Functional Verification of an Embedded Network Component by Co-Simulation with a Real Network

R. Paško, R. Cmar, P. Schaumont and S. Vernalde
IMEC, Kapeldreef 75, B3001 Leuven, Belgium
pasko@imec.be

Abstract

In this paper, we propose a technique for verification of the functionality of a hardware networking component by including an existing real-world network into the simulation loop. As a consequence, there is no need for a high-level network model to create the system simulation. Instead, third party hardware/software can be used for the cross-checking of the design's functionality. The technique is most suitable for C/C++ based design methodologies which can directly access the operating system (OS) network interface functions. Because of that, the integration of a real network into the simulation loop can be straightforward. We will demonstrate the method on verification of a Hypertext Transfer Protocol (HTTP) hardware implementation used in a design of an embedded web-camera with direct Internet connectivity.

1 Introduction

The explosive growth of Internet creates a huge demand for various network appliances in diverse areas ranging from packet routing to data processing. Computer based software (SW) solutions, though common and easy realised, do not necessarily offer the best performance/power consumption figures. Often, some particular tasks can be more efficiently handled by hardware (HW) or HW/SW embedded components serving either as accelerators (e.g. for fast encryption/decryption) or even as stand-alone applications (the above mentioned web-camera). Another impulse for using embedded HW/SW solutions comes from the world of reconfigurable devices. Current FPGA's or PLD's can provide re-programmable HW elements with almost ASIC performance, while delivering enough capacity for even complex applications. This significantly improves the device flexibility which is one of the important requirements, for a networking appliance, today. At the same time, it makes questionable one of the most frequent arguments for the SW implementation which is the flexibility issue.

A design of such embedded network appliance in HW (HW/SW) requires a careful verification strategy, since the real network behaviour is very complex and non-deterministic. This makes the common HW verification strategy, based on use of high level models describing the non-implemented parts of the system, somewhat questionable. Since the network protocols are mostly written in SW, an interesting possibility would be the use of existing SW models. For example, a verification of the functionality of a HW Internet Protocol (IP) implementation by co-simulation with its SW model is definitely something to look about. There exists a gradual convergence between the SW and HW design techniques in the last years. However, porting of such complex SW models into a HW simulation would still require a significant additional effort.

We faced these problems during the design of a stand-alone embedded camera directly accessible from Internet. The network protocol layers, apart from the Ethernet layer, as well as the image processing, were implemented as HW elements in FPGA, as shown in Fig. 1. To verify the designed network elements functionality, we have used a heterogeneous simulation setup plugging the simulated entities into a real-network environment. This approach has two significant advantages. First, we avoided the necessity of writing good high-level models of network protocols, and second, the porting of complex SW models into the HW simulation was not needed as well.

There are similar techniques used for HW verification and rapid prototyping [1]. However, these are usually constrained to generation of real-data test-benches, or evaluation of some non-deterministic elements like transmission channels. The proposed technique is a one step beyond of such approaches. The simulation actually creates an independent SW element in the real system, from which it is taking stimuli, and in response effecting the system as well. We can consider it as a SW equivalent of a HW emulation with all the apparent advantages, like shorter design time, more flexible system environment or cheaper realisation. The approach is most suitable for C/C++ based design methodologies, thanks to the flexibility and speed of-

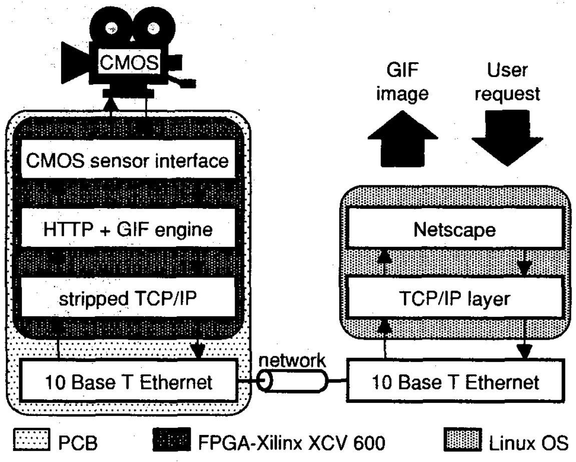


Figure 1. Stand-alone Internet camera with embedded networking

fered by a high-level programming language. It can be an interesting addition in testing strategies using C++ based methodologies, like the TestBuilder library [2]. We have used a novel version of the *OCAPI* design environment [3] called *OCAPI-XL*, however, all C++ based design methodologies, e.g. SystemC [4], provide the features necessary to implement such scheme.

