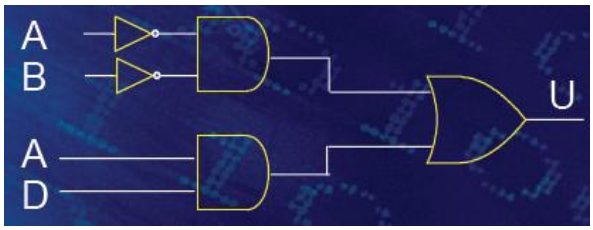
Fonte testgroup.polito.it

Prima idea: rappresento solo i punti dove f=1, dove non li rappresento allora f=0.
esempio f=0 when x=0 & y=0 & z=0, x=0… ma è lungo e barboso.
Proviamo a semplificare: ho la funzione and, or, x (cioè x=1), x' (cioè x=0) -> diventa un po' più semplice.
f=x'y'z'+xy'z'+… ma siamo ancora distanti.
al posto di rappresentare singolarmente i 4 vertici inferiori, non posso rappresentare tutta la faccia in modo univoco e più veloce? In comune hanno che z=0, quindi è un z'. analogamente per indicare uno spigolo posso scrivere xy (x=1 & y=1) anziché elencare i due spigoli. Quindi la mia funzione diventa F=z'+xy. Ricordiamoci che il mio obiettivo è arrivare ad avere un hardware che faccia quello che voglio a costo minimo. Inoltre tra un vertice e l'altro tra loro adiacenti cambia solo un bit per volta.

Dalla funzione al circuito

L'operazione di somma corrisponde all'or, l'and al prodotto e il not al complemento. Esempio:

U=A'B'+AD ⇔



Applicando i teoremi dell'algebra booleana, posso ottenere circuiti equivalenti.
esempio
U=A'B'+AB=exnor. Tutti questi circuiti sono equivalenti ma avranno costi minori.

Osservazioni

L'espressione f=x'y'z' + x y'z' + x y z' + x'y z' + x y z (1) nella quale rappresento o elenco tutti i vertici del mio cubo in cui la funzione vale 1 viene anche definita ***minterms.***
Nell'espressione finale F=z'+xy (2) la funzione è rappresentata utilizzando solo ***sotto-cubi*** (facce e/o spigoli).
Se non ragionavo nello spazio 3D riuscivo comunque a dimostrare che (1) e (2) sono equivalenti? Sì, facendo tutti i passaggi. La difficoltà sta nell'esperienza, applico le regole di riscrittura, il rischio tipico è di fare un po' di passaggi e tornare al punto di arrivo, soprattutto se inizio a pasticciare a caso.

L'espressione (2) è ***minimale*** cioè non può essere ulteriormente semplificata (è un minimo locale, non so se c'è un minimo assoluto partendo da un'espressione completamente diversa). (1) e (2) sono equivalenti.

Implementazione pratica

Per (2) mi serviranno un inverter, una porta and e una porta or, oppure utilizzando (1) avrei 8 inverter, 5 and e un or a 5 ingressi. Indipendentemente dalla funzione di costo, la (2) sembra più ragionevole.

Concetto di costo: il costo dell'implementazione è proporzionale al costo dell'espressione da implementare.

Disegnare manualmente circuiti combinatori

Premessa: non c'è un'unica soluzione, ci sono vari approcci. Gli approcci per circuiti combinatori sono diversi, dipende da quant'è grosso il circuito e dal tipo di sintesi (piccoli o grandi).

Nel caso di sintesi automatica, il circuito che il tool tirerà fuori sarà combinatorio solo se io programmatore VHDL lo faccio nella maniera giusta (altrimenti correi il rischio di avere un mare di flip-flop). Viceversa, se la sintesi è manuale e il circuito è piccolo utilizzeremo equazioni booleane, se è grande utilizzeremo una sintesi a RT.

