# CHAPTER 8

# SOLVING THE UNATE AND BINATE COVERING PROBLEMS

* 1. **Introduction**

Unate covering problem finds applications in test generation, minimization of Boolean functions and many others. Binate covering problems are used in many aspects of logic circuit minimization and in state machine minimization problem.

## Solving the Unate Covering Problem.

Given is a function from Figure 8.1. All its prime implicants are marked as ovals (loops). Using the minterm compatibility graph G all primes are found as maximum cliques. They can be also found as maximum independent sets of graph  (G complement). Based on KMap and primes we can create the covering table from.



Table 8.1 Finding graphically all prime implicants for minimal covering of a SOP circuit



Table 8.2. Covering table for function from Error! Reference source not found.

From the table, denoting rows A, B, C, D, E we compile the Petrick function in a standard way:

This function can be simplified using the Boolean law as follows.

1 = A • B • D • C

Therefore,



Figure 8.3. Oracle for the function 

Another search method for (another) unate covering table from **Error! Reference source not found.** is illustrated in Figure 8.4. Figure 8.4 shows the branching tree for the unate covering problem from **Error! Reference source not found.**. All leafs are solutions, as showed in Figure 8.4. Both these methods can be used to build search oracles, as well as hybrid parallel searches.

 

Figure 8.4. Solving the Petrick Function from the unate covering Table 8.3. This method can be combined with oracle methods using the mixed parallel SAT approaches



Table 8.5 Another example of an unate covering problem represented by a table.



Figure 8.6. *Oracle for the function f=(AB+AD+CBF+CBE). The circuit is not optimized.*

//-----------------------------------------------------

//File name: unate1.v

//Description: Verilog code for Unate covering problem1

//-----------------------------------------------------

module unate1(f,a,b,c,d,clk,reset);

input clk,reset;

input a,b,c,d;

output f;

reg f;

always@(posedge clk)

begin

 if(reset)

 begin

 f<=0;

 end

 else

 f <= (((~a)&(~c)&(~d))|((~a)&b&c)|(a&c&d)|(a&b&(~c))|(b&d));

 end

endmodule

//--------------------------------------------------

//File Name:unate1\_tb.v

//Description: Testbench for Unate covering problem1

//--------------------------------------------------

module unate1\_tb(f,clk,reset);

input clk,reset;

output f;

reg [3:0]A;//temperory variable

//Instatiation

unate1 unate1\_inst(.f(f),.a(A[3]),.b(A[2]),.c(A[1]),.d(A[0]),.clk(clk),.reset(reset));

always@(posedge clk)

begin

 if(reset)

 begin

 A <= 4'b0;

 end

 else

 begin

 if(A == 4'b1111)

 A <= 0;

 else

 A <= A+4'b0001;

 end

end//always

endmodule

e Name:unate1\_tb.v

//Description: Testbench for Unate covering problem1

//--------------------------------------------------

module unate1\_tb(f,clk,reset);

input clk,reset;

output f;

reg [3:0]A;//temperory variable

//Instatiation

unate1 unate1\_inst(.f(f),.a(A[3]),.b(A[2]),.c(A[1]),.d(A[0]),.clk(clk),.reset(reset));

always@(posedge clk)

begin

 if(reset)

 begin

 A <= 4'b0;

 end

 else

 begin

 if(A == 4'b1111)

 A <= 0;

 els

//-----------------------------------------------------

//File name: unate2.v

//Description: Verilog code for Unate covering problem2

//-----------------------------------------------------

module unate2(f1,a,b,c,d,e,f,clk,reset);

input clk,reset;

input a,b,c,d,e,f;

output f1;

reg f1;

always@(posedge clk)

begin

 if(reset)

 begin

 f1 <=0;

 end

 else

 f1 <= ((a|c)&(b|d|f)&(a|b)&(b|d)&(a|e|f)&(a|b|d|f));

 end

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//--------------------------------------------------

