Electronic System Design Automation using High Level Petri Nets*

Patrick Rokyt [1], Wolfgang Fengler[2], Thorsten Hummel[2]

[1] Siemens AG, OEN ET S21, Machtlfingerstr. 1
D-81359 Muenchen, Germany
Patrik.Rokyta@oen.siemens.d

[2] Technical University of Ilmenau, P.O. Box 100 565,
D-98684 Ilmenau, Germany
{wfengler, thummel}@theoinf.tu-ilmenau.d

Abstract. A design and implementation methodology for system specification, modelling and implementation using a special kind of high level Petri nets is described. Electronic system design automation tools are used to generate synthesizable VHDL code from a Petri net model. For the design of large systems with regular structures using of coloured Petri nets will improve the handling and flexibility. Two design examples illustrate the described methodology.

1 Introduction

A design and implementation methodology must provide tools for system specification, modelling and implementation.
The expected behaviour of the system (behavioural model) is captured in the specification. This model is useful to check the algorithm of the system without any constraints towards implementation.
A formal language is used to capture a RTL model of the design. The most popular languages for capturing a RTL model are Verilog and VHDL [5]. These hardware description languages can be synthesized to a gatelevel netlist.
Petri nets have shown to be a powerful formal language to specify and model the behaviour of parallel systems at a high abstraction level. Since a PN can be analysed, the error detection is possible without any verification methods.

Electronic System Design Automation (ESDA) allows to capture a graphical model of the design using state diagrams, flow charts or truth tables.
The creation of a synthesizable VHDL model is provided by a built-in VHDL generator.

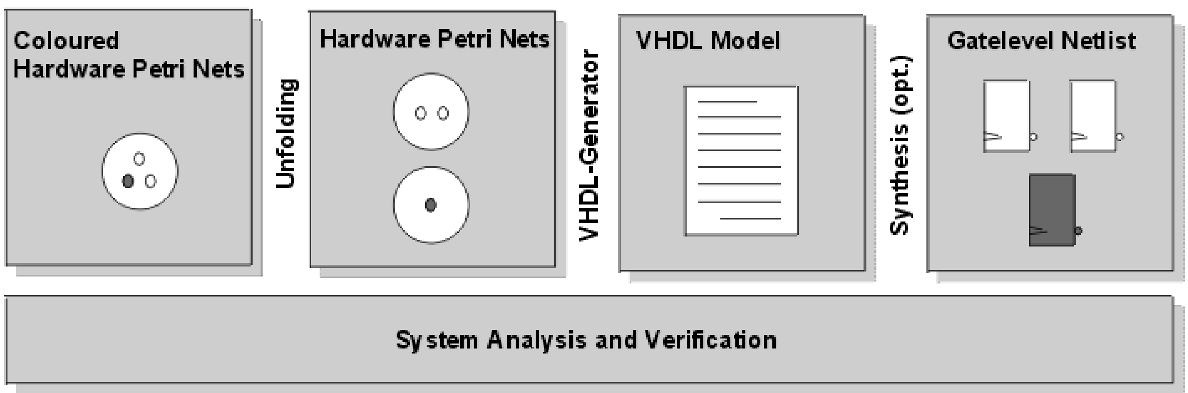


Fig. 1 ESDA based System Design using Hardware Petri Nets

* This work is supported in parts by DFG grant GRK 164/1-96

Since Petri nets are more flexible for modelling parallel designs they should be used instead of state machines. Using Petri nets the merger between the behaviour and the state machine can be fulfilled. The analysis of the design can be extended to its behaviour (i.e. reachable values of the output signals instead of reachable state vectors).
The VHDL generator must extract a fully synthesizable VHDL code out of any type of Petri net model.

