UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Triennale in Ingegneria Informatica

Sintesi ottima di funzioni booleane multi-uscita mediante programmazione lineare intera

Relatore:

Prof. Marco Cococcioni

Prof. Beatrice Lazzerini

Candidato:

Alessandro Versari

ANNO ACCADEMICO 2021/2022

"I just wanted to be one of The Strokes"

Abstract

Questo elaborato ha l'obiettivo di trovare una soluzione ottima al problema della sintesi a costo minimo di funzioni booleane multi-uscita su due livelli di logica.

Tradizionalmente il problema della sintesi di funzioni booleane multi-uscita viene risolto nel seguente modo: viene prima effettuata la sintesi a costo minimo di ciascuna uscita, poi viene sintetizzato il circuito completo come somma delle sintesi prodotte.

Tuttavia questo metodo non fornisce una sintesi ottima, intendendo per ottima quella di costo minimo (assumendo un opportuno criterio di costo). Infatti, concentrandosi su una uscita alla volta, non si tengono in conto eventuali guadagni dovuti alla possibilità di riutilizzare più volte una medesima sotto-circuiteria.

Di conseguenza, per trovare la soluzione ottima globale al problema multi-uscita, la scelta di una determinata sotto-circuiteria va effettuata in funzione delle sotto-circuiterie scelte per sintetizzare ciascuna delle singole uscite. Ciò rende il problema difficile, in quanto assume natura combinatorica.

La soluzione proposta nel presente lavoro consiste, in primo luogo nel formulare un modello matematico lineare che valuti il circuito nella sua interezza, in secondo luogo nel trovare la soluzione ottima attraverso la programmazione lineare intera.

Per risolvere il problema di programmazione lineare intera è stato utilizzato l'algoritmo del Branch&Bound, disponibile in Matlab (comando `intlinprog`).

La principale difficoltà del lavoro è stata quella di trovare la formulazione matematica corretta del problema, l'implementazione dell'algoritmo Quine-McCluskey e l'analisi statistica dei vantaggi forniti dall'uso del Branch&Bound rispetto al metodo classico. Ovvero il metodo basato sulla sintesi di funzioni a singola uscita, effettuata in maniera indipendente per ciascuna uscita, utilizzando l'algoritmo di Quine-McCluskey.

Keywords: Minimizzazione di funzioni booleane multi ingresso e multi-uscita, Programmazione Lineare Intera, Algoritmo di Quine-McCluskey.

Indice

1 Introduzione **1**

1.1 Sintesi di funzioni booleane 1

1.1.1 Reti combinatorie 1

1.1.2 Algebra di Boole 2

1.1.3 Sintesi in forma SP a costo minimo 3

1.1.4 Sintesi ad una uscita 3

1.1.5 Algoritmi di enumerazione degli implicanti principali 4

1.1.6 Sintesi in forma PS 5

1.2 Programmazione lineare 6

1.3 Premesse 6

1.3.1 Scelta del tipo di sintesi 6

1.3.2 Adozione della PLI anche per il problema ad una uscita 7

1.3.3 Utilizzo di ogni implicante nella formulazione delle variabili . . 7

2 Problema ad una uscita **9**

2.1 Modellazione del problema 10

2.2 Definizione delle variabili 10

2.3 Definizione dei vincoli 10

2.3.1 Vincoli di copertura 10

2.3.2 Vincoli di interezza 11

2.4 Funzione obiettivo 11

2.4.1 Costo a porte 11

2.4.2 Costo a diodi 12

2.5 Modello matematico 12

2.5.1 Generico 12

2.5.2 In formato primale standard 12

3 Problema a più uscite **13**

3.1 Modellazione del problema 14

3.2 Definizione delle variabili 14

3.3 Definizione dei vincoli 17

3.3.1 Vincoli di copertura 17

3.3.2 Vincoli di scelta 17

3.3.3 Vincoli di interezza 18

3.4 Funzione obiettivo 18
3.4.1 Costo a porte 18
3.4.2 Costo a diodi 18
3.5 Modello matematico 19
3.5.1 Generico 19
3.5.2 In formato primale standard 19

4 Esempi esplicativi 21
4.1 Uscite con un implicante in comune 21
4.2 Scelta di un implicante influenzata da un' altra uscita 23
4.3 Scelta di un implicante non principale 24
4.4 Uso dei "non specificato" 26

5 Esempio di funzionamento 29
5.1 Input 29
5.2 Sintesi della prima uscita 29
5.3 Sintesi di entrambe le uscite 31

6 Risultati e conclusioni 35
6.1 Risultati 35
6.2 Conclusioni 37

A Codice 39
A.1 `utils` 39
A.2 `getAllImplicants` 42
A.3 `oneOutputSynthesis` 44
A.4 `multipleOutputSynthesis` 46
A.5 `displayImplicants` 49
A.6 `synthesisCheck` 50
A.7 `statistics` 51
A.8 `plotStatistics` 53
A.9 `distribution` 54

B Sintesi a singola uscita mediante metodi classici 55
B.1 Espansione di Shannon 55
B.2 Forma canonica SP 55
B.2.1 Implicanti principali 55
B.3 Lista di copertura 56
B.3.1 Eliminazione degli implicanti principali ridondanti 56

Notazione utilizzata

- *mintermine* := uno stato di ingresso riconosciuto dalla rete
- *implicante* := il prodotto di alcune variabili di ingresso, dirette o negate, che riconosce alcuni stati di ingresso
- $\begin{bmatrix} A \\ B \end{bmatrix}$:= concatenazione verticale
- $\#A$:= cardinalità di A (numero di elementi dell'insieme)

Caso singola uscita

- y_1 uscita
- X vettore delle entrate di dimensione N ed indice n
- Δ matrice di copertura di dimensione $I \times J$ con indici i e j
- V vettore colonna di dimensione J e indice j, associato alla scelta degli implicanti

Caso multi-uscita

- Y vettore delle uscite di dimensione K ed indice k
- X vettore delle entrate di dimensione N ed indice n
- Δ matrice a blocchi diagonali formata da Δ^k matrici di copertura
- Δ^k matrice di copertura di dimensione $I \times \#V^k$ con indici i e j
- V vettore colonna di dimensione U e indice u, associato alla scelta degli implicanti all'interno delle uscite e formato dalla concatenazione dei V^k vettori
- V^k vettore colonna di dimensione $\#V^k$ e indice j, associato alla scelta degli implicanti all'interno di una uscita
- Z vettore colonna di dimensione L e indice l, associato alla scelta degli implicanti all'interno del circuito

Capitolo 1

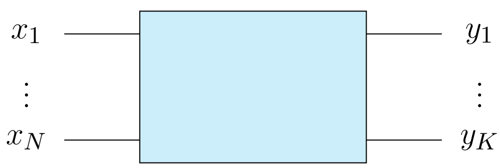
Introduzione

In questo capitolo saranno introdotti i concetti necessari per affrontare lo studio della sintesi di funzioni booleane multi-uscita.

1.1 Sintesi di funzioni booleane

1.1.1 Reti combinatorie

Una rete combinatoria è caratterizzata da N variabili logiche di ingresso, K variabili logiche di uscita e una funzione f che mappa uno stato di ingresso in uno stato di uscita.



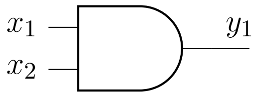
Una rete combinatoria può essere descritta mediante una tabella di verità, nella quale sono presenti: a sinistra l'insieme dei possibili stati di ingresso e a destra l'insieme degli stati di uscita corrispondenti.

x_N	$\dots$	x_1	y_K	$\dots$	y_1
0	$\dots$	0	1	$\dots$	0
0	$\dots$	1	0	$\dots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	$\dots$	1	1	$\dots$	0

Esempi di reti combinatorie

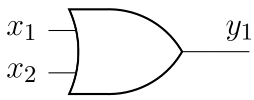
Porta AND

x_2	x_1	y_1
0	0	0
0	1	0
1	0	0
1	1	1



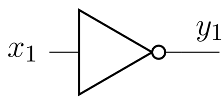
Porta OR

x_2	x_1	y_1
0	0	0
0	1	1
1	0	1
1	1	1



Porta NOT

x_1	y_1
0	1
1	0



Livelli di logica di una rete combinatoria

I livelli di logica di una rete combinatoria corrispondono al numero massimo di porte logiche che il segnale elettrico in ingresso deve attraversare per raggiungere l'uscita.

1.1.2 Algebra di Boole

L'algebra di Boole è un sistema algebrico basato su variabili logiche e operatori logici, utilizzato per la descrizione di reti combinatorie. Gli operatori logici di cui si avvale sono:

- Complemento := $\bar{x}$
- Prodotto logico := $x_1 \cdot x_2$, per semplicità espresso anche come $x_1 x_2$
- Somma logica := $x_1 + x_2$

Proprietà

- Involutiva $\bar{\bar{x}} = x$
- Commutativa
- Associativa e distributiva, sia della somma sia del prodotto
- Complementazione $x \cdot \bar{x} = 0 \qquad x + \bar{x} = 1$

- Unione e intersezione $x+0=x, x+1=1 \qquad x\cdot 0=0, x\cdot 1=x$
- Idempotenza $x+x=x \qquad x\cdot x=x$

Legge di De Morgan

$$\overline{x_1+x_2+\ldots+x_N}=\overline{x_1}\cdot\overline{x_2}\cdot\ldots\cdot\overline{x_N}$$

$$\overline{x_1\cdot x_2\cdot\ldots\cdot x_N}=\overline{x_1}+\overline{x_2}+\ldots+\overline{x_N}$$

1.1.3 Sintesi in forma SP a costo minimo

Scelti i due criteri di costo:

- a porte, in cui il costo equivale al numero di porte logiche
- a diodi, in cui il costo equivale al numero di diodi in ingresso alle porte logiche

si vuole sintetizzare una funzione booleana ad una uscita, producendo una rete a due livelli di logica in forma SP.

Non è certo che la sintesi ottenuta sia quella a costo minimo assoluto, poiché potrebbero esistere altri circuiti a due livelli di logica con un costo minore, oppure altri circuiti con un costo minore ma su più livelli di logica.

Sarà, però, affrontato solo il caso a due livelli di logica, dato che le reti con questa proprietà sono più veloci.

Per eseguire la sintesi di funzioni booleane a più uscite, tradizionalmente, si esegue la sintesi considerando ogni uscita come una funzione a sé stante e formando il circuito finale come unione dei circuiti risultanti, è evidente come questo metodo non sia ottimale.

1.1.4 Sintesi ad una uscita

Tradizionalmente la sintesi in forma SP di una funzione booleana ad una uscita si ottiene avvalendosi del seguente algoritmo:

- Vengono enumerati tutti gli implicanti principali
- Tra di essi vengono selezionati solo quelli non ridondanti, che corrispondono alla sintesi di costo minimo

*Per approfondire l'argomento di questo paragrafo controllare l'**appendice B**, in cui viene approfondito il significato dei termini: "mintermine", "implicante", "implicante principale".*

1.1.5 Algoritmi di enumerazione degli implicanti principali

Gli algoritmi che verranno mostrati in questo paragrafo utilizzano un diverso approccio per la ricerca degli implicanti principali, entrambi si basano sull'espansione di Shannon.

Il primo algoritmo è utilizzabile per un numero di variabili di ingresso elevato, mentre il secondo è utilizzabile solo con sei variabili di ingresso o meno.

Algoritmo di Quine-McCluskey

- Si raggruppano i mintermini in base al numero di "1" all'interno di essi
- Si effettuano delle fusioni tra gli implicanti dei gruppi adiacenti: se due implicanti differiscono per una sola variabile allora si fondono
- Vengono creati altri gruppi con gli implicanti fusi, i quali sono raggruppati per numero di "1" non includendo eventuali ripetizioni di implicanti
- Si iterano i passi precedenti finché non si possono effettuare più fusioni

Alla fine, gli implicanti che non hanno generato fusione sono quelli principali.