//File Name:unate2\_tb.v

//Description: Testbench for Unate covering problem2

//--------------------------------------------------

module unate2\_tb(f1,clk,reset);

input clk,reset;

output f1;

reg [5:0]A;//temporary variable

//Instantiation

unate2 unate2\_inst(.f1(f1),.a(A[5]),.b(A[4]),.c(A[3]),.d(A[2]),.e(A[1]),.f(A[0]),.clk(clk),.reset(reset));

always@(posedge clk)

begin

 if(reset)

 begin

 A <= 6'b0;

 end

 else

 begin

 if(A == 6'b1)

 A <= 0;

 else

 A <= A+1;

 end

end//always

endmodule

\_tb.v

//Description: Testbench for Unate covering problem2

//--------------------------------------------------

module unate2\_tb(f1,clk,reset);

input clk,reset;

output f1;

reg [5:0]A;//temporary variable

//Instantiation

unate2 unate2\_inst(.f1(f1),.a(A[5]),.b(A[4]),.c(A[3]),.d(A[2]),.e(A[1]),.f(A[0]),.clk(clk),.reset(reset));

always@(posedge clk)

begin

 if(reset)

 begin

 A <= 6'b0;

 end

 else

 begin

 if(A == 6'b1)

## Solving the Binate Covering Problem

Binate covering problem was also known as “Covering with closures”. This is because in binate covering problem, variables in the function appears both in positive form and negative form. Unate covering always has a solution whereas binate covering may or may not have a solution. Binate covering problem is used in artificial intelligence, state machine minimization, technology mapping and Boolean relations.

One example of binate covering problem is in Figure 7‑4 Implication graph for the binate covering problem. The covering-closure table for the implication graph from Figure 7‑4 is shown in **Error! Reference source not found.**.



Figure 7‑4 Implication graph for the binate covering problem

**Table 7‑2 Covering-Closure table for the implication graph**

The equation for the binate covering problem is evaluated as



The function can be simplified using the logic transformation rule  as





Figure 7‑5 Oracle for the binate covering problem with function .

Given is a finite state machine in figure 2.3.2.1(a). A triangular table in Figure 2.3.2.1(b) was generated covering all possible cases to minimize the number of states in the state machine. The ‘X’ symbol in the table indicates there is no possibility for grouping the corresponding states, ‘V’ indicates that the states can be combined without any problem and variables in the table indicates that states can be grouped only if the states mentioned in the block can be combined without any problem. Based on the triangular table, a compatibility graph for the state machine can be generated shown in **Figure 7‑6 (a) Finite state machine (b) Triangular table indicating the compatibility for combining different states in the FSM.** **Figure 7‑6**



**Figure 7‑6 (a) Finite state machine (b) Triangular table indicating the compatibility for combining different states in the FSM.**

To minimize the state machine, a covering-closure table shown in **Error! Reference source not found.** is created by considering the maximum cliques and all its subsets as rows and all the states in the machine and the implications in the compatibility graph as columns.



**Figure 7‑7 (a) Compatibility graph for the FSM (b) Incompatibility graph for the FSM.**



Table 7‑3 Covering-Closure table for the FSM

From the table, binate covering problem can be specified using the equation



The function can be simplified using the Boolean laws *(A→B = Ā+B)* and *A.(A+E) = A.A+A.E =A(1+E) =A .*

The minimized state machine is shown in Figure 7‑8



Figure 7‑8 Minimized FSM using binate covering problem.

# SPECIALIZED PROCESSOR FOR “MAN, WOLF, GOAT AND CABBAGE” CONSTRAINT SATISFACTION SEQUENTIAL PROBLEM

## Introduction and goals

A man has a wolf, a goat, and a cabbage. He must cross a river with the two animals and the cabbage. There is a small rowing-boat, in which he can take only one thing with him at a time. If, however, the wolf and the goat are left alone, the wolf will eat the goat. If the goat and the cabbage are left alone, the goat will eat the cabbage.

How can the man get across the river with the two animals and the cabbage?

Rules for the transport

1) Man has to present in every move.

2) Not more than 2 objects are can move in any move.

There are two solutions to this. These two solutions can be seen in flow graph as follows.