2 The Hardware Petri Net

In order to use Petri nets for capturing a design for verification or a design for manufacturing, the Place Transition Net must be extended for purposes, which allow high level signal modelling. This kind of Petri net is refered to as the Hardware Petri Net (HPN):

- The Places store a certain amount of tokens up to their capacity.
This feature allows modelling of bus signals. The capacity of the place must correspond with the width of the bus.
- When a place stores at least one token the corresponding asynchronous I/O-function, which is attached to the place, is executed. This is supposed to be used for modelling of asynchronous behaviour. The function is defined using VHDL statements and must be fully synthesizable.
- When a transition fires, the corresponding synchronous I/O-function, which is attached to this transition, is executed. This is useful for modelling of synchronous behaviour. The function is defined using VHDL statements and must be fully synthesizable.
- Besides known arcs, enabling and setting arcs are defined.
Enabling arcs define an additional firing condition related to the marking of the place the arc is connected with. Any condition is allowed on this arc (e.g. < 3, >= 4). An inhibitor arc, which is included in the enabling arcs, provides an enabling function, when the place stores no token (= 0).
Setting arcs cause the place to obtain a certain amount of tokens, which is defined on this arc. Since a place is represented by a flip-flop in a logical design, the setting arc can be used to synchronize external signals with the active edge of the system clock.
- An additional external firing condition can be attached to any transition.
Though, this can cause timing violations at the flip-flops.

Using HPN for signal modelling allows the modelling of either synchronous or asynchronous behaviour. Setting the corresponding default value for asynchronous signals, a latch, a tri-state or a combinational logic design can be synthesized. A reset value can be defined for any signal of the design which takes place if reset is active.

Using I/O-functions either on places or on transitions, a Mealy machine is synthesized. For a critical design using places with the corresponding capacity (1 for 1 bit, 3 for 2 bits, 15 for 4 bits) to represent the value of the output signals, a Moore machine is synthesized.
Firing transitions are synchronized with the active edge of a clock.

3 Hierarchical Design and Partitioning

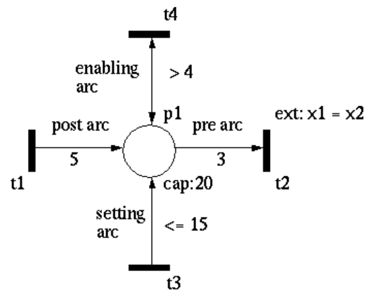
The state vector of a HPN is built out of bit vectors where each of them corresponds with the capacity of the related place. Since the design can have an unlimited amount of places with unlimited (the highest integer used in the synthesis tool) capacity, the state vector can become very large. Since the FSM optimization does not work with large state vectors, the design must be split into several (hierarchical) blocks. The FSM optimization (reachable states, encoding style) can be applied to the lowest level of the hierarchy.
The partitioning of a HPN design is fulfilled at the top level. In this case the design is split into several units. Each unit is synchronized with individual clock and reset.
Withi each unit a hierarchical HPN can be captured using macros. A macro allows to descent to the next hierarchy level. Since the hierarchy is directly translated to a VHDL model, incremental synthesis is possible.
For each signal within a unit only one driver is synthesized. To get more drivers for one signal, the signal must be modeled in more than one unit and must be defined as a tri-state signal.

4 Generation of synthesizable VHDL-Code

Using HPN a generation of synthesizable VHDL-Code is possible for any kind of design. Using the reachable states the synthesis result can be optimized since the lowest level of the hierarchy is always a FSM.
The top level of a HPN design consists of the unit instances. Since a unit requires a synchronous reset the reset is synchronized outside this unit with the unit clock and is handled as an asynchronous reset within this unit in order to save logic and get more efficient synthesis results.
Each VHDL model of a HPN unit consists of the synchronous and the asynchronous I/O-process where al I/O-functions, which are attached to the places and the transitions of this unit, are executed. The execution order is controlled by the priority value attached to these elements.

The most important part within a HPN unit is the FSM instance which simulates the firing of transitions. This FSM is a separated VHDL model. In this model the firing conditions of the transitions are calculated. A special VHDL process detects and removes any firing hazards. Another VHDL process controls the token flow within the unit.

Since the FSM can be a hierarchical design using macros it can consist of other instances of FSM at the lowe level.