Esempio:

x_3	x_2	x_1	y_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$\Rightarrow$

x_3	x_2	x_1
0	0	1
0	1	0
0	1	1
1	0	1

$\Rightarrow$

x_3	x_2	x_1
0	-	1
-	0	1
0	1	-

Possiamo notare come gli unici implicanti che non fondono sono $\overline{x_3}x_1, \overline{x_2}x_1, \overline{x_3}x_2$, ovvero quelli dell'ultima iterazione. Può anche succedere che non fondano implicanti in iterazioni precedenti, in quel caso anch'essi sono principali.

Mappe di Karnaugh

Data una tabella di verità essa si può esprimere sotto forma di mappa di Karnaugh:

x_3	x_2	x_1	y_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

y_1

$x_3 \backslash x_2x_1$	00	01	11	10
0	0	1	1	1
1	0	1	0	0

L'algoritmo utilizzato consiste nel:

- Raggruppare tutti gli implicanti in base al loro grado, che corrisponde alla loro dimensione
- Partendo dal grado maggiore, selezionare tutti gli implicanti appartenenti al gruppo con il grado corrente
- Se con gli implicanti selezionati si copre tutta la mappa allora l'algoritmo termina, altrimenti si passa al gruppo successivo, ovvero quello con grado minore

Gli implicanti selezionati con questo algoritmo corrispondono agli implicanti principali della rete.

1.1.6 Sintesi in forma PS

Per ogni funzione boolena, oltre alla sintesi in forma SP (somma di prodotti), esiste anche quella in forma PS (prodotto di somme). Non è detto che il costo ottenuto dalle due sintesi sia uguale, anzi quasi sempre risulta diverso. Per questo motivo per sapere qual è la sintesi a costo minimo su due livelli di logica, è necessario eseguire entrambe le sintesi e infine scegliere quella a costo minore.

Algoritmo

- Data la funzione f si ricava la funzione $\bar{f}$, che fa corrispondere ad ogni stato di ingresso di f il suo complemento
- Si realizza la sintesi SP di $\bar{f}$
- Attraverso l'inserimento di un invertitore in uscita alla rete ottenuta si ottiene la sintesi di f

- Si applica il teorema di De Morgan in modo da ricavare l'espressione sotto forma di prodotti di somme

1.2 Programmazione lineare

La programmazione lineare (PL) è la branca della ricerca operativa che si occupa dello studio di algoritmi di risoluzione per problemi di ottimizzazione lineari.

Ogni problema di PL è composto da:

- una funzione obiettivo lineare
- un insieme di vincoli lineari

Ciascun problema di PL può essere trasformato in formato primale standard, ciò risulta vantaggioso poiché in questo modo ogni problema di PL può potenzialmente essere risolto con lo stesso algoritmo: il simplesso.

Formato primale standard

$$\begin{cases} \min CX \\ AX \leq B \end{cases}$$

dove C è il vettore dei costi, X è il vettore delle variabili, A è la matrice dei coefficienti e B è il vettore delle costanti.

Programmazione lineare intera

La programmazione lineare intera (PLI) comprende tutti i problemi lineari, le cui variabili hanno vincoli di interezza, spesso i problemi di PLI vengono risolti con l'algoritmo del "Branch&Bound".

1.3 Premesse

1.3.1 Scelta del tipo di sintesi

Una funzione booleana può essere sintetizzata sia come somma di prodotti sia come prodotto di somme.

Queste due sintesi sono rispettivamente equivalenti alla sintesi a porte NAND e alla sintesi a porte NOR.

Date le prime due ricavare le seconde è immediato, quindi la sintesi a porte NAND e la sintesi a porte NOR non verranno trattate.

Da quanto scritto nel **paragrafo 1.1.6**, è possibile evincere che un algoritmo di risoluzione del problema della sintesi in forma SP permetta di risolvere anche il problema in forma PS.

Perciò d'ora in avanti verrà trattato solamente il problema della sintesi in forma SP.

1.3.2 Adozione della PLI anche per il problema ad una uscita

Il problema della sintesi di funzioni booleane ad una uscita, come discusso nel **paragrafo 1.1**, può essere risolto con degli algoritmi euristici, i quali forniscono in tempi brevi la soluzione ottima ed hanno bassa complessità di esecuzione.

Nonostante ciò, anche la sintesi di funzioni booleane ad una uscita viene affrontata come problema di PLI, in modo da trattare il problema multi-uscita con più facilità.

1.3.3 Utilizzo di ogni implicante nella formulazione delle variabili

Contrariamente a quanto accade nel caso ad una uscita, nella sintesi di funzioni booleane a più uscite è possibile che convenga sintetizzare delle porte logiche che, per una o più uscite, sono descritte da implicanti non principali. Questo risulta evidente nell' **esempio 4.3**.

Capitolo 2

Problema ad una uscita

Data una funzione combinatoria ad una uscita del tipo:

$$y_1 = f(x_1, x_2, ..., x_N)$$

si vuole trovare la sintesi a costo minimo in forma SP di essa. I criteri di costo sono:

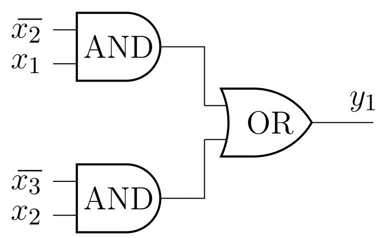
- numero di porte
- numero di diodi

Nella sintesi in forma SP è necessaria una porta AND per ogni implicante e una porta OR. Il numero di diodi necessario corrisponde al numero totale di input per ogni porta logica.

Quindi il costo a porte si può ottenere dal numero di implicanti necessari per la sintesi più uno.

Il costo a diodi corrisponde al numero di variabili all'interno di ogni implicante (il numero di input in ciascuna porta AND) più il numero di porte AND utilizzate (il numero di input nella porta OR).

Esempio:



Nel circuito in figura il costo a porte è 3, di cui 2 per il numero di porte AND e 1 per la porta OR.

Il costo a diodi è il numero di ingressi di ciascuna porta: 2 per ciascuna porta AND (4) più 2 per la porta OR.

2.1 Modellazione del problema

Per risolvere il problema della sintesi a costo minimo in forma SP attraverso la PLI, è stato scelto di utilizzare tutti i possibili implicanti come variabili ed il numero di porte o diodi come criteri di costo.

Per trovare tutti i possibili implicanti è stato utilizzato l'algoritmo di Quine-McCluskey.

Una volta trovati tutti gli implicanti, per definizione sappiamo che il circuito formato dalla somma di essi è necessariamente una soluzione ammissibile del nostro problema.

Questo ci fornisce un buon punto di partenza in quanto non sempre, dato un problema di PLI, esiste una soluzione ammissibile.

Il problema di ricerca della soluzione ottima corrisponde al problema di sintesi del circuito utilizzando solo gli implicanti il cui costo sommato è minimo, rispettando i vincoli imposti dalla funzione booleana fornita.

Per costruire i vincoli imposti viene creata una matrice di copertura, la quale indica per ciascun implicante se copre o meno uno stato della funzione. Tutti gli stati, infatti, dovranno essere coperti da almeno un implicante.

In seguito è possibile vedere la formulazione delle variabili e dei vincoli che permettono di risolvere questo tipo di problema utilizzando la PLI.

2.2 Definizione delle variabili

Dato I numero di mintermini e J numero di implicanti, si definiscono:

- La matrice di copertura Δ, di dimensione $I \times J$, in cui ogni cella $\delta_{ij} \in \{0, 1\}$ è del tipo

$$\delta_{ij} = \begin{cases} 1 & \text{se l'implicante j-esimo copre lo stato i-esimo} \\ 0 & \text{altrimenti} \end{cases}$$

- Il vettore colonna V, di dimensione J, in cui ogni cella $v_j \in \{0, 1\}$ è del tipo

$$v_j = \begin{cases} 1 & \text{se l'implicante j-esimo viene scelto} \\ 0 & \text{altrimenti} \end{cases}$$

2.3 Definizione dei vincoli

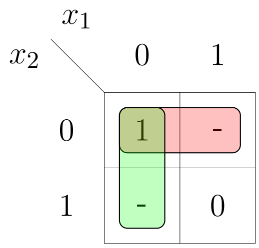
2.3.1 Vincoli di copertura

Ogni mintermine i-esimo deve essere coperto da almeno uno degli implicanti.

Questo vincolo, utilizzando disequazioni lineari, può essere ottenuto con la seguente formula:

$$\sum_{j=1}^{J} \delta_{ij} v_j \geq 1 \qquad \forall i \in [1, I]$$

la quale indica che per ogni mintermine i-esimo, almeno un implicante che copre il suddetto mintermine, deve essere utilizzato. Di conseguenza almeno un prodotto $\delta_{ij} v_j$ deve valere 1.



Nella mappa di Karnaugh sopra riportata, il vincolo di copertura comporta la scelta di almeno uno tra i due implicanti che coprono il mintermine $\overline{x_1}\,\overline{x_2}$.

2.3.2 Vincoli di interezza

Le variabili associate agli implicanti sono di tipo booleano, quindi ogni implicante può essere scelto oppure no.

$$v_j \in \{0, 1\} \qquad \forall j \in [1, J]$$

2.4 Funzione obiettivo

La funzione obiettivo varia in base al criterio di costo scelto. In seguito vengono affrontati i due tipi di funzione obiettivo: quella nel caso di costo a porte e quella nel caso di porte a diodi.

2.4.1 Costo a porte

Il costo a porte viene calcolato con la seguente formula:

$$\min CV + 1$$

dove il vettore dei costi C è un vettore riga del tipo $C = [1, ..., 1]$, di dimensione J.

Il costo di ciascuna porta è 1:

- quello delle porte AND viene considerato nel caso vengano scelte
- quello della porta OR è fisso

2.4.2 Costo a diodi

Il costo a diodi viene calcolato con la seguente formula:

$$\min C \begin{bmatrix} V \\ V \end{bmatrix}$$

dove il vettore dei costi C è un vettore riga del tipo $C = \begin{bmatrix} c_1, c_2, ..., c_J, 1, ..., 1 \end{bmatrix}$, di dimensione $2J$.

Ogni elemento c_j indica il costo a diodi di ciascun implicante, mentre ogni 1 indica il costo di ciascun diodo in entrata alla porta OR per ogni implicante scelto.

Semplificazione

La funzione obiettivo può essere semplificata considerando il costo dei diodi in ingresso alla porta OR come parte del costo j-esimo, da questa considerazione ne consegue che il vettore dei costi diventa $\tilde{C} = [c_1 + 1, c_2 + 1, ..., c_J + 1]$, di dimensione J, mentre la funzione obiettivo diventa:

$$\min \tilde{C} V$$

2.5 Modello matematico

Di seguito è mostrato il modello matematico del problema della sintesi booleana ad una uscita, scegliendo come criterio di costo quello a diodi.

2.5.1 Generico

$$\begin{cases} \min \tilde{C} V \\ \Delta V \geq 1 \\ v_j \in \{0, 1\} \qquad \forall j \in [1, J] \end{cases}$$

2.5.2 In formato primale standard

$$\begin{cases} \min \tilde{C} V \\ -\Delta V \leq -1 \\ v_j \in \{0, 1\} \qquad \forall j \in [1, J] \end{cases}$$

Capitolo 3

Problema a più uscite

Data una funzione combinatoria a più uscite del tipo:

$$y_1, y_2, ..., y_K = f(x_1, x_2, ..., x_N)$$

si vuole trovare la sintesi a costo minimo in forma SP di essa. I criteri di costo sono:

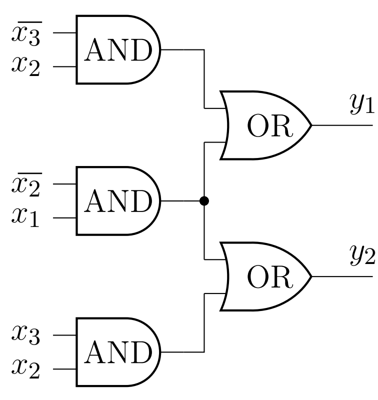
- numero di porte
- numero di diodi

Il costo a porte nel caso a più uscite equivale al numero totale di porte AND utilizzate più il numero di porte OR (che corrisponde al numero delle uscite).