**Solution 1:**

Man takes the goat across the river, from bank 1 to bank 2,

Leaving the wolf and the cabbage at 1.

Man returns empty to pick up the wolf from bank 1 and takes it to 2, where he leaves it and picks up the goat and bring it back to bank 1.

Man takes Cabbage to bank 2 and return empty handed back to bank 1 leaving wolf and cabbage to bank 2.

Now Man takes goat from bank 1 to bank 2 and all now man get across the river with 2 animals and cabbage.

**Solution 2:**

Man takes the goat across the river, from bank 1 to bank 2,

Leaving the wolf and the cabbage at 1 and returns empty to pick up the cabbage from bank 1 and takes it to 2, where he leaves it and picks up the goat and bring it back to

bank 1.

Man takes wolf from bank 1 to bank 2 and and return empty handed back to bank 1 leaving wolf and cabbage to bank 2.

Now Man takes goat from bank 1 to bank 2 and all now man get across the river with 2 animals and cabbage.



## Checking Safe Condition at Both Bank

We can find the safe condition from unsafe condition. Because state expression for unsafe condition is simple and we can simply negate this expression in order to get safe state expression.

Unsafe condition at bank 1 is when Man is not there but wolf and goat are there 

Or Man is not there and goat and cabbage are there 

So Unsafe condition at bank 1 is 

Similarly Un-safe condition at bank 2 

Unsafe condition at bank 1 and 2 is 

 Safe state equation will be negation of above expression.

Safe condition at both banks ……………………………………. Eq. 1

## Checking for invalid move

Cheating or invalid move will be the move with

A move in which man does not move at all

A move in which man but more than 2 objects also moves with him.

Let’s take a first case,

When man doenot moves ex-or of two variable representing man in two successive stage is 0

If is variable representing man in stage n and  is variable representing man in stage n+1

then

 

Here subscript n denote states

So in order to check whether man is moving between successive move or not , the above equation should be negated and should be checked for the value 1

That means check for

 

When Mn denote man variable in nth state and Mn+1 represent man variable in n+1 state.

Final equation for checking invalid move of man does not moving will be Oring operation of all moves between successive stages.

Now for the 2nd condition of invalid move in which man moves but more than 2 object moves with him.

When object moves between successive states, the ex-or operation between the literal representing that objet is 1.

This means when goat is moving 

When Cabbage is moving 

When wolf is moving then 

When man is moving then 



So when man moves but wolf and cabbage also moves with him then 

Similarly when wolf and goat moves with man then .

and when goat and cabbage moves with man then 

so final equation for bad or invalid move is 

in summary, For any move to be invalid,

  ………………………………………………………………Eq 2

The Final oracle will be implementation of Eq 1 and 2



Figure 8‑1 Oracles for checking invalid state



Figure 8‑2 Oracle for checking invalid move

Final oracle will be AND operation between Output Y1 and Output Y2.



Figure 8‑3 Final Oracle

**VHDL Code for State Checking**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity check\_state is

port(Man,Wolf, Goat, Cabbage :in std\_logic;

 Valid\_state : out std\_logic);

end check\_state;

architecture Behavioural of check\_state is

 signal Man\_bar,Wolf\_bar,Cabbage\_bar,Goat\_bar,temp1,temp2,temp3,temp4,invalid\_state : std\_logic;

begin

 --- Implementing equation for state checking

 Man\_bar <= not Man;

 Wolf\_bar <= not Wolf;

 Cabbage\_bar <= not Cabbage;

 Goat\_bar <= not Goat;

 temp1 <= Man\_bar and Goat;

 temp2 <= Wolf or Cabbage;

 temp3 <= Man and Goat\_bar;

 temp4 <= Wolf\_bar or Goat\_bar;

 invalid\_state <= (temp1 and temp2) or (temp3 and temp4);

 Valid\_state <=not invalid\_state;

end behavioural;

## VHDL Code for Move Checking

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity check\_move is

port(Man\_ps,Wolf\_ps,Cabbage\_ps,Goat\_ps :in std\_logic;

