Intellectual Property Re-use in Embedded System Co-design: an Industrial Case Study

E. Filippi[1], L. Lavagno[2], L. Licciardi[1], A. Montanaro[1], M. Paolini[1], R. Passerone[3], M. Sgroi[3], A. Sangiovanni-Vincentelli[3]

1) CSELT - Via Reiss Romoli 274 - 10148 Torino, Italy
2) Politecnico di Torino - Corso Duca degli Abruzzi 24 - 10129 Torino, Italy
3) University of California at Berkeley - 515 Cory Hall - Berkeley, CA 94720, USA

Abstract

Design of large systems on a chip would be infeasible without the capability to flexibly adapt the system architecture to the application and the re-use of existing Intellectual Property (IP). This in turn requires the use of an appropriate methodology for system specification, architecture selection, IP integration and implementation generation. The goals of this work are: a) verification of the effectiveness of the POLIS HW/SW co-design methodology for the design of embedded systems for telecom applications; b) definition of a methodology for integrating system level IP libraries in this HW/SW co-design framework. Methodology evaluations have been carried out through the development of an industrial telecom system design, an ATM node server.

1. Introduction

Two key factors allow fast and reliable implementation of complex embedded systems: the availability of effective methodologies and tools for system HW/SW co-design and a sizable library of reusable and customizable cores, system-level Intellectual Property (IP) modules. Co-simulation and co-synthesis tools allow system engineers to explore different architectures and choose the most efficient design solutions in terms of architecture and final implementation, while the use of customizable IP modules leads to significant improvements of design productivity and reliability.

This paper describes our experience in designing an industrial test case using the POLIS HW/SW co-design methodology for embedded systems [1] and the CSELT *VIP Library*™ [2]. Our main goals have been the assessment of the effectiveness of POLIS for the design of embedded systems for telecom applications, and the definition of a methodology for the integration of system level IP libraries in the POLIS co-design framework. Evaluations have been performed through the development of an industrial telecom system design – an ATM (Asynchronous Transfer Mode) node server prototype. The server is a statistical multiplexing unit that performs support functions needed to implement Virtual Private Networks (VPN) in ATM switching nodes. It includes an intelligent buffer that controls the bandwidth of the outgoing flows using a weighted fair queuing service discipline and implements a message selective discarding technique to avoid node congestion.

The paper is organized as follows. First, a brief description of the POLIS co-design methodology and software tools is given. Then, the CSELT *VIP library*™ is presented, together with details on the integration of the IP library in the co-design framework. The following paragraphs describe the architecture and the implementation of the ATM server. Finally, the results of the design exploration and the outline of the lessons learned from this project are given.

2. The POLIS co-design methodology

POLIS is a co-design environment for synthesis and validation of embedded systems [1]. It assists the designer in manually partitioning the target system into a set of interacting hardware and software modules. After a partition and a suitable architecture are chosen, it allows to synthesize both its hardware and software components and the interfaces between them.

POLIS is based on a formal model of computation that consists of a network of Co-design Finite State Machines (CFSMs). CFSMs are extended finite state machines that communicate asynchronously by means of events. The asynchronous interaction between CFSMs allows to

capture the behavior of systems that consist of hardware and software components.

In the POLIS flow, the behavior of each CFSM is described using ESTEREL [3], a synchronous reactive language with underlying FSM semantics. Analysis at the behavioral level can be carried out either with formal verification tools or by co-simulation. System level co-simulation, performed either in the Ptolemy environment or using a VHDL simulator, allows designers to validate the functionality of the system and evaluate design choices, like HW-SW partitioning, architecture and scheduling selection, at an early stage of the design flow.

Performance evaluation can be carried out by simulating the behavior of the designed architecture with an abstract timing model of the selected processor. In VHDL co-simulation, POLIS produces a behavioral VHDL model of each CFSM where the structure and timing of the model depend on the implementation (HW or SW) selected for the corresponding CFSM. HW models emulate the behavior and timing of the synthesized HW with the specified clock cycle, while SW models [4] emulate the behavior and timing of the software – including the real time operating system overhead – on a specific micro-controller. SW timing is estimated using a timing library that contains the number of clock cycles required by different micro-controllers to execute atomic operations (e.g. assign, test, etc.). At this point, system simulation, taking into account the selected HW/SW partition and scheduling policy, is carried out with a conventional VHDL simulator.

