Tesi

Andrea Locatelli

Indice

Capitolo 1

Analisi

Il protocollo di Yao prevede, durante i suoi scambi, che le due parti concordino anche sulla trasmissione di un circuito a cui entrambi gli attori debbano inserire dei valori di input. Uno degli scopi della nostra tesi consiste nel testare l'efficacia dell'utilizzo di un multi valore rispetto all'utilizzo "classico" di un circuito booleano.

Per poter iniziare questa fase di analisi siamo partiti da dei circuiti binari per poi convertirli nella loro alternativa multivalore implementando un metodo di conversione efficace che non cambi il funzionamento di tali circuiti

1.1 Studio del circuito binario

La tecnica che abbiamo deciso di utilizzare è molto semplice ma efficace e si basa sul raggruppamento di n bit di ogni riga di ogni tabella di verità binaria che possano rappresentare il massimo numero possibile del dominio multivalore scelto.

$$\begin{array}{ccc} \overbrace{10} & \overbrace{11} & \overbrace{00} \\ \downarrow & \downarrow & \downarrow \\ 2 & 3 & 0 \end{array} \quad mod_3$$

Con questo approccio non andiamo a snaturare quella che è la logica della tabella originale e otteniamo una conversione efficace. I vincoli dettati da questo stanno nel fatto che il numero di ingressi e uscite del circuito devono essere pari.

Per la scelta del dominio multivalore si sono voluti sperimentare 3 approcci:

- Per tutti i circuiti utilizzare un dominio multivalore standard di valore 3

- Per ogni circuito viene calcolato M.C.D tra input e output e:
  - Se il valore di M.C.D consente di avere almeno 2 input viene tenuto quello
  - Se non consente al minimo 2 input viene dimezzato
- L'ultimo approccio è un ibrido tra i 2, ovvero: Viene calcolato M.C.D tra input e output

Per potere effettuare queste operazioni si è dovuto cercare un formato standard di descrizione dei circuiti che rappresentasse per ogni riga della tabella delle verità del circuito tutti gli input e tutti gli output.

Tra i vari formati a disposizione la scelta è ricaduta sul formato `PLA` già ampiamente utilizzato per la rappresentazione di circuiti.

1.1.1 Analisi della struttura PLA

Un file PLA ha la seguente struttura

```
.i 4
.o 2
.ilb x1 x2 y1 y2
.ob f1 f2
0--0 00
0001 01
0-11 --
1-11 01
0101 10
10-- 01
11-- 00
.end
```

Come possiamo vedere questo circuito presenta:

- 4 input
- 2 output

Notiamo che nella tabella di verità non sono presenti solamente valori booleani ma anche un simbolo `-` esso si prende il nome di **don't care** e rappresenta:

- Negli output implica che quella specifica combinazione non mi interessa
- Negli input che può essere qualsiasi valore del dominio.

1.1.2 La gestione dei don't care durante la conversione

Nelle successive fasi di conversione dobbiamo gestire i don't care in maniera funzionale al dominio multi valore che utilizzeremo, per spiegare meglio come tratteremo questa caratteristica dei PLA utilizzerò un esempio utilizzando un circuito semplice e come valore di dominio di conversione il valore 3. Il circuito avrà la seguente struttura.

```
.i 4
.o 2
.ilb x1 x2 y1 y2
.ob f1 f2
0--0 00
0001 01
0-11 --
1-11 01
0101 10
10-- 01
11-- 00
.end
```

Per passare al nuovo dominio analizziamo quanti bit sono necessari per poter rappresentare il massimo numero possibile, nel nostro caso il 3, e notiamo che abbiamo bisogno di 2 bit poiché la rappresentazione binaria del numero 3 è 11. Andiamo quindi a raggruppare in blocchi di 2 bit gli ingressi e le uscite del circuito andando a convertire i valori binari dati dal raggruppamento in valore del dominio scelto.