 Man\_ns,Wolf\_ns,Cabbage\_ns,Goat\_ns :in std\_logic;

 Valid\_move : out std\_logic);

end check\_move;

architecture Behavioural of check\_move is

signal Z1,Z2,Z3,Z4,invalid\_move : std\_logic;

begin

 -- Checkin valididy of move 1

 Z1 <= Man\_ps xor Man\_ns; -----MAN MOVES

 Z3 <= Wolf\_ps xor Wolf\_ns;

 Z3 <=Cabbage\_ps xor Cabbage\_ns;

 Z4 <= Goat\_ps xor Goat\_ns;

 invalid\_move <= (not Z1) or (Z1 and Z2 and Z3) or ( Z1 and Z2 and Z4) or (Z1 and Z3 and Z4);--- tells wheather move from present state to next state is valid or not

 Valid\_move <= invalid\_move;

end Behavioural;

## VHDL code for Final Oracle

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity oracle is

port(M1\_state1,M1\_state2,M1\_state3,M1\_state4,M1\_state5,M1\_state6,M1\_state7,M1\_state8,M1\_state9,M1\_state10,M1\_state11,M1\_state12 :in std\_logic;

 M2\_state1,M2\_state2,M2\_state3,M2\_state4,M2\_state5,M2\_state6,M2\_state7,M2\_state8,M2\_state9,M2\_state10,M2\_state11,M2\_state12 :in std\_logic;

 M3\_state1,M3\_state2,M3\_state3,M3\_state4,M3\_state5,M3\_state6,M3\_state7,M3\_state8,M3\_state9,M3\_state10,M3\_state11,M3\_state12 :in std\_logic;

 C1\_state1,C1\_state2,C1\_state3,C1\_state4,C1\_state5,C1\_state6,C1\_state7,C1\_state8,C1\_state9,C1\_state10,C1\_state11,C1\_state12 :in std\_logic;

 C2\_state1,C2\_state2,C2\_state3,C2\_state4,C2\_state5,C2\_state6,C2\_state7,C2\_state8,C2\_state9,C2\_state10,C2\_state11,C2\_state12 :in std\_logic;

 C3\_state1,C3\_state2,C3\_state3,C3\_state4,C3\_state5,C3\_state6,C3\_state7,C3\_state8,C3\_state9,C3\_state10,C3\_state11,C3\_state12 :in std\_logic;

 Valid\_state : out std\_logic);

end oracle;

architecture structural of oracle is

component check\_state

port(M1,M2,M3,C1,C2,C3:in std\_logic;

 Valid\_state : out std\_logic);

end component;

component check\_move

port(M1\_ps,M2\_ps,M3\_ps,C1\_ps,C2\_ps,C3\_ps :in std\_logic;

 M1\_ns,M2\_ns,M3\_ns,C1\_ns,C2\_ns,C3\_ns :in std\_logic;

 valid\_move : out std\_logic);

end component;

begin

 State\_1 : check\_state port map(M1=>M1\_state1,M2=>M2\_state1,M3=>M3\_state1,C1=>C1\_state1,C2=>C2\_state1,C3=>C3\_state1);

 State\_2 : check\_state port map(M1=>M1\_state2,M2=>M2\_state2,M3=>M3\_state2,C1=>C1\_state2,C2=>C2\_state2,C3=>C3\_state2);

 State\_3 : check\_state port map(M1=>M1\_state3,M2=>M2\_state3,M3=>M3\_state3,C1=>C1\_state3,C2=>C2\_state3,C3=>C3\_state3);

 State\_4 : check\_state port map(M1=>M1\_state4,M2=>M2\_state4,M3=>M3\_state4,C1=>C1\_state4,C2=>C2\_state4,C3=>C3\_state4);

 State\_5 : check\_state port map(M1=>M1\_state5,M2=>M2\_state5,M3=>M3\_state5,C1=>C1\_state5,C2=>C2\_state5,C3=>C3\_state5);

 State\_6 : check\_state port map(M1=>M1\_state6,M2=>M2\_state6,M3=>M3\_state6,C1=>C1\_state6,C2=>C2\_state6,C3=>C3\_state6);

