The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

FAIL-SAFE VHDL DESCRIPTIONS OF PETRI NET SPECIFICATIONS

Miguel Pereira[1], Enrique Soto[2]

1 Intelsis Sistemas Inteligentes S.A. - R&D Digital Systems Department, Vía Edison 16
Polígono del Tambre 15890, Santiago de Compostela (La Coruña), mpereira@intelsis.es;
2 Dept. Tecnología Electrónica, Universidad de Vigo, Apdo. Oficial, 36200 Vigo,
España, esoto@uvigo.es, http://www.dte.uvigo.es;

Abstract. *This work presents a method for obtaining fail-safe systems based in parity alternation from Petri net specifications. A fail-safe system is a system with adequate redundancy for detecting failures and preventing them. This method generates a VHDL description of a system from a Petri net or state diagram description. These results have relevance in the integration of access technologies to high speed telecommunication networks, where fail safe mechanisms are becoming an important concern.*

Key Words. *Fail-safe Systems, Hardware Description Languages, Petri Net specifications*

1. Introduction

This work uses the parity alternation method to provide fail-safe [1] [2] [3] characteristics to graphical specifications of the type state diagram or Petri net [4]. The starting point is the unsafe state diagram or Petri net specification from which a VHDL description of the equivalent fail-safe system is generated. To do that a program providing a graphical interface reads the specification and internally generates a state array from a binary structure characterising the specification. The array is transformed appropriately to obtain a fail-safe system and turn it into a VHDL description. A block diagram of the processes involved is shown in figure 1.

2. Parity alternation method

This work uses the parity alternation method to provide the fail-safe mechanism, as described in [1]. The method and the proof of equivalence between the safe and unsafe specifications, which are available to the reader in the references, is not the objective of this paper. However, it can be summarised in the following paragraph.

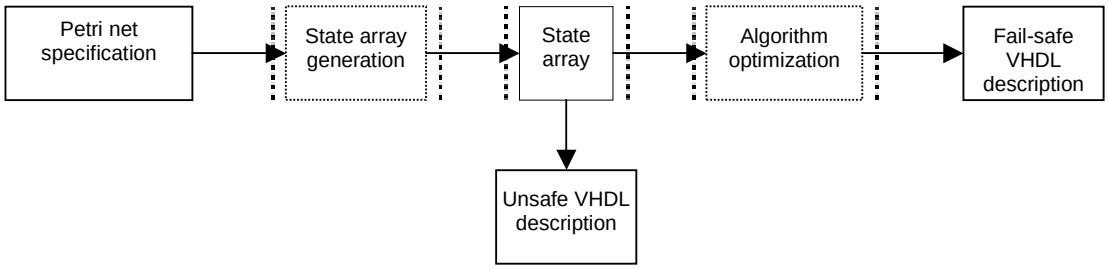

Fig. 1.Block diagram of the proposed methods

The method consists of coding every state in a state diagram with a different parity respect to those contiguous states in the diagram. Every transition in the state diagram goes from a source state to a destination state with a different parity. Fail-safe means that if an error occurs and a transition between two states with the same parity has occurred, then the error can be detected and the system driven to a known 'safe' state where no harm can be done. If the system could correct the error and keep working, then it would be called fault-tolerant (which is not the case). The new diagram is equivalent in the sense that its behaviour is identical from an external point of view.

3. Algorithm process

The initial structure from which all the rest derive is the starting state diagram. Figure 3a shows an example of a state diagram data structure for a Petri net. That information is stored in a structure formed by three object lists. The first is the list of places or states, the second is the list of transitions and the third is the list of the connections between places and transitions. The handling of the information is best achieved using an object oriented programming language. The hierarchy of the objects involved in the treatment of the specification is shown in figure 2, where objects in double boxes form the data structure, while the simple boxes are objects defined to inherit the properties provided by the programming language.

Tlugar defines the places and Ttransicion the transitions in the Petri net. TVar_ES is dedicated to input/output signals. TPuntero and inherited objects are pointers that relate all the structure. TPetriNet contains, with the help of all the other definitions, the information of the whole Petri net.

In the case of a state diagram, the information to store and the process to follow is simplified. The structure is first processed to obtain a state array, searching in the structure for every state its predecessor. This array is transformed as per [1] to get a new state array (see figure 1). This new state array is stored in a new class of objects from which the VHDL is generated. Since the process multiplies the number of states, it may be necessary to compress the information in memory.

