**CHAPTER 9**

# SPECIALIZED PROCESSOR FOR MISSIONAIERS AND CANNIBALS CONSTRAINT SATISFACTION SEQUENTIAL PROBLEM.

## Problem Statement:

Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. The final goal is to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.

Rules of Transport:

1. At the max only 2 object can sail through the boat.
2. Any one of the 6 objects or any combination out of 6 objects can sail through the boat.

Main satisfiability condition here is count of missionaries and cannibals. For a move to be valid, the resulting states at both bank should be valid. i.e. the number of cannibals can’t be greater than number of missionaries.



So we need to have some kind of computational circuit which will tell whether the achieved state at both bank is valid or not. We also need other combinational circuits to check for the validity of move.

Let the Positive literal here means respective missionary or cannibals is on bank 1 and negated literal means respective missionary or cannibals is on bank 2.

One solution to this problem can be as shown in flowchart



Figure 9‑1 One possible solution of the puzzle

## Checking Safe Condition at Both Bank

Lets find the logical sets of equations for safe condition at both banks

Lets have following notation for 3 cannibals and 3 missionaries

Denotes 1st Cannibals at bank 2.

Denotes 1st Cannibals at bank 1.

C2 denotes 2nd Cannibals at bank2

Denotes 2nd Cannibals at bank 1

C3 denotes 3rd Cannibals at bank 2

Denotes 3rd Cannibals at bank 1

Denotes 1st Missionary at bank 2.

Denotes 1st Missionary at bank 1.

M2 denotes 2nd Missionary at bank 2

Denotes 2nd Missionary at bank 1

M3 denotes 3rd Missionary at bank 2

Denotes 3rd Missionary at bank 1

Lets us derive Valid state equation from invalid state equation. We can just negate the expression for invalid state to get valid state expression

Situation at Bank will be invalid when

This comprises of three cases

1 ) 3 Cannibals and 2 missionaries.

2 ) 3 Cannibals and 1 missionary

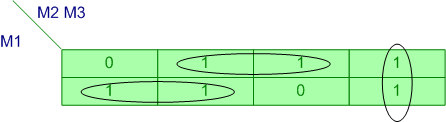
3) 2 Cannibals and 1 missionary

Lets take first Case of 3 Cannibals and 2 missionaries

The equation for 3 Cannibals and 2 missionaries at bank 1 is

Let us simplify expression

The k-map for this expression is



The final equation for Safe condition at bank 1 is   
…………….Eq A

Similarly safe condition at bank 2 will be simply negation of literal in above equation

……………….Eq B

So Un-safe condition at both bank with 3 cannibals and 2 missionaries is Eq A + Eq B

+ ………………………Eq C

Lets take the 2nd case of 3 Cannibals and 1 missionary at bank 1 is

…………….. Eq. D

Similarly the 2nd case of 3 Cannibals and 1 missionary at bank 2 is

……………….. Eq. E

And last take 3rd case of Similarly Equation for 2 cannibals and 1 missionary at bank 1 is



……………………………………………………………………………………Eq.F

Similarly we can find invalid condition for 2 cannibals and 1 missionary at bank 2 is



…………………………………………………………………………………Eq.G

So finally expression for Invalid condition at both banks including all possibilities will be

Eq.C + Eq.D +Eq.E +Eq.F + Eq.G

The expression for invalid condition will be negation of above expression.



Figure 9‑2 Oracle for checking invalid state

## Checking for invalid move

Cheating moves is when more than 2 object moves.

To simplify the equation for cheating move we can check for valid move.

A Move is valid when one only one object moves between banks or maximum two objects moves between banks.

Let’s find expression for these conditions separately and later we can just logically OR them.

If represent 1st missionary in state n when  represent 1st missionary in state n+1.

When M1 moves between two successive states then 

When M2 moves between two successive states then 

When M3 moves between two successive states then 

When C1 moves between two successive states then 

When C2 moves between two successive states then 

When C3 moves between two successive states then 

Let 

Let 

Let 

Let 

Let 

Let 

For every move to be correct

When only 1 object moves

……………………..Eq A

Now the second case of valid move in which maximum 2 objects can move between two banks is

+ + ++ …………….….Eq B

So for any move to be correct Eq A + Eq B =should evaluate to one.

