

# Petri Net Based Analysis of VHDL Behavioral Models

Neal S. Stollon P.E.  
Texas Instruments  
Defense Systems and Electronics Group  
Dallas, Texas

## 1: Introduction

The management of VHDL model complexity and related evaluation of VHDL model parameters at the behavioral level is of interest as a mechanism in system performance analysis. This paper addresses the use of Petri Net models and applications for performance analysis of behavioral level VHDL designs.

Performance analysis relies on a class of model views that simplify through the abstraction of selectively (un)interpreting model information [1]. For VHDL, one uninterpreted model view can be defined by analyzing only the control structure information in the model architecture and where most computational and datapath information is not utilized. The uninterpreted model retains the necessary timing and sequential information needed to analyze the system performance and complexity of the design with significantly reduced computational overhead. One of the primary mechanisms for expressing uninterpreted models is through the formalism of Petri Net theory.

This paper expands on previous work on performance analysis [2] for structured programs by addressing Petri Net modeling and the statistical structures associated with VHDL statements used in RTL and behavioral modeling. The creation of Petri Net representations from VHDL models appears to be a useful and appropriate mechanism for analyzing attributes of VHDL models such as information paths and complexity of model behavior. Petri Nets have been used in the evaluation of design complexity measures of related modeling areas [3, 7]. Petri Nets (PN) and VHDL share several characteristics, including data concurrency and hierarchy, that facilitate modeling of design complexity and information.

The paper is organized into two parts. First, it is shown how a VHDL model's statements can be represented using a PN. Second, two applications of a Petri Net representation of a VHDL model are discussed. A complexity measure for a VHDL model based on the reachability concept of Petri Nets is explored through a method of determining simple data paths and loops in the PN representation; This measure provides information that can be used for verifying bounds of a behavioral VHDL model's testability. A method of using the probabilistic delay information of a Timed Petri Net is then demonstrated as an approach to estimating critical path timing of a behavioral model.

## 2: Petri Net representation of a VHDL model

A Petri Net is an abstract, formal model of information flow and control. Petri Nets were first introduced in [4] and related to electronics modeling in [5,7]. A PN is structured as a bipartite directed graph which contains two types of nodes called places and transitions. The input and output places of a transition are connected by incoming and outgoing arcs of the transition. Each place on a PN is assigned some (non-negative) number of tokens which represent data items or hold some initial condition information for the places in the PN. The distribution of tokens in the places is called the state or marking of a PN. A transition is enabled to fire (or pass token information) only if each of its input places contains at least one token.

On firing, a transition removes one token from each of its input places and puts a token on each of its output places, changing the PN marking. Transition firing typically corresponds to execution of a computation or occurrence of a sequential event in the VHDL model and firing occurs only when certain model conditions (tokens on all input places) are satisfied. In modeling a VHDL behavior, each condition can be represented by a place and each statement by a transition. After firing a transition (ie. a sequential event has occurred), the updated state of the PN reflects the changed conditions in the model. This new state corresponds to activation of a new set of model statements that can be executed.

Petri Net models can be created for any arbitrary level of design complexity and model accuracy and detail. For the discussion that follows, the PN model has been simplified to represent only the information required for performance modeling of the control information flow. Fig. 1 illustrates a PN representing VHDL model (shown in 2.5 to consist of parallel wait and case processes).

