Using Petri nets for RT level digital systems test scheduling

Jaroslav Škarvada*
skarvada@fit.vutbr.cz

Richard Růžička
ruzicka@fit.vutbr.cz

Abstract: This paper deals with test scheduling for digital systems. Approach with C/E Petri nets is presented and formal model of digital system under test is introduced. Main purpose of this model is identification of structural conflicts and dead locks that may occur during test application phase. The digital system is analyzed on register transfer (RT) level. The obtained results can be used for digital system design partitioning. In this step individual blocks of logic are identified. Finally concurrent test for non-conflicting blocks of logic is scheduled. The advantage of this approach is, that with partitioned circuit, it is possible to view digital circuit design as system on chip (SOC) design and use existing test scheduling methods for SOC.

Key Words: Digital circuit, C/E Petri Net, test scheduling, I-paths, structural conflicts

1 Introduction

Integrated circuits continue to grow in size and complexity and even with the state of the art technology it is impossible to eliminate probability of defects during manufacturing process. As market demands circuits without manufacturing defects, every manufactured chip must be tested before packaging and defective chips must be discarded. If manufacturers want to achieve effective and defect free production, considerations about testing must be taken into acount in early stage of the design process. Such approach is called Design-for-Testability (DfT). It was developed a methodology for testability analysis of register transfer (RT) level digital circuits at the department [1]. This methodology could be successfully applied in the early stage of the design process. Apparently, the effort is to make all DfT procedures automated. The cost of the test and duration of the test are main tracked parameters of these methodologies. The cost of the test cover cost of the external automated test equipment (ATE) as well as cost of additional logic that must be added to design for its successful testability [5]. The duration of the test is important parameter for production rate. Lower test duration can be achieved by parallel testing of components. Due to structural conflicts, it is practically impossible to test all components in parallel. So in this case test scheduling takes place.

The structure of a digital circuit on the RT level of modeling consists of two main parts: data paths and the circuit controller. Data path part consists of components interconnected through connections. Components on the RT-level data paths are functional units, which perform data transformation, registers are used to store the state of the circuit and multiplexers establish connections among elements. An RT-level circuit synthesis based on a multiplexed data path strategy [4] is expected to be used and the presented approach does not take into account bi-directional buses and ports. The circuit controller is a kind of state machine implementation. For test purposes, a special controller, which enables diagnostic data transfer paths in accordance with a test schedule replaces the original controller (which controls the circuit to perform the proposed function). Test scheduling generally proceeds in three steps. First step is

* FIT BUT, Department of Computer Systems, Božetěchova 2, CZ-612 66 Brno

identification of resources that are necessary for test of every component. Second step is identification of components that are in conflict during test application. Components that are in conflict couldn't be tested simultaneously. Finally test sequence is determined. For first and second steps, it is possible to exploit formal model based on C/E Petri nets. It is especially suitable in cases, when testability verification [3] takes place before test scheduling. The advantage is that Petri nets models that were automatically generated by software [7], [8] in verification phase can be reused for test scheduling. The main idea behind test scheduling on RT level is that components that are close enough could be grouped together to individual blocks and non-conflicting blocks could be then tested simultaneously. The partitioning of circuit allows us to use common test scheduling methods from SOC world.

2 Circuit Model

Within previous works there was implemented software [7] in the department for automatic transformation of structural VHDL code into this formal model. The formal model was based on approach from [11]. Every analysis and Petri net transformations uses this model. In this chapter, the basic definitions that are necessary for understanding the transformation procedure presented in next chapter, are described.

2.1 Unit under Analysis

For the purposes of the formal approach to the RT-level testability analysis, a Unit Under Analysis (UUA) is described as a 5-tuple $UUA=(E, P, C, PI, PO)$, where: E is the set of circuit elements, P is the set of ports of all circuit elements from E, C is a set of connections, (it is a relation between pairs of ports), PI is the set of primary input ports (ports wired to package pins) of the UUA and PO is the set of primary output ports of the UUA.