The use of VHDL as a simulation language permits easy integration with existing hardware IP specified in synthesizable VHDL, as we show in the next sections.

After an architecture and a partition are chosen, synthesis takes place. The output of POLIS is the C code of the SW modules and a synthesizable netlist of the HW modules in XNF, VHDL or Verilog format.

3. IP Integration

Designing complex systems with a high performance reusable library achieves two simultaneous goals – shorter time-to-market and lower design cost – making system know-how available in the form of customizable and reliable pre-designed modules.

The CSELT *VIP Library*™ is a library of customizable IP soft cores, written in RTL VHDL that can be mapped onto gate level netlists through automatic synthesis. The library includes telecom application modules, mainly targeted to the design of ATM components, video application modules and general-purpose modules (Figure 1). The synthesizable VHDL source code of the VIP modules is tailored for the Synopsys synthesis tools and can also be used as functional model in simulation. The use of the same input code for simulation and synthesis guarantees design consistency.

The *VIP library*™ has been used in the implementation of ICs and systems for ATM Switching and access nodes, Optical Network Units, Switched Digital Video Broadcast equipment [2]. The integration of the VIP modules in an industrial system or IC design flow usually requires minimum effort from the designer, who can focus his attention on the implementation of specific custom functionality. This saves time not only in the design and simulation phases but also in system integration and testing. Starting from functional and timing specifications, the system architecture is designed and partitioned to use the *VIP library*™ modules wherever possible. After that, the design of custom hardware and software parts and their integration with the chosen *VIP library*™ modules take place.

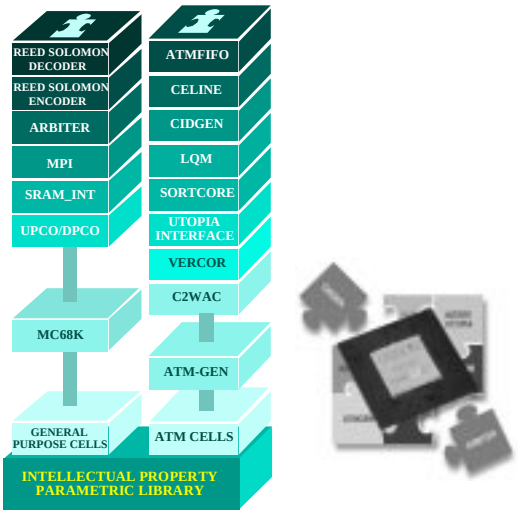

Figure 1: A subset of the *VIP Library*™

Integrating IP library modules and standard components in the POLIS co-design environment has also proven to be an easy – but not trivial – task. When pre-existing IP blocks (as well as memories and off-the-shelf components) do not follow the POLIS event-based communication model, they require handwritten protocol adapters in order to be integrated with HW/SW co-design subsystems. In most cases these interfaces are very simple and efficient. However, the integration of modules that are intrinsically based on rendezvous protocols (like memories) may result in a loss of performance: a more sophisticated communication model between CFSMs and external modules could help in solving this problem. An example of protocol adapter VHDL code is shown in Figure 2.

```
-- events  from the algorithm CFSMs  to the IP modules
QUID <= CONV_UNSIGNED( v_PUSH_QUID, MAXQUEUE )
when e_PUSH_QUID = '1' else
       CONV_UNSIGNED( v_POP_QUID, MAXQUEUE )
when e_POP_QUID = '1' else
       CONV_UNSIGNED( v_QUERY_QUID, MAXQUEUE );
QUID_VALID <= TO_STDULOGIC( e_QUERY_QUID or e_PUSH_QUID or e_POP_QUID );
POP_NPUSH <= TO_STDULOGIC( e_QUERY_QUID or e_POP_QUID );

-- events  from the IP modules  to the algorithm CFSMs
e_CID <= TO_BIT(CID_MATCH);
v_CID <= CONV_INTEGER(UNSIGNED(CID));
e_QUEUE_STATUS <= TO_BIT(IS_A_PUSH_QUERY and QUESTAT_VALID);
v_QUEUE_STATUS <= CONV_INTEGER(UNSIGNED(QUESTAT));
e_FIFO_OCCUPATION <= TO_BIT(IS_A_POP and QUESTAT_VALID);
v_FIFO_OCCUPATION <= CONV_INTEGER(UNSIGNED(QUESTAT));

FSM_COMB: process (C_STATE, e_POP_QUID, e_PUSH_QUID, e_QUERY_QUID,
QUESTAT_VALID)
begin
   IS_A_POP <= '0'; IS_A_PUSH_QUERY <= '0';
   case C_STATE is
   when IDLE =>
      if (e_POP_QUID='1') then F_STATE <= POP;
      elsif (e_PUSH_QUID or e_QUERY_QUID)='1'
         then F_STATE <= PUSH_QUERY;
      else F_STATE <= IDLE; end if;
   when POP =>
      IS_A_POP <= '1';
      if (QUESTAT_VALID='1') then F_STATE <= IDLE;
      else F_STATE <= POP; end if;
   when PUSH_QUERY =>
      IS_A_PUSH_QUERY <= '1';
      if (QUESTAT_VALID='1') then F_STATE <= IDLE;
      else F_STATE <=PUSH_QUERY; end if;
   end case;
end process FSM_COMB;

FSM_SEQ: process (CLK, N_RST)
begin
   if N_RST = '0' then C_STATE <= IDLE;
   elsif (CLK'event and CLK = '1') then C_STATE <= F_STATE;
   end if;
end process FSM_SEQ;
```