 State\_7 : check\_state port map(M1=>M1\_state7,M2=>M2\_state7,M3=>M3\_state7,C1=>C1\_state7,C2=>C2\_state7,C3=>C3\_state7);

 State\_8 : check\_state port map(M1=>M1\_state8,M2=>M2\_state8,M3=>M3\_state8,C1=>C1\_state8,C2=>C2\_state8,C3=>C3\_state8);

 State\_9 : check\_state port map(M1=>M1\_state9,M2=>M2\_state9,M3=>M3\_state9,C1=>C1\_state9,C2=>C2\_state9,C3=>C3\_state9);

 State\_10 : check\_state port map(M1=>M1\_state10,M2=>M2\_state10,M3=>M3\_state10,C1=>C1\_state10,C2=>C2\_state10,C3=>C3\_state10);

 State\_11: check\_state port map(M1=>M1\_state11,M2=>M2\_state11,M3=>M3\_state11,C1=>C1\_state11,C2=>C2\_state11,C3=>C3\_state11);

 State\_12 : check\_state port map(M1=>M1\_state12,M2=>M2\_state12,M3=>M3\_state12,C1=>C1\_state12,C2=>C2\_state12,C3=>C3\_state12);

 Move\_1 : check\_move port map (M1\_ps=>M1\_state1,M2\_ps=>M2\_state1,M3\_ps=>M3\_state1,C1\_ps=>C1\_state1,C2\_ps=>C2\_state1,C3\_ps=>C3\_state1,

 M1\_ns=>M1\_state2,M2\_ns=>M2\_state2,M3\_ns=>M3\_state2,C1\_ns=>C1\_state2,C2\_ns=>C2\_state2,C3\_ns=>C3\_state2);

 Move\_2 : check\_move port map (M1\_ps=>M1\_state2,M2\_ps=>M2\_state2,M3\_ps=>M3\_state2,C1\_ps=>C1\_state2,C2\_ps=>C2\_state2,C3\_ps=>C3\_state2,

 M1\_ns=>M1\_state3,M2\_ns=>M2\_state3,M3\_ns=>M3\_state3,C1\_ns=>C1\_state3,C2\_ns=>C2\_state3,C3\_ns=>C3\_state3);

 Move\_3 : check\_move port map (M1\_ps=>M1\_state3,M2\_ps=>M2\_state3,M3\_ps=>M3\_state3,C1\_ps=>C1\_state3,C2\_ps=>C2\_state3,C3\_ps=>C3\_state3,

 M1\_ns=>M1\_state4,M2\_ns=>M2\_state4,M3\_ns=>M3\_state4,C1\_ns=>C1\_state4,C2\_ns=>C2\_state4,C3\_ns=>C3\_state4);

 Move\_4 : check\_move port map (M1\_ps=>M1\_state4,M2\_ps=>M2\_state4,M3\_ps=>M3\_state4,C1\_ps=>C1\_state4,C2\_ps=>C2\_state4,C3\_ps=>C3\_state4,

 M1\_ns=>M1\_state5,M2\_ns=>M2\_state5,M3\_ns=>M3\_state5,C1\_ns=>C1\_state5,C2\_ns=>C2\_state5,C3\_ns=>C3\_state5);

 Move\_5 : check\_move port map (M1\_ps=>M1\_state5,M2\_ps=>M2\_state5,M3\_ps=>M3\_state5,C1\_ps=>C1\_state5,C2\_ps=>C2\_state5,C3\_ps=>C3\_state5,

 M1\_ns=>M1\_state6,M2\_ns=>M2\_state6,M3\_ns=>M3\_state6,C1\_ns=>C1\_state6,C2\_ns=>C2\_state6,C3\_ns=>C3\_state6);

 Move\_6 : check\_move port map (M1\_ps=>M1\_state6,M2\_ps=>M2\_state6,M3\_ps=>M3\_state6,C1\_ps=>C1\_state6,C2\_ps=>C2\_state6,C3\_ps=>C3\_state6,

