Reachability Analysis of Petri-nets by Using FPGA Based Accelerators

Ass. Prof. András Pataricza

Technical University of Budapest
Department of Measurement and Information Systems
H-1521 Budapest, Műegyetem rkp. 9. R. ép. I em
Hungary

E-mail: pataric@mmt.bme.hu
Tel: (+36 1) 463 3595 / (+36 1) 463 2057
Fax: (+36 1) 463 4112

Reachability Analysis of Petri-nets by Using FPGA Based Accelerators[1]

Gy. Csertán, I. Majzik, A. Pataricza
Department of Measurement and Information Systems
Technical University of Budapest, H-1521 Budapest, Műegyetem rkp. 9
E-mail: csertan, majzik, pataric@mmt.bme.hu

S. C. Allmaier
Department of Computer Structures (IMMD3)
University of Erlangen, D-91058 Erlangen, Martensstr. 3
E-mail: snallmai@informatik.uni-erlangen.de

Abstract

The application of an FPGA-based accelerator for reachability analysis of Petri nets (PN) is presented. Since the simple components of the PN can be easily realized in high-density FPGAs, the complete problem can be mapped to silicon providing a solution environment faster than the traditional software-based simulators.

As the complexity of computer systems increases and more reliance is placed on the service these systems deliver, the *formal verification* of fundamental system properties like absence of deadlock, freedom from starvation, unsafe states etc. becomes a necessary step in the design process. A number of these properties to be verified are effectively reducible or equivalent to the reachability problem, i.e. to check whether some arbitrary state(s) can be reached from a fixed initial state of the system. Different abstract mathematical models, like dataflow networks, Petri-nets, process algebra are used to study these properties. These modeling paradigms share the very same key problem of state reachability analysis. In the following, we focus on a selected formalism, the Petri-nets.

[1] have shown exponential space lower bound for the reachability problem. Thus, although the computational power of modern computing equipment increases rapidly (even accompanied by a radical drop in the price/performance ratio), the processing capacity is often insufficient. In the case of complex target systems, solving the reachability problem either requires radical model simplifications or extremely long run times.

In the literature, a number of methods are proposed for efficient reachability analysis, each having its advantages and drawbacks. The symmetry method [4] needs the help of the analyst. Stubborn sets [7] and incomplete reachability analysis [3] in its original form aim at only finding deadlocks. Symbolic model checking [6] uses binary decision diagrams (BDD) to analyze huge state spaces, however, it is not guaranteed that most distributed systems have a compact BDD-based representation.

In this paper we focus on a completely different appoach by investigating how the performance bottleneck can be resolved by utilizing the advantages of custom computing, i.e. the application of complex, high-speed user-programmable and reconfigurable FPGA circuits which enable to map the complete problem to silicon.

The paper is stuctured as follows. Section 1. introduces our approach of using custom computing. Section 2. describes our target formalism, the Petri nets. In Section 3. the structure of the accelerator and the synthesized components of the Petri-net are presented. Section 4. discusses the advantages and limitations, Section 5. describes our experimental implementation.

1. The approach

The traditional solutions to overcome the performance bottleneck can be grouped into two typical categories. In *multi-processors* the task to be performed is distributed between different general purpose processing elements according to a structural or functional problem decomposition. In *co-processors* application dependent dedicated hardware subunits perform some elementary operations typically by about one order of magnitude faster, than a program running on the

[1]This work was supported by the Hungarian NFS under the grant T15728 and by the German-Hungarian intergovernmental research contract under the acronym ACCUSE

CPU itself. In many cases, the large hardware overhead of a massively parallel system is unacceptable, thus either a monoprocessor or a minor configuration of a multiprocessor extended by a co-processor per computing node is used.

