VLSI Protocol Parser Generator for Packet Processing

Jorge Guedes Silveira *
Imad Sabouni **
* DEE - Depto. Eng. Elétrica - Escola Politécnica-PUCRS Av. Ipiranga 6681- Porto Alegre- Brasil
** NRL - National Research Laboratory- Damascus -Siria
guedes@ee.pucrs.br

Abstract :

The higher bit rate in the communication processing networks, the support of new technologies as cell switching, and the need of a higher throughput in the packet processing domain is the major factor that explains the designers' trend to develop VLSI implementation of communication protocols. It becomes a necessity in many ISO, ITU and IEEE protocol implementations.

This paper discusses first the cross-impact of VLSI technology in future communication evolution, then it discusses the functional partitioning and the key characteristics of communication circuits, and it will present a flexible approach to designing the physical protocol parser, which is the central module of each reception machine of a communication circuit. It will be emphasized the VHDL validation of the module's architecture. The generation of this module depends on the protocol type and its skeleton, the grammar of the frame description and the expected real-time treatment of the packet processing. The conception of communication protocols requires several phases: specification, validation, simulation, implementation. Today, these steps are done with independent tools.

This paper presents an original methodology using only one language all along the specification and design process. This approach is based on VHDL, a high level hardware description language normalized by IEEE. It is finally outlined in this work an approach to systematically implement protocols, based on separation between the frame syntax analysis and the field data processing. Two case studies are presented.

Resumo :

Os principais fatores que induzem à necessidade de implementação de protocolos de comunicação com circuitos VLSI são o processamento de altíssimas taxas de transmissão nas redes de alta velocidade, o suporte de novas tecnologias como a comutação de células, e a necessidade de maiores velocidade quando do processamento de pacotes. A implementação em "hardware" tem se mostrado ser uma necessidade crescente na implementação de muitos protocolos normalizados pela ISO, ITU&T ou pelo IEEE.

Inicialmente este artigo se propõe a discutir o impacto da tecnologia VLSI na evolução dos circuitos de comunicação, apresentando a seguir uma segmentação funcional de acordo com as características chaves dos circuitos de comunicação. Apresenta uma abordagem flexivel no projeto de um circuito Analisador de Protocolos, o qual é o módulo principal de cada Máquina de Recepção nos circuitos de comunicação. É dado especial ênfase à validação dos módulos com VHDL. A geração do módulo Analisador de Protocolos depende sobretudo do tipo de protocolo e de sua estrutura, da gramática da descrição do quadro de dados e do processamento do pacote em tempo real. A concepção de protocolos de comunicação requer várias fases : especificação, validação, simulação, implementação e testes os quais são realizados por diferentes ferramentas.

Este artigo introduz uma metodologia original que utiliza uma única linguagem ao longo de todo o processo desde a especificação até o projeto. Esta abordagem é baseada em VHDL, uma linguagem de descrição "hardware" de alto nível normalizada pelo IEEE. É também destacado neste trabalho um enfoque que permite a implementação de protocolos de forma sistemática, baseado principalmente na separação entre a análise da sintaxe do quadro e no processamentos dos dados nos campos específicos. São apresentados e analisados para validação dois diferentes protocolos de comunicação.

Keywords :

Packet Processing, Hardware Protocol Implementation, Communication Circuit Architecture, Function-Code, Frame Specification, VLSI Parser Generator, VHDL Validation, AAL3/4 protocol.

1 - Introduction

High-level ISO protocol implementation has been traditionally carried out using software approaches. New technologies in networking, namely the new-generation transmission links, have resulted in the need of communication controllers that can handle high-speed traffic. It will become more difficult to implement the corresponding protocols in software [1][3] since such implementations would be too slow and offer unacceptable transfer delays for many applications [4][12]. To reduce the message transfer delay and achieve a higher processing speed, a specialized, VLSI-based implementation would be necessary [2][16]. Therefore we assist an increasing trend for hardware by designing ASIC's dedicated to the execution of low-layer ISO and IEEE protocols [5] as well as circuits dedicated for cell based high speed networks like B-ISDN, DQDB or LANs ATM(FORUM ATM)[20].

The design process of communication protocols requires three main step: specification, validation and implementation. There are also phases like simulation to aid the designer in the debug. Nowadays, these steps are carried out by using distinct languages. The protocol specification is often written in natural language or using formal description techniques such as LOTOS, ESTELLE or SDL. The description of the architecture uses specific languages such as VHDL [17] for hardware implementation points to go right to silicon. Software implementation is done using classic programming languages like C, ADA or OCCAM [13] for parallel execution. Some tools exist to generate C source code from formal description languages. But they often produce non-optimized programs, and the programmer must re-write a part of code. Moreover, they are adapted only for software implementation, and the emergence of high speed protocols increase the importance of the hardware points. This large disparity of languages during the design introduces uncertainty. We show in this paper how to reduce this incertitude by using the VHDL language all along the design process.

In the most general case, however, the commercial circuits are not entirely autonomous. They depend, in fact, on the host microprocessor in order to perform the complete protocol execution. Furthermore the upper levels are in this case managed by an executable code which is loaded in the station system kernel. These software implementations are slow, their transfer delays are, for many applications, unacceptable [10]. A hardware solution can provide, under certain considerations, a good increase to this rate throughput [6][13][19].

The throughput requirement is not the only reason that can explain the tendency for developing VLSI circuits, because economic necessities are also a primordial factor: integration provides substantial decrease in the equipment cost, the supplementary costs due to research and study supplementary costs are redeemed by the number of manufactured pieces. Moreover, miniaturization can reasonably come up to industrial needs related to environment and size requirements.

Our main design goal is the implementation of a communication circuit dedicated to the execution of system protocols that can greatly reduce the delay to transfer a message, hence enabling the host station to take full advantage of the network capacity, and minimizing the intervention of software in the management of such data protocols and services.

We must take into account that protocol specifications are, in the most general case, software-oriented ; the frame structure is often defined in such a way that a *real-time processing* cannot be carried out "easily" using a hardware circuit. Our approach is based on a flexible architectural model that performs two separate tasks: the analysis of the frame structure and the processing of data associated to its different fields. We propose a design methodology based on automatic module generation and leading to communication architecture synthesis. By studying ATM protocol we have realized that there are some redundancies within certain protocols levels, and that the AAL frames present fixed lengths in their control fields, with a specific purpose in each field, which allowed us to adapt our generators to process the AAL3/4 (for WAN in accord with ITU standards)[16][19] and the AAL5 (in accord with the Forum ATM recommendation)[20]. A data communication circuit has to run a certain number of parallel and communicating processes [3]. Consequently, it is composed of dedicated machines which are allocated, statistically or dynamically, to these processes. The Reception Machine is the part of the communication circuit which is in charge of processing the incoming frame. If we want our circuit to implement a protocol regardless of its level and of the standard it is issued from, then a specialized module within this machine must be available to analyze the structure of the frame, called in this paper by the skeleton, to understand the static and/or the dynamic fields it contains and finally to "hack" the frame in order to transmit fields data to some processing units like address recognition unit, network clock management unit, CRC management unit, and others. This module will be called : The *Protocol Parser*.

In the present paper, we will focus over the analysis of the frame in the reception case and the architecture of the protocol parser module. Chapter 2 is intended to the study of the frame skeleton and the definition of the associated parser architecture, place and role within the reception machine of a communication circuit. Chapter 3 specifies and implements the module's architecture. In chapter 4, we present the validation to a well-defined class of protocol parser modules by presenting a dedicated *silicon generator* and by carrying out two case studies of the HDLC protocol and the AAL3/4 ATM protocol. Finally, chapter 5 gives out some general conclusions concerning the protocol formal description and relationships to the silicon architecture generation domain.

2 - Hardware Protocol Implementation

2.1 - Behavioral Modeling

The implementation of a MAC-level protocol [7] is based on two processes: the transmit process and the receive process, each of which executes a number of functions to manage the outgoing or the incoming frames.

The transmit process consists in assembling the different fields of the frame in a given order. Those fields can be either provided from the application-side interface and are considered as data or commands in the MAC level, or "internally" generated by the transmit process itself. One typical example of an internally generated field is the FCS (Frame Check Sequence) which is computed by the transmit process and appended to the outgoing frame. The receive process can be seen as the reverse of the latter. It consists in disassembling (parsing) the different fields of the incoming frame in order to send them either to the application or to some internal processing units. The FCS field, for instance, is treated during the reception to check the coherence of the frame.

