MASTER

Design of an ASIC model in SDL

Ait Moulay, M.

Award date:
2000

Link to publication
Design of an ASIC model in SDL

M. Ait Moulay

Coach: dr.ir. J.P.M. Voeten
dr.ir. G. de Jong en ir. S. de Smet (Alcatel Bell Antwerp)
Supervisor: prof.ir. M.P.J. Stevens
Period: March – December 1999

The Faculty of Electrical Engineering of the Eindhoven University of Technology does not accept any responsibility regarding the contents of Master’s Theses
Abstract

The HdS (hardware dependent Software) department of Alcatel Bell Antwerp, part of the Carrier Data division is responsible for the design of embedded software, firmware and off-line test software towards verification and test of ASICs, PCB-boards and complete systems.

The HdS designers are interested in an executable hardware model of the system to be designed as a virtual target for debugging the HdS verification software. The model should have the capability to be interfaced with the HdS verification software, implemented in C, so that validation and co-verification of the SW/HW system can be performed during the top-level design phase. This model can also be used by the ASIC-designers as a reference model and for early validation of the architecture decisions in the design trajectory.

Starting from an ASIC specification, it is important to investigate what abstraction level is needed to satisfy both ASIC and HdS engineers. Quantitative information, about the effort required for designing a HW model has been also connected. This information will be indispensable for management planning in case HW modeling will be introduced in the design flow.

The results of this project can be used as a basis for further development of a co-design methodology.

The ASIC will be modeled in SDL. SDL was originally developed for the specification and description of communicating systems consisting of hardware and software, however SDL is currently mainly used in the software design process only.

In this thesis, a part of the MTM (Multi Path Self-Routing Traffic Manager) ASIC is modeled in SDL. Through simulation, the model is validated and verified. After that, the possibilities are studied to integrate the HdS verification SW with the HW model for co-verification purposes.
Acknowledgements

I want to thank a number of people who helped me during this project.

First, I want to thank Jeroen Voeten who encourages me to choose for this project. Next, I want to thank Prof. Stevens for his interest during this project.

Also, thanks to Stefaan de Smet for his help, the nice co-operation and the discussion we had during the whole project. Thanks to Gjalt de Jong for his advises and interest in this project.

Thanks for Jos van Sas for the opportunity to work on this interesting project.

Finally, I would like to thank the ASIC and HdS engineers who helped me to understand functionality of the ASIC and the software.
List of figures

Figure 1: hierarchy representation in SDL ................................................................. 5
Figure 2: basic principle of SDL process diagram ..................................................... 6
Figure 3: Example of an ADT ................................................................................. 7
Figure 4: Example of a Message Sequence Chart ..................................................... 13
Figure 5: the environment of MTM ASIC ............................................................... 14
Figure 6: MSC-8 format ......................................................................................... 15
Figure 7: functional block diagram TLK .................................................................. 16
Figure 8: MTM Core Outline, without LoopBack units, TG and TA ...................... 18
Figure 9: connection of LoopBack units, TG and TA to Egress and Ingress .......... 19
Figure 10: Egress functional blocks ....................................................................... 20
Figure 11: Block diagram of the system .................................................................. 26
Figure 12: block diagram of The Egress ................................................................. 26
Figure 13: SDL block diagram of the Traffic Generator ......................................... 27
Figure 14: block diagram of the Traffic Analyzer .................................................. 28
Figure 15: block diagram of the OBC .................................................................. 29
Figure 16: Process diagram of MSC_Reception ..................................................... 30
Figure 17: SDL process diagram of Routing-Multicast .......................................... 31
Figure 18: Process diagrams of the Queue Acceptance Block ............................... 31
Figure 19: SDL representation of the Queue Manager ........................................... 32
Figure 20: SDL representation of MSC_ATM_Translation module ...................... 34
Figure 21: structure example of a generated application ..................................... 42

List of tables

Table 1: SONET/SDH hierarchy ............................................................................ 16
Table 2: Simulation results with maximum input load ........................................... 36
Table 3: Simulation results with maximum input-output traffic ............................ 36
Table 4: Simulation results by one connection ....................................................... 37
Table 5: simulation results by eight connections ................................................... 37
## Abbreviation

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAL</td>
<td>ATM Adaptation Layer</td>
</tr>
<tr>
<td>ADT</td>
<td>Abstraction data Type</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>CCITT</td>
<td>Comité Consultative de Télégraphe et de téléphone</td>
</tr>
<tr>
<td>CCS</td>
<td>Calculus of Communicating System</td>
</tr>
<tr>
<td>CCD</td>
<td>Clock Conversion and Distribution</td>
</tr>
<tr>
<td>NRT</td>
<td>Non Real-Time</td>
</tr>
<tr>
<td>EFSM</td>
<td>Extended Finite State Machine</td>
</tr>
<tr>
<td>ERN</td>
<td>External Reference Number</td>
</tr>
<tr>
<td>FAB</td>
<td>Frame Aware Buffer</td>
</tr>
<tr>
<td>FDT</td>
<td>Formal Description Technique</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>FMS</td>
<td>Frame Merging Session</td>
</tr>
<tr>
<td>GSN</td>
<td>Group Switching Network</td>
</tr>
<tr>
<td>Hds</td>
<td>Hardware dependent Software</td>
</tr>
<tr>
<td>IDRFC</td>
<td>Internal Dynamic Rate based Flow Control</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>IRN</td>
<td>Internal Reference Number</td>
</tr>
<tr>
<td>LOTOS</td>
<td>Language Of Temporal Ordering and Specification</td>
</tr>
<tr>
<td>LT</td>
<td>Line Termination</td>
</tr>
<tr>
<td>MBC</td>
<td>Mounted Board Controller</td>
</tr>
<tr>
<td>MMM</td>
<td>MPSR Memory Manager</td>
</tr>
<tr>
<td>MPSR</td>
<td>Multi Path Self-Routing</td>
</tr>
<tr>
<td>MSC</td>
<td>Multi Slot Cell Message Sequence Chart</td>
</tr>
<tr>
<td>MSE</td>
<td>Multiplexing and Switching Element</td>
</tr>
<tr>
<td>MTM</td>
<td>MPSR Traffic Manager</td>
</tr>
<tr>
<td>OMT</td>
<td>Object Modeling Technique</td>
</tr>
<tr>
<td>PHT</td>
<td>Police Header and Translation</td>
</tr>
<tr>
<td>POOSL</td>
<td>Parallel Object Oriented Specification Language</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>ROM</td>
<td>Read Only Memory</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RT</td>
<td>Real-Time</td>
</tr>
<tr>
<td>SB</td>
<td>Slot Body</td>
</tr>
<tr>
<td>SC</td>
<td>Slot Control</td>
</tr>
<tr>
<td>SDL</td>
<td>Specification and Description Language</td>
</tr>
<tr>
<td>SDL-GR</td>
<td>SDL Graphical Representation</td>
</tr>
<tr>
<td>SDL-PR</td>
<td>SDL Phrase Representation</td>
</tr>
<tr>
<td>SDH</td>
<td>Synchronous Digital Hierarchy</td>
</tr>
<tr>
<td>SONET</td>
<td>Synchronous Optical Network</td>
</tr>
<tr>
<td>SM</td>
<td>Switching Module</td>
</tr>
<tr>
<td>SRT</td>
<td>Self-Routing Tag</td>
</tr>
<tr>
<td>STI</td>
<td>Scheduling Time Interval</td>
</tr>
<tr>
<td>STM</td>
<td>Synchronous Transfer Mode</td>
</tr>
<tr>
<td>STS</td>
<td>Synchronous Transmission Signal</td>
</tr>
<tr>
<td>TA</td>
<td>Traffic Analyzer</td>
</tr>
<tr>
<td>TG</td>
<td>Traffic generator</td>
</tr>
<tr>
<td>TLK</td>
<td>Termination LineK board</td>
</tr>
<tr>
<td>TLTB</td>
<td>TLK Line Termination Board</td>
</tr>
<tr>
<td>TSU</td>
<td>Terminal Sub-Unit</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
</tbody>
</table>
Table of Contents

1 INTRODUCTION .................................................................................................................................... 1
   1.1 SPECIFICATION OF SYSTEMS ............................................................................................................ 1
   1.2 POSSIBLE APPROACHES ....................................................................................................................... 1
   1.3 PROJECT DESCRIPTION .......................................................................................................................... 2
   1.4 MODELING LANGUAGE .......................................................................................................................... 3
   1.5 TOOL SUPPORT .................................................................................................................................. 3
   1.6 OUTLINE OF REPORT ............................................................................................................................. 4

2 OVERVIEW OF SDL .................................................................................................................................. 5
   2.1 INTRODUCTION .................................................................................................................................. 5
   2.2 STRUCTURE ...................................................................................................................................... 5
   2.3 BEHAVIOR .................................................................................................................................... 5
   2.4 DATA .............................................................................................................................................. 6
   2.5 COMMUNICATION ................................................................................................................................. 7
   2.6 NEW FEATURES IN SDL ......................................................................................................................... 7
   2.7 CONCLUSION .................................................................................................................................. 8

3 HARDWARE MODELING IN SDL ........................................................................................................... 9
   3.1 INTRODUCTION .................................................................................................................................. 9
   3.2 MODELING FEATURES IN SDL ............................................................................................................. 9
       3.2.1 Structure .................................................................................................................................. 9
       3.2.2 Communication ............................................................................................................................ 9
       3.2.3 Data ......................................................................................................................................... 9
   3.3 ABSTRACTION LEVEL ........................................................................................................................... 10
   3.4 MODELING STRATEGY .......................................................................................................................... 10
   3.5 VERIFICATION AND VALIDATION ......................................................................................................... 11
   3.6 USE OF MESSAGE SEQUENCE CHARTS .............................................................................................. 12
   3.7 CONCLUSION .................................................................................................................................. 13