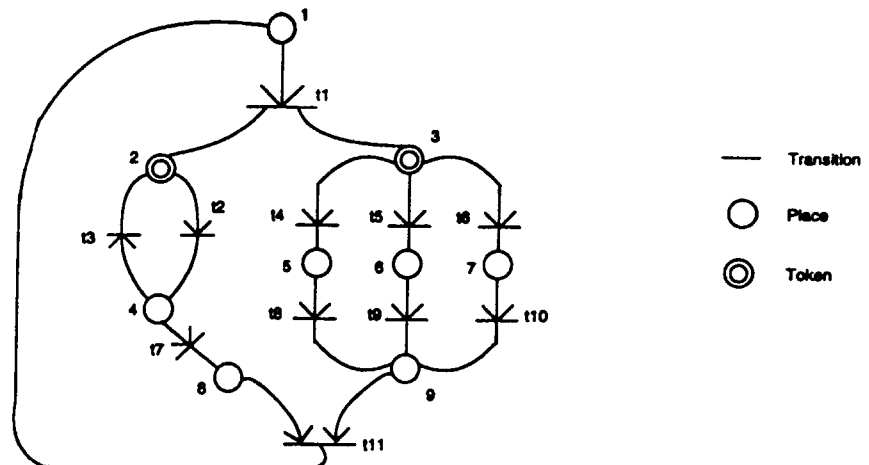


Fig 1: Example of a Petri Net

In conventional PN theory, it is assumed that the updating of the state after transition firing is immediate and that the firing has no time delay. To more accurately correspond to VHDL model timing,

an execution delay of an operation can be associated with each transition. A PN with an associated transition time is known as a Timed Petri Net (TPN) [6]. A Timed Petri Net can be formalized as a 5-tuple:

$$TPN:=(P, T, A, Mo, Td)$$

where P is a set of places  
 T is a set of transitions  
 A is a set of directed arcs  
 Mo is an initial distribution of tokens on each place  
 Td is a time delay associated with each transition

In VHDL models, many sequential operations are timed by clocks. Clocks can be represented in PN representations as output places generating tokens on a periodic basis to synchronous transitions.

Petri Nets of interest in behavioral modeling require a decision property that is implied by multiple paths. This characteristic is typical of sequential operations in VHDL. Concurrent assignment operations can be modeled using Petri Nets, however they are often trivial in application, consisting of only a single output place and transition, with execution delay equal to the number of times the operation is executed. For the PN model assumed in this paper, concurrent assignment operations are ignored unless they impact in some way the VHDL sequential operations in the behavioral model.

Four basic sequential operations of a structured VHDL model and their Petri Net representations are discussed: IF\_THEN\_ELSE, LOOP, WAIT, and CASE statements. These operations are chosen since they correspond to a core VHDL modeling subset used in logic synthesis. PN models can be created for other VHDL operations similarly.

### 2.1: IF\_THEN\_ELSE Operation

A block representation of a IF-THEN-ELSE statement of the form:

```
IF (n) THEN (n1); ELSE (n2);
```

is given in Fig 2A. A PN representation of this block is shown in fig 2B. Execution times can be assigned to operations in this block (conditional operation n, statements n1 and n2) associated with the transitions (ie.n:Tn, n1:Tn1, n2:tn2) through them. The execution time of the conditional operation can be expanded to Tnt and Tnf for, respectively, test of IF true and false conditions. Total execution time for the model sequence is either (Tn+Tn1) or (Tn+Tn2) according to the data at place 1. In the PN, a token at place 1 will reach place 4 either through the firing of transitions t1 and t3, or t2 and t4, depending on the data value of the conditional operation at place 1. Firing of the transitions t1 and t3 indicate a true condition, while firing of t2 and t4 indicates that the condition is false. The execution times associated with these transition paths can be given by:

$$K(Tnt+Tn1) + (1-K)(Tnf+Tn2) \quad (1)$$

where  $K$  is proportional to the probability of execution of the condition  $IF=true$ , ie. transitions  $t1$  and  $t3$  will fire  $K\%$  of the time.

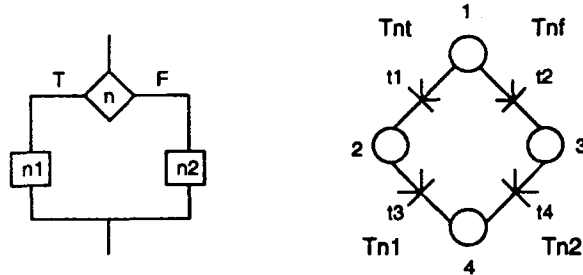


Fig 2: IF\_THEN\_ELSE (A) block (B) PN

This model can be extended for IF-THEN-ELSIF-THEN-ELSE statements in VHDL by nesting of the PN representation in Fig. 2B.