2 Heterogeneous Simulation with Real HW/SW Elements

The verification of a designed hardware element from the "system" point of view is usually performed as shown in Fig. 2. The designed entity is gradually refined to a low-level clock-accurate model, while the rest of the system remains described at the high-level. This approach is feasible for typical hardware designs, where all the elements have well-defined behaviour and interfaces (though the functionality might be very complex) and the high-level models are

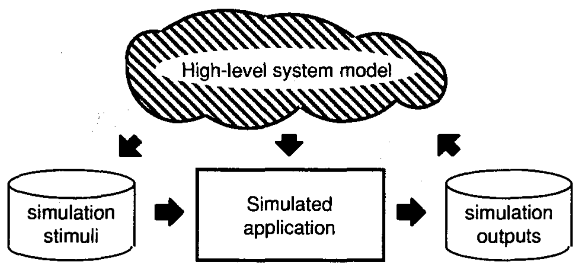


Figure 2. Simulation loop in HW design

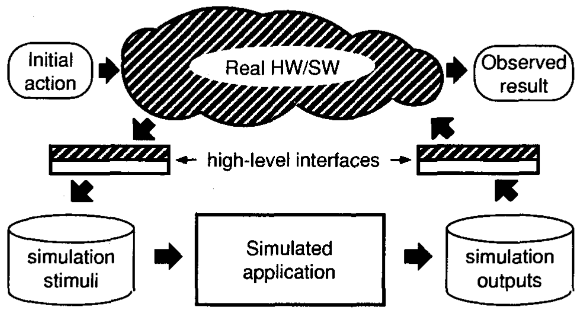


Figure 3. Introduction of a HW simulation into a real system

often needed during the initial stages of the design for profiling or performance modeling.

A typical network environment, however, is too complex and non-deterministic to rely on simulation results featuring some simplified high-level model. The use of some existing complex network simulator would solve the issue of simulation reliability, but on the other hand, an additional effort for incorporating of such element into the HW simulation would have to be spent. The proposed "hybrid" solution provides a reasonable compromise between the above mentioned approaches. At one hand, it provides much more reliable results compared to the self-made simplified model, on the other hand, there is almost no additional effort necessary. The basic idea is shown in Fig. 3. The simulation runs as an independent SW entity, when the inputs and outputs are connected to the rest of the system via high-level interfaces. It can be initiated by normal system activity, e.g. a HTTP request made by a standard WWW browser, while the results can be directly observed in similar way. The method was used for verification of the functionality of Internet Protocol (IP) and Hypertext Transfer Protocol (HTTP) layers (see [5])used in the web-camera design, as shown in Fig. 1.

3 Verification of HTTP and IP Layers Described in *OCAPI-XL*

OCAPI-XL is a C++ library containing classes for a timed multi-thread system description with automated code generation. It supports high-level behavioural objects like processes (threads), semaphores, conditions, messages, etc. The HDL code generator translates these high-level constructs into appropriate low-level elements, like processes into FSMD's or messages in FIFO's.

The complete simulation setup is shown in Fig. 4. The elements, which will be synthesised, must be refined down to a clock-accurate, fixed point description (HTTP or

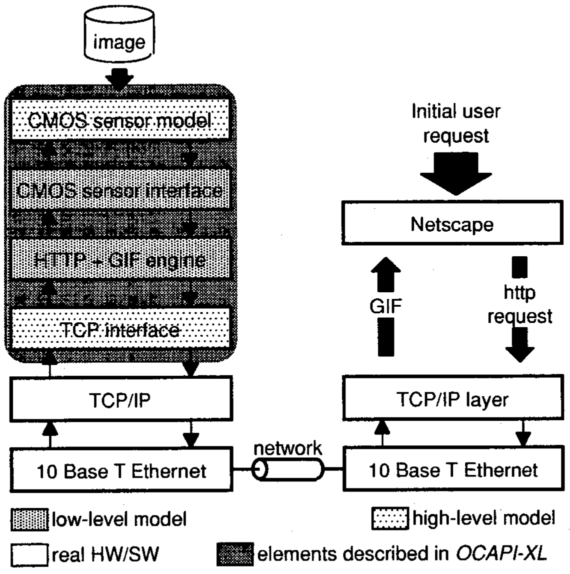
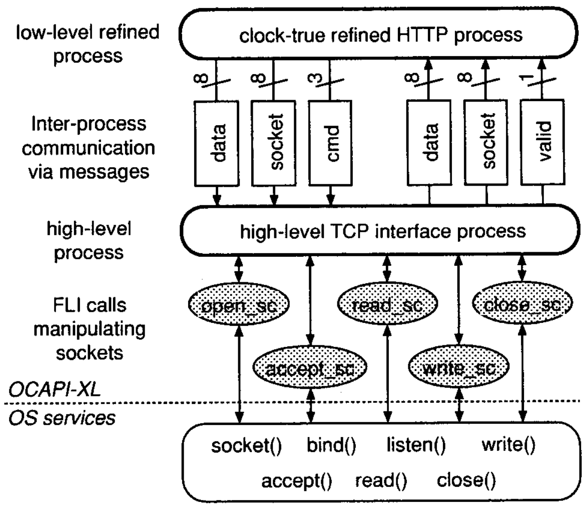


