DIRECT IMPLEMENTATION OF PETRI NET BASED MODEL IN FPGA

Hana Kubátová

Department of Computer Science and Engineering,
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13, 121 35 Praha 2
kubatova@fel.cvut.cz

Abstract: This paper shows that a non-deterministic system modeled by Petri-nets (analyzed and simulated by professional software tools, e.g. Design/CPN, JARP, before its hardware implementation) can be successfully hardware implemented (here in an FPGA). We have concentrated to the models with really concurrent actions, with different types of dependencies (mutual exclusion, parallel, scheduled) and their direct hardware implementation. Our Petri-nets model need not be equivalent to a FSM, it means that our Petri-net model can non-deterministically choose a transition to be fired from several enabled ones. The way how such non-determinism can be hardware implemented and moreover how to obtain this hardware automatically is presented in this paper. The way how to do it, how to describe Petri-net model in VHDL, how to translate the real Petri-net model described in PNML (Petri Nets Markup Language) to VHDL with respect to Xilinx synthesized coding hints and the quantitative results from its FPGA implementation will be shown.

Keywords: Petri-nets (PN), Field Programmable Gate Arrays (FPGAs), modeling, simulation, design methodology, non-determinism, random number generator

1. INTRODUCTION

Petri-nets (PN) are a well established mechanism for system modeling. They are mathematically defined formal model and can be subjected to a large variety of systems. PN based models have been widely used due to their easy understanding, declarative, logic based and modular modeling principles and finally due to their possible graphic representation. Our research and experiments were exited by the expanding of the Petri-nets in the hardware design process (Adamski, 2001; Erhard, *et al*., 2001; Gomes and Barros 2001; Khomenko, *et al*., 2003; Uzam, *et al*., 2001; Yakovlev, *et al*., 2000, Yakovlev, *et al*., 2000b).

Our methodology is intended to the high level design of the processor or control system architecture. It allows the development of PN models at some Petri-nets design tool (Design/CPN, JARP, CPN Tools – see DesignCPN; JARP 2002; CPN Tools) the analysis and simulation of the model by these tools. After this high-level design has been developed and validated it becomes possible, through the automatic translation to VHDL description to employ FPGA implementation that will allow a custom device to be rapidly prototyped and tested. An FPGA version of a digital circuit is likely to be slower than an equivalent ASIC version, due to FPGA regular and regularly structured wiring channels rather then custom-built logic, but the easier custom design changes, possibility of an easy FPGA reconfiguration and a relatively easy manipulation make FPGAs very good final implementation bases for experiments.

Most models used in the hardware design process are equivalent to the finite state machine (FSM) (Adamski, 2001; Erhard, *et al*., 2001; Gomes and Barros 2001; Uzam, *et al*., 2001). It is said that the resulting hardware must be deterministic. But we have presented such real models, that are not equivalent to a FSM and their real behavior were tested on the final FPGA design kit platform (Projects 2003). Therefore we have concentrated to those models with really concurrent actions, with different types of dependencies (mutual exclusion,

parallel, scheduled) and their hardware implementation.

In our recent publications (Kubátová, 1998, 1999, 2001, 2003, 2003b; Koblížek, 2001) we searched for the automatic transmission from the PN formal description, described, analyzed and tested by some used software tool for the PN design (e.g. Design/CPN) to its hardware implementation. Our Petri-nets hardware structure has been block oriented. These basic building blocks (places and transitions) are connected automatically by the translation process from the output of Petri-nets software tool according to the real model. The VHDL description of the basic blocks and also of the all model has used.

The main problem was the hardware implementation of the random choice of one transition to be fired from more enabled ones. This property is an essential difference between our approach and those considering only the PN model equivalent to the FSM. This non-determinism seems to be very important and basic for modeling by the real Petri-net and for the modeling of real world problems (Češka, 1994; Khomenko, *et al*., 2003, Yakovlev, *et al*., 2000b). The real PN models are widely used for such recent applications, where embedded fail-safe system is necessary to be modeled at the different levels of abstraction, where the appropriate trade-off between hardware and software parts is searched for (Kopetz, 1997; HW/SW System Design, 1995).

