Design of an ATM Network Adapter Board by using VHDL-based Simulation and Synthesis Tools

J. Carlos Calderón, José M. Tapia, Enric Corominas, Lluis Paris
Centre Nacional de Microelectrónica, CNM-CSIC.
Universitat Autònoma de Barcelona
Campus Universitat Autònoma de Barcelona 08193 Bellaterra, Barcelona, SPAIN

This paper describes the design and validation methodology used in the development of a complex Network Computer Interface Card, and two monolithic ICs. We have used VHDL to specify and validate the system from a functional point of view (pre-synthesis simulations). After the logic synthesis step, we have used non-VHDL descriptions for both the circuits and the test-data. The stimulus and expected-responses for the non-VHDL simulator (time-vector format) have been obtained automatically during a pre-synthesis (VHDL) simulation run. To make this process independent from both the VHDL-simulator and the logic simulator, the time-vector stimulus and responses files have been generated by using procedural VHDL testbenches.

I. Introduction

Widely found benefits of VHDL have been exposed by different authors: independence of CAD Vendor’s data proprietary formats, independence of ASIC libraries or processes, independence of simulation languages and formats, a wide range of descriptive capability, etc. However, in almost all the present design flows using VHDL high-level descriptions of ASICs, the use of VHDL stops at the logic synthesis step. This is because of different reasons: the lack of VHDL gate libraries in the foundry offerings, the simulation time (simulating a flat ASIC netlist with a full VHDL library on a full VHDL simulator the simulation time becomes tremendous), the lack of support for regular structures, etc.

Our research group is working in the development of an internetworking unit called UNICORN which is used to interconnect LAN’s as Ethernet and MAN’s as DQDB transporting IP over an ATM Network based on SDH and carrying data at the basic rate of 155 Mbps. The system architecture is based on a multiprocessor bus which interconnects the different elements on it: an ATM, an SMDS (Switched Multi-megabit Data Service, derived from DQDB) and an Ethernet Network Adapter Boards (NAB), and several processor boards. The ATM NAB gives interconnection with RECIBA (the Telefonica I+D B-ISDN testbed), and the SMDS NAB will be used to give interconnection with an SMDS based network by Siemens.

Three monolithic Telecom IC controllers have been developed to perform the critical NAB functions: the TCS chip [1, 2], the ATM-AAL chip [2], and the MPC chip. The TCS and ATM-AAL chips (a B-ISDN chip-set) are being used into the developed ATM NAB, and the MPC chip will be used into the SMDS NAB, that is being assembled at present.

---

1 UNidad de InterConexión para Redes IEEE802.6 e IEEE802.3 con Nóde ATM. The UNICORN project belongs to PLANBA (Plan Nacional de Banda Ancha), and is supported by the CICYT project TIC-1249/93
The paper is structured as follows. Section II is a brief description of the developed InterNetworking Unit. Sections III and section IV are a detailed description of the used design and validation methodology. Section V presents some experimental results and, finally, our conclusions are presented in section VI.

II. System Description

2.1. System Organization

The system architecture is based on the VMEbus-VME64 (I/O bus) which interconnects the different elements on it: an ATM, an SMDS and an Ethernet NAB, and several processor boards. The VMEbus [3] is the multiplexed VMEbus newest revision, offering the high bandwidth required by UNICORN. We are using two different commercial processor board models: a Motorola-MVME147 board (the system Host) and several Motorola-MVME162 boards (controlling the different NABs). Each data-packet goes across the VMEbus once (NAB ↔ processor-board. TCP-IP processing) or twice (NAB ↔ processor-board ↔ NAB, just IP processing) as maximum. On this multiprocessor system the main speed limiting factor is the I/O bus (VMEbus) usage, because the upper protocol layers (TCP-IP) are processed in parallel by separated processor boards. The ATM and the SMDS NABs have been developed to realize an efficient usage of both the Network and the VMEbus bandwidths in a cost effective manner. Finally, a commercial Ethernet board is being used as the Ethernet NAB.