Procedendo con la conversione ci accorgiamo che ci sono alcuni casi in cui dobbiamo andare a sviluppare i dont't care perché se non lo facessimo perderemmo dei valori significativi per il nostro sviluppo. Andiamo quindi a distinguere 2 casi:

- Quando il numero di '-' è uguale al numero di bit del raggruppamento multivalore
  - *es.*. 10 – 01: in questo caso viene messo semplicemente il simbolo - non causa nessuna perdita di valori nel circuito. Quindi

$$\begin{array}{ccc} 10 & -- & 01 \\ & \downarrow & \\ 2 & - & 1 \end{array}$$

- Quando nel gruppo di bit raggruppati il numero di - è minore del numero di bit richiesti

- *es.* 1-11 01: in questo caso non posso subito convertire nel nostro nuovo dominio ma devo prima sviluppare tutte le possibili combinazioni e poi procedere alla conversione. Quindi
  - Sviluppo il dont'care

    10 11 01
    11 11 01

  - Procedo a convertire nel dominio scelto

    2 3 1
    3 3 1

Così facendo riusciamo a sviluppare tutto il circuito binario e ottenere una conversione esatta. Il circuito convertito sarà quindi.

```
00 0
02 0
10 0
12 0
01 1
03 -
13 -
23 1
11 2
2- 1
3- 0
```

Con questa struttura posso proseguire con l'analisi dei circuiti proseguendo con l'attività di sintesi logica.

1.2 Implementazione della conversione

La conversione dei circuiti in analisi viene implementata utilizzando il linguaggio di programmazione Python nella versione 3.8 senza l'ausilio di nessun package esterno, vengono utilizzate solamente librerie comprese nel linguaggio, questo fa si che il sistema sia "ready to use" una volta installato il linguaggio di programmazione se non già presente all'interno del SO.

1.2.1 Analisi del circuito

```
def read_pla(path_file):
    inp = None
    out = None
    inp_array = []
    out_array = []
    truth_table = []
    with open(path_file, 'r') as input_file:
        for line in input_file.readlines():
            if '.i' in line and line[2] == ' ':
                inp = line.split(' ')[1]
            elif '.o' in line and line[2] == ' ':
                out = line.split(' ')[1]
            elif '.ilb' in line:
                inp_array = line.strip().split(' ')[1:]
            elif '.ob' in line:
                out_array = line.strip().split(' ')[1:]
            elif '.end' in line:
                continue
            else:
                line = {
                    'inp':  line.strip().split(' ')[0],
                    'out':  line.strip().split(' ')[1]
                }
                truth_table.append(line)
    return inp, out, inp_array, out_array, truth_table
```

La funzione prende in ingresso il percorso di un circuito e memorizza all'interno di un dizionario informazioni tipo:

- Tabella delle verità
- Numero di input
- Numero di output

Queste informazioni serviranno successivamente per convertire il circuito e calcolare costi.

1.2.2 Espansione dei don't care

Una volta ottenuto tutte le informazioni disponibili dal circuito dato bisogna andare ad identificare all'interno delle tabelle di verità quali sono i *don't care* a cui bisogna espandere i valori e quelli che si possono ignorare. Questa parte è stata la parte più impegnativa di questa parte di funzionalità del programma.

```
n_dont_care = ''
    for i in range(dv):
        n_dont_care += '-'
```

Questo semplice ciclo va a replicare il numero di - consecutivi che rispecchiano i gruppi di don't care da ignorare in base al dominio di conversione dato al circuito.

```
len_truth_table = len(truth_table)
i = 0
while i < len_truth_table:
    if '-' in truth_table[i]['inp']:
        truth_table[i]['inp'] = ''.join(truth_table[i['inp'])
        truth_table[i]['inp'] = [truth_table[i]['inp'[a:a+dv]
                                 for a in range(0, le(truth_table[i]['inp'], dv)]
        for a in range(len(truth_table[i]['inp'])):
            if truth_table[i]['inp'][a] == n_dont_care:
                truth_table[i]['inp'][a] = 'k'*le(n_dont_care)
        truth_table[i]['inp'] = ''.join(truth_table[i['inp'])
        new_lines = resolve_dont_care(truth_table[i],'inp')
        truth_table = truth_table[:i] + new_lines +truth_table[i+1:]
        len_truth_table = len(truth_table)
    i += 1
```

Per andare a differenziare quali siano i *don't care* da espandere e quali no a quest ultimi viene sostituito il simbolo - con un valore k in modo da poterli gestire meglio nelle funzioni successive. Alla fine delle operazioni verranno ripristinati con il simbolo corretto.