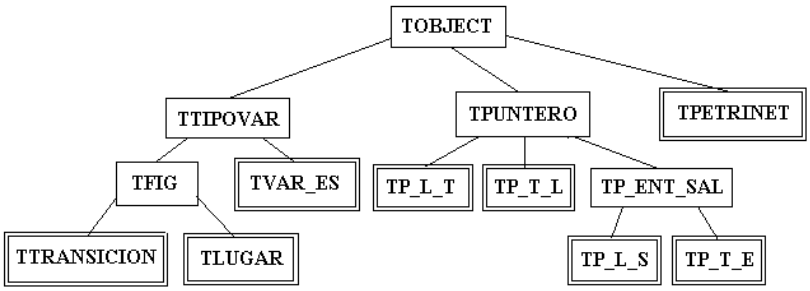

Fig. 2. Data structure

In the case of a Petri net, the process starts by transforming it in an equivalent state diagram, trying all the transitions between the places that can form a state, even if most will never happen. For a Petri net with 10 places, the equivalent state diagram can have at most 2 to the power of 10 states. That will make a 1024x1024-state array that if necessary to duplicate in the fail-safe version will make 2048x2048, forcing a compression of the information. The compression algorithm is facilitated by the fact that most of the values of the array will be '0's.

The application program identifies whether it is working with a Petri net or a pure state diagram, but uses the same data structure. If there is a transition associated to several places, coming from or leading to the transition, the structure is processed as a Petri net. Petri nets are encoded using one-hot encoding. State diagrams are encoded using binary codification. The number of bits used to code each state in this last case follows:

number_of_bits = exc(log2(number_of_states)) (1)

where exc (x) means rounded up.

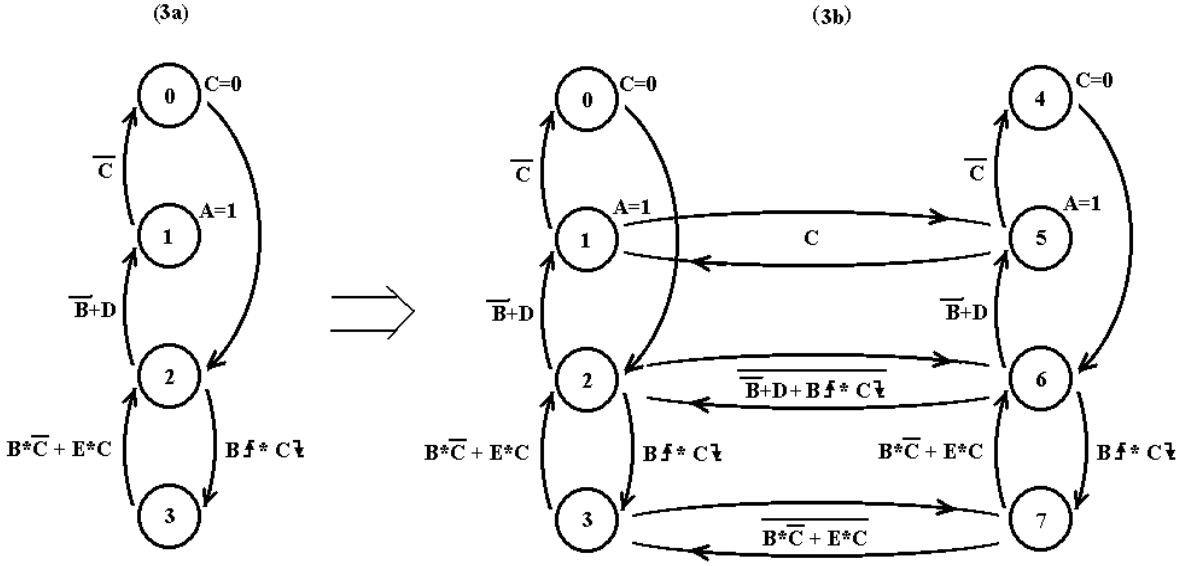

Fig. 3. Example

4. Examples

Figure 3a shows a state diagram, list 1 its VHDL entity description, list 2 the unsafe version and list 3 the results obtained after been transformed by the algorithm into a fail-safe VHDL description (see figure 3b). Observe that it was necessary to duplicate the number of states to obtain the required parity alternation.

REFERENCES

[1] J.J.Rodríguez Andina, J.Álvarez y E.Mandado, “Design of safety systems using Field Programmable Gate Arrays”, FPL’94 – 4th International Workshop on Field Programmable Logic and Applications, Springer-Verlag, ISBN 3-540-58419-6, Praga 94.

[2] J.J.Rodríguez Andina, S.Fernández y E.Mandado, “Implementation of logic controllers with concurrent fault detection capabilities in PLDs”, , IOLTW’96 – 2nd IEEE International On-Line Testing Workshop, St. Jean-de-Luz 96.