### 2.2: LOOP/WHILE Operation

A block representation of a LOOP/WHILE statement of the form:

```
WHILE (mi < m) LOOP (mi); END LOOP;
```

is given in Fig 3A. Its PN representation is shown in fig 3B. Execution times of the conditional statement associated with transitions  $t1$ ,  $t2$ , and  $t3$  are  $Tmt$ ,  $Tm$ , and  $Tmf$ , respectively. PN firing characteristics allow the token at place 1 to go to either place 2 or 3 depending on the loop count value. If the token moves to place 2 (and back to place 1) after firing of transitions  $t1$  (and  $t3$ ) then the loop is active (condition true); if the token moves to place 3 after firing of transition  $t2$ ; the loop is broken (condition false). The PN for a FOR/LOOP statement can be created similarly.

The execution time associated with CASE transition paths are given by:

$$m(Tmt+Tm)+Tmf \tag{2}$$

where  $m$  is the loop count; ie. transitions  $t1$  and  $t3$  will fire  $m$  number of times.

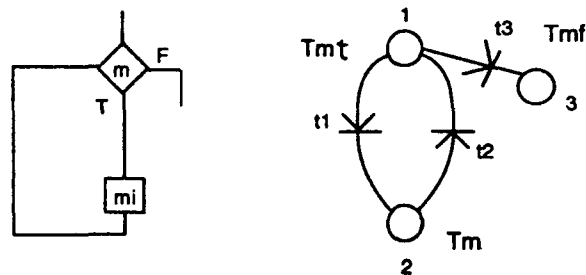


Fig 3: WHILE/LOOP (A) block (B) PN

### 2.3: WAIT Operations

Block and PN representations of a WAIT statement with sensitivity clause (x) of the form:

```
out1 <= x AFTER Tpl;   WAIT UNTIL x;
```

are given in Fig 4A and 4B. Execution times of conditional operation associated with transitions t1, t2, and t3 are Tpl, Tpt, and Tpf; where Tpt and Tpf represent true/false transition times of the wait operation. Execution time of the wait transitions can be modeled as:

$$p(Tpf) + (1-p)Tpl + Tpt \quad (3)$$

where p is the number of times the sensitivity condition is tested false, ie. the number of times that transitions t1 and t3 will fire. A PN for a WAIT ON statement can be generated similarly.

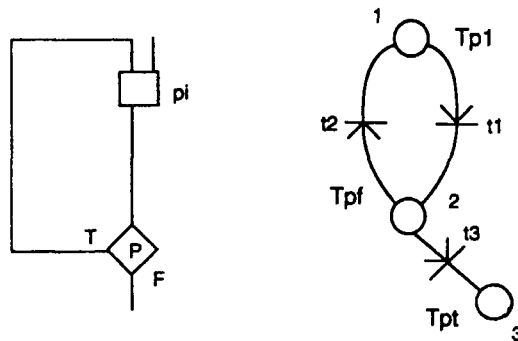


Fig 4: WAIT (A) block (B) PN

### 2.4: CASE Operations

A CASE PN representation of a CASE statement of the type

```
CASE xx IS
WHEN c1 => V1 <= s(1);
...
WHEN cn => Vn <= s(n);
END CASE;
```

is given in Fig. 5 (for case of 3). CASE statement data flow can be modeled as an extended IF\_THEN\_ELSE statement, with a PN that is structured as equivalent to an extended IF\_THEN\_ELSE block.