Fields generated or processed internally in the MAC level are sometimes called *function codes*. They are related to events occurring during the frame transmission or reception (data request, data acknowledge, ...). In many cases, they may have data associated, as arguments, to them. In both the transmit and the receive processes, the assembling/parsing operation is essentially based on the different structures the frame may have. These structures represent *the frame syntax*. On the other hand, for each possible structure of the frame, a set of processing tasks must be executed. Those represent *the frame semantic*.

2.1.1 Frame definition

A given frame can be defined as a variable length vector of N bits. We call frame skeleton an increasing sequence $\{b_i\}_{i=1,\ldots,n}$ where :

- $b_0 = 0$; $b_i > 0$ for each i in [1..n], ; where b_n is the frame total length.

Two successive terms, b_{i-1} & b_i, of this sequence delimit a field F_i (with 1_i_n). The frame is thus organized as a set of n fields. For a given i, the knowledge of the terms b_j, (j = i+1, i+2, ..., n) can be predetermined ; this is the case of a "simple" frame where the skeleton is said to be *static*. On the contrary, values of $\{bj\}_{j\_i+1}$ can depend on the content of the field F_i ; starting from b_j, the frame can have a multitude of structures following the value of that field and the skeleton is then said to be *dynamic*.

2.1.2 - Function-Code Specification

A basic ISO protocol implementation is based on two finite state machines. Each one transmits an event to its correspondent either to inform it of a data transmission or an acknowledgment, or to formulate a request. This event is coded in-frame within a given field that is called as s Function-code.

Let us present some formal features of the function-code field :

- The function-code is a predefined field within a frame. The field location, length and definition domain are fully known.
- The function-code type is enumeration. This means that the corresponding field takes its value from a predefined finite set.
- Data can be associated to a function-code.
- A function-code can modify the structure of the rest of the frame. This is the case of the dynamic skeleton frame discussed before. The new structure, that is the new values of the skeleton sequence can be carried by some data frame fields associated to the function-code and the skeleton knowledge is then *explicit* (length and type of parameters for example). In the opposite case, the skeleton knowledge is *implicit*.

The identification of a certain value which belongs to the function-code definition set (by a certain function-code recognition unit) is known as function-code recognition. This recognized value induces usually the triggering of a corresponding task to perform some specific treatment. As we have said previously, data can eventually be associated to this function-code to indicate, for instance, the length of the frame or a variable part of it, a clock value, etc... Corresponding data field (or group of fields) is then considered as argument(s) to such actions launched by the recognition process.

A classical example of function-codes is the address field within a frame, this field can be used *to invoke a function* besides its "natural" role of *routing*. In the IEEE 802.2 protocol specifications [11], management function uses reserved addresses. This is an example in which the function-code is *implicitly carried* in a field.

The Polling bit (P) in the HDLC protocol is another example [9]. Here, the function-code is *explicited* by the transmitting station. In fact, put to 1, this function-code request the secondary station(s) to give an *immediate* response. In some cases, and in order to simplify the treatment, this bit can be seen as a part (a sub-field) of the HDLC frame command field, the later being considered as a global function-code.

Another example taken from the HDLC protocol specification(figure 1) is the first bit of the command field (8 bits). In fact, the off (0) value of this bit indicates an information frame format. In this case, the rest of the field is structured as below. On the on (1) value, another function-code is encountered : the next bit. It indicates either a supervision format or a non-sequential format.

1	2	3	4	5	6	7	8	*bits*
0	N(S)			P/F	N(R)			Information format
1	0	S		P/F	N(R)			Supervision format
1	1	M	M	P/F	M	M	M	Non-sequential format

figure 1 - HDLC function codes

This is an illustration of function-codes modifying the frame skeleton. We will call them *skeleton function-codes*. Let us notice that the second skeleton function code do not exist in the information format case.

Command field function-codes involving the frame skeleton definition appear also in many protocol specifications, for example in the X25 and ISO transport protocols [9][21]. In the IEEE 802.4 protocol specifications [11], the first two bits of the global control field define a skeleton function-code. If those bits are off, then the field describes a MAC frame. The next six bits indicate the MAC control frame type. The recognition unit has to match the value of these bits, which belongs to a restricted definition set, among the 2^6 theoretical possibilities. On the other hand, if the skeleton function code value is 01, then the field describes a LLC data frame. In this case, the next six bits have a different structure : bits 3, 4 and 5 constitute the MAC-action field, which is a static function-code, while bits 6, 7 and 8 constitute the priority field. Information contained in this last field must be associated as arguments to the previous skeleton function-code. The function-code is said to be *pre-fixed*. In the X25.3 protocol specifications [9], the skeleton function code is defined by the first half-byte and the third byte of the frame (except for the RR, RNR and DT frames). The DT frame is defined only by the first bit of the third byte (if equal to zero), the seven last bits of this byte contain the P(R), M and P(S) information. This type of function-code is said to be *post-fixed*. In the ISO transport protocol [23], the skeleton function-code is defined by the second byte of the frame. The variable part of the following frames (CR, CC, DR, DC and ER) contains other function-codes particularly for exchanging parameters. This type of function-code is said to be *nested*. From the previous examples, we notice that the function-codes could be static or dynamic, pre-fixed or post-fixed, explicit or implicit. They also could be cascaded or nested.

2.1.3 Protocol syntax

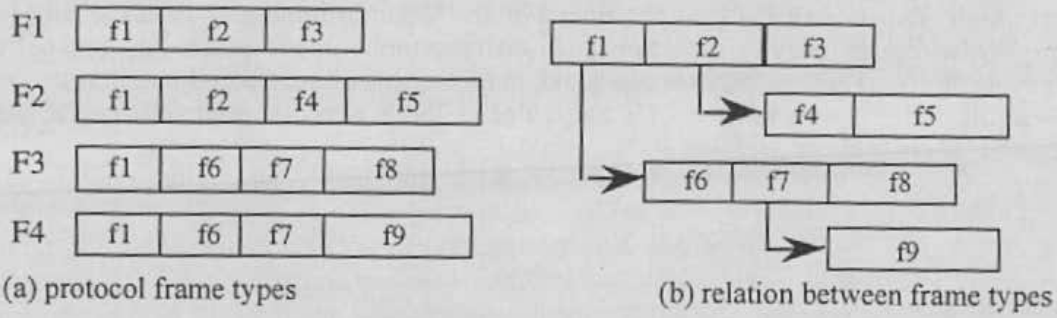
The protocol syntax is, then, related to two attributes: (i) the set of all the fields involved in the frame composition and (ii) a set of rules that govern the sequencing of these fields in the frame structures.

• The field: A field is a set of bits $\{b_i\}$ properly coded for transmission in the physical layer. Every *field type* is characterized by its size (the number of bits it contains) and its value. It is initiated by a *source unit* and conveyed to a *target unit*. Two sorts of fields can be encountered within a frame: a *MAC field* and a *physical field*. The first one is intended for the MAC level while the second is entirely processed by the physical layer. Its value is ignored by the MAC layer.

The value and the size of a given field can be constant or variable. Variable-size fields (simply referred to as variable fields) have a number of bits that may change from one frame to another. The actual size of every one of them is determined during the frame processing.

Many protocols contain special function codes which value can change the remaining structure of the current frame. We call them structure-modifying function-codes (SFC). More than one level of hierarchy of SFC may exist.

• The field sequencing: The sequence of fields in a given frame determine the *frame type*. The different types are defined in the protocol specifications. For example, consider an hypothetical protocol with four frame types illustrated in figure 2-a.



(a) protocol frame types (b) relation between frame types

figure 2 - frame types

Let **F** be the set of these types:

$$\mathbf{F} = \{F_1, F_2, F_3, F_4\}$$

and **f** the set of the field types involved in all the possible frame structures:

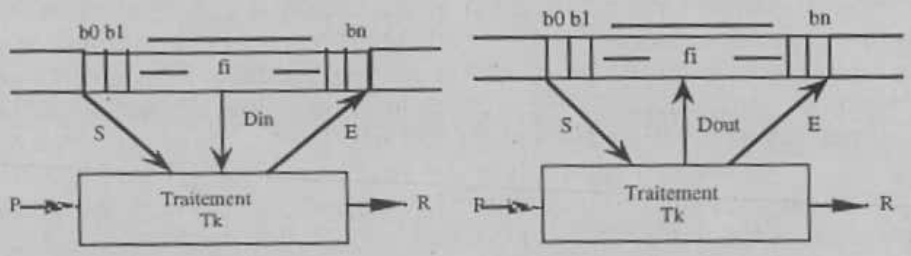
$$\mathbf{f} = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9\}$$

The relation between the different frame types is represented in figure 2-b.