 M1\_ns=>M1\_state7,M2\_ns=>M2\_state7,M3\_ns=>M3\_state7,C1\_ns=>C1\_state7,C2\_ns=>C2\_state7,C3\_ns=>C3\_state7);

 Move\_7 : check\_move port map (M1\_ps=>M1\_state7,M2\_ps=>M2\_state7,M3\_ps=>M3\_state7,C1\_ps=>C1\_state7,C2\_ps=>C2\_state7,C3\_ps=>C3\_state7,

 M1\_ns=>M1\_state8,M2\_ns=>M2\_state8,M3\_ns=>M3\_state8,C1\_ns=>C1\_state8,C2\_ns=>C2\_state8,C3\_ns=>C3\_state8);

 Move\_8 : check\_move port map (M1\_ps=>M1\_state8,M2\_ps=>M2\_state8,M3\_ps=>M3\_state8,C1\_ps=>C1\_state8,C2\_ps=>C2\_state8,C3\_ps=>C3\_state8,

 M1\_ns=>M1\_state9,M2\_ns=>M2\_state9,M3\_ns=>M3\_state9,C1\_ns=>C1\_state9,C2\_ns=>C2\_state9,C3\_ns=>C3\_state9);

 Move\_9 : check\_move port map (M1\_ps=>M1\_state9,M2\_ps=>M2\_state9,M3\_ps=>M3\_state9,C1\_ps=>C1\_state9,C2\_ps=>C2\_state9,C3\_ps=>C3\_state9,

 M1\_ns=>M1\_state10,M2\_ns=>M2\_state10,M3\_ns=>M3\_state10,C1\_ns=>C1\_state10,C2\_ns=>C2\_state10,C3\_ns=>C3\_state10);

 Move\_10 : check\_move port map (M1\_ps=>M1\_state10,M2\_ps=>M2\_state10,M3\_ps=>M3\_state10,C1\_ps=>C1\_state10,C2\_ps=>C2\_state10,C3\_ps=>C3\_state10,

 M1\_ns=>M1\_state11,M2\_ns=>M2\_state11,M3\_ns=>M3\_state11,C1\_ns=>C1\_state11,C2\_ns=>C2\_state11,C3\_ns=>C3\_state11);

 Move\_11 : check\_move port map (M1\_ps=>M1\_state11,M2\_ps=>M2\_state11,M3\_ps=>M3\_state11,C1\_ps=>C1\_state11,C2\_ps=>C2\_state11,C3\_ps=>C3\_state11,

 M1\_ns=>M1\_state12,M2\_ns=>M2\_state12,M3\_ns=>M3\_state12,C1\_ns=>C1\_state12,C2\_ns=>C2\_state12,C3\_ns=>C3\_state12);

 end structural;

## Test bench for testing Oracle and finding solution

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_textio.all;

use std.textio.all;

entity oracle\_test is

end oracle\_test;

architecture beha of oracle\_test is -- Sequential Modelling Style

component final\_oracle

port(M1,M2,M3,M4,M5,M6,M7,M8 :in std\_logic;

 W1,W2,W3,W4,W5,W6,W7,W8 :in std\_logic;

 G1,G2,G3,G4,G5,G6,G7,G8 :in std\_logic;

 C1,C2,C3,C4,C5,C6,C7,C8 :in std\_logic;

 Valid\_solution:out std\_logic);

end component;

signal M1,M2,M3,M4,M5,M6,M7,M8 :std\_logic;

signal W1,W2,W3,W4,W5,W6,W7,W8 :std\_logic;

signal G1,G2,G3,G4,G5,G6,G7,G8 :std\_logic;

signal C1,C2,C3,C4,C5,C6,C7,C8 :std\_logic;

signal temp\_1,temp\_2,temp\_3,temp\_4 : std\_logic\_vector(7 downto 0):=(others=>'0');

signal Valid\_solution :std\_logic;

file outfile : text open WRITE\_MODE is "//khensu/Home04/gunavant/Desktop/output.txt";

procedure write\_result is

variable buff : line;