Figure 2: Example of protocol adapter VHDL code

4. The target design: ATM Virtual Private Network server

Our target system is a server that supports ATM-based Virtual Private Networks (A-VPNs) in ATM nodes. It is a real-life telecom system: a first release of the server that was implemented and tested in CSELT [6] required a design effort of approximately 3 man-years.

The A-VPN service [5] provides a user having several sites on a geographical area the means for interconnecting these sites via ATM. At every location, the user has access to a Virtual Path Connection (VPC) with a reserved bandwidth carrying several Virtual Circuit Connections (VCCs), each of which corresponds to a destination in a logical mesh configuration.

The A-VPN server is a network element that controls the bandwidth assigned to the VCCs sharing the capacity of a VPC, and implements the egress policing [6] needed to ensure that the resource allocated to a VPC is not exceeded. The functional model of the server is shown in Figure 3.

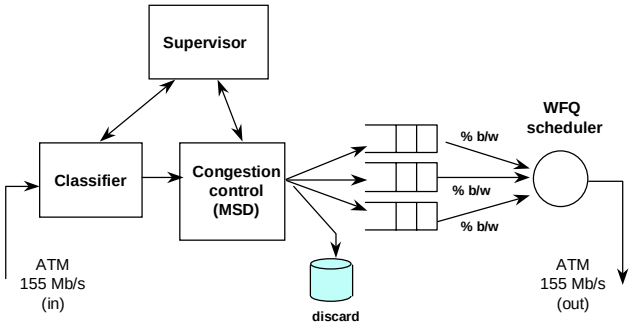

Figure 3 - Functional description of the ATM server

The input of the server is a stream of ATM cells that belong to different VCCs. Incoming cells are either discarded (that occurs when the server is congested) or accepted. A message selective discarding technique (MSD) avoids node congestion while preserving the flow integrity at the message level. If accepted, cells are stored in an internal buffer that is shared by several queues, one for each output VPC. The cell output process is based on a scheduling mechanism that is a variation of the Virtual Spacing Algorithm [8]; it shapes the output traffic by assigning to each incoming cell the timestamp at which it must be emitted.

5. Design Implementation

The architectural design of the A-VPN server has been driven by two goals: high design performance and low design time. The tight timing constraints of the server (for a 155 Mbit/s application one input cell and one output cell must be processed every 2.7 μs, also leaving some spare time for management), together with the availability of most of the required functions in the *VIP library*™, have led to the choice to implement the cell data path in hardware. The design of the server control part and its integration with the *VIP library*™ based cell data path has taken place within the POLIS framework. The internal architecture of the system is shown in Figure 4. It is composed of two parts: a fast datapath and a control unit.

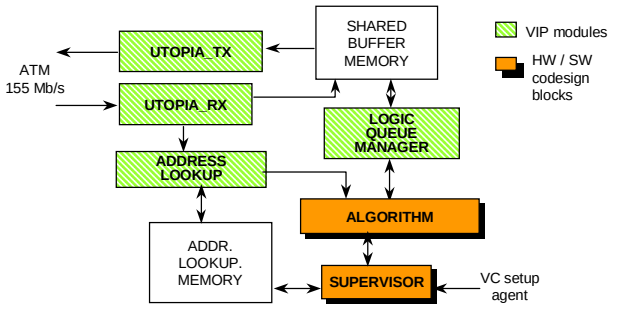

Figure 4 - Internal architecture of the ATM server

The datapath is implemented with seven VIP modules and two commercial memories, plus a small amount of custom logic (designed as RT-level VHDL) which translates the IP modules I/O signals into the event based communication formalism of the CFSM model of POLIS.