In this example, f1, f2 and f7 are SFC. According to the actual value of f1, either f2 or f6 follows. If the second field is f2, then one might have F1 or F2 type, and so on. This relation between the different frame types may be expressed by a graph where the nodes represent the fields and the arcs indicate the possible sequences of fields.

2.1.4 Protocol semantic

The frame semantic is concerned with the treatment triggered during the frame processing. It allows to map the different fields of **F** to a set of processing tasks $\mathbf{T} = \{T_k\}$. To every processing task are associated (i) two signals: start-activation, S and end-activation, E, (ii) a set of parameters, **P**, (iii) the data carried within the field, D and (iv) the operation result, R (see figure 3). For example, consider a field that contains the address of the target station. The associated processing task, activated during the reception of the field, have as input the value carried within the field (D). This value is to be compared to the address of the station, regarded as a parameter (P). The output (R) is the fact that the address has been recognized or not.



(a) field processing during reception (b) field processing during transmission

figure 3 - field processing

2.2 - Structural Modeling

After having presented in section 2.1 the behavioral modeling of a communication protocol, we will focus in this section on the architecture of a dedicated communication circuit. Such a circuit can be divided into three parts [7]: the MAC protocol processing part (the circuit core), the application-side interface, and the medium-side interface.

2.2.1 The MAC protocol processing part

The representation of a MAC-level communication protocol by two graphs can be translated to a circuit architecture. Such an architecture is composed of two control parts and a set of processing units that are bounded to the different treatments associated to each field. The control parts implement the frame syntax (the field types and sequence). They are also in charge of triggering the processing units by driving the (S,E) couples of signals. Each control part is modeled by an array (see figure 4) in which the rows refer to the different (possible) fields $\{f_i\}$ of the different frames and the columns represent the features of each one of them (size, type, function, ...). The receive/transmit control part is synchronized by the receive/transmit bit clock. For each field being processed, a different state of the array is entered depending on the sequencing conditions (in the case of more-than-one-bit wide field, the current state remains the same).

2.2.2 The application-side interface

The application-side interface must allow the communication circuit to exchange data with the host (the station). It essentially depends on the complexity and on the "level of intelligence" of the later (microprocessor-based system, hardwired station, ...). One can functionally separate this interface into two parts: an adaptation part and a memory buffer. The first part permits to adapt the signals of an external bus (the station-shared bus) to the internal circuit ones. The memory buffer can be of different sorts: specialized register set, FIFO queue, DPRAM,...

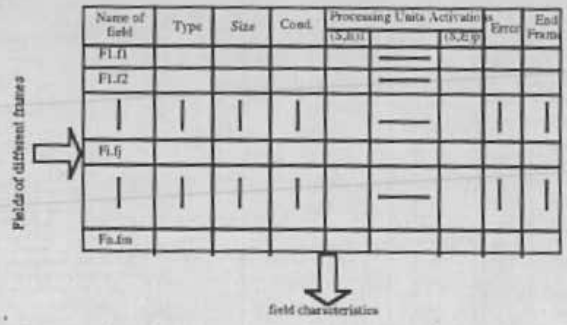


figure 4 - the array model of the control part

2.2.3 The medium-side interface

The medium-side interface implements the physical layer. It depends on both the medium characteristics and the communication protocol features and performs chiefly synchronization and data encoding/decoding tasks.

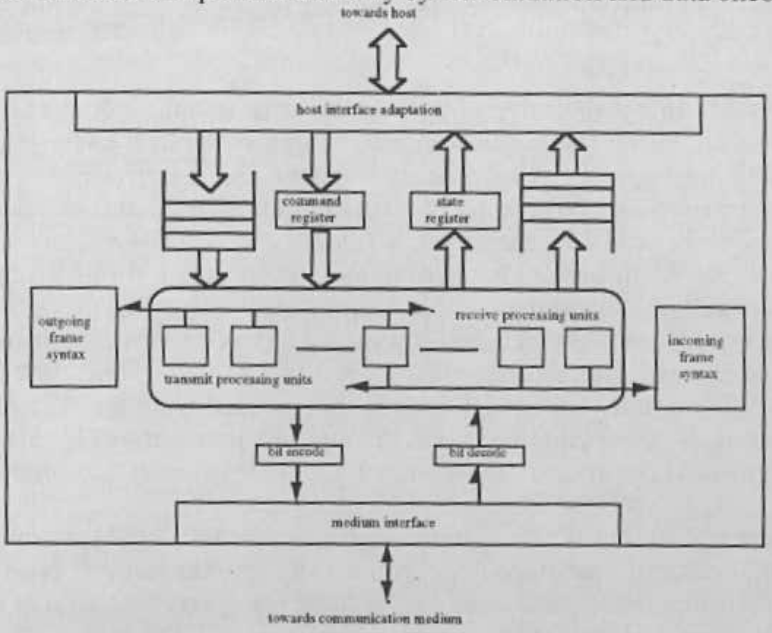


figure 5 - architecture of the communication circuit

Different control signals are exchanged with this interface such as the receive and transmit clocks, the start-of-frame detection, a contention detection, ... Figure 5 presents an architectural model of the MAC-level communication circuit

2.3 - Reception Machine Architecture on a Communication Circuit

As we have mentioned above, a VLSI communication circuit is composed of a certain number of processors (or machines) dedicated or allocated to the treatment of the parallel processes taking part in the transfer operation [12]. Let us suppose, for simplification purposes, that the circuit performs serial data transmission and reception. One principal machine will then be the serial interface managing the serial link. A parallel interface is usually available to manage the host microprocessor bus accessing the shared memory. Two data flows can be distinguished :

- data originated from the memory via the parallel interface, processed by a transmission machine and transmitted on the serial link via the serial interface, this is the transmission data flow, and
- data received over the serial link via the serial interface, processed by a reception machine and stored in the memory via the parallel interface, this is the reception data flow.

All the machines within the circuit are monitored by a control part to which other annex processors are linked ; let us mention for instance the initialization process, the error treatment process and others. Figure 6 and 7 give respectively a simplified synoptic of a communication integrated circuit and the Reception Machine block diagram. The role of the reception machine of a given layer is to process the frames provided by the distant transmission machine of the same layer. Let us give hereinafter the principal functional modules composing this processor whose modules are schematized in figure 7.

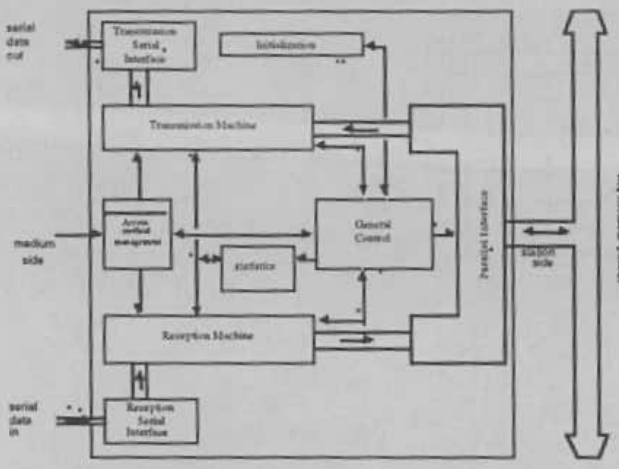


figure 6 - communication circuit functional partitioning

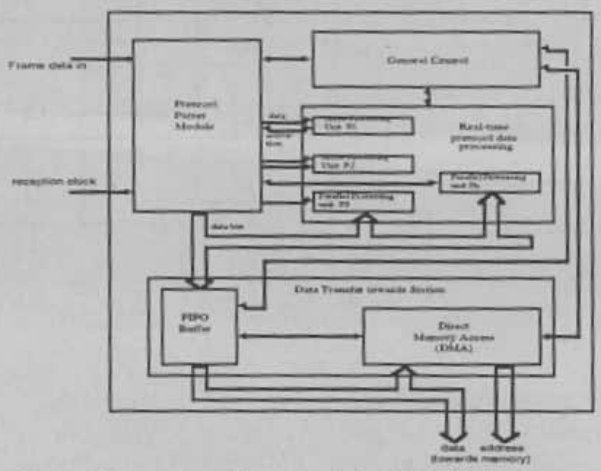


figure 7 - a reception machine block diagram

The **Protocol Parser Module** is in charge of parsing the serial incoming frame. This means that the module must "understand" and "consume" the frame skeleton providing a syntactical analysis process. This operation is accompanied with the frame parsing and cutting processes. That is the reason why we will call such a module : a parser. Data and other information and parameters contained in the frame fields are then supplied to specialized processing units which must be activated by the parser.