begin

 write(buff,string'("Solution found to be "));

 writeline (outfile, buff);

 write(buff,string'("M1 = "));

 write(buff,M1);

 write(buff,string'("M2 = "));

 write(buff,M2);

 write(buff,string'("M3 = "));

 write(buff,M3);

 write(buff,string'("M4 = "));

 write(buff,M4);

 write(buff,string'("M5 = "));

 write(buff,M5);

 write(buff,string'("M6 = "));

 write(buff,M6);

 write(buff,string'("M7 = "));

 write(buff,M7);

 write(buff,string'("M8 = "));

 write(buff,M8);

 writeline (outfile, buff);

 write(buff,string'("W1 = "));

 write(buff,W1);

 write(buff,string'("W2 = "));

 write(buff,W2);

 write(buff,string'("W3 = "));

 write(buff,W3);

 write(buff,string'("W4 = "));

 write(buff,W4);

 write(buff,string'("W5 = "));

 write(buff,W5);

 write(buff,string'("W6 = "));

 write(buff,W6);

 write(buff,string'("W7 = "));

 write(buff,W7);

 write(buff,string'("W8 = "));

 write(buff,W8);

 writeline (outfile, buff);

 write(buff,string'("C1 = "));

 write(buff,C1);

 write(buff,string'("C2 = "));

 write(buff,C2);

 write(buff,string'("C3 = "));

 write(buff,C3);

 write(buff,string'("C4 = "));

 write(buff,C4);

 write(buff,string'("C5 = "));

 write(buff,C5);

 write(buff,string'("C6 = "));

 write(buff,C6);

 write(buff,string'("C7 = "));

 write(buff,C7);

 write(buff,string'("C8 = "));

 write(buff,C8);

 writeline (outfile, buff);

 write(buff,string'("G1 = "));

 write(buff,G1);

 write(buff,string'("G2 = "));

 write(buff,G2);

 write(buff,string'("G3 = "));

 write(buff,G3);

 write(buff,string'("G4 = "));

 write(buff,G4);

 write(buff,string'("G5 = "));

 write(buff,G5);

 write(buff,string'("G6 = "));

 write(buff,G6);

 write(buff,string'("G7 = "));

 write(buff,G7);

 write(buff,string'("G8 = "));

 write(buff,G8);

 writeline (outfile, buff);

end procedure write\_result;

begin

u : final\_oracle port map(M1=>M1,M2=>M2,M3=>M3,M4=>M4,M5=>M5,M6=>M6,M7=>M7,M8=>M8,

 G1=>G1,G2=>G2,G3=>G3,G4=>G4,G5=>G5,G6=>G6,G7=>G7,G8=>G8,

 C1=>C1,C2=>C2,C3=>C3,C4=>C4,C5=>C5,C6=>C6,C7=>C7,C8=>C8,

 W1=>W1,W2=>W2,W3=>W3,W4=>W4,W5=>W5,W6=>W6,W7=>W7,W8=>W8,

 Valid\_solution =>Valid\_solution);

P1 : process

file outfile : text open WRITE\_MODE is "//khensu/Home04/gunavant/Desktop/output.txt";

variable buff : line;

begin

L1: for k in 0 to 255 loop

 temp\_1 <= temp\_1+ "00000001";

 L2: for m in 0 to 255 loop

 temp\_2 <= temp\_2+ "00000001";

 L3: for n in 0 to 255 loop

 temp\_3 <= temp\_3+ "00000001";

 L4: for p in 0 to 255 loop

 temp\_4 <= temp\_4+ "00000001";

 W1<=temp\_2(0); W2<=temp\_2(1); W3<=temp\_2(2); W4<=temp\_2(3); W5<=temp\_2(4); W6<=temp\_2(5); W7<=temp\_2(6); W8<=temp\_2(7);

 M1<=temp\_1(0); M2<=temp\_1(1); M3<=temp\_1(2); M4<=temp\_1(3); M5<=temp\_1(4); M6<=temp\_1(5); M7<=temp\_1(6); M8<=temp\_1(7);

