Logic Synthesis using Programmable Logic Gates

Taiwan Semiconductor Research Institute (TSRI), NARL

1. Introduction

場域可程式化閘陣列 (Field Programmable Gate Array，FPGA)是透過預先建立的可程式化邏輯電路與路由(routing)電路，讓使用者可以藉由設定功能與連線的機制，設定邏輯電路模組的功能以及相互間的連線，以組合成所需要的電路。因為可以重新設定功能與可以直接在設計場域使用，如實驗室或研發單位，故能即時調整電路的效能與功能，以加速商品上市時間。因此，FPGA 被廣泛地使用於商品與雛型品(prototyping)驗證、仿真(emulation)驗證等應用。FPGA 是可程式化的邏輯電路，組合電路可以使用 FPGA 內之 RAM 區塊來做查表(LUT)而完成，也就是說將輸入當成記憶體區的位址，而預計得到的邏輯值則為記憶體的內容。例如位址 00 填 1，位址 01，10，及 11 都填 0，則此記憶體區塊即成為 NOR 閘之邏輯功能，位址的輸入則為 NOR 閘的輸入，記憶體區塊的輸出則為 NOR 閘的輸出。而另一種作法就是使用較複雜的 universal gate 來實現所需的邏輯，利用保險絲燒斷或是沒燒斷的機制來設定可程式邏輯的輸入訊號以及輸出連接的方式。本題目將針對複雜的 universal gates 開發邏輯合成的程式，將給定的邏輯函數合成以 universal gates 來實現的電路，並同時依據給定的 cost function 進行最佳化。

2. Problem Statement

本題目將給定三種複雜的 universal gates，針對只有這三個 universal gates 所組成的元件庫(library)，開發邏輯合成程式，也就是將給定的邏輯函數轉換成只利用這三種 universal gates 來實現，而不能使用 assign 以及其他的邏輯閘。這三種 universal gates 的輸入也不能使用任何邏輯運算。為了簡化問題，三種 universal gates 的 area cost 都為 1，timing cost 也都為 1。參賽者需要讀入待合成的邏輯函數，以及三個 universal gates 的功能，並將此邏輯函數合成出只使用此三種 universal gates 的電路。

這三種 universal gates 都只有一個輸出，功能分別如下:

gate1:

4-to-1 MUX, Output=$\overline{S_0 S_1 I_3 + S_0 \bar{S_1} I_2 + \bar{S_0} S_1 I_1 + \bar{S_0} \bar{S_1} I_0}$

gate2:

Actel's ACT 1 logic module, Output=$\overline{(I_0 \bar{S_0} + I_1 S_0)\overline{(S_2 + S_3)} + (I_2 \bar{S_1} + I_3 S_1)(S_2 + S_3)}$

gate3:

H-bridge function, Output=$\overline{I_1(I_2 + I_4 I_5) + I_3(I_4 + I_2 I_5)}$

題目固定使用這三種 universal gates，參賽者需要將給定的組合電路以此三種 universal gates 來實現，可以單獨選用一種或兩種，不需要全部的邏輯閘都使用，但也可以三種邏輯閘都使用。合成後的電路，area 為所使用的邏輯閘總數，而 timing 則為從輸入端到輸出端的最長路徑上的邏輯閘數目，也就是 longest path 上的邏輯閘數目。合成時的最佳化目標為降低 cost = area × timing 的值。

以圖 1 的一個全加法器(full adder)為例，輸入為 A，B，C，輸出為 Sum，Carry，如果選擇所給定的 gate1 (4-to-1 MUX)來實現，則需要三個邏輯閘，所以 area =3，timing = 2，cost = area × timing = 6。

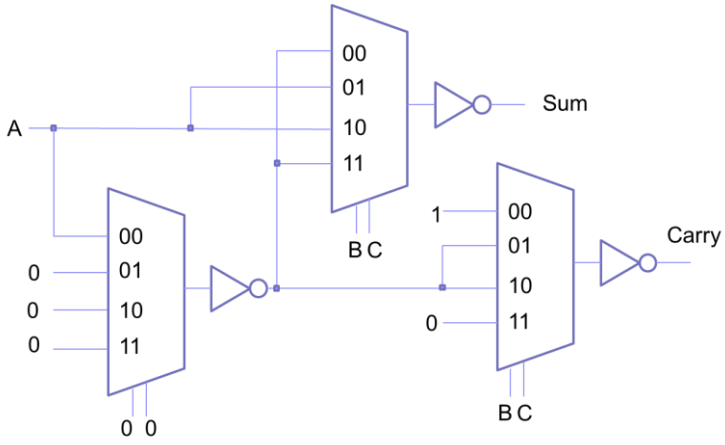

圖 1: 以 4-to-1 MUX 來實現之全加法器。

3. Inputs

輸入格式將採用 LGSynth91 的 Verilog 檔格式，以 assign 方式描述輸入電路，所有的輸入檔案都是組合邏輯電路，沒有序向電路。另外需要輸入三種 universal gates 所組成的元件庫，兩個檔案格式與範例如下:

