TOWARDS HARDWARE VERIFICATION

Aleš SMRČKA, Doctoral Degree Programme (2)
Dept. of Intelligent systems, FIT, BUT
E-mail: smrcka@fit.vutbr.cz

Supervised by: Prof. Milan Češka, Dr. Tomáš Vojnar

ABSTRACT

This paper gives an introduction to the approach of verification of discrete timed hardware design. It roughly offers low-level verification hand in hand with high-level verification (which is described in more detail). The main part of this paper takes an example of the top of view verification of the lookup processor as a part of the COMBO cards developed by Liberouter project.

1 INTRODUCTION

There is no need to recall the importance of the verification but it is necessary to bring back the knowledge that there is still the main verification problem – state space explosion. By increasing of complexity of the hardware design the verificators (people administrating the verification process) are forced to use various methods to reduce this problem.

There are two main purposes of the verification. The first is to find out the properties which satisfy the system design. The second purpose serves to accelerate the development to find out any bugs in the implementation of the system during the earliest phase of the development. If we look inside the first purpose there are different types of properties we want to check. One type are the properties which express the correctness of the system. The others express properties we want to know about the system (e.g. performance of the system). This leads towards two approaches of the verification discussed in the first chapter.

The second chapter is the overview of the Liberouter project as an introduction to the next chapter describing the abstract model verification of one component. It is the first step of verification the whole design of Liberouter project and servers as the base for next verification including other adjacent components.

2 TWO APROACHES OF THE VERIFICATION

This chapter offers the approaches of the formal verification. It consists of two phases – high-level and low-level verification.

The first phase looks at the whole design and considers its main idea. Verificators make the abstract model of the part of system or of the whole system and verify its interesting properties. We call the first phase “Abstract model verification”.

The low-level verification looks into the source codes of the designed system and verifies the correctness of the system according to them. We call this second phase “Implementation verificaton”.

2.1 ABSTRACT MODEL VERIFICATION

The second phase looks from top of view on the whole system or on some bigger parts where are checked the interrelation reactions. This phase consists of two main steps.

In the first step there is a need to create an abstract model of system. It is often not possible to describe common behaviour by both accurate (in the meaning of system behaviour) and satisfactory (in the meaning of the state space explosion) way and therefore it is necessary to make abstract model more simple.

The second step is either an analysis – in cases of the very simple claims – or automatic verification by means of created abstract model.

2.2 IMPLEMENTATION VERIFICATION

The most important condition to use the verification of the complex system is to think by modular way already during design phase. Every module must have explicitly described outputs as a reaction on particular inputs. In these cases of design it is possible to eliminate the explosion problem at state space exploration of the whole system to the verification problem of each module. This method allows to use the source codes like VHDL as a model to be verified. Unfortunately for practical use, VHDL is not an input language of any available model checker so it is necessary to translate to other language. There is sucessful use of translation from VHDL to SMV (see [1]) in the Liberouter project [3].

Nevertheless there are only a few cases when the verification can be done automatically by machine without human modifications. One of the methods to let verification go is usage of variables abstraction. For example, one module uses 10 bit address bus and model checker has to explore all states with all different 1024 values of address which causes a huge state space while use of 32 values does not. If smaller width of the bus is not critical for the verified property then the verificator can modify the model to use only 5 bit address bus. We will not go into the details of used methods (reader can find one of them in [2]).

3 THE CORE OF THE LIBEROUTER

The aim of the Liberouter [3] project is to design and develop the hardware accelerated router. The most important part of this project is to develop the COMBO6 hardware accelerator card allowing hardware to route the most of IPv6 traffic of a gigabit ethernet.

This chapter roughly describes decision function of routing and firewalling in the router. This is done by descriptions of different phases and components affecting the life of the packet in the system.

Router has 4 input and output ethernet ports. When IP packet comes through one of them it is stored to the one of four input buffer (appropriate to the ethernet port). If this buffer is full packet is thrown away because there is no space for it. It is good to note that the minimal size of an IP packet is 64 bytes, maximum size of a packet can be 64kB but only approximately 2kB packets are currently supported.