4 MTM FUNCTIONAL DESCRIPTION ......................................................................................................... 14
   4.1 INTRODUCTION .................................................................................................................................. 14
   4.2 MPSR SWITCH MATRIX .......................................................................................................................... 14
       4.2.1 Switching principles ....................................................................................................................... 14
       4.2.2 Multi slot cell ............................................................................................................................... 15
   4.3 TERMINATION LINK BOARD .................................................................................................................. 15
   4.4 MTM ASIC ....................................................................................................................................... 17
       4.4.1 The Traffic generator and Traffic Analyzer .................................................................................. 18
       4.4.2 Ingress ..................................................................................................................................... 19
       4.4.3 Egress ..................................................................................................................................... 19
       4.4.4 Common part ............................................................................................................................... 23
   4.5 CONCLUSION .................................................................................................................................. 23

5 EGRESS MODELING ............................................................................................................................. 24
   5.1 INTRODUCTION .................................................................................................................................. 24
   5.2 ABSTRACTION LEVEL ........................................................................................................................... 24
   5.3 ARCHITECTURE DESIGN ....................................................................................................................... 25
       5.3.1 The Egress pipe ............................................................................................................................. 26
       5.3.2 The Traffic Generator .................................................................................................................... 27
       5.3.3 The Traffic Analyzer ..................................................................................................................... 28
       5.3.4 The OBC .................................................................................................................................... 28
   5.4 DETAILED DESIGN ............................................................................................................................... 29
       5.4.1 The Egress .................................................................................................................................. 29
       5.4.2 The Traffic Generator ................................................................................................................... 34
       5.4.3 The Traffic Analyzer ..................................................................................................................... 35
1 Introduction

1.1 Specification of systems

Specification is an important step in the system development process. It is a point of reference for the system developers during the design process. Conventional specifications are usually written in a natural language. This gives the illusion of being easily understood, but leads to ambiguity, omission and lack of understanding. As size and complexity of systems increase, specification documents become lengthy and difficult to check for completeness and correctness.

To cope with this problem, modeling languages can be introduced in the development process. Modeling languages are supported by a set of software tools that aid in the specification process, help to master the complexity and reduce development time.

Using modeling languages, an executable specification can be built, which will serve as documentation during all phases of the development process. Furthermore, describing a system through an executable specification in the earliest stage of the design trajectory will help to avoid taking wrong decisions. These wrong decisions may jeopardize the requirements of time to market.

An executable specification can be simulated. This provides the verification and validation of the system-properties. Simulation can be performed even before the design phase is started.

1.2 Possible approaches

A great number of methods and languages can be used for hardware modeling at system level. In this section, an overview of some of those languages is given.

VHDL is an IEEE standard used for the development, synthesis, verification and documentation of hardware design. It allows descriptions at three different levels: structural, data flow and behavioral.

This language is very popular amongst European hardware designers today. However, when applied to system specification they require a very detailed behavioral description. It forces the designer to make design decisions in the specification phase. Therefore, VHDL is not suitable for making high level specifications.

The same meaning hold for the Verilog language meanly used in the USA and Japan.

SDL is a Formal Description Technique (FDT) based on the Extended Finite State Machine (EFSM) model. A system described in SDL (Specification and Description Language) is seen as a set of concurrently active processes that communicate using messages through signal routes. The communication is asynchronous. Every process has an input FIFO (First In First Out) in which arriving signals are stored.

Estelle is a formal description technique well known in the telecommunication area. Like SDL, it is based on the extended finite state machine model. An Estelle specification defines a system of hierarchically structured state machines. The communication between the state machines is achieved by exchanging messages through bi-directional channels.
Estelle supports asynchronous communication. More details on Estelle can be found in [Tur93].

**LOTOS** (Language Of Temporal Ordering Specification) is a formal description technique that is based on CCS (Calculus of Communicating Systems) and CSP (Communicating Sequential Processes). Systems and their components are represented in LOTOS by parallel processes, which communicate through gates. LOTOS permits modeling of synchronous communication. More details on POOSL can be found in [Tur93].

**POOSL**: (Parallel Object Oriented Specification Language) is a very young modeling language. It is based upon the object-oriented paradigm as well on the basic concepts of CCS. A specification in POOSL consists of a fixed number of statically interconnected distributed processes, which synchronously communicate through channels by exchanging messages. A message may carry parameters in the form of data objects. POOSL is a language with a great potential that searches for a place between the specification languages. This language is not yet introduced in large projects in industry, because of the absence of commercial support. More details on POOSL can be found in [Pdv97].

**Conventional languages**: like Pascal, C and C++ are very popular in the industry and need no specific training. A lot of design engineers are using them to model their designs. However, these languages do not support concepts such as real-time and concurrency (only sequential programming is provided) required for system level modeling. Several attempts are made to use programming languages for hardware modeling. Attempts are made to extend these languages with proprietary libraries in order to cover the missing properties.

### 1.3 Project description

The broadband switching network of Alcatel is based on the Multi Path Self-Routing (MPSR) technology. Therefore, it is also named the MPSR switching Network. Various modules are used to provide the interface of external accesses to the MPSR switching matrix. One of these modules is the termination link board (TLK). The TLK allows to connect synchronous transport module level 1(STM-1) links (155 Mbps) to the MPSR switch. Those links contain virtual connections filled with ATM cells. To provide the MPSR switch interface with a flow control mechanism, the MPSR Traffic Manager (MTM) ASIC is developed for the TLK board. MTM consists of two main modules: The Ingress and the Egress. The Ingress is processing traffic from the user towards the switch whereas the Egress is working in the opposite direction (from the switch towards the user). Parallel to the MTM development, the Egress part is used as a test case to build an executable SDL model. For test purposes, a Traffic Generator (TG) and Traffic Analyzer (TA) are also modeled. The ASIC designers can use the model as an unambiguous specification during the ASIC design trajectory and also to verify some design decisions before the beginning of the implementation phase. The Hds engineers can use the SDL model as a target for the Hds verification software so that hardware / software co-verification can be performed.
When the SDL model of system consisting of the Egress, the Traffic Generator and the Traffic Analyzer is developed, it will be simulated for verification and validation purposes. After that, the possibilities are studied to interface it with the HdS verification software.

1.4 Modeling language

In this thesis, SDL is used to develop the ASIC model. It seems that SDL has achieved a wider acceptance in the telecommunication area than any other modeling language. The reasons of the tendency in using SDL in the industry can be attributed to the fact that SDL meets the main requirements of hardware systems:

- Hardware systems are parallel: hardware systems often consist of concurrent behaviors that together realize the functionality of the whole system. Processes in SDL run concurrently so that parallelism can be expressed.

- Hardware systems are distributed: hardware systems are often made of a set of basic computation units that may process data at different speeds and that may use different clock rates. The communication in SDL is asynchronous so that this kind of behavior can be described easily.

- Hardware systems are timed: in the specification of hardware systems, the timing concepts are very important for synchronization of parallel and distributed systems behavior. Timing also makes simulations of the system more realistic. SDL allows only for high-level specification of timing. This is adequate for making a high-level specification.

- Hardware specification languages should be recognized as a standard: this is very important for the industry when investments in training and tools are considered. SDL is a standard that is widely used, particularly in the telecommunication area.

Further, there has been a lot of research to generate VHDL from SDL specifications. Although there are problems that remain unsolved, there are algorithms that translate a limited subset of SDL to VHDL. Several approaches have been reported in [Glu91], [Ism93], [Bon95], [Had97] and [Dav97].

1.5 Tool support

SDL is well supported by tool sets containing editors, analyzers, transformers, test environments and code generators. During this project, the ObjectGEODE tool of Verilog has been used. ObjectGEODE consists of highly context-sensitive graphical tools. The environment is composed of four sets of tools for modeling, simulation, implementation and integration.

Modeling tools
Modeling is performed using four different tools: OMT Object editor, MSC editor, OMT Behavior editor and SDL editor.
Simulation tools
The SDL simulator allows performing different kinds of simulations such as interactive, random and exhaustive simulation.

Implementation tools
These tools generate the C code of the SDL parts of the system and C++ of the passive data objects.

1.6 Outline of report
In the previous section, a brief presentation of problems encountered by specifying systems is given. Some languages, which may be used to model the hardware, are also introduced. After that the problem description and some properties of SDL are given.

The rest of the report is divided in six chapters organized as follows:

- Chapter 2 presents of the SDL language.
- Chapter 3 explains the functionality and the environment of the MTM ASIC.
- In Chapter 4, a hardware model, composed of the Egress, the TG, the TA is built in SDL.
- Chapter 5 explains how to model hardware in SDL.
- Chapter 6 gives the possibilities to interface the hardware model with the HdS verification software.
- Chapter 7 presents the conclusions and recommendation for future work.
2 Overview of SDL

2.1 Introduction

SDL is a formal description technique, which is standardized by ITU (former CCITT) as Recommendation Z.100. The standard defines two representations of the language: a graphical one, known as SDL-GR and a textual one, known as SDL-PR. SDL is originally developed for describing communication protocols. However, SDL is widely used in many other industrial areas, such as in process control systems and high level programming. The first steady and complete version of SDL has been issued in 1988 as SDL-88. SDL provides three views of a system: structural, behavioral and data. These views will be explained in the next paragraphs.

2.2 Structure

A system in SDL consists of one or more block instances interconnected by channels. Four main hierarchical levels are supported: system, block, process and procedure. Each block can be partitioned into sub-blocks or will be described in term of processes. Figure 1 gives a hierarchical representation of a system in SDL.

![Figure 1: hierarchy representation in SDL](image)

2.3 Behavior

Process diagrams are used to describe the dynamic behavior of the system. Each process is represented by means of a state machine (see figure 2). A process can be activated either at the time of system initialization or at run-time.
A process waits in a state until it finds an acceptable signal in its queue. The signal is consumed and a transition is triggered. If a signal cannot be accepted in a specific state, it will be removed from the queue except when it is saved using the special construction save.

![Diagram of SDL process diagram]

**Figure 2: basic principle of SDL process diagram**

Within a process, procedures can be used to define behavior patterns. Like processes, procedures are also described by means of state machines.

