Petri net-based modeling and VHDL implementation of digital systems

Jun Wan
School of Urban Railway Transportation
Changzhou University
Changzhou Jiangsu, China

***Abstract*—Petri net-based digital systems modeling and hardware implementation method is well-studied. However, the existing methods are mainly applied to synchronous circuits. In this paper, a new approach is proposed for the modeling and VHDL implementation of digital systems based on an extended class of Petri nets. The generalized synchronous self-modifying net (GSSN) is defined to describe digital systems. Based on a new kind of D flip-flop, known as multi-input & multi-clock D flip-flop, the specific conversion algorithm from Petri nets models to VHDL codes is presented. A design example is given to illustrate the capabilities of the proposed approach. The approach presented in the paper is suitable for both asynchronous systems and synchronous systems.**

Keywords—Petri net; modeling; digital system; VHDL

I. INTRODUCTION

Petri nets are a powerful tool for describing asynchronous and concurrent systems. Petri net-based digital system design method attracts the attention of many researchers, who extended the definition of Petri nets and developed related tools so as to support the entire digital system design process ranging from modeling to hardware implementation. In [1], Signal Interpreted Petri Nets (SIPN) was used to model logical controllers and a SIPN-Editor toolbox was developed that allows the design and analysis of SIPN algorithms. In order to implement algorithms on FPGA, the SIPN-Editor supports directly mapping to VHDL language. In [2-5], L. Gomes et al. propose Input-Output Petri Net (IOPT) represented by using the Petri Net Markup Language (PNML) for the specification of digital controllers, and has developed a set of tools including a graphical editor, state space analyzer, conflict resolution, automatic code generation, and emulators, thereby the complete system development process from specification to implementation is strongly supported. H. Kubátová also uses PNML to describe Petri net model and designs the Petri-nets basic building blocks for its hardware implementation [6].Based on the basic building block blocks, the automatically conversion program named Pnml2vhdl is designed, which can perform the conversion process from PNML to VHDL code. A digital system design method based on direct translation of a Petri net model into a FPGA circuit netlist was described [7, 8]. In [7], a CAD tool has been proposed, which allows digital system specification, modeling and implementation by using ordinary Petri nets. According to the methods mentioned above, the synthesized circuits from Petri net models are all synchronous circuits.

Petri nets have been used for asynchronous systems modeli-ng, analysis and verification for a long time. Compared with synchronous circuit, asynchronous circuit has many advantages such as low power consumption, low electromagnetic radiation, high speed, high degree of modularity, no clock skew, etc. As [9] pointed out,” Petri nets model for asynchronous circuit design can be as important as that of a state machine for synchronous circuit design. We strongly believe that this model can play a pivotal role in future synthesis tools development for asynchronous systems.” A new method of asynchronous controller modeling and implementation of FPGA is proposed in [10], where the modeling of discrete event control systems is achieved by Safe Automation Petri Nets (SAPN), furthermore the hardware implementation of SAPN is accomplished by direct mapping from SAPN to circuit elements. Colored Petri nets and Labelled Petri nets are used as intermediate formats for datapath and control representation respectively [11]. Speed-independent asynchronous circuits are obtained from these nets via direct translation, that is, the control channel and some of the data channels are mapped directly to David unit control circuit. Paper [12] used a new kind of Petri net called PN-DFG as a bridge between the high-level behavior description and the logic gate level, and developed the PAiD tool to design and verify asynchronous circuits. In paper [13], Petri nets-based methods of modeling, analysis, verification and synthesis of asynchronous systems were introduced in detail. D flip-flop is used to implement Petri nets in [14]. But if a place involves multiple transitions, the decoding control circuit is complex and it is difficult to capture the pulse-trigger signal and resolve the conflict.

Using Petri nets for asynchronous circuit design needs to solve two important problems. i) Modeling capability of Petri nets. For Petri nets, only describing the system in the Register Transfer Level is not enough. Some scholars have proposed to adopt the globally asynchronous locally synchronous(GALS) method [15], which can make full use of IP resources of synchronous design. ii) Only mapping of Petri nets to circuits directly can avoid the state explosion problem, but the circuit implementation requires supporting logic devices. In synchronous circuit， the flip-flops used require only one global clock, which are not suitable for the implementation of places with multiple input or output transitions.

In this paper, a new class Petri nets, generalized synchronous self-modifying net, is proposed and used to modeling digital system. Based on a new D flip-flop, hardware implementation of generalized synchronous self-modifying net model is realized. Finally, a design example illustrates the capabilities of the proposed approach.