Il costo a diodi coincide con il numero di variabili all'interno di ogni implicante, che corrisponde al numero di input di ciascuna porta AND, più il numero di input di ciascuna porta OR.

Il fulcro di questo studio consiste nel sintetizzare il circuito corrispondente alla funzione booleana multi-uscita, scegliendo porte logiche che siano riutilizzabili, nel caso in cui questo approccio riduca il costo complessivo.

Esempio:



Nell'esempio appena mostrato risulta evidente come la porta AND centrale venga riutilizzata. Per questo il numero di porte diminuisce di 1 rispetto al metodo tradizionale, che invece produrrebbe 2 circuiti a sé stanti.

3.1 Modellazione del problema

Per risolvere il problema della sintesi di funzioni booleane multi-uscita è stato ampliato il problema visto nel **capitolo 2**.

Rispetto al caso ad una uscita, il nuovo criterio da tenere in considerazione per la scelta delle porte logiche è che il costo di ciascuna porta, se utilizzata all'interno di più uscite, viene considerato una sola volta.

È da notare che il numero di diodi utilizzati nella sintesi di una porta AND dipende sia dal numero dei suoi ingressi sia dal numero di volte che essa viene utilizzata nel circuito. Riutilizzare una porta AND non è quindi un'operazione completamente priva di costi.

Per costruire i vincoli di copertura viene creata, per ogni uscita k-esima, una matrice di copertura. Ciascuna matrice di copertura indica se ciascun implicante copre o meno uno stato della funzione k-esima.

Affinché il costo delle porte riutilizzate venga considerato una volta sola, vengono utilizzate delle variabili ausiliarie, le quali indicano se una porta è stata impiegata o meno all'interno del circuito.

Il problema risultante consiste nel ridurre al minimo il costo relativo alle porte logiche, considerando il loro costo una singola volta e rispettando tutti i vincoli di copertura di ogni uscita.

In seguito possiamo vedere la formulazione delle variabili e dei vincoli che permettono di risolvere questo problema utilizzando la PLI.

3.2 Definizione delle variabili

Dati:

- K numero di uscite
- I numero di mintermini in un'uscita, varia a seconda dell'uscita
- $\#V^k$ numero di implicanti in un'uscita
- U numero totale di implicanti delle uscite
- L numero totale di implicanti nel circuito

Si definiscono:

- La matrice Δ a blocchi diagonali, costituiti da K matrici di copertura Δ^k, del tipo:

$$\begin{bmatrix} \Delta^1 & 0 & \cdots & 0 \\ 0 & \Delta^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Delta^K \end{bmatrix}$$

Ogni matrice Δ^k ha dimensione $I \times \#V^k$ e ogni cella $\delta_{ij}^k \in \{0,1\}$ è del tipo

$$\delta_{ij}^k = \begin{cases} 1 & \text{se l'implicante j-esimo copre il mintermine i-esimo nell'uscita k-esima} \\ 0 & \text{altrimenti} \end{cases}$$

- Il vettore colonna V, formato dalla concatenazione dei vettori colonna V^k, del tipo

$$\begin{bmatrix} V^1 \\ V^2 \\ \vdots \\ V^K \end{bmatrix}$$

Ciascun vettore V^k ha dimensione $\#V^k$ e ogni cella $v_j^k \in \{0,1\}$ è del tipo

$$v_j^k = \begin{cases} 1 & \text{se l'implicante j-esimo viene scelto nell'uscita k-esima} \\ 0 & \text{altrimenti} \end{cases}$$

V, in quanto concatenazione di K vettori, può anche essere visto come un vettore unico $V = [v_1, v_2, ..., v_U]^T$ con indice u e dimensione U, dove U è calcolata con la seguente formula:

$$U = \sum_{k=1}^{K} \#V^k$$

ovvero la somma delle dimensioni di ogni vettore V^k e quindi il numero totale di implicanti considerando tutte le uscite della rete.

- Il vettore colonna Z di dimensione L, formato da variabili ausiliare che si riferiscono alla scelta o meno degli implicanti all'interno del circuito. Ogni cella $z_l \in \{0,1\}$ è del tipo

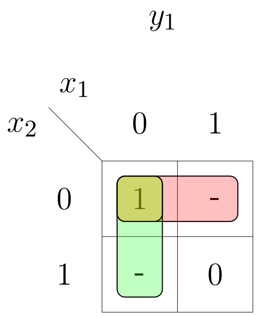
$$z_l = \begin{cases} 1 & \text{se l'implicante l-esimo viene utilizzato in almeno una delle uscite} \\ 0 & \text{altrimenti} \end{cases}$$

- La matrice di scelta Φ, di dimensione $L \times U$, in cui ogni cella ϕ_{lu} indica la corrispondenza tra un implicante associato ad un elemento del vettore V e un implicante associato ad un elemento di Z.

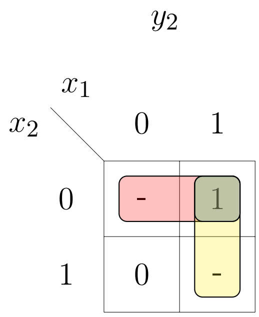
  Ogni cella ϕ_{lu} della matrice di scelta è del tipo:

$$\phi_{lu} = \begin{cases} 1 & \text{se l'implicante u-esimo associato a V è uguale} \\ & \text{all'implicante l-esimo associato a Z} \\ 0 & \text{altrimenti} \end{cases}$$

Esempio: data una funzione booleana a due entrate e due uscite del tipo:



Il vettore V^1 avrà dimensione 3 e gli implicanti ad esso associato saranno $[\overline{x_2}, \overline{x_1}, \overline{x_2}\,\overline{x_1}]$.



Il vettore V^2 avrà dimensione 3 e gli implicanti ad esso associato saranno $[\overline{x_2}, x_1, \overline{x_2}x_1]$.

Il vettore V, formato dal concatenamento dei V^k, avrà dimensione 6 e gli implicanti ad esso associato saranno $[\overline{x_2}, \overline{x_1}, \overline{x_2}\,\overline{x_1}, \overline{x_2}, x_1, \overline{x_2}x_1]$.

Il vettore Z, che indica la scelta o meno degli implicanti all'interno del circuito, avrà dimensione 5 e gli implicanti ad esso associato saranno
$[\overline{x_2}, \overline{x_1}, \overline{x_2}\,\overline{x_1}, x_1, \overline{x_2}x_1]$.

Al contrario di V, si può vedere come in Z $\overline{x_2}$ sia presente una volta sola, questo perché gli elementi z_l si riferiscono agli implicanti utilizzabili all'interno di tutto il circuito, senza considerare eventuali ripetizioni tra le uscite.

In questo caso la matrice Φ sarà del tipo:

$$\begin{array}{cc|cccccc} & & & & V & & & \\ & & \overline{x_2} & \overline{x_1} & \overline{x_2}\,\overline{x_1} & \overline{x_2} & x_1 & \overline{x_2}x_1 \\ & \overline{x_2} & 1 & 0 & 0 & 1 & 0 & 0 \\ & \overline{x_1} & 0 & 1 & 0 & 0 & 0 & 0 \\ Z & \overline{x_2}\,\overline{x_1} & 0 & 0 & 1 & 0 & 0 & 0 \\ & x_1 & 0 & 0 & 0 & 0 & 1 & 0 \\ & \overline{x_2}x_1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

All'implicante $\overline{x_2}$, presente in entrambe le uscite, corrispondono due 1 nella prima riga. Ciò indica che le variabili z_1, v_1 (o v_1^1) e v_4(o v_1^2) sono associate alla scelta dello stesso implicante ($\overline{x_2}$).

3.3 Definizione dei vincoli

3.3.1 Vincoli di copertura

Ogni mintermine i-esimo deve essere coperto da almeno uno degli implicanti in ogni uscita k-esima.

Questo vincolo, utilizzando disequazioni lineari, può essere ottenuto con la seguente formula:

$$\sum_{j=1}^{\#V^k} \delta_{ij}^k v_j^k \geq 1 \qquad \forall i \in [1, I] \qquad \forall k \in [1, K]$$

Il suddetto vincolo corrisponde a quello di copertura nel caso ad una uscita, ripetuto k volte.

3.3.2 Vincoli di scelta

Le variabili Z indicano la scelta o meno degli implicanti all'interno del circuito. Perciò ogni variabile z_l deve essere settata ad 1 nel caso in cui l'implicante l-esimo sia scelto in almeno una delle uscite, a 0 altrimenti.

Dato che le variabili v_u corrispondenti all'implicante l-esimo indicano la scelta o meno di quell'implicante all'interno di un'uscita generica, il valore di ciascuna variabile z_l si può ricavare attraverso l'OR logico delle variabili v_u corrispondenti all'implicante l-esimo. L'OR logico delle suddette variabili viene espresso con la seguente formula:

$$z_l = \phi_{l1} v_1 \vee \phi_{l2} v_2 \vee ... \vee \phi_{lU} v_U \qquad \forall l \in [1, L]$$

Se anche una sola v_u corrispondente all'implicante l-esimo è settata ad 1, allora anche z_l deve essere settata ad 1. Questo, utilizzando solamente vincoli lineari, è raggiungibile imponendo:

$$z_l \geq \sum_{u=1}^{U} \frac{1}{K+1} \phi_{lu} v_u \qquad \forall l \in [1, L]$$

Semplificazione

In modo da semplificare la notazione si può definire la matrice $\tilde{\Phi} = \frac{1}{K+1}\Phi$, quindi il vincolo di scelta diventa:

$$z_l \geq \sum_{u=1}^{U} \tilde{\phi}_{lu} v_u \qquad \forall l \in [1, L]$$

3.3.3 Vincoli di interezza

Le variabili associate agli implicanti sono di tipo booleano, quindi ogni implicante può essere scelto oppure no.

$$\begin{cases} v_u \in \{0,1\} & \forall u \in [1,U] \\ z_l \in \{0,1\} & \forall l \in [1,L] \end{cases}$$

3.4 Funzione obiettivo

3.4.1 Costo a porte

$$\min CZ + K$$

dove $C = [1, ..., 1]$ è di dimensione L. In questo caso il costo viene calcolato in modo simile al caso ad una uscita, con la differenza che, grazie alle variabili ausiliare, ogni porta AND viene considerata una volta sola e il numero di porte OR non è più 1 ma K (numero delle uscite).

3.4.2 Costo a diodi

$$\min C \begin{bmatrix} V \\ Z \end{bmatrix}$$

dove $C = \left[\ 1, ..., 1 \;\middle|\; c_1, c_2, ..., c_L\ \right]$ è di dimensione $U + L$ e il costo c_l indica il costo a diodi di ciascun implicante (porta AND).

Il costo dei diodi in ingresso alle porte AND, nel caso in cui esse vengano scelte, viene considerato una sola volta. Mentre il costo dei diodi in ingresso ad ogni porta OR equivale ad 1 per ogni porta AND scelta in ciascuna uscita.

3.5 Modello matematico

Di seguito è mostrato il modello matematico del problema della sintesi booleana multi-uscita, scegliendo come criterio di costo quello a diodi.

3.5.1 Generico

$$\begin{cases} \min C \begin{bmatrix} V \\ Z \end{bmatrix} \\ \\ \Delta V \geq 1 \\ IZ \geq \tilde{\Phi} V \\ \\ v_u \in \{0,1\} & \forall u \in [1,U] \\ z_j \in \{0,1\} & \forall j \in [1,J] \end{cases}$$

3.5.2 In formato primale standard

$$\begin{cases} \min C \begin{bmatrix} V \\ Z \end{bmatrix} \\ \\ \begin{bmatrix} -\Delta & 0 \\ \tilde{\Phi} & -I \end{bmatrix} \begin{bmatrix} V \\ Z \end{bmatrix} \leq \begin{bmatrix} -1 \\ 0 \end{bmatrix} \\ \\ v_u \in \{0,1\} & \forall u \in [1,U] \\ z_j \in \{0,1\} & \forall j \in [1,J] \end{cases}$$

Capitolo 4

Esempi esplicativi

In questo capitolo sono presenti degli esempi che illustrano come la sintesi proposta in questo elaborato fornisca delle reti combinatorie a costo inferiore rispetto a quelle prodotte dalla sintesi tradizionale., alcuni esempi sono stati tratti dalla fonte **[3]**.