### 2.4 Data

Variables in SDL are associated with a particular predefined or user-defined data type. All types are defined using Abstract Data Types (ADTs) or sorts.

A sort has the following specifications:

- A set of values.
- A set of literals defining a part of the value of a sort.
- A set of operations carried on these values.

An abstract data type consists of a set of sorts, which are associated via operations using values from different sorts. The properties of those operations can be specified by means of a set of equations. An example of an ADT is given in figure 3.
Newtype Bool
Literals true, false;
Operators
Not: Bool -> Bool;
Axioms
Not (true) == false;
Not (false) == true;
Endnewtype;

Figure 3: Example of an ADT

In SDL, all variables of a system should be described local to a process. Global variables are not supported. However, processes can make their data visible to other processes. The Reveal construction allows variables to be viewed by other processes in the same Block. Still the viewing processes cannot change the value of those variables. For processes from different blocks, the Import/Export mechanism allows to send a copy of a data item from one process to another (also read-only).

2.5 Communication

The communication between the entities of an SDL system is achieved by exchange of signals through channels. A signal may carry one or more values. Channels connect blocks with each other and the environment. They may be uni- or bi-directional. Processes are connected by signal-routes. A signal-route is a communication path without delay. The communication between processes is asynchronous. Therefore, there is an unbounded input buffer associated with every process. This buffer acts like a FIFO except when the special construction save is used.

In addition to the communication via channels, processes can communicate values via shared variables as explained in section 2.4. Nevertheless, the use of shared variables is not recommended in general because the make formal verification difficult.

2.6 New features in SDL

New features were introduced in SDL published in 1992 as SDL-92. A brief presentation of these new features is given below:

- Object orientation: object-oriented properties of SDL, such as genericity and inheritance, allow building reusable components and generic architectures.

- Additional non-determinism: provided by spontaneous transition using the construction none in an input list and undecided value generated by the new imperative operator any.

- Remote Procedure Call (RPC): the concept of RPC allows procedures to be called from other process instances than the one where they are defined. It also can be used to reuse parts of the behavior.
Alternatives to axioms: since axioms are difficult to understand, the properties of the operators of an ADT can be described using SDL procedures. It is also possible to solely state the signature of the properties in SDL and implement the behavior by means of other data formalism like ASN.1 or C.

In 1996, a new version of SDL was issued. This version is comparable to SDL-92 and it only provides some corrections and minor modification.

2.7 Conclusion

SDL is a formal language in evolution. Each four years, a new version of the language is issued. The ITU is trying to update the SDL language to the development in system modeling.
3 Hardware Modeling in SDL

3.1 Introduction

Today's systems are characterized by being very large and very complex. This results in specifications that are also very large and complex. Using SDL, these systems can be described at an appropriate level of abstraction so that the complexity of those systems can be mastered. This is possible because SDL concentrates only on the behavior of systems, which may include data if necessary. The specification and description of other aspects is outside the scope of SDL.
In the next sections, the use of SDL in hardware modeling will be presented.

3.2 Modeling features in SDL

3.2.1 Structure

An SDL specification contains a hierarchy of elements or components, which are ordered in a tree-like structure. This gives the possibility to model a system having different levels of abstraction or hierarchy. The hierarchy provided by SDL is very important to support iterative development (hierarchical composition of objects) of complex systems. Therefore, the structure of a hardware model can be mapped straightforwardly into an SDL structure.
However at run-time, intermediate structuring is irrelevant and the application is made of process instances only.

3.2.2 Communication

SDL provides asynchronous communication which, is an important concept of high-level system modeling. It allows the designer to abstract from low level details, which are not relevant at this moment of the development phase. The Message-passing model is attractive for system modeling because it represents the communication in an most abstract form.

3.2.3 Data

SDL is able to specify a system at an appropriate level of abstraction, without getting into implementation details. This also holds for data. SDL provides simple data for specification purposes, such as integers, reals, time, etc. For advanced data, Abstract Data Types (ADT) can be used. An ADT is a data type with no specified data structure. Instead, it specifies a set of values, a set of operations allowed on the data type and a set of equations that the operations must fulfil. This approach make it possible to map SDL data type to data types used in other high-level languages. ADTs have the possibility to specify data at different levels of abstraction. For example, an ADT can be used to solely specify the result of operations on data objects without specifying how this result is obtained. Specifying the properties of the operation can be performed later, using a set of equations known as axioms. Mainly in the first versions of SDL, all properties of operators of an ADT are specified using axioms. However, the drawback of axioms is:
- They are hard to understand.
- They are hard to be implemented.
- Their execution is slow.
- They are hard to use for building complex data structures.

Therefore, other possibilities are provided to define the operators of ADTs. For example, the SDL state machines can be used to implement a specific operator. This has limited usefulness if complex data structures have to be modeled. Therefore, SDL can be extended with C in order to model all necessary complex computation constructs. The interface of a data structure is given in SDL and the implementation of operations in C. Before, starting the simulation, C code will be linked to the SDL model.

Extending SDL with C should be used with care because it restricts the capabilities of test and formal verification tools.

At present, there are tools that join the use of the Object Modeling Technique (OMT) or the Unified Modeling Technique (UML) and SDL. Especially UML is becoming very popular as a graphical object oriented design language. SDL and UML complement each other. While UML is suitable for analyzing and expressing requirements, SDL has the formal action semantics for describing behavior, which UML lacks. This allows modeling many aspects of the system using an appropriate and standard formalism.

3.3 Abstraction Level

Abstraction is ignoring some aspects of phenomena, in order to clearly identify important qualities, capabilities and properties of these phenomena.

By building a model of a specific system, the designer is not interested in the physical features of a system. By means of formalisms, the system will be modeled without referring to any implementation details.

The aim of abstraction is to handle complexity. Due to the growing complexity of systems, it is necessary to abstract from some details that are not relevant at system level.

The choice of the abstraction level is crucial for managing the complexity of the system.

The higher the abstraction level, the manageable the complexity of the system to be modeled. Therefore, it is very important to choose an appropriate level of abstraction of the system to be modeled. This is necessary, in order to fully understand its purposes and functionality, before the implementation phase is started. This choice will be influenced by the aim of the model. Although the model should be as abstract as possible, it must also satisfy the purposes for which it will be developed.

SDL supports different levels of abstraction. Its EFSM concept is implementation independent. Only the behavior of a system is described. The communication between the EFSM of a system in SDL is performed by means of signals. Those signals and events have data parameters that can be defined at different level of abstraction. If e.g. an SDL process is sending cells to another SDL process, those cells can be sent by bit, by byte or by a complete cell. It is clear that defining the data at bit level will make the model more complex then doing so at cell level.

3.4 Modeling strategy

In an early phase of system development, the system should be modeled in SDL at a very high level of abstraction. The SDL model can be used as a reference during all system development disciplines. Later, this model can be refined and adapted to answer a
specific demand, e.g. for HdS verification software purposes. The development of a model in SDL should be performed in four steps:

- **Step 1**: a global SDL model of the system should be built. In this model, for example, only signature of ADTs is stated. The implementation of operations is postponed until a later phase. In addition, some tasks can be described informally. This step is performed in two phases: the architecture phase and the detailed phase. In the first phase, the block level of the system and the channels that connect those blocks are created. Further, the processes that make the system and the signal routes are defined. In the detailed phase, each process will be described using FSM. This model should be finished as soon as possible in order to be used as a specification document and as a starting point to build other models that will be used for specific purposes.

- **Step 2**: the SDL model will be detailed. The operators of ADTs can be implemented through SDL state machines. If this is not possible, because of the FSM constraints, C or OMT/UML should be used. If there are informal tasks, these will be described formally using the SDL primitives. In case of open models (models waiting for stimuli generated by the environment), the environment should be specified in order to perform exhaustive simulation. Processes, representing the environment, are introduced within the model to supply it with the required signal inputs. This model is complete and it can be verified and validated.

- **Step 3**: the SDL model will be further detailed to satisfy a specific demand. For example, the model will be refined in order to fulfill some specific requirements posed by the engineers. Such a model can be used e.g. as a test bench for the software verification or for performance analysis.

- **Step 4**: the external tasks are involved, in case of legacy code of other software that is not appropriate to be modeled with SDL. When the software is appropriately interfaced, the code of the SDL system can be generated. In this case, the Design Tracer should be used to trace the application at SDL level and to debug the software at code level (see chapter 6 for more information). This should be the last step of the modeling process because at this stage the simulator cannot be used. Therefore the SDL model must be already verified and validated at the end of step 3.

When a system is to be modeled, a model will be built to particularly satisfy one purpose, because it will be difficult to build a general purposes model. To satisfy different purposes, it is necessary to develop different models. Building all those models will require a lot of time. Therefore, it is better to do the job in a number of steps as stated above. A common model should be developed from which all other models can be derived.

### 3.5 Verification and validation

SDL provides powerful capabilities for verification and validation of a system behavior and automated code generation. This allows performing system validation at a higher level of abstraction and earlier in the development cycle.
Three kinds of simulations are possible:

- Interactive Simulation: using the interactive simulator, the user can play the role of environment by sending stimuli to the model and controlling the progress of the time. That is very important at the beginning of SDL modeling, when the environment is not yet modeled and some tasks are still informal. The user will then provide the SDL model with necessary stimuli for simulation.

- Random simulation: it helps to quickly detect errors. The simulator selects randomly a transition amongst all the possible ones. This is very useful when the system / parts of it are not yet finished. Quick error detection helps to early make modifications to the model.

- Exhaustive simulation: herein is a complete exploration of all the system states performed so that all encountered errors can be detected. Each error scenario is saved as a MSC. The simulator is performing the job automatically. It sends signals, increases time, selects transitions. Exhaustive simulation stops only when all system states are reached.

3.6 Use of message sequence charts

The message sequence chart formalism is complementary to SDL. While SDL is used to describe the behavior, MSCs can be used to describe the sequential ordering of interaction and time between SDL entities. MSCs mainly focus on message interchange by communicating entities and their environment. A main advantage of an MSC is its clear graphical layout, which immediately gives an intuitive understanding of the described system behavior.