2.2. Critical design aspects

ATM and DQDB are cell-based Networks, on which the data is transmitted and received into fixed-size packets, or cells, with a 5-byte header (cell identification) and a 48-byte information payload. For transmission, the higher-layer data-packets (with a large and unfixd size) must be segmented and transmitted into a number of cells. In reception, the incoming cell stream must be used to reassemble the received higher-layer packets. In order to realize an efficient Network bandwidth usage, the transmitter must support several segmentation processes in parallel; that is to say, the transmitter must be able to segment a number of packets simultaneously, realizing a per-cell line multiplexing. Analogously, the receiver must be able to demultiplex the incoming cell-stream and to simultaneously reassemble the received higher-layer packets. Three important implementation problems [4, 5, 6] must be taken into account: the shared I/O bus usage, the segmentation and reassembly (SAR) memory resources management and the circuitry complexity. First, to avoid an inefficient shared I/O bus usage, the developed NABs only use the shared bus to move higher-layer packets by using DMA. Second, to avoid an inefficient SAR memory resources usage the developed NABs realize a dynamic memory management: each higher-layer packet is stored as a linked list of small memory partitions (an ATM cell). Finally, to simplify the chips circuitry, they are implemented as single reassembly (segmentation) machines which operate on different reassembly (segmentation) contexts. Each context (CRC partial computation, etc.), corresponding to a particular instance of a reassembly (segmentation) in progress, is stored into external memory.

2.3. The ATM Network Adapter Board

The ATM NAB is a VME64 Double-Height Board, supporting the Physical [7, 8], the ATM [9] and the AAL 3/4 [10] Layers functions of the BroadBand ISDN (B-ISDN) Protocol Reference Model (PRM) in an efficient and cost-effective manner. Two ASICs have been developed to perform the critical ATM NAB functions: the TCS chip and the ATM-AAL chip. The ATM NAB is divided into six different blocks; fig. 1 shows the board organization.

The physical medium access block implements the Physical Medium of the B-ISDN User Network Interface (UNI). It consists of the lightwave transmitter and the receiver blocks, a clock recovery module, a 155.52 MHz crystal oscillator and five high-speed PECL↔TTL level shifters.

The TCS chip is a highly flexible B-ISDN controller. It implements the Transmission Convergence Sublayer (TCS) functions of the B-ISDN UNI: framing, HEC processing, cell
alignment, cell rate decoupling by using two on-chip FIFO memories, frame scrambling-descrambling, cell information scrambling-descrambling, SDH overhead octets generation-checking, etc. Only the bit to byte and byte to bit conversions, the byte alignment and loopbacks are performed at 155.52 MHz by using a pipelined approach. Excluding these functions, the chip works in a per-byte fashion at 19.44 MHz. An 8-bit microprocessor interface (MPI) allows the software to access 315 internal addresses: 45 of them are general registers, and the 270 remaining addresses correspond to three 90-word internal memories.

The ATM-AAL chip is a monolithic B-ISDN controller, supporting the ATM and AAL 3/4 functions. To properly process each incoming ATM cell, each cell is identified by means of two on-chip lookup tables (LUT). Each LUT is a 16-word CAM, allowing to store sixteen VPI, VCI and MID values. The chip supports a total of 214 AAL connections and 214 ATM connections. It supports a total of 32 simultaneous connections for packet segmentation and reassembly, and an unsettled number of ATM connections. The ATM-AAL chip comprises a complex memory controller that performs the SAR memory resources management. An 8-bit MPI allows the software to access 81 internal registers.

The SAR is a 3.5 MB single-port SRAM, organized in 32-bit words. The ATM-AAL chip access the SAR at a 19.44 MHz rate; this allowed us to use moderate-speed memory devices, with an access time of 35 ns. This memory is divided into two parts: 0.5 MB store two 256 KB FIFOs, and 3 MB are used to store 64K SAR partitions. Each FIFO is used in transmission and reception to store entire ATM cells. Each SAR partition has 48 bytes; the first word is used to store a 16-bit partition tag (pointing to the next partition); the remaining 11 words store data.

