



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

Sommario

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

AV: scrivere l'abstract non in latino

Keywords: Multiple-Input Multiple-Output Boolean Function Minimization, Integer Linear Programming, Quine-McCluskey Method.

Indice

1	Introduzione	7
1.1	Sintesi di funzioni booleane	7
1.1.1	Le reti combinatorie	7
1.1.2	Algebra di Boole	8
1.1.3	Sintesi in forma SP a costo minimo	9
1.1.4	Algoritmi di ricerca degli implicant principali	10
1.1.5	Sintesi in forma PS	11
1.2	Programmazione lineare	12
1.3	Premesse	12
1.3.1	Scelta della forma canonica	12
1.3.2	Scelta dell'utilizzo della PLI anche per il problema ad una uscita	13
1.3.3	Utilizzo di ogni impicante nella formulazione delle variabili . .	13
2	Problema ad una uscita	15
2.1	Modellazione del problema	16
2.2	Definizione delle variabili	16
2.3	Definizione dei vincoli	16
2.3.1	Vincoli di copertura	16
2.3.2	Vincoli di interezza	17
2.4	Funzione obiettivo	17
2.4.1	Costo a porte	17
2.4.2	Costo a diodi	18
2.5	Modello matematico (costo a diodi)	18
2.5.1	Generico	18
2.5.2	In formato primale standard	18
3	Problema a più uscite	19
3.1	Modellazione del problema	20
3.2	Definizione delle variabili	20
3.3	Definizione dei Vincoli	23
3.3.1	Vincoli di copertura	23
3.3.2	Vincoli di scelta	23
3.3.3	Vincoli di interezza	24
3.4	Funzione obiettivo	24

3.4.1	Costo a porte	24
3.4.2	Costo a diodi	24
3.5	Modello matematico (costo a diodi)	24
3.5.1	Generico	24
3.5.2	In formato primale standard	25
4	Esempi esplicativi	27
4.1	Implicante in comune	27
4.2	Scelta di un implicante influenzata da un altro output	28
4.3	Scelta di un implicante non principale	30
4.4	Uso dei "non specificato"	31
5	Esempio di funzionamento	35
5.1	Input	35
5.2	Sintesi della prima uscita	35
5.3	Sintesi di entrambe le uscite	37
6	Conclusioni	41
7	Codice	43
7.1	Utils	43
7.2	getAllImplicants	46
7.3	oneOutputSynthesis	48
7.4	multipleOutputSynthesis	50
7.5	displayImplicants	53
7.6	synthesisCheck	54
7.7	statistics	55
7.8	plotStatistics	56
7.9	distribution	58

Notazione

- *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 al suo interno

Caso ad una 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 a più uscite

- 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, 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

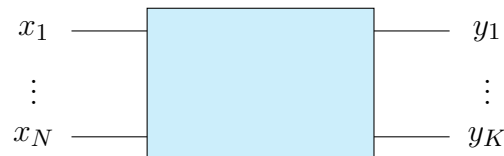
Introduzione

In seguito saranno introdotti i concetti necessari per studiare il problema della sintesi di funzioni booleane multi-uscite.

1.1 Sintesi di funzioni booleane

1.1.1 Le 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 di uscita.



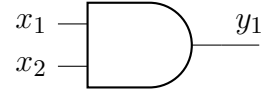
Una rete combinatoria può essere descritta mediante una tabella di verità in cui sono presenti a sinistra l'insieme dei possibili stati di ingresso e a destra l'insieme degli stati di uscita corrispondenti.

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

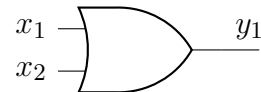
Porta AND

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



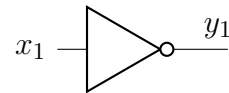
Porta OR

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



1.1.2 Algebra di Boole

È un sistema algebrico basato su variabili logiche e operatori logici utilizzato per la descrizione di reti combinatori.

Gli operatori logici 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 che del prodotto
- Complementazione $x \cdot \bar{x} = 0 \quad x + \bar{x} = 1$
- Unione e intersezione $x + 0 = x, x + 1 = 1 \quad x \cdot 0 = 0, x \cdot 1 = x$
- Idempotenza $x + x = x \quad x \cdot x = x$

Legge di De Morgan

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

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

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

si vuole sintetizzare, partendo da una funzione booleana ad una uscita complessa, un circuito che abbia due livelli di logica e che sia formato da sole porte AND e OR. Questo in algebra di Boole corrisponde ad una funzione booleana formata solo da somme di prodotti.

Per eseguire la sintesi di funzioni booleane a più uscite, tradizionalmente, viene eseguita la sintesi considerando ogni uscita come una funzione a sé stante e formando il circuito finale come unione dei circuiti risultanti (questo metodo non è sicuramente ottimale).

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 :

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

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.

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.

Un implicante è un insieme di mintermini che riconoscono diversi stati di una funzione booleana, 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 è detto che la sintesi formata dalla somma di implicanti principali sia non ridondante e quindi a costo minimo.

Eliminazione degli implicanti principali ridondanti

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

- Si trovano gli implicanti essenziali (quelli che coprono stati coperti solo da loro)
- Si trovano gli implicanti assolutamente eliminabili (quelli che coprono solo stati coperti da implicanti essenziali)
- Si trovano gli implicanti semplicemente eliminabili (quelli che coprono stati non coperti da implicanti essenziali, ma coperti da altri implicanti semplicemente eliminabili)
- Si valutano tutte le possibili liste di copertura (formate dagli implicanti essenziali e da una combinazione di implicanti semplicemente eliminabili)
- Si sceglie la lista di copertura a costo minimo

1.1.4 Algoritmi di ricerca degli implicanti principali

Entrambi gli algoritmi mostrati utilizzano un diverso approccio per la ricerca degli implicanti principali senza utilizzare l'espansione di Shannon.

Algoritmo di Quine-McCluskey

- Si raggruppano in mintermini in base al numero di "1" all'interno di essi
- Vengono fatte delle fusioni tra i gruppi adiacente, se due implicanti differiscono per una sola variabile fondono
- Si creano altri gruppi con gli implicanti fusi, senza ripetizioni e raggruppando i nuovi implicanti secondo la regola iniziale
- si itera finché i gruppi della prossima iterazione non sono vuoti separandoli in partizioni distinte

Alla fine, gli implicantanti che non hanno generato fusione sono quelli principali.
Esempio:

x_3	x_2	x_1	y_1		x_3	x_2	x_1		x_3	x_2	x_1
0	0	0	0		0	0	1		0	—	1
0	0	1	1		0	1	0		—	0	1
0	1	0	1	\Rightarrow	0	1	1	\Rightarrow	0	1	—
0	1	1	1		1	0	1				
1	0	0	0								
1	0	1	1								
1	1	0	0								
1	1	1	0								

Possiamo vedere che gli unici implicantanti che non fondono sono gli $\overline{x_3}x_1$, $\overline{x_2}x_1$, $\overline{x_3}x_2$ ovvero quelli dell'ultima iterazione, può succedere però che non fondano implicantanti in iterazioni precedenti, in quel caso anch'essi sono principali.

Mappe di Karnaugh

Data una tabella di verità essa può essere espressa sotto forma di mappe 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				

L'algoritmo consiste nel:

- Cercare tutti gli implicantanti di ordine maggiore, non già completamente coperti
- Se con essi copro tutta la mappa ho finito altrimenti controllo l'ordine inferiore

Gli implicantanti principali consistono in tutti quegli implicantanti necessari per coprire la mappa, selezionando tutti gli implicantanti del grado corrente (possono esserci ridondanze che vengono eliminate in seguito).

1.1.5 Sintesi in forma PS

Per ogni funzione booleana oltre alla sintesi in forma SP esiste anche quella in forma PS (prodotto di somme). Non è detto che il costo delle due sintesi sia uguale, anzi

quasi sempre è diverso, per questo per sapere qual'è la sintesi a costo minimo su due livelli di logica è necessario eseguire entrambe le sintesi e vedere quella a costo minore

algoritmo

- data la funzione f mi ricavo la funzione \bar{f} che fa corrispondere ad ogni stato di ingresso di f il suo complemento
- realizzo la sintesi SP di \bar{f}
- ottengo la sintesi di f inserendo un invertitore in uscita alla rete ottenuta
- applico il teorema di De Morgan in modo da ricavarmi l'espressione sotto forma di prodotti di somme

1.2 Programmazione lineare

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

Ogni problema di PL è composto da:

- una funzione obiettivo lineare
- un insieme di vincoli lineari

Ogni problema di PL può essere trasformato in formato primale standard, questo ci è comodo perché così lo stesso algoritmo di risoluzione (il simplesso) può essere utilizzato per ciascun problema di PL.