A set of specified MSCs provides a partial description of the behavior. Therefore, mainly standard cases are candidates for MSCs. Basically, the information interchange is carried out by sending messages from one entity to another. In an SDL specification, those messages correspond to the signals that are sent from one process and consumed in another process. An entity can be any part of the SDL specification such as a system, block, process or process instance.

The message sequence charts can be used in various application areas:

- For requirement definition.
- As an overview specification of process communication.
- For simulation and consistency check of an SDL specification.
- As a basics for selection and specification of test cases.
- For documentation.

Figure 4 describes a message exchange between two SDL entities. MSCs do not have just an illustrative character in the system development. They can be used as test proposes for the automated test generation. MSCs are very useful for verification and validation of the system as whole or as entities consisting of blocks or processes.
The basic language of MSCs includes all constructs, which are necessary to specify the pure message flow. These constructs are entities (system, block, process or process instance), messages, environment, actions, timer set and reset, time-out, instance creation, instance stop, and condition.

The structural language elements of MSCs contain other constructs to specify more general MSCs or to refine MSCs.

![Figure 4: Example of a Message Sequence Chart](image)

### 3.7 Conclusion

In this Chapter, the SDL language is presented as a modeling language for hardware systems. The SDL formalism provides features that are very important for hardware modeling. On the abstract side, it allows to describe a system informally. This informal description can be refined on relatively abstract level dependent on the purposes of the SDL model.
4 MTM functional description

4.1 introduction

The Alcatel 1000 System 12 exchange consists of the broadband Group Switching Network (GSN) and the existing S12 narrow band Terminal Sub-Units (TSUs). The TSUs are connected to the GSN via dedicated internetworking elements. The GSN switching system has a modular structure. The system is composed of a number of boards that together implement the exchange functionality. The main modules are the Switching Modules (SM) which constitute the Multi Path Self-Routing switching matrix. Other modules provide the interface of the external accesses to the MPSR switch. One of these modules is the Termination Link board (TLK) which connect STM (synchronous transport module) traffic to the MPSR switch. Figure 3 shows a connection of the TLK to the MPSR switch.

![Figure 5: the environment of MTM ASIC](image)

A variant of applique provides both electrical and optical termination (e.g. Multi-Mode Fiber termination).

One of the modules the TLK board is composed of, is the MTM ASIC. The aim of developing this ASIC is to add a flow control mechanism to the switch interface. The MTM ASIC, the TLK board and the MPSR switch will be explained in the next paragraphs.

4.2 MPSR switch matrix

4.2.1 Switching principles

The Alcatel GSN switching system is based on the MPSR concept. Therefore, it is also referred as the MPSR switching network. This network is built up by multiple stages of switching modules (multi-stage). As the name Multi-Path mentioned, there is more than one path between a given input port and an output port. The MPSR switch of Alcatel is a folded network. It uses bi-directional links and all input and output ports are located at the same side of the switching network. When the cells travel towards the reflection point (it is always situated in the last stage of the switch),
they will be routed to any destination (randomization). If the cells return from the reflection punt, it is routed to the correct destination.

4.2.2 Multi slot cell

The MPSR switching network operates on an internal format called MSC (Multi Slot Cell). A MSC consists of an unlimited number of slots. Each slot has a fixed length of 68 bits. The first slot of the MSC is an overhead slot, the Self-Routing Tag (SRT). It contains all information needed to route the MSC through the network. The following slots are the body slots and contain payload data. Every 17th slot is used as a synchronization slot. A MSC is transported through the switching network as a non-interruptible sequence of slots. Each slot consists of 64 bits Slot Body and 4 bits Slot Control (SC). The SC is used to indicate the type of slot.

There are three types of slots:

- The first slot of a MSC is used for routing purposes.
- There are slots carrying user information. Those slots are transported transparently through the network.
- Idle slots are transferred through the network to fill the available bandwidth.

The MPSR uses an MSC that consists of eight slots referred as MSC-8 (see figure 5).

![Figure 6: MSC-8 format](image)

4.3 Termination link board

The termination link board is developed for the Alcatel 1000 S12 broadband exchange. The TLK allows to connect eight STM-1/OC-3 or two STM-4c/OC-12c links (see table 1) to the MPSR switch. The links contain virtual connections (VP) filled with ATM cells destined for the broadband exchange. The ATM cells are policed according to the agreed ATM traffic capability. The ATM cells are then put in MSC-8 and switched through the MPSR network to another broadband board (e.g. TLK).
### Table 1: SONET/SDH hierarchy

<table>
<thead>
<tr>
<th>Optical Carrier level</th>
<th>SDH level (ITU)</th>
<th>SONET level (ANSI)</th>
<th>Data rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC-1</td>
<td>STS-1</td>
<td>STS-1</td>
<td>51.48Mbps</td>
</tr>
<tr>
<td>OC-3</td>
<td>STM-1</td>
<td>STS-3</td>
<td>155.52Mbps</td>
</tr>
<tr>
<td>OC-12</td>
<td>STM-4</td>
<td>STS-12</td>
<td>622.08Mbps</td>
</tr>
<tr>
<td>OC-24</td>
<td>STM-8</td>
<td>STS-24</td>
<td>1.244Gbps</td>
</tr>
<tr>
<td>OC-48</td>
<td>STM-16</td>
<td>STS-48</td>
<td>2.488Gbps</td>
</tr>
<tr>
<td>OC-N</td>
<td>STM-N/3</td>
<td>STS-N</td>
<td>N*51.84Mbps</td>
</tr>
</tbody>
</table>

OC: optical carrier  
SONET: synchronous optical network  
SDH: synchronous digital hierarchy  
STM: synchronous transport module  
STS: synchronous transport signal

The TLK is able to process both the SDH (ITU-T standard) and SONET (ANSI standard) streams.

![Figure 7: functional block diagram TLK](image)

Figure 4 shows a functional block diagram of the TLK. The board consists of a number of elements such as ASICs, memories, etc.
The main components of TLK are explained below.

**MSE ASIC**
The Multiplexing and Switching Element ASIC provides the switching and multiplexing/de-multiplexing functions between four links of 155.52 Mbps and one link of 622.08 Mbps. The switching part of this ASIC can be considered as the first/last stage routing of cells towards/from the MPSR switch.

**MTM**
The MTM ASIC performs the ATM-to-MSC and MSC-to-ATM cell conversion. It also re-sequences the MSCs leaving the MPSR switch. Further functions are AAL-5 (ATM Adaptation Layer) and scheduling. For some of these functions, MTM is assisted by a DSP (Digital Signal Processing) (see section 3.4 for more information about the MTM ASIC).

**PHT ASIC**
These two ASICs provide Policing, Header and Translation functions. They also perform OAM (Operation Administration and Maintenance) functions. Further, all ATM specific functions are carried out by these two ASICs.

**S/UNI Devices**
Two SDH/SONET user network interfaces perform overhead termination and extraction/insertion of ATM cells from/to the synchronous payload envelope, using ATM cell delineation.

**OBC**
The On-Board Controller is the manager of the board. It is used to program the devices of the board. The OBC receives also information about board alarms and errors and performs the defined actions on them. It consists of a Pentium-compatible processor, PCI bus interface, RAM controller, RAM and ROM.

**MTR-C**
The Maintenance and Test Routiner C provide the OBC access to the ASICs on the TLK.

### 4.4 MTM ASIC

The MTM ASIC will be designed to add the flow control mechanism to the MPSR switch interface. Through its flow control mechanism, the ASIC is able to protect the internal switch resources at high load on external links. The MTM ASIC consists of two main parts: the Egress pipe and Ingress pipe. MTM also has common modules shared by both datapaths: IDRFC (Internal Dynamic Rate based Flow Control), AAL-5 (ATM Adaptation Layer level 5), OBC control and interfaces. For test and verification purposes a Traffic Generator, Traffic Analyzer and LoopBack units are also implemented within the MTM ASIC. The MTM ASIC is able to handle up to 64k connections. Each connection is identified uniquely by its ERN (External Reference Number). Different kinds of connections are supported such as Real Time, Non Real Time Guaranteed and Non Real Time Non Guaranteed.
In Figure 5 a functionally block representation of the MTM ASIC is given. In this figure, the Traffic Generator, the Traffic Analyzer and the LoopBack units are not represented. Those elements are only connected to the MTM in test mode.

**Figure 8: MTM Core Outline, without LoopBack units, TG and TA**

### 4.4.1 The Traffic generator and Traffic Analyzer

For test and verification purposes, a Traffic Generator and a Traffic Analyzer are implemented in the MTM ASIC. By means of the two LoopBack units, ATM cells leaving the Egress can be send back to the Ingress whereas MSCs coming from the Ingress will be put at the input of the Egress. Figure 6 gives a presentation of Egress and Ingress when connected to the TG and TA via the LoopBack units. The TG generates different kinds of connections for RT and NRT traffic. Both the cell mode and packet (fixed packet length) mode is supported. At most, eight connections can be generated.

Each connection can be generated using different kinds of characteristics such as parity, priority, etc. Each connection is assigned a specific bandwidth. The Traffic Generator is able to produce both ATM and MSC traffic. All generated connections must have a constant bit rate.
The TA is able to maximally analyze the traffic of eight connections. These connections can be any of the 64k connections processed by the MTM. Each connection can be identified by a filter. Both MSC and ATM traffic can be analyzed. For each connection under analysis, different measurements will be performed such as minimal/maximal inter-cell spacing, number of received cells, average bandwidth calculation, etc.

Figure 9: connection of LoopBack units, TG and TA to Egress and Ingress

4.4.2 Ingress

The Ingress refers to cells being transferred from the physical interface to the MPSR switch. The Ingress pipe of the MTM ASIC performs the following functions:

- Cell reception of complete ATM cells.
- Queue acceptance and traffic monitoring.
- Frame gathering and packet merging.
- Scheduling. It determines which flow will be served next.
- MSC assembly and transmission. ATM cells are transformed to MSCs and transmitted to MSE.