**Figure 9‑3 Oracle for checking invalid moves**

Final Oracle will look like



## VHDL Code for Checking validity of state

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

Valid\_state : out std\_logic);

end check\_state;

architecture Behavioural of check\_state is

signal M1\_BAR,M2\_BAR, M3\_BAR,C1\_BAR,C2\_BAR,C3\_BAR:std\_logic;

signal temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9,temp10,temp11,temp12,temp13,temp14,temp15,temp16,temp17,temp18:std\_logic;

signal invalid\_state\_temp1,invalid\_state\_temp2,invalid\_state\_temp3,invalid\_state\_temp4,invalid\_state:std\_logic;

begin

M1\_BAR <= not M1; M2\_BAR <= not M2; M3\_BAR <= not M3;

C1\_BAR <= not C1; C2\_BAR <= not C2; C3\_BAR <= not C3;

-- Checking validity of state with 3 cannibals and 1 or 2 missionaries on any bank

temp1 <= C1\_BAR and C2\_BAR and C3\_BAR;

temp2 <= C1 and C2 and C3;

temp3 <= ((M1\_BAR and M3) or ( M2 and M3\_BAR) or (M1 and M2\_BAR));

temp4 <= ((M1 and M3\_BAR) or ( M2\_BAR and M3) or ( M1\_BAR and M2));

---Checking validity of state 2 cannibals and 1 missionary is

invalid\_state\_temp1 <= (temp1 and temp3) or ( temp2 and temp4);

temp5 <= (M1\_BAR and M2 and M3)or (M1 and M2\_BAR and M3) or (M1 and M2 and M3\_BAR);

temp6 <= (M1\_BAR and M2\_BAR and M3)or (M1 and M2\_BAR and M3\_BAR) or (M1\_BAR and M2 and M3\_BAR);

invalid\_state\_temp2 <= (temp1 and temp5) or (temp2 and temp6);

temp7 <= C1\_BAR and C2\_BAR and C3;

temp8 <= C1 and C2\_BAR and C3\_BAR;

temp9 <= C1\_BAR and C2 and C3\_BAR;

temp10<= M1\_BAR and M2 and M3;

temp10<= M1 and M2\_BAR and M3;

temp10<= M1 and M2 and M3\_BAR;

invalid\_state\_temp3 <= (temp7 and temp10) or (temp7 and temp11) or(temp7 and temp12) or (temp8 and temp10) or (temp8 and temp11) or(temp8 and temp12)

or (temp9 and temp10) or (temp9 and temp11) or(temp9 and temp12);

temp13 <= C1\_BAR and C2 and C3;

temp14 <= C1 and C2\_BAR and C3;

temp15 <= C1 and C2 and C3\_BAR;

temp16 <= M1\_BAR and M2\_BAR and M3;

temp17 <= M1\_BAR and M2 and M3\_BAR;

temp18 <= M1 and M2\_BAR and M3\_BAR;

invalid\_state\_temp4 <= (temp13 and temp16) or (temp13 and temp17) or(temp13 and temp18) or (temp14 and temp16) or (temp14 and temp17) or(temp14 and temp18)

or (temp15 and temp16) or (temp15 and temp17) or(temp15 and temp18);

invalid\_state <= invalid\_state\_temp1 or invalid\_state\_temp2 or invalid\_state\_temp3 or invalid\_state\_temp4;

valid\_state <= not invalid\_state;

end Behavioural;

## VHDL Code for checking validity of Move

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(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 check\_move;

architecture Behavioural of check\_move is

signal Z1,Z2,Z3,Z4,Z5,Z6:std\_logic;

signal temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9,temp10,temp11,temp12,temp13:std\_logic;

signal temp14,temp15,temp16,temp17,temp18,temp19,temp20,temp21:std\_logic;

signal valid\_move\_temp1,valid\_move\_temp2:std\_logic;

begin

Z1 <= C1\_ps xor C1\_ns; --C1 MOVES

Z2 <= C2\_ps xor C2\_ns;

Z3 <= C3\_ps xor C3\_ns;

Z4 <= M1\_ps xor M1\_ns;

Z5 <= M2\_ps xor M2\_ns;