The advent of high-speed and complex in-system programmable FPGA circuits opened new horizons for the use of dedicated hardware solutions by providing a similar circuit complexity, as earlier VLSI designs at a very low price for the entire spectrum of computer architects. The new field of reconfigurable (or in another terminology custom) computing supports the design of general-purpose co-processors. As both the structure and instruction set of these co-processors can be downloaded just before starting the program, they serve simply as a silicon compiled problem dedicated subroutine set for a variety of applications. The possibility of a run-time reconfiguration allows the use of adaptive algorithms from a pre-compiled library prepared in the algorithm development phase.

Another promising, but still unexplored possibility is the use of *accelerators*, which are well-proven solutions in hardware emulation. In contrary to the co-processor approach, where only some parts of the algorithm are mapped to silicon, in accelerators some complete parts of the problem are realized on the FPGA. This way, additionally to a part of the solution algorithm, some data structures corresponding to the actual data are realized on the FPGA as well. A potential way to reduce the synthesis related overhead is to adopt a similar philosophy as the standard cell based ASIC design approach does it: an application dependent basic cell library is defined and only the interconnection network is synthesized during the algorithm development phase. This way the majority of the synthesis task is accomplished already prior to the actual program run.

All of the above mentioned solution alternatives are candidates for the use in modeling and validation of digital systems. In our case, reachability analysis of systems modeled by PN, the accelerator is a favorite candidate, since the very simple components of the PN (places and transitions) can easily be mapped to silicon.

2. Petri-nets and reachability analysis

A Petri-net is a 4-tuple $N = \{P, T, F, m_0\}$ where $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places, $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions satisfying $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation (set of arcs) and $m_0 : P \mapsto I\!N$ is the initial state. The symbols $*t$, $t*$, $*p$, $p*$ denote the pre-set and post-set of a transition t or a place p.

A state of a Petri-net is an assignment of nonnegative integers to each place. If a place p is assigned k then it can be referred to as p is marked with k tokens. A transitions t is enabled if each $p \in *t$ has at least one token. Firing of a transition removes one token from each place in its pre-set and puts a token to each place in its post-set. A marking m is reachable from the initial marking m_0 if there exists a sequence of firings that transform m_0 to m. A Petri-net is said to be *k-bounded* if for all reachable markings, the number of tokens in any place is less than or equal to k. An 1-bounded net is said to be *safe*.

The set of reachable states forms the reachability set (RS). If it is accompanied by the transition relation, then a reachability graph (RG) is formed. The RS (or RG) can be computed by the following algorithm:

Algorithm 1 (Generation of the RS) *Given a Petri-net in its initial state. The algorithm performs an exhaustive simulation (by a breadth-first-search). Function* ChooseRemove *selects a state and removes it from the set of found ones,* NewState *generates a successor state,* SearchInsert *examines whether the actual state is in the union of the completed and found states; if not, then adds the actual state to the set of found states. The reachability set is generated in* completed_states*.*

```
completed_states = ∅
found_states = {Initial_state}
while found_states ≠ ∅ do begin
        i = ChooseRemove(found_states);
        completed_states = completed_states ∪ i
        for each transition t that is firable in i do begin
                j = NewState(i,t);
                SearchInsert(j, completed_states ∪ found_states, found_states);
done; done
```

3. Structure of the accelerator

The task of the accelerator is the generation of the RS of the PN by simulation as described in Algorithm 1. A co-processor could realize the Petri-net simulator engine by interpreting transition firings (state changes) stored in some tabular form in local memory, in a similar manner as a pure software simulator would do it, but using a silicon compiled

interpreter. An accelerator provides a more efficient solution. It performs the very same task after a synthesis phase implementing a realization of the PN, this way a speed as high as one transition firing per FPGA clock period can be reached by this direct emulation of the PN. The price of the higher speed is the run time needed for the synthesis of the PN. However, the few number of regular components (places and transitions) can be used as precompiled macros requiring only their interconnection (wiring) to be generated in each run.