The process of the automatic translation from the PN model to the real design is expressed in Fig. 1. The real well-known software tools for Petri-nets or Colored Petri-nets (CPN) design analysis and simulation having the outputs in PNML can be used as an input. From the PNML language model description the VHDL description according Xilinx synthesized coding hints (Xilinx HDL Coding Hints 2002) of the modeled designed system can be obtained by our program *pnml2vhdl.* Its output – the VHDL description of a modeled system can be the input to some CAD system for an FPGA or ASIC final implementation.

Our experiments were performed for JARP (JARP, 2002), Design/CPN (DesignCPN) and CPN Tools (CPN Tools) as description and verification tools for the PN model, the Xilinx CAD system for the FPGA design and simulation and the FPGA circuit Spartan platforms for the real hardware implementation and experiments (The Product Data Sheets, 2002; Projects, 2003).

The paper is organized as follows. The brief description of a PNML language is in section 2. Section 3 contains graphical and functional descriptions of Petri-nets basic building blocks used for its hardware implementation and simple example for the connection of these blocks. The brief experiment description for the dinning philosophers' problem, the producer-consumer PN model and a very primitive (but with its hardware implementation by an FPGA design kit) experiment with a railway critical rail are described in section 4. Both the practical results with comparisons and the structure of our improved methodology are presented in section 5.

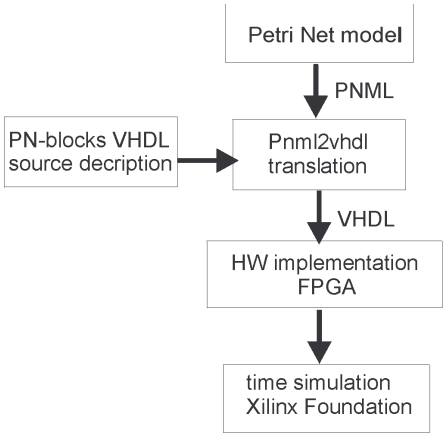

Fig. 1. Algorithm description

2. PNML-LANGUAGE

The PNML format (Petri Net Markup Language) is the data exchange format for Petri-nets description (Petri Net Markup Language, 2002). It is based on XML language. The PNML file can contain several nets structured to pages. Every net contains places, transitions and arcs with attributes (name, initial marking, etc.) and the graphic information. There is the PNML description of the place in Fig. 2. The place has the identifier (id) here *p1*, the place name <*name*>, the initial marking <*initialMarking*> with their attributes <*value*> (the number of tokens) and <*graphical*> (the absolute position of the object "*position*" or the relative position "*offset*").

```
<place id="p1">
<graphics>
<position x="-20" y="10"/>
</graphics>
<name>
<value>ready to produce</value>
<graphics>
<offset x="0" y="0"/>
</graphics>
</name>
<initialMarking>
<value>1</value>
<graphics>
<offset x="0" y="0"/>
</graphics>
</initialMarking>
</place>
```

Fig. 2. PNML description of the place

3. BASIC BUILDING BLOCKS AND THEIR CONNECTIONS

The translation from PNML to the VHDL description is derived from the net structure. We have used modular description of net components – net's elements. The basic element is a **place** (Fig. 3). Each place has to contain information about the actual number of tokens as an output - *out_place*. It has 2 input signals *in_place* (logical OR of the edges from transitions to places) and *ack_place* (the transition firing signal for tokens taking from places). The realization is the counter with the next standard signals: *clk*, *reset* and *ce* (for action enabling). The VHDL input parameter is the size of this counter.

The transitions remove tokens from their input places and put them into their output places. The transition can be fired only when it is enabled. The selection what from more enabled transition shall be fired is performed in the block "Random selection". **Transitions** are implemented by a structure of logic expressions – the AND-gate for transition enabling and next signal for selection the transition to be fired from more enabled ones. The enabled transitions are marked by the inputs of the AND gate (Fig. 4).

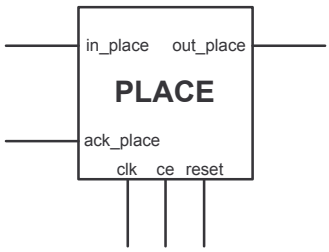

Fig. 3. A place – counters with control logic parts.