2.2 Circuit Elements

Set $E = (R \cup SR \cup MX \cup DMX \cup DP)$ is set of all circuit elements, where: R is the set of all registers in the circuit, SR is set of all scan registers in the circuit, MX is set of all multiplexers in the circuit, DMX is set of all demultiplexers in the circuit, DP is set of all functional units in the circuit (Data Processors). $TE \subseteq DP$ is set of all functional units that will be tested.

2.3 Circuit Gates

Set $P = (IN \cup OUT \cup CI)$ is set of all ports of circuit elements, where: IN is set of all input ports (excluding primary), OUT is set of all output ports (excluding primary) and CI is set of all synchronizing an controlling gates.

2.4 Connections in the Circuit

Relation C is used for modeling all metallic connections in the circuit. $C \subseteq (PI \cup PO \cup P) \times (PI \cup PO \cup P)$. Relation C is transitive, reflexive and symmetric; it can be traced from properties of metallic connection.

2.5 I-paths in the Circuit

Relation I is used for modeling all I-paths [6] in the circuit $I \subseteq (PI \cup PO \cup P) \times (PI \cup PO \cup P)$. $C \subseteq I$. I-path is a virtual path from one port to another that can be used for data transmission. On I-path can be located one or more circuit elements, but only those elements, that have I-mode of operation (in I-mode data can be transmitted

from input to output without change, for example adder has I-mode when one input is set to zero). Relation I is transitive, reflexive. Relation is not symmetrical, because most of DP elements allow only one way data transmission.

2.6 Circuit Nodes

Set $V \subset (P \cup PI \cup PO)$ is called node, if $\forall\ p_1, p_2 \in V$: $(p_1, p_2) \in C$.

2.7 Determining Input Gates of Given Element – in(e), $e \in E$

$in\colon\ E \rightarrow 2^{IN}$

1) For $\forall e \in E$: $in(e) = \{\, p \mid p \in IN \wedge p$ is gate of element $e\}$

2) For $\forall e_1, e_2 \in E$: $e_1 = e_2 \Leftrightarrow in(e_1) \cap in(e_2) \neq \varnothing$ (elements can't share gates)

2.8 Determining Output Gates of Given Element – out(e), $e \in E$

$out\colon\ E \rightarrow 2^{OUT}$

1) For $\forall e \in E$: $out(e) = \{\, p \mid p \in OUT \wedge p$ is gate of element $e\}$

2) For $\forall e_1, e_2 \in E$: $e_1 = e_2 \Leftrightarrow out(e_1) \cap out(e_2) \neq \varnothing$ (elements can't share gates)

3 UUA to Petri Net Transformation

3.1 C/E Petri Net

C/E Petriho net [2] $N_{fu} = (B, \Sigma, F, M_0)$ is 4-tuple that represents test application process on circuit element $fu \in TE$ from UUA, where: set B is set that corresponds to all circuit nodes and it may be extended with some special nodes ($(P \cup PI \cup PO) \subseteq B$). Set Σ is set of all transitions that correspond to elements participating in transmission of diagnostic information (elements that have data dependant I mode). F is flow relation that describes topology of net $F \subseteq (B \times \Sigma) \cup (\Sigma \times B)$. M_0 is initial marking of net: $M_0 \subseteq B$.

3.2 Transforming Function vid(i), $i \in P \cup E$

$vid\colon\ P \cup E \rightarrow B$ Assigns circuit gates and elements to corresponding net places.

The next 3 rules must hold:

1) For $\forall e_1, e_2 \in E$: $vid(e_1) = vid(e_2) \Leftrightarrow e_1 = e_2$

2) For $\forall p_1, p_2 \in P$: $vid(p_1) = vid(p_2) \Leftrightarrow (p_1, p_2) \in C \wedge (p_2, p_1) \in C$

3) For $\forall e \in E$, $\forall p \in P$: $vid(p) \neq vid(e)$

3.3 Transforming Function eid(e), $e \in E$

$eid\colon\ E \rightarrow \Sigma$ Assigns circuit elements to corresponding net transitions.