II. Generalized Synchronous self-modifying Net

Definition 1 A generalized self-modifying net is a tuple Σ= (S, T; F, K, W, M) satisfying the following requirements:

1) S is a finite set of places;

2) T is a finite set of transition;

3) F is a finite set of arcs, such that $F \subseteq S \times T \cup T \times S$;

4) K={K_L, K_H} is a capacity function of Σ,where K_L is an lower bound capacity function and K_H is an upper bound capacity function. $\forall s \in S$, the capacity of s can be expressed as [K_L (s), K_H (s)]. When K_L (s) and K_H (s) are infinite, the closed interval of capacity turns into open interval;

3) W: $F \rightarrow \mathbf{R} \cup$ Exp (S) is a weight function, where **R** is a real number set and Exp (S) is a set of function expressions with elements in S as variables;

4) M:$S \rightarrow \mathbf{R}$ is the marking function. M_0 is the initial marking.

Definition 2 (firing condition and result of a transition)

1) The weight of an arc under marking M is defined as:

$\forall (x, y) \in S \times T \cup T \times S$,

$$W_M(x,y)=\begin{cases} W(x,y) & W(x,y) \notin Exp(S), \\ e_M & W(x,y)=e, \ \ e \in Exp(S). \end{cases} \tag{1}$$

By using $M(s_i)$ to replace $s_i \in S$(i=1, 2,…) in the expression e *and evaluate it*, e_M, the result of expression e, can be obtained.

2) The firing conditions for transition t under marking M are as follows:

$\forall s \in {}^\bullet t$:($M$($s$) -$W_M$($s$,$t$))$\in$[$k_L$($s$),$k_H$($s$)] and

$\forall s \in t^\bullet$:(M(s) +W_M(t, s))$\in$[k_L(s), k_H(s)]. (2)

In (2), ${}^\bullet t$ and $t^\bullet$ is the preset and postset of transition t.

3）If transition t can be firable under M, M is changed to its successor M' after firing of t. M' is defined as follows:

$$M'(s)=\begin{cases} M(s)-W_M(s,t), & s \in {}^\bullet t - t^\bullet, \\ M(s)+W_M(t,s), & s \in t^\bullet - {}^\bullet t, \\ M(s)-W_M(s,t)+W_M(t,s), & s \in {}^\bullet t \cap t^\bullet, \\ M(s), & s \notin {}^\bullet t^\bullet. \end{cases} \tag{3}$$

Definition 3 A generalized synchronous self-modifying net (GSSN) is a triple (Σ, E, G) satisfying the following requirements:

1) Σ=(S, T; F, K, W, M_0) is a generalized self-modifying net;

2) E is a set of external events;

3) G:$T \rightarrow$ E $\cup$ {e} is an event function in which e is the always occurring event.

In a GSSN, an event is associated with each transition, and the firing of a transition will occur if the transition is enabled and its associated event occurs. Since a net can be simplified by eliminating the transitions associated with the always occurring event, this paper only considers external events.

Fig.1(a) is a generalized synchronous self-modifying net. Fig.1(b) shows the timing of X1,X2 as well as the evolution of the markings. In a GSSN, two kinds of special weight control arcs, reading arc and writing arc, are introduced. In Fig.1(a), arc (P_1,T_1) is reading arc because the tokens in place S_1 keep without any change when transition T_1 fires, but arc (T_2,P_3) is writing arc. When transition T_2 fires, the tokens in place S_3 is updated only by the weight function value of the arc (T_2,P_3), regardless of its original number.

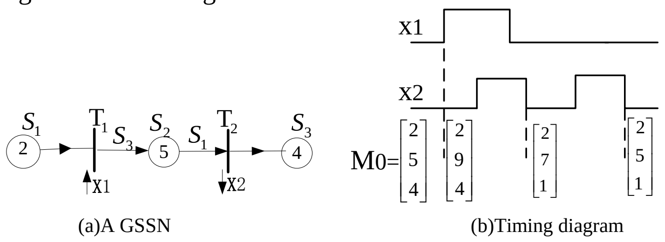


Fig. 1. Example of a GSSN

III. VHDL implementation of GSSN model

Conversion from the GSSN model to VHDL code is based on the direct mapping strategy, namely, each place is mapped to a memory element and each weight function is mapped to the combination circuit on behalf of a specific operation expression. The firing of transition is to latch the operation result of a combination circuit into the memory.