4.1 Uscite con un implicante in comune

Nel seguente esempio vengono sintetizzate due uscite con un implicante in comune. Si può notare come, utilizzando la sintesi a più uscite un implicante viene riutilizzato, mentre utilizzando la sintesi tradizionale il costo di esso viene considerato due volte.

Listing 4.1: Codice Esempio 1

```
y_1 = {[1,2,3,5] + 1, []};
y_2 = {[1,5,6,7] + 1, []};

[implicants_1, v_1] = oneOutputSynthesis(y_1{1}, y_1{2}, InputsNumber = 3);
[implicants_2, v_2] = oneOutputSynthesis(y_2{1}, y_2{2}, InputsNumber = 3);
displayImplicants({implicants_1})
displayImplicants({implicants_2})

[implicants, v] = multipleOutputSynthesis(3, {y_1, y_2});
displayImplicants(implicants)

savings = round((v_1 + v_2 - v) / (v_1 + v_2) * 100, 2);
fprintf('The gateInput cost is improoved by %.2f%% ', savings)
```

Figura 4.1: Sintesi tradizionale

y_1

x_3 \ x_2x_1	00	01	11	10
0	0	1	1	1
1	0	1	0	0

y_2

x_3 \ x_2x_1	00	01	11	10
0	0	1	0	0
1	0	1	1	1

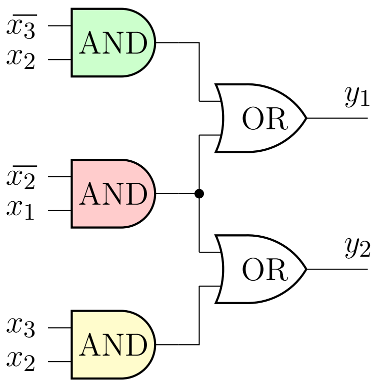
Di seguito si può vedere l'output della sintesi eseguita dal programma, dove risulta evidente come l'implicante $\overline{x_2}x_1$ venga utilizzato due volte per la sintesi a più uscite.

In questo modo si ottiene una riduzione del costo a diodi del 16.67%.

Listing 4.2: Output Esempio 1

```
Solution: [x2'x1 + x3'x2]
Solution: [x2'x1 + x3x2]
Solution: [x2'x1 + x3'x2][x2'x1 + x3x2]
The gateInput cost is improoved by 16.67%
```

Figura 4.2: Sintesi ottima



4.2 Scelta di un implicante influenzata da un' altra uscita

In questo esempio si nota come la scelta di un implicante possa essere influenzata dalla presenza dello stesso all'interno di un'altra uscita.

Listing 4.3: Codice Esempio 2

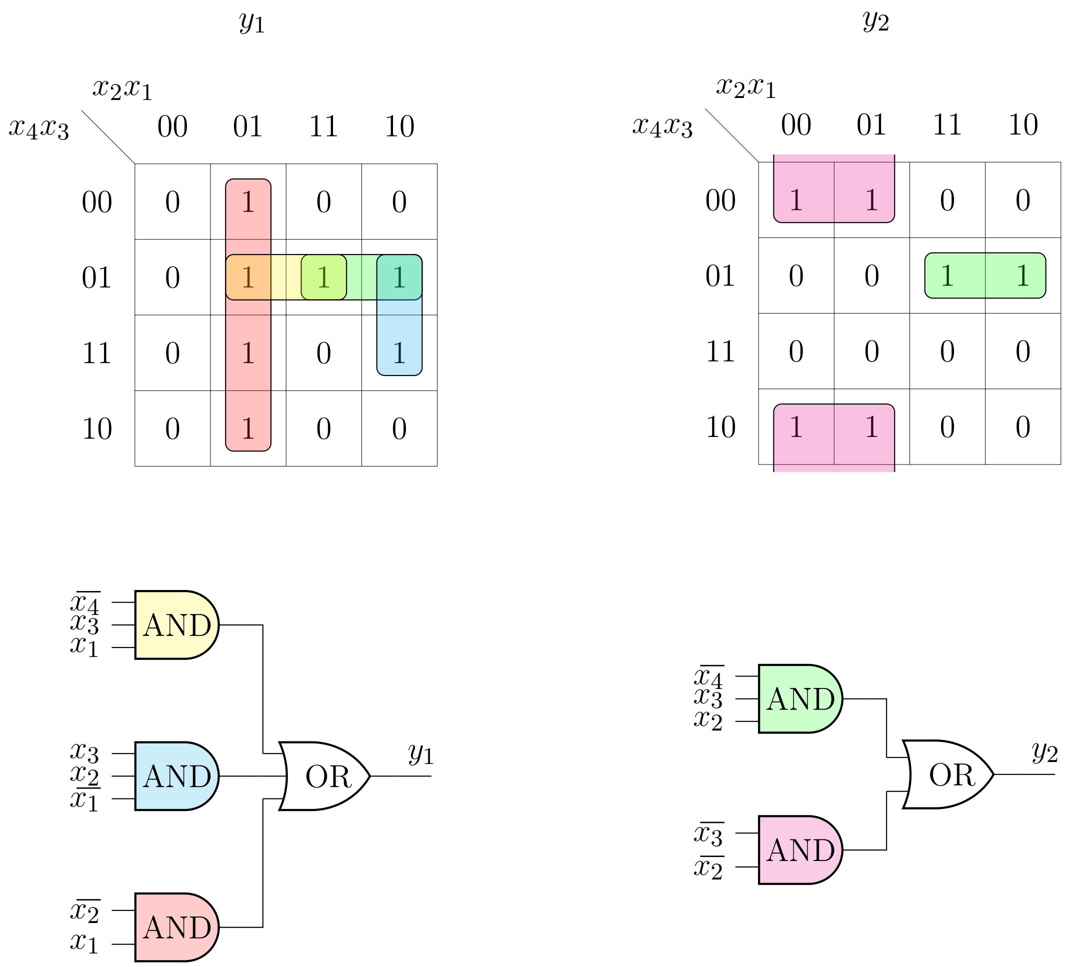
```
y_1 = {[1,5,6,7,9,13,14] + 1, []};
y_2 = {[0,1,6,7,8,9] + 1, []};

[implicants_1, v_1] = oneOutputSynthesis(y_1{1}, y_1{2}, InputsNumber = 4);
[implicants_2, v_2] = oneOutputSynthesis(y_2{1}, y_2{2}, InputsNumber = 4);
displayImplicants({implicants_1})
displayImplicants({implicants_2})

[implicants, v] = multipleOutputSynthesis(4, {y_1, y_2});
displayImplicants(implicants)

savings = round((v_1 + v_2 - v) / (v_1 + v_2) * 100, 2);
fprintf('The gateInput cost is improoved by %.2f%% ', savings)
```

Figura 4.3: Sintesi Tradizionale



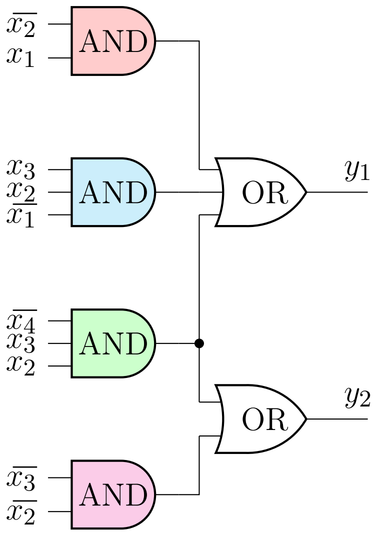
Si può notare come la scelta dell'implicante $\overline{x_4}x_3x_2$ risulti vantaggiosa in quanto lo stesso deve comunque essere sintetizzato, poiché presente in entrambe le uscite.

Il costo a diodi complessivo è stato ridotto del 16.67%.

Listing 4.4: Output Esempio 2

```
Solution: [x4'x3x1 + x3x2x1' + x2'x1]
Solution: [x4'x3x2 + x3'x2']
Solution: [x4'x3x2 + x3x2x1' + x2'x1][x4'x3x2 + x3'x2']
The gateInput cost is improoved by 16.67%
```

Figura 4.4: Sintesi ottima



4.3 Scelta di un implicante non principale

In questo esempio si nota come la scelta migliore sia quella di utilizzare un implicante non principale, in quanto esso è necessario per la sintesi di un'altra uscita.

Listing 4.5: Codice Esempio 3

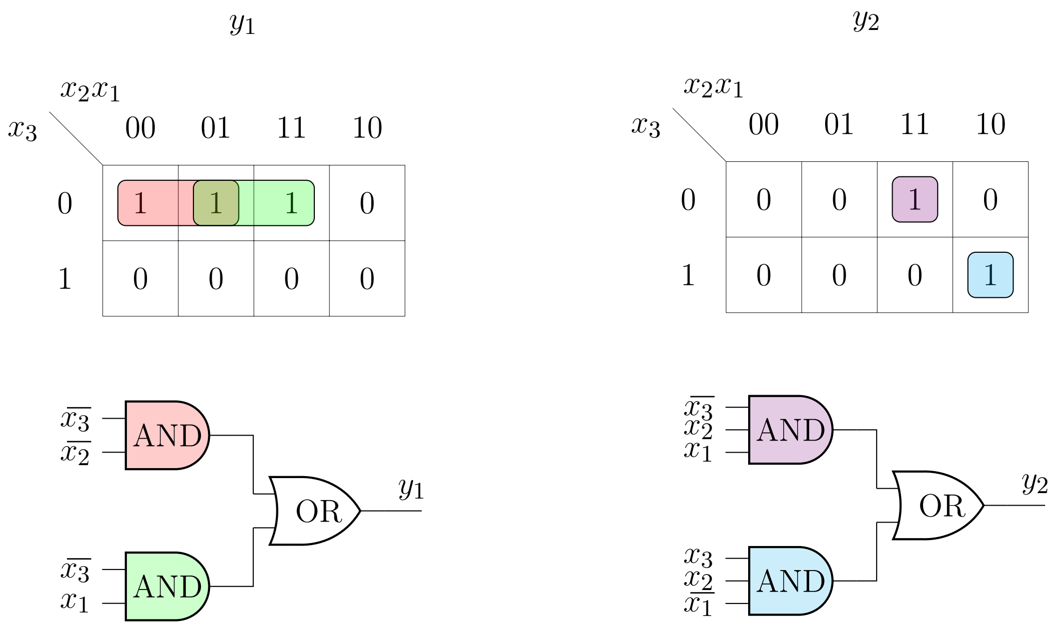
```
y_1 = {[0,1,3] + 1, []};
y_2 = {[3,6] + 1, []};

[implicants_1, v_1] = oneOutputSynthesis(y_1{1}, y_1{2}, InputsNumber = 3);
[implicants_2, v_2] = oneOutputSynthesis(y_2{1}, y_2{2}, InputsNumber = 3);
displayImplicants({implicants_1})
displayImplicants({implicants_2})

[implicants, v] = multipleOutputSynthesis(3, {y_1, y_2});
displayImplicants(implicants)

savings = round((v_1 + v_2 - v) / (v_1 + v_2) * 100, 2);
fprintf('The gateInput cost is improoved by %.2f%% ', savings)
```

Figura 4.5: Sintesi tradizionale

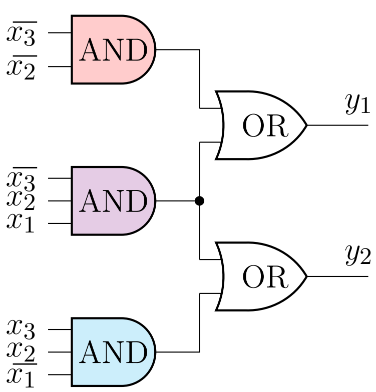


L'implicante $\overline{x_3}x_2x_1$ se utilizzato per la sintesi di entrambe le uscite permette di utilizzare una porta in meno, riducendo quindi il costo a diodi del 14.29%.