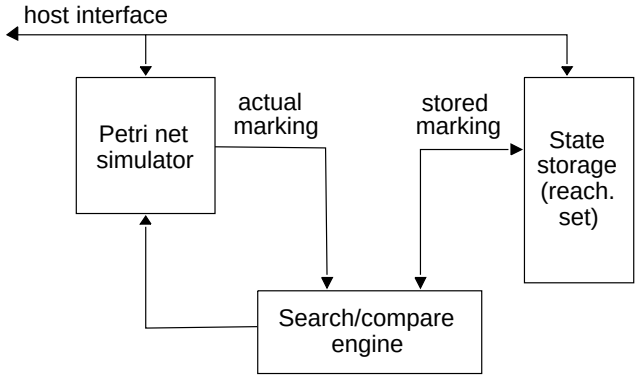

Figure 1: General structure of the accelerator

Accordingly, our accelerator is structured as follows (Figure 1). The *synthesized PN simulator* implements the successor function in silicon. The *state storage* contains the set of found states. A new state reached by the simulator is examined. If it has not occurred yet, then it is inserted into the storage; otherwise a successor state is generated. The task of the *search/compare engine* is to compare the actual state in the simulator with the states in the state storage or choose a new state from the found set if it is needed. This module is implemented as a state machine.

3.1. Representation of the Petri net

Places (Figure 2) are realized in the FPGA by the following elements: (i) a counter representing the number of tokens in the place, (ii) zero logic signaling if the counter is non-zero, (iii) enabling logic for counting up/down of the counter.

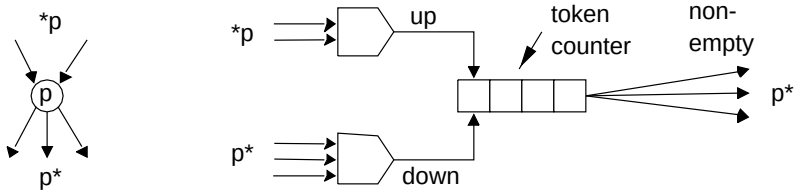

Figure 2: Representation of a place

Transitions (Figure 3) are realized by (i) a flip-flop which indicates if the transition has already fired, (ii) logic enabling the transition if input places contain the necessary number of tokens, (iii) logic signaling if the transition is firable i.e. it is enabled and selected to fire, (iv) daisy chain logic (see below).

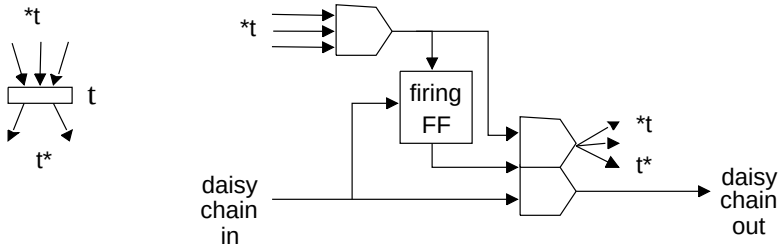

Figure 3: Representation of a transition

All transitions of the net are connected to form a daisy chain (DCT, Figure 4). In a given marking, the enabled transitions will fire in this order. (Since all enabled transitions will fire, their order is irrelevant from the point of view of the RS.) The input of the first transition is connected to an active level, the output of the last transition (i.e. the output of the daisy chain) is connected to the controller of simulation. A transition fires if it is enabled (enabling logic is active), it has not fired (firing flip-flop is inactive) and it has an active input signal in the daisy chain (it is the first in the chain among the transitions being enabled). In this case it does not propagate the active level for its successors

in the chain until it has fired and the firing flip-flop is set. Transitions which are not firable propagate the level on their input to their output in the chain. Transitions which are enabled but are later in the chain can not fire before the preceding ones have fired.

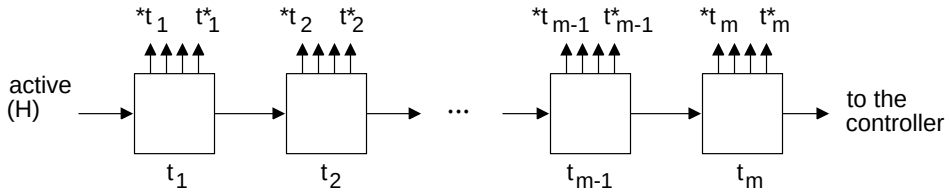

Figure 4: Daisy chain of transitions (DCT)

3.2. Simulation control

The control of the simulation can be implemented in breadth-first-search (BFS) or depth-first-search (DFS) manner.