A set of Processing Units are in charge of treating information furnished by the parser. Examples of these units are : address recognition unit, network clock management unit, CRC processing unit and, naturally, the skeleton function-code recognition units which perform the matching between the incoming values and those of the definition set. On the other hand, signals originated from theses skeleton function-code recognition units must be used by the frame analysis module, a feedback loop is then necessary in the case of dynamic skeleton frame analysis. The total number of the processing units and the internal architecture (serial or parallel) of each one of them depend on the implemented communication protocol and the *degree of expected real-time treatment in the packet processing*.

It is fairly obvious that different types of function-codes demand different architecture implementations. The CRC processing units is naturally serial machine, whereas an address recognition is performed more easily in a parallel way. Basic implementation of these units can be done using serial or parallel comparison operators, the definition values of the function-code being for example stored in a ROM. A PLA-based implementation of the recognition binary tree can be also considered in the case of a serial bit-by-bit comparison.

In order to permit the storage of data in memory, the frame analysis module performs data deserialization into an internal bus. Parallel data is stocked in a FIFO queue, sequenced by a specialized control part, whose function is to match the speed of this internal bus to the shared memory one. The optimal FIFO size and its word size in function of the microprocessor working frequency have been studied in [15].

A Direct Memory Access Module is in charge of transferring the deserialized data, temporarily stocked in the FIFO, towards the host shared memory. Let us mention here that data which cannot be processed by the current level reception machine are generally stored in the shared memory and are to be processed by the software programs. The whole reception operation is globally sequenced by a General Control Part.

In the next section, we will present the protocol parser functional architecture. The other modules are not immediately concerned by the present study.

3 - The Protocol Parser Architecture

The necessity of frame cutting depends on the implemented protocol and particularly on the degree of the expected real-time treatment of some specific fields of the frame. Real-time treatment are carried out by the communication circuit without having recourse to microprocessor resources and programs[8][16].

Figure 8 gives a generic functional architecture of the protocol parser and figure 9 gives the generated Parser Module that follows it. The Serial Control Part is in charge of sequencing the frame analysis process. We suppose that the start-of-message field has been recognized earlier (in a serial manner) by an appropriate detection unit. The frame parser is then activated by a control signal. Furthermore, the control part communicates with the delimiting field recognition block which indicates the limits of the frame being processed. This block can be a simple ROM in which the skeleton sequence are memorized in the case of a static protocol implementation. The terms of this sequence are then furnished one after the other to a comparison device related to a frame bit counter. In a dynamic protocol implementation, however, this block is much more complicated. Feedback signals coming from the skeleton function-code recognition units are used by this block to compute the implicit terms of the sequence. Speed is a primordial factor in this case, especially if a real-time treatment is engaged.

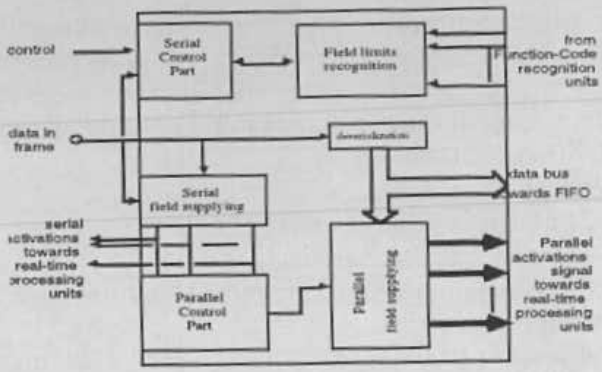


figure 8 - protocol parser block diagram

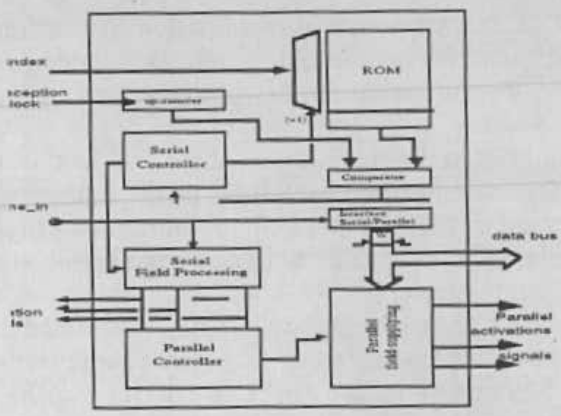


figure 9 - a static protocol parser architecture

The serial field supplying block delivers the field data on serial lines taken as derivations to the frame line. Moreover, *activation signals* are delivered to the concerned processing units in order to awake them and to tell them whenever data are available.

The interface serial-to-parallel block is a simple parallel shift register. Deserialized data are transported on a reception machine internal bus towards the FIFO queue. The same bus is used by the Parallel Field Supplying Block in order to deliver parallel data to the concerned processing units. Parallel activation signals are also furnished to these units via the Parallel Control Part, which treats the serial activation signals and outputs a set of control signals aimed at the parallel field supplying block.

The internal architecture of the parser components as well as that of the associated processing units depend on the implemented protocol specifications.

3.1 - A Dedicated Protocol Parser Silicon Generator

The term Silicon Generation (or the well-known but improper term Silicon Compilation) has been widely used in the domain of Integrated Circuits design since it was first introduced by Dave Johansenn in 1979 at the 16th DAC. Intuitively, the silicon generation denotes all the processes that lead to the final layout of an integrated circuit starting from a behavioral description of its functioning in the same way that the program compilation leads from a high-level language description of an algorithm to a machine-executable code.

With the arising of the circuit complexity, manual methods of design became less and less convenient. They offer reduced possibilities to evaluate timing, consumption and/or area figures of a given architecture so that we understand easily why the silicon compilation term is also applied to every program helping to reduce the complexity of the design process, even if it is not dedicated to layout generation. It is clear that ASICs are the main domain of application for silicon generation tools.

A module generator can be defined, in the most general case, as a reduced silicon generator which, taking into account the function of the module and some given parameters, outputs a logical model of the block plus its final layout [20].

One can distinguish two categories of module generators :

- Generators of general-purpose blocks[15], such as FIFO, DMA, ROM, RAM, PLA,... which are intended to output regular structures. Generation parameters are typically the number of word, the number of bits per word and the ratio aspect in the case of the ROM and RAM generators and the number of inputs/outputs plus a truth table in the case of the PLA generators. We can add to this category the data path generator which could be used to implement a circuit in a bit-slice style for instance[14].
- Generators of specific modules. In this case, a target architecture must exist. The generator takes as input a certain number of parameters (called parameters of flexibility) and generate the architecture according to them.

In the area of communication system design where system engineers are more and more invited to design VLSI circuits to implement communication protocols. Appropriate module generators[20], which accept a certain formal description of the protocol specifications in order to generate the corresponding frame analysis module, would be a very useful system CAD utility which relieve system engineers of the difficult VLSI circuit manual design task and avoid having recourse to professional VLSI circuit design engineers.

3.2 - The Protocol-Parser Module

3.2.1 - Functional Specifications

The object of this chapter is the validation of the frame-parser concept by a dedicated study. We will present a well-defined class of flexible modules intended to be automatically generated according to certain parameters, namely the number of fields within the frame and the size of each one of them. The FIFO bus width is another parameter of flexibility, yet its choice is not entirely open since it must be compatible with the direct memory access unit bus in the reception machine and the host microprocessor bus.

The target class of circuits is characterized by the following features :

- frames processed by the generated circuit must have a limited (though variable) number of fields ;
- the frame skeleton is statistically defined (dynamic function-codes will not be considered) ;
- the skeleton sequence term values will only be known at the generation time. This means that the target architecture is not intended to the processing of a determined protocol frame ;
- the considered function-codes can be either simple (without associated data) or, in the opposite case, pre-fixed. Post-fixed function-codes are not considered. Nevertheless, the CRC, which can be seen as an example of such function-codes can be treated if it is located at the end of the frame. The treatment manner is detailed in the case study below.
- generation parameters are inspired from a target sub-set of skeletons defined in OSI high-level protocol standards.