[3] J.J. Rodriguez Andina, Santiago Fernandez and Enrique Mandado, "Design and Validation of Fail-Safe FSMs Using Regular Structures", Proceedings of the XII Design of Circuits and Integrated Systems Conference - DCIS'97, pp. 131-136. Sevilla, November 18-21, 1997.

[4] Zurawski, R., M.C. Zhou. "Petri Nets and Industrial Applications: a Tutorial". IEEE Trans. on Industrial Electronics, Dec. 1994.

ACKNOWLEDGEMENTS

This work was financed by the European Commission and the Comisión Interministerial de Ciencia y Tecnología (Spain) through research grant TIC 1FD97-2248-C02-02 in collaboration with the company Versaware S.L. (Vigo, Spain).

List 1

```
-- Entity description of Petri net example
Entity Example is
Port (
clk: In std_logic;
INIT: In Std_logic;
B, C, D, E: In Std_logic;
X,Y: Out Std_logic  );
end Example;
```

List 2

```
-- Unsafe architecture
architecture unsafe of Example is
--Biestables for detecting transition activated inputs
Signal  ant_B, ant_C:  Std_logic;
-- Current state register
Signal  current_state: Integer range 0 to 3;

begin
-- Descriptions of the transitions
transic: process
begin -- Only with a positive transition in clk
wait until (clk'event and clk='1');
if INIT = '0' then current_state <= 2 ;  --Init. signal
else
case current_state is
-- state 0
when 0 =>  current_state <= 2 ;

-- state 1
when 1 => if C='0' then current_state <= 0 ; end if;

-- state 2
when 2 =>
if ( (B='1') and (ant_B='0') ) or ( (C='0') and (ant_C='1') )
then  current_state <= 3 ;  end if;
if ( B='0' ) or ( D='1' ) then current_state <= 1;  end if;

-- state 3
when 3 =>  if ( (B='1') and (C='0') ) or ( (E='1') and (C='1')
)
then  current_state <= 2 ;   end if;

when others => null;

end case;
end if;
end process;

-- Description of the outputs
salidas: process
begin
wait until clk'event and clk='1';
case current_state is
when 0 =>  Y <= '0';
when 1 =>  X <= '1';
when others => null;
end case;
end process;

-- Description of transition activated inputs
flancos: process
begin -- Only with a positive transition in clk
wait until clk'event and clk='1';
ant_B <= B;  ant_C <= C;
end process;
end unsafe;
```

List 3

```
architecture safe of Example is
--Biestables for detecting transition activated inputs
Signal  ant_B, ant_C:  Std_logic;
-- Current state register
Signal  current_state: Integer range 0 to 7;

begin
 -- Description of the transitions
transic: process
begin -- Only with a positive transition in clk
wait until clk'event and clk='1';

if INIT = '0' then current_state <= 2;  -- Init. signal
else
case current_state is

-- state 0
when 0 =>  current_state <= 2 ;

-- state 1
when 1 => if C='0'then current_state <= 0 ;
else current_state <= 5 ;   end if;

-- state 2
when 2 =>
if ((B='1') and (ant_B='0')) or ((C='0') and (ant_C='1'))
then current_state <= 3 ;
elsif (B='0') or (D='1')then current_state <= 5 ;
else  current_state <= 6 ;  end if;

-- state 3
when 3 => if((B='1') and(C='0')) or((E='1') and(C='1'))
then current_state <= 2 ;
else current_state <= 7 ;    end if;

-- state 4
when 4 =>  current_state <= 6 ;

-- state 5
when 5 => if C='0' then current_state <= 4 ;
else current_state <= 1 ;  end if;

-- state 6
when 6 =>   if(B='0') or(D='1') then current_state <= 1 ;
elsif ( (B='1') and (ant_B='0') ) or ((C='0') and (ant_C='1')
) then current_state <= 7 ;
else current_state <= 2 ;  end if;

-- state 7
when 7 =>  if ( (B='1') and(C='0') ) or( (E='1') and(C='1') )
then current_state <= 6 ;
else current_state <= 3 ;   end if;

when others => null;
end case;
end if;
end process;

 -- Descriptions of the outputs
 salidas : process
begin  --Only with a positive transition in clk
 wait until clk'event and clk='1';
case current_state is
when 0 =>  Y <= '0';
when 1 =>  X <= '1';
when 4 =>  Y <= '0';
when 5 =>  X <= '1';
when others => null;
end case;
end process;

-- Description of transition activated inputs
flancos: process
begin
-- Only with a positive transition in clk
wait until clk'event and clk='1';
 ant_B <= B;  ant_C <= C;
end process;

end safe;
```