The next rule must hold:

1) For $\forall e_1, e_2 \in E$: $e_1 = e_2 \Leftrightarrow eid(e_1) = eid(e_2)$

3.4 Algorithm for Creation of New Element ID: neid

1) Find not so far used transition symbol e' such that $e' \notin \Sigma$

2) Return e'

3.5 Algorithms for Creation of New ID: nvid

1) Find not so far used place symbol $\boldsymbol{b'}$ such that $\boldsymbol{b' \notin B}$
2) Return $\boldsymbol{b'}$

3.6 Algorithm for DP Element e ∈ DP Transformation: dp(e, Nfu)

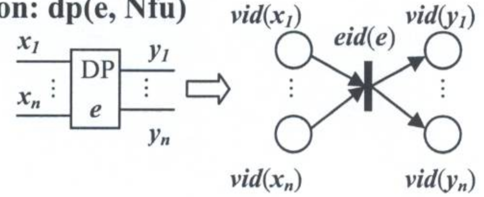

Fig. 3.1: *DP element transformation*

1) $\boldsymbol{\Sigma = \Sigma \cup \{eid(e)\}}$ Insert transition representing DP
2) For $\forall\, \boldsymbol{x}: \boldsymbol{x} \in \boldsymbol{in}(\boldsymbol{e})$: Creation of input

   $\boldsymbol{B = B \cup \{vid(x)\}, F = F \cup \{(vid(x), eid(e))\}}$
3) For $\forall\, \boldsymbol{y}: \boldsymbol{y} \in \boldsymbol{out}(\boldsymbol{e})$: Creation of output

   $\boldsymbol{B = B \cup \{vid(y)\}, F = F \cup \{(eid(e), vid(y))\}}$

3.7 Algorithm for Primary Input p∈PI Transformation: pi(p, Nfu)

1) $\boldsymbol{b = nvid(), e' = neid(), \Sigma = \Sigma \cup \{e'\}}$
2) $\boldsymbol{B = B \cup \{vid(p), b\}, M_0 = M_0 \cup \{vid(p)\}}$
3) $\boldsymbol{F = F \cup \{(vid(p), e'), (e', b)\}}$
4) For $\forall \boldsymbol{e}: \boldsymbol{e} \in \boldsymbol{E}$ if $\exists \boldsymbol{c}: \boldsymbol{c} \in \boldsymbol{C} \wedge \boldsymbol{c} = (\boldsymbol{p}, \boldsymbol{d}) \wedge \boldsymbol{d} \in \boldsymbol{in}(\boldsymbol{e})$

   $\boldsymbol{\Sigma = \Sigma \cup \{eid(e)\}}$,

   $\boldsymbol{F = F \cup \{b, eid(e)), (eid(e), vid(p))\}}$

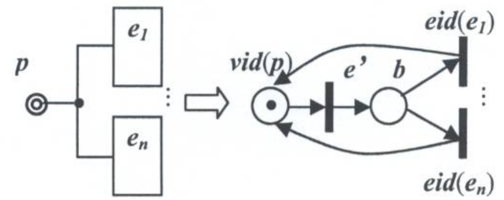

Fig. 3.2: *Primary input transformation*

3.8 Algorithm for Primary Output p∈ PO Transformation: po(p, Nfu)

1) $\boldsymbol{p' = neid()}$
2) $\boldsymbol{B = B \cup \{vid(p)\}, \Sigma = \Sigma \cup \{p'\}, F = F \cup \{(vid(p), p')\}}$

Fig. 3.3: *Primary output transformation*

3.9 Algorithm for Register r ∈ R Transformation: reg(r, Nfu)

1) $\boldsymbol{r' = neid(), r'' = neid(), x \in in(r), y \in out(r)}$
2) $\boldsymbol{B = B \cup \{vid(r)\} \cup \{vid(x)\} \cup \{vid(y)\}, \Sigma = \Sigma \cup \{r', r''\}}$

   $\boldsymbol{F = F \cup \{(vid(x), r'), (r', vid(r)), (vid(r), r''), (r'', vid(y))\}}$

Rem: $|\boldsymbol{in}(\boldsymbol{r})| = |\boldsymbol{out}(\boldsymbol{r})| = 1$, it's because dealing with register