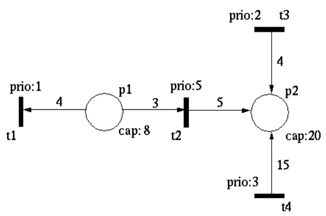


firing conditions:

```
t1 <= (p1 <= 15);
t2 <= (p1 >= 3);
t3 <= (15 <= 20); -- TRUE
t4 <= (p1 > 4);
```

Fig. 2 Firing condition of the transition

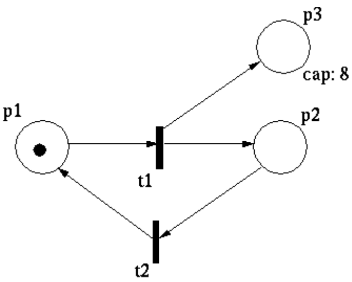
The firing condition of the transitions are calculated out of the markings of the connected places and the token value of the related arcs. The firing condition is extended by the external firing condition which can be attached to the transition. Enabling arcs and setting arcs affect the firing condition of the transition as well (Fig. 2). The token value of the setting arc must not exceed the capacity of the related place element.




```
conflict solution:¶
¶
v_p1 := p1;¶
n_p2 := 20 - p2; -- count out the remaining token amount¶
¶
if v_t1 and v_p1 >= 4 then -- highest priority¶
>  v_p1 := v_p1 - 4;¶
end if;¶
v_t2 := v_t2 and v_p1 >= 3;¶
¶
if v_t3 and n_p2 >= 4 then¶
>  n_p2 := n_p2 - 4;¶
end if;¶
if v_t4 and n_p2 >= 15 then¶
>  n_p2 := n_p2 - 15;¶
else¶
>  v_t4 := false;¶
end if;¶
v_t2 := v_t2 and n_p2 >= 5;§
```

Fig. 3 Removing of firing hazards

Firing hazards are removed by defining a firing priority of the transitions which are conflicting. Onl transitions with the highest priority will fire at the specified clockedge. Since one transition can take part in more firing hazards at one time, the removing of the firing hazards must be done using VHDL variables (Fig. 3).The functionality of solving firing hazards is implemented as a VHDL sequential process.




```
token flow:¶
¶
if rst = '0' then¶
>  p1 <= 0;¶
>  p2 <= 0;¶
>  p3 <= 0;¶
elsif clk'event and clk = '1' then¶
>  p1 <= p1 + getm(t1,-1) + getm(t2,1);¶
>  p2 <= p2 + getm(t1,1) + getm(t2,-1);¶
>  p3 <= p3 + getm(t1,1);¶
end if;§
```

Fig. 4 Token flow with active clock edge

The token flow is controlled by another VHDL process. It consists of the detection of the reset phase and the active clock edge. During reset the marking of the place elements is set to their start value. If the active clock edge is detected the token flow will takes place (Fig. 4). The user defined function 'getm' returns the token amount which will relatively change the marking of the related place if the corresponding transition fires.

Using I/O functions the optimal scheduling must be done in advance. Since scheduling is not supported by Petri nets, it must be done manually by synthesizing examples which correspond to the functionality of the design. The synthesis results in the number of multicycles or the scheduling plan for the corresponding arithmetic operation. Scheduling can be done using formal analysis methods as well - synthesizing test examples does relate to the target technology and includes the timing of the wiring and the total area needed for the gatelevel design.

Since adding or substracting of tokens is an arithmetic operation as well, the clock period must be set to a value, where the marking of the places can be calculated at once. Using multicycles is not possible at this point. For time critical designs, the design can be turned to a finite state machine and FSM optimization can be run. This will remove all arithmetic operations from the token flow process. The design bases on changes, which are done to the state vector of the design. The state vector is built out of the marking of all places in the design. FSM optimization is supported by the synthesis tool. The reachable markings should be known to allow an extended optimization of the state vector and to save the amount of the state registers.