4.4.3 Egress

This is the part of the MTM ASIC that is modeled in SDL. The Egress refers to the MSCs being transferred from the MPSR switch to the physical interface.
The Egress is composed of the following modules:

**MSC Reception**
This module receives MSCs from MSE via 16/14 parallel elementary links operated at 155 Mbps. It then performs the (bit) serial to (slot) parallel conversion for each incoming flow. Idle slots will be discarded and Non idle slots are collected into 16/14 buffers. A time division multiplexer will scan the buffers and when a complete MSC is available, it is sent to the MMM (MPSR Memory Manager). IDRFC and AAL-5 messages are transmitted respectively to the IDRFC controller and the AAL-5 receiver. Before sending an MSC to the MMM, some information must be retrieved from its header (slot one and two), because they are needed for further processing in the Egress. This information and the cell address, obtained from the MMM, will be sent to the Routing module. In MSC Reception module, also erroneous data are discarded and various error rates will be measured and monitored. In test mode, a parallel input is provided and will be used as an interface with the Traffic Generator. From this interface, the MSCs are coming by slots. In addition, a mode without MMM is foreseen for test purposes.

**Routing, Re-sequencing and Multi-cast**
A cell (header information and cell address) received from the MSC Reception module will be routed to one of the 16 re-sequencing units. Those units will put the incoming cells back in the correct order. Cells with excessive switching delay must be discarded. Re-sequencing is needed because of the randomly distribution of cells toward the reflection point in the MPSR switch. Therefore, it is possible that two cells belonging to the same connection arrive in reverse order. To enable the re-sequencing of cells, each cell leaving the Ingress toward the MPSR switch must get a Time STamP (TSTP). This TSTP value will be used to equalize the delays of the incoming cells in such a way that when the cells leave the module they will have the same order as that indicated by the TSTP. Two kinds of traffics are treated here: Real Time (RT) and Non Real Time (NRT). The RT cells are transmitted to the Queue Manager and the NRT cells are sent to the Queue
Acceptance block. Multicast of RT cells is performed before re-sequencing while multicast of NRT cells is done afterwards.

Each time a cell must be routed, the context table (or routing table) is read. This table gives the number of the re-sequencing unit destined by the cell. In case of multicast, the table contains a pointer to an entry of the table where the destination of copy must be retrieved. Maximal 8 copies are allowed for the same cell.

Some restrictions are imposed to the routing and multicast actions, e.g. the NRT cell must not be routed to re-sequencing units reserved for RT traffic and vice versa.

When the level of RT queues in the Queue Manager grows until it reaches the threshold, a Backpressure signal will be generated and transmitted to the Re-sequencing unit. This will stop the transmitting of cells to the Queue Manager until the Backpressure signal is deactivated.

In test mode, the Re-sequencing unit is not used because the Egress consumes cells in the same order at which they are produced by the TG. The cells routed are transmitted immediately to the output. Furthermore, multicast will not be performed for those cell.

**Queue acceptance**

This module is only dealing with NRT cells. When a cell is received from the Routing module it will be checked whether it can be accepted in the buffer or not. Accepted cells are transmitted to the Queue Manager. Cells that are not accepted must be discarded.

To determine whether a cell will be accepted or dropped, a context table is consulted. The context table contains, for each connection the disciplines applied to that connection, the low and high threshold of its queue. This table is set up by the OBC for each new connection established by the system.

There are two types of Loss disciplines applied to cells entering the Queue Acceptance block: Cell Loss discipline and Frame Loss Discipline. The first discipline is dealing with individual cells and the second is applied to packets.

For each queue, another table is used that maintains the queue-level (number of cells in a queue), the discarded-level (number of cells discarded) and the discipline applied to each connection.

**Queue Manager Frame merging**

This block manages both the RT and NRT queues where cells wait to be scheduled.

RT cells entering this module are stored in their corresponding real time queues. NRT individual cells (which do not belong to a packet) are also put in its queues.

For a connection with a frame merging (connection with packets), the cells will be added to a Frame Aware Buffer (FAB). A FAB is special queue utilized to store frames (packets) that belong to a connection. There are in total 2K FABs, each 32 FABs represent a Frame Merging Session (FMS). The scheduling of those queues is done in two phases. First, a FMS is chosen, next a FAB from 32 FABs of this FMS is selected.

When the Queue Manager receives a trigger from the Scheduler, it sends the appropriate cell to the next block. The Scheduler selects one VP among seven or one ATM path among eight (or two). For RT traffic, each VP or ATM path is corresponding with one of fifteen (or nine) queues foreseen for RT traffic. RT traffic has priority so that for each ATM/VP generated by the Scheduler its corresponding queue is firstly checked. Only if this queue is empty, the NRT traffic is next.
For NRT traffic, the selection of a next cell to be scheduled is a bit complicated because of the number of connections (64K). Therefore, a discrete rate approach is applied. All connections with the same services are grouped together in a rate FIFO queue. For each ATM/VP, a number of rate groups (56 for NRT Guaranteed traffic and 8 for NRT Non Guaranteed) are defined which cover a bandwidth from 32 Kbps to 622 Mbps. A queue number of individual cells will be added to its corresponding rate group when the queue contains at least one cell. A FMS is appending to the rate group queue when this FMS comprises at least one FAB with a complete packet. If a NRT cell is scheduled, its queue number is sent back to the Queue Acceptance to update its corresponding queue level. For RT cells there is a possibility to send a Backpressure signal to the Routing-Multicast block in order to stop routing of cells to this queue when its threshold is reached.

Scheduler
The scheduler performs the traffic shaping of a particular virtual path connection (VPC). Traffic shaping in an ATM switch is the mechanism by which the stream of cells into a VPC or VCC (Virtual Channel Connection) is altered to achieve the desired traffic characteristics. Each time slot, the Scheduler selects (Time Division Multiplexing) one ATM path from the sequence ATM1 to ATM8 (or ATM1 to ATM2). If this ATM path has one or more VPs, the one with the smallest TSTP will be chosen. Then the Scheduler will generate a trigger that contains the number of VP/ATM to be served. When a VP is served its TSTP will be updated. The Scheduler can receive a Backpressure signal from the ATM termination device to indicate that generation of triggers for that ATM path will be skipped.

MSC assembly and ATM transmission
In the previous block, only some header information and the cell address are processed. The complete cells (MSC) are stored in a Large Output Buffer that is administrated by the MMM. When the cell information has been received from the Queue Manager, the corresponding MSCs will be retrieved from the MMM. For each cell, the SRT will be modified. The ERN in the SRT will be substituted by the Internal Reference Number (IRN). For each ERN, the corresponding IRN can be found in the ERN-to-IRN translation table. From the same table, the Line Termination (LT; also known as ATM path) number will be fetched and entered in the corresponding field of the SRT slot. Each MSC will then be converted to an ATM cell and put into a FIFO. For each LT number (eight or two), a FIFO is available.

There are two interfaces available to transmit the ATM cells: the All (ATM Internal Interface) and the UTOPIA interface. Each one is divided into sub interfaces; the first one processes the LT (FIFO) numbers 0, 2, 4 and 6 and the second one the numbers 1, 3, 5 and 7. For test purposes, a mode without MMM is foreseen to make tests possible even when the MMM is not accessible due the failure. In this mode the MMM interface is not used.
4.4.4 Common part

The MTM ASIC also has a common part consisting of the following elements:

- IDRFC: this flow control mechanism is introduced between termination boards to protect internal switch resources and still reach a good utilization of the switch capacity and high load on external links. IDRFC is designed for dynamically assigning the allocated cell rates at which the input buffer scheduler can serve the various input buffer queues.

- AAL-5: ATM Adaptation Layer protocol 5 handles control messages that are extracted from the Egress and dropped to the OBC or received from the OBC and inserted in the Ingress stream.

4.5 Conclusion

In this Chapter, the MTM ASIC is presented. The parts of the MTM and its environment are also explained.

The choice to model the Egress is because this is independent of the IDRFC algorithm. Data-oriented models are not very suitable to be modeled in SDL.
5 Egress modeling

5.1 Introduction

In this chapter, the Egress pipe, the traffic generator and the traffic analyzer are modeled in SDL. The modeling process is performed in three phases. First, a model processing only real-time traffic is designed. Second, the non real time traffic is added to the model. Finally, the system is expanded with a packet mode.

In each of these phases where a specific model is developed, the SDL modeling is performed in three steps. In the first step, the architecture structure is created. Herein the system is described by means of blocks and processes interconnected by channels and signal routes. In the second step, the state machines of each process are written. When the module is successfully verified, some detail description has been added.

Because the Hds verification SW is very close to HW, a detailed description of the behavior is needed. It is important to make a trade-off between the advantages of modeling at a high level of abstraction and the SW requirement to give some HW specific implementation which maybe irrelevant for the system-behavior. This demands extra effort during the development process.

Another problem of the modeling process is caused by the initial specification, which is very detailed. Therefore, it was very difficult to distinguish the elements to be modeled, from those that could be abstracted from.

5.2 Abstraction level

As stated above the initial specification of the system is very detailed. Actually, it is made to be used for VHDL implementation. At system level, the Egress can be considered as a black box, which receives MSCs from the TG and sends ATM cells to the TA. The abstraction level of the flow processed by the Egress will be decisive in the resulting complexity of this system. Therefore, the abstraction level of the flow of cells sent from the TG to the Egress pipe must be defined. Three possibilities, which will be considered, are explained below:

- Each SDL signal carries one bit as described in the specification. This method demands a high degree of effort and is time consuming. Describing cells at bit level will undermine the advantages of SDL to model at systems level. In this case modeling at SDL is much alike an implementation in VHDL. Replacing implementation details by similar abstraction detail will not yield any profit.

- An SDL signal will carry one slot of a MSC. An MSC-8 consists of eight slots. Therefore, it is possible to consider the traffic flow as a set of slots. Since the Egress just uses a few parameters from the header and the complete cell is stored in the MMM, this approach has only effect on the MSC Reception module.