Formato primale standard

$$\begin{cases} \min CX \\ AX \leq 1 \end{cases}$$

Programmazione lineare intera

La programmazione lineare (PLI) comprende tutti i problemi lineari le cui variabili hanno vincoli di interezza.

1.3 Premesse

1.3.1 Scelta della forma canonica

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

Da quanto precedentemente scritto, possiamo evincere che la risoluzione del problema della sintesi in forma SP permette, a patto di seguire l'algoritmo trattato, anche quello della risoluzione del problema in forma PS. Perciò d'ora in avanti tratteremo solo il problema della sintesi in forma SP.

1.3.2 Scelta dell'utilizzo 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 la soluzione ottima ed hanno bassa complessità di esecuzione e tempi di risoluzione brevi.

Il motivo per cui la sintesi di funzioni booleane ad una uscita viene affrontato come un problema di PLI è per trattare in seguito il problema a più uscite con più facilità.

1.3.3 Utilizzo di ogni implicante nella formulazione delle variabili

Al contrario del caso ad una uscita, nella sintesi di funzioni booleane a più uscite è possibile che convenga sintetizzare delle porte che per un uscita sono descritte da implicanti non principali, questo sarà evidente nell' **esempio 4.3**.

Capitolo 2

Problema ad una uscita

Data una funzione combinatoria ad una uscita del tipo:

$$y_1 = f(x_1, x_2, \dots, x_N)$$

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

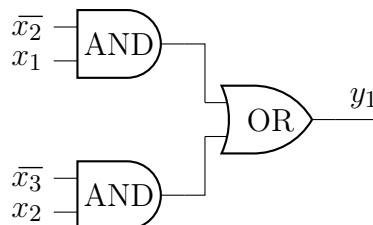
- numero di porte
- numero di diodi

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

Quindi il costo a porte può essere ottenuto come il numero di implicant necessari per la sintesi più uno.

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

Esempio:



Nel circuito in figura il costo a porte è tre, di cui due per il numero di porte AND e uno per la porta OR. Il costo a diodi è il numero di ingressi di ciascuna porta: due per ciascuna porta AND (quattro) più due 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 QM.

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 viene creata una matrice di copertura, che 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 possiamo 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

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

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

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

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:

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

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.

		x_1	
		0	1
x_2	0	1	-
	1	-	0

Nella mappa di Karnaugh sovrastante 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.

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

2.4 Funzione obiettivo

La funzione obiettivo varia in base al criterio di costo scelto. In seguito sono presenti 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 $C = [1, \dots, 1]$, di dimensione J .

Il costo di ciascuna porta è 1:

- quello delle porte AND viene considerato nel caso vengano scelte
- mentre 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 $C = [c_1, c_2, \dots, c_J, 1, \dots, 1]$, di dimensione $2J$.

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

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

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

2.5 Modello matematico (costo a diodi)

2.5.1 Generico

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

2.5.2 In formato primale standard

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

Capitolo 3

Problema a più uscite

Data una funzione combinatoria a più uscite del tipo:

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

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

- numero di porte
- numero di diodi

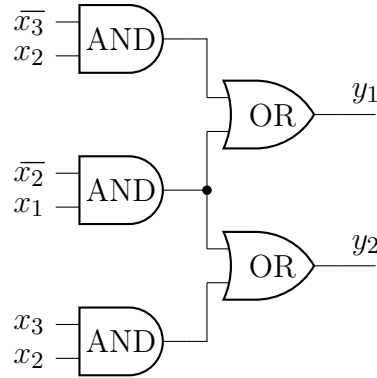
Il costo a porte in questo caso equivale al numero totale di porte AND utilizzate più il numero di porte OR (che corrisponde al numero delle uscite).

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

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

L'ottimizzazione che cerchiamo di raggiungere avviene sintetizzando il circuito come se l'insieme di tutte le uscite fosse un circuito unico, a differenza del metodo euristico che consiste nel sintetizzare ciascuna uscita come se fosse un circuito a sé stante e costruire poi il circuito finale come somma dei circuiti a sé stanti.

Esempio:



In questo esempio è evidente come la porta centrale venga riutilizzata, per questo il costo a porte equivale a tre porte AND più due porte OR, una porta AND in meno rispetto alla sintesi tradizionale.

Il costo a diodi equivale a due per ciascuna porta AND (sei) più due per ciascuna porta OR (quattro).

3.1 Modellazione del problema

Per risolvere il problema della sintesi di funzioni booleane a più uscite è stato ampliato il problema visto in precedenza.

Rispetto al problema precedente, il nuovo criterio da tenere in considerazione per la scelta delle porte, 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 utilizzeremo delle variabili ausiliarie, le quali indicano se una porta è stata impiegata o meno all'interno del circuito.

Il problema che ne risulta è ridurre al minimo il costo relativo alle porte considerando una singola volta, 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 implicant in un uscita
- U numero totale di implicant delle uscite
- L numero totale di implicant nel circuito

Si definiscono:

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

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

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 è 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 $V = [v_1, v_2, \dots, v_U]^T$ con indice u e dimensione U , dove U è calcolata secondo 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.

- 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

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

- 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:

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

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

y_1

		x_1	
		0	1
x_2	0	1	-
	1	-	0

y_2

		x_1	
		0	1
x_2	0	-	1
	1	0	-

Il vettore V^1 avrà dimensione tre e gli implicanti ad esso associato saranno $[\bar{x}_2, \bar{x}_1, \bar{x}_2\bar{x}_1]$.

Il vettore V^2 avrà dimensione tre e gli implicanti ad esso associato saranno $[\bar{x}_2, x_1, \bar{x}_2x_1]$

Il vettore V formato dal concatenamento dei V^k avrà dimensione sei e gli implicanti ad esso associato saranno $[\bar{x}_2, \bar{x}_1, \bar{x}_2\bar{x}_1, \bar{x}_2, x_1, \bar{x}_2x_1]$.

Il vettore Z , che indica la scelta o meno degli implicanti all'interno del circuito, avrà dimensione cinque e gli implicanti ad esso associato saranno $[\bar{x}_2, \bar{x}_1, \bar{x}_2\bar{x}_1, x_1, \bar{x}_2x_1]$.

Si può vedere come \bar{x}_2 sia presente una volta sola, questo perché gli elementi z_l si riferiscono agli implicantti utilizzabili all'interno di tutto il circuito, senza considerare eventuali ripetizioni tra le uscite.

In questo caso la matrice Φ sarà del tipo:

$$\begin{array}{c}
 \begin{array}{c} Z \\ \bar{x}_2 \\ \bar{x}_1 \\ \bar{x}_2\bar{x}_1 \\ x_1 \\ \bar{x}_2x_1 \end{array}
 \end{array}
 \begin{array}{c}
 V \\
 \begin{array}{ccccccc}
 \bar{x}_2 & \bar{x}_1 & \bar{x}_2\bar{x}_1 & \bar{x}_2 & x_1 & \bar{x}_2x_1 \\
 1 & 0 & 0 & 1 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 0 & 1
 \end{array}
 \end{array}$$

Si può vedere come visto che l'implicante \bar{x}_2 è presente in entrambe le uscite avrà due 1 nella prima riga, ad indicare che z_1 , $v_1(v_1^1)$ e $v_4(v_1^2)$ sono associate alla scelta dello stesso implicante.

3.3 Definizione dei Vincoli

3.3.1 Vincoli di copertura

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

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

3.3.2 Vincoli di scelta

Le variabili z_l devono essere settate ad 1 nel caso in cui l'implicante l-esimo sia scelto in almeno una delle uscite, a 0 altrimenti.

Quindi il loro valore si può ricavare attraverso l'OR logico delle variabili v_u che corrispondono all'implicante da cui dipende la variabile z_l .