The datapath operation can be summarized as follows. Input cells arriving through the UTOPIA_Rx interface [7] are eventually stored in the Shared Buffer Memory. For each incoming cell, the Address Lookup module computes a compressed form (CID) of the VC identifier in the header. Using the CID, the Algorithm module can efficiently access the internal tables that store status information of the VCCs. The Logic Queue Manager handles the shared buffer so that it can be seen as a set of logic queues with FIFO access. At each cell time, a cell is extracted from the Shared Buffer and emitted through the UTOPIA_Tx [7] module, which adapts the internal data format to the UTOPIA standard.

The control unit, which implements the server custom functionality and is logically divided into the *algorithm* and *supervisor* blocks, has been designed as a network of 25 CFSMs in POLIS. The *algorithm* module decides which input cells must be accepted or discarded to avoid node congestion, and implements the shaping and bandwidth partition functions among ATM VPCs. The *supervisor* acts as an interface between the other modules and the external world: it interprets a set of configuration commands (e.g. add/remove VCCs/VPCs) and updates the internal tables and memories, translating the command protocol into the event based formalism required by the design methodology.

Figure 5 provides a functional description of *algorithm*.

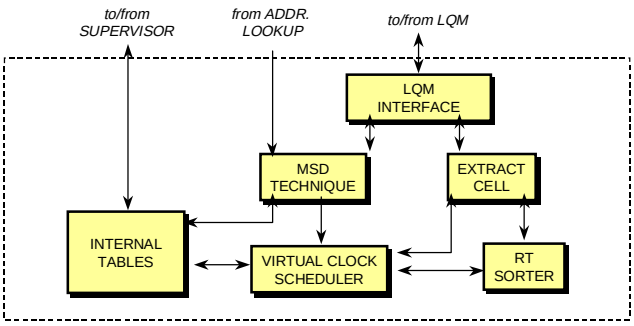

Figure 5 - Internal architecture of the *algorithm* module

Upon arrival of a new cell, *algorithm* receives the cell CID from the address lookup module. If the cell is accepted, m*sd_technique* sends instructions to the Logic Queue Manager (LQM) about where (i.e. in which queue) to store the cell in the shared buffer. Communication between the *algorithm* module and the LQM is handled by the *lqm_interface* block, which performs the required protocol adaptations.

When a cell is inserted in the buffer, the part of *algorithm* that performs the egress policing function is activated. This function is centered on the *rt_sorter* block that, upon request, returns a pointer to the cell with the smallest timestamp [9]. The *virtual_clock_scheduler* block computes the output timestamp value of the cell and inserts this value in the *rt_sorter*. When a cell is to be emitted, the *extract_cell* module queries the *rt_sorter* and gives instructions to the LQM Manager to output the cell with the smallest timestamp. A set of three *arbiters*, not shown in Figure 5, coordinates the access of the modules to shared resources (internal tables, virtual clock scheduler, rt_sorter).

6. Results

6.1. Design exploration

Starting from an informal specification of the system given in natural language, we have described the control unit using ESTEREL [3], for a total of about 1000 lines of code, and run simulations in the POLIS environment. This has allowed us not only to check the functional correctness of the design but also to explore several functional partitions of the control unit into modules and make design choices that minimize the custom hardware size and the communication among modules. For instance, in this phase we have decided to reduce the number of modules of the real-time sorter at the expense of a more complex control mechanism.