The VMEbus access block contains the VMEbus interface circuitry. For simplicity, sanity, space savings and speed we are using a commercially available VMEbus interface chipset, the Cypress VIC64 [11, 12] (VMEbus controller) and the Cypress CY7C964 [12] (VIC64 companions). Even using this chipset, a lot of extra components were needed to complete the required functionality: VMEbus addresses decoding, NAB local addresses decoding, NAB local bus arbitration, etc. Two Altera EPLDs (the MAX7000 7032 and 7128 devices) and several discrete components (245's, 573, etc.) are being used.

Fig. 1. UNICORN ATM NAB architecture

III. Design and Validation Methodology

Four different design tools have been used to design the cards. The Vantage VHDL simulator was used to realize board-level simulations (pre-synthesis validation), simulating altogether the synthesizable VHDL descriptions (ASICS and EPLDs) and the non-synthesizable VHDL models of the commercial board components (memory components, VMEbus interface circuitry, etc.). The Compass Design Automation Tools were used to make the specification capturing (ASICS,
EPLDs and commercial components) and to design the three ASICs: VHDL synthesis, regular structures compilation, layout-generation, and post-synthesis and post-layout validations. The Altera Max+PlusII tools were used to program the EPLDs using its synthesizable VHDL descriptions. Finally, the Mentor Graphics Board-Station software was used to design the cards (layout generation, tracks analysis, etc.).

3.1. Specification capturing
First, in order to simplify the design specification capturing of the boards and the ASICs, a mixed textual-graphical description has been created by using the Compass schematic and text editors. The design structure has been specified graphically, by drawing and interconnecting boxes. Each box contains other graphical descriptions (design hierarchy) or manually written VHDL descriptions (block behavior). Compass automatically generates the equivalent VHDL descriptions for the graphically specified parts: VHDL files containing the component declarations (the graphical boxes) and the component port maps; this is necessary to enter a full-VHDL description of the design into the Vantage simulator.

Second, the ASICs have not been fully specified by using manually written VHDL descriptions. The ASICs contain regular parts, such as register-files, FIFOs, CAMs, monitoring counter banks, etc. These blocks have been created by using the Compass data path compiler, obtaining efficient silicon-area implementations. Again Compass generates automatically the behavioral VHDL descriptions of these layout blocks, as required to realize the pre-synthesis simulations. The data-path compiler implementation of some partially regular block (such as the memory management process unit in the ATM-AAL chip) does not presented significant silicon-area reductions when compared to the VHDL synthesizer random-logic implementation (obtained by using a VHDL specification). In spite of that we used the data-path compiler implementation because it strongly simplified the place and route and the post-layout validation steps: we obtained “pre-routed” blocks with pre-determined track delays.

Third, we directly wrote synthesizable VHDL models; we have not used a top-down refinement process because of different reasons. For this application field the non-synthesizable VHDL description of almost all the circuit blocks (control state machines, CRC generators, etc.) is very similar to the equivalent synthesizable description. By directly writing synthesizable VHDL descriptions we reduced the amount of design data (easier design management) and we were fully aware of the realities of hardware.

3.2. ASIC design and validation
Fig. 2 shows a scheme of the design and validation methodology used in the development of the three ASICs; the figure represents the full design flow, starting at the specification capturing and ending in the test of the ASICs. The flow diagram has two different entry points: the circuit specification capturing, and the test benches specification capturing, by using procedural VHDL descriptions.

In the pre-synthesis validation step, we have validated the specification of each ASIC (synthesizable VHDL descriptions for random-logic and behavioral VHDL for regular parts) and each EPLD, only from a functional point of view, by means of board-level VHDL simulations. These simulations have also been used to validate the board-design (its functionality and its timings). The board elements (mainly commercially available components) were described with accurate timings, by using its data-sheets. For the synthesizable ASIC and EPLD VHDL descriptions we described only its external delays (input and output pin delays). The wide range of descriptive capability of VHDL allowed us to write accurate models for both the hardware components (single-port and dual-port SRAMs, VMEbus interface circuitry, etc.) and the software (the card drivers that control the boards through the VMEbus).