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

Questo, utilizzando solo vincoli lineari, è raggiungibile imponendo:

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

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

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

3.3.3 Vincoli di interezza

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

3.4 Funzione obiettivo

3.4.1 Costo a porte

$$\min CZ + K$$

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

3.4.2 Costo a diodi

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

dove $C = [1, \dots, 1 \mid c_1, c_2, \dots, c_L]$ è di dimensione $U + L$, dove il costo c_l indica il costo a diodi di ciascun implicante (porta AND).

Il costo dei diodi in ingresso alle porte AND viene considerato una sola volta e solo nel caso in cui vengano scelte. Per quanto riguarda i diodi in ingresso ad ogni porta OR si può notare come esso corrisponda al numero di porte AND scelte per ciascun uscita.

3.5 Modello matematico (costo 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

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

Capitolo 4

Esempi esplicativi

4.1 Implicante in comune

In questo esempio possiamo vedere che sintetizzando due uscite con un implicante in comune utilizzando la sintesi a più uscite esso viene riutilizzato, mentre nella sintesi ad un'uscita il costo di esso sarà considerato due volte.

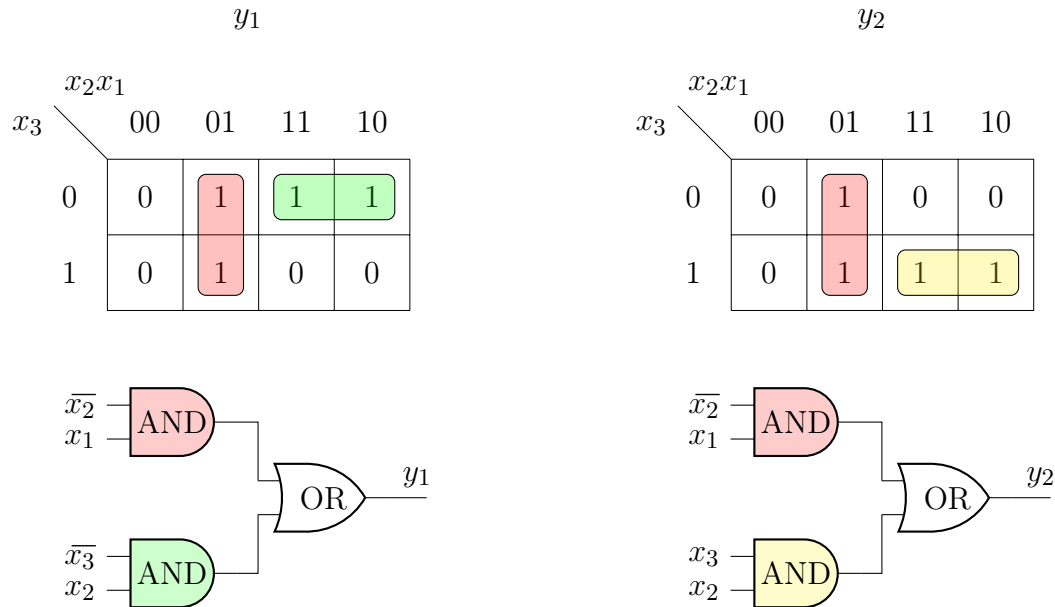
Listing 4.1: Codice Esempio 1

```
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 improved by %.2f%% ', savings)
```

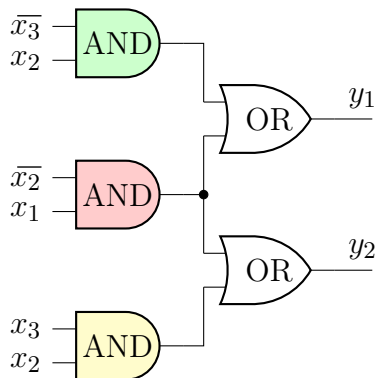


Di seguito possiamo vedere l'output della sintesi eseguita dal programma, è evidente come l'implicante \bar{x}_2x_1 sia utilizzato due volte per la sintesi a più uscite.

Il costo a diodi complessivo è stato ridotto 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 improved by 16.67%
```



4.2 Scelta di un implicante influenzata da un altro output

In questo esempio possiamo vedere che la scelta o meno di un implicante può 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 improved by %.2f%% ', savings)

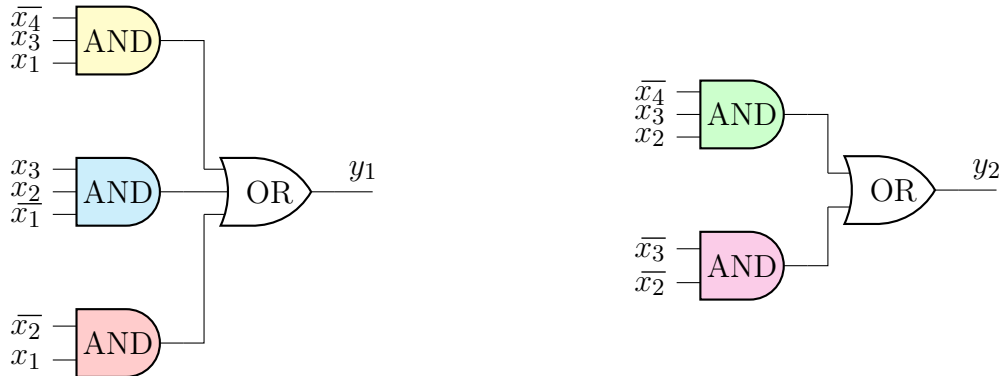
```

y_1

	x_2x_1	00	01	11	10
x_4x_3	00	0	1	0	0
	01	0	1	1	1
	11	0	1	0	1
	10	0	1	0	0

y_2

	x_2x_1	00	01	11	10
x_4x_3	00	1	1	0	0
	01	0	0	1	1
	11	0	0	0	0
	10	1	1	0	0



Si può notare come la scelta dell'implicante $\bar{x}_4x_3x_2$ sia vantaggiosa in quanto esso deve comunque essere sintetizzato perché presente nell'altra uscita.

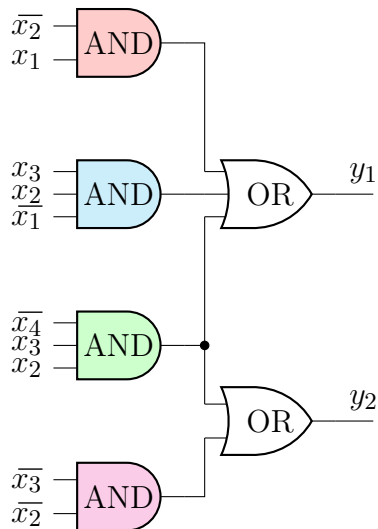
Il costo a diodi complessivo è stato ridotto del 16.67%.

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 improved by 16.67%

```



4.3 Scelta di un implicante non principale

In questo esempio possiamo notare che la scelta migliore è quella di utilizzare un implicante non principale in quanto esso è necessario per la sintesi un'altra uscita.

Listing 4.5: Codice Esempio 3

```

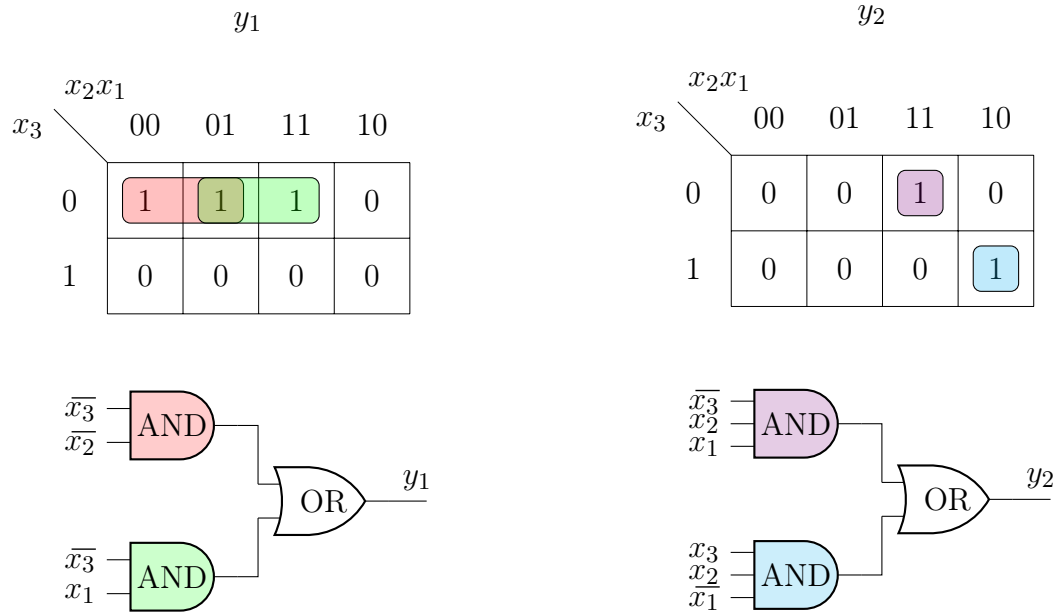
y_1 = {[0,1,3] + 1, []};
y_2 = {[3,6] + 1, []};