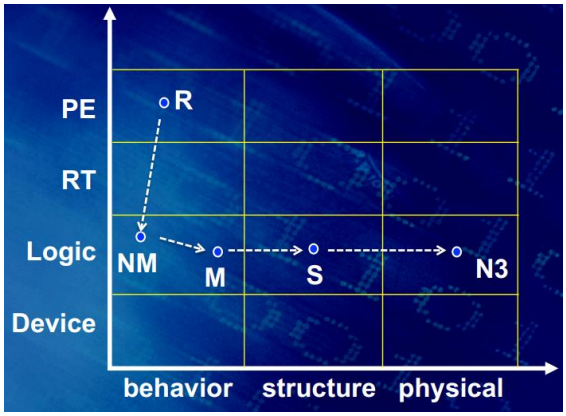
PROCESSI DI SINTESI MANUALE PER CIRCUITI VERAMENTE PICCOLI

Il 'veramente piccoli' vuol dire circuiti che hanno al massimo 5 ingressi, altrimenti non riusciamo a fare mappa di Karnaugh.
Per questi ci sono due approcci: quello basato sulla mappa di Karnaugh (garantisce costo e ritardo minimo) e poi ci sono altri approcci dove non è minimizzata la funzione di costo ma se ho 8 ingressi riesco ancora uscirne vivo.

Approccio mappa di Karnaugh:

Se lo uso minimizzo il ritardo, max 6 Pis, difficile minimizzare l'area, (fino allo step 2 incluso siamo ancora a livello comportamentale).

- ***Requisiti utente***: punto di partenza.
- ***STEP 1***: dalle req devo ottenere un'equazione booleana non minima (per esempio su di una mappa di Karnaugh).
- ***STEP 2***: a partire dalla rappresentazione esaustiva cerco di ottenerne una minima. (Minor costo della funzione, minor costo dell'hardware) (lo faccio cercando i sottospazi (k-cubi) dove la funzione assume lo stesso valore).



Fonte: testgroup.polito.it

- ***STEP 3***: passo da behavior a structure (***library binding***) devo ottenere una netlist nel dominio strutturale.

- ***STEP 4***: ***Technology binding***, passo da strutturale a fisico, sostituisco generica porta or con la specifica or della libreria che ho scelto.

In questo corso ci fermiamo allo step 3, da 1 a 2 basta conoscere un po' di trucchi, la difficoltà è mappare la specifica in valori booleani non minimizzati.

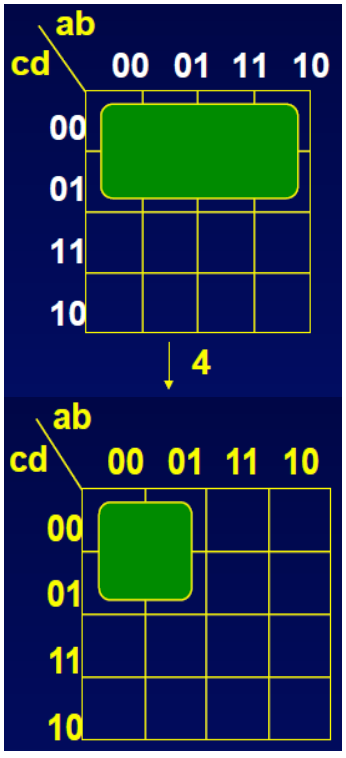
Covering Karnaugh maps

Analizziamo solo come passare da mappa di Karnaugh all'ottenere una funzione booleana il più possibile minimizzata (ovvero come la si copre).
Per ora ci occupiamo di circuiti con pochi ingressi e un'uscita.

*Chiamiamo **k-cubo** o uno sottospazio a k-dimensioni un sottoinsieme di 2^k vertici (solo spigoli, facce o cubi) nei quali **n-k** variabili d'ingresso assumono lo stesso valore costante.*

Implicante di una funzione f: un k-cubo che non include alcun vertice appartenete all'off- set (solo 1 o don't care).