A. multi-input & multi-clock D flip-flop and IP core

To meet the need for data latch in the event-driven application, thereby providing effective support device for the asynchronous circuit design, a kind of D flip-flop, named multi-input & multi-clock D flip-flop has been used[16]. Such a D flip-flop is formed by a combination of one or more input units and one multi-input RS flip-flop. Every input unit has one data port, one clock port and two output ports which are connected to the input ports of a RS flip-flop. The clock port receives an external event, whose occurrence will cause the data of the corresponding data port be latched. RST port and PRESET port are used for setting the initial value. Considering the priority of the clock trigger events, clock priority circuits are constructed by using AND gates. If two or more clock signals are valid simultaneously, the signal of the data port in the input unit who has higher priority will output to Q port.

Fig. 2 demonstrates the circuit diagram of 2-input & 2-clock D flip-flop. This flip-flop has two data ports D1 and D2. The corresponding three clock ports are CP1 and CP2, respectively, with their priority order of CP1>CP2.

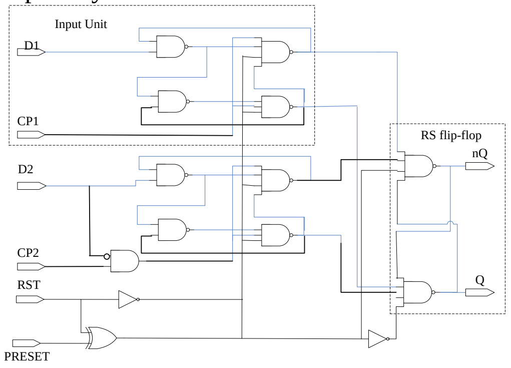


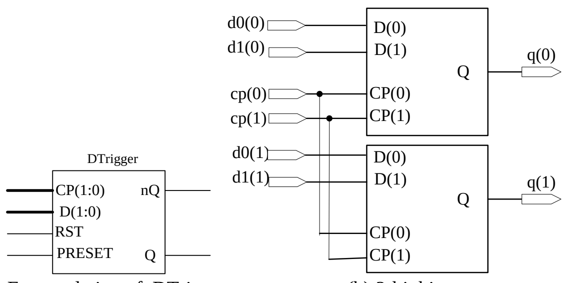
Fig. 2. Schematic diagram of multi-input and multi-clock D flip-flop

The truth table of such D flip-flop is given in Table 1.

TABLE I. TRUTH TABLE OF MULTI-INPUT & MULTI-CLOCK D FLIP-FLOP

RST	PRESET	D	CP	Q	nQ
1	0	X	X	0	1
1	1	X	X	1	0
0	X	0	↑	0	1
0	X	1	↑	1	0

By using hardware description language, such kind of D flip-flop is designed to soft IP core which can be configured and invoked. Fig.3(a) is the interface of IP core named DTrigger, corresponding to the 2-input & 2-clock D flip-flop. One multi-input & multi-clock D flip-flop latches only one-bit data, so for every place in GSSN model, it requires the memory composed of multiple flip-flops. Fig.3(b) shows a 2-bit binary memory composed by two DTrigger components. It has two groups of inputs (d0[1‥0], d1[1‥0]), one group of outputs(q[1‥0]) and two clocks (cp[1‥0]).



(a) Encapsulation of DTrigger (b) 2-bit binary memory

Fig. 3. IP core of multi-input and multi-clock D flip-flop and memory

B. VHDL code conversion from *GSSN*

The code conversion strategy from model to VHDL code is place-centered, that is, mapping every place to the corresponding memory constituted by multi-input & multi-clock D flip-flop mentioned above.

Considering a place *s* in a model, VHDL code conversion algorithm mapping *s* to memory *V* will consider the following steps.

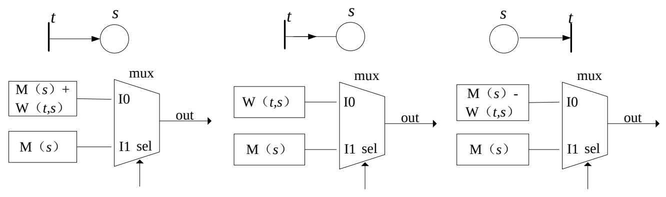
1) Construct clock signals connected to clock ports of *V*.

The number of clock ports of *V* is equal to the number of transitions in the preset and postset of *s*.

Suppose $t \in {}^{\bullet}s \cup s^{\bullet}$, ${}^{\bullet}s$ and $s^{\bullet}$ is the preset and postset of *s*. If *t* associates with the external signal *e*, *e* will be mapped to one of the clock signals connected to clock ports of *V*.

2) Construct data signals connected to data ports of V.