[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 improved by %.2f%% ', savings)

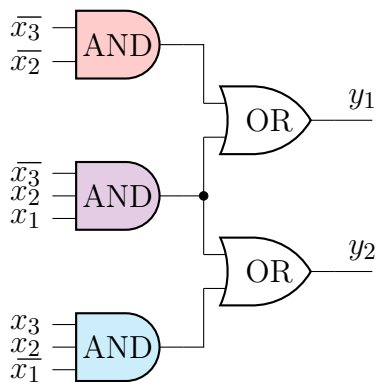
```



L'implicante $\bar{x}_3x_2x_1$ 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 improved by 14.29%
```



4.4 Uso dei "non specificato"

In questo esempio possiamo vedere come l'uso dei non specificato influenza la scelta degli implicanti nel caso della sintesi a più uscite, è possibile infatti che convenga sintetizzare implicanti più piccoli (e quindi più costosi) in modo da riutilizzarli all'interno di più uscite.

Listing 4.7: Codice Esempio 4

```

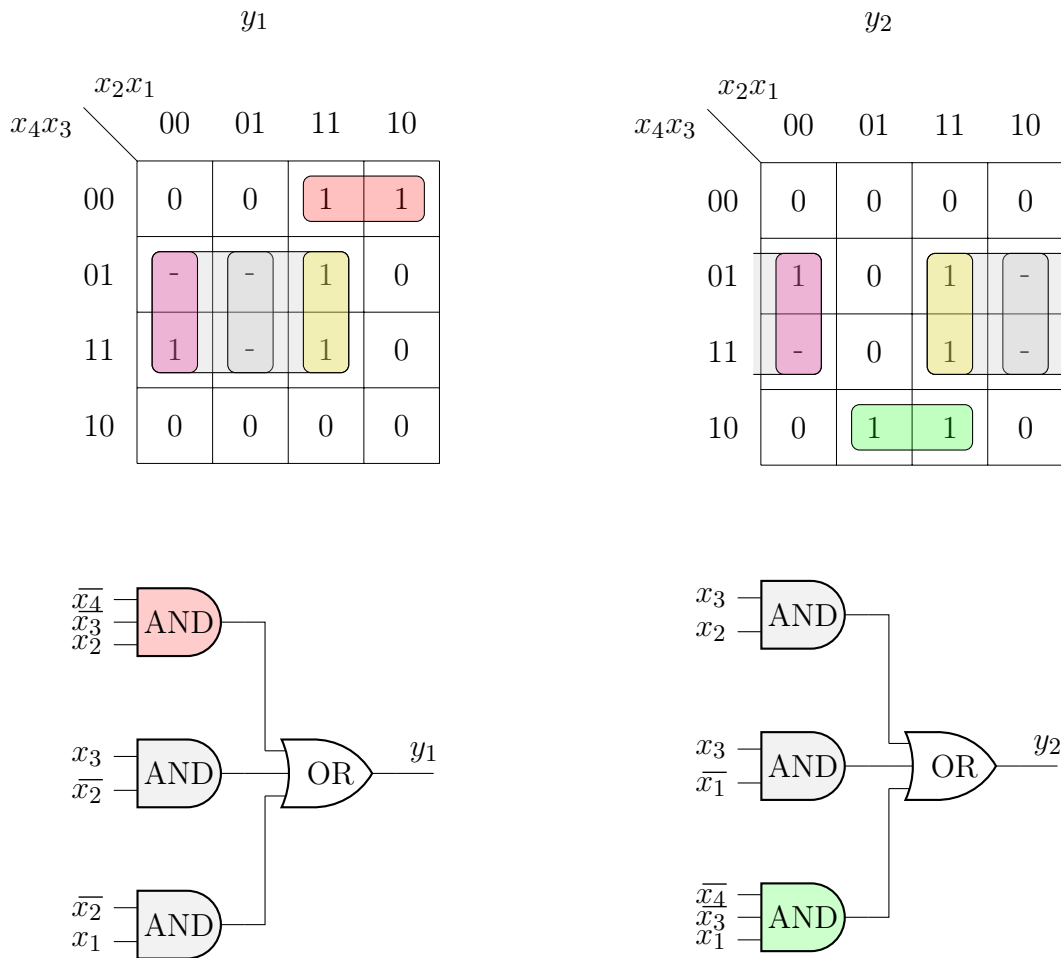
y_1 = {[2,3,7,12,15] + 1, [4,5,13] + 1};
y_2 = {[4,7,9,11,15] + 1, [6,12,14] + 1};

[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 improved by %.2f%% ', savings)

```



Si può notare come nonostante nella sintesi ad un'uscita convenga sintetizzare gli implicant di ordine 2, nella sintesi a più uscite la scelta migliore sia quella di riutilizzare gli implicant di ordine 1.

Avendo così un miglioramento del 10%.

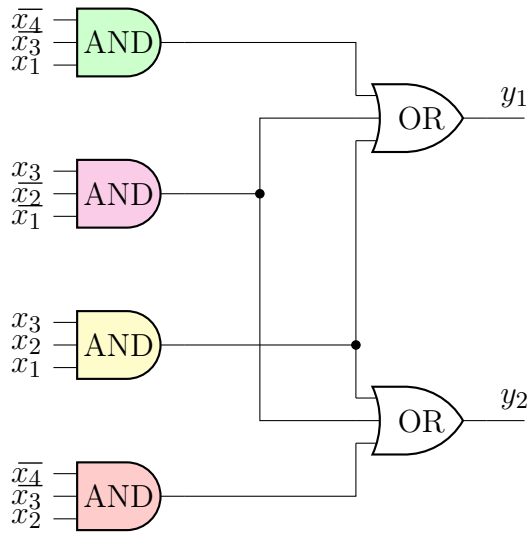
Listing 4.8: Output Esempio 4

```

Solution: [x4'x3'x2 + x3x2' + x3x1]
Solution: [x4x3'x1 + x3x1' + x3x2]

```

Solution: $[x_4'x_3'x_2 + x_3x_2'x_1' + x_3x_2x_1][x_3x_2'x_1' + x_4x_3'x_1 + x_3x_2x_1]$
 The gateInput cost is improved by 10.00 %



Capitolo 5

Esempio di funzionamento

5.1 Input

È stato utilizzato l'input del primo caso noto per mandare in esecuzione il programma per la sintesi ad'una e più uscite, in modo che il flusso di esecuzione sia chiaro al lettore.

