High-Level Modelling, Analysis, and Verification on FPGA-based Hardware Design

Petr Matoušek, Aleš Smrčka, and Tomáš Vojnar

FIT, Brno University of Technology, Božetěchova 2, CZ-612 66 Brno, Czech Republic
{matousp,smrcka,vojnar}@fit.vutbr.cz

Abstract. The paper presents high-level modelling and formal analysis and verification on an FPGA-based multigigabit network monitoring system called Scampi. Uppaal was applied in this work to establish some correctness and throughput results on a model intentionally built using patterns reusable in other similar projects. Some initial experiments with parametric analysis using TReX were performed too.

1 Introduction

Implementation of network components in hardware is a trend in advanced high-speed network technologies which applies also for the network monitor and analyser Scampi developed within the Liberouter project [4] that we consider here. The Scampi analyser is implemented in FPGA on a special add-on card. FPGA-based hardware provides a similar functionality of a system as software implemented on general microprocessors. However, in comparison to a software solution, programmable hardware is very fast—it allows Scampi to communicate in multiples of gigabits per second.

In the paper (and its full version [3]), we discuss our experience from high-level modelling and formal analysis and verification of certain important correctness and throughput properties of Scampi. Our analysis of the system started with a preliminary manual analysis, which we do not discuss here, and then continued by an application of automated formal analysis and verification methods. We divide the model of Scampi we used for automated formal analysis and verification into three kinds of components: a model of the environment (generators), a model of buffers (queues, channels), and a model of executive units. We show how the different model components may be constructed and especially in the case of generators and buffers, we obtain general templates that may be reused in different models of systems of the considered kind. Next, we discuss the properties we handled by automated formal analysis and verification using Uppaal [5] and—in some initial attempts for a parametric analysis—TReX [1].

2 The Design of Scampi

Scampi is a network adapter working at the speed of 10 Gbps. The system consists of several components—input buffers, preprocessing units (a header field extractor—HFE), and searching units (a lookup processor—LUP and processing units—PU). The Scampi adapter reads data from one input port and distributes them into four independent paths working in parallel. An IP packet is processed by an HFE unit at first where the IP header is translated into a unified header containing adjusted data like the source/destination IP address, MAC address, port number, VLAN tag, etc. Then, the unified header is processed by a lookup processor, see Fig. 1, where it is classified according to some pre-defined rules. Searching of the LUP consists of packet matching (parallel searching) encoded into a TCAM memory (Content Address Memory) and of additional sequential searching in an SSRAM memory performed by PU.

The results give us information what to do with the packet—e.g., to increment the number of dangerous packets found, to forward the packet to the software layer over Scampi, to broadcast the packet, or to simply release the packet from the Scampi system.

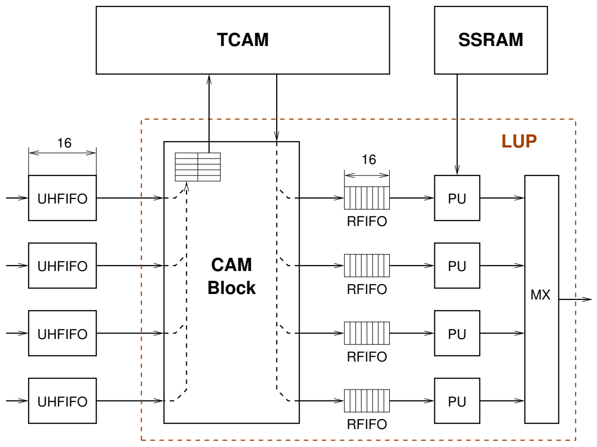

Fig. 1. The structure of the Lookup Processor

3 Modelling Scampi

We now sketch models of several components of Scampi important for its correctness and throughput analysis—their detailed description can be found in [3].

In our approach, we recognise three basic types of components that occur in some form in many complex systems: (i) *waiting FIFO queues* (buffers, channels)—deterministic, stochastic, or non-deterministic; lossy queues, delayed queues, etc., (ii) *executive components*—multiplexers, processing units (lookup processors, preprocessing units), etc., and (iii) *environment*—generators of incoming requests (packets) or output units consuming the results.

Modelling Waiting Queues. A FIFO queue is a typical abstract data structure that contains a sequence of stored data. Here, we abstract away the content of the queue items and we concentrate only on the number of items in

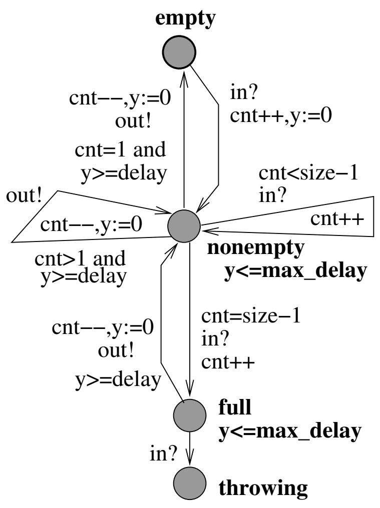

Fig. 2. A model of a delayed queue

the queue. FIFO queues are used to represent transmitting channels, intermediate buffers between a processing unit and a memory, etc. We can have lossy queues where some data may be lost. There are delayed queues where data are delayed. We can model bounded or unbounded queues, or we can also deal with queues where a symbolic constant value—a parameter—defines the maximum length of the queue. We try to model all these queues without expecting any highly specialised features of the modelling language.