3.2.2 - The Architecture

The flexible circuit architecture has already been shown in figure 8. The values of the $\{b_i\}$terms which define the frame skeleton are stored, at the generation phase, in a ROM which corresponds to a given frame specifications. We have chosen, however, to enhance the dynamic character of this memory by storing more than one skeleton in it. An index signal will initialize the memory address counter in order to choose the appropriate stage. This is fairly useful if we want the circuit to process a group of frames whose skeleton is entirely defined by a code figured in the beginning of their header. A dedicated recognition unit is associated to this function-code in order to generate the index signal.

In spite of the use of a "layer-organized" ROM, the previous architecture remains somehow rigid. A more flexible choice consists in using a RAM, but then we come up against the problem of the writing interface in the memory at the initialization phase. A possible solution could be the use of the serial link (data in link) to enter the desired values in the memory, (notice that this is not usually done in communication circuit architecture).

Elements of the memory are compared, one after the other, to the value of the bit up-counter and the comparison result is sent to the serial control part. All the other module components have the same function explained hereinabove.

4 - The Parser Validation

4.1 - Case Study 1 : The HDLC protocol

The objective of this case study is to illustrate, via a widely-known example, the architecture that has been chosen, where it is generated the analysis module of a simplified **HDLC** protocol. The skeleton of a data frame is illustrated below (figure 10).

fanion	address	command	data	FCS	fanion
01111110	(8 bits)	(8 bits)	*variable*	(16 bits)	(8 bits)

figure 10 - HDLC data frame skeleton

The fanion field constitute the delimiting frame sequence, every frame must start and end by a fanion. The same flag can be used simultaneously as a closing fanion of a frame and an opening fanion of the next one. We will suppose, in our example, that this flag is detected by the start-of-message detection unit and an enabling signal is sent to the circuit in order to start functioning.

The address field (one byte size) must identify the secondary station(s) involved in the considered frame transfer operation.

The command field (one byte size) contains the commands of the primary station, the responses of the secondary one(s) and the sequence numbers. An example of its internal structures has been given above. In our case, we will consider it as a single field intended to only one processing unit. The HDLC frame can thus be seen as static.

The data field can be composed of an undetermined number of bits (in fact, HDLC is a bit-oriented protocol). Nevertheless, in practical implementations, it is often structured into characters. We will adopt, in this study, a multiple-of-bytes data field. Furthermore, we suppose that the zero elimination process has already been performed.

The Frame Check Sequence (FCS) field calculation at the transmission time takes into accounts all the bits between the start fanion and the first bit of the FCS field. It is a post-fixed function-code. The detection of the corresponding Cyclic Redundancy can be used to indicate the end of this field. Notice that the beginning of the field (the end of the data field) is not known, data can have an undetermined (but bounded) number of bytes.

As it has been already said, the simplified HDLC protocol we are implementing is a static protocol, it belongs to our generator target class. As the data field length is not known, we will load in the skeleton ROM the maximum authorized size. If the frame is error-free, the cyclic redundancy check (CRC) is used to determine the end of the FCS field. In this case, the data receiving unit must count the number of bytes it receives and not validate the last two of them.

The circuit simulation waveforms were obtained from the input frame given in the figure below :

01111110	10101010	00001000	1100001100111100	1110010010011011	01111110
	address	command	data	FCS	
	(SrData1)	(SrData2)	(SrData3)		

figure 11 - input frame stimulus

We can recognize the serial and the parallel activation signals corresponding to the address, the command and data fields (resp. SrAct[1..3] and PrAct1, PrAct2, PrAct3[1..3] in the simulation file). For clarity reasons, the serial frame derivations corresponding to each unit are figured (the equivalent name to every field in the simulation file is done above). Parallel data can be sampled from the parallel bus (PrData[0..7] in the simulation file) after each parallel activation pulse. The CRC detection signal indicates the end of processing, for this reason, the third signal of the parallel activation pulses corresponding to the data field (a ty5 field) is not used.

4.2 - Simulation waveforms and the width of the Data Bus

As we have said before, the deserialization width, noted W, is one of the generation parameters. This width has an important influence on the field processing manner. In fact, if we divide the frame into segments of size W(figure 12), we can distinguish five types of fields :

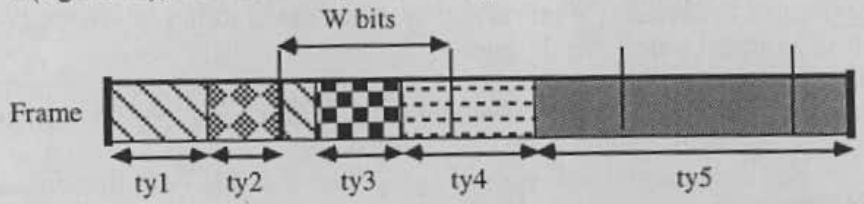


figure 12 - frame segmentation

- ty1 : The field width is less than W, its beginning coincides with a W-division instant ;
- ty2 : The field width is less than W, its ending coincides with a W-division instant ;
- ty3 : The field width is less than W, it is entirely situated between two successive W-division instants ;
- ty4 : The field width is less than or equal to 2*W, but only one W-division separate it into two pieces ;
- ty5 : The field width is greater than W, it occupies several successive W-segments.

One can extract the following two relations between the different field types :

ty4 :: <ty2><ty1> ;

ty5 :: <ty2><W>*<ty1>.

Notice that, the ty4 type can be seen as a particular case of the ty5 type. In figure 13 it is illustrated the functioning of the circuit by means of waveforms, we can distinguish the serial activation signals and the parallel activation signals of the correspondent real-time processing units.

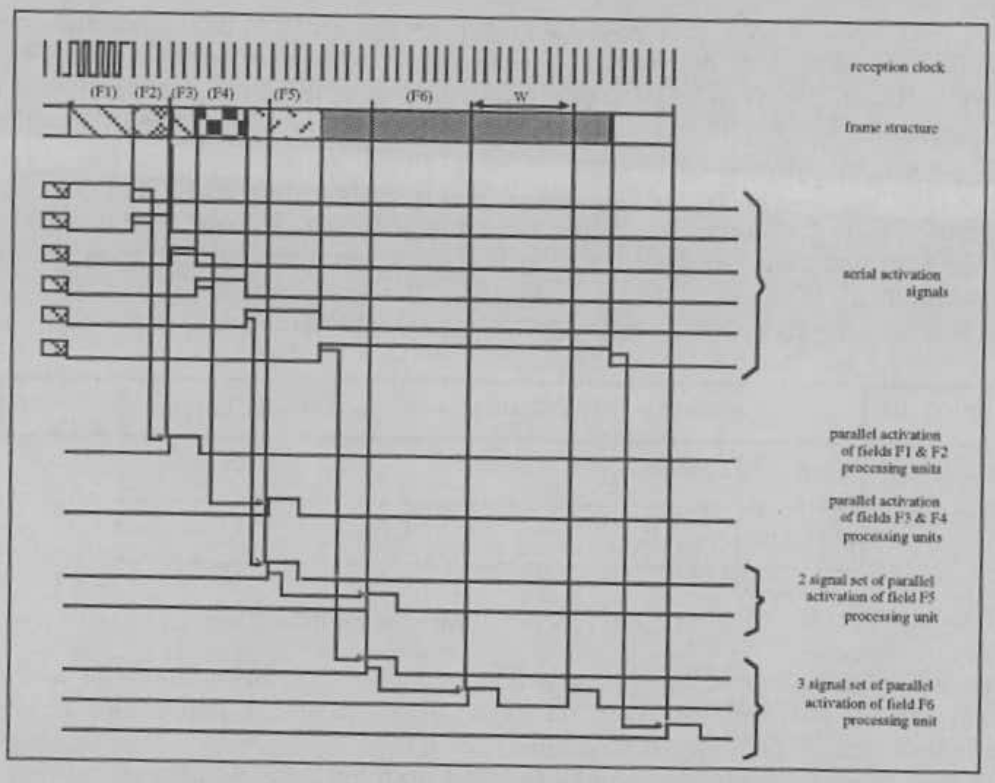


figure 13 - derived circuit signals to activate real time processing units

The serial activation signals, delivered by the serial field supplying block and sequenced by the serial control part, delimit each processed field. A typical use of these incoming signals is the enabling of a 3-state gate having as input a derivation to the frame serial link. There is as many 3-state gates as the number of fields in the considered frame. To every serial activation signal corresponds one or a set of parallel activation pulse(s) which indicates the first data availability instant on the internal bus.

For ty1, ty2 and ty3 types, the rising front of the serial activation signal is translated to the following W-division instant to construct the parallel activation signal (one cycle duration).