1.2.2.1 Resolve don't care

```
def resolve_dont_care(line, in_out):
    # Conto qunati - e creo 2^n nuove linee
    n_dc = pow(2, line[in_out].count('-'))

    input_entry = line[in_out]
    new_array = []

    for h in range(line[in_out].count('-')):
        val_array = create_0_1_array(n_dc, pow(2, h))
        if h == 0:
            for i in range(n_dc):
                it = 0
                new_line = []
                for j in range(len(input_entry)-1, -1, -1):
                    if input_entry[j] == '-' and it == 0:
                        new_line.append(val_array[(len(val_array)-1) - i])
                        it += 1
                    else:
                        new_line.append(line[in_out][j])
                new_array.append(new_line[::-1])
        else:
            it = 0
            for c, l in enumerate(new_array):
                for j in range(len(l)-1, -1, -1):
                    if l[j] == '-' and it == 0:
                        l[j] = val_array[(len(val_array)-1) - c]
                        it += 1
                it = 0

    if in_out == 'inp':
        return [{'inp':  l, 'out':  line['out']} for l in new_array]
    else:
        return [{'inp':  line['inp'], 'out':  l} for l in new_array]
```

Questa funzione prende in ingresso la linea da espandere e per ogni sua iterazione *(it)* va a sostituire ogni simbolo - con un valore di verità. La funzione restituirà la l'espansione della linea.

Come possiamo notare dal `return` della funzione questa operazione di espansione è possibile farla sia sugli input che su gli output.

1.2.2.2 create_0_1_array

```
def create_0_1_array(le, pad):
    count_1 = pad
    count_0 = pad
    return [str(1) if i % (count_1 + count_0) < count_1
        else str(0) for i in range(le)]
```

Questa funzione restituisce una tabella di verità data una line con all'interno un numero di *don't care* che necessitano di espansione.

Questo tabella creata verrà inserita al posto della linea con i *don't care* del circuito in così da avere la tabella corretta per la conversione.

1.3 La conversione in multi valore

Le funzioni viste nelle sezioni precedenti restituiscono come valore di output una matrice corrispondente alla tabella della verità sviluppata. Questa tabella verrà utilizzata dalla funzione di conversione seguendo lo schema spiegato all'inizio del capitolo.

```
def create_mv_truth_table(truth_array, dv):
    conv_truth = []
    for line in truth_array:
        line['inp'] = ''.join(line['inp'])
        line['inp'] = [line['inp'][i * dv:(i + 1) * dv]
            for i in range((len(line['inp']) + dv - 1) // dv)]
        if '-' not in line['out']:
            line['out'] = ''.join(line['out'])
            line['out'] = [line['out'][i * dv:(i + 1) * dv]
                for i in range((len(line['out']) + dv - 1) // dv)]
            l_supp_inp = []
            l_supp_out = []
            for val in line['inp']:
                if '-' not in val:
                    l_supp_inp.append(int(val, 2))
                else:
                    l_supp_inp.append('-')
            for val in line['out']:
                l_supp_out.append(int(val, 2))
            conv_truth.append({
                'inp':  l_supp_inp,
                'out':  l_supp_out
```

```
            })
    return conv_truth
```

La funzione prende in ingresso la matrice espansa creata precedentemente e il numero di bit da utilizzare per rappresentare il massimo numero del nuovo dominio multivalore.

La funzione inizialmente 'spezza' la stringa dei valori di input in gruppi di tanti elementi quanti i bit per rappresentare il massimo numero, successivamente viene controllato se il gruppo è composto da:

- **dont't care**: si procede sostituendo con un singolo simbolo '-'
- **numeri binari**: si procede con la conversione tramite il metodo int(), ad esso servono 2 parametri:
  - un numero o una stringa di numeri da convertire
  - la base a cui si vuole fare la conversione

Una volta scandito tutta la tabella di verità abbiamo il circuito convertito, l'utlimo passo ora consiste nell'andare a creare un file compatibile per la sintesi dei circuiti.