Every data signal is provided by a 2-to-1 multiplexer. The select control signal of multiplexer is determined by the enabling condition of transition connected to *s*, while the two input signals of multiplexer are determined according to the type and weight of the arcs connected to *s*. In Fig.4(a), the type of arc (*t*, *s*) is ordinary arc, when *t* is enabled, the input signal whose *value is* $W(t, s)+M(s)$ will be selected, otherwise the input signal whose *value is* $M(s)$ will be selected. Fig.4(b) and Fig.4(c) are similar to Fig.4(a), except that the type of arc (*t*, *s*) in Fig.4(b) is writing arc and the arc (*s*, *t*) in Fig.4(c) is output arc of *s* instead of input arc.



(a) Ordinary input arc of *s* (b) Writing arc of *s* (c) Ordinary output arc of *s*

Fig. 4. Mapping from arcs of place *s* to circuit

The 2-to-1 multiplexer can be described easily by the condition assignment statement in VHDL Language.

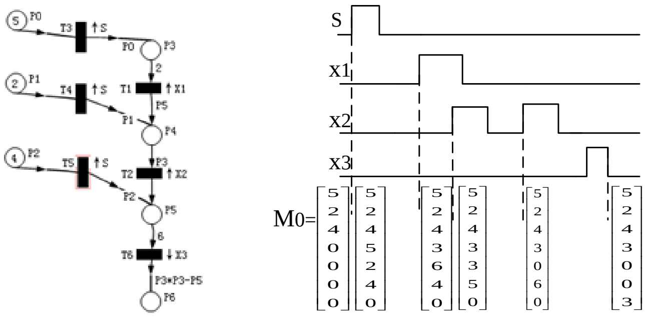
3) Construct memory *V* by creating instances of DTrigger component.

The capacity function of place *s* can determine the memory width of *V*, that is, can determine the number of DTrigger components. For an instance of DTrigger component, the signals of its CP ports are clock signals constructed in step 1), and the signals of its D ports are combination of data signals constructed in step 2). The signal PRESET is determined by the number of initial tokens in *s*.

IV. A design example

To clarify the concepts presented in the preceding sections, a simple calculation example is presented. This example relates to four data: X, Y, Z and W. When the event ↑S occurs, three data are read into X, Y and Z from the outside. If the event ↑X1 occurs and X≥2 is satisfied, X=X-2 and Y=Y+Z. When the event ↑X2 occurs, if Y≥X, Y=Y-X and Z=Z+1. If event ↓X3 occurs and Z≥6 is satisfied, Z=Z-6 and W=X*X-Z.

The GSSN model of this example was created and shown in Fig.5(a). In this model, there are seven places, wherein, places P0, P1 and P2 holds the initial value of *X*, *Y and Z* respectively, while places P3, P4, P5 and P6 represents *X*, *Y*, *Z* and *W* respectively. Under the control of the given timing of signals in Fig.5(b), all markings reachable from the initial marking can be obtained.



(a)A GSSN model (b) Timing diagram and markings evolution graph

Fig. 5. GSSN model and its markings evolution graph of the example

By using the conversion algorithm in section III(B), this model can be converted to corresponding VHDL code. TABLE II illustrates the generated VHDL code for place P6.

TABLE II. VHDL CODE FOR THE MODEL IN FIG. 5(A)

```
entity DTriggerGroup is
Port  (P0,P1,P2,P3,P4,P5,P6  :  inout  std_logic_vector(widthn-1
downto 0);
   X1,X2,X3,S : in std_logic;
   rst :in std_logic
);
end DTriggerGroup;
architecture Behaviour of DTriggerGroup is
COMPONENT DTrigger
GENERIC ( n : INTEGER );
PORT(D,CP : IN STD_LOGIC_VECTOR(0 TO n-1);
      RST,PRESET:IN STD_LOGIC;
      Q, nQ: OUT STD_LOGIC);
end COMPONENT;
begin
…
-------------------------------------P6---------------------------------
P6_CP (0)<= NOT X3;
P6_D1<=conv_std_logic_vector(conv_integer(P3*P3-P5),5);
instanceP6: for j in 0 to widthn-1 generate
      data6(j)(0)<= P6_D1 (j);
      P6Triggerj: DTrigger GENERIC MAP (n =>1)
      PORT MAP(D=> data6(j), CP=> P6_CP,
      RST=> rst, PRESET =>P6Init(j),Q=> P6(j),nQ=>qb6(j));
end generate instanceP6;
end Behaviour;
```

In the above code, the constant widthn represents the number of DTrigger components in instanceP6 .The signal rst is the system reset signal and P6Init holds the initial tokens in P6. P6_CP and P6_D1 hold the clock signal and data signal of instanceP6 respectively. Functional simulation of the generated VHDL code has been performed on the Xilinx ISE platform and the simulation result (see Fig. 6) is consistent with the marking evolution in Fig. 5(b).