 G1<=temp\_3(0); G2<=temp\_3(1); G3<=temp\_3(2); G4<=temp\_3(3); G5<=temp\_3(4); G6<=temp\_3(5); G7<=temp\_3(6); G8<=temp\_3(7);

 C1<=temp\_4(0); C2<=temp\_4(1); C3<=temp\_4(2); C4<=temp\_4(3); C5<=temp\_4(4); C6<=temp\_4(5); C7<=temp\_4(6); C8<=temp\_4(7);

 wait for 5 ns;

 if(Valid\_solution= '1') then

 write\_result;

 end if; ----- This is procedural call to write results in file

 end loop L4;

end loop L3;

end loop L2;

end loop L1;

 wait;

end process;

end beha;

## The Output file contents

Solution found to be:

M1 = 0 M2 = 1 M3 = 0 M4 = 1 M5 = 0 M6 = 1 M7 =0 M8 = 1

W1 = 0 W2 = 0 W3 = 0 W4 = 1 W5 = 1 W6 = 1 W7 =1 W8 = 1

C1 = 0 C2 = 1 C3 = 1 C4 = 1 C5 = 0 C6 = 0 C7 =0 C8 = 1

G1 = 0 G2 = 0 G3 = 0 G4 = 0 G5 = 0 G6 = 1 G7 =1 G8 = 1

## Emulator testbench

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_textio.all;

use std.textio.all;

entity oracle\_test is

 port (clock,enable,reset:in std\_logic);

end oracle\_test;

architecture beha of oracle\_test is -- Sequential Modelling Style

component final\_oracle

port(M1,M2,M3,M4,M5,M6,M7,M8 :in std\_logic;

 W1,W2,W3,W4,W5,W6,W7,W8 :in std\_logic;

 G1,G2,G3,G4,G5,G6,G7,G8 :in std\_logic;

 C1,C2,C3,C4,C5,C6,C7,C8 :in std\_logic;

 Valid\_solution:out std\_logic);

end component;

signal M1,M2,M3,M4,M5,M6,M7,M8 :std\_logic;

signal W1,W2,W3,W4,W5,W6,W7,W8 :std\_logic;

signal G1,G2,G3,G4,G5,G6,G7,G8 :std\_logic;

signal C1,C2,C3,C4,C5,C6,C7,C8 :std\_logic;

signal counter : std\_logic\_vector(31 downto 0):=(others=>'0');

signal Valid\_solution ,test\_vector\_enable:std\_logic;

begin

u : final\_oracle port map(M1=>M1,M2=>M2,M3=>M3,M4=>M4,M5=>M5,M6=>M6,M7=>M7,M8=>M8,

 G1=>G1,G2=>G2,G3=>G3,G4=>G4,G5=>G5,G6=>G6,G7=>G7,G8=>G8,

 C1=>C1,C2=>C2,C3=>C3,C4=>C4,C5=>C5,C6=>C6,C7=>C7,C8=>C8,

 W1=>W1,W2=>W2,W3=>W3,W4=>W4,W5=>W5,W6=>W6,W7=>W7,W8=>W8,

 Valid\_solution =>Valid\_solution);

P1 : process (CLOCK)

begin

 if( reset ='1') then

 counter <="00000000000000000000000000000";

else

 counter <= counter+"00000000000000000000000000001";

 end if;

end process;

 W1<=counter(0); W2<=counter(1); W3<=counter(2); W4<=counter(3); W5<=counter(4); W6<=counter(5); W7<=counter(6); W8<=counter(7);

 M1<=counter(8); M2<=counter(9); M3<=counter(10); M4<=counter(11); M5<=counter(12); M6<=counter(13); M7<=counter(14); M8<=counter(15);

 G1<=counter(16); G2<=counter(17); G3<=counter(18); G4<=counter(19); G5<=counter(20); G6<=counter(21); G7<=counter(22); G8<=counter(23);

 C1<=counter(24); C2<=counter(25); C3<=counter(26); C4<=counter(27); C5<=counter(28); C6<=counter(29); C7<=counter(30); C8<=counter(31);

end beha;