In addition to these VHDL models (stimulus generators), it was written VHDL program that generates, during a VHDL simulation run, the time-vector stimulus and expected responses files to be used in the next validation steps: the post-synthesis and post-layout simulations and the test of the ASICs. This process is described in the next subsection.

134
Fig. 2. ASIC design and validation methodology

For the post-synthesis and post-route validations we have used two different tools: the Compass QTV timing analyzer (critical paths identification, clock-skew detection, etc.) and the Compass Qsim simulator. Each ASIC is divided in several large synchronous blocks that use different clock signals (network clock, on-board crystal oscillator clocks and the internally divided clocks), and asynchronous blocks, such as the microprocessor interface circuitry. By using the QTV we have analyzed each synchronous block one by one, and by using the Qsim simulator the ASIC circuitry has been validated as a whole, allowing us to detect the timing problems (communication between asynchronous parts, etc.) and some synthesis problem. Since the Qsim simulation files were generated during the pre-synthesis simulation containing both the stimulus and the expected responses, during the post-synthesis and post-layout simulations we just needed to watch the Qsim output messages: error messages (hold and setup violations, etc.) and the test-failed messages (simulation results differences).

3.3. Non-VHDL stimulus and responses generation

Fig. 3 shows the connection between the different VHDL models (ATM Interface Card modeling) entered into the VHDL simulator during the pre-synthesis simulation of the ATM-AAL chip; the TCS chip was simulated in a similar way. During a simulation run, one of these VHDL models (the low-level test-data generator) reads the input / output of the DUT (the ATM-AAL chip) and writes a text file containing the strobed vectors (stimulus and expected-responses).

To reuse already-debugged VHDL code in the simulations of the three ASICs, and also in future designs, the low-level test-data generator model is divided into three different parts: a DUT-dependent entity-architecture, a test-format dependent package and a generic package. The generic package declares shared data-structures (records containing the DUT input / output configuration and status). The test-format dependent package contains two different functions: the write_header() function, that initializes the output text-file, and the write_vector()
function, that appends the strobed vectors to the output text-file. The DUT-dependent entity-architecture contains the DUT-specific information (pin names, pin types, pin values, strobe pin, etc.), initializes data-structures and calls the write_vector() function at every clock cycle.

We have written three DUT-dependent entity-architectures (TCS, ATM-AAL and MPC chips), a single generic package and two different test-format dependent packages: the Qsim_package, that generates the stimulus responses in the Qsim simulator format, and the TEKsWAV_package, that generates the test vectors for a Tektronix LV-500 ATE.

Two different types of pre-synthesis simulations have been made: the asynchronous simulations and the synchronized simulations. In the asynchronous simulations the different board components (ASICs, EPLDs, VMEbus interface circuitry, etc.) use different clock signals: network derived clocks, on-board oscillator clocks, etc. These are realistic simulations that faithfully reproduce the NAB behavior. In the synchronized simulations the different board components are synchronized by using the same clock signal (or related frequencies). These unrealistic simulations were realized to obtain the non-VHDL stimulus-responses. This is necessary because of the synchronous nature (cycle-based) of the low-level validation tools. This simple approach, based into the VHDL TEXTIO package (write and writeln procedures), presents two important benefits. First, this simple method allows to use two simulation tools without a common interface, for instance the Vantage and the Qsim simulators. Second, this VHDL-based (VHDL-written) interface is independent of specific simulators, in contrast with a tool-based interface.

![Diagram of circuit](image)

Fig. 3. Pre-synthesis simulation example: ATM-AAL chip simulation

3.4. Software Modeling
To perform the board-level simulations we have modeled the card drivers by using VHDL. That is to say, VHDL has been used to validated altogether three different design levels: the ASICs (behavior), the board (behavior and timings) and the software. Our software model is a VHDL entity-architecture connected to the VMEbus lines (address, data, interrupts, reset, etc.). This VHDL description uses different functions that allow to describe the card drivers in a
software-like manner: the execute_file() function, that executes a program (stored in a text-file) written in pseudo-code, the TxDMA() function, that reads a text-file (a data-packet to be transmitted) and writes the contents in the on-board memories, etc. The pseudo-code programs only use three instructions: R (read), W (write) and N (nop). Fig. 4 shows a pseudo-code program (reduced) example: the TCS chip registers configuration program. The card drivers have been written in C, by translating the VHDL software model and the pseudo-code programs. These drivers have been created by using VMEexec, an industrial software development package by Motorola, that allows to generate the kernel (based on pSOS), the drivers and the application tasks to be executed on the Motorola processor boards.