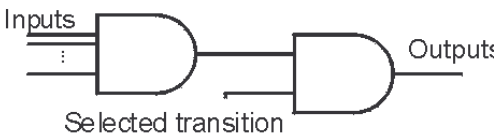

Fig. 4. A transition

The block "**random selection**" selects one transition to be fired from all enabled ones. For the possible hardware implementation of random numbers generator see (Chu and Jones, 1999; Shackleford, *et al*., 2002). The block has three parts: LFSR, counter and shifter, see Fig. 5. The hardware "pseudo-random" number generation can be based on Linear Feedback Shift Register (LFSR) connected according the polynomial (primitive, irreducible) (Stroud, 2002). We have implemented a 16 bit LFSR that is initially seeded by all ones. The down-counter is preset by the random value obtained from LFSR. The number of bits of this "random number" depends on the model structure (the number of transitions). The end of the counting period becomes when the counter underflows "0", then the output "stop" is set. The shifter is realized by a circular shift register with flexible number of Flip-Flops (according to the number of enabled transitions). It shifts only one "1" so many times as the counter underflows "0". Therefore after pseudo-random number of cycles one from enabled transitions is selected. The outputs of Flip-Flops are connected with all Petri-nets transitions, see input "Selected transition" in Fig. 4.

The input of the "**random selection**" block is the vector representation of all enabled transitions. This output vector contains only one "1" choosing one firing transition and the rest values are zeros.

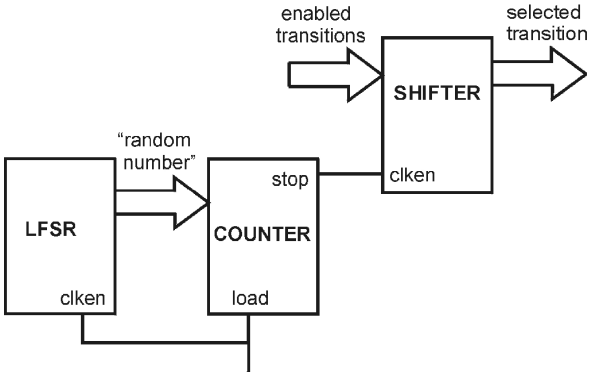

Fig. 5. A "random selection" block

Concrete connections and a concrete structure of these blocks depend on the real PN architecture. The integral design has a block structure, the re-design or some improvements of some blocks are independent processes. The number of the signals is changed according the actual PN structure.

The behavior of the whole PN model realization is controlled by the state graph (Fig. 6):

- State 0: Initial state (after RESET signal).
- State 1: Initialization of the block SHIFTER
- State 2 : Set the COUNTER by the LFSR output by the signal "load"
- State 3 : The COUNTER works until "stop"=1
- State 4: The selected transition is fired.

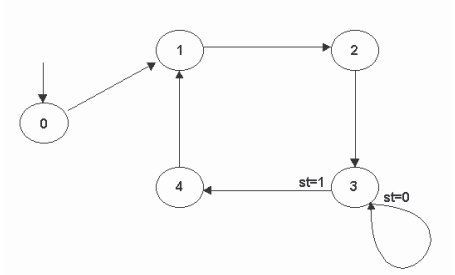

Fig. 6. The state graph of the controller

The conversion from the PNML description into the VHDL code is performed by the *pnml2vhdl.exe* program, written in C language. Input parameters are PNdescription in PNML format, source VHDL files

with predefined blocks and the maximum capacity of the places (implicit value is 5). The edges in our PN model must be only simple, but our new tested version enables multiple edges. The structure of the PN model is tested during the translation – the number of transitions, the number of places - to choose appropriate parameters of the "**random selection**" block. The last phase is the construction of the logical connections of all blocks by gates as an expression of the edges and transitions. Simple example of the PN model with 4 places and 3 transitions and connections of the blocks is in Fig. 7. This model is presented only to show the interconnection of the net elements.