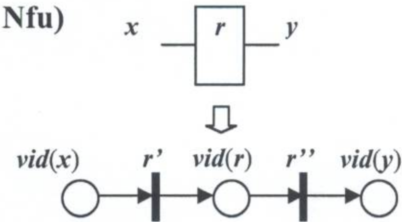

Fig. 3.4: *Registry transformation*

3.10 Algorithm for Scan Register r ∈ SR Transformation: sreg(r, lastscan, Nfu)

1) $\boldsymbol{r' = neid(), r'' = neid(), sco = neid(), x \in in(r), y \in out(r)}$
2) $\boldsymbol{B = B \cup \{vid(r)\} \cup \{vid(x)\} \cup \{vid(y)\}, \Sigma = \Sigma \cup \{r', r'', sco\}}$

   $\boldsymbol{F = F \cup \{(vid(x), r'), (r', vid(r)), (vid(r), r''), (r'', vid(y)),}$
   $\boldsymbol{(lastscan, vid(r)), (vid(r), sco)\}}$
3) $\boldsymbol{prevscan = lastscan}$
4) $\boldsymbol{lastscan = sco}$

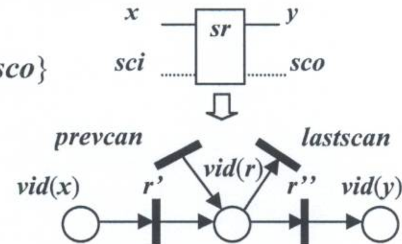

Fig. 3.5: *Scan register transformation*

3.11 Algorithm for Multiplexer m ∈ MX Transformation: mx(m, Nfu)

1) $\boldsymbol{y \in out(m)}$, $\boldsymbol{B = B \cup \{vid(y)\}}$ Only one output

2) For $\forall \boldsymbol{x}$: $\boldsymbol{x \in in(m)}$:

$\boldsymbol{mux = neid(), B = B \cup \{vid(x)\}, \Sigma = \Sigma \cup \{mux\}}$

$\boldsymbol{F = F \cup \{(vid(x), mux), (mux, vid(y))\}}$

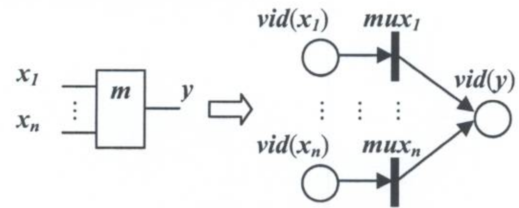

Fig. 3.6: *Multiplexer transformation*

3.12 Algorithm for Demultiplexer dm ∈ DMX Transformation: dmx(dm, Nfu)

1) $\boldsymbol{x \in in(dm)}$, $\boldsymbol{B = B \cup \{vid(x)\}}$ Only one output

2) For $\forall \boldsymbol{y}$: $\boldsymbol{y \in out(dm)}$:

$\boldsymbol{dmux = neid(), B = B \cup \{vid(y)\}, \Sigma = \Sigma \cup \{dmux\}}$

$\boldsymbol{F = F \cup \{(vid(x), dmux), (dmux, vid(y))\}}$

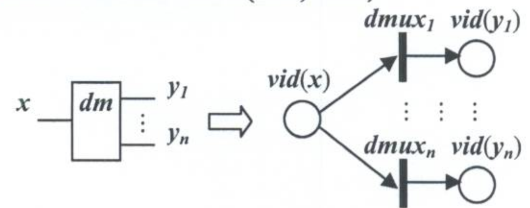

Fig. 3.7: *Demultiplexer transformation*

3.13 Transformation to Petri Net

Transformation to Petri net is recursive process. It can be informally described as follows:

Net(N_{fu}):

Select one so far not processed element $\boldsymbol{e}$, $\boldsymbol{e \in TE}$

For $\forall$ $\boldsymbol{p \in PI}$

$\boldsymbol{pi(p)}$

For $\forall$ $\boldsymbol{p \in PO}$

$\boldsymbol{po(p)}$

Element ($\boldsymbol{e}$, N_{fu})