Header field extractor (HFE) is the block which loads packet from the input buffer and extracts its header and data which it carries. From this point on the body of the packet and the headers continue by two different ways. Data body of the packet is sent to the storage (DRAM). Meanwhile the packet header is parsed and from information which it carries is created a structure called "unified header". This is the fixed structure containing aligned information (source and destination IP address, MACs, VLAN id, protocol type, port numbers etc.). HFE also detects errors in the packet and stores this information into the unified header. Unified header is passed to the UHFIFO.

UHFIFO is the first-in-first-out queue which stores unified headers for the next process.

Lookup processor (LUP) performs a lookup in the routing and firewall tables. Input for LUP is a unified header subsequently loaded from the four UHFIFOs. LUP output is a record, that controls packet processing in the following components what – record carries information what must be done with packet. Lookup processor consists of two parts - CAM block and processing unit (PU). Matching the header according to the routing and firewalling rules is done by TCAM (Content Addressable Memory) and sequential search done by PU. Since the LUP has four independent inputs (UHFIFO) and one output way the main interest is to find out, whether it can manage all requests (unified headers from all inputs) in real-time. This property is discussed in the next chapter.

Further components with their descriptions are beyond the scope of this paper. For this paper has the biggest importance the record from the LUP.

4 LOOKUP PROCESSOR VERIFICATION

This chapter describes more detailed function and the abstract model verification of the UHFIFO, Lookup processor and TCAM memory working concurrently. System is modelled using timed automata [5] and verified by Uppaal [6]. Every component and inside block is described by timed automata communicating via synchronizating channels, there is also use of variables. Every timing is synchronous and each delay is modelled by two states. System holds at the first state until the delay reaches wanted value (the state has invariant like "t <= delay"). There is a transition from this state to the second one with guard "t == delay". Thus, this fact induces that verification of this type of model is a discrete-time systems so for the further work it can be considered to verify it not like a real-time model.

We are only interested in maximal speed of these interactive components. Let's consider that unified headers are incoming one by one at maximum speed. Size of the unified header is 512 bits, data bus between header field extractor and UHFIFO has length of 32 bits and all components work at frequence 50Mhz. One unified header is stored into the UHFIFO every 320ns (512/32/50Mhz). We will consider only whether FIFO is full or not because of simplicity. Size of UHFIFO is 16 items, there is an interger variable keeping the number of actual stored items. Automaton of packet generator subsequently sends new packet to each instance of UHFIFO every 320ns. UHFIFO has size of 16 items. The automaton has only 3 states – empty, nonempty and full (see 1). State “throwing” is only for satisfaction of the property which is being checked (see Verified properties bellow).

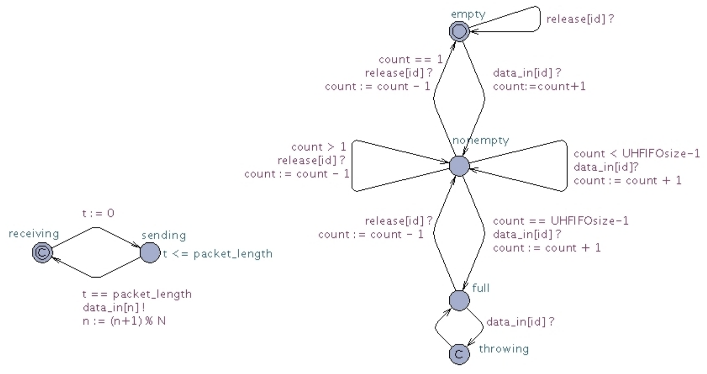

Figure 1: Generator and UHFIFO

Synchronously CAM block loads a part of the unified header in 16 time slots (LOADING - see 2) (every time slot is 20 ns due to 50Mhz frequence) to match it with the TCAM memory. At this time CAM block loads parts from all 4 UHFIFOs and prepares them for loading into the TCAM memory. Whole unified header still takes place in the UHFIFO. Matching in the TCAM memory holds for 12 time slots for every unified header (RESULT delay). The four results from the matching procedure are subsequently stored to the appropriate four buffers (FIFO-of-results with 16 items).