For ty4 type, corresponding field information must be activated twice since it is separated into two parts belonging to two successive W-segments ; a group of two signals is then needed, the first pulse is generated as before, the second pulse having its rising front W cycles afterwards.

As for ty5 type, a set of three pulse signals is necessary : the first one is generated as in ty1 type, the second pulse having its rising front W cycles afterwards and repeated as long as the associated serial activation signal does not fall down within a W-segment, the third pulse having its rising front coincided to the W-instant following the falling front of the same serial activation signal. (see functioning waveforms for illustration).

All these signals are used for example to load different parallel registers, this is the reason why we have preferred not to carry the ty4 or ty5 set of pulses on only one physical wire.

4.3 - Case Study 2 : Implementation of AAL3/4 protocol

The ATM protocol reference model, following the principles of a layered communication, has three identified layers: Physical, ATM and ATM Adaptation. All the layers above the ATM Adaptation layer are referred by the name Higher Layers.

The two lowest layers of the model are completely defined and some components already exist. The physical layer deals with the access to the medium and cell delineation. The ATM layer is in charge of cell header generation and extraction, cell demultiplex and multiplex and virtual channel identification. The ATM Adaptation Layer (AAL) functions [20] are only performed at the network boundaries. The specification of this layer has been made in the international standards organization (ITU, ANSI, ...). It performs functions required by the user, control and management planes. Its functions depend on the higher layer requirements. It is split in two parts: the Segmentation And Reassembly (SAR) sub layer deals with the segmentation and reassembly of message, lost or miss-inserted cell detection and multiplexing/demultiplexing and the Convergence Sublayer (CS) is in charge of error detection and handling and identification of information.

4.3.1 Protocol specification

This section presents the concepts of VHDL used to write functional specifications of a protocol. This presentation is based on an example taken from the Segmentation And Reassembly (SAR) sublayer of the AAL3/4 as normalized by the ITU&T. It deals with the control of ATM cells sequencing inside a message, by means of the Sequence Number (SN) and the SAR Type (ST) fields. The SN field is a number incremented for each new cell in the message. The ST field discriminates the first cell (BOM : Beginning Of Message), the last

cell (EOM : End Of Message) and others cells (COM : Continuation Of Message) of a message. The SSM value in the SN field indicates a message containing only one ATM cell. A finite state machine in accord with the ST control algorithm defined in [17] is implemented to control the SAR34 Type field. This controller is necessary to ensure that no cell on a connection has been lost or miss-ordered. But it must be done simultaneously for all multiplexed connections opened at the current time. The VHDL source code corresponding to the state diagram is described in [18]. The process implementing the state diagram reacts on two events: the arrival of a new cell (IN_SYNC) and the timer alarm (SAR_TIMER_IN). All parameters of the entity are bounded to a validity signal like at the layer interface.

To manage the cell multiplexing on ATM connections, we must use tables containing the current state, the last value of SN field encountered on all the connections (ST_SN_TABLE_OUT_PARAM), the connection parameters (maximum delay between two consecutive cells) (SAR_DELAY_TABLE_OUT_PARAM). These tables are defined in a package like record of the previous objects. A timer is used to control the delay between two consecutive cells on a same connection. It is accessed by a signal named SAR_TIMER_OUT_PARAM containing the connection identifier, the command (START, STOP, ...) and the time at which it must ring (NOW + SAR_DELAY_TABLE_IN_PARAM). The last parameter is a time value containing the current time of the simulation incremented by the maximum delay allowed for this connection. The SAR timer is implemented with a scheduling list, i.e. an array of dates (VHDL dates) sorted by time. Functions and procedures are used in the timer process to update this scheduling list.

4.3.2 Services Specification

We describe, in this section the method we have used to specify service in ATM systems. Examples illustrating this step are taken from the AAL3/4 functional implementation.

A layer is represented in VHDL by a design entity. Service primitives offered by this layer are represented by the "ports" (signal parameters) of the design entity. "Ports" are typed signals. The data unit contains user data represented as an array of bytes (USER_DATA for example) and service parameters like for example ATM-CEI(Connection Endpoint Identifier).

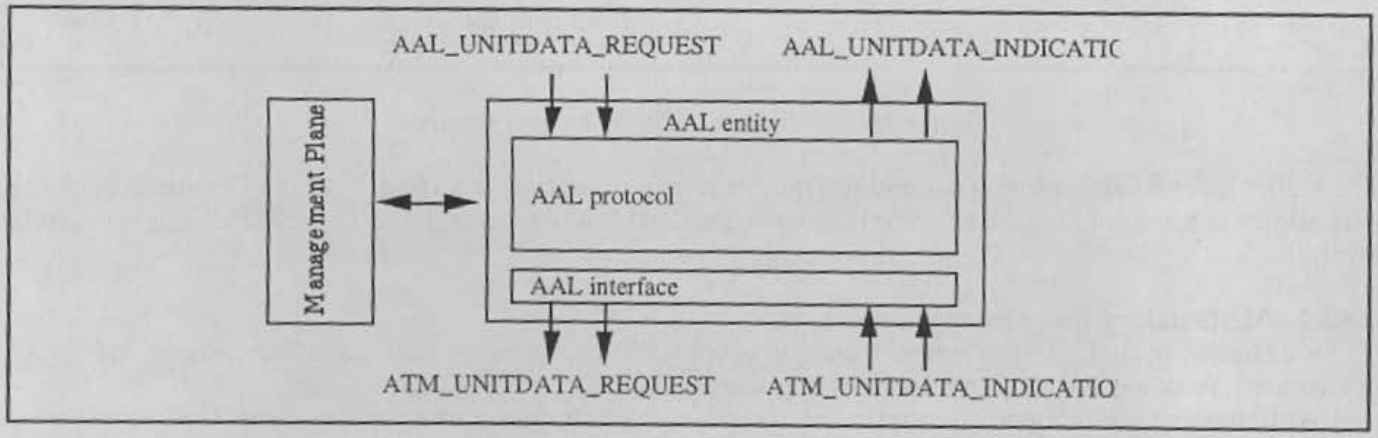


figure 14 - AAL Interfaces

As it is showed in figure 14, each service primitive is made of two parts: a data unit (ATM_UNITDATA_INDICATION_PARA) and a signal that indicates information validity (ATM_UNITDATA_INDICATION).

This second signal is necessary because VHDL does not immediately set complex signals. A gap has to be created to invalidate data during a short time when a signal is modified.

The entity describes, by means of its "ports", all interactions with upper layer, lower layer and management plane as shown on figure 14 for the ATM Adaptation Layer.

A data unit transferred between two layers doesn't have the same meaning for each of them. Indeed, the upper layer asks to the lower layer to send a data unit. This unit contains only user-data for the lower layer but it contains Protocol Control Information and user data for the upper layer. Thus we have implemented a translation mechanism at layers interface. This entity works in both directions (receive and send) and translates a non structured information into a structured information in the receipt direction and the opposite in the sending direction. This entity consists only in type casting and translation functions.

4.3.3 - Validation of the Protocol Parser Silicon Generator

Before the validation simulation operation, a top-down analysis of the circuit functionalities is done. This results in the definition of the different functional blocks and consequently of the operators (parallel registers, counters, comparators, ...) which make them up. The validation itself has been divided into two successive parts : the validation of the functional specification with a protocol layer test, and the consistency checking interface with the architectural description.

4.3.3.1 - Validation of functional specification

The validation of functional specification can be executed in two phases: first, each protocol layer is tested independently and then the global verification on all protocol layers is realized. VHDL makes these tests easier thanks to its modularity and the layer sharing.

The protocol layer test consists in connecting the layer to be tested with the management plane and an entity dealing with traffic generation and analysis. This control entity can take place either above (figure 15) or below the tested layer. The path at the top (respectively. bottom) of the layer is buckled to allow to test both directions, i.e. sending and receipt. Traffic is composed of upper layer Service Data Unit or lower layer Service Data Unit. In this configuration, packets can be analyzed at the top and the bottom of the entity, what is an interesting point to debug the protocol specification.

For the global validation, we connect together a sub-set or all layers of the protocol reference model.