Listing 4.6: Output Esempio 3

```
Solution: [x3'x2' + x3'x1]
Solution: [x3'x2x1 + x3x2x1']
Solution: [x3'x2x1 + x3'x2'][x3'x2x1 + x3x2x1']
The gateInput cost is improoved by 14.29%
```

Figura 4.6: Sintesi ottima



4.4 Uso dei "non specificato"

In questo esempio si nota come l'uso dei "non specificato" influenza la scelta degli implicanti nel caso della sintesi a più uscite. È possibile, infatti, che risulti conveniente sintetizzare implicanti più piccoli (e quindi più costosi) in modo da poterli riutilizzare all'interno di più uscite.

Listing 4.7: Codice Esempio 4

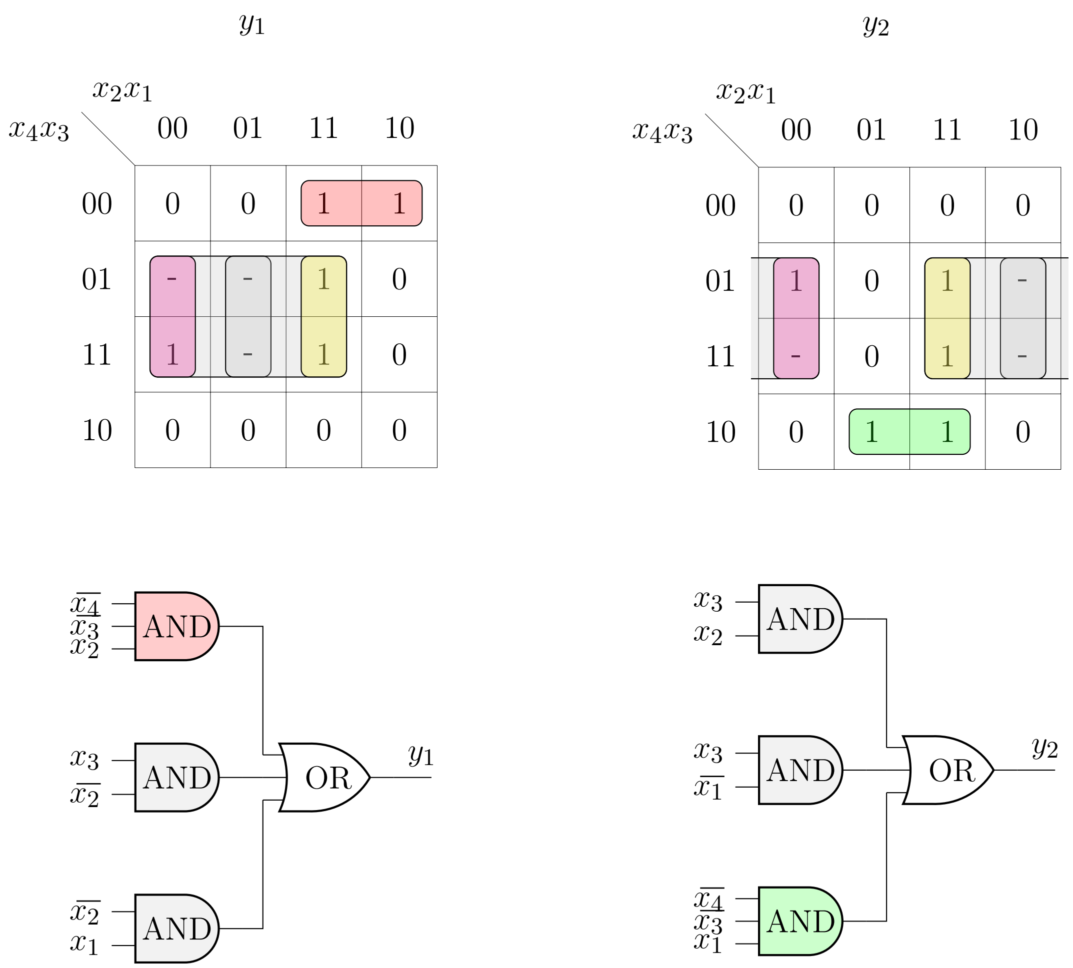
```
y_1 = {[2,3,7,12,15] + 1, [4,5,13] + 1};
y_2 = {[4,7,9,11,15] + 1, [6,12,14] + 1};

[implicants_1, v_1] = oneOutputSynthesis(y_1{1}, y_1{2}, InputsNumber = 4);
[implicants_2, v_2] = oneOutputSynthesis(y_2{1}, y_2{2}, InputsNumber = 4);
displayImplicants({implicants_1})
displayImplicants({implicants_2})

[implicants, v] = multipleOutputSynthesis(4, {y_1, y_2});
displayImplicants(implicants)

savings = round((v_1 + v_2 - v) / (v_1 + v_2) * 100, 2);
fprintf('The gateInput cost is improoved by %.2f%% ', savings)
```

Figura 4.7: Sintesi tradizionale



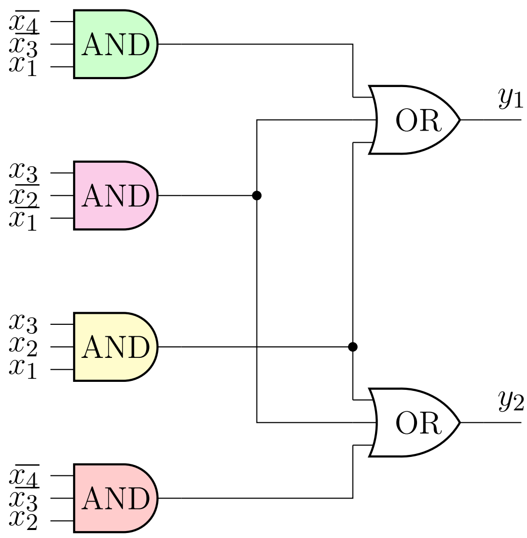
Si può notare come, nonostante nella sintesi tradizionale convenga sintetizzare gli implicanti di ordine 2, nella sintesi a più uscite la scelta migliore sia quella di riutilizzare gli implicanti di ordine 1.

Ottenendo così un miglioramento del costo a diodi del 10%.

Listing 4.8: Output Esempio 4

```
Solution: [x4‘x3‘x2 + x3x2‘ + x3x1]
Solution: [x4x3‘x1 + x3x1‘ + x3x2]
Solution: [x4‘x3‘x2 + x3x2‘x1‘ + x3x2x1][x3x2‘x1‘ + x4x3‘x1 + x3x2x1]
The gateInput cost is improoved by 10.00%
```

Figura 4.8: Sintesi ottima



Capitolo 5

Esempio di funzionamento

5.1 Input

In modo che il flusso di esecuzione sia chiaro al lettore, è stato utilizzato l'input dell'**esempio 4.1** per eseguire il programma di sintesi ad una e più uscite.

Listing 5.1: Codice esempio prolisso

```
y_1 = {[1,2,3,5] + 1, []};
y_2 = {[1,5,6,7] + 1, []};

[implicants_1, v_1] = oneOutputSynthesis(y_1{1},y_1{2}, Verbose=true);
displayImplicants({implicants_1})

[implicants_2, v_2] = oneOutputSynthesis(y_2{1}, y_2{2}, Verbose=true);
displayImplicants({implicants_2})

[implicants, v] = multipleOutputSynthesis(3, {y_1, y_2}, Verbose=true);
displayImplicants(implicants)
```

In seguito si può vedere l'output stampato dalle funzioni se lanciate con il parametro opzionale `Verbose=true`.

5.2 Sintesi della prima uscita

All'inizio della sintesi ad una uscita vengono generati tutti i possibili implicanti utilizzando l'algoritmo di Quine-McCluskey, del quale è possibile vedere le iterazioni.

```
QM iteration 1:

    {0x0 double     }
    {["001"    "010"]}
    {["011"    "101"]}
    {0x0 double     }


QM iteration 2:

    {0x0 double              }
```

```
{["0-1"    "-01"    "01-"]}
{0x0 double             }
{0x0 double             }
```

In seguito viene visualizzata la lista dei possibili implicanti, la matrice di copertura a loro collegata e il vettore dei costi. Per ogni implicante:

- "1" indica che la variabile logica è presa diretta
- "0" che è presa negata
- "-" che non è presente all'interno dell'implicante

Ad ogni colonna j-esima della matrice di copertura corrisponde un implicante, ogni "1" indica se l'implicante in questione copre il mintermine i-esimo.

Ogni elemento j-esimo del vettore dei costi indica il costo di ciascun implicante.

```
All possible implicants are:

    "001"
    "010"
    "011"
    "101"
    "0-1"
    "-01"
    "01-"

The coverage matrix is:

     1     0     0     0     1     1     0
     0     1     0     0     0     0     1
     0     0     1     0     1     0     1
     0     0     0     1     0     1     0

The cost vector is:

     4
     4
     4
     4
     3
     3
     3
```

In seguito i dati ottenuti vengono convertiti in formato standard e passati a `intlinprog` che troverà una soluzione ottima rispettando i vincoli imposti.

```
LP:                Optimal objective value is 6.000000.


Optimal solution found.

Intlinprog stopped at the root node because the objective value is within a
     gap tolerance of the optimal value, options.AbsoluteGapTolerance = 0 (
    the default value). The intcon
variables are integer within tolerance, options.IntegerTolerance = 1e-05 (
    the default value).

Solution:
```

```
    0
    0
    0
    0
    0
    1
    1

The cost is:

    6
```

In seguito la soluzione viene formattata in modo da essere facilmente leggibile, il simbolo ‘ indica che la variabile alla sua sinistra è negata.

```
Solution: [x2‘x1 + x3‘x2]
```

5.3 Sintesi di entrambe le uscite

All'inizio della sintesi multi-uscita vengono trovati gli implicanti di entrambe le uscite e poi generate le matrici di copertura corrispondenti.

```
Output 1:

All possible implicants are:

   "001"
   "010"
   "011"
   "101"
   "0-1"
   "-01"
   "01-"

The coverage matrix is:

    1    0    0    0    1    1    0
    0    1    0    0    0    0    1
    0    0    1    0    1    0    1
    0    0    0    1    0    1    0


Output 2:

All possible implicants are:

   "001"
   "101"
   "110"
   "111"
   "-01"
   "1-1"
   "11-"

The coverage matrix is:

    1    0    0    0    1    0    0
    0    1    0    0    1    1    0
    0    0    1    0    0    0    1
    0    0    0    1    0    1    1
```

Poi viene generato il vettore degli implicanti di tutto il circuito (senza ripetizioni) e la matrice di scelta $\tilde{\Phi}$ (visualizzata trasposta) in cui ogni colonna corrisponde ad un implicante del circuito. In ogni colonna l-esima ogni coefficiente u-esimo diverso da zero indica la corrispondenza con un implicante in una uscita.

```
The not redundant implicants are:

    "-01"
    "0-1"
    "001"
    "01-"
    "010"
    "011"
    "1-1"
    "101"
    "11-"
    "110"
    "111"

The transposed choice matrix is (each column is an implicant):

       0     0    0.33   0     0     0     0     0     0     0     0
       0     0     0     0    0.33   0     0     0     0     0     0
       0     0     0     0     0    0.33   0     0     0     0     0
       0     0     0     0     0     0     0    0.33   0     0     0
       0    0.33   0     0     0     0     0     0     0     0     0
      0.33   0     0     0     0     0     0     0     0     0     0
       0     0     0    0.33   0     0     0     0     0     0     0
       0     0    0.33   0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0    0.33   0     0     0
       0     0     0     0     0     0     0     0     0    0.33   0
       0     0     0     0     0     0     0     0     0     0    0.33
      0.33   0     0     0     0     0     0     0     0     0     0
       0     0     0     0     0     0    0.33   0     0     0     0
       0     0     0     0     0     0     0     0    0.33   0     0
```

Viene generato anche il vettore dei costi. Si può vedere come i costi iniziali sono tutti equivalenti ad 1 (i costi dei diodi in ingresso alle porte OR). Mentre i costi finali sono variabili in base al numero di ingressi di ciascuna porta AND.

```
The cost vector is:

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     2
     2
     3
     2
     3
```

```
3
2
3
2
3
3
```