Listing 5.1: Codice esempio verbose

```
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 vengono generati tutti i possibili implicantii utilizzando l'algoritmo di QM di cui si possono vedere le iterazioni. Ad ogni iterazione sono formati dei grupppi, ogni gruppo i-esimo contiene tutti gli implicantii che hanno i uni.

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 è presa diretta, 0 che è presa negata, - che non è presente all'interno dell'implicante.

Ad ogni colonna della matrice di copertura corrisponde un implicante, ogni 1 indica se l'implicante copre il mintermine. Ogni 0 indica che non lo copre.

Ogni elemento 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 ad intlinprg 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: [$x_2'x_1 + x_3'x_2$]

5.3 Sintesi di entrambe le uscite

All'inizio vengono trovati gli implicants di entrambe le uscite e 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 vengono trovati gli implicants non ridondanti e generati sia la matrice di scelta che il vettore dei costi.

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

The cost vector is:

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

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).

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

Solution:

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

The cost is:

10.0000

Solution: [x2'x1 + x3'x2] [x2'x1 + x3x2]

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

Capitolo 6

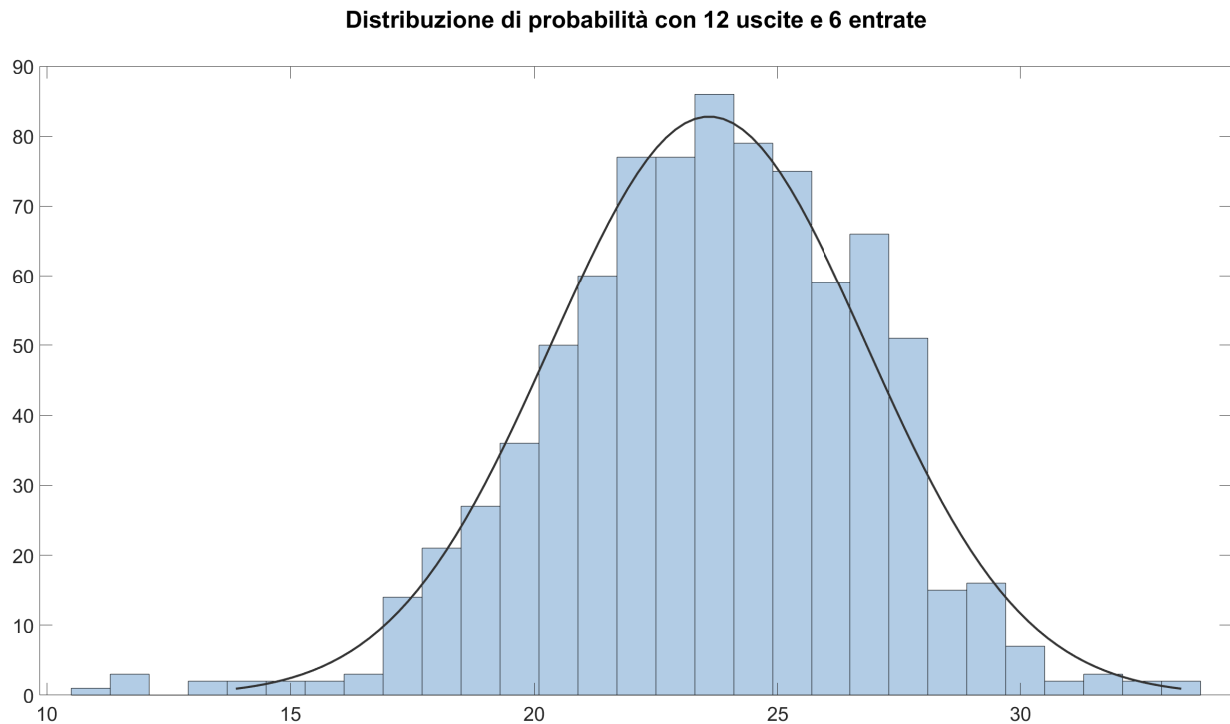
Conclusioni

È stata effettuata una serie di test per capire a quanto ammonta il risparmio percentuale medio, al variare del numero di entrate e numero di uscite.

Una volta eseguiti i test, per ogni combinazione possibile, sono state ricavate \bar{X} media e S^2 varianza campionarie attraverso le formule:

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

È stato osservato che i miglioramenti percentuali possono essere considerati come una variabile aleatoria con distribuzione gaussiana. Possiamo vedere un esempio della distribuzione di probabilità dei miglioramenti percentuali nel caso con 12 uscite e 6 entrate:



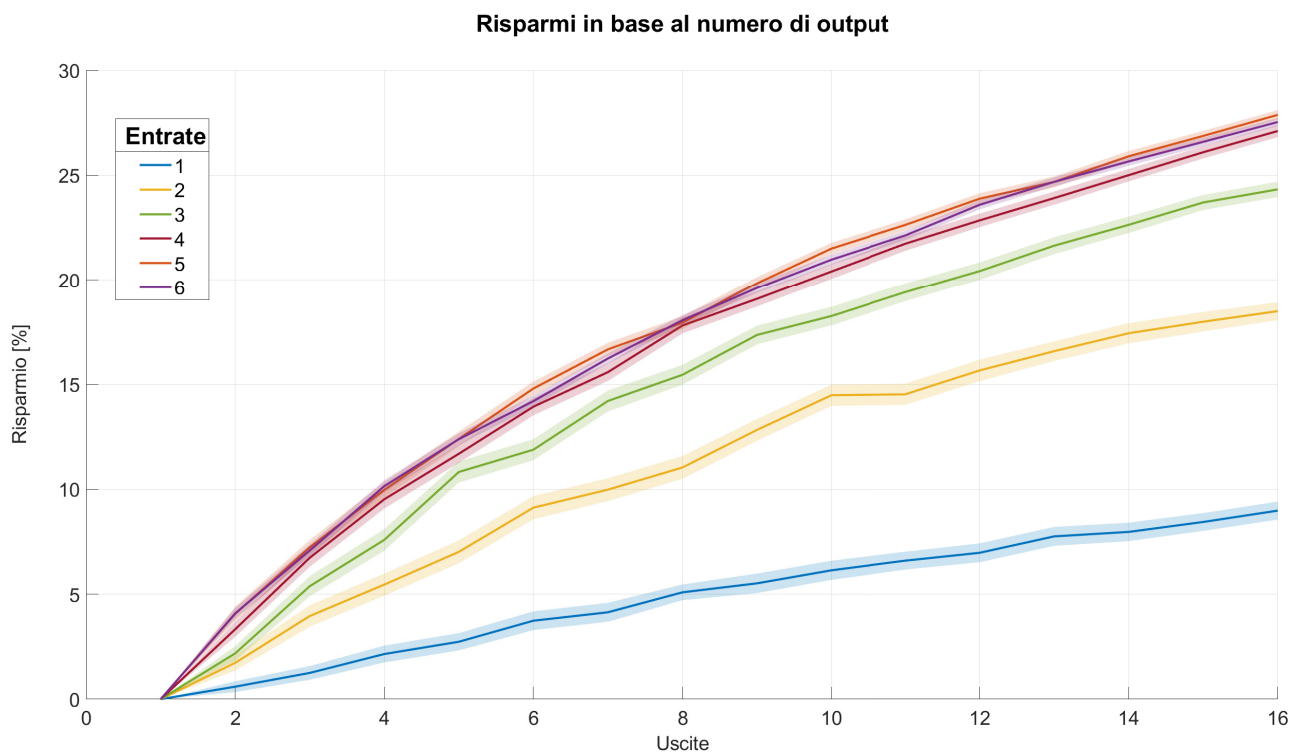
Attraverso questi dati individueremo degli intervalli di confidenza, essi sono ricavabili con la formula:

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

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.



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

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

Capitolo 7

Codice

7.1 Utils

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

Listing 7.1: Funzioni di utilità

```
1 classdef utils
2
3     methods (Static)
4
5         function res = mergeSorted(array_1,array_2)
6
7             arguments
8                 array_1 (1,:) double ...
9                     {mustBeInteger,mustBePositive,utils.mustBeAscending}
10                array_2 (1,:) double ...
11                    {mustBeInteger,mustBePositive,utils.mustBeAscending}
12            end
13
14            % returns
15            %   a sorted array
16
17            i_1 = 1;
18            i_2 = 1;
19            total_length = length(array_1) + length(array_2);
20            res = zeros(1,total_length);
21
22            for i = 1: total_length
23
24                if i_1 > length(array_1)
25                    res(i) = array_2(i_2);
26                    i_2 = i_2 + 1;
27                    continue;
28                end
29
30                if i_2 > length(array_2)
31                    res(i) = array_1(i_1);
32                    i_1 = i_1 + 1;
33                    continue;
34                end
35
36                if array_1(i_1) < array_2(i_2)
37                    res(i) = array_1(i_1);
```