Figure 4. Simulation setup for the HTTP design verification



low-level refined process
clock-true refined HTTP process
8
8
3
8
8
1
Inter-process communication via messages
data
socket
cmd
data
socket
valid
high-level process
high-level TCP interface process
FLI calls manipulating sockets
open_sc
read_sc
close_sc
accept_sc
write_sc
OCAPI-XL
OS services
socket() bind() listen() write()
accept() read() close()

Figure 5. Communication between OS and *OCAPI-XL*

CMOS sensor interface), while others, like the Transmission Control Protocol (TCP) interface, provide just access to the OS services, thus can remain at higher abstraction level. The complete simulation can run in the background as an independent SW process, equivalent to a normal Unix demon, and various scenarios can be tested by generating an appropriate HTTP request. It is also worth mentioning, that the methodology provides natural way of mixing low and high level processes in single simulation, which can be very beneficiary from the speed point of view.

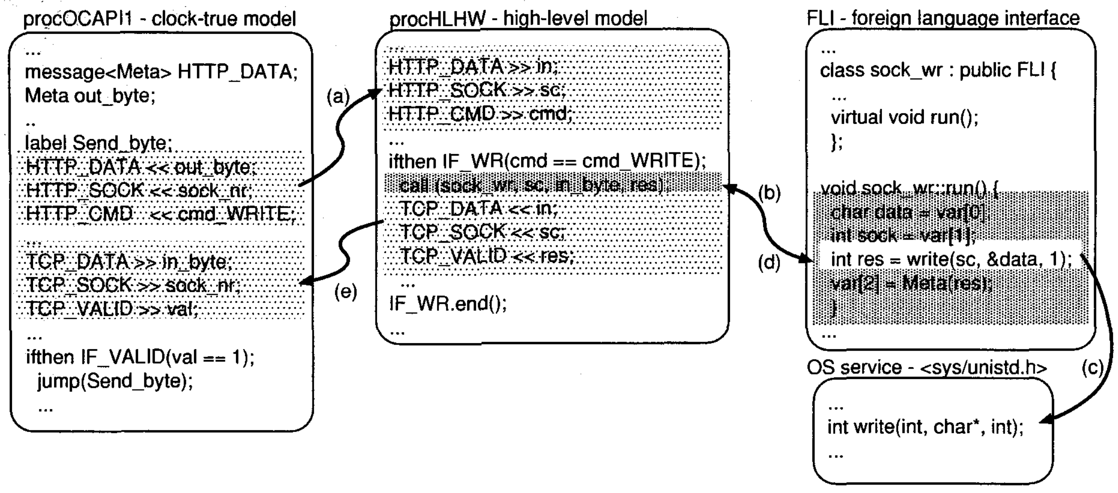
In order to be able to run this sort of heterogeneous simulation, the design methodology must provide some way of accessing and executing foreign code. *OCAPI* provided this feature in form of a data-flow block, while *OCAPI-XL* offers a so called *Foreign Language Interface*, (FLI) call. Other C++ based design methodologies also provide some equivalent sort of service, so this strategy is not *OCAPI-XL* specific. It might be even possible to use it in VHDL or Verilog, thanks to the CLI and PLI interfaces, however, the much higher simulation speed of C/C++ based design methodologies makes it much more suitable for this kind of applications.

During the initialisation phase, root socket connections are created at selected network port, as with normal TCP server [6]. The simulation is initiated by a HTTP request made by an ordinary web browser. It is forwarded to the socket interface, where it is caught by an appropriate FLI call (e.g. *accept_sc* and *read_sc*), and translated via the TCP interface to regular OCAPI-XL stimuli, which triggers the further simulation. The detailed interface between *OCAPI-XL* simulation and OS services is given in Fig. 5. The raw image data for the GIF engine is generated in the same way, i.e. the high-level camera model access the data stored in a file via another FLI call. The resulting output, whether GIF image or an error message, is translated back to the socket interface in the same way and forwarded through the network to the HTTP client, which initiated the connection.