In Figure 2 there is a model of a delayed FIFO queue where every request is guaranteed to be delayed at least *delay* time units before it is released, but at maximum *max_delay* time units. The delayed queue is modelled using timed automata. Transitions that release an element of the queue are augmented with time constraints allowing to release an item only if the $y \geq delay$ guard is satisfied ensuring the lower bound on the delay. The upper bound is ensured by the $y \leq max\_delay$ invariants of the appropriate states. This pattern of a waiting queue was applied to model four UHFIFO queues and four RFIFO queues of the Scampi system working in parallel.

Modelling Executive Components. While creating a model, one often has to reflect the goal of the verification. In our case, we are interested in timing of the components. If executive components have an accurate timing plan, we distinguish two kinds of states in the model. The first is an urgent state that we use for observers. The second type is a state that models delays of the system. The latter kind of a state has an incoming transition resetting a clock ($t := 0$), an invariant that defines a time constraint over the clock ($t \leq delay$), and an outgoing transition constrained by a condition on the clock ($t = delay$). Using these principles, we modelled the TCAM memory, PU, and multiplexer components of the LUP.

We are interested in the minimum guaranteed throughput of the system which can be calculated from the size S of an incoming packet in bits—we take the minimum possible size which is the worst case in the given setting, the average delay D of the rule matching process in time slots for the worst possible scenario, and the size T of a time slot in seconds: $throughput = S/(D * T)$. We approximate the average delay for the worst case from the worst delay possible for transmitting x results (a user chosen, reasonably large value). To compute the worst delay for x results, we can use a counter of outgoing results and a clock. When the counter reaches x, the system fires a transition to an observer state (the `slow` state in Fig. 3) provided the clock value is greater than the allowed delay. The analysis of the throughput is then based on manually finding the minimum value of the delay (by running model checking several times) such that the system does not reach `slow`.

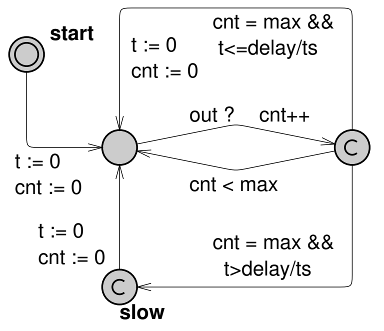

Fig. 3. A timed automaton MX for throughput checking

4 Verified Properties

Verification by Uppaal. Let us now mention a few examples of properties (written in CTL) that we verified over the above presented models—more can be found in [3]: (i) $A\,\Box\,\neg\, deadlock$—no deadlock is possible in the system. (ii) $A\,\Box\,\neg(RFIFO0.full \vee RFIFO1.full \vee RFIFO2.full \vee RFIFO3.full)$—this property holds even if the RFIFO size is 2. It means that RFIFO can be replaced by a one-place buffer. We can use a similar property on other queues and see whether some data is thrown away. (iii) $A\,\Box\,\neg\, MX.slow$—this property expresses the throughput checking mentioned above. The property is satisfied when the delay for 1000 counted results is set at least to 16000 time slots (the average delay for the worst case is 16 time slots). Now, we can calculate the minimum system throughput (from the smallest supported packets—64 bytes) caused by the Lookup Processor: $(64{*}8\ bits)/(16{*}20{*}10^{-9}\ seconds) = 1.6\ Gbps$.

Parametric verification by TReX. Parametric verification is a technique that can help one to discover values of the parameters of the system that satisfy certain pre-defined constraints and cause the system to behave in a certain way. Here, we are interested in the length of buffers preventing a buffer overflow and in an optimal timing of the system maximizing the throughput of the system. At first, we used a similar model as for Uppaal for which the analysis did not finish. So, we started to go from the simple building blocks of the system.

We created a simple parametric model of the FIFO queue with three parameters: the maximal length of the queue $FIFOsize$, the rate of incoming request uh_time, and the rate of reading data from the queue $read\_time$. At first, we asked what values the buffer $UHFIFO$ overflows for. We get the following results: If $uh\_time \geq read\_time$, the queue never overflows, and the length of the buffer is not important. If $uh\_time < read\_time$, the analysis does not finish. After setting the initial size of the queue $FIFOsize = 3$, we found that the buffer overflows if $4 * read\_time = uh\_time$. Then, we asked how many packets are accepted until the first one is dropped.

In the future, we plan to apply TReX to more specific parts of the Scampi system abstracted in a suitable way and we also intend to experiment with alternative (perhaps newly developed or improved) symbolic representations in TReX (e.g., based on parameterized intervals).

Acknowledgements. The work was supported by the FP5 projects No. IST-2001-32603 and IST-2001-32404, the CESNET activity “Programmable hardware”, and the Czech Grant Agency project No. 102/04/0780.

References

1. A. Bouajjani, A. Collomb-Annichini, and M. Sighireanu. TReX: A tool for Reachability Analysis of Complex Systems. In *Proc. of CAV’01*, *LNCS* 2102, 2001. Springer-Verlag.
2. J. Holeček, T. Kratochvíla, V. Rehák, D. Šafránek, and P. Simeček. How to Formalize FPGA Hardware Design. Technical Report 4/2004, CESNET, October 2004.
3. P. Matoušek, A. Smrčka, and T. Vojnar. High-Level Modeling, Analysis, and Verification of Scampi2. Technical report, CESNET, 2005. To appear.
4. J. Novotný, O. Fučík, and D. Antoš. Project of IPv6 Router with FPGA Hardware Accelerator. In *Proc. of FPL’03*, *LNCS* 2778, 2003. Springer-Verlag.
5. P. Pettersson and K.G. Larsen. Uppaal2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.