(1) Verilog 檔格式之輸入檔，範例如下:

```
module fa(a, b, c, sum, carry);
input
   a,
   b,
   c;
output
   sum,
   carry;
wire
   \[1] ;
assign
   \[1]    = (~b & a) | (b & ~a),
   sum = (~\[1] & c) | (\[1] & ~c),
   carry= (a & b) | (b & c) | (a & c);
endmodule
```

(2) Verilog 檔格式之元件庫，範例如下:

```
module gate1(s0, s1, i0, i1, i2, i3, o);
input
   s0,
    s1,
   i0,
   i1,
   i2,
   i3;
output
   o;
assign
   o = ~(( s0 & s1 & i3) | (s0 & ~s1 & i2) | (~s0 & s1 & i1 ) | (~s0 & ~s1 & i0));
endmodule

module gate2(s0, s1, s2, s3, i0, i1, i2, i3, o);
input
   s0,
   s1,
   s2,
   s3,
   i0,
   i1,
  i2,
   i3;
output
   o;
assign
   o = ~((( i0 & ~s0) | (i1 & s0) )& ~(s2 | s3)) | (((i2 & ~s1) | (i3 & s1)) & (s2 | s3)));
endmodule

module gate3(i1, i2, i3, i4, i5, o);
input
   i1,
   i2,
   i3,
   i4,
   i5;
output
   o;
assign
   o = ~((i1 & (i2 | (i4 & i5))) | (i3 & (i4 | (i2 & i5))));
endmodule
```

4. Requirements

需要提供 readme.txt 以說明環境設定與執行方法等資訊，而編譯好的程式需放在提供的目錄最上層，於提示符號下執行 "**time your_program_name –i in.v –l lib.v –o out.v**"，其中 in.v，lib.v，out.v 不是固定的名稱，是要能被置換成其他的檔名，也就是說，in.v 可以自由換成 in1.v，in2.v，…等，out.v 也可以換成 out1.v，out2.v，…等。其

中 in.v 是需要被合成的輸入檔，lib.v 則是三個 universal gates 的 Verilog 檔，而輸出 out.v 則是合成後以 gate1，gate2，或是 gate3 組合成的電路(netlist)檔。

5. Outputs

輸出檔案為 Verilog 檔格式， 輸入輸出和輸入檔設定相同，但是是由 gate1，gate2，或是 gate3 所組成的結構化模型(structural modeling)方法，module 連結方式可以利用 connection by name 方式，需要連接宣告的名稱。另外，也可以利用 connection by order 方式，利用 module 宣告的順序連接，兩種方式都可以使用。底下範例是使用 connection by order 方式，將圖 1 的合成結果展現出來。注意，合成後的檔案不能有 assign 指令，也不能有其他邏輯閘存在，全部都只能利用 gate1，gate2，或是 gate3 組合而成。

```
module fa(a, b, c, sum, carry);
input
   a,
   b,
   c;
output
   sum,
   carry;
wire
    ab ;
   gate1 g0(1'b0, 1'b0, a, 1'b0, 1'b0, 1'b0, ab);
   gate1 g1(b, c, ab, a, a, ab, sum);
   gate1 g2(b, c, 1'b1, ab, ab, 1'b0, carry);
endmodule
```

6. Grading Criteria

- 正確性: 合成後的電路和原先合成前的電路功能需要相同，我們將會驗證功能是否一致(functional equivalence)。
- 程式需要在競賽單位指定的機器內，不能執行超過兩小時，超過兩個小時將強迫終止，如果終止後仍有結果，將執行驗證作業並列入正常的成績計算。
- 所有 test case 裡(包含給定與隱藏的 test cases)，功能正確數目最多的獲勝；當功能正確的數目相同時，則比所有功能正確的 test case(s)之加總的 cost，最低者獲勝；當 cost 仍然相同時，則比加總的執行時間，越少者獲勝。

7. Reference

[1] Logic Synthesis and Optimization Benchmarks User Guide Version 3.0.

[2] Actel ACT 1 Series FPGAs data sheet.

[3] Shun-Wen Cheng, “Configurable CMOS H-tree Logic Module,” Proc. of 2009 IEEE Int’l Conf. on Field-Programmable Technology (FPT’09), Sydney, Australia, Dec. 9-11, 2009, pp. 431-434.