1.4 La creazione del file blfmv

Per i circuiti multi valore non viene usato il formato PLA in quanto viene utilizzato solamente per i circuiti binari. Utilizziamo un altro formato standard appositamente creato per la logica multi valore, il formato *blfmv*.

Questo formato deriva dal formato *blif* utilizzato per la logica binaria.

```
import string
mv_input = [i for i in list(string.ascii_lowercase)[
    :len(mv_table[0]['inp'])]]
mv_output = ['o{}'.format(i) for i in range(len(mv_table[0]['out']))]
with open('{}/blfmv/{}.mv'.format(working_dir, nomefile), 'w') as blif:
    blif.write('.model {}\n'.format(working_dir, nomefile))
    blif.write('.inputs {}\n'.format(
        ' '.join(map(str, mv_input))))
    blif.write('.outputs {}\n'.format(
        ' '.join(map(str, mv_output))))
    blif.write('.mv {} {}\n'.format(
        ', '.join(map(str, mv_input)), mv))
    blif.write('.mv {} {}\n'.format(
        ', '.join(map(str, mv_output)), mv))
```

```
    for count, out in enumerate(mv_output):
        blif.write('.table {} {}\n'.format(' '.join(map(str,mv_input)), out))
        for line in mv_table:
            blif.write('{} {}\n'.format(
                ' '.join(map(str, line['inp'])), line['out'][count]))
    blif.write('.end\n')
```

Questa funzione crea un file *.mv* relativo al circuito creato. Utilizziamo questa tipo di file e sintassi perché nelle fasi successive utilizzeremo per valutare e sintetizzare i circuiti dei software che accettano questa sintassi.

1.5 La sintesi

Una volta ottenuto sia il circuito booleano che quello multivalore procediamo con la sintesi dei circuiti, la sintesi dei circuiti si pone l'obiettivo di ridurre e ottimizzare la struttura dei circuiti stessi andando a diminuire il numero di ingressi e di tabelle della verità modo da avere dei costi totali minori.

I programmi utilizzati per la sintesi logica utilizzati sono entrambi sviluppati dall'università di Berkley e sono disponibili con licenza open source.

Per la sintesi durante l'analisi sperimentale sono state utilizzate 2 alternative:

- MVSIS
- ABC

1.5.1 MVSIS

Primo programma utilizzato per la sintesi, contiene diversi metodi da poter utilizzare in che utilizzano tecniche differenti con scopi differenti. La particolarità di questo tool consiste nel fatto che il programma accetti come input sia circuiti binari che multivalore e tratta quest ultimi come multivalore, non leggendoli e convertendoli in circuiti binari per poi sintetizzare. Il lato negativo dell'utilizzo di questo porgramma è che l'ultima versione di questo software risale al 2005 e quindi non è più mantenuto.

1.5.2 ABC

ABC, come la sua alternativa descritta sopra, mette a disposizione delle tecniche di sintesi con io vantaggio di avere degli script pronti che uniscono più metodi in modo da avere la certezza di non commettere errori o di utilizzare una concatenazione di metodi inefficace.

ABC è un'evoluzione di MVSIS e del più vecchio SIS, viene tuttora mantenuto ma, a differenza di MVSIS, se gli viene dato in ingresso un valore di input viene successivamente convertito in binario e poi si possono utilizzare i metodi di sintesi.

Una caratteristica molto utile di questi programmi sta nel fatto che essi accettino come parametri di ingresso degli script contenenti tutte le istruzioni da eseguire, successivamente mostrerò come tutto il processo di analisi viene automatizzato sfruttando questa caratteristica.