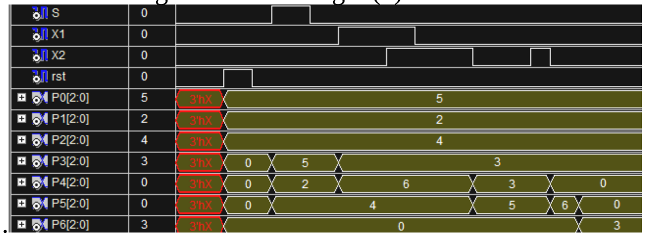


Fig. 6. Functional simulation result on the ISE simulator

V. CONCLUSION

Petri nets are an ideal tool for modeling of asynchronous concurrent systems. However, due to the insufficiency of data type and algorithm, lacking of support device for circuit implementation as well as effective software tools, the application of Petri nets in asynchronous systems has been limited. By expanding the definition of Petri nets, generalized synchronous self-modifying net is presented to describe the digital systems. The use of multi-input & multi-clock D flip-flop enables the GSSN model to be directly mapped into asynchronous circuit. As the further works, we will develop a CAD tool which can provide a complete development environment for model input, functional analysis and VHDL conversion.

ACKNOWLEDGMENT

This research was supported by Cooperative innovation fund project in Jiangsu Province under Grant SBY2014020188.

REFERENCES

[1] F.Wagner, P. Münch, S. Liu, G. Frey, "Development process for dependable high-performance controllers using Petri Nets and FPGA technology," in *Proc. DCDS*, 2007, pp. 139-144.

[2] L. Gomes, A. Costa, J.P. Barros, P. Lima "From Petri Net models to VHDL Implementation of digital controllers," in *Proc. IECON*, 2007, pp. 94-99.

[3] L. Gomes, J.P. Barros, A. Costa, "The Input-Output Place-Transition Petri Net class and associated tools," in *Proc. INDIN*, 2007, pp. 509-514.

[4] A. Costa, L. Gomes, J.P. Barros, "Petri nets tools framework supporting FPGA-based controller implementations," in *Proc. HSI*, 2008, pp. 2477-2482.

[5] F. Moutinho, L. Gomes, "From models to controllers integrating graphical animation in FPGA through automatic code generation," in *Proc. ISIE*, 2009, pp. 712-717.

[6] H. Kubátová, "Direct implementation of Petri Net based model in FPGA," in *Proc. DESDes*, 2004, pp. 31-36.

[7] V. Sudacevschi, V. Ababii, E. Gutuleac, "Digital systems synthesis based on direct translation of Petri Net model," in *Proc. DAS*, 2012, pp. 149-153.

[8] V. Sudacevschi, V. Ababii, E. Gutuleac, "HDL implementation from Petri Nets description," in *Proc. DAS*, 2010, pp. 236-240.

[9] D. Sokolov, A. Yakovlev, "Clockless circuits and system synthesis," *IEE Proc.-Computers & Digital Techniques*, vol. 152, no. 3, pp. 298-316, Mar. 2005.

[10] U. Murat, I. B. Koç, G. Gelen, and B. H. Aksebzeci, "Asynchronous implementation of discrete event controllers based on safe automation Petri nets," *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 5-6, pp. 595–612. Mar.2009.

[11] D. Shang, A. Burns, K.A. Yakovlev, "Asynchronous system synthesis based on direct mapping using VHDL and Petri nets," *IEE Proc.-Computers & Digital Techniques*, vol. 141, no. 3, pp. 209–220, Mar. 2004.

[12] K. Lehuu, T. Nguyen, H. B.Thang, A. V. Dinhduc, "Asynchronous circuit verification: from specification to circuit", In: International Conference on Computing, Management and Telecommunications, Ho Chi Minh, Vietnam, 2013, pp. 420-425.

[13] Yakovlev A, Gomes L, Lavagno L. Hardware Design and Petri Nets. New York: Springer-Verlag New York Inc., 2010.

[14] B.H. Zhao, Y.G. Yan, "Petri Net-Based design method of digital Circuits," in *Proc. WCICA*, 2002, pp. 10-14.

[15] F. Moutinho, L.Gomes, P.Barbosa, J.P. Barros, F. Ramalho, J. Figueiredo, A.Costa, A. Monteiro, "Petri net based specification and verification of globally-asynchronous-locally-synchronous system," in *Proc. DoCEIS, 2011 ,pp.237-245.*

[16] B. H. Zhao, "A multi input and multi clock maintenance obstructive type D trigger," China Patent 102355235, Feb. 15, 2012.