L'implicante viene rappresentate con il prodotto, fra tutti gli implicanti c'è ***l'implicante principale*** il quale ***non è incluso in nessun altro implicante*** (è il più grande) ***e non copre solamente i don't care*** (non m'interessa, sarebbe solo hardware sprecato). Qual è il legame fra spigoli e facce con la mappa di Karnaugh? Un k-cubo sulla mappa di Karnaugh con k=1 (uno spigolo) è rappresentato da due celle adiacenti (in verticale o in orizzontale) sulla mappa di Karnaugh perché cambio una sola variabile passando fra due celle (e quindi anche fra i due vertici adiacenti). Se invece cerco le facce, allora devo cercare 4 celle-> posso prenderle 4 in orizzontale adiacenti dove una variabile rimane costante e le altre cambiano, oppure posso prendere il quadrato 2x2 dove rimane costante a o b.



Con la mappa di Karnaugh celle che sono fisicamente adiacenti allora lo sono anche logicamente, fra le due cambia un solo valore -> hanno distanza 1 -> è uno spigolo. L'adiacenza logica viene riportata anche graficamente. Tutto si riduce a cercare nella mappa gli uni tale che questi uni formino dei k-cubi. Il mio obiettivo è rappresentare insiemi composti da 1, 2, 4, 8 uni i quali corrispondono a spigoli, facce o vertici. Il cubo devo immaginarlo su di una superficie chiusa, per cui anche quelli su prima riga possono stare insieme a quelli della ultima riga, o prima colonna e ultima colonna.

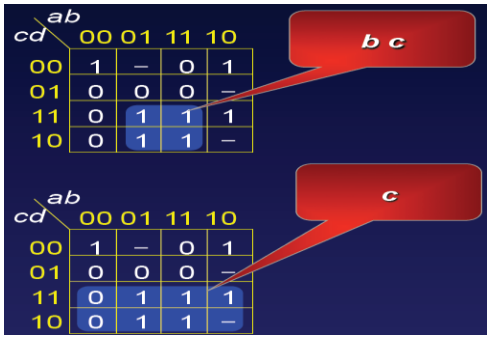
(codice Gray: disposizione di valori non in binario puro ma con distanza pari a 1). Usando i k-cubi riesco a trovare le funzioni di costo minimo, piuttosto che arrivarci con i teoremi di algebra booleana. C'interessano gli implicanti principali perché sono quelli più efficienti, noi vogliamo avere il numero minimo di implicanti per coprire tutti gli 1.

Fino a 4 variabili in ingresso, ne metto 2x2, e se ne ho 5? Immagino di avere due mappe 2x2 e poi disegnandole le affianco.

Somma completa di una funzione: *la somma di tutti gli implicanti principali*

della funzione.
Copertura di una funzione: *data f, la sua copertura è una funzione c che copre tutti gli 1 di f, non copre nessun 0, e dove f=don't care allora c può assumere un qualsiasi valore (faccio quello che mi conviene).*
Questo perché ho scelto di rappresentare la funzione con gli 1, altrimenti avrei potuto rappresentarla con gli 0, è una convenzione. Insieme minimo di implicanti per ricoprirlo.

Copertura non ridondante: *iff togliendo un k-cubo qualsiasi che la compone non si ottiene più una copertura di f.*
Teorema: una funzione di costo minimo è sempre ottenuta come somma di implicanti principali (una funzione di costo è tale che aggiungendo una porta o un ingresso allora aumento la funzione di costo).



Fonte: testgroup.polito.it

Corollario: se una sop è minima, allora ogni implicante è principale.

Proprietà: se cancello una variabile da un implicante principale di una funzione, allora non ottengo più un implicante. Cancellare una variabile vuol dire settare quella variabile a zero. Per esempio passo da bc a c, allora c coprirà anche 0 -> non è più un implicante.

Boolean representation

Le funzioni booleane sono rappresentate usando i vertici con 1 e i don't care set.