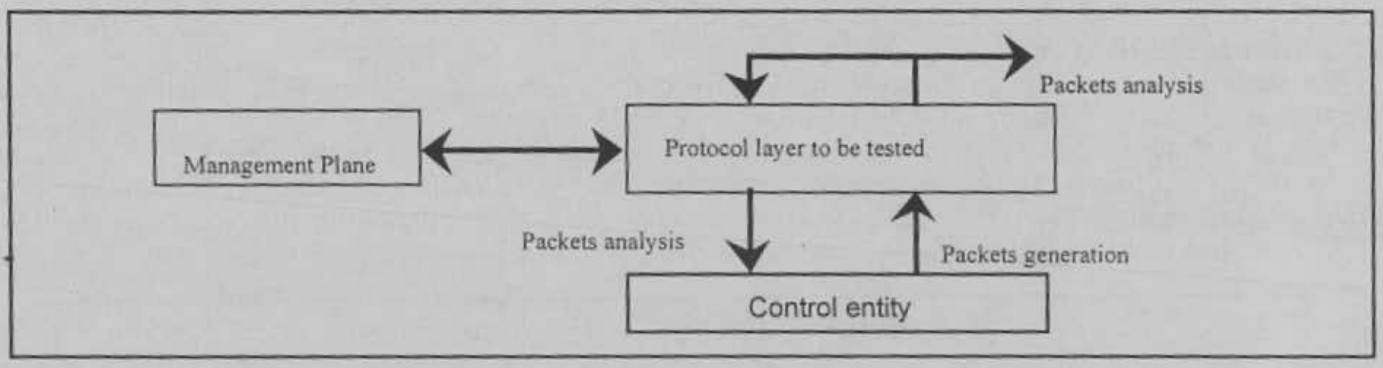


figure 15 - configuration for a protocol layer test

The higher layer is buckled on itself and the lower layer is connected to a control entity. The structure defined here allows to test conformance of packets at the higher and lower level and also at each interface between two layers.

4.3.3.2 - Methodology consistency checking

To describe an architecture from functional specifications and to be sure about consistency of the two descriptions, there are currently two methods in research progress:

- **Automatic Architectural Synthesis** (or High Level Synthesis) : using a functional description at the behavioral level, it gives a description at the register transfer level. Logic synthesis tools such as Design Compiler from Synopsys, Silsync from Racal-Redac, High-Design from Genrad and Autologic from Mentor Graphics allow to make the ultimate phase, i.e. the components and circuits synthesis on silicon. Many high level synthesis systems exist such as CMU-CAD and its successor the System Architect's Workbench from Carnegie Mellon University, Yorktown Silicon Compiler from IBM, and Amical from the IMAG-TIM3 laboratory.

- **Formal Verification**: using functional and architectural descriptions written by hand, it validates the consistency in a symbolic way. Formal verification groups two kinds of methods: the **deductive proof** that consists in transforming both the descriptions with symbolic manipulations to obtain a set of equations to be compared [18] and the **functional verification** that associates a model with the system, verifications being done on this model [17]. However, both these methods are currently limited by a sub-set of VHDL and by small sized descriptions. This section defines a by hand but systematic method to ensure the consistency between functional and architectural specifications, using some rules.

4.3.3.3 - Principle of the consistency checking

The consistency checking between functional and architectural specifications consists in simulating both descriptions independently. Produced results by each of them are compared. This verification depends on the tests quality.

Nevertheless, the organization of these two specifications is fundamentally different about tasks partitioning between entities, interfaces and entities structure. So, interfaces must be fitted to reuse test sequences realized during the functional specification step. In particular, signals of architectural specification interfaces must be converted to make them identical with functional specification interfaces. The consistency can be checked at the global level. The global design entity of architectural description must be encapsulated with a signal converter to

obtain compatible signals (see figure 16). The build of this signal converter is described in the next sections. Then, test sequences are injected in the architectural description and functional specification and results are compared to verify the consistency.

This method can also be used for a sub-entity. Indeed, a design entity can be described at an architectural level although the remainder of the system is described at a functional specification level.

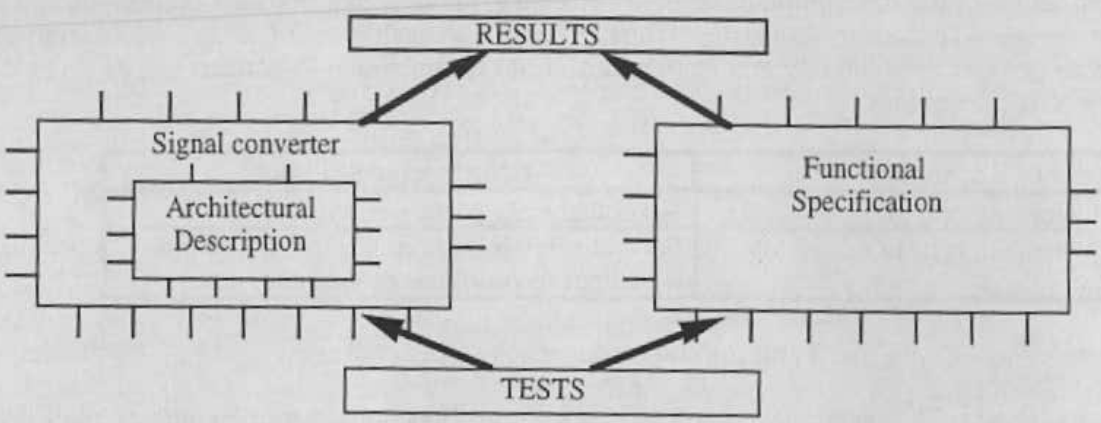


figure 16 - configuration for the global level consistency checking

Thus, it is possible to test only a part of the system. Nevertheless, a specific signals converter must be developed. The structure of the system is only a recursive view of the global level one.

4.3.3.4 - Primary rules

This method needs the functional specification of the system and a set of time constraints associated with, and the architectural description of the system to be studied. Two important rules are the basis of this method. First, one or more entities of the architectural description must have exactly the same functions as the ones they replace. In other words, architectural description entities can't only execute a part of the functions of the replaced entities, the other part being done in other architectural description entities not inserted for the verification. Thus, functional and architectural descriptions must realize the same functions until a given abstraction level at which the verification is performed. This concept is useful for the consistency checking of a sub-set of the description. Indeed, the verification at the global level is accomplished on only one entity of each description, each of them executing all functions.

Secondly, the functional specification can't be entirely independent of the architectural description time characteristics. It is often necessary to synchronize the two descriptions modifying time criteria in the functional specification. Moreover, functional specifications without time constraints are out of mind. Indeed, this corresponds to operations executed in a null time, that is unrealistic in architecture.

4.3.3.5 - Signals converter building rules

The signals converter is the basis of the verification by simulation presented in this section. Several types of correspondence between functional and architectural descriptions signals can be identified. The different problems encountered are: different signals types, different signals number and different time constraints between functional and architectural descriptions. Of course, these problems can be combined in one interface. The first case deals with the homogeneous types conversion. We suppose in this case that a functional specification signal has only one corespondent in the architectural description, and we forget time constraints. A **conversion function** from the functional specification parameter type to the architectural description one and an other one in the reverse way solve the problem. One function per distinct type of interface parameters must be define in each way. All these functions can be grouped in a VHDL package. Table 1 shows the homogeneous conversions. Example of hardware applications are put in brackets. The last conversion is often mixed with time constraints.

Functional specification signal type	Architectural description signal type
Integer	Bit vector (wires)
Enumeration	Bit vector (1 to N coding, minimum coding, ..)
Floating point	Fixed point representation Number and exponent representation
Array index	Address (Address bus)

Table 1 - homogeneous types conversion

The second case concerns the non homogeneous types conversion. We suppose, like in the previous case, that signal temporal constraints are the same for functional and architectural descriptions, the next part dealing with synchronization problems. In this case, a conversion function is insufficient. The system must react on events from functional and architectural descriptions to do the correspondence between signals. A **conversion entity** is necessary. Parameters of this entity are those of both descriptions. Functional specification parameters stay in the same way but architectural description ones are inverted, i.e. an *in* signal becomes an *out* one and an *out* signal becomes an *in* one. This entity consists of one process for each way of conversion (from functional to architectural to convert data entering and from architectural to functional to convert results). Table 2 shows the non homogeneous conversions.

Functional specification	Architectural description
1 signal with elementary type	Several signals (data + control)
1 signal with complex type (a record for example)	Several signals more or less complex (one signal per record field for example)

Table 2 - non homogeneous types conversion

The last case deals with synchronization of the descriptions. Functional specification describes the system with the functions it realizes and it doesn't involved in data transfer. The data transfer schedule between functional and architectural descriptions can be fundamentally different, for example with parallel or pipeline executions. Functional specifications often contains one signal per request or command. In the architectural description, we can choose to transfer data on a bus. We have several signals from the functional specification that must load a single signal concurrently. In all these cases, we must synchronize both the (functional and architectural) descriptions by means of clocks or operators of latency.