```

38         i_1 = i_1 + 1;
39     else
40         res(i) = array_2(i_2);
41         i_2 = i_2 + 1;
42     end
43 end
44 end
45
46 function res = mergeStrings(a,b,substitute_char)
47
48     arguments
49         a string
50         b string
51         substitute_char char
52     end
53
54     a_chars = a{1};
55     b_chars = b{1};
56     res = a_chars;
57
58     if length(a_chars) ~= length(b_chars)
59         res = "";
60     end
61
62     count = 0;
63
64     for i = 1:length(a_chars)
65
66         a_char = a_chars(i);
67         b_char = b_chars(i);
68
69         % if the chars differ
70         if a_char ~= b_char
71
72             % if one of the chars is the substitute_char
73             if a_char == substitute_char ...
74                 || b_char == substitute_char
75                 res = "";
76                 return;
77             end
78
79             res(i) = substitute_char;
80             count = count + 1;
81         end
82
83         if count > 1
84             res = "";
85             return;
86         end
87     end
88
89     res = string(res);
90
91 end
92
93 function idx = findString(string_array,string)
94
95     idx = -1;
96
97     for i = 1:length(string_array)
98         if strcmp(string_array(i,:),string)
99             idx = i;
100             return;
101         end

```

```

102         end
103
104     end
105
106     function count = countMatches(char_array,pat)
107
108         arguments
109             char_array string
110             pat pattern
111         end
112
113         % returns
114         %     how many times the pattern is found
115
116         count = length(strfind(char_array,pat));
117     end
118
119
120     function res = areAllCellsEmpty(cell_array)
121
122         res = 1;
123         for i = 1:length(cell_array)
124             if ~ isempty(cell_array{i})
125                 res = 0;
126                 return;
127             end
128         end
129     end
130
131
132     function mustBeAscending(a)
133
134         if utils.isInAscendingOrder(a) ; return ; end
135
136         eidType = 'mustBeAscending:notInAscendingOrder';
137         msgType = 'Input must be in ascending order';
138         throwAsCaller(MException(eidType,msgType))
139     end
140
141     function res = isInAscendingOrder(a)
142
143         res = 1;
144         last = 0;
145
146         for i = a
147
148             if i <= last
149                 res = 0;
150                 return;
151             end
152             last = i;
153         end
154
155     end
156
157     function res = minimumBits(numbers)
158         res = ceil(log2(max(numbers)));
159     end
160
161     function [x,v,timedOut] = ...
162         intlinprogWrap(C,A,b,variablesNumber,verbose,timeout)
163
164         % calls intrlinprog
165         %     setting intcon,lb,ub with the specified variablesNumber

```

```

166         % sets on or off the display using the verbose parameter
167
168         intcon = 1:variablesNumber;
169         lb = zeros(variablesNumber,1);
170         ub = ones(variablesNumber,1);
171
172         if timeout <= 0
173             timeout = 7200;
174         end
175
176         if verbose
177             intlinprogOptions = ...
178                 optimoptions('intlinprog',MaxTime = timeout);
179         else
180             intlinprogOptions = optimoptions( ...
181                 'intlinprog',...
182                 Display = 'off',...
183                 MaxTime = timeout ...
184             );
185         end
186
187         [x,v,exitflag] = ...
188             intlinprog(C,intcon,A,b,[],[],lb,ub,intlinprogOptions);
189
190         timedOut = exitflag == 2;
191     end
192 end
193
194 end

```

7.2 getAllImplicants

Funzione per trovare tutti gli implicantsi e generare la matrice di copertura che utilizza l'algoritmo di QM per trovare tutti gli implicantsi possibili.

Listing 7.2: Trova tutti gli implicantsi

```

1 function [implicants,A] = ...
2     getAllImplicants(inputsNumber,minterms,dontCares,options)
3
4     arguments
5         inputsNumber (1,1) double ...
6             {mustBeInteger,mustBePositive}
7         minterms (1,:) double ...
8             {mustBeInteger,mustBePositive,utils.mustBeAscending}
9         dontCares (1,:) double ...
10            {mustBeInteger,mustBePositive,utils.mustBeAscending} = []
11
12         options.Verbose (1,1) double {mustBeNumericOrLogical} = 0
13     end
14
15     % uses
16     % QM algorithm to retrieve a list of all implicants
17     % returns
18     % implicants := list of all implicants generated using QM algorithm
19     % A := the coverage matrix
20
21     notZeros = utils.mergeSorted(minterms,dontCares);
22
23     if inputsNumber < utils.minimumBits(notZeros)
24         error('Variables number: %d is too small',inputsNumber)

```

```

25     end
26
27     if ~ utils.isInAscendingOrder(notZeros)
28         error('Inputs minterms and dontCares have an element in common')
29     end
30
31     % given the indexes, get the value (-1) and then convert it to binary
32     notZerosBinaries = string(dec2bin(notZeros - 1, inputsNumber));
33
34     % first iteration of QM
35     QM_iteration_count = 1;
36     groups = cell(inputsNumber + 1, 1);
37
38     for i = 1:length(notZeros)
39
40         % get group index from number of ones
41         index = utils.countMatches(notZerosBinaries(i,:), "1") + 1;
42
43         % insert the not_zero in the correct group
44         groups{index} = [groups{index}, notZerosBinaries(i,:)];
45
46     end
47
48
49     % at first
50     %   implicants are minterms
51     %   the A matrix is an identity, every implicant covers itself
52     implicants = notZerosBinaries;
53     A = eye(length(implicants));
54
55     % other QM iterations
56
57     while ~ utils.areAllCellsEmpty(groups)
58
59         if options.Verbose
60             fprintf('\nQM iteration %d:\n\n', QM_iteration_count)
61             disp(groups)
62         end
63
64         nextIterationGroups = cell(inputsNumber + 1, 1);
65
66         % compare the i-group with the (i+1)-group
67         % if one implicant merge :
68         %   insert it to the next QM iteration i-group
69         %   insert it into the implicant list and update A
70
71         for i = 1:length(groups) - 1 ; group = groups{i};
72
73             if isempty(group) ; continue ; end
74
75             groupToCompareWith = groups{i + 1};
76
77             if isempty(groupToCompareWith) ; continue ; end
78
79             for j = 1:length(group)
80
81                 implicant = group(j);
82
83                 for k = 1:length(groupToCompareWith)
84
85                     implicantToMatch = groupToCompareWith(k);
86
87                     mergedImplicant = ...
88                         utils.mergeStrings(implicant, implicantToMatch, '-');

```



```

89
90      % if the mergedImplicant is empty continue
91      if mergedImplicant == "" ; continue ; end
92
93      % if the mergedImplicant is redundant continue
94      if any(strcmp(nextIterationGroups{i},mergedImplicant))
95          continue;
96      end
97
98      % if mergedImplicant is correct add it to the list
99      implicants = [implicants ; mergedImplicant];
100
101      % insert a new line in the coverage matrix
102      A(length(implicants),1) = 0;
103
104      % get the two implicants' indexes
105      implicantIndex = ...
106          utils.findString(implicants,implicant{1});
107      toCompareWithIndex = ...
108          utils.findString(implicants,implicantToMatch{1});
109
110      % the mergedImplicant covers the OR bitmap
111      % of the two generator implicants by definitions
112      A(length(implicants),:) = ...
113          A(implicantIndex,:) | A(toCompareWithIndex,:);
114
115      % add the mergedImplicant in the nextIterationGroups
116      nextIterationGroups{i} = ...
117          [nextIterationGroups{i},mergedImplicant];
118
119      end
120  end
121  end
122
123      groups = nextIterationGroups;
124      QM_iteration_count = QM_iteration_count + 1;
125  end
126
127      % remove dontCares constraints from coverage matrix
128      % because they don't need to be covered
129      for i = flip(dontCares)
130          A(:,notZeros == i) = [];
131      end
132
133      % remove implicants that covered only dontCares
134      for i = length(implicants):-1:1
135          if ~ any(A(i,:) == 1)
136              A(i,:) = [];
137              implicants(i) = [];
138          end
139      end
140
141      A = A.';
142
143  end

```

7.3 oneOutputSynthesis

Funzione che esegue la sintesi ad un'uscita di una certe rete combinatoria utilizzando intlinprog.

Listing 7.3: Sintesi ad un'uscita