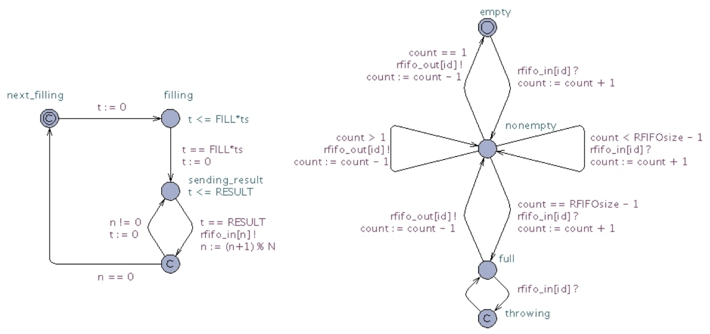

Figure 2: CAM, TCAM and FIFO of results

Processing unit is sequential processor that uses the result in the FIFO-of-results as an initial address of the program for sequential search in the SSRAM memory. Execution of every instruction holds for 8 time slots. Routing and firewalling rules are translated into this program with at most 8 instructions (i.e. delay in the PU for one sequential search is at most 64 time slots = “seqsearch” – see 3). Sequential search uses concrete matching and conditional jump instruction. While executing these instructions PU uses data from the UHFIFO. When sequential search is done the result (which carries the information what to do with the packet) can be produced out of this system and whole unified header can be now released from UHFIFO. At this place the four way system is sequentially processed to the one way system. This is done using multiplexor.

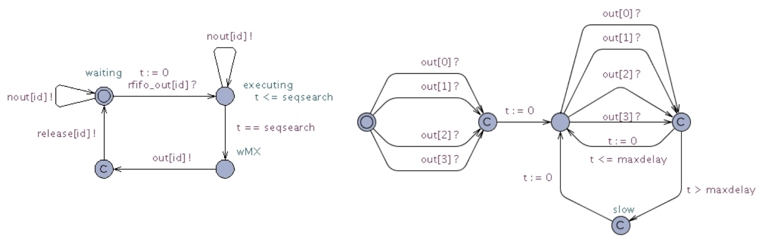

Figure 3: Processing unit and Multiplexor

Next part discuss verified properties. They are written as CTL formulas. Model checking of all of them needs at most 1 second of running time.

- A[] not deadlock = there is never deadlock in the system – satisfied
- A[] not (UHFIFO0.throwing or UHFIFO1.throwing or UHFIFO2.throwing or UHFIFO3.throwing) = packet is never thrown away from the UHFIFO – satisfied
- A[] not MX.slow = maximum delay between two packets outgoing from the multiplexor is at most “maxdelay”. Constant maxdelay is set to 560ns. The bottleneck of this system is the CAM block with long loading and matching delays. The result 560ns may not be accurate due to the timings in the multiplexor. Real result differs between 560 - 600ns.

5 CONCLUSION

This paper describes the first overview of the case study of high-level verification in the Liberouter project. The results bring designers the first and useful analysis about the Lookup processor although it is modelled only for monotonous input – unified headers one by one without any delay between them. This case modelled the situation of the input full of small packets (e.g. 64 bytes).

For the next research it can be considered different nondeterministic input or to integrate to the model other components of the system design. Because of usage of the discrete-time system we will also consider to use another tool for the discrete systems than Uppaal.

REFERENCES

[1] Řehák V., Kratochvíla T., Šimeček P.: Verification of COMBO6 VHDL Design, Technical report number 17/2003, CESNET, 2003

[2] Řehák V., Kratochvíla T.: Verification of FIFO BRAM, Technical report, http://www.liberouter.org/formal_verification/work/fifo_bram5_ver.html

[3] Liberouter, Liberouter Project WWW Pages, http://www.liberouter.org/

[4] CVS repository of Liberouter project at /liberouter/vhdl_design/units, http://www.liberouter.org/cgi-bin2/cvsweb.cgi/

[5] Alur R.: Timed Automata, NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems, LNCS 1633, p. 8-22, Springer-Verlag, 1999

[6] Uppaal – real-time system model checker, http://www.uppaal.com/