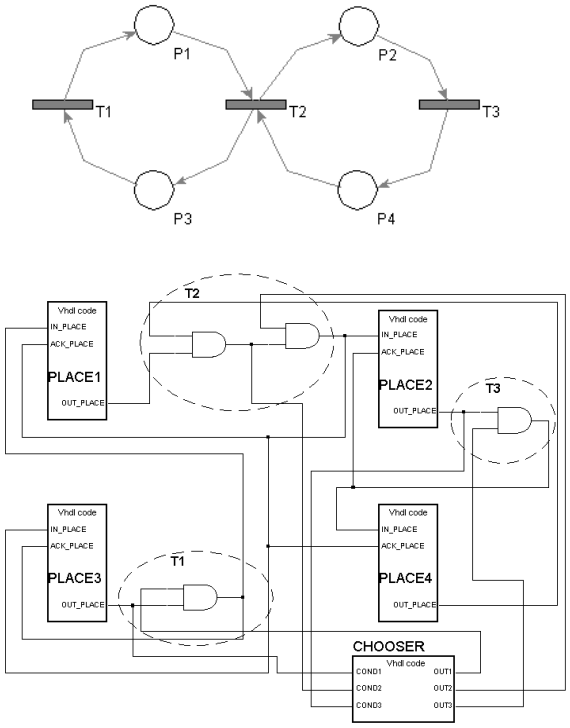

Fig. 7. Simple example of the block connections

4. EXPERIMENTS

We have modeled 5 philosophers, who are dining together, Fig. 8. The philosophers each have two forks next to them, both of which they need in order to eat. As there are only five forks it is not possible for all 5 philosophers to be eating at the same time. The Petri-net shown here models a philosopher that takes both forks simultaneously thus preventing the situation where some philosophers may only have one fork but are not able to pick up the second fork as their neighbors has already done so. The token in the *fork* place (places P1, P2, …, P5) means that this fork is free. The token in the *eat* place (places P6, P7, …, P10) means that this philosopher eats.

Our implementation has used 81 CLB, 57 flip-flops with maximal working frequency 14 MHz. We have verified that the hardware "random selection**"** block has worked with a random and relatively uniform probability of the transition to be fired choice, see Table 1. (Kubátová, 2003).

Table 1: Distribution of the philosophers' activity for 200 cycles – transition firing.

Ph.1	Ph.2	Ph.3	Ph.4	Ph.5	Nobody
13%	20%	16%	16%	19%	16%

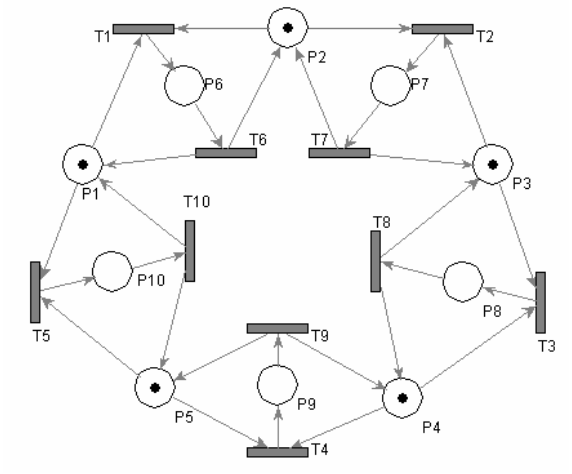

Fig. 8. The dining philosophers PN model

We have performed experiments with a "producer-consumer" system. Our FPGA implementation has used 59 CLB blocks, 47 flip-flops with maximal working frequency 24.4 MHz. The maximum input capacity parameter for places (the size of the counter) was set to the value 3. The average buffer occupation during 120 cycles (transition firings) was 1.43, (Koblížek 2001).

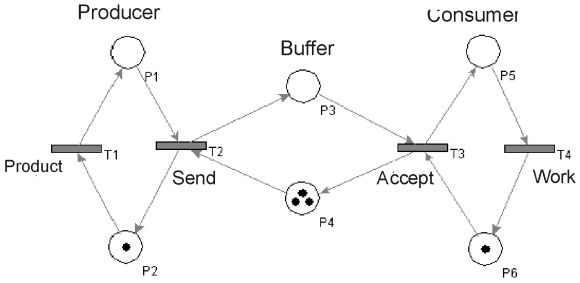

Fig. 9. Producer – consumer model

Our real application experiment modeled the railway with one common critical part – rail, see Fig. 10. The PN model (Fig. 11) has initial marking where tokens are in places "1T" and "3T" (two trains are on the rails 1 and 3 respectively), "4F" (a critical rail is free) and 2F (a rail 2 is free). This model has eight places, two ones T (train) and F (free) for each rail: a token in the first one means that the train is in this rail (T-places) and the second means that this rail is free (F-places). It was described and simulated in the Design/CPN system and then it was implemented in

the real FPGA design kit (ProMoX), (Projects, 2003; Kubátová, 2003).