Z6 <= M3\_ps xor M3\_ns;

temp1<= Z1 and (not Z2) and (not Z3) and (not Z4) and (not Z5) and (not Z6);

temp2<= (not Z1) and Z2 and (not Z3) and (not Z4) and (not Z5) and (not Z6);

temp3<= (not Z1) and (not Z2) and Z3 and (not Z4) and (not Z5) and (not Z6);

temp4<= (not Z1) and (not Z2) and (not Z3) and Z4 and (not Z5) and (not Z6);

temp5<= (not Z1) and (not Z2) and (not Z3) and (not Z4) and Z5 and (not Z6);

temp6<= (not Z1) and (not Z2) and (not Z3) and (not Z4) and (not Z5) and Z6;

valid\_move\_temp1 <= temp1 or temp2 or temp3 or temp4 or temp5 or temp6;

temp7<= Z1 and Z2 and (not Z3) and (not Z4) and (not Z5) and (not Z6);

temp8<= Z1 and (not Z2) and Z3 and (not Z4) and (not Z5) and (not Z6);

temp9<= Z1 and (not Z2) and (not Z3) and Z4 and (not Z5) and (not Z6);

temp10<= Z1 and (not Z2) and (not Z3) and (not Z4) and Z5 and (not Z6);

temp11<= Z1 and (not Z2) and (not Z3) and (not Z4) and (not Z5) and Z6;

temp12<= (not Z1) and Z2 and Z3 and (not Z4) and (not Z5) and (not Z6);

temp13<= (not Z1) and Z2 and (not Z3) and Z4 and (not Z5) and (not Z6);

temp14<= (not Z1) and Z2 and (not Z3) and (not Z4) and Z5 and (not Z6);

temp15<= (not Z1) and Z2 and (not Z3) and (not Z4) and (not Z5) and Z6;

temp16<= (not Z1) and (not Z2) and Z3 and Z4 and (not Z5) and (not Z6);

temp17<= (not Z1) and (not Z2) and Z3 and (not Z4) and Z5 and (not Z6);

temp18<= (not Z1) and (not Z2) and Z3 and (not Z4) and (not Z5) and Z6;

temp19<= (not Z1) and (not Z2) and (not Z3) and Z4 and Z5 and (not Z6);

temp20<= (not Z1) and (not Z2) and (not Z3) and Z4 and (not Z5) and Z6;

temp21<= (not Z1) and (not Z2) and (not Z3) and (not Z4) and Z5 and Z6;

valid\_move\_temp2 <= temp7 or temp8 or temp9 or temp10 or temp11 or temp12

or temp13 or temp14 or temp15 or temp16 or temp17 or temp18 or temp19 or temp20 or temp21;

valid\_move <= valid\_move\_temp1 or valid\_move\_temp2;

end Behavioural;

## Final VHDL Code for 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;

## Testbench For Emulation

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 oracle

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 component;

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

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

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

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

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

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

signal Valid\_State:std\_logic;

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(71 downto 0):=(others=>'0');

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

begin

u : oracle port map(M1\_state1=>M1\_state1,M1\_state2=>M1\_state2,M1\_state3=>M1\_state3,M1\_state4=>M1\_state4,M1\_state5=>M1\_state5,M1\_state6=>M1\_state6,

M1\_state7=>M1\_state7,M1\_state8=>M1\_state8,M1\_state9=>M1\_state9,M1\_state10=>M1\_state10,M1\_state11=>M1\_state11,M1\_state12=>M1\_state12,

M2\_state1=>M2\_state1,M2\_state2=>M2\_state2,M2\_state3=>M2\_state3,M2\_state4=>M2\_state4,M2\_state5=>M2\_state5,M2\_state6=>M2\_state6,

M2\_state7=>M2\_state7,M2\_state8=>M2\_state8,M2\_state9=>M2\_state9,M2\_state10=>M2\_state10,M2\_state11=>M2\_state11,M2\_state12=>M2\_state12,

M3\_state1=>M3\_state1,M3\_state2=>M3\_state2,M3\_state3=>M3\_state3,M3\_state4=>M3\_state4,M3\_state5=>M3\_state5,M3\_state6=>M3\_state6,