For example, we can look at the interface between the ATM and the ATM Adaptation Layers described earlier. We are going to extract the part dealing with the reception side of the protocol. The functional specification signals are *ATM_UNITDATA_INDICATION* and *ATM_UNITDATA_INDICATION_PARAM*. The architectural description signals are *cell_sync-in*, *byte_clock-in* and *cell_data_in*. The conversion entity needed to performs consistency checking between both the descriptions is presented below.

```
entity Convert_entity is
   port  ( -- Ports for communication with the ATM entity
           ATM_UNITDATA_INDICATION        : in BIT;
           ATM_UNITDATA_INDICATION_PARAM  : in LOWER_ATM_SDU;
           -- SAI_Rx interface
         cell_sync-in                     : out type_bit;
         byte_clock-in                    : out type_bit;
         cell_data-in                     : out type_data_transfer_8);
end Convert_entity;
architecture  Convert of Convert_entity is
begin    -- Conversion from functional to architectural
  process
    variable i :   integer;
  begin
 wait on  ATM_UNITDATA_INDICATION until ATM_UNITDATA_INDICATION = '1';
    cell_sync-in <= '1', '0' after delay_cell;
 for i in 1 to 53 loop
       cell_data_in <= Cell_to_Byte(ATM_UNITDATA_INDICATION_PARAM, i);
       byte_clock-in <= '1', '0' after delay_byte;
 end loop;
  end process;  -- Conversion from architectural to functional
end Convert_entity;
```

The *Cell_to_Byte* conversion function core is not very interesting. It consists of taking the i^{th} byte of the parameter. The reverse order conversion (*Byte_to_Cell*) performs the translation from the architectural parameters to the functional signals. The core of the conversion entity has two processes, one in each conversion way.

5 - General Conclusions

We have shown in this paper that VHDL provides interesting features to design communication protocols. We can underline the work done with the same language in the same environment all along the conception, the means to set executable specifications and the use of time at each level of the description. The functional specification

contains more than 15000 lines of VHDL code. The architectural description and specified integrated circuits are currently being developed by SGS Thomson (France) and KTAS (Denmark).

The previous study has been done from the point of view of the realization of certain system CAD utilities to help system engineers in designing VLSI circuits implementing communication protocols. Within studies intended to the communication circuit architecture synthesis, the ultimate goal is the development of a silicon architecture generator of protocol parsing modules using a formal high-level description of the frame skeleton.

We will briefly discuss below the formal description tools that could be used in this work for the sake of comparisons :

- Distributed system description languages normalized by the ISO can be used to combine the specifications of protocol and services and those of frame skeletons. This track is worth thinking over for obvious reasons of clarity and compactness of description. Nevertheless, the existing languages at the present time, such as LOTOS and ESTELLE, do not to offer possibilities for all levels specification and for hardware silicon implementation.
- All-purpose programming languages like ADA and C constitute another important possibility. These languages offer the advantage of being known by a wide population but they are not specifically designed to describe communication protocols or hardware components.
- A description based on the graph theory and/or Petri networks can also be considered if it is combined with a distributed system description method.

Whatsoever the tool one shall use, it must have the important following features which we've encountered in VHDL :

- The description tool must be completely independent of the design technology. This is a well-known principle in the silicon compilation domain
- It must be able to follow, completely or partially, the protocol specification derivations.
- The generated target architecture must be verified and validated before being used. The tool does offer the possibility of a formal verification of this architecture, the simulation-based validation of all of the possible target circuits being simply out of question.

Another conclusion concerns the protocol implementation complexity. In fact, due to the variety of function-code nature, corresponding protocol frames have skeleton whose definition can be more or less complex. If we want to perform hardware implementation of these protocols then some evaluation tools become necessary to draw comparisons between the different choices of frame skeletons and real-time processing units and then between the different generated architectures.

Circuits generated by a silicon generator have usually less performance (speed criterion) and are less optimized (surface criterion) than full-custom integrated circuits. In our case, the rapid availability of results must be motivating enough to encourage the choice of silicon generation tools and to facilitate the choice of a hardware solution to the protocol implementation problem.

REFERENCES

[1] "A Binding Architecture for Multimedia Networks", A. Lazar, S. Bhonsle, K.S. Lim. Disponível por WWW em : http:/samurai.ics.hawaii.edu/related_works.html, Fevereiro 1996.

[2] "Multimedia Document Handling - A Survey of Concepts and Methods", Computer Science Report, Universidade de Sttugart. Disponível por ftp anônimo em: ftp.fokus.gmd.de/pub/step no arquivo multimedia.ps, Fevereiro de 1996.

[3] "Three Competing Design Methodologies for ASIC's: Archit. Synthesis, Logic Synthesis and Module Generation", K. Keutzer, Proc. of the 26th ACM/IEEE DAC, 1989, pp. 308-313.

[4] "Document Conferencing: Real Time, Real Data", David Newman; K. Tolly, Data Communications, p81-94, Junho de 1995

[5] "Translation of Formal Protocol Specific. to VLSI Designs", A. Krishnakumar, B. Krishnamurthy & K. Sabnani, Protocol Spec., Test. and Verification, VII IFIP1987. p375-390.

[6] "Architecture of a new ASIC data communication circuit intended for the ISO levels 3, 4 & 5", Ansade, Ng. Dang, M. Diaz-Nava, G. Michel, J. Rarivomanana, L.Sponga. Proceedings of the 1988 IEEE International Symposium on Circuits and Systems. Helsinski. 6/1988.

[7] "An ASIC Architecture for Implementing Low-level Communication Circuits", I. Sabouni, H. Saheb & M. Dang, The Fifth Annual IEEE ASIC Conference and Exhibit, 1992.

[8] "RSVP: A New Resource ReSerVation Protocol", Lixia Zhang et all, IEEE Network, v.7, n.5, p8-17, Setembro 1995.

[9] X25, Packet level. Bluebook.CCITT. 1992.

[10] IBM leaks performance details of forthcoming TR network. IBM.. Data Commun.. 6/1996.

[11] IEEE Local Area Network Standards, IEEE Computer Soc. Silver Spring, MD. 1992.

[12] The effect of High Speed LANs on the design of the VLSI commun. circuits.Ng. X. Dang - P. Rolin , L. Sponga , G. Votsis. IFIP WG 6.4 Workshop on High Speed LANs. Aachen. 2/1987.

[13] Performances of a Communicating Circuit Intended for the Implementation of the High Level Layers of the OSI Model. C. Diot, Ng. X. Dang . 1989 IEEE International Conference on System Engineering. Dayton. Ohio-8/1989.

[14] "Étude et Implantation des Modules DMA et FIFO" - Jorge GUEDES SILVEIRA - Mémoire de DEA en Microelectronique - Université Joseph Fourier de Grenoble - France - Sept 1989

[15] "A Special-Purpose Flexible FIFO Design intended for Commun. Circuits" - J. Guedes Silveira, H. Saheb, I. Sabouni & Michel Dang - Proceedings of the 3rd International Conference on Microelectronics ICM'91 - Caire, Egypt - Dec 21-23, 1991.

[16] "High Speed Interworking Architectures" - J. Guedes Silveira- Michel Dang et al. -Proceedings of the First International Symposium on Interworking - INTERWORKING'92 - Bern, Switzerland - November 18-20, 1992

[17] "A VHDL based Methodology to Specify and Design ATM systems" - J.F. Guillaud, N. Idirene, J. Guedes Silveira et Gérard Michel - Procedings of the VHDL_Forum for CAD in Europe Spring '93 Meeting- Innsbruck, Austria- March 14-17, 1993

[18] "Utilisation de VHDL pour la spécification fonctionnelle du protocole ATM" - JF GUILLAUD, N. IDIRENE et Jorge GUEDES SILVEIRA-INPG - afcet 93 - Versailles, 8-10 Juin 1993

[19] "Specific ICs for High Speed Internetworking" - J. Guedes Silveira, .F. Guillaud, Michel Dang et Gérard Michel - SBMicro 93 - Campinas - Brasil - September 8-10, 1993

[20] "Conception de modules flexibles destinés à l'implantation de protocoles de communication dans les réseaux structurés en cellules" - J. GUEDES SILVEIRA - Tese de Doutorado na obtenção do titulo de "Docteur de l'Institut National Polytechnique de Grenoble" - France , 26 Novembre, 94.