Problemi: capire se due espressioni booleane sono equivalenti (***equivalent checking***); trovare una rappresentazione equivalente a quella data con costo minimo; garantire che dato un metodo per rappresentare la funzione, tutte le funzioni hanno una rappresentazione univoca (***canonicità***).

Ci sono 4 possibili rappresentazioni:

- ***Rappresentazione esaustiva***: è una rappresentazione esponenziale (se no n ingressi, ho 2^n possibili combinazioni). È una rappresentazione che non scala (non riesco a usarla allo stesso modo se aumento moltissimo il numero di ingressi).
  Per questa ci sono due possibili rappresentazioni: le tavole di verità e le mappe di Karnaugh. Nel secondo caso l'etichettatura delle righe e delle colonne è fatta in modo che celle logicamente vicine sono anche fisicamente adiacenti. Questo viene ottenuto utilizzando il codice Gray. In generale dove c'è un don't care metto un trattino. Si usano tipicamente per al max 5 ingressi.
- ***Rappresentazione compatta***: sfrutto il concetto di copertura, usa un'altra funzione che copra tutti gli uni ed eventualmente dei don't care. Anche per questa sono possibili due diversi tipi di rappresentazioni: attraverso espressioni booleane o attraverso i cubi. Nella prima rappresento i singoli vertici in modo algebrico.
  Attenzione: una funzione booleana può essere rappresentata da un numero infinito di espressioni. E.g. a+a=a. provare a dimostrare che le rappresentazioni sulle slide sono

equivalenti. Uso questa rappresentazione nella sintesi manuale e in letteratura.
L'alternativa è usare i cubi e posso scegliere se farlo come termini di prodotto (è sempre un SOP sum-of-product f=x'yz+z'y), o in termini di sum term (prodotto di somma, es. f=(x+y)(z'+xy)).
Oppure la funzione è rappresentata attraverso k-cubi, quest'ultimo è il modo più comune per scrivere programmi da dare in pasto ai tool di EDA.

- ***ITE (if-then-else):*** una funzione la posso rappresentare come una serie di istruzioni if-then-else annidiate. Esempio:
  ab⇔ ITE(a,b,0) : if a, then b, else 0. (se a è vero l'uscita vale b).
  A+b⇔ ITE (a,1,b).
  Questa si usa nella costruzione di BDD. La funzione booleana viene rappresentata nel calcolatore con un albero/grafo, se vero vado a sx, altrimenti vado a dx. la ITE può aiutare a rappresentarla.
- Grafi.

Date due espressioni, come ne verifico l'equivalenza? O a carta a mano mano sfruttando le regole di riscrittura oppure in modo automatico con dei tool che vengono chiamati tautology checkers o theorem provers (programmi che dimostrano teoremi).

Canonicità: la rappresentazione di una funzione booleana in SOP è canonica ⇔ è rappresentata in termini di minterms, ovvero elenco sempre e solo i singoli vertici.