---

-- Start partial reset (register 0)
W 0 "00000010"
-- Eight NOP cycles
N 8
-- Master configuration
W 2 "00010000" (register 2)
R 3
-- New administrative unit pointer
W 28 "00000000"
W 91 "10010001"
W 94 "01101000"
(etc.)
-- Alarm masks
W 5 "11111111"
W 9 "11111111"
W 14 "11111111"
W 22 "11111111"
W 31 "11111111"
W 39 "11111111"
W 44 "11111111"
-- End partial reset (register 0)
W 0 "00000000"

Fig. 4. Pseudo-code program example: TCS chip configuration

IV. Developed VHDL Models

Table I shows a summary of the developed VHDL models. This table only shows the manually written VHDL descriptions; that is to say, it does not consider the automatically generated VHDL descriptions: the structural (boards and ASICs structure and hierarchy) and the behavioral (ASIC regular structures) VHDL descriptions.

For the EPLDs, the table does not show an exact number of written lines; this is because we have used slightly different EPLDs descriptions on each board (ATM and SMDS).

The VHDL memory models are parametrizable VHDL descriptions allowing to configure its organization (number of words and word-width) and its timing characteristics by changing the values of several VHDL constants. Initially, each memory was modeled as a variable array(0 to wordDepth-1) of Std_logic_vector(wordSize). However, by using this simple approach the simulation time became tremendous when simulating the segmentation and reassembly memory (SARM). The SARM is a 3.5 MB memory organized into 32-bit words, i.e., the workstation memory stores 2^8*1024^2 (28 M) Std_logic values. To solve this problem we modeled the memories by using two different variable arrays: an array(0 to wordDepth-1) of integers and an array(0 to wordDepth-1) of booleans. Each integer (a word in the workstation memory) is used to store 32 bit values (a word of the SARM), and each boolean indicates the status of this memory word: unknown value (true) or valid value (false). In this way, we reduced the
workstation memory requirements to $3.5 \times 10^{24} \text{ integer}$ values plus $3.5 \times 10^{24} \text{ boolean}$ values, obtaining reasonable simulation times.

The employed VMEbus chipset supports a lot of complex functions: VMEbus master, VMEbus controller, etc. Our VMEbus chipset VHDL description is a very simplified model, that only describes a set of basic functions; this has been possible because the developed NABs are VMEbus slave boards that only use some function of this chipset.

<table>
<thead>
<tr>
<th>Component</th>
<th># lines</th>
<th>organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCS chip (synth.)</td>
<td>8490</td>
<td>1 package, 31 entity-archit.</td>
</tr>
<tr>
<td>ATM-AAL chip (synth.)</td>
<td>11764</td>
<td>1 package, 28 entity-archit.</td>
</tr>
<tr>
<td>EPLD1 (synth.)</td>
<td>= 200</td>
<td>1 entity-archit.</td>
</tr>
<tr>
<td>EPLD2 (synth.)</td>
<td>= 600</td>
<td>1 entity-archit.</td>
</tr>
<tr>
<td>single-port SRAM</td>
<td>276</td>
<td>1 entity-archit.</td>
</tr>
<tr>
<td>dual-port SRAM</td>
<td>443</td>
<td>1 entity-archit.</td>
</tr>
<tr>
<td>VMEbus chipset</td>
<td>612</td>
<td>2 entity-archit.</td>
</tr>
<tr>
<td>Software model</td>
<td>= 1200</td>
<td>1 package, 1 entity archit.</td>
</tr>
<tr>
<td>Non-VHDL test-data gen.</td>
<td>1000</td>
<td>2 package, 1 entity-archit.</td>
</tr>
<tr>
<td>Others: electro optical interface, 245's, etc.</td>
<td>= 500</td>
<td>11 entity-archit.</td>
</tr>
</tbody>
</table>