Its tight real-time constraints make the ATM Server an ideal design to explore limits and capabilities of the co-design methodology. Most system functions must fit within a cell time-slot (2.7 μs for an input rate of 155 Mbit/s). Therefore, timing constraints have been the driving factor of the partitioning evaluation process. We have run co-simulation experiments where some or all the larger modules in the control *unit (extract_cell*, *msd*, and *virtual_clock_scheduler*) are implemented in software. The target processor family chosen to estimate the performance of the SW tasks is the MIPS R3000 RISC processor family. Our experiments have shown that, in order to implement a reasonable number of CFSMs in SW, the CPU speed must reach at least 200 MHz, that corresponds to 544 CPU cycles for each cell time-slot. However, the faster processors in the R3000 family run at 50 MHz. So, for the prototype implementation we are using another RISC family with similar characteristics but running at a higher frequency, the Motorola PowerPC™ family. Table I presents the results of the partitioning evaluation: in particular it shows only the two partitions that meet the 544 CPU cycles requirement and implement

in SW at least one of the three largest modules. One arbiter and the supervisor, that is a natural candidate for SW implementation because of its low priority functionality, are also implemented in software. Partitions that stay within the limit of the cell time-slot but implement in SW only modules of small size are not considered.

Modules (lines of code)	Partitions (200Mhz) I	II
Msd_technique (180)	SW	HW
Extract_cell (85)	HW	SW
Virtual_clock_sched (95)	HW	HW
Arbiter_1 (33)	HW	HW
Arbiter_2 (34)	SW	HW
Arbiter_3 (37)	HW	SW
Rt_sorter (300)	HW	HW
Lqm_Interface (75)	HW	HW
Supervisor (120)	SW	SW

Table I: HW/SW partitioning evaluation results

6.2. Design validation

VHDL co-simulation has been used to validate the whole system including the IP modules. Here we used the POLIS VHDL model generator [4] which was developed during the design of the ATM server. The VHDL code for the ATM server is composed of about 14000 lines, of which about 7000 are RTL-level IP modules, about 6700 lines have been automatically generated within POLIS from 1000 lines of source code, and the remaining 300 are hand-written code (mainly protocol adapters).

Processor selection and HW/SW partitioning are done by modifying the parameters that select the timing estimates and regenerating the VHDL models. Since the main difference between the HW and the SW VHDL behavioral models of a CFSM is its timing characterizations, the code size and the simulation performance [4] are almost independent of the chosen HW/SW partition. Moreover, since the VHDL entity is the same for both HW and SW implementations, *no changes are needed in the VHDL test bench code when a different partition is tried.*

The test bench that we used for functional validation is composed of about 2000 VHDL code lines (excluding two memory models from a commercial library, for which the source code was not available). About 1800 lines belong to the ATM test bench (for cell flow generation and analysis) included in the VIP library, while a small amount of hand-written code describes a simple behavioral process reading the external server management commands. The availability of a pre-existing ATM test bench allowed a fast but very accurate functional validation of the whole design.

6.3. Design complexity and mapping results

After design exploration and validation, we have evaluated the implementation cost of the design. The hardware datapath including the IP modules has been mapped onto an ALTERA 10K70 device using commercial synthesis and P&R tools. It required about 25K equivalent gates, 1Kbyte of embedded memory (which can be mapped directly on the FPGA) and 1.5 Mbytes of external memory, implemented on commercial DPRAMs.

The hardware complexity of the control unit in the different HW/SW partition explored is shown in table II. The 25 Kbytes of memory required to store the internal tables are not included in the literals and registers count.

Partition	Literals (sop)	Registers
I	16145	1535
II	18699	1742

Table II: Control Unit HW complexity

On the basis of the reported results, we have selected partition I for the design implementation. Moreover, partition I implements in software the functionality of the MSD algorithm that is more likely to be modified in future upgrades of the system. For this partition, the software part of the control unit requires about 4500 code bytes, while the hardware part fits on a Xilinx XC4062XL device. The target clocks are 200 MHz for the microprocessor, and 25 MHz for the custom hardware. Since the standard HW-SW interfaces available in POLIS assume that the SW clock is slower than or equal to the HW clock, we had to define a new set of those interfaces. A gate-level simulation of the HW part of the design has been performed by using the same VHDL test bench we used for functional validation. A prototype is now under implementation, using the CSELT fast prototyping environment based on a commercial emulation bread-board (APTIX-MP3) with field programmable interconnect devices realizing the interconnections among the components placed on the board (FPGA and memories). The APTIX board is also connected to a microcontroller board, test instruments (logic analyzer, pattern generator) and a control workstation.