Allocation, structuring, flattening and mapping to the target technology is done by the synthesis tool i.e. DESIGN_ANALYZER (Synopsys, Inc.) or AUTOLOGIC (Mentor Graphics, Inc.) since the VHDL code generated out of a Hardware Petri Net uses full synthesizable RTL (Register Transfer Level) VHDL subset. Behavioural synthesis is not necessary.

5 Unfolding Coloured Petri Net Designs to HPN

Using ordinary Petri nets for modeling of large designs the resulting nets often seem to be badly arranged and confused. But typically, they frequently contain regular subnet structures. Using coloured Petri nets [6] simplifies the net structure by enfolding, i.e. information of the net structure will be transfered into the description of the net elements. The entire modeling ability will be kept but the handling and flexibility will be improved.

For simulating and generating the VHDL code the coloured HPN has to be unfolded, i.e. the information contained in the description of the net elements will be transfered back into a net structure without any loss of information. There are existing tools [10] including unfoulding rules to automate these processes.

6 Design Examples

6.1 A Motor Superviso

This device controls a motor [4]. The motor is moved to a specific angle which is defined by the user. The motor will move in that direction which is shorter to reach the specific angle. The motor receives pulses, where each pulse means a movement of 1 degree. The new angle is defined by PHI and will be set using the signal LOAD (Fig. 5). The supervisor stores the old angle in the variable OLDPHI and counts out the difference (DIFF) and the direction DIR. Once these values are counted out, the supervisor sends the corresponding pulses toward the motor.

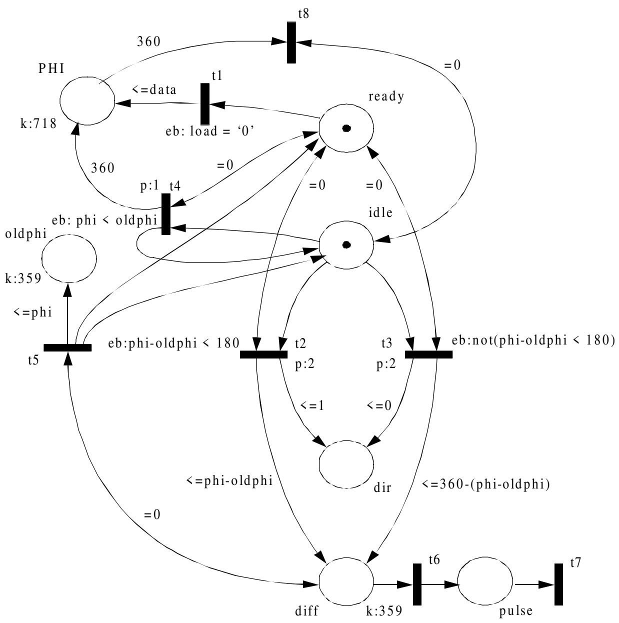


Fig. 5 Motor Supervisor - Petri net model

The functional simulation shows motions from 0 to 5, from 5 to 350, from 350 to 349, and from 349 to 0 degrees (Fig. 6).

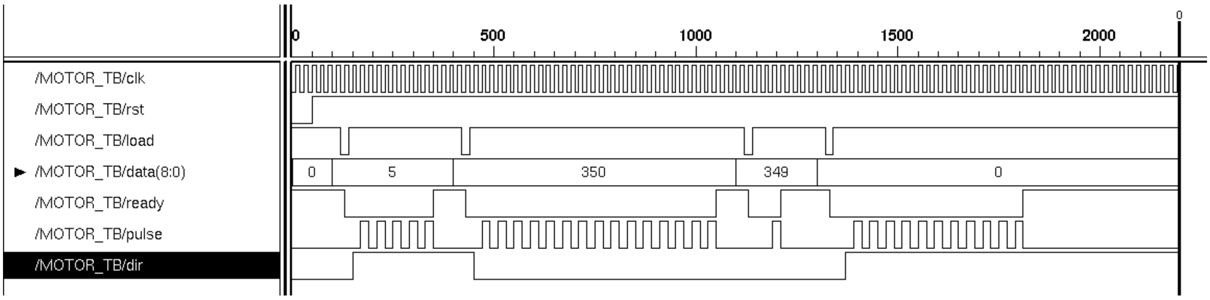


Fig. 6 Motor Supervisor - Functional Simulation

The synthesis results are constraint by area since the timing is not the point of interest.
Using the lca300kv target library (LSI Logic), following results are achieved:

15 ports, 393 nets and 293 cells.

