Abstract model verification of the Lookup processor

Aleš Smrčka *
smrcka@fit.vutbr.cz

Abstract: This paper describes the abstract model verification of the Lookup processor and its neighborhood. Lookup processor is the one component of the COMBO-2XFP card which is developed by the Scampi project (the part of a network adapter for getting statistic of transfered IP packets). The main part of this paper takes a description of system model creation and description of chosen abstractions with their reasons and limitations. The models are created with respect to the performance checking. There are also results from verification of the entire abstract model.

Keywords: Scampi, abstract model, verification.

1 Introduction

Scampi is a European project to develop a scalable monitoring platform for the Internet. It also aims to promote the use of monitoring tools for improving services and technology. Scampi project develops a network adapter with extensible monitoring architecture to support a secure and programmable shared monitoring infrastructure. Scampi is also known as the name for the system design of the network adapter - the Scampi phase 2 is the network adapter at 10Gbps speed, which uses two add-on cards - COMBO-2XFP and COMBO6 (see [1]).
For the checking of the correctness of the Scampi design there is used not only simulation and testing phases but also the way of the formal verification. There are two approaches of verification of hardware components, which uses model checking method – *abstract model verification* and *verification of implementation*. The goal of the abstract model verification is to check the performance, weakness and the general properties of the system design. It is based on manual creation of the abstract model with respect to the verified properties. This approach often covers behavior of a bigger component or few connected components of the system. The second approach verifies the properties of the implementation written by hardware developers. Because it often uses unchanged bit representation (i.e. without abstraction on variables), it can cover only behavior of one small components. If we want to know the behavior of the entire system, we have to use both high- and low-level approaches. This paper gives looks only to the COMBO-2XFP card of the system design. In the first chapter we will closer look at the design of searching engine. The second chapter introduces the approach to the abstract model verification of Lookup processor and consists of the sections considering the description of model of each component in the design.

2 An Overview of the Scampi System

The input for the Scampi network adapter is the Ethernet at 10Gbps speed which is connected to the XFMII interface connected with COMBO-2XFP card (see Figure 1). In order to guarantee the speed of the adapter, there are four ways working in parallel. Every

* Faculty of Information Technology BUT, Božetěchova 2, 612 66 Brno

way consists of several executive components and several buffers (in order to prevent the information throwing away with regard to the next busy executive component).
When an IP packet comes, it is forwarded to one of the ways, affiliated with the time stamp generated by Time Stamp Unit (TSU). Whole packets wait in IO_BUF buffer (which is fifo) for processing by Header Field Extractor unit (HFE). This component divide packet into two parts – header and the body of the packet. The IP header is translated into the unified header – an aligned structure storing the main information about the packet including the source IP address, destination IP address, MAC addresses, port numbers, VLAN id, etc. Body of the packet is stored into Packet FIFO (PFIFO) and flows by another way in the system design.

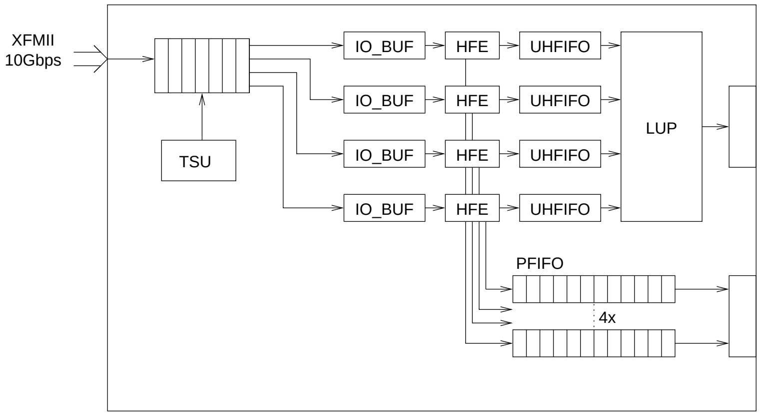

Fig. 1: XFP part of the Scampi design

2.1 Lookup Processor

Lookup processor (LUP) is the core of the Scampi design, it recognizes the packets according to the IP headers. LUP has four inputs from buffer for the unified headers (UHFIFO) and one output consisting the identification of the packet and the type of the packet (as a result of the recognition process). Every recognizing rule is encoded into two parts. The first part is for packet matching (parallel searching) encoded into the TCAM memory (Content Address Memory), the second are instructions for sequential searching via Processing Units (PU) stored in the SSRAM memory. The translation from the router/firewall recognizing rules into the content of the TCAM and SSRAM memory is beyond the scope of this paper.