Circuiti a più uscite

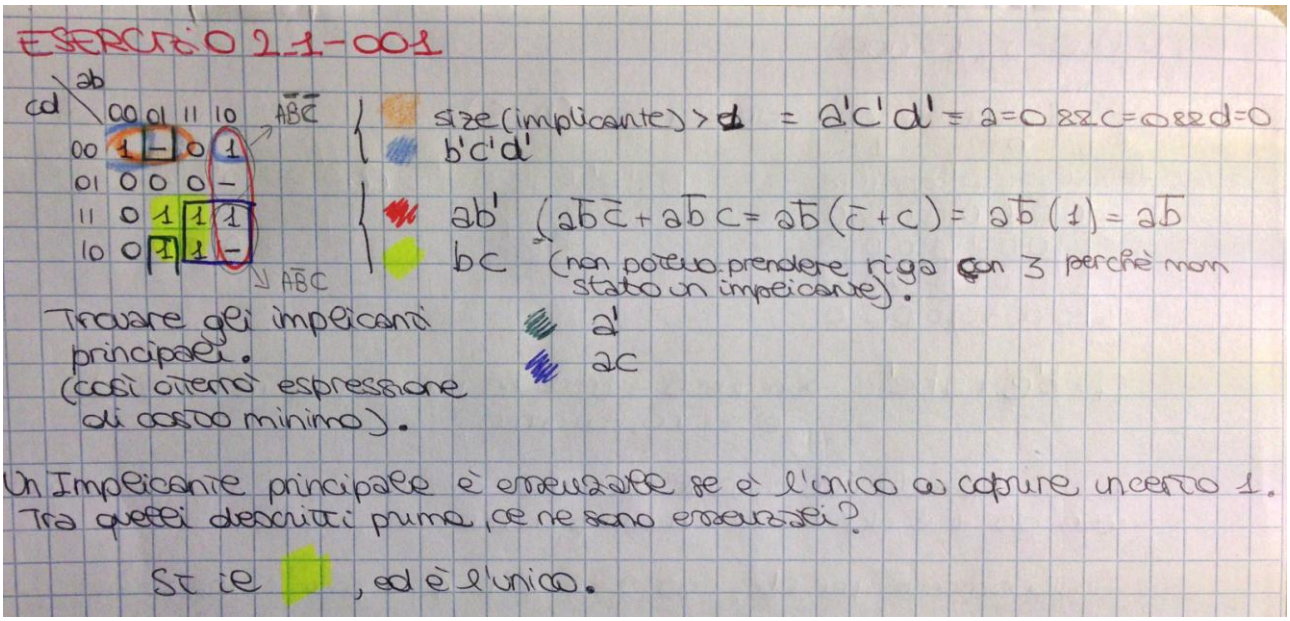
Ci sono dei metodi per ottimizzare le mappe di Karnaugh con uscite multiple: tratto i singolarmente le varie mappe ottimizzandole e poi le unisco. Ok, però se non avessi avuto un implicante principale ma un normale implicante sarebbe stato in comune alle due mappe-> il costo finale è minore della somma dei due singoli costi precedenti ottimizzati. Se manualmente voglio ottimizzare espressioni con più uscite, devo usare tecniche più sofisticate. In questo corso comunque se dobbiamo trattare più uscite insieme, le trattiamo indipendentemente e ci freghiamo che complessivamente ci sono soluzioni più economiche.

ESERCIZIO 1: progettare un circuito combinatorio a 1 uscita e un input a 4 bit il quale quando riceve una cifra esadecimale, allora fornisce sull'uscita 1 ⇔ il numero in ingresso è <4 o >8 (comparatore a due soglie).

La tavola di verità è un po' inutile, parto subito da mappa 2x2. Il suggerimento è scrivere dentro ogni cella la cifra decimale corrispondente. Poi segno gli 1 dove mi servono (x<4 e x>8). Nelle altre celle deve valere zero (no don't care). Questo è il primo passo della sintesi.