- The whole MSC cell will be carried by a SDL signal: this approach corresponds with the decision not to model the MMM (see below). In this case, the Egress will process a complete cell instead of a few parameters from the header.
Within the Egress pipe, due to efficiency reasons, the system is just processing a few parameters from the header of a MSC cell. The complete cell which consists of the header and the payload will be stored in the MMM (MPSR Memory Manager) when it enters the Egress. Before leaving the Egress the cell will be retrieved from the MMM. The cells belonging to the traffic processed in the SDL model of the Egress pipe have an empty payload (no information). Therefore it is not necessary to model the MMM. The model will process the complete cell, which will give the advantage of a high level of abstraction. The choice not to model the MMM is supported by the fact that the MTM ASIC can be used in two modes: normal mode and bypass mode. The last one is used for test purposes. In bypass mode, the Traffic Generator and Traffic Analyzer are connected to the MTM ASIC. Further, the MMM is disconnected so that the ASIC will process the whole cell. In bypass mode the Re-sequencing block is also disabled. After all, this block just restores the order of cells leaving the switch. Although the choice is made to model the traffic using a complete MSC, still there is a possibility to use other forms of modeling, especially between the TG and the MSC reception. The choice for bit, slot or cell has only effect for the Traffic Generator and the MSC Reception. If there is a need to model at a low level, these modules must be detailed. Another possibility is to add a process to the TG, which will divide the generated MSC into slots (or bits). Another process in MSC Reception will receive these slots (bits) and transform them into a MSC. This shows an important concept of SDL, which is modeling at a different level of abstraction.

5.3 Architecture design

The architecture of the SDL model is almost the same as the architecture of the Egress in test mode. In that mode, the Egress is connected to the Traffic Generator and the Traffic analyzer. The OBC is modeled here to supply the Egress, the Traffic Generator and the Traffic Analyzer with the configuration parameters, which is actually the task of the HdS verification software. At system level, four blocks are modeled, representing:

- The Traffic Generator.
- The Egress.
- The Traffic Analyzer.
- The OBC.

At system level, the Egress is considered as a black box that receives MSC from the traffic Generator, produces ATM cells and transmits them to the traffic Analyzer. (in figure 8, M_S_C signal is used because MSC is a reserved word in SDL).

These three blocks, which compose the total system, are split up in a number of sub-blocks or processes.
5.3.1 The Egress pipe

The Egress is split into five blocks. Except to some small details, the SDL architecture is the same as the hardware architecture (see figure 7 and figure 9). The Scheduler and the Queue Manager are integrated in one block, because the Scheduler is composed of one process and is only concerned with the traffic inside the Queue Manager.
The Egress is composed of five blocks:

- MSC Reception: this module receives MSCs from the TG through the channel TGtoMR and transmits them to the next block.

- Routing-Multicast: cells transmitted from the MSC Reception are routed to the Queue Manager (RT cells) and to the Queue Acceptance module (NRT cell). The Routing-Multicast can also receive signals (Backpressure) from the Queue Manager, which indicate that a threshold of a specific queue is reached.

- Queue Acceptance: NRT cells transmitted from the Routing-Multicast module will be sent to the Queue Manager, if they are accepted.

- The Queue Manager: it receives RT cells from the Routing-Multicast and NRT cells from the Queue Acceptance. The two traffics are scheduled, and sent to the MSC_ATM_Translation module.

- MSC_ATM_Translation: this module receives MSCs from the Queue Manager and sends ATM cells to the output.

### 5.3.2 The Traffic Generator

The traffic generator is split up into two processes: the ConnectionPr process and the TG process (see figure 10). The ConnectionPr represents a manager of the TG. It receives a list of parameters from the OBC and it decides to dynamically (dashed line) start a new connection or to end an already started connection. Maximally eight instances of TG can be started. Each instance of TG represents a connection.

When the ConnectionPr process starts a connection, it provides it the configuration parameters needed to start generating traffic of MSCs. The generated MSCs are transmitted via the TGtoEG channel toward the Egress.

![Figure 13: SDL block diagram of the Traffic Generator](image-url)
5.3.3 The Traffic Analyzer

The traffic Analyzer consists of three processes: The PrAdministration, the All_Interface and the TA (see figure 11). The PrAdministration receives parameters from the OBC and will start or stop a given connection. Up to eight process instances of TA can be started. Each process instance corresponds to a Traffic Analyzer for one connection. When a process is started, it will be provided with the configuration parameters needed to analyze a given connection. Each process instance of the TA started will send a signal to All_Interface process. With this signal, the All_Interface will find the Process identification (Pid) of this instance, which can be used to address this process.

The PrAdministration is also communicating with the All_Interface in order to select one of the two possible All interfaces.

![Block diagram of the Traffic Analyzer](image)

**Figure 14: block diagram of the Traffic Analyzer**

5.3.4 The OBC

The OBC is used here to provide the SDL model with the necessary configuration parameters needed to process its traffic. This task is actually performed by the HdS verification software. Because the software can not be interfaced at this stage, the OBC is modeled here.

Three processes are modeled representing the OBC:

- **TGTester**: provides the Traffic Generator with some configuration parameters needed when the processes of the TG are started.
- **TATester**: supplies the Traffic Analyzer with a list of parameters via the channel OBC_TA.
- **EgressTester**: transfers four lists of parameters via the channel OBC_TG to the Egress. Each list is destined for a specific block of Egress.
5.4 Detailed design

In sections 4.2 and 4.3, the architecture of the system and the abstraction level of the traffic have been defined. In this section, the processes of each block will be described. In the detailed design, the behavior of the system will be described by representing each process by means of a state machine.

Describing the behavior is performed in a number of stages. First, the most important functions of the system are modeled. Complex behavior is described as informal tasks and the operations of ADTs are not yet implemented. After system verification, the informal parts will be described and ADTs will be implemented.

So far, the system (Egress, TG and TA) described is an open model that waits for stimuli from the environment. The model has been transformed into a closed model. This is very important when the exhaustive simulation will be done (see paragraph 4.5). Therefore, the OBC is modeled to represent the environment, which will supply the system with the necessary stimuli.

In this section, a global functionality of the system modeled in SDL will be given.

5.4.1 The Egress

MSC Reception

Because of the choice made to model the traffic between the TG and Egress with signals transporting a complete MSC, the behavior of the MSC Reception is straightforward. It receives MSC at its input and transfers them to the next module. The number of cells entering this block will be counted.
Routing-Multicast

This block consists of one process, performing the routing and multicast of both real-time and non real-time cells. This process checks its queue for the presence of cells. If there is any, it will be fetched from the queue and routed to the appropriate output. RT cells will be transmitted to the RT Queue Manager process situated in the Queue Manager block. NRT cells/frames are transmitted to the Queue Acceptance block. Because not all the fifteen real-time links will be modeled, all cells will be sent over one channel. Each RT cell is then transmitted with its link number. This number will be used by the RT Queue Manager to store a cell in its appropriate queue.

In SDL, each process has only one input queue, therefore it is not possible to model all the fifteen links between the Routing-Multicast and the Queue Manager. Further, this will not be practical if we have more links; drawing all those links is time-consuming. Another possibility to model the fifteen links is to make use of the process instances set. Fifteen instances of the RT Queue Manager process should be created when the SDL system is initiated or dynamically (at run-time). Each process instance has an address (Pid: Process Identifier) which will be used by the Routing-Multicast process to transfer a cell to the appropriate queue (process instance). This approach seems acceptable at the first instance. However, when the queues must be dimensioned, it is difficult to manage them. Therefore, a data structure implementing the queues (circular FIFO) is introduced in the RT Queue Manager. There are fifteen queues, each one corresponds to a link number.

For NRT cells, there are no difficulties because only one link is provided to send the cells to the Queue Acceptance Block.

In case of multicast, for both RT and NRT maximal eight copies of a cell will be generated. The ERN of each copy will be replaced by the ERN fetched from the routing table. Other restrictions are for example that the copies of RT cells should not be routed to the NRT link and vice versa.
Queue Acceptance

The Queue Acceptance block consists of processes: the QA process and the RandomGen process. The RandomGen process is pseudo-random bit sequence generator, which generate a random number. This number can be used by the Frame Loss Discipline to determine whether a cell must be accepted or dropped.

Figure 17: SDL process diagram of Routing-Multicast

Figure 18: Process diagrams of the Queue Acceptance Block
The Queue Acceptance block is only dealing with NRT cells. When the QA process receives a cell from the Routing-multicast block, it determines if a cell will be accepted or not into the queue of the NRT Queue Manager process. Cell acceptance or deletion is performed according to Cell/Frame Loss Disciplines. If a cell is accepted it will be transmitted to the next block, else it will be discarded.

After the process is started, it waits for configuration information from the OBC. This information is used to set up a context for each connection. This context defines the parameters needed to manage the connection such as the Loss Discipline, the low and the high threshold of the queues.

Queue Manager

This module consists of three processes: the Scheduler, the RT Queue Manager and the NRT Queue Manager (see figure 15).

![SDL representation of the Queue Manager](image)
The Scheduler selects (TDM) one ATM path from eight or two (programmable). Each ATM path can have zero or up to seven VPs. If an ATM path has more than one VP, the one with the smallest TSTP will be selected. If a VP is selected its TSTP will be updated. After that, the Scheduler generates a trigger with the ATM/VP number and transmits it to the RT Queue Manager.

The RT Queue Manager checks a queue corresponding with this ATM/VP number. If there is any cell, it will be transmitted to the next block. In case the queue is empty, the trigger is forwarded to the NRT Queue Manager.

The NRT Queue Manager processes both the NRT cells and the NRT packets. For each connection, a queue (circular FIFO) is provided, named an ERN queue. The Egress must be able to process 64 k connections. However, the TG can generate only eight connections. Thus, it is not necessary to model all those ERN queues. Actually, the Queue Manager is working with a sorting technique, which is independent of the number of connections. All connections with the same service rate are grouped together in a rate group FIFO queue. Each VP/ATM path has 64 rate group queues. Empty ERN queues are not added to the rate group queues.