2.2 Packet Type Recognition and Description of Components

Recognition process divides the unified header to the part for the matching (let us call this part MP) and the part for the sequential searching (SP part). Unified header has the size of 1024 bits, the part for the parallel searching has the size of 256 bits. CAM block sequentially loads MP from all four inputs (UHFIFOs) and prepares the block of the data into the internal FIFO for the TCAM. This FIFO has two parts (which de facto works like two FIFOs) – one for the loading process and one for the TCAM searching. When the loading and searching process is completed, these two parts of FIFO mutually switches. The loading process continues with the new data to the other FIFO.

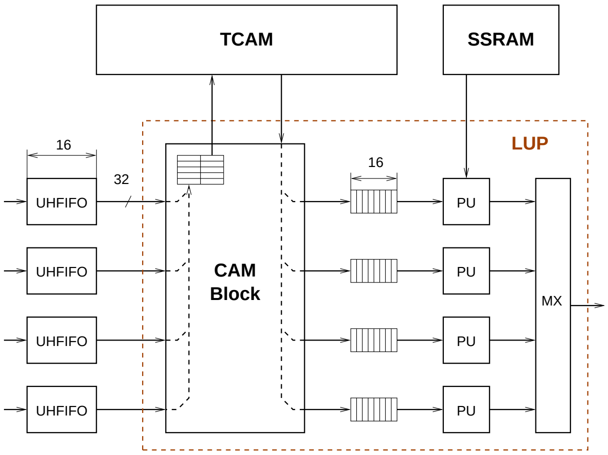

Fig. 2: LUP structure and neighborhood

UHFIFO parameters: UHFIFO can store up to 16 unified headers. Every header has the size of 1024 bits.
Timing of the loading matching part to the FIFO: CAM block loads at one time slot from two UHFIFOs, 32 bits from each. It loads 32 bits from two inputs twice, then turns to other two inputs i.e. 2x64 bits during two time slots. While loading of data from two inputs (UHFIFOs) the others have to wait. Because we want to load 256 bits from every UHFIFO, it takes 16 time slots.

TCAM compares the data, stored in appropriate FIFO part, according to the encoded recognizing rules. This is done simultaneously for all four inputs. Every way has one UHFIFO, one FIFO of results and one PU, so the result from TCAM is stored into the FIFOs of results at the same way the appropriate MP has been loaded from.
Timing of the TCAM searching: At first TCAM loads the data for matching, which takes 16 time slots. Then TCAM searches the four parts at four 12 time slots. The result for every part is sequentially produced every 12 time slots. Loading, matching and storing results take 64 time slots together.

Sequential searching is done by Processing Unit (PU). It loads the result from the matching process and uses it as the entry point of the sequential program in the SSRAM. Once it has the address of an instruction, it loads the instruction. Every instruction is compare & conditional jump instruction except one, finishing, instruction carrying result. We can imagine recognizing rules (encoded into instruction) like a tree where every node is branching instruction and leaves are finishing instructions carrying the result. This result is the result from the recognizing process of the packet.
PU timing: Loading the next instruction takes 4 time slots. Precise protocol of accessing all PUs to the SSRAM memory ensures mutual exclusion and it is transparent (see [2]). Executing of an instruction takes another 4 time slots. There must be at least one finishing instruction, supported maximum length of the program are 8 instructions. Sequential search takes at least 8 time slots (one instruction) and 64 time slots at most (8 instructions).

The last part of the Lookup processor is the multiplexer (MX) which serializes the results from every Processing unit. It simultaneously every single time slot asks every PU for the result and with its presence MX forwards the result to the output of the LUP.

3 Abstract Model of the LUP

This chapter describes the process of creating an abstract model of Lookup processor, UHFIFO buffer and TCAM memory (all components at Figure 2 except PFIFO and SSRAM). The system design is modeled using time automata [3]. While creating of the model it has been considered that the purpose of it is only for the system performance checking.
Every component of the model is described in its section. We have applied several abstractions with respect to the state explosion problem and time of property verification, thus the model checking can be done in moderate time. These abstractions, which we discuss bellow, are tuned with our main interest which is to obtain some performance properties.

3.1 Generator

All components in the COMBO-2XFP card works at 50MHz, only distribution component together with TSU works at 200MHz. The generator is the model of the neighborhood which generates the unified headers and sends them to the UHFIFO.