In seguito, i dati ottenuti vengono convertiti in formato standard e passati a `intlinprog` che troverà una soluzione ottima rispettando i vincoli imposti.

```
LP:                Optimal objective value is 6.666667.

Heuristics:        Found 1 solution using ZI round.
                   Upper bound is 10.000000.
                   Relative gap is 0.00%.


Optimal solution found.

Intlinprog stopped at the root node because the objective value is within a
     gap tolerance of the optimal value, options.AbsoluteGapTolerance = 0 (
    the default value). The intcon
variables are integer within tolerance, options.IntegerTolerance = 1e-05 (
    the default value).

Solution:

     0
     0
     0
     0
     0
     1
     1
     0
     0
     0
     0
     1
     0
     1
     1
     0
     0
     1
     0
     0
     0
     0
     1
     0
     0

The cost is:

   10.0000
```

La soluzione viene formattata in modo da essere facilmente leggibile, il simbolo ‘ indica che la variabile alla sua sinistra è negata.

```
Solution: [x2‘x1 + x3‘x2][x2‘x1 + x3x2]
```

Capitolo 6

Risultati e conclusioni

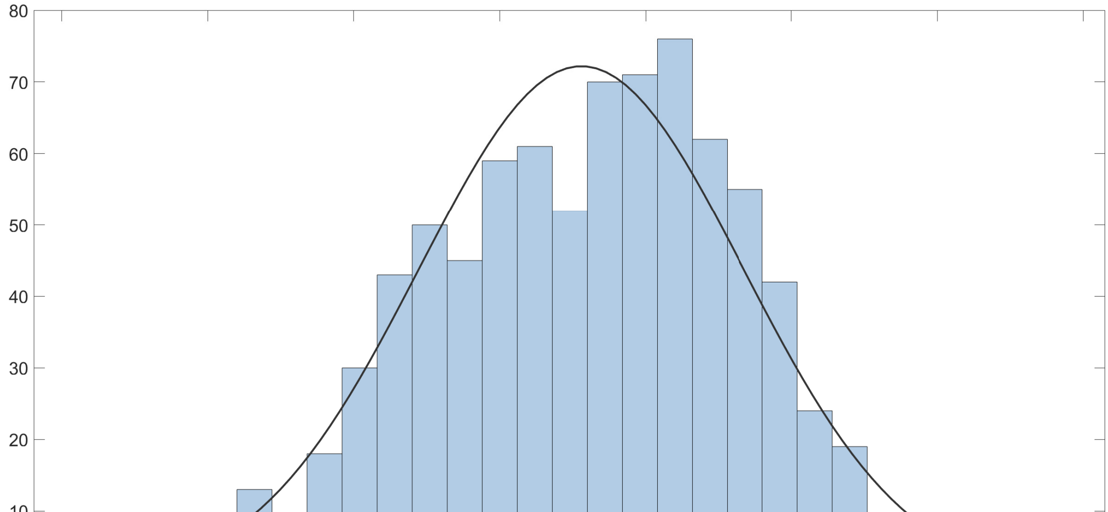
6.1 Risultati

È stata effettuata una serie di test per scoprire a quanto ammonta il risparmio percentuale medio a diodi, utilizzando la sintesi proposta dalla tesi, al variare del numero di entrate e numero di uscite. In seguito, per ogni combinazione possibile sono state ricavate $\overline{X}$ media e S^2 varianza campionarie attraverso le formule:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} \qquad S^2 = \frac{\sum_{i=1}^{n}(X_i^2 - \overline{X})}{n-1}$$

È stato osservato che i miglioramenti percentuali possono essere considerati come una variabile aleatoria con distribuzione gaussiana. Esempio:

Figura 6.1: Distribuzione di probabilità relativa al caso con 12 uscite e 6 entrate



Attraverso i dati ricavati dai test eseguiti saranno individuati degli intervalli di confidenza, ricavabili con la formula:

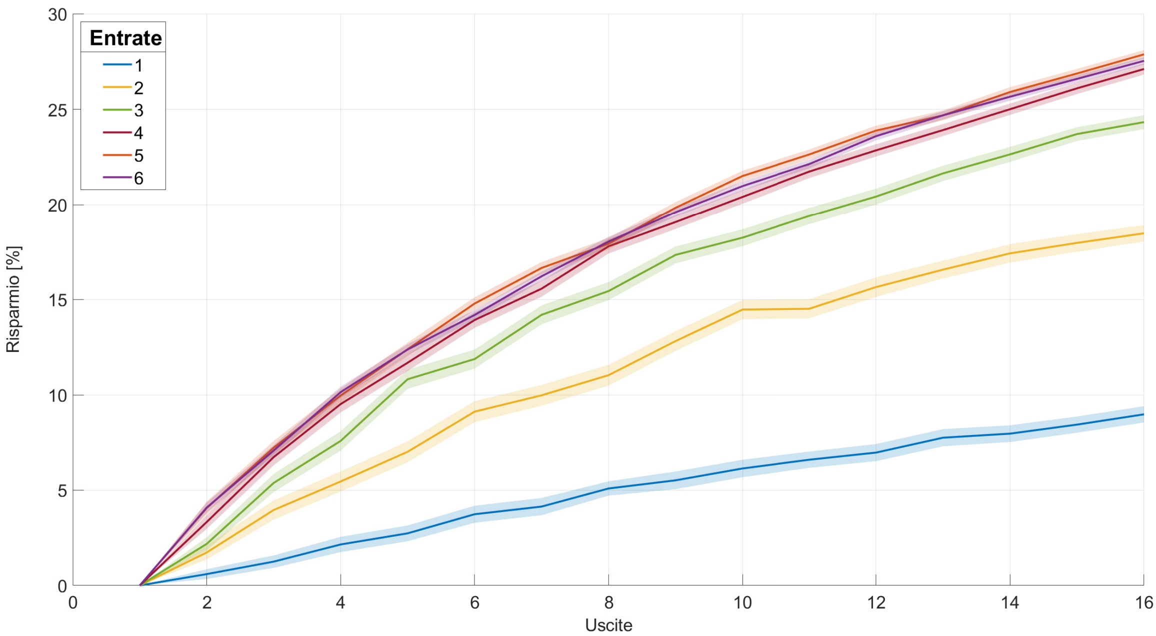
$$\overline{X} \pm \frac{S}{\sqrt{n}}\tau(1-\frac{1}{2}, n-1)$$

dove τ è il quantile della variabile di Student, che dipende dal numero di test eseguiti e dal livello di confidenza utilizzato nella misurazione.

Grazie all'elevato numero di test effettuato τ può essere approssimato al quantile di una gaussiana che, con un livello di confidenza del 95%, ha il valore di 1.96.

In seguito è mostrato un grafico contenente i dati raccolti, nel quale gli intervalli di confidenza assumono un colore più chiaro.

Figura 6.2: Risparmi in base al numero di output



Si può notare come all'aumentare del numero delle uscite vi è un progressivo miglioramento del risparmio percentuale.

Il risparmio percentuale medio massimo registrato è del **27.87%**, mentre gli intervalli di confidenza hanno tutti dimensione inferiore all' 1.10%.

6.2 Conclusioni

Nel presente lavoro di tesi è stato affrontato il problema della sintesi ottima *esatta* di una funzione booleana a più ingressi e più uscite. Dopo aver formalizzato il problema come un problema di ottimizzazione lineare intera, lo si è risolto in Matlab utilizzando l'algoritmo Branch&Bound. La complessità della soluzione ottenuta è stata confrontata con quella basata sull'utilizzo dell'algoritmo (approssimato, e dunque sub-ottimo) di Quine-McCluskey, invocato tante volte quante sono le uscite.

La sintesi ottima mediante Branch&Bound è risultata significativamente migliore come complessità (costo) del circuito risultante.

La disponibilità di un metodo esatto per la sintesi ottima del caso multi-uscita apre le porte allo studio di algoritmi che, con tempi di risoluzione brevi, forniscono soluzioni approssimate.

In particolare consentirà di valutare la bontà dell'approssimazione di una soluzione fornita da un algoritmo, conoscendo il tipo di rete combinatoria da sintetizzare e il tempo computazionale richiesto.

Appendice A

Codice

A.1 utils

Funzioni generiche utilizzate all'interno del codice per eseguire controlli o operazioni sui dati.

Listing A.1: Funzioni di utilità

```
classdef utils

    methods (Static)

        function res = mergeSorted(array_1,array_2)

            arguments
                array_1 (1,:) double ...
                    {mustBeInteger,mustBePositive,utils.mustBeAscending}
                array_2 (1,:) double ...
                    {mustBeInteger,mustBePositive,utils.mustBeAscending}
            end

            % returns
            %   a sorted array

            i_1 = 1;
            i_2 = 1;
            total_length = length(array_1) + length(array_2);
            res = zeros(1,total_length);

            for i = 1: total_length

                if i_1 > length(array_1)
                    res(i) = array_2(i_2);
                    i_2 = i_2 + 1;
                    continue;
                end

                if i_2 > length(array_2)
                    res(i) = array_1(i_1);
                    i_1 = i_1 + 1;
                    continue;
                end

                if array_1(i_1) < array_2(i_2)
                    res(i) = array_1(i_1);
```

```
                i_1 = i_1 + 1;
            else
                res(i) = array_2(i_2);
                i_2 = i_2 + 1;
            end
        end
    end

    function res = mergeStrings(a,b,substitute_char)

        arguments
            a string
            b string
            substitute_char char
        end

        a_chars = a{1};
        b_chars = b{1};
        res = a_chars;

        if length(a_chars) ~= length(b_chars)
            res = "";
        end

        count = 0;

        for i = 1:length(a_chars)

            a_char = a_chars(i);
            b_char = b_chars(i);

            % if the chars differ
            if a_char ~= b_char

                % if one of the chars is the substitue_char
                if a_char == substitute_char ...
                || b_char == substitute_char
                    res = "";
                    return;
                end

                res(i) = substitute_char;
                count = count + 1;
            end

            if count > 1
                res = "";
                return;
            end
        end

        res = string(res);

    end

    function idx = findString(string_array,string)

        idx = -1;

        for i = 1:length(string_array)
            if strcmp(string_array(i,:),string)
                idx = i;
                return;
            end
```

```
            end

        end

        function count = countMatches(char_array,pat)

            arguments
                char_array string
                pat pattern
            end

            % returns
            %     how many times the pattern is found

            count = length(strfind(char_array,pat));
        end


        function res = areAllCellsEmpty(cell_array)

            res = 1;
            for i = 1:length(cell_array)
                if ~ isempty(cell_array{i})
                    res = 0;
                    return;
                end
            end
        end


        function mustBeAscending(a)

            if utils.isInAscendingOrder(a) ; return ; end

            eidType = 'mustBeAscending:notInAscendingOrder';
            msgType = 'Input must be in ascending order';
            throwAsCaller(MException(eidType,msgType))
        end

        function res = isInAscendingOrder(a)

            res = 1;
            last = 0;

            for i = a

                if i <= last
                    res = 0;
                    return;
                end
                last = i;
            end

        end

        function res = minimumBits(numbers)
            res = ceil(log2(max(numbers)));
        end

        function [x,v,timedOut] = ...
            intlinprogWrap(C,A,b,variablesNumber,verbose,timeout)

            % calls intrlinprog
            %   setting intcon,lb,ub with the specified variablesNumber
```

```
            %   sets on or off the display using the verbose parameter

            intcon = 1:variablesNumber;
            lb = zeros(variablesNumber,1);
            ub = ones(variablesNumber,1);

            if timeout <= 0
                timeout = 7200;
            end

            if verbose
                intlinprogOptions = ...
                    optimoptions('intlinprog',MaxTime = timeout);
            else
                intlinprogOptions = optimoptions( ...
                    'intlinprog',...
                    Display = 'off',...
                    MaxTime = timeout ...
                );
            end

            [x,v,exitflag] = ...
                intlinprog(C,intcon,A,b,[],[],lb,ub,intlinprogOptions);

            timedOut = exitflag == 2;
        end

    end
end
```

A.2 getAllImplicants

Funzione che utilizza l'algoritmo di Quine-McCluskey per trovare tutti gli implicanti e generare la matrice di copertura ad essi associata.