Remove all isolated places from $\boldsymbol{B}$

Element ($\boldsymbol{e}$, N_{fu}):

If $\boldsymbol{e \in R}$, then $\boldsymbol{reg(e, N_{fu})}$

If $\boldsymbol{e \in SR}$, then $\boldsymbol{sreg(e, N_{fu})}$

If $\boldsymbol{e \in DP}$, then $\boldsymbol{dp(e, N_{fu})}$

If $\boldsymbol{e \in MX}$, then $\boldsymbol{mx(e, N_{fu})}$

If $\boldsymbol{e \in DMX}$, then $\boldsymbol{dmx(e, N_{fu})}$

For $\forall$ I paths beginning and ending at element $\boldsymbol{e}$

For $\forall$ $\boldsymbol{e}$' elements on this I path

Element ($\boldsymbol{e'}$, N_{fu})

4 Net Analysis

For next analysis it is supposed digital circuit design, which successfully passed testability verification and that the corresponding Petri nets $N_e = (B_e, \Sigma_e, F_e, M_{e0})$ for $\forall e \in TE$ was built. The described approach was based on approach from [10].

4.1 Executability of Transition e_e:

Let $e_e \in \Sigma_e$, $c_e \subseteq B_e$. Transition e_e is c_e executable if:

$^{\bullet}e_e \subseteq c_e \wedge e_e^{\bullet} \cap c_e = \emptyset$

Remarks: $^{\bullet}e_e$ means preset of transition e_e, $^{\bullet}e_e = \{y \mid yF_e e_e\}$, where F_e is flow relation of Petri net N_e

$e_e^{\bullet}$ means preset of transition e_e, $e_e^{\bullet} = \{y \mid e_e F_e y\}$, where F_e is flow relation of Petri net N_e

4.2 Successive Markings c_e':

Let $e_e \in \Sigma_e$, $c_e \subseteq B_e$ and transition e_e is c_e executable. Marking

$c_e' = (c_e \setminus {}^{\bullet}e_e) \cup e_e^{\bullet}$

is called successive markings to markings c_e:

4.3 Transition Between Markings $\Rightarrow$:

1) $c_{e1}, c_{e2} \subseteq B_e$

2) $c_{e1} \Rightarrow c_{e2} \quad \Leftrightarrow \quad c_{e2}$ is successive marking to c_{e1}

4.4 Transition Between Zero, One or More Markings $\Rightarrow$*:

1) $c_{e1}, c_{e2}, \ldots, c_{en} \subseteq B_e$

2) $c_{e1} \Rightarrow^{*} c_{en} \quad \Leftrightarrow \quad c_{e1} \Rightarrow \ldots \Rightarrow c_{en} \wedge n \geq 0$

4.5 Set of Places Γe:

1) $\Gamma_e \subseteq B_e$

2) $\Gamma_e = \{c \mid \forall\, m: M_{e0} \Rightarrow^{*} m \Rightarrow^{*} M_{e0}, c \in m\}$

Remarks: Initial marking M_{e0} is same as end marking. Initial marking matches state before test vectors application and end markings matches state after acquiring element response to test stimuli as used in [7] (marks are transported to primary outputs where disappear)

4.6 Dependency Relation z

$z \subseteq TE \times TE$

and must hold:

1) For $\forall\, e_1, e_2 \quad (e_1, e_2) \in z \Leftrightarrow \Gamma_{e1} \cap \Gamma_{e2} \neq \emptyset$.

Relation identifies elements that use same test resources. If any two elements e_1, e_2 needs for successful test same resources, it means that these two elements are test dependant and can't be tested concurrently. So: $(e_1, e_2) \in z$. Relation z is reflexive (element e can't be tested

concurrently with ***e)*** a symmetrical (if (e_1, e_2) then (e_2, e_1) and vice versa). The dependency relation was constructed using the INA software [9] which was remotely controlled from the main application by the TCL interface. The main application was written in pure C++ with usage of STL library. The application takes model of digital circuit generated with software framework from [8], transforms it to Petri net and than analyze the net. Output from the application is dependency relation that could be processed with test circuit partitioning and test scheduling software.