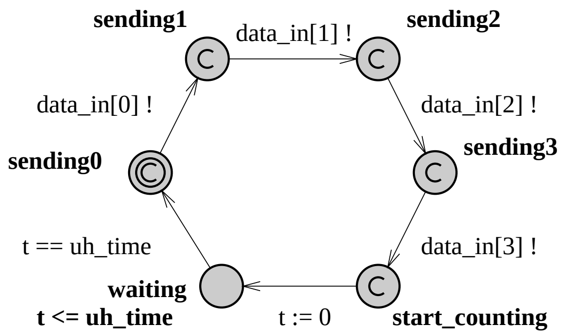

Fig. 3: Unified header generator

Because we are checking the performance of the system, generator sends unified header one by one with no delay. Unified header has size of 1024 bits. The data bus between HFE and UHFIFO has 32 bit length. Every unified header is sent during $1024/32 = 32$ time slots (i.e. every 640ns according to 50MHz). Sending the data is modeled by synchronizing with model of UHFIFO via four channels data_in[0]-data_in[3]. The committed states are used to decrease the state space size.

3.2 Unified Header FIFO (UHFIFO)

Component UHFIFO is storage for unified headers (coming from Generator) remaining here during entire searching process. When the result comes from the LUP, appropriate unified header is released.
UHFIFO can store up to 16 items, every item has length of 1024 bits plus 1 bit indicated whether the information is valid or not. We can not cover such huge state space so we choose to remember only information how many items are stored. We do not need to store the content of the unified header, because we do not model the behavior of the searching through the recognizing rules. We also assume that the system is full of data so we do not need to store the valid flag of the item stored in UHFIFO. The time automaton has four

states, three of them indicating the state of the UHFIFO – *empty*, *nonempty* and $full$. State *throwing* is an observer to check if the UHFIFO throws away the incoming unified header.

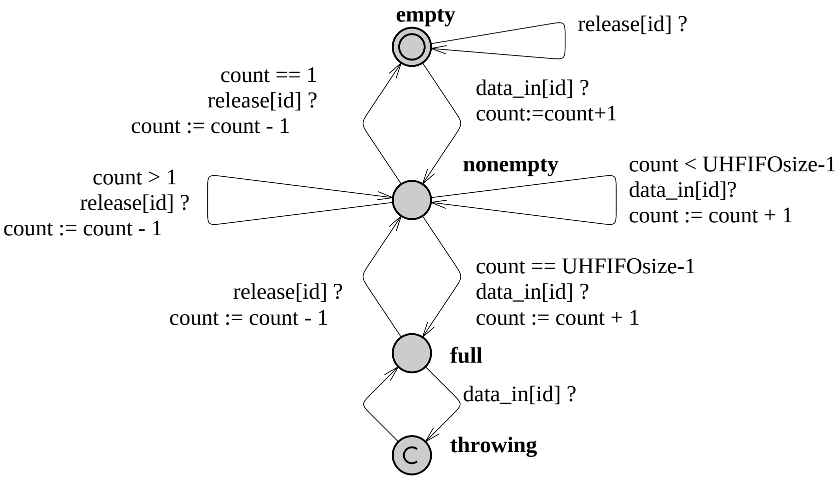

Fig. 4: Unified header FIFO

This model is a template for creation of 4 instances with $id \in \{0, 1, 2, 3\}$. There is one variable *count* which stores the count of the stored items ($UHFIFOsize$ is a constant set to 16). Transitions on the right side of the model are filling the component via channel *data_in*. When the release command comes (via channel *release*), one item is released from the storage (transitions on the left side).

3.3 CAM Block and TCAM

Here is only one common model for the CAM block and the TCAM memory. Note that we are only interested in delays of loading and matching four input parts. Due to the transparency of the protocol for filling and switching the FIFO blocks inside the CAM block, we exclude the entire FIFO block and model only the delay to load and match the data. The simple analysis about comparing the delays of CAM block and TCAM points to the fact that the timing of the TCAM causes the delay of these two components.
The model has two waiting states: $filling$ when the data from UHFIFO is loaded to the FIFO block simultaneously with filling the TCAM from the FIFO block (constant $FILL$ is set to 16 time slots), and *matching_result* state which waits for one result from TCAM (constant $RESULT$ is set to 12 time slots). The result is subsequently sent via channel $rfifo\_in$ to the next component. When the fourth result is forwarded, CAM block and TCAM continues loading the new four parts of unified header.

3.4 FIFO of Results (RFIFO)

This components serves like a buffer for the result from the TCAM and waits for the Processing Unit to send its new result when it is not busy.
The model is very similar to the model of the UHFIFO except the communicating channels (compare Figures 4 and 6). The filling channel is $rfifo\_in$ (from the CAM model), the releasing channel is $rfifo\_out$ (to the PU model). The constant $RFIFOsize$ is set to 16 items.