For a CASE statement consisting of n alternative tests, execution times Tci are associated with the (pre-place) transitions ti which represent transition times of the ith CASE conditional operation. Execution times associated with (post-place) statement transition paths are modeled by Tsi. Overall execution time model of the CASE statement is given as:

$$\sum_i^n c_i (T_{ci} + T_{si}) \quad \text{s.t.} \quad \sum c_i = 1 \quad (4)$$

where  $c_i$  is probability of conditional execution thru test path  $i$

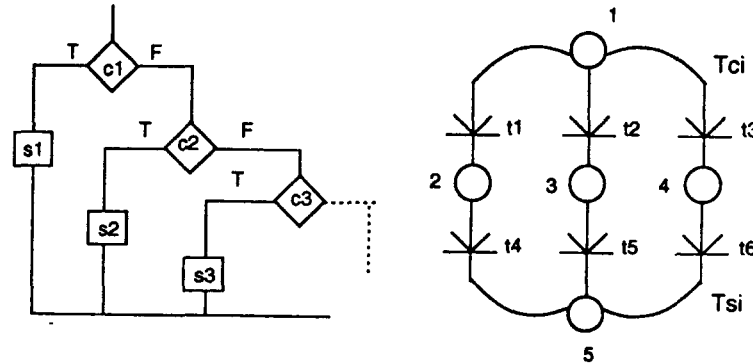


Fig 5: CASE (A) block (B) PN

PN models can provide useful insights. The WHILE/LOOP PN structure provides essentially the same loop capability as a WAIT structure, but they differ in respect of their placement of test data. In a WAIT statement, the test is performed after the firing of the transition  $t_1$ . In the WHILE/LOOP, the test is performed before the firing of transition  $t_1$ . Given the nature of these statements as duals in VHDL applications, a symmetric relationship would intuitively be expected.

### 2.5: A Petri Net example

Using the techniques discussed, the PN model in Fig. 1 is seen to represent a VHDL model fragment made up of 2 concurrent behavioral processes;

```

left_side: PROCESS
BEGIN
  WAIT UNTIL n0;  -- transitions t2,t3,t7
END PROCESS;
right_side: PROCESS
BEGIN
  CASE right_side IS
  WHEN P5 => P9  <= n1  -- transitions t4,t8
  WHEN P6 => P9  <= n2  -- transitions t5,t9
  WHEN P7 => P9  <= n3  -- transitions t6,t10
  END CASE;
END PROCESS;

```

where  $n(i)$  represent computational or datapath behavior occurring outside the fragment. The diverging transition  $t_1$  and reconverging transition  $t_{11}$  model the parallel processes. Sequential elements in a PN typically map to a single process. A single process VHDL model and its PN representation is given in Fig. 6.

```

-- VHDL model of Petri Net
-- representation in Fig. 6
PROCESS
BEGIN
  LM1: WHILE (nx) LOOP
    -- transitions t1-t3
    IF (n0) THEN n1;
    -- transitions t5,t7
    ELSE n2;
    -- transitions t6,t8
    END IF;
    WAIT UNTIL n3;
    -- transitions t10-t12
  END LOOP;
END PROCESS;

```

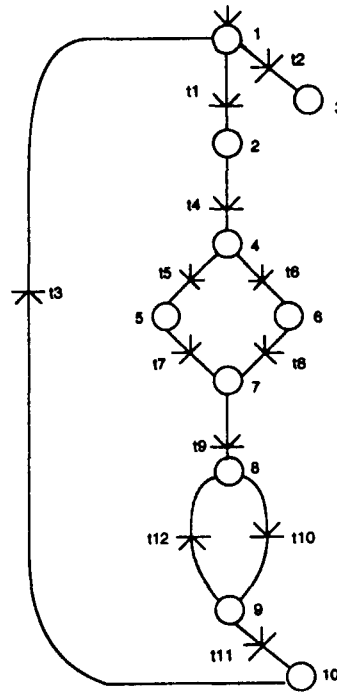


FIG. 6: PN of a single process VHDL model

### 3: Applications of Petri Net representations of a VHDL model

The previous discussion describes how VHDL model statements are represented using Petri Nets. The Petri Net models can in turn be used to analyze complexity of information flow and timing related estimates of the behavioral model.