```

1 function [implicants, v, timedOut] = ...
2     oneOutputSynthesis(minterms, dontCares, options)
3
4     arguments
5
6         minterms (1,:) double ...
7             {mustBeInteger, mustBePositive, mustBeNonempty, utils.
mustBeAscending}
8         dontCares (1,:) double ...
9             {mustBeInteger, mustBePositive, utils.mustBeAscending}
10
11         options.InputsNumber (1,1) double ...
12             {mustBeInteger, mustBePositive} = utils.minimumBits( ...
13                 [minterms, dontCares] ...
14             )
15         options.GatesInputCost (1,1) double ...
16             {mustBeNumericOrLogical} = 1
17         options.Verbose (1,1) double ...
18             {mustBeNumericOrLogical} = 0
19     end
20
21     % returns
22     % the minimal cost synthesis
23
24     if options.Verbose
25         fprintf('\nThe inputs number is:\n\n')
26         disp(options.InputsNumber)
27     end
28
29     [implicants, A] = getAllImplicants( ...
30         options.InputsNumber, ...
31         minterms, ...
32         dontCares, ...
33         Verbose = options.Verbose ...
34     );
35
36     % literal cost for every AND port
37     C = ones(length(implicants), 1);
38     b = ones(length(minterms), 1);
39
40     % diodes cost
41     if options.GatesInputCost
42         for i = 1:length(implicants)
43             C(i) = utils.countMatches(implicants(i, :), "0" | "1") + 1;
44         end
45     end
46
47     if options.Verbose
48         fprintf('All possible implicants are:\n\n') ; disp(implicants)
49         fprintf('The coverage matrix is:\n\n') ; disp(A)
50         fprintf('The cost vector is:\n\n') ; disp(C.')
51     end
52
53     [x, v, timedOut] = ...
54         utils.intlinprogWrap(C, -A, -b, length(implicants), options.Verbose
, 0.1);
55
56     % if it's literal cost add the OR one
57     if ~ options.GatesInputCost
58         v = v + 1;
59     end
60
61     if options.Verbose

```

```

62         fprintf('Solution:\n\n')      ; disp(x.')
63         fprintf('The cost is:\n\n') ; disp(v)
64     end
65
66     % remove all implicants that are not used
67     for i = flip(x)
68         implicants(x == 0) = [];
69     end
70
71 end

```

7.4 multipleOutputSynthesis

Funzione che esegue la sintesi a più uscite di diverse reti combinatorie utilizzando intlinprog.

Listing 7.4: Sintesi a più uscite

```

1 function [implicants,v,timedOut] = ...
2     multipleOutputSynthesis(inputsNumber,outputs,options)
3
4     arguments
5         inputsNumber (1,1) double ...
6             {mustBeInteger,mustBePositive}
7         outputs (1,:) cell
8
9         options.GatesInputCost (1,1) double ...
10            {mustBeNumericOrLogical} = 1
11         options.Verbose (1,1) double ...
12            {mustBeNumericOrLogical} = 0
13         options.Timeout (1,1) double ...
14            {mustBeGreaterThanOrEqual(options.Timeout,0)} = 0
15     end
16
17     % returns
18     %   the minimal cost synthesis
19
20     % set of all implicants
21     implicantsSet = [];
22     outputsImplicantsCount = zeros(length(outputs),1);
23
24     A = [];
25     totalMintermLength = 0;
26
27     for i = 1:length(outputs) ; output = outputs(i);
28
29         minterms = output{1}{1};
30         dont_cares = output{1}{2};
31
32         if ~ utils.isInAscendingOrder(minterms)
33             error('Minterms must be in ascending order')
34         end
35
36         if ~ utils.isInAscendingOrder(dont_cares)
37             error('Dont_cares must be in ascending order')
38         end
39
40         [implicants_k,A_k] = ...
41             getAllImplicants(inputsNumber,minterms,dont_cares);
42
43         outputsImplicantsCount(i) = length(implicants_k);

```

```

44     implicantsSet = [implicantsSet ; implicants_k];
45
46     % every coverage matrix' constraints are independent
47     A = blkdiag(A,A_k);
48
49     totalMintermLength = totalMintermLength + length(minterms);
50
51     if options.Verbose
52         fprintf('\nOutput %d:\n\n',i);
53         fprintf('All possible implicants are:\n\n')
54         disp(implicants_k)
55         fprintf('The coverage matrix is:\n\n')
56         disp(A_k)
57     end
58
59 end
60
61 uniqueImplicants = unique(implicantsSet);
62
63 % add one more constraint for every uniqueImplicants
64 % every uniqueImplicants must be chosen if
65 % a corresponding implicant is chosen
66
67 E = eye(length(uniqueImplicants));
68 A = blkdiag(A,E);
69 Phi = zeros(length(uniqueImplicants), length(implicantsSet));
70
71 % choice matrix
72 for i = 1:length(uniqueImplicants)
73
74     uniqueImplicant = uniqueImplicants(i);
75
76     Phi(i, implicantsSet == uniqueImplicant) = ...
77         -1 / (length(outputs) + 1);
78 end
79
80 A(totalMintermLength + 1:end,1:length(implicantsSet)) = Phi;
81
82 variablesLength = length(implicantsSet) + length(uniqueImplicants);
83
84 % literal cost
85 % only the uniqueImplicants must be considered in the cost
86 C = ones(variablesLength,1);
87 C(1:length(implicantsSet)) = 0;
88
89 % in the choice constraints the constant value is 0
90 % in the cover constraints the constant value is 1
91 b = zeros(totalMintermLength + length(uniqueImplicants),1);
92 b(1:totalMintermLength) = 1;
93
94 % gateInput cost
95 if options.GatesInputCost
96
97     % 1 every time a port is used in an output
98     for i = 1:length(implicantsSet)
99         C(i) = 1;
100     end
101
102     % c_i every time a port is chosen
103     for i = 1:length(uniqueImplicants)
104         C(i + length(implicantsSet)) = ...
105             utils.countMatches(uniqueImplicants(i,:), "0" | "1");
106     end
107 end

```

```

108
109     if options.Verbose
110         fprintf('The not redundant implicants are:\n\n')
111         disp(uniqueImplicants)
112
113         fprintf('The choice matrix is (one implicant each column):\n\n')
114
115         format bank
116         disp(-Phi.')
117         format default
118
119         fprintf('The cost vector is:\n\n')
120         disp(C.')
121     end
122
123     [x,v,timedOut] = utils.intlinprogWrap( ...
124         C, ...
125         -A, ...
126         -b, ...
127         variablesLength, ...
128         options.Verbose, ...
129         options.Timeout ...
130     );
131
132     % if it's literal cost add 1 for, every OR/output
133     if ~ options.GatesInputCost
134         v = v + length(output);
135     end
136
137     if options.Verbose
138         fprintf('Solution:\n\n')
139         disp(x.')
140         fprintf('The cost is:\n\n')
141         disp(v)
142     end
143
144     % build the result implicants
145
146     implicants = cell(length(outputs),1);
147
148     outputIndex = 1;
149     outputImplicantsIndex = 1;
150
151     % cycle through every implicant
152     for i = 1:length(implicantsSet)
153
154         % if the ith implicant is has been chosen add it
155         if x(i)
156
157             % if the outputImplicantsIndex overflow got to the next output
158             if outputImplicantsIndex > outputsImplicantsCount(outputIndex)
159                 outputImplicantsIndex = 1;
160                 outputIndex = outputIndex + 1;
161             end
162
163             implicants{outputIndex} = ...
164                 [implicants{outputIndex} ; implicantsSet(i)];
165         end
166
167         % increment the current outputImplicantsIndex
168         outputImplicantsIndex = outputImplicantsIndex + 1;
169     end
170 end
171

```

7.5 displayImplicants

Funzione che visualizza gli implicanti in un 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 7.5: Visualizzazione degli implicanti

```

1 function displayImplicants(outputs_implicants,options)
2     arguments
3         outputs_implicants (1,:) cell
4
5         options.LatexSyntax (1,1) double ...
6             {mustBeNumericOrLogical} = 0
7     end
8
9     if options.LatexSyntax
10         error('unimplemented')
11     end
12
13     fprintf('Solution: ')
14
15     for i = 1:length(outputs_implicants)
16         implicants = outputs_implicants{i};
17
18         fprintf('[ ');
19
20         for j = 1:length(implicants)
21
22             implicant = implicants{j};
23
24             for k = 1:length(implicant)
25
26                 letter = implicant(k);
27
28                 if strcmp(letter,'-') ; continue ; end
29
30                 fprintf('x%d',length(implicant) - k + 1)
31
32                 if strcmp(letter,'1') ; continue ; end
33
34                 fprintf(',')
35             end
36
37             if j == length(implicants) ; continue ; end
38             fprintf(' + ')
39
40         end
41
42         fprintf('] ');
43

```