Listing A.2: Trova tutti gli implicanti

```
function [implicants,A] = ...
    getAllImplicants(inputsNumber,minterms,dontCares,options)

    arguments
        inputsNumber (1,1) double ...
            {mustBeInteger,mustBePositive}
        minterms (1,:) double ...
            {mustBeInteger,mustBePositive,utils.mustBeAscending}
        dontCares (1,:) double ...
            {mustBeInteger,mustBePositive,utils.mustBeAscending} = []

        options.Verbose (1,1) double {mustBeNumericOrLogical} = 0
    end

    % uses
    %   QM algorithm to retrieve a list of all implicants
    % returns
    %   implicants := list of all implicants generated using QM algorithm
    %   A := the coverage matrix

    notZeros = utils.mergeSorted(minterms,dontCares);

    if inputsNumber < utils.minimumBits(notZeros)
        error('Variables number: %d  is too small',inputsNumber)
```

```
    end

    if ~ utils.isInAscendingOrder(notZeros)
        error('Inputs minterms and dontCares have an element in common')
    end

    % given the indexes,get the value (-1) and then convert it to binary
    notZerosBinaries = string(dec2bin(notZeros - 1,inputsNumber));

    % first iteration of QM
    QM_iteration_count = 1;
    groups = cell(inputsNumber + 1,1);

    for i = 1:length(notZeros)

        % get group index from number of ones
        index = utils.countMatches(notZerosBinaries(i,:),"1") + 1;

        % insert the not_zero in the correct group
        groups{index} = [groups{index},notZerosBinaries(i,:)];

    end


    % at first
    %   implicants are minterms
    %   the A matrix is an identity, every implicant covers itself
    implicants = notZerosBinaries;
    A = eye(length(implicants));

    % other QM iterations

    while ~ utils.areAllCellsEmpty(groups)

        if options.Verbose
            fprintf('\nQM iteration %d:\n\n',QM_iteration_count)
            disp(groups)
        end

        nextIterationGroups = cell(inputsNumber + 1,1);

        % compare the i-group with the (i+1)-group
        % if one implicant merge :
        %   insert it to the next QM iteration i-group
        %   insert it into the implicant list and update A

        for i = 1:length(groups) - 1 ; group = groups{i};

            if isempty(group) ; continue ; end

            groupToCompareWith = groups{i + 1};

            if isempty(groupToCompareWith) ; continue ; end

            for j = 1:length(group)

                implicant = group(j);

                for k = 1:length(groupToCompareWith)

                    implicantToMatch = groupToCompareWith(k);

                    mergedImplicant = ...
                        utils.mergeStrings(implicant,implicantToMatch,'-');
```

```
                    % if the mergedImplicant is empty continue
                    if mergedImplicant == "" ; continue ; end

                    % if the mergedImplicant is redundant continue
                    if any(strcmp(nextIterationGroups{i},mergedImplicant))
                        continue;
                    end

                    % if mergedImplicant is correct add it to the list
                    implicants = [implicants ; mergedImplicant];

                    % insert a new line in the coverage matrix
                    A(length(implicants),1) = 0;

                    % get the two implicants' indexes
                    implicantIndex = ...
                        utils.findString(implicants,implicant{1});
                    toCompareWithIndex = ...
                        utils.findString(implicants,implicantToMatch{1});

                    % the mergedImplicant covers the OR bitmap
                    % of the two generator implicants by definitions
                    A(length(implicants),:) = ...
                        A(implicantIndex,:) | A(toCompareWithIndex,:);

                    % add the mergedImplicant in the nextIterationGroups
                    nextIterationGroups{i} = ...
                        [nextIterationGroups{i},mergedImplicant];

                end
            end
        end

        groups = nextIterationGroups;
        QM_iteration_count = QM_iteration_count + 1;
    end

    % remove dontCares costraints from coverage matrix
    % because they don't need to be covered
    for i = flip(dontCares)
        A(:,notZeros == i) = [];
    end

    % remove implicants that covered only dontCares
    for i = length(implicants):-1:1
        if ~ any(A(i,:) == 1)
            A(i,:) = [];
            implicants(i) = [];
        end
    end

    A = A.';

end
```

A.3 oneOutputSynthesis

Funzione che esegue la sintesi di una funzione booleana ad una uscita, utilizzando `intlinprog`.

Listing A.3: Sintesi ad un'uscita

```
function [implicants, v, timedOut] = ...
    oneOutputSynthesis(minterms, dontCares, options)

    arguments

        minterms (1,:) double ...
            {mustBeInteger, mustBePositive, mustBeNonempty, utils.
                mustBeAscending}
        dontCares (1,:) double ...
            {mustBeInteger, mustBePositive, utils.mustBeAscending}

        options.InputsNumber (1,1) double ...
            {mustBeInteger, mustBePositive} = utils.minimumBits( ...
                [minterms, dontCares] ...
            )
        options.GatesInputCost (1,1) double ...
            {mustBeNumericOrLogical} = 1
        options.Verbose (1,1) double ...
            {mustBeNumericOrLogical} = 0
    end

    % returns
    %   the minimal cost synthesis

    if options.Verbose
        fprintf('\nThe inputs number is:\n\n')
        disp(options.InputsNumber)
    end

    [implicants, A] = getAllImplicants( ...
        options.InputsNumber, ...
        minterms, ...
        dontCares, ...
        Verbose = options.Verbose ...
    );

    % literal cost for every AND port
    C = ones(length(implicants), 1);
    b = ones(length(minterms), 1);

    % diodes cost
    if options.GatesInputCost
        for i = 1:length(implicants)
            C(i) = utils.countMatches(implicants(i, :), "0" | "1") + 1;
        end
    end

    if options.Verbose
        fprintf('All possible implicants are:\n\n') ; disp(implicants)
        fprintf('The coverage matrix is:\n\n')      ; disp(A)
        fprintf('The cost vector is:\n\n')          ; disp(C.')
    end

    [x, v, timedOut] = ...
        utils.intlinprogWrap(C, -A, -b, length(implicants), options.Verbose
            , 0.1);

    % if it's literal cost add the OR one
    if ~ options.GatesInputCost
        v = v + 1;
    end

    if options.Verbose
```

```
        fprintf('Solution:\n\n')    ; disp(x.')
        fprintf('The cost is:\n\n') ; disp(v)
    end

    % remove all implicants that are not used
    for i = flip(x)
        implicants(x == 0) = [];
    end

end
```

A.4 multipleOutputSynthesis

Funzione che esegue la sintesi ottima di una funzione booleana multi-uscita, utilizzando `intlinprog`.

Listing A.4: Sintesi a più uscite

```
function [implicants,v,timedOut] = ...
multipleOutputSynthesis(inputsNumber,outputs,options)

    arguments
        inputsNumber (1,1) double ...
            {mustBeInteger,mustBePositive}
        outputs (1,:) cell

        options.GatesInputCost (1,1) double ...
            {mustBeNumericOrLogical} = 1
        options.Verbose (1,1) double ...
            {mustBeNumericOrLogical} = 0
        options.Timeout (1,1) double ...
            {mustBeGreaterThanOrEqual(options.Timeout,0)} = 0
    end

    % returns
    %   the minimal cost synthesis

    % set of all implicants
    implicantsSet = [];
    outputsImplicantsCount = zeros(length(outputs),1);

    A = [];
    totalMintermLength = 0;

    for i = 1:length(outputs) ; output = outputs(i);

        minterms = output{1}{1};
        dont_cares = output{1}{2};

        if ~ utils.isInAscendingOrder(minterms)
            error('Minterms must be in ascending order')
        end

        if ~ utils.isInAscendingOrder(dont_cares)
            error('Dont_cares must be in ascending order')
        end

        [implicants_k,A_k] = ...
            getAllImplicants(inputsNumber,minterms,dont_cares);

        outputsImplicantsCount(i) = length(implicants_k);
```

```matlab
        implicantsSet = [implicantsSet ; implicants_k];

        % every coverage matrix' costraints are indipendent
        A = blkdiag(A,A_k);

        totalMintermLength = totalMintermLength + length(minterms);

        if options.Verbose
            fprintf('\nOutput %d:\n\n',i);
            fprintf('All possible implicants are:\n\n')
            disp(implicants_k)
            fprintf('The coverage matrix is:\n\n')
            disp(A_k)
        end

    end

    uniqueImplicants = unique(implicantsSet);

    % add one more costraint for every uniqueImplicants
    %   every uniqueImplicants must be chosen if
    %   a corresponding implicant is chosen

    E = eye(length(uniqueImplicants));
    A = blkdiag(A,E);
    Phi = zeros(length(uniqueImplicants), length(implicantsSet));

    % choice matrix
    for i = 1:length(uniqueImplicants)

        uniqueImplicant = uniqueImplicants(i);

        Phi(i, implicantsSet == uniqueImplicant) = ...
            -1 / (length(outputs) + 1);
    end

    A(totalMintermLength + 1:end,1:length(implicantsSet)) = Phi;

    variablesLength = length(implicantsSet) + length(uniqueImplicants);

    % literal cost
    % only the uniqueImplicants must be considered in the cost
    C = ones(variablesLength,1);
    C(1:length(implicantsSet)) = 0;

    % in the choice costraints the costant value is 0
    % in the cover constraints the costant value is 1
    b = zeros(totalMintermLength + length(uniqueImplicants),1);
    b(1:totalMintermLength) = 1;

    % gateInput cost
    if options.GatesInputCost

        % 1 every time a port is used in an output
        for i = 1:length(implicantsSet)
            C(i) = 1;
        end

        % c_i every time a port is chosen
        for i = 1:length(uniqueImplicants)
            C(i + length(implicantsSet)) = ...
                utils.countMatches(uniqueImplicants(i,:), "0" | "1");
        end
    end
```

```

    if options.Verbose
        fprintf('The not redundant implicants are:\n\n')
        disp(uniqueImplicants)

        fprintf('The choice matrix is (one implicant each column):\n\n')

        format bank
        disp(-Phi.')
        format default

        fprintf('The cost vector is:\n\n')
        disp(C.')
    end

    [x,v,timedOut] = utils.intlinprogWrap( ...
        C, ...
        -A, ...
        -b, ...
        variablesLength, ...
        options.Verbose, ...
        options.Timeout ...
    );

    % if it's literal cost add 1 for, everty OR/output
    if ~ options.GatesInputCost
        v = v + length(output);
    end

    if options.Verbose
        fprintf('Solution:\n\n')
        disp(x.')
        fprintf('The cost is:\n\n')
        disp(v)
    end

    % build the result implicants

    implicants = cell(length(outputs),1);

    outputIndex = 1;
    outputImplicantsIndex = 1;

    % cycle through every implicant
    for i = 1:length(implicantsSet)

        if outputImplicantsIndex > outputsImplicantsCount(outputIndex)
            outputImplicantsIndex = 1;
            outputIndex = outputIndex + 1;
        end

        % if the ith implicant is has been chosen add it
        if x(i)

            % if the outputImplicantsIndex overflow got to the next output

            implicants{outputIndex} = ...
                [implicants{outputIndex} ; implicantsSet(i)];
        end

        % increment the current outputImplicantsIndex
        outputImplicantsIndex = outputImplicantsIndex + 1;

    end
```

```
end
```

A.5 displayImplicants

Funzione che visualizza gli implicanti in formato leggibile, riceve in input gli implicanti in questo formato:

- 1 se la variabile in ingresso è attiva
- 0 se la variabile in ingresso è disattiva
- - se la variabile in ingresso non è considerata

Listing A.5: Visualizzazione degli implicanti

```
function displayImplicants(outputs_implicants,options)
    arguments
        outputs_implicants (1,:) cell

        options.LatexSyntax (1,1) double ...
            {mustBeNumericOrLogical} = 0
    end

    if options.LatexSyntax
        error('unimplemented')
    end

    fprintf('Solution: ')

    for i = 1:length(outputs_implicants)
        implicants = outputs_implicants{i};

        fprintf('[');

        for j = 1:length(implicants)

            implicant = implicants{j};

            for k = 1:length(implicant)

                letter = implicant(k);

                if strcmp(letter,'-') ; continue ; end

                fprintf('x%d',length(implicant) - k + 1)

                if strcmp(letter,'1') ; continue ; end

                fprintf('`')
            end

            if j == length(implicants) ; continue ; end
            fprintf(' + ')

        end

        fprintf(']');
```

```
    end
    fprintf('\n')
end
```

A.6 synthesisCheck

Funzione che dati degli implicanti, dei mintermini e dei "non specificato" controlla che la tabella di verità generata dagli implicanti corrisponda a quella richiesta.