7. Lessons learned and future work

7.1. Interfacing memories

Memory access is a critical bottleneck of this design, since information about the connection status and parameters has to be retrieved from memory for every

incoming cell. At first, each memory was represented as a CFSM awaiting data and address (write) and emitting data (read) upon request. This mechanism is clearly inefficient because in the write operation data and address are handled as separate events without relative timing guarantee. Moreover, in the read operation the data is awaited by the reader and, in case the latter is implemented as software, it arrives after an unnecessary delay that degrades the performances. Hence, we modified the specification model for memories using C functions to access the locations directly from the reading and writing modules. This allows a more realistic estimation of the performance during the co-simulation step. Currently, we are developing tools for the automatic generation of the interface when the memory sits at the border of hardware and software partitions (reader is HW and writer is SW or vice versa). Note that this is a typical situation in the design of telecom systems.

7.2. Scheduling policy

POLIS offers two kinds of scheduling policies: Round Robin (RR) and Static Priority (SP). In RR tasks are checked for readiness in a cyclic pre-determined order and those that are found to be ready are immediately executed. In SP tasks are checked for readiness based on a statically defined priority order. For applications like the ATM Server, where the interaction with the environment is mostly periodic and has little influence on the time when tasks become enabled, it would be more efficient to adopt a *quasi-static* scheduling policy where the topological dependency among CFSMs is computed at compile time and tasks are checked for readiness at run-time based on this pre-computed portion of the schedule. By checking for readiness first the blocks that are most likely to be active, run-time overhead can be significantly reduced. Current work is going on in this direction [10].

8. Conclusions

In this paper we presented our experience in applying a design methodology based on design reuse and HW/SW co-design of embedded systems on an industrial test case in the Telecom domain. Our goals were the evaluation of the effectiveness of co-design tools and methodology on telecom designs, on one hand, and the definition and assessment of a methodology for integrating IP libraries in the co-design flow, on the other. The results of our evaluation can be summarized as follows.

The POLIS co-design environment is an efficient tool for fast evaluation and simulation of HW/SW partition choices. However, some limitations of the current version of the synthesis software make it unfit to the design of industrial telecom applications. It has to be pointed out that POLIS was originally targeted to the design of automotive applications, whose specifications (particularly in terms of event frequency and synchronicity) are quite different from those in telecom designs. One of the main problems highlighted by this study is that every inter-module event definition in the ESTEREL specification is translated into an inter-module communication primitive. However, not all the events that are defined in a specification are actually needed in its implementation (e.g. if event A directly causes event B and the delay between A and B has a computable upper bound, then event B is not necessary). Optimizing away useless events would lead to a huge embedded software performance improvement.

As far as the design time is concerned, the whole project required about 8 man months (to be compared against the 3 man years taken by the original design). CAD debug and extension time is not included in this figure.

The presented case study has also led to debugging and extensions of the POLIS tools (new processor characterizations and HW/SW interface models have been added to the POLIS library), and to the development of a new VHDL co-simulation methodology allowing easy integration of IP modules in the design framework.

References

[1] F.Balarin et al., "Hardware-Software Co-design of Embedded Systems - The POLIS experience", Kluwer Academic Pub., 1997.

[2] E.Filippi, L. Licciardi, A. Montanaro, M. Paolini, M. Turolla, M. Taliercio, "The Virtual Chip Set: A Parametric IP Library for System on a Chip Design", CICC, Santa Clara, May 1998.

[3] CISI Ingenierie, "Esterel V3 Language Reference Manual, V3.1, rel.4"

[4] Tabbara, "Validation of Reactive Real-Time Systems Using VHDL" Masters Thesis, University of California at Berkeley, May 1998.

[5] S.M. Walters, M. Ahmed, "Broadband Virtual Private Networks and their Evolution", International Switching Symposium ISS'92, Yokohama, October 25-30, 1992

[6] Coppo, M. D'Ambrosio, V. Vercellone, "The A-VPN Server, a solution for ATM Virtual Private Networks", Proc. of ICCS 94, Singapore, November 14-18, 1994

[7] The ATM Forum. "UTOPIA, an ATM-PHY Interface specification, level 1, v2.01", March 1994.

[8] J.W. Roberts. "Resource sharing in B-ISDN using virtual spacing scheduling", Proceeding of the 6th International Network Planning Symposium, 1994.

[9] J.W. Roberts, P.E. Boyer, M.J. Servel. "A real-time sorter with application to ATM traffic control", Proc. ISS'95

[10] M. Sgroi, "Quasi-Static Scheduling of Embedded Software Using Free-Choice Petri Nets", M.S. Thesis, University of California at Berkeley, May 1998.