The synthesized area amounts to:

1813 (total), 1304 (cells) and 509 (nets).

6.2 A Simple ATM Cell Rate Police

Using STM (synchronous transfer mode) a channel with fixed transmission rate is reserved for each connection. ATM (asynchronous transfer mode) allows connections with variable transfer rate. This rate can be defined b the user and should not be crossed at any time. To detect any violations a policer is necessary to watch over the actual rate of an ATM connection. If a violation is detected, extra fee has to be paid by the user or the connection will be refused. Because some devices can cause a higher cell rate than allowed (multiplexers) a temporary crossing of the specific rate should be allowed.

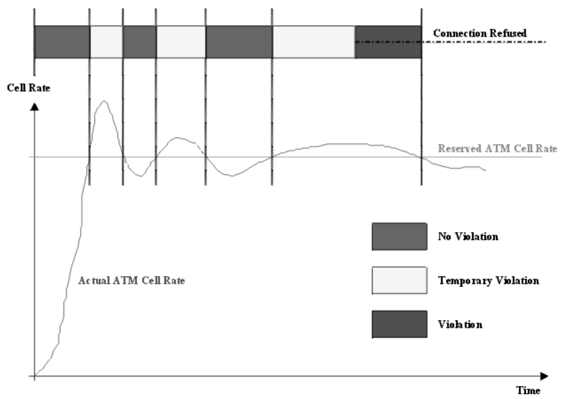


Fig. 7 Simple ATM Cell Policer - The Algorithm

The ATM cell rate policer uses the *leaky bucket* algorithm [2]. It consists of two parallel working parts - the decrement counter (ZT) and the cell counter (Z).
Each time an ATM cell is received (ATM) the cell counter increments. If the number of cells exceed the allowed limit (S) the acknowledge of the incoming cell will be denied (ACK). The decrement counte decrements the cell counter each T period. The decrement value is specified by D.
The period and the decrement value specifies the time where crossing the reserved cell rate is allowed.

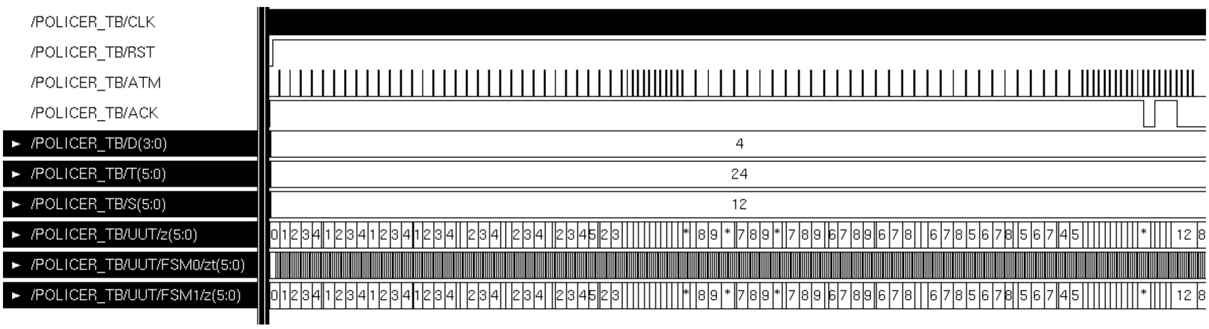
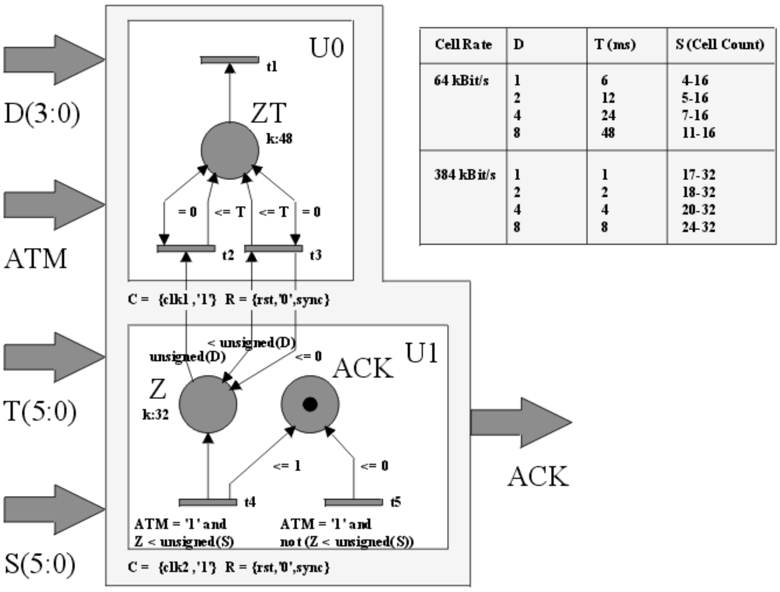