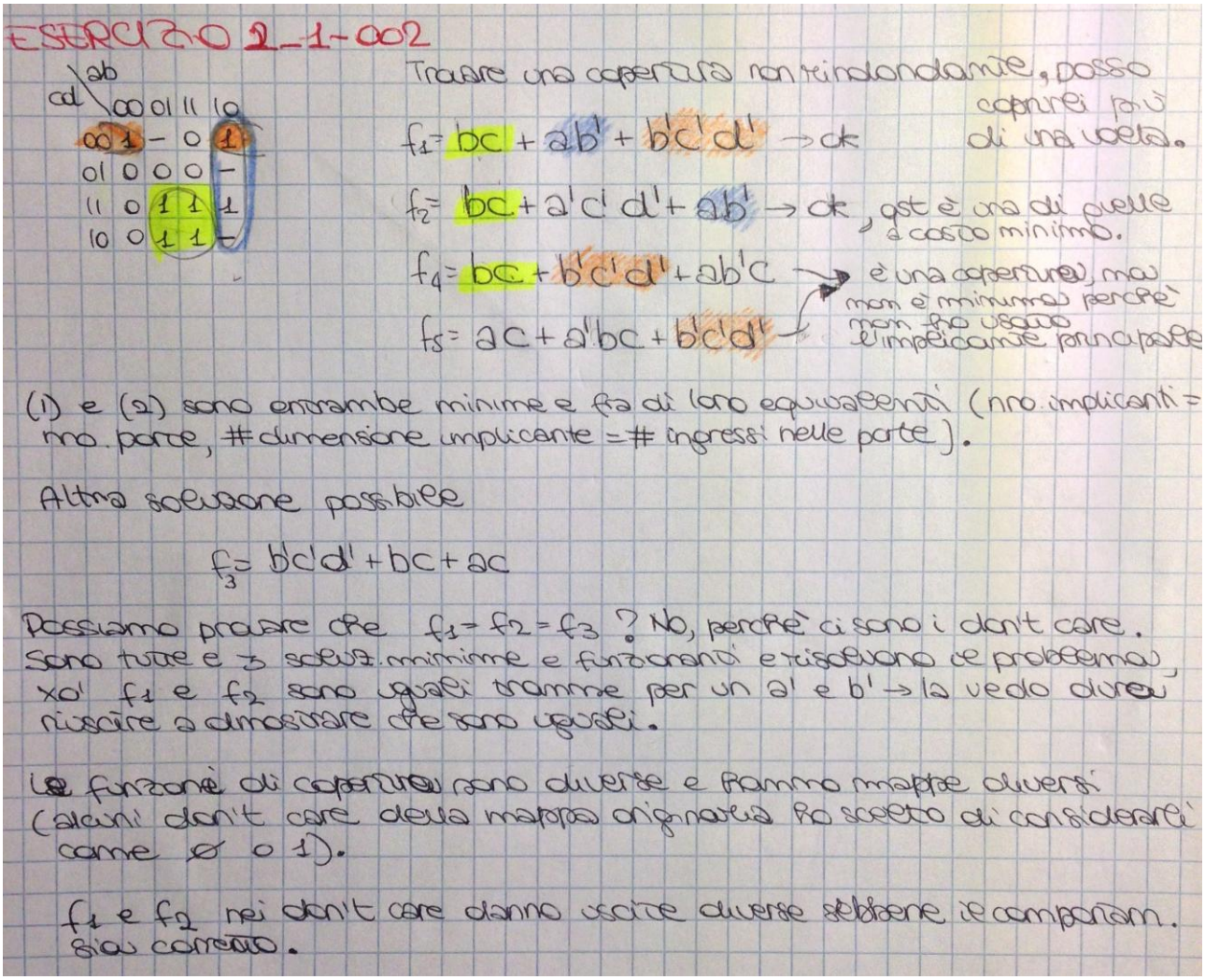
Scegliamo come soluzione U=X(3)'X(2)'+X(3)X(2)+X(3)… ok, ho trovato rappresentazione a costo minimo nel dominio comportamentale. Di qui è facile passare al circuito.

Il metodo di copertura delle mappe finisce qua il suo lavoro, però posso fare alcune osservazioni. Utilizzando De Morgan posso sostituire la prima porta con gli inverter a xor. E poi posso fare altre modifiche, però a questo livello sono tutte abbastanza equivalenti.



ESERCIZIO 2

Stesso di prima, ma i 4 bit in ingressi corrispondono a un numero decimale. Le specifiche sono diverse, se leggo bene le specifiche capisco che ci sono dei don't care e che quindi posso giocare con questi.



ESERCIZIO 6

Progettare un 2 ingressi, 1 segnale di controllo e 1 uscita. Multiplexer. Se c=0, allora u=a, else u=b (se avessi 4 segnali in ingresso avrei dovuto avere 2 segnali di controllo). Non posso collegare direttamente due uscite perché altrimenti inietterei corrente in un'uscita-> fondo le porte.

O con la mappa di Carnot e approccio a forza bruta, oppure dicendo c=0=c', quindi u=c'a+cb.