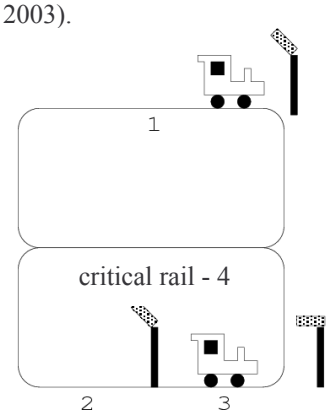

Fig. 10. Railway semaphore model

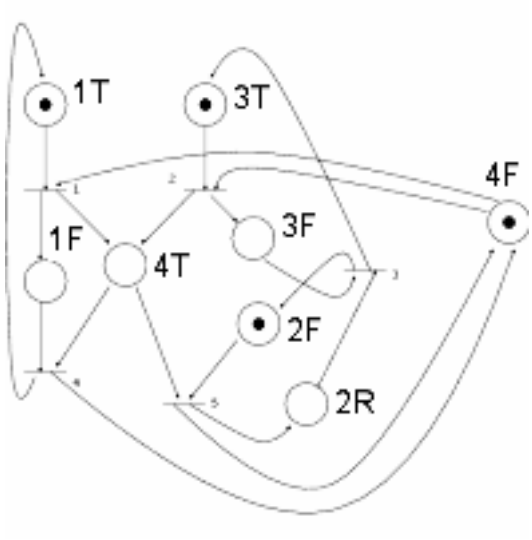

Fig. 11. The PN model of 4 rails

5. IMPROVEMENTS AND CONCLUSIONS

We have performed many experiments with different PN models and also with different final VHDL implementation. Several improvements of our Pnml2vhdl compiler from a PN model to VHDL hardware description were made, too. The most space demanding is implementation of the "**random selection**" block. The quantitative properties of this block with respect to the number of maximum transitions that can be fired at one time (at one PN marking) are in Table 2. Table 3 contains parameters from "**place**" block implementation.

Table 2: Parameters of the "Random selection" block implementation.

Number of transitions	Number of bits of "random number"	CLB blocks	Flip-Flops
5	3	33	31
10	4	47	37
15	4	59	42
20	5	72	48
25	5	67	53

Our main aim is to propose the methodology for the unified design process from the highest level of abstraction allowing modeling both hardware and software parts, their verifications and modifications according the simulation process. We propose to use Petri-nets and commonly used Petri-nets tools for this purposes since the standardized output of these tools was made public (Petri Nets Markup Language 2002).

We have presented a process of direct transmission from Petri-nets representation to its hardware implementation. But not all model expressed and modeled by a Petri-net are non-deterministic, many of them are equivalent to a FSM. However, the hardware implementation of a FSM is a standardized part of FPGA CAD tools. Therefore the improved algorithm has been suggested, see Fig. 12. Only such models which are not equivalent to a FSM are implemented according methodology based on VHDL implementation of PN building blocks (section 3). The rest e.g. safe Petri-nets (Češka, 1994, Yakovlev, 2000b) are transformed to VHDL description of the FSM.

Table 3: Parameters of the "place" block implementation.

Max. place capacity (size)	CLB blocks	Flip-Flops
1	2	1
3	3	2
5	5	3
10	6	4
15	6	4
25	7	5

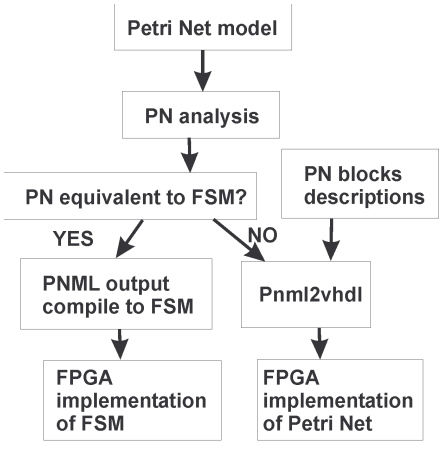

Fig. 12. Improved algorithm

Our other "work in progress" is compilation of some parts of PN model to the assembler program of some processor on a chip and verification of the all design both FPGA and processor (software) parts. Since the

high-level representation of the designed system is by PN model easy applicable, the all parts of designed system can be verified altogether.

The main conclusion of this paper is that we are able to directly implement the verified high-level PN model in the desired hardware, here FPGAs. Our model can be a Petri-net not equivalent to a FSM. Our main aim is to create a methodology applicable to a wide spectrum of systems with respect to their appropriate trade-off between hardware and software parts.