Fig. 8 Simple ATM Cell Rate Policer - The Simulation

The synthesis results have been done using the area optimization for the decrement counter and the time optimization for the cell counter.



Cell Rate	D	T (ms)	S (Cell Count)
64 kBit/s	1 2 4 8	6 12 24 48	4-16 5-16 7-16 11-16
384 kBit/s	1 2 4 8	1 2 4 8	17-32 18-32 20-32 24-32

Fig. 9 Simple ATM Cell Rate Policer - The HPN Model

The ATM cell rate policer is necessary for each ATM channel. Using a coloured Petri net the entire functionality can be flexible captured in the same Petri net as shown in this example.

7 Summary and Conclusions

Using Petri nets to specify and model a digital system is an improvement in comparison to other ESDA capturing methods e.g. state machines. Using the analysing methods and the capability to describe parallel tasks, a powerful method is found to detect a large number of design errors prior to system implementation.

Since the VHDL generation out of a Hardware Petri Net is possible, a synthesizable design (e.g. FPGA o ASIC) can be captured at a higher abstraction level than VHDL. The creation of an executable specification even for large systems is possible as well.

To enlarge the flexibility of Petri nets in a hardware design process, the system should be captured as a coloured HPN and unfolded to a HPN before generating the VHDL code. Both steps unfolding a CHPN and the VHDL code generation can be done automatically.

References

1. Carlson, S.: Introduction To HDL-Based Design Using VHDL, Synopsys Inc., 1991
2. G. Daisenberger, J. Oehlerich, G. Wegmann: Two concepts for Overload Regulation in SPC Switching Systems STATOR and TAIL, Telecommunication Journal, vol. 56, V/1989
3. Visual HDL for VHDL on UNIX - Edition 4.0, Summit Design Inc., 1997
4. W. Fengler, A. Karg: Design of Complex Embedded Systems Based on Different Petri-Net Interpretations, 1998 Advanced Simulation Technologies Conference, Boston, MA, 5.-9.4. 1998
5. IEEE Standard VHDL Language Reference Manual - IEEE Std 1076-1993, The Institute of Electrical and Electronics Engineers Inc., 1994
6. Jensen, K.: Coloured Petri Nets: Basis Concepts, Analysis Methods and Practical Use, Springer Verlag, Berlin, Heidelberg, New York, 1992
7. Perry, D. L.: VHDL, McGraw-Hill Inc., 1991
8. Peterson, J. L.: Petri Net Theory and the Modelling of Systems, Prentice-Hall Inc., 1981
9. The System Specs VHDL Generator - beta2-Version, Data Sheet, Ivy Team, 1994
10. Wikarski, D. (Ed.): Petri Net Tools: a Comparative Study, ISST-Bericht 39/96 of the Fraunhofer-Gesellschaft e.G., Berlin, 1996