Listing A.6: Controllo di una sintesi

```
function check = synteshisCheck(implicants, minterms, dontCares)
    arguments
        implicants (1,:) string ...
            {mustBeNonempty}

        minterms (1,:) double ...
            {mustBeInteger, mustBePositive, mustBeNonempty, utils.mustBeAscending} = []

        dontCares (1,:) double ...
            {mustBeInteger, mustBePositive, utils.mustBeAscending} = []
    end

    inputsNumber = strlength(implicants(1));
    check = true;
    truthTable = zeros(2 ^ inputsNumber);

    for i=1:length(implicants)

        implicant = implicants(i);

        for minterm=1:length(truthTable)

            if isCovered(dec2bin(minterm - 1, inputsNumber), implicant{1})
                truthTable(minterm) = 1;
            end
        end
    end

    for i=1:length(truthTable)

        % if this element don't care skip it
        if any(dontCares == i) ; continue ; end

        % if it's a one
        if truthTable(i) == 1

            % check wheter it's wrong
            if ~ any(minterms == i)
                check = false;
                return
            end

            continue
        end

        % if it's a zero
        % check wheter it's wrong

```

```
        if any(minterms == i)
            check = false;
            return
        end

    end

end


function res = isCovered(minterm, implicant)

    res = true;

    for i=1:length(minterm)

        if implicant(i) == '-'; continue ; end
        if implicant(i) ~= minterm(i)
            res = false;
            return
        end
    end

end
```

A.7 statistics

Script che stampa a video i valori dei miglioramenti percentuali ottenuti dalla sintesi di funzioni booleane multi-uscita con un certo numero di uscite ed entrate.

Listing A.7: Script per generare le statistiche

```
addpath(genpath("./../"))

rng('shuffle')

MAX_TEST_NUMBER = 420;
TIMEOUT = 23 * 60;

% if this variables are not set, set them to default values
if exist('outputsNumber','var') ~= 1 ; outputsNumber = 8 ; end
if exist('inputsNumber','var') ~= 1 ; inputsNumber = 4 ; end
if exist('lastTest','var') ~= 1 ; lastTest = 1 ; end

biggest_value = 2^inputsNumber;

for test = lastTest:MAX_TEST_NUMBER

    outputs = cell(outputsNumber, 1);
    one_out_cost = 0;

    % one output synthesis

    for i = 1:outputsNumber

        p = randperm(biggest_value);

        minterms_count = round(biggest_value * rand(1,1));

        while minterms_count < 1
            minterms_count = round(biggest_value * rand(1,1));
```

```
        end

        dontcares_count = round(biggest_value * rand(1,1));

        while (minterms_count + dontcares_count) > biggest_value
            dontcares_count = round(biggest_value * rand(1,1));
        end

        p_1 = sort(p(1:minterms_count));

        if minterms_count < biggest_value
            p_2 = sort( ...
                p(minterms_count + 1: minterms_count + dontcares_count) ...
            );
        else
            p_2 = [];
        end

        out = {p_1, p_2};

        [~, v, timedOut] = oneOutputSynthesis( ...
            out{1}, ...
            out{2}, ...
            InputsNumber = inputsNumber, ...
            Timeout = TIMEOUT ...
        );

        if timedOut ; break ; end

        one_out_cost = one_out_cost + v;

        outputs{i} = out;
    end

    if timedOut ; fprintf('%d timed_out\n', test) ; continue ; end

    tStart = tic;

    % multiple output synthesis

    [~, multiple_out_cost, timedOut] = multipleOutputSynthesis( ...
        inputsNumber, ...
        outputs, ...
        Timeout = TIMEOUT ...
    );

    elapsedTime = toc(tStart);

    if timedOut ; fprintf('%d timed_out\n', test) ; continue ; end

    fprintf( ...
        '%d %f %f %.20f\n', ...
        test, ...
        one_out_cost, ...
        multiple_out_cost, ...
        elapsedTime...
    )

end
```

A.8 plotStatistics

Script che visualizza i miglioramenti percentuali in un grafico.

Listing A.8: Script per visualizzare le statistiche

```

delimiterIn = ' ';
maxInputsNumber = 6;
maxOutputsNumber = 16;

avgs = zeros(maxInputsNumber,maxOutputsNumber);
std_devs = zeros(maxInputsNumber,maxOutputsNumber);
errors = zeros(maxInputsNumber,maxOutputsNumber);
worst_error = 0;
best_save = 0;

% ALPHA = 0.95;
QUANTILE = 1.96;

for o = 1:maxOutputsNumber
    for i = 1:maxInputsNumber

        filename = sprintf('out_web/%d-%d.log', o, i);
        A = importdata(filename,delimiterIn);

        savings = (A(:,2) - A(:,3)) ./ A(:,2) * 100;


        avgs(i,o) = mean(savings);

        if mean(savings) > best_save
            best_save = mean(savings);
        end

        std_dev = sqrt(var(savings));
        error = std_dev / sqrt(length(savings)) * QUANTILE;

        if isnan(std_dev) ; continue ; end

        std_devs(i,o) = std_dev;
        errors(i,o) = error;

        if error > worst_error ; worst_error = error; end

    end
end

input_legends = cell(1, maxInputsNumber);

for i = 1:2:maxInputsNumber * 2
    input_legends{i} = sprintf('%d', round(i/2));
    input_legends{i + 1} = '';
end

hold on

gaps = errors;

for i = 1:maxInputsNumber

    p = plot(avgs(i,:));

    p(1).LineWidth = 2;
```

```

    c = get(p,'Color');

    x = 1:length(avgs(i,:));
    curve1 = avgs(i,:) + gaps(i,:);
    curve2 = avgs(i,:) - gaps(i,:);
    x2 = [x, fliplr(x)];
    inBetween = [curve1, fliplr(curve2)];
    h = fill(x2, inBetween, c, 'LineStyle','none');
    h.FaceAlpha = 0.2;

end

grid on

ax = gca;
ax.FontSize = 18;

lgd = legend(input_legends{:}, 'Location','northwest', 'FontSize', 18);
title(lgd,' Entrate ', 'FontSize', 22)


xlabel({'Uscite', '',}, 'FontSize', 18);
ylabel({'Risparmio [%]', ''}, 'FontSize', 18);

fprintf('The worst error is %f\n', worst_error);
fprintf('The best median saving is %f\n', best_save);
```

A.9 distribution

Script che visualizza i miglioramenti di un set di test, dati numero di uscite e numero di entrate, e confronta la loro distribuzione con quella di una gaussiana.

Listing A.9: Script per controllare la distribuzione di probabilità dei test

```

delimiterIn = ' ';

% if this variables are not set, set them to default values
if exist('outputsNumber','var') ~= 1 ; outputsNumber = 8 ; end
if exist('inputsNumber','var') ~= 1 ; inputsNumber = 4 ; end

filename = sprintf('folder/%d-%d.log', outputsNumber, inputsNumber);
A = importdata(filename,delimiterIn);

savings = sort((A(:,2) - A(:,3)) ./ A(:,2) * 100);

h = histfit(savings);

h(1).FaceColor = [0.4 0.6 0.8];
h(1).FaceAlpha = 0.5;

h(2).Color = [.2 .2 .2];

ax = gca;
ax.FontSize = 18;
```

Appendice B

Sintesi a singola uscita mediante metodi classici

B.1 Espansione di Shannon

Il teorema di Shannon afferma che è sempre possibile scrivere qualunque legge f, di una rete combinatoria, come somma di prodotti degli ingressi (diretti o negati).

È possibile quindi ricavare l'espansione di Shannon come:

$$
\begin{aligned}
y = f(0, \ldots, 0, 0) \cdot \overline{x_N} \cdot \ldots \cdot \overline{x_2} \cdot \overline{x_1} + \\
f(0, \ldots, 0, 1) \cdot \overline{x_N} \cdot \ldots \cdot \overline{x_2} \cdot x_1 + \\
\vdots \qquad\qquad \\
f(1, 1, \ldots, 1) \cdot x_N \cdot \ldots \cdot x_2 \cdot x_N
\end{aligned}
$$

B.2 Forma canonica SP

Dall'espansione di Shannon è possibile ricavare la forma canonica SP, selezionando solo gli stati riconosciuti dalla rete. Ciascun addendo della forma canonica SP è detto mintermine.

B.2.1 Implicanti principali

Per realizzare la sintesi a costo minimo è necessario fondere più mintermini possibili, utilizzando le regole dell'algebra di Boole, in modo da utilizzare meno porte possibili nella sintesi.

Dalla fusione di più mintermini nasce un implicante, esso ha un grado che aumenta con il numero di fusioni da cui è stato formato (un mintermine è un implicante di grado 1).

Un implicante che non si può più fondere è detto principale, non sempre la sintesi formata dalla somma di implicanti principali è non ridondante e quindi a costo minimo.

Per avere una sintesi a costo minimo è quindi necessario eliminare tutti gli implicanti ridondanti.

B.3 Lista di copertura

Una lista di copertura è una lista di implicanti, la cui somma è una forma SP per una funzione booleana, esempio:

- lista dei mintermini
- lista degli implicanti
- lista degli implicanti principali

La sintesi a costo minimo è costituita dall'insieme di implicanti principali che, avendo il costo sommato minimo, coprono interamente il circuito, questo insieme viene chiamato "lista di copertura a costo minimo".

B.3.1 Eliminazione degli implicanti principali ridondanti

Una volta enumerati gli implicanti principali è necessario rimuovere quelli ridondanti, questo avviene attraverso il seguente algoritmo:

- Si trovano gli implicanti essenziali, sono quelli che coprono stati coperti solo da loro
- Si trovano gli implicanti assolutamente eliminabili, sono quelli che coprono solo stati coperti da implicanti essenziali
- Si trovano gli implicanti semplicemente eliminabili, sono quelli che coprono stati non coperti da implicanti essenziali, ma coperti da altri implicanti semplicemente eliminabili
- Si valutano tutte le possibili liste di copertura
- Si seleziona la lista di copertura a costo minimo

Bibliografia

[1] M.Passacantando M.Pappalardo. *Ricerca Operativa*. 2013.

[2] Maurizio Pratelli. *Un corso di Probabilità e Statistica*. https://people.dm.unipi.it/pratelli/Informatica/Didattica/Appunti%20Statistica.pdf. 2021.

[3] Goutam Saha. *Lecture 13: Cost Criteria and Minimization of Multiple Output Functions*. Youtube, 2019. URL: https://www.youtube.com/watch?v=NgFW-3_w_W4 (cit. a p. 21).

[4] Giovanni Stea. *Appunti sulle Reti Combinatorie*. http://docenti.ing.unipi.it/m.cococcioni/siselab/Stea_Reti_Combinatorie.pdf. 2018.

Ringraziamenti

Ringrazio il mio relatore Marco Cococcioni, che mi ha fornito un tema di tesi appassionante e non scontato, seguendomi nel percorso di sviluppo con cura.

Ringrazio i miei genitori, amici, zii, cugini, parenti, cumpar' e nipot' che da vicino e da lontano mi hanno accompagnato fino a questo punto.

Un ringraziamento particolare alla mia correttrice di bozza per il suo aiuto, ma in realtà un po' per tutto.

Da qualche parte c'è un punto e virgola che sta tutto kekkato.