The technique was later used in a similar way to verify an implementation of a IP layer. The only difference was that the IP block was connected to a high-level interface providing a gateway to an Ethernet layer via raw socket interface (see [6]).

4 Heterogeneous Simulation Example in *OCAPI-XL*

An example of the simulation flow is shown in Fig. 6. Let us consider a situation, when the process implementing the HTTP server has a byte ready to send to the TCP layer. It is pushed to a message queue to the TCP interface (Fig. 6(a)). TCP interface process must identify the requested service (i.e. *cmd_WRITE*) to invoke the proper FLI call (Fig. 6(b)). Since the *OCAPI-XL* data types are nor regular C++ types, conversion to standard *int* must be performed at the beginning of the FLI call. During the FLI call execution, a regular OS service *write()*, which writes to a file (since socket connections are also handled as files in UNIX OS) is executed and an integer, indicating suc-


procOCAPI1 - clock-true model
procHLHW - high-level model
FLI - foreign language interface
OS service - <sys/unistd.h>
(a)
(b)
(c)
(d)
(e)

```
...
message<Meta> HTTP_DATA;
Meta out_byte;
..
label Send_byte;
HTTP_DATA << out_byte;
HTTP_SOCK << sock_nr;
HTTP_CMD  << cmd_WRITE;
...
TCP_DATA >> in_byte;
TCP_SOCK >> sock_nr;
TCP_VALID >> val;
...
ifthen IF_VALID(val == 1);
  jump(Send_byte);
  ...
```

```
...
HTTP_DATA >> in;
HTTP_SOCK >> sc;
HTTP_CMD >> cmd;
...
ifthen IF_WR(cmd == cmd_WRITE);
  call (sock_wr, sc, in_byte, res);
  TCP_DATA << in;
  TCP_SOCK << sc;
  TCP_VALID << res;
  ...
IF_WR.end();
...
```

```
...
class sock_wr : public FLI {
  ...
  virtual void run();
  };

void sock_wr::run() {
  char data = var[0];
  int sock = var[1];
  int res = write(sc, &data, 1);
  var[2] = Meta(res);
  }
...
```

```
...
int write(int, char*, int);
...
```

Figure 6. Interaction between OS kernel and *OCAPI-XL* simulation

cess or failure of the operation, is returned back, as shown in Fig. 6(c,d)). This value must be translated back to the *OCAPI-XL* native data type *Meta* and returned to the HTTP process via a message queue (Fig. 6(e)). Based on its value, the HTTP process can take further action, e.g. sent the next byte.

5 Practical Example

As an example of the usefulness of the presented technique, we can give the following example. It relates to a problem discovered by the co-simulation. The TCP connection initiated by a HTTP client is normally closed by the HTTP server to indicate the end of the data stream. However, the connection might be closed due to reasons not related to the HTTP protocol layer, e.g. TCP time-out. This possibility was not taken into account in the first version of the HTTP code and resulted in a server dead-lock (since it waited for an acknowledge from an already closed TCP connection). It was also not discovered during the first simulations, since these were used only to test the interface between HTTP/TCP and did not consider the time-out behaviour of TCP.

6 Conclusions

We have presented a technique for a system level validation of embedded HW/SW networking applications. The idea is to use an existing real network as a part of the simulation loop instead of writing a high-level system model. This way, the functionality of designed element can be cross-verified by the interaction with real network environment, which results in very realistic simulation results. The technique is very suitable for C/C++ based design methodologies thanks to the simple interfacing of the real-network components into the simulation due to the OS support.

References

[1] P. Schaumont, G. Vanmeerbeeck, E. Watzeels, S. Vernalde, M. Engels and I. Bolsens, *A technique for combined virtual prototyping and hardware design*, proc. of Rapid System Prototyping Workshop, pp. 156-161, Leuven, Belgium, June 1998.

[2] TestBuilder, An Open Source Standard for C++ Development of HDL Test Benches, *http://TestBuilder.net/*

[3] P. Schaumont, S. Vernalde, L. Rijnders, M. Engels and I. Bolsens, *A Programming Environment for the Design of Complex High Speed ASICs*, proc. of Design Automation Conference (DAC 1998), pp. 315-320, San Francisco, USA, June 1998.

[4] The Open SystemC Initiative, *http://www.systemc.org*.

[5] R. Stevens, *TCP/IP Illustrated, Volume 1 and 3*, Addison-Wesley, Reading, MA 01867,1994 and 1996.

[6] R. Stevens, *Unix Network Programming, Volume 1*, Prentice Hall, Upper Saddle River, NJ 07458, 1998.