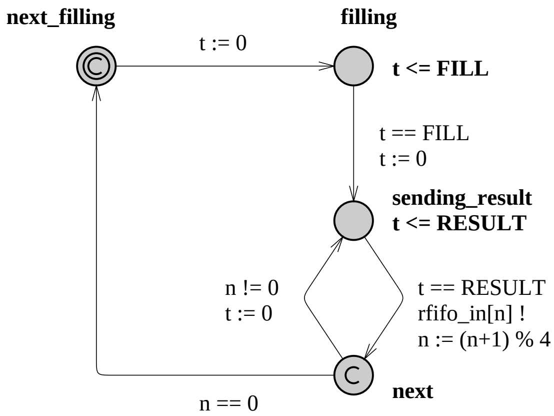

Fig. 5: CAM block and TCAM model

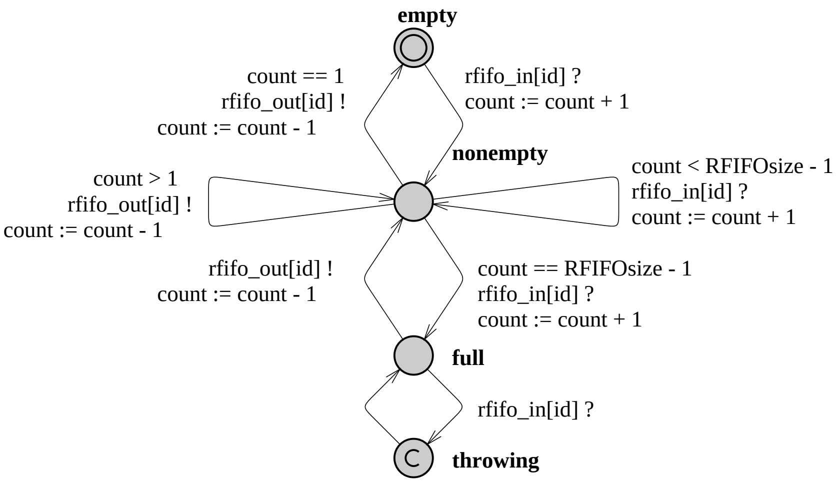

Fig. 6: Model of RFIFO

3.5 Processing Unit (PU)

Processing units are the executive components for the sequential searching the appropriate routing (in the meaning of recognizing) rule. The process is done by executing the instructions stored in the SSRAM memory. There is a simple protocol based on the mutual rotation for accessing the four PUs to the one SSRAM ([2]). Because the timing of this protocol is based on delay with only one time slot, there is too much time slots for access to the SSRAM memory between two bigger events in the entire model of the system, which causes growing its state space. At this component we abstract from this protocol and claim than there can never be the situation that two of four Processing Units access to the SSRAM memory to load next instruction – when PU wants the next instruction, it obtains the instruction. We also abstract from the dividing the delay for the loading and executing few instructions to only one delay (constant *seqsearch*) counted from the multiplication of the maximum supported instructions (8 instructions) with delay for loading and executing of the instruction (8 time slots).

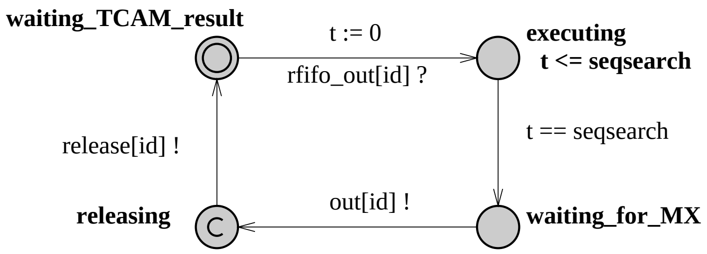

Fig. 7: Template model of the Processing Unit

3.6 Multiplexer (MX)

As discussed before, MX waits only one time slot to check next PU whether it has a result or not. There is many events of checking the result between two forwarding of them, so the same problem happens like the problem with the SSRAM accessing protocol in the model of the PU. So we assume that we have the multiplexer which forward any result immediately.
The model in Figure 8 has an observer (state *slow*) which serves to the throughput checking of the system design. The throughput checking is done by the counting the delay between a number of forwarded results (we call them $Wave$ of results). At the model, the multiplexer waits for the first result and that reset the timer of delay and the counter of the $Wave$ (variable n). The state *next_or_last* is the branching state whether the $Wave$ is complete or not. When the $Wave$ of the results is complete, the transitions after the state *last_result* distinguish if the $Wave$ is forwarded in the time which corresponds to average delay for one result (*result* constant). The $Wave$ constant was chosen to 1024 results.