M3\_state7=>M3\_state7,M3\_state8=>M3\_state8,M3\_state9=>M3\_state9,M3\_state10=>M3\_state10,M3\_state11=>M3\_state11,M3\_state12=>M3\_state12,

C1\_state1=>C1\_state1,C1\_state2=>C1\_state2,C1\_state3=>C1\_state3,C1\_state4=>C1\_state4,C1\_state5=>C1\_state5,C1\_state6=>C1\_state6,

C1\_state7=>C1\_state7,C1\_state8=>C1\_state8,C1\_state9=>C1\_state9,C1\_state10=>C1\_state10,C1\_state11=>C1\_state11,C1\_state12=>C1\_state12,

C2\_state1=>C2\_state1,C2\_state2=>C2\_state2,C2\_state3=>C2\_state3,C2\_state4=>C2\_state4,C2\_state5=>C2\_state5,C2\_state6=>C2\_state6,

C2\_state7=>C2\_state7,C2\_state8=>C2\_state8,C2\_state9=>C2\_state9,C2\_state10=>C2\_state10,C2\_state11=>C2\_state11,C2\_state12=>C2\_state12,

C3\_state1=>C3\_state1,C3\_state2=>C3\_state2,C3\_state3=>C3\_state3,C3\_state4=>C3\_state4,C3\_state5=>C3\_state5,C3\_state6=>C3\_state6,

C3\_state7=>C3\_state7,C3\_state8=>C3\_state8,C3\_state9=>C3\_state9,C3\_state10=>C3\_state10,C3\_state11=>C3\_state11,C3\_state12=>C3\_state12);

P1 : process (clock)

begin

if( reset ='1') then

counter <="000000000000000000000000000000000000000000000000000000000000000000";

else

counter <= counter+"000000000000000000000000000000000000000000000000000000000000000001";

end if;

end process;

M1\_state1<=counter(0); M1\_state2<=counter(1);M1\_state3<=counter(2);M1\_state4<=counter(3); M1\_state5<=counter(4);M1\_state6<=counter(5);

M1\_state7<=counter(6); M1\_state8<=counter(7);M1\_state9<=counter(8);M1\_state10<=counter(9); M1\_state11<=counter(10);M1\_state12<=counter(11);

M2\_state1<=counter(12); M2\_state2<=counter(13);M2\_state3<=counter(14);M2\_state4<=counter(15); M2\_state5<=counter(16);M2\_state6<=counter(17);

M2\_state7<=counter(18); M2\_state8<=counter(19);M2\_state9<=counter(20);M2\_state10<=counter(21); M2\_state11<=counter(22);M2\_state12<=counter(23);

M3\_state1<=counter(24); M3\_state2<=counter(25);M3\_state3<=counter(26);M3\_state4<=counter(27); M3\_state5<=counter(28);M3\_state6<=counter(29);

M3\_state7<=counter(30); M3\_state8<=counter(31);M3\_state9<=counter(32);M3\_state10<=counter(33); M3\_state11<=counter(34);M3\_state12<=counter(35);

C1\_state1<=counter(36); C1\_state2<=counter(37);C1\_state3<=counter(38);C1\_state4<=counter(39); C1\_state5<=counter(40);C1\_state6<=counter(41);

C1\_state7<=counter(41); C1\_state8<=counter(43);C1\_state9<=counter(44);C1\_state10<=counter(45); C1\_state11<=counter(46);C1\_state12<=counter(47);

C2\_state1<=counter(48); C2\_state2<=counter(49);C2\_state3<=counter(50);C2\_state4<=counter(51); C2\_state5<=counter(52);C2\_state6<=counter(53);

C2\_state7<=counter(54); C2\_state8<=counter(55);C2\_state9<=counter(56);C2\_state10<=counter(57); C2\_state11<=counter(58);C2\_state12<=counter(59);

C3\_state1<=counter(60); C3\_state2<=counter(61);C3\_state3<=counter(62);C3\_state4<=counter(63); C3\_state5<=counter(64);C3\_state6<=counter(65);

C3\_state7<=counter(66); C3\_state8<=counter(67);C3\_state9<=counter(68);C3\_state10<=counter(69); C3\_state11<=counter(70);C3\_state12<=counter(71);

end beha;