In a BFS approach, all successors of a given state are generated and stored in the state storage before a next state is set. In a given state, the transitions are examined. If there is a firable transition then it fires. The resulting state is searched in the RS; if it is not found then it is stored. Then the previous state is restored in the simulator and, if exists, the next firable transition fires and the procedure is repeated. In this way, all enabled transitions in a given state fire, in the order determined by the DCT. After all enabled transitions fired, the examination of the actual (repeatedly restored) state is completed. This fact is indicated in the state storage by a flag associated with the state (C flag). A new state, which is not completed yet, is choosen from the state storage, and its successor states are examined as described. The algorithm is sketched as follows:

Algorithm 2 (BFS based generation of the RS in the FPGA) *: Given the Petri-net in its initial state in the simulator (the initial state is loaded into the counters corresponding to the places, the firing flip-flops of the transitions are reset).*

- *If the output of the DCT is inactive then there is a firable transition. By providing a clock signal, the firable transition in the DCT fires, which steps the counters of the places corresponding to the transition and sets the firing flip-flop of the transition. A new state is reached.*
  The new state is read from the simulator and searched in the state storage. If it is not found then it is stored without an active C flag. (If the RG is also to be built then a pointer is stored which identifies its predecessor state, i.e. the previous one.)
  The previous state of the Petri-net is restored in the simulator and the algorithm restarts (the flip-flops of the DCT are not changed after the last firing).
- *If the output of the DCT is active then there is no firable transition. The actual state has no further successors, its C flag in the state storage is activated. A new state without an active C flag is choosen from the state storage.*
  - *If there is such state then it is loaded into the simulator. The firing flip-flops in the DCT are reset. The algorithm restarts.*
  - *If none is found then the generation of the reachability set is completed, the algorithm exits.*

In comparison with the above BFS, the DFS approach has the advantage that the original state needs not be restored after each simulation step, thus the speed of the simulation increases. However, it requires a modification of the hardware structure of the simulator. If a transition fired then this fact has to be remembered (returning to this state it should not fire again). To do this, a firing counter should be assigned to each state, marking the transition which last fired. The input of the DCT is not wired to an active level, since only that part of the DCT is enabled which is located after the last fired transition. The logic implementing this modification is quite complex [5], this way the BFS was implemented.

3.3. State storage and the search engine

Theoretically, the state storage and the search engine should be realized by a content addressable memory (CAM). However, since a state is a binary word consisting of a large number of bits (a few hundreds of places each represented by 4..8 bits), the efficient implementation is difficult. Thus we focussed on an alternative solution. A (hierarchical structure of) hash tables is built, and the set of states corresponding to the same hash code is stored dynamically in the state storage, in the form of linked lists. The advantage of this approach is, that the state storage is divided dynamically between the lists, no static splitting is needed.

The efficiency of the storage and the search engine can be improved further if the hash function is based on the structure of the net and can group together (some) successor states. Since the firing of a transition modifies the state in limited number of places, the difference between successor states is restricted to small parts of the binary word representing the state. In this way, only the first element should be stored in its full extent in the list corresponding to a given hash code, the next elements can be stored only as incremental differences. While doing the search, each element can be reproduced each after the other.

4. Advantages and limitations

The performance of the accelerator is significantly higher compared to a software simulator in the subtasks of (i) selection of the transition to fire, (ii) update of the marking in input/output places of a transition and (iii) search and compare in the reachability set. The reconfigurable FPGA has the advantage that different initial states can be set without reprogramming the device.