For packets/frames, the situation is different, a Frame Aware Buffer (a special name for an ERN queue with packets) must first contain a complete packet. Then, its number will be added to its Frame Merge Session array (there are 32 FABs for each FMS). After that, it will be checked if this FMS is part of a rate group queue or not. If a FMS is not active - this means it is not part of a rate group queue - it will be activated by adding its number to this rate group queue.

When a VP/ATM signal is received from the RT Queue manager, the ERN queue or FMS number will be extracted from the head of the rate group queue which belongs to that VP/ATM number. If this number corresponds to an ERN queue number, a cell from this queue will be sent to the next block. The number of this ERN queue will be added to the tail of its rate group queue except when the cell sent is the last one in the queue.

When the extracted number is an active FMS, the FAB of this FMS with the smallest TSTP will be served first. When a FAB is served, it will be served for the whole packet. As long as the End of Packet (EoP) of this FAB is not sent, each time its FMS should be served, this FAB must be served first. When an FMS is served its number will be add to its rate group queue only if it has at least an active FABs.

**MSC Assembly**

This block consists of five processes: MSCtoATM, M_A, ATMtoAll_UTOPIA and two processes coordinating the traffic from ATMtoAll_UTOPIA to the next blocks.

After receiving the configuration parameters from the OBC, The MSCtoATM waits for cells to enter its queue. If there is any, the transition is triggered and a MSC will be translated to an ATM cell. The ERN in the SRT of a MSC will be substituted with the IRN stored in the translation table. Other parameters will also be modified and checked like the VP/VC field. The ATM cell will be then transmitted with the LT (= ATM path) number (fetched from the translation table) to the M_A process.

The M_A process will start two or eight (programmable) instances of ATMtoAll_UTOPIA. Each process corresponds to an LT number. The Select Interface register selects the interface to be used: UTOPIA or All. The M_A process transmits each ATM cell to the appropriate ATMtoAll-UTOPIA process instance, which corresponds
with an ATM path (physical links). The process FIFO_1 will multiplex the four odd ATM path numbers to one output link, whereas the process FIFO_0 will multiplex the other even four path.

Figure 20: SDL representation of MSC_ATM_Translation module

5.4.2 The Traffic Generator

The TG receives configuration parameters from the OBC. Up to eight connections can be generated. Both cell and packet mode is supported. For each generated connection, there is a possibility to insert OAM cells to a stream of user cells, e.g. one OAM cell per 10 user cells. Other parameters are also programmable, which give the proportion of cells with and without priority, the proportion.

In packet mode, a connection generates packets with a fixed length. The rate at which a connection is producing cells is programmable by means of the inter-cell space parameter (the time between two consecutive generated cells). This rate must be constant.
All the parameters are programmable from the OBC.

### 5.4.3 The Traffic Analyzer

The TA can analyze up to eight connections from the 64k connection, which can be managed by the MTM. To distinguish cells that belong to an analyzed connection, masks are used. In principle, all cells entering the AII_Interface process are transferred to the active process instances of the TA. A connection accepts only cells to belong to it and the rest will be filtered.

The connection under analysis can be a connection processing individual cells or packets. For each connection, different parameters are monitored such as the number of received cells, the inter-space cell, etc.

### 5.4.4 The OBC

The OBC supplies the TG, the TA and the Egress with the necessary parameters needed to start their processes.

It gives the green light to the TG to start or to stop a given connection. For each started connection, the OBC can specify a number of parameters such as priority, the proportion of OAM and user cells, cell or packet mode, etc.

For the Egress, the OBC can specify a number of links and the interfaces to be used. It also transfers information tables such as the routing table, the context table, etc.

### 5.5 Rapid prototyping

Once a part of the system is developed, it is checked against the SDL semantics rules. If this check is successful, the simulation can be started even if some processes are missing and some informal SDL statements are present in the SDL model. The simulator can automatically detect design errors like deadlocks, data access violation, output signals with multiple receivers, etc.

The Interactive and Random simulation will be repeated each time that a part of the system is modeled. This helps to detect errors in the early stage of system modeling. Only if these simulations are properly performed, the modeling of other parts of the system will be started.

Once an SDL system or subsystem is completed, the exhaustive simulation will be performed, which explores all the system states if the model is small. For each erroneous scenario, a message sequence chart is stored in a file.

### 5.6 Use of Message Sequence Charts

MSCs are usually used to analyze the behavior at the requirements stage. When the SDL modeling is started, the requirement phase is already finished. The signals exchanged between the entities of the system have been defined.

Therefore, only the automatically generated MSCs are used. When a system or a part of it is simulated, there is a possibility to automatically generate a scenario of MSCs. Each entity of SDL can be monitored either individually or with other entities. For each part that the MSCs will be generated for, it is possible to choose only the signals to be represented. Signals that are not desirable can be disabled. This possibility gives the designer an opportunity to easily analyze the generated MSCs.

The generated MSCs will be examined and analyzed. From this analysis, it is possible to check if the system is doing what it should do.
5.7 Simulation results

In this section some simulation results will be given. This results, gives an idea about the simulation time required to process a fixed number of cells by the SDL model when some parameters are changed. The parameters that influence the simulation time are the number of connection and the STI (Scheduler Time Interval). The STI can be found in the Traffic Generator, the Scheduler and the Output (MSC_ATM_Translation) blocks.

First, the STI value of the three blocks is kept equal to 1. The obtained results are summarized in table 2.

From these results, it is clear that the Egress is not capable to process all the cells generated by the TG. Half amount of the traffic did not reach the output. Cells, which did not leave the Egress, are either deleted or buffered in the Queue Manager.

<table>
<thead>
<tr>
<th>Number of input cells</th>
<th>Steps</th>
<th>SDL time</th>
<th>Simulation time</th>
<th>Number of output cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1042</td>
<td>50</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>100</td>
<td>2290</td>
<td>100</td>
<td>23</td>
<td>50</td>
</tr>
</tbody>
</table>

**Table 2: Simulation results with maximum input load**

In table 3, the STI of the TG is increased (STI=2). With this new value, the Egress is able to process all the traffic produced by the TG.

<table>
<thead>
<tr>
<th>Number of input cells</th>
<th>Steps</th>
<th>SDL time</th>
<th>Simulation time</th>
<th>Number of output cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1728</td>
<td>100</td>
<td>18</td>
<td>50</td>
</tr>
<tr>
<td>100</td>
<td>3421</td>
<td>200</td>
<td>36</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table 3: Simulation results with maximum in-output traffic**

To make it possible to process all the traffic generated by the TG in table 2, small values of STI are needed in the Scheduler and the Output. However, the smallest value that can be attributed to the STI parameter is 1.

There is a possibility to speed up the Egress model by making the Schedule and the Output synchronous. In this case, the Scheduler and the Output did not have to wait an STI time to process a cell. When a cell is entering The Queue Manager, they must be triggered immediately.

**One/Eight connection (s)**

For one connection, the simulation time increases considerably when the STI of the TG is incremented (the STI of the Scheduler and the output are kept constant). This applies also for the number of steps required to process a fixed number of cells (see figures 4 and 5).
If the number of the connections is increased to eight, the simulation time is decreased a factor 2.5 if the STI is equal to 5 and a factor 5 by an STI equal to 15. The simulation time can also be reduced a factor 2.5 a 4 if the traces are set off. In this case, automatic generation of MSCs is no more possible. The STI and connection numbers are very important parameters for the traffic processed by the SDL model. By combining a small value of STI with a large number of connections, the amount of the traffic can be considerably increased.

The number of simulation steps (= SDL process transition) needed to process a fixed number of cells increases when the STI is augmented. This can be explained with the internal activities of some processes, which are not dependent of the input traffic. For example, the Scheduler generates a trigger each time slot (= STI) independent of the traffic processed by the Egress. In the MSC_Assembly block, there are also autonomous processes, which implement the AII interface. There are eight FIFOs, each corresponds with an ATM path. Each four links (one link, 155 Mbps) are multiplexed to one link (622 Mbps). The FIFOs are scanned (TDM) independent of the input traffic.
5.8 Conclusion

In this Chapter, the phases have been gone through to model the Egress are explained. The Egress is modeled in a top-down manner in different stages. At each stage, the necessary complexity is added to the system. First, a model with RT traffic is developed. Next, the NRT traffic functionality is added. After that, the model is expanded to able to process packets.
6 Interfacing SW with the HW model

6.1 Introduction

After the development of a hardware model, the possibilities to interface the verification software with it have been studied. This integration will allow to perform H/SW co-verification of the Egress.

Currently HW/SW verification is performed through enabling simulation between HW (VHDL) and SW (C code) components of the design. A co-verification environment is used for this purpose. The mean problem encountered when performing HW/SW co-verification is the speed. The hardware simulator represents a speed bottleneck. Further, the level of details offered by VHDL code is much higher than required by HdS. Therefore, fast HW models are required to satisfy the requirements of HdS.

When the SDL model is completed and by means of simulation verified and validated, it is ready to be interfaced with the software.

From a purist point of view, it would be best to describe all parts of the system, including the HdS verification software using SDL. Specifying both the software and hardware at system-level is very interesting. For software, this is ideal because of the existence of synthesis tools. The coding phase will not be necessary any more. However, SDL is not well suitable for design of HdS software for the following reason:

- The degree of parallelism and reactivity within the HdS modules is quite low. The HdS verification functions are called sequentially.
- The execution speed of the code generated from the SDL specification will be slower than a manually written code.
- The size of the code. By code generation from SDL specification, there is a significant increase in the size of the executable.

The last two issues are less important when using some constructs that aid in performance increase and reduction in size of the executable.

It is thus necessary to look for possibilities to link the HdS software to the SDL model of the hardware.

6.2 How to integrate SW with HW model?

The tools used during this project allow to describe SDL blocks or processes in C code instead of state machines. This possibility allows to integrate an SDL system with external code (sub-systems developed in C code). Blocks and processes implemented in C are referred as External Tasks.

Complex data structures, which are difficult to express in SDL can be described in C or OMT/UML and interfaced to the SDL system using ADTs. The possibility to interface with external routines is very important. Since implementing complex data structures using axioms is very difficult, the C or OMT/UML alternative gives SDL more opportunities.