ACKNOWLEDGEMENTS

This research was in part supported by a grant 102/04/0737 of the Czech Grant Agency (GAČR) and the MSM 212300014 research program.

REFERENCES

Adamski, M. (2001). A Rigorous Design Methodology for Reprogrammable Logic Controllers. *Proc. DESDes'01* Zielona Gora, Poland, pp. 53 - 60.

Chu, P. P., Jones, R.E. (1999). Design Techniques of FPGA Based Random Number Generator. *Military and Aerospace Applications of Programmable Device Conferrence*. Cleveland State University, Cleveland, Ohio, USA.

Češka, M. (1994). *Petriho sítě.* Akademické nakladatelství CERM Brno, (in Czech).

Erhard, W., Reinsch, A. Schober, T. (2001). Modeling and Verification of Sequential Control Path Using Petri Nets. *Proc. DESDes'01* Zielona Gora, Poland, pp. 41 - 46.

Gomes, L., Barros, J-P. (2001). Using Hierarchical Structuring Mechanism with Petri Nets for PLD Based System Design. *Proc. DESDes'01* Zielona Gora, Poland, pp. 47 - 52

Khomenko, V., Koutny, M., Yakovlev, A. (2003). Logic Synthesis Avoiding State Space Explosion. *Technical Report Series CS-TR-813*, University of Newcastle upon Tyne.

Kopetz, H. (1997). *Real-Time Systems. Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers.

Koblížek, M. (2001). Hardware Implementation of Petri Nets. *Diploma theses, CTU*, Prague.

Kubátová, H. (1998). Petri Nets in Hardware Design. In: *Proceedings of the 32nd International Conference MOSIS'98*. Ostrava: VŠB-TU, Department of Computer Science of FEI. pp. 249 - 254

Kubátová, H. (1999). Hardware Implementation of Petri Nets. In: *MOSIS'99*. Vol. 1. Ostrava: VSB-TU, pp. 88 - 92.

Kubátová, H. (2001). Petri Net Simulation Using FPGA. In: *Proceedings of XXIIIrd International Autumn Colloquium.* Ostrava : MARQ, , p. 129-134.

Kubátová, H. (2003). Petri Net Models in Hardware. *ECMS 2003*. Liberec, Technical University, pp. 158 - 162

Kubátová, H. (2003b). Direct Hardware Implementation of Petri Net based Models. Proceedings of the *Work in Progress Session of EUROMICRO conf.* Linz: J. Kepler University – FAW. pp. 56 - 57.

Shackleford, B., Tanaka, M., Carter, R. J. and Snider, G. (2002). FPGA implementation of neighborhood-of-four cellular automata random number generators. *International Symposium on Field Programmable Gate Arrays,* ACM/SIGDA California USA, pp. 106 – 112.

Stroud, C. E. (2002). *A Designer's Guide to Built-In Self-Test*. Kluwer Academic Publisher.

Uzam, M., Avci, M. and Kürsat, M. (2001). Digital Hardware Implementation of Petri Nets Based Specification: Direct Translation from Safe Automation Petri Nets to Circuit Elements. *Proc. DESDes'01* Zielona Gora, Poland, pp. 25 - 33.

Yakovlev, A., Gomes, L. and Lavagno, L. (2000). *Hardware Design and Petri Nets*. Kluwer Academic Publishers.

Yakovlev, A., Xia, F.and Shang, D (2000b). Synthesis of a signal-type asynchronous data communication mechanism and its hardware implementation. *Technical Report 720*, Department of Computing Science, University of Newcastle, 10 p.

Used web pages:

JARP (2002). Home Page.http://jarp.sourceforge.net/

Petri Nets Markup Language (2002). Humboldt-Universität Berlin. http://www.informatik.hu-berlin.de/top/pnml

HW/SW System Design (1995): http://markun.cs.shinshu-u.ac.jp/kiso/projects/paper/Doctor/1995/pauline/

Projects (2003): http://service.felk.cvut.cz/courses/36APZ/prj/36APZ114

DesignCPN: http://www.daimi.au.dk/designCPN/

CPN Tools: http://wiki.daimi.au.dk/cpntools/cpntools.wiki

The Product Data Sheets (2002). Xilinx, Inc. http://www.xilinx.com/partinfo.

Xilinx HDL Coding Hints (2002). Xilinx, Inc.. http://service.felk.cvut.cz/courses/36SIM/CODING_HINTS/Xilinx_hdl-coding.pdf.