Now the digital circuit partitioning takes place. It is very time consuming task, because some additional constraints must be taken into account. Most common constraints are power constraints. These constraints are used for limiting chip power dissipation level during test application process; because it is known that power dissipation in diagnostic mode could be significantly higher than in normal functional mode. The circuit partition algorithm must deal with this. The main goal of the method is to group together as many elements as possible. Mutually conflicting elements could be than tested together as one block. The fault coverage for these grouped blocks is lower but test time could be dramatically reduced. The algorithm must take it into account and group only these blocks where fault coverage is still on acceptable level or to insert another scan register to scan chain. The algorithm must also recalculate test vectors and responses to cope with these new blocks. After that non conflicting blocks could be tested concurrently. Due to huge solution space of this problem some heuristic must be used. The approach with genetic algorithm is being developed and is subject of next research.

5 Conclusions

For test scheduling it is possible to use any of test scheduling methods that are common for SOC test scheduling. Namely it can be used mixed integer linear programming (MILP) test scheduling method or some another graph based method working with test application conflict graph models (TACG) or test compatibility graph (TCG). There were carried out some experiments with these methods in our department. Some approaches with Tabu search and genetic algorithm were implemented over TACG and TCG models for SOC test scheduling so it can be successfully ported to RT level test scheduling. It is common that test scheduling phase is preceded by test verification phase, because only for really testable circuits, it is possible to make successful test schedule. As test verification method developed in our department [3] uses similar Petri net models, it is possible to share these models and save computational time during design process. And this is another advantage of presented approach.

For the future, we have an intension to integrate our methodology into the system of formal methodologies developed at our department. The goal is to demonstrate that formal models based on concepts of discrete mathematics, theory of graphs and other disciplines can be used to solve problems from the area of diagnostics and testing.

This research was supported by the Czech Ministry of Education – FRVŠ grant No. 3383/2006/G1 and the Grant Agency of the Czech Republic under grant No. 102/04/0737, Modern Methods of Digital Systems Design.

References

1. Kotásek Z., Růžička R., Hlavička J.: Formal Approach to RTL Testability Analysis, Proc. of IEEE LATW 2000, IEEE, Rio de Janeiro, 2000, pp. 98–103
2. Reisig, W.: Petri Nets, An Introduction. Springer Verlag, Berlin Heidelberg, 1985, 160 p.
3. Ruzicka, R.: Testable Design Verification Using Petri Nets, Proc. of Euromicro Symposium on Digital System Design 2003, Belek, Turkey, 2003, pp. 304-311
4. Wakerly, J. F.: Digital Design, Principles and Practices., Prentice Hall, New Jersey, 2000, ISBN 0-1376-9191-2
5. Niraj K. Jha, Sandeep Gupta: Testing of Digital Systems. Cambridge University Press, 2003, ISBN 0-5217-7356-3
6. Růžička, R.: Data Dependent I Path and their Utilisation in DFT. Symposium of students and doctoral works FEI VUT, Academic publisher CERM, s.r.o., Brno, 2000, pp. 228-230
7. Škarvada J: Testability verification of digital circuit design [diploma thesis]. Faculty of Information Technology, Brno, 2004
8. Růžička R., Škarvada J.: RTL Testability Verification System. Proceedings of the Work In Progress Session of 30th Euromicro Conference, Linz, Germany, 2004, ISBN 3-9024-5705-8
9. Roch, S., Starke, P. H.: INA Integrated Net Analyzer Version 2.2 Manual. Humboldt-Universität zu Berlin Institut für Informatik Lehrstuhl für Automaten- und Systemtheorie. URL: http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina/ , May 2005
10. Růžička R: On the Petri Net Based Test Scheduling. Proceedings of the 8th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, Porto, Portugal, 2005
11. Růžička, R.: Formal approach to testability analysis of digital circuits on RT level [dissertation thesis 2001]. VUT FIT, Brno, 2001