The possibility to integrate legacy software is of vital significance mostly for firms producing a family of products. Their new systems are not produced from scratch.
Two possibilities are thus available to integrate external C code. These are via ADTs or via external tasks. Which of both allows best to interface the HdS verification software will be discussed in the next paragraphs.

6.2.1 Abstract Data Types

Abstract data types are used to represent the communication of reactive parts of the system, designed by means of SDL state machines, with external passive objects, developed in C. ADTs cannot access a process in SDL. For instance sending/receiving a signal to/from a specific process cannot be performed in a body of an ADT. Therefore, ADTs can not be used to integrate the HdS verification software considering that this software mostly performs hardware access (read from and write to registers).

6.2.2 External tasks

External tasks can be used to represent the environment, an SDL block or an SDL process. In this case, a dummy process or block is modeled in SDL. The behavior of such process or block will be described with external C code. In a configuration file, a reference is made to the external code in order to link it to the SDL application. The communication between external tasks and other SDL tasks takes place via mailboxes. A Run-Time Library provides a set of primitives, which facilitates this communication.

External tasks seem to be the unique possibility that can be used to integrate the HdS verification software. To make this integration possible SW must be adapted for this task. Read and write functions, which write to or read from the hardware registers, will be transformed to functions that send or receive signals to/from a channel. The code also can be left intact. Some macros must to be defined which will be executed when such a read/write function is called. The external task will have then the possibility to communicate via channels.

The external tasks are not supported by the simulator. In this case, the SDL description must be transformed into executable C code.

6.2.3 Co-verification environment

The External Tasks will be used to interface the software. The verification software is composed of a number of functions, which are sequentially called. Actually, the software is running on a processor called OBC. Therefore, it is necessary to model a processor, which will also synchronize the software and the hardware model. The OBC block already modeled in SDL was designed to play the role of the HdS software when this is not yet interfaced. Therefore, it was possible to verify and validate the SDL model at SDL level. The OBC is still necessary this time as an interface between the SDL model and the software. First, it will be used for the memory map. The addresses of the HW registers must be mapped to the corresponding SDL signal. Some HdS verification functions, which read to or write from the hardware, are performing this in a number of steps depending of the size of this data. This is necessary because only a 16-bit word can be read or written at once. The OBC block has to collect this data and send it to the Egress, TA or TG in one operation.
One of the problems encountered when trying to interface the SW and the HW is the memory map. The HdS software is using a common structure from which all other structures are derived. All registers of HW are given an address. At SDL level other structure are used and not all registers are represented. This can be solved for example by introducing a table in the OBC block, which contain for each address the corresponding signal name. A disadvantage of this solution is that it is time-consuming and not flexible. The addressing of registers depends in fact on the start address in the memory. However, this can be automated.

6.3 Generation of the application

When the External Tasks are involved, the SDL simulator is no longer available. This means that the SDL description must be transformed into C executable code. This transformation is performed using a code generator. Before generating the code of the SDL description, the logical structure of the SDL system must be mapped into the physical architecture of the target system. A target refers to a system on which the application will be run and it consists of a set of nodes (processors).

To define the physical architecture, four deployment strategies are used:

- **TS**: all process instances run on the same CPU and the same operating system task. This is useful when the application is run on a PC under DOS.
- **TB**: operating system tasks correspond to SDL blocks. All active process instances belonging to the same block or to the same block's sub-hierarchy, run in pseudo-parallel with the operating system task corresponding to an SDL block.
- **TP**: each process runs in a dedicated operating system task. All active instances of a specific process run in pseudo-parallel with the same operating system task.
- **TI**: each operating system task is mapped to an SDL process instance. Process instances run in true parallel.

Figure 9 shows an example of a generated application using TP partitioning. The application is distributed over two nodes, which communicate through the UDP/IP protocol. Inter-task communication on the same node is performed via the RTOS services.

The tool used in this project allows fully automatic code generation from SDL specifications. Because of the systematic use of macros, the generated code is very tunable and adaptable. This is necessary for example to make the generated application independent of the real-time operating system. The use of macros also allows to reach performances similar to those obtained with manual coding.
6.4 Co-verification

After the code generation from the SDL specification, the application is now ready for co-verification. The executable code of the hardware model and the HdS software can be run together. The whole application can be compiled. The produced application can then be executed either on the host machine or on the target platform. The Design Tracer tool allows to graphically view the application at SDL level. The HdS software implemented in the external tasks will also be debugged. At the end of the session, the execution scenarios can be transformed into MSCs to verify consistency with the SDL design.

6.5 Conclusion

In this Chapter, the possibilities to interface the verification software have been investigated. The dedicated External Tasks give the possibilities to perform HW/SW integration.

To use the External Tasks, the SDL description must be transformed into C executable code. Still, there are some problems unsolved when generating the code of the SDL model. However, Those problems are tool problems.

There is a need to building an environment in order to automate the customization of the code generation and the configuration of the SDL processes.
7 Conclusion and recommendations

7.1 Conclusions

In this project, the SDL formalism is used to create an executable specification of an ASIC. Through behavioral simulation and automatic message sequence chart test-generation, the model is verified and validated against the specification. The development of the SDL model is performed in a top-down manner in a number of steps. Each step, the necessary complexity is added. The abstraction level of SDL is very important for system level modeling. More details can be ignored so that the effort will be expended to describe the functionality of the system.

From this project, it can be concluded that SDL is feasible for hardware modeling for different purpose, e.g. as an executable specification and as a test-bench for the software. The main advantage of SDL is the possibility to model at different level of abstraction. The asynchronous communication and queuing mechanism is also important for the software at SDL level.

The level of abstraction is kept a bit low in order to satisfy the HdS verification software requirements. For the ASIC designer, modeling at system level will be useful since they are interested in the model for documentation and for high-level verification and validation purposes.

The nature of the HdS software demands a detailed memory map model to interface with the hardware. To be able to verify all HdS functions, all registers must be implemented in the SDL model. This requires much effort and time so that advantage provided by SDL to model at high level will not be optimally exploited.

The SDL simulator supports three simulation modes which, can be used at different phases of modeling. Especially, the exhaustive simulation is very useful in verification and validation of the SDL model. It explores all possible scenario's. Erroneous scenario's are saved as a message sequence chart. However, the size of the model must be reasonable which generally means small.

Message sequence charts are usually used to analyze the behavior at the requirements stage. When the SDL modeling is started, the requirement phase is already finished. Therefore, MSC are not exploited in this thesis. For building test plans, MSCs are very difficult and time-consuming when dealing with large models. MSCs have been automatically generated after each simulation, which gave the opportunity to view and analyze the behavior of the system.

The weak punt of SDL is data. At the beginning of the modeling, the use of data implemented in C is avoided. However, some data structure are needed, and the only alternative was C.
7.2 Future work and recommendation

First, the HdS verification software should be interfaced with the HW model, using external tasks. For these purposes, a co-verification environment should be developed as explained in Chapter 6.

It will be interesting to study the possibilities to develop the HdS software at system level using SDL, so that co-design can be performed. The whole system can then be developed in SDL.

During this project, it was clear that a lot of modeling languages are used for different purposes. A study should be done to those languages to check, which one is most appropriate for system-level hardware modeling.
Bibliography

[Alc] Alcatel
Alcatel 1000 S12: broadband specific hardware, Internal document.
Alcatel Telecom, Antwerp.

[Alc99] Alcala F., G. van Wauwe and...
MTM FEASIBILITY REPORT
Antwerp, February 1999.

[Bel91] Belina F., D. Hogrefe and...
SDL with application from protocol specification,

[Bon95] Bonati S.I. And R.J.O. Figueiredo
Stoht - An SDL-to-Hardware Translator
Proceedings of the ASPDAC/CHDL95/VLSI95,
Asia and South Pacific Design Automation Conference. P: 33-36
Nihon Gakkai Jimu Senta, Tokyo, Japan 1995

[Dav97] Daveau J.M, G.F. Marchioro and...
VHDL generation from SDL specification
Hardware Description Languages and their Applications, IFIP TC 10 WG10.5,
P: 182-201

The Alcatel SDL Method, volume 1, Internal rapport.

Introduction to SDL 92.

Specification and design of embedded systems

Using SDL for Hardware design

[Had97] T. Hadlich
Use of VHDL within a System Level Design flow
Proceedings VHDL International Users' Forum, IEEE Computer Society
P: 150-155
Los Alamitos, CA, USA, 1997
[Ism93] Ismail T.B, A.A. Jerraya and…
Linking system design tools and hardware design tools
Proceedings of the 11th IFIP WG10.2 International Conference on Computer
Hardware Description Languages and their Applications,
Ottawa, Ontario, Canada, A-32, pp345-351, 26-28 April, 1993

[Obj99] ObjectGEODE
Building and testing a real-time application.

[Obj97] ObjectGEODE
Code Generation Tutorial, version 4.

[Obj96] ObjectGEODE,
Object-Oriented Real-Time Techniques: method guidelines, version 1.0.

[Obj97] ObjectGEODE,
SDL C code generator, reference manual, version 2.4.

[Obj98] ObjectGEODE,
Training: Design and Simulation of Real-Time Systems.

[Obj98] ObjectGEODE
Training: Exercises, getting started with the ObjectGEODE editors.

[Pdv97] Putten van der P.H.A and J.P.M. Voeten
Specification of Reactive Hardware/Software Systems. Doctoral thesis,
Technical University of Eindhoven, Eindhoven The Netherlands, 1997

[Rah98] Rahman M. A.
Guide to ATM Systems and technology.

[Spe] Specification and description language, Internal document
Bell education center, Antwerp.

[Sta95] Stallings W.
ISDN and Broadband ISDN with frame relay and ATM, third edition.
[Tur93] Turner K.J.
Using Formal Description Techniques:
An introduction to Estelle, LOTOS and SDL.
John Wiley & Sons Ltd. 1993.

[Ver99] Verkinderen