#### 3.1: Reachability based complexity analysis of a VHDL model

Recognized approaches to modeling HDL complexity rely on analysis of the structure of the model and derivation of a directed graph representation of the model information flow. Related research into structured programming complexity has used cyclomatic number, number of paths, and reachability as measures of complexity [3].

Reachability is a measure of PN models that dictates whether a given state (marking) can be reached from another state by multiple firing of the PN. By assigning markings to entry and exit points of a model, the reachability concept of PNs can be used to extract the number of information paths in the model. Research in complexity measures has shown applicable results for reliability analysis and in generating testability strategies for analysis of structured models. Reachability analysis is a computationally hard (possibly NP-complete) problem; however there exist straightforward techniques requiring only column vector additions on a PN model's incidence matrix that may be utilized (with only moderate computer resources) in determining path information of models.

In order to find all paths between entry and exit of a sequential VHDL model, it is sufficient to find which transitions of the corresponding PN would fire such that a marking which has the only

token at the exit place is reachable from a marking which has the only token at the entry place.

Reachability can be determined by solving the PN state equation:

$$\text{sgn} (\bar{C} \times \bar{F}) = \bar{M}d \quad (5)$$

where: C is a place to transition (P,T) incidence matrix that captures the PN connectivity structure such that

$$\bar{C} = \begin{matrix} (C(p,t)) & \begin{matrix} -1, & \text{if } (p,t) \text{ connects input place and transition} \\ +1, & \text{if } (p,t) \text{ connects output place and transition} \\ 0, & \text{no place to transition connectivity} \end{matrix} \end{matrix}$$

$p \in P$   
 $t \in T$

$\bar{M}d$  is a (p) column vector of the difference between the entry (initial) and exit (final) markings; s.t.  $Md(p) = 1$  for entry and  $Md(p) = -1$  for exit places; (0 for all others)

$\bar{F}$  is a column vector of the firing count list of transitions included in a path of the PN

The incidence matrices for the PNs from Figs. 1 and 6 are shown in Fig. 7 (A) and (B) respectively.

$C(9,11) = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & -1 & 0 \end{bmatrix}$	$C(10,12) = \begin{bmatrix} -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$
---	---

(A) : Fig. 1 PN

(B) : Fig. 6 PN

Fig. 7: Incidence Matrices of Petri Nets

Enumeration through the state space of transitions which satisfy (5) allow the simple paths that represent information flow through the Petri Net to be determined and analyzed.

For  $\bar{M}d = [-1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1]$  in the PN in Fig. 6 (simple paths between places 1 and 10), the following paths of fired transitions satisfy the reachability equation:

$$(t1 \ t4 \ t5 \ t7 \ t9 \ t10 \ t11) \text{ and } (T1 \ t4 \ t6 \ t8 \ t9 \ t10 \ t11)$$

Similarly, for the PN in Fig. 1, three simple paths between places 1 and 9 can be determined; (t1 t4 t8), (t1 t5 t9), (t1 t6 t10). A single path between places 1 and 8 is given by (t1 t2 t7). The two sets of simple paths represent data path options in the concurrent information flow of the two VHDL processes modeled by the PN.

Testability and reliability (redundancy) analysis of behavioral models with numerous concurrent processes is a problem addressable through PN application. Path information of the PN can be used to generate testability (coverage) measures of VHDL behavioral models by comparison of generated test vector sets for process blocks to simple paths to determine test coverage and overlap. Similarly, a reliability measure of a VHDL model can be evaluated by examining information paths of the PN for deadlocks and redundancy.

### 3.2: Petri Net timing analysis of a VHDL model

A probabilistic execution time that models an asynchronous cyclic delay of a behavioral model can be analyzed from nested transition structures in PNs. Execution delays can be assigned to transitions connected with each conditional place and those of the transitions connected to each statement. These delay models can be equated to inertial and delta delay times used in VHDL. If execution delays for each statement is known, then model execution time estimates can be calculated from cyclic probabilities of the IF\_THEN\_ELSE, WAIT, CASE and WHILE/LOOP operations.