1.5.3 La sintesi tramite i software

La letteratura mette già a disposizione delle sequenze di metodi di sintesi e pulizia dei circuiti efficace ed efficiente, questi comandi si possono chiamare semplicemente dando in input ai due programmi un file contenente un alias per questa sequenza di istruzioni.

```
source abc_alias.abc
read_blif_mv ./prova/blfmv/amd.mv
strash
compress2
cl
resyn2
cl
write_blif_mv ./prova/blfmv/synth/amd.mv
```

Il programma, in questo caso *abc*, non fa altro che prendere prendere in input uno dei circuiti creati precedentemente e applicare i metodi di sintesi per poi creare un altro file contenente il circuito sintetizzato in modo da poter fare dei successivi confronti.

1.6 Il calcolo dei costi del circuito

Il costo del circuito viene calcolato sulla base 2 fattori fondamentali:

- Quanti input devono inviare entrambe le parti
- In base al dominio quanti valori devono inviare per ciascun valore di input

Per calcolare questo tipo di informazioni si devono effettuare delle operazioni di lettura sui file precedentemente creati andando a guardare:

- ingressi di ogni tabella di verità del circuito, andando a controllare quali dei valori di input devono essere inseriti da una delle parti e quali sono ingressi di valori di output delle altre tabelle.
- Dominio dei valori di input che devono inserire le parti.

1.6.1 blfmv vs pla

Per fare il confronto sopra citato si è preferito utilizzare file che abbiano la stessa sintassi di rappresentazione del circuito, sia nel caso multi valore che in quello booleano.

Il formato *pla* non dispone di un'alternativa per i file multivalore ma non è ancora supportata dai tool che abbiamo in utilizzo, si è deciso di utilizzare quindi il formato *blif* per rappresentare i circuiti binari. *Blif* è l'altranativa binaria a *blfmv*, la sintassi è la stessa con la differenza che non viene specificato il dominio dei valori di input in quanto sempre booleano.

La differenza con *pla* invece sta nel fatto che gli output non possono essere più di 1 per tabella nella rappresentazione, avrò quindi, per ogni nodo del circuito, tante tabelle quanti gli output di quel nodo. Questa tipologia di rappresentazione è utilizzata anche nei file *blfmv* quindi avrò uno comparazione 1:1 tra binario e multivalore.

Per effettuare questa conversione vengono in aiuto i tool *ABC* e *MVSIS*, entrambi contengono metodi di conversione automatica da *pla* a *blif*. Questa operazione viene effettuata tramite uno script contenente le istruzioni da eseguire e dato come parametro di ingresso al programma.

```
read_pla ./prova/pla/alu2.pla
write_blif ./prova/blif/alu2.blif
```

Ora abbiamo a disposizione tutti i file per poter fare il confronto dei costi

1.6.2 Implementazione

```
with open('{}/{}'.format(working_dir, circuito)) as circ:
    input = None
    output = None
    mv = int(0)
    table_array = []
    for line in circ.readlines():
        if '.inputs' in line.strip():
            input = line.strip().split(' ')[1:]
        if '.outputs' in line.strip():
            output = line.strip().split(' ')[1:]
        if '.mv' in line.strip():
            if mv < int(line.strip().split(' ')[-1]):
                mv = int(line.strip().split(' ')[-1])
        if '.table' in line.strip():
            l = line.strip().split(' ')
```

```
            table_array.append(
                {
                    'input':    l[1:len(l)-1],
                    'output':   l[-1]
                }
            )
        if '.names' in line.strip():
            l = line.strip().split(' ')
            table_array.append(
                {
                    'input':    l[1:len(l)-1],
                    'output':   l[-1]
                }
            )
    # i blif non hanno .mv, quindi gli do il valore di dominio
    if mv == 0:
        mv = 2
    return {
        'dominio':   mv,
        'input':    input,
        'output':   output,
        'tabelle':  table_array
    }
```

In questa funzione, dato un cricuito sia binario che mutlivalore, esso prende tutte le informazioni utili per poter effettuare il calcolo

```
costo = 0
for t in circuito['tabelle']:
    intersection = len(set(circuito['input']).intersectio(t['input']))
    costo = costo + pow(circuito['dominio'], intersection)
return costo
```

Capitolo 2

Risultati sperimentali