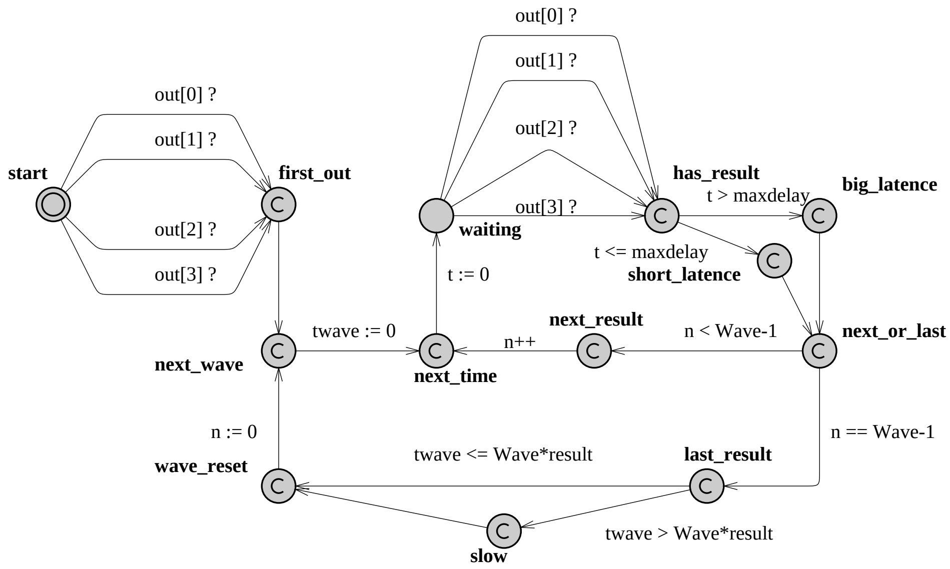

Fig. 8: Model of the multiplexer

3.7 Results from the Model Checking

This section discuss verified properties (the model checker Uppaal [4] was used). They are written as CTL invariance formulas (shortly for the semantics: A means over all paths, [] means always).

- `A[] not deadlock`
  There is never deadlock in the system, wich means that the system will always work. Model satisfies this property.
- `A[] not (UHFIFO0.throwing or UHFIFO1.throwing or UHFIFO2.throwing or UHFIFO3.throwing)`
  This property shows us whether the unified header is ever thrown away from the UHFIFO. Model not satisfies it which means that the input stream is faster than the output stream (this is the first result of the performance checking).
- `A[] not MX.slow`
  This property expresses the throughput checking that the $Wave$ of the results from the multiplexer is sent at time corresponding to the average delay for the one result. This property was satisfied for the constant set at least to 16 time slots (which is 320ns at 50MHz). Today, system design supports IP packet with the length between 64B and 2kB. If the input for the whole system are the smallest packets (64B), theoretically the throughput can be 1.6Gbps at most (64*8/320ns).

4 Conclusion

Verification of the behavior of the Lookup processor and its close neighborhood is the first step of the abstract model verification in the Scampi project. Hardware developers wanted to know the minimal throughput of the Lookup processor and this case study gives them first result. The goal of the Scampi project is to create the network adapter working at 10Gbps speed, so the given results points to the bottle-neck of the system design.
Creating an abstract model of the Lookup processor is the first step for the high-level verification. We plan to use this model to add other neighboring components to it and extend the behavior description towards entire system design. The other extension of the model is the nondeterministic input of the unified header.

Acknowledgement. *This work has been supported by the Grant Agency of the Czech Republic under the contracts 102/04/0780, by the FP5 projects No IST-2001-32603 and IST 2001-32404, and by the CESNET activity "Programmable hardware".*

Bibliography

1. *Description of COMBO cards.* Description of COMBO cards WWW Page, http://www.liberouter.org/hardware.php, 2005.
2. Jan Kořenek: *Project of IPv6 Router with FPGA Hardware Accelerator.* Diploma thesis, FIT VUT Brno, 2003.
3. Rajeev Alur: *Timed Automata.* NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems, A revised and shorter version appears in 11th International Conference on Computer-Aided Verification, LNCS 1633, p. 8–22, Springer-Verlag, 1999.
4. *Uppaal – real-time system model checker.* Uppaal WWW Pages, http://www.uppaal.com