The speed can be estimated in clock cycles of the simulator. If it is assumed that (i) the PN contains N places each having a bound of 16 tokens (ii), in a single cycle 32 bits are accessed in the simulator and 64 bits in the state storage, (iii) the number of elements in a sub-list of the state storage is S, then a simulation step requires less than $N(S+5)/16$ clock cycles (e.g. if $N = 100$ and $S = 10$ then less than 100 clock cycles).

Main limitations of the approach (and their partial solutions) are as follows:

- Only a bounded PN can be analyzed. Unbounded nets should be examined by approximate analysis or by introduction of more sophisticated formalisms like ω-states.
- The size of the PN is limited by the FPGA. This problem can be solved by model decomposition. Since reachability analysis in general Petri-nets is not compositional, candidate nets are restricted to PNs built in a top-down manner by predefined refinement rules or special net classes like Superposed Generalized Stochastic Petri Nets (SGSPN, [2]) which provide decomposition by definition.
- The size of the RS is limited by the available state storage of the accelerator. The implementation of a more efficient state storage (e.g. custom memory architecture, BDD-based state storage) and a corresponding search engine is a task of our future research.

5. Experimental implementation of the accelerator

An experimental version of the accelerator was built to estimate the time required to synthesize the net and the size of the Petri-net which could fit into the FPGA. The search engine and the state storage were designed to be as simple as possible and the class of Petri-nets was restricted to safe nets.

The evaluation board is the VCC H.O.T.Works DS, a complete PCI-based programming and development system built upon the Xilinx XC6200 chip family. Its main features include XC6216 on-the-fly reconfigurable FPGA with 64x64 logic cells (each containing a flip-flop and/or an arbitrary two-input logic gate), 128Kx32bit on-board SRAM and programmable clock generator. The accelerator was built as a single-chip one, i.e. both the Petri-net and the search/compare engine were implemented in the available FPGA and the on-board memory was used as state storage.

The accelerator was designed by using a VHDL elaborator and the XACT-Step6000 place and route (P&R) software provided for the evaluation board. Our first experiments showed that the main bottleneck of the technology is that the synthesis tools are not able to handle efficiently the regular structure of the PN to be mapped to the FPGA. This way, in order to avoid the P&R process to be an order of magnitude slower than the simulation itself, specialized P&R tools have to be developed.

6. Conclusion

In this paper we have studied the application of an FPGA-based accelerator for reachability analysis of Petri-nets. The experiments showed that up-to-date high-density, high-speed FPGA circuits are able to simulate Petri-nets of reasonable size at encouraging speed. However, in order to reduce the time required for synthesis, specific tools are required which are optimized to the problem domain, i.e. to the components and structure of Petri-nets.

References

[1] E. Cardoza, R. Lipton, and A. Meyer. Exponential space complete problems for Petri nets and commutative semi-groups. In *Proc. 8th annual ACM Symposium on Theory of Computing*, pages 50–54, 1976.

[2] S. Donatelli. Superposed Generalized Stochastic Petri Nets: Definition and efficient solution. In *Proc. 15th Int. Conf. on Application and Theory of Petri Nets 1994, Zaragoza, Spain*, pages 258–277. Springer Verlag, LNCS-815, 1994.

[3] G. J. Holzmann. On limits and possibilities of automated protocol analysis. In *Proc. 7th Protocol Specification, Testing and Verification*, 1987.

[4] P. Huber, A. M. Jensen, I. O. Jensen, and K. Jensen. Towards reachability trees for high-level Petri-nets. Technical report PD-174, DAIMI, Dept. of Computer Science, Aarhus University, 1985.

[5] I. Majzik, A. Pataricza, and S. C. Allmaier. Support of formal verification by FPGA based accelerators. Internal report, Dept. of Computer Structures (IMMD3), University of Erlangen-Nuremberg, Erlangen, Germany, 1997.

[6] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia. Petri net analysis using Boolean manipulation. In *Proc. 5th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain*, 1994.

[7] A. Valmari. Stubborn sets for reduced state space generation. In *Supplement to Proc. 10th Int. Conference on Application and Theory of Petri-nets*, pages 1–22, Bonn, 1989.