Table I. Manually written VHDL code summary

V. Implementation

The TCS chip has been implemented using the 1\textmu m CMOS ES2 (European Silicon Structures) technology with a die area of $8.4 \times 9.3 \text{ mm}^2$. The chip has been packaged in a 100-pin CPGA and contains 142,989 transistors. Six layout blocks were obtained using the data-path compiler: three 90-word register files, and two FIFO memories. Although the TCS chip design was finished in December-1993, sent to the foundry in March-1994, and tested in September-1994, the chip was not used in the final application board (the ATM NAB), together with its companion the ATM-AAL chip, until July-1995.

The ATM-AAL chip has been implemented using the 0.7 \textmu m CMOS ES2 technology with a die area of $8.4 \times 7.8 \text{ mm}^2$. We used the VLSI Inc. 0.8 \textmu m libraries, which are fully compatible with the 0.7 \textmu m ES2 process. Ten layout blocks were obtained by using the data-path compiler: two CAM memories, a timers bank, the memory manager process unit, etc. The chip has been packaged in a 180-pin CPGA and contains 218,672 transistors.

<table>
<thead>
<tr>
<th></th>
<th>TCS</th>
<th>ATM-AAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology (2 metal)</td>
<td>1\textmu m</td>
<td>0.7\textmu m</td>
</tr>
<tr>
<td>Libraries</td>
<td>VLSI 1\mu m</td>
<td>VLSI 0.8\mu m</td>
</tr>
<tr>
<td>chip-size (mm$^2$)</td>
<td>78.61</td>
<td>65.74</td>
</tr>
<tr>
<td>#pins</td>
<td>100</td>
<td>180</td>
</tr>
<tr>
<td>#regular blocks</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>#transistors (tr.)</td>
<td>142,989</td>
<td>218,672</td>
</tr>
<tr>
<td>% tr. in standard cells</td>
<td>62.6 %</td>
<td>65.1 %</td>
</tr>
<tr>
<td>%tr. in regular blocks</td>
<td>37.4 %</td>
<td>34.9 %</td>
</tr>
</tbody>
</table>

Table II. Developed ASICs layout characteristics summary

The UNICORN ATM NAB is a Double-Height Board; it was implemented by using a class-5 process (a track width of 0.15 mm) and six conductor layers (four signal layers and two power
VI. Conclusion

Three important VHDL features, the technology and process independence and its wide range of
descriptive capability, made VHDL a very appropriate (but insufficient) stepping-stone to
develop this complex card. On the one hand, the use of synthesizable VHDL descriptions for the
circuits and VHDL-based synthesis tools strongly simplified the design specification capturing
of the ASICs and the EPLDs.

The presently available VHDL-based design tools are not sufficient to completely specify and
validate a complex VLSI system. We perceived a gap between the VHDL-based design tools
(upper design layers), which application ends at the logic synthesis step, and the non-VHDL
design tools (post-synthesis validations, layout generation, post-layout validations and chip
test). To eliminate the gap between the procedural VHDL testbenches and the low-level test-
data formats we propose a simple and tool-independent approach: to use a VHDL-written
interface that generates the low-level test-data.

VHDL synthesis is not a "push-button" process, it is a far more iterative process than, for
instance, the physical layout generation. During the design of these complex circuits the top-
down design techniques were continuously tempered with bottom-up information. Also, writing
VHDL code is a simple task when compared to a classical design specification capturing by
using schematics (manual optimization and manual technology mapping). However, writing
ten-thousand lines of VHDL code is a tedious and error-prone task, so tools which generate
VHDL code automatically are highly desirable.

References

ISDN Using VHDL Based Synthesis Tools”, Proc. EURO-DAC’94 with EURO-VHDL’94, p520-
[5] IEEE Network, special issue on end-system support for high-speed networks, Vol. 7, No. 4,
July 1993
IEEE, Dec. 1990