```

44     end
45     fprintf('\n')
46 end

```

7.6 synthesisCheck

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

Listing 7.6: Controllo di una sintesi

```

1 function check = syntesisCheck(implicants, minterms, dontCares)
2     arguments
3         implicants (1,:) string ...
4             {mustBeNonempty}
5
6         minterms (1,:) double ...
7             {mustBeInteger, mustBePositive, mustBeNonempty, utils.
mustBeAscending} = []
8
9         dontCares (1,:) double ...
10             {mustBeInteger, mustBePositive, utils.mustBeAscending} = []
11     end
12
13     inputsNumber = strlen(implicants(1));
14     check = true;
15     truthTable = zeros(2 ^ inputsNumber);
16
17     for i=1:length(implicants)
18         implicant = implicants(i);
19
20         for minterm=1:length(truthTable)
21
22             if isCovered(dec2bin(minterm - 1, inputsNumber), implicant{1})
23                 truthTable(minterm) = 1;
24             end
25         end
26     end
27 end
28
29 for i=1:length(truthTable)
30
31     % if this element don't care skip it
32     if any(dontCares == i) ; continue ; end
33
34     % if it's a one
35     if truthTable(i) == 1
36
37         % check wheter it's wrong
38         if ~ any(minterms == i)
39             check = false;
40             return
41         end
42
43         continue
44     end
45
46     % if it's a zero
47     % check wheter it's wrong
48
49     if any(minterms == i)

```

```

50         check = false;
51         return
52     end
53
54 end
55
56 end
57
58
59 function res = isCovered(minterm, implicant)
60
61     res = true;
62
63     for i=1:length(minterm)
64
65         if implicant(i) == '-'; continue ; end
66         if implicant(i) ~= minterm(i)
67             res = false;
68             return
69         end
70     end
71
72 end

```

7.7 statistics

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

Listing 7.7: Script per generare le statistiche

```

1  addpath(genpath("./../"))
2
3  rng('shuffle')
4
5  MAX_TEST_NUMBER = 420;
6  TIMEOUT = 23 * 60;
7
8  % if this variables are not set, set them to default values
9  if exist('outputsNumber','var') ~= 1 ; outputsNumber = 8 ; end
10 if exist('inputsNumber','var') ~= 1 ; inputsNumber = 4 ; end
11 if exist('lastTest','var') ~= 1 ; lastTest = 1 ; end
12
13 biggest_value = 2^inputsNumber;
14
15 for test = lastTest:MAX_TEST_NUMBER
16
17     outputs = cell(outputsNumber, 1);
18     one_out_cost = 0;
19
20     % one output synthesis
21
22     for i = 1:outputsNumber
23
24         p = randperm(biggest_value);
25
26         minterms_count = round(biggest_value * rand(1,1));
27
28         while minterms_count < 1
29             minterms_count = round(biggest_value * rand(1,1));
30         end

```



```

31
32     dontcares_count = round(biggest_value * rand(1,1));
33
34     while (minters_count + dontcares_count) > biggest_value
35         dontcares_count = round(biggest_value * rand(1,1));
36     end
37
38     p_1 = sort(p(1:minters_count));
39
40     if minters_count < biggest_value
41         p_2 = sort( ...
42             p(minters_count + 1: minters_count + dontcares_count) ...
43         );
44     else
45         p_2 = [];
46     end
47
48     out = {p_1, p_2};
49
50     [~, v, timedOut] = oneOutputSynthesis( ...
51         out{1}, ...
52         out{2}, ...
53         InputsNumber = inputsNumber, ...
54         Timeout = TIMEOUT ...
55     );
56
57     if timedOut ; break ; end
58
59     one_out_cost = one_out_cost + v;
60
61     outputs{i} = out;
62 end
63
64 if timedOut ; fprintf('%d timed_out\n', test) ; continue ; end
65
66 tStart = tic;
67
68 % multiple output synthesis
69
70 [~, multiple_out_cost, timedOut] = multipleOutputSynthesis( ...
71     inputsNumber, ...
72     outputs, ...
73     Timeout = TIMEOUT ...
74 );
75
76 elapsedTime = toc(tStart);
77
78 if timedOut ; fprintf('%d timed_out\n', test) ; continue ; end
79
80 fprintf( ...
81     '%d %f %f %.20f\n', ...
82     test, ...
83     one_out_cost, ...
84     multiple_out_cost, ...
85     elapsedTime...
86 )
87
88 end

```

7.8 plotStatistics

Script che visualizza i miglioramenti percentuali in un grafico.

Listing 7.8: Script per visualizzare le statistiche

```

1
2 delimiterIn = ' ';
3 maxInputsNumber = 6;
4 maxOutputsNumber = 16;
5
6 avgs = zeros(maxInputsNumber,maxOutputsNumber);
7 std_devs = zeros(maxInputsNumber,maxOutputsNumber);
8 errors = zeros(maxInputsNumber,maxOutputsNumber);
9 worst_error = 0;
10 best_save = 0;
11
12 % ALPHA = 0.95;
13 QUANTILE = 1.96;
14
15 for o = 1:maxOutputsNumber
16     for i = 1:maxInputsNumber
17
18         filename = sprintf('folder/%d-%d.log', o, i);
19         A = importdata(filename,delimiterIn);
20
21         savings = (A(:,2) - A(:,3)) ./ A(:,2) * 100;
22
23
24         avgs(i,o) = mean(savings);
25
26         if mean(savings) > best_save
27             best_save = mean(savings);
28         end
29
30         std_dev = sqrt(var(savings));
31         error = std_dev / sqrt(length(savings)) * QUANTILE;
32
33         if isnan(std_dev) ; continue ; end
34
35         std_devs(i,o) = std_dev;
36         errors(i,o) = error;
37
38         if error > worst_error ; worst_error = error; end
39
40     end
41 end
42
43 input_legends = cell(1, maxInputsNumber);
44
45 for i = 1:2:maxInputsNumber * 2
46     input_legends{i} = sprintf('%d', round(i/2));
47     input_legends{i + 1} = '';
48 end
49
50 hold on
51
52 gaps = std_devs;
53
54 for i = 1:maxInputsNumber
55
56     p = plot(avgs(i,:));
57
58     p(1).LineWidth = 2;
59
60     c = get(p,'Color');
61
62     x = 1:length(avgs(i,:));
63     curve1 = avgs(i,:) + gaps(i,:);

```

```

64     curve2 = avgs(i,:) - gaps(i,:);
65     x2 = [x, fliplr(x)];
66     inBetween = [curve1, fliplr(curve2)];
67     h = fill(x2, inBetween, c, 'LineStyle','none');
68     h.FaceAlpha = 0.2;
69
70 end
71
72 grid on
73
74 ax = gca;
75 ax.FontSize = 18;
76
77 lgd = legend(input_legends{:}, 'Location','northwest', 'FontSize', 18);
78 title(lgd, 'Entrate ', 'FontSize', 22)
79
80
81 xlabel({'Uscite', ''}, 'FontSize', 18);
82 ylabel({'Risparmio [%]', ''}, 'FontSize', 18);
83
84 title({'Risparmi in base al numero di output', ''}, 'FontSize', 22)
85
86 disp(worst_error);
87 disp(best_save);

```

7.9 distribution

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

Listing 7.9: Script per controllare la distribuzione di probabilità dei test

```

1
2 delimiterIn = ' ';
3
4 % if this variables are not set, set them to default values
5 if exist('outputsNumber','var') ~= 1 ; outputsNumber = 8 ; end
6 if exist('inputsNumber','var') ~= 1 ; inputsNumber = 4 ; end
7
8 filename = sprintf('folder/%d-%d.log', outputsNumber, inputsNumber);
9 A = importdata(filename,delimiterIn);
10
11 savings = sort((A(:,2) - A(:,3)) ./ A(:,2) * 100);
12
13 h = histfit(savings);
14
15 h(1).FaceColor = [0.4 0.6 0.8];
16 h(1).FaceAlpha = 0.5;
17
18 h(2).Color = [.2 .2 .2];
19
20 ax = gca;
21 ax.FontSize = 18;
22
23 title({'Distribuzione di probabilita con 12 uscite e 6 entrate', ''}, ...
24       'FontSize', 22)

```

Bibliografia