As an example, assume execution times for the VHDL and PN model in in Fig. 6; delay assigned to transition connections to conditional places are 10 ns and of transition to each statement as 5 ns.

Based on a priori information on the model application, we further assume that:

1. WHILE/LOOP executes through its true condition 4 times, ie  $m=4$ ;
2. IF-THEN-ELSE block executes 10 times and yields a true condition 80 of the times it is executed, ie.  $K=.8$ ;
3. WAIT loop executes to give a false condition 6 times, ie.  $p=6$

Places 1, 4 and 8 correspond to the conditional operations and the transitions connected to them (ie.  $t_1$ ,  $t_5$ ,  $t_6$ , and  $t_{10}$ ) represent the execution of these operations.

In order to estimate total execution time of the model, the nested conditional blocks (IF/THEN/ELSE and WAIT) are replaced with their timing equation equivalents (equations (1) and (3) respectively):

$$\text{IF/THEN/ELSE: } K(T_{nt}+T_{n1})+(1-K)(T_{nf}+T_{n2}) = .8(10+5)+.2(10+5) = 15 \text{ ns}$$

$$\text{WAIT: } p(T_{pf})+(1-p)T_{p1}+T_{pt} = 6(10)+(6+1)5+10 = 105 \text{ ns}$$

The execution time associated with the true condition path of the WHILE/LOOP (between places 1 and 10) is the delayed transitions:

$$t_1+t_4+(\text{IF/THEN/ELSE})+t_9+(\text{WAIT}) = 10+5+15+10+105 = 145 \text{ ns}$$

Total execution time of this PN equivalent of the VHDL model can be calculated from the WHILE/LOOP operation by (2):

$$m(T_{mt}+T_{m1})+T_{mf} = 4(145+5)+10 = 610 \text{ ns}$$

Similarly, the execution time for the PN in Fig. 1 is calculated as the maximum delay path through its concurrent processes. Since both processes must pass tokens to t11 for it to fire and complete the cycle, the execution delay of the model is that of the longest process.

#### 4: Summary

This paper discussed an approach for applying Petri Net analysis techniques to VHDL behavioral models. PN representations provide a useful model view that appears to be appropriate to relevant areas of VHDL analysis. Petri Net analysis has been under investigation for over two decades; several potential application areas appear appropriate to automation and analysis of HDL based designs.

The use of Petri Nets in performance modeling of digital designs has been addressed, but has only rarely been applied to hardware design. Some Petri Net techniques (such as reachability analysis, which offer insights into the complexity and performance of VHDL behavior) are known as exponentially time and space-hard (although straightforward) computational tasks. In many cases however, Petri Nets can provide simplified insights into behavioral designs which are not readily available from simulation.

#### References:

- [1] Ayler J., Waxman R., et al "The Integration of Performance and Functional Modeling in VHDL"; chap. 2 in Performance and Fault Modeling with VHDL; J. Schoen ed. 1992
- [2] Mekly L., Yao S. "Software Design Representation Using Abstract Process Networks" IEEE Trans. Software Eng., Sept. 1980
- [3] Hura G., Singh H., Nanda N. "Petri Net approach to evaluation of the complexity of a program" Int. Journal of Electronics: vol. 51 #1, 1981
- [4] Petri C. "Communication with Automata" translated in RADC Tech Report TR-65-377, 1966
- [5] Agerwala T. "Putting Petri Nets to Work" Computers, Dec. 1979
- [6] Holiday P., Vernon M. "A Generalized Timed Petri Net model for Performance Analysis" IEEE Trans Software Eng., Dec. 1987
- [7] Murata T. "Petri Nets: Properties, Analysis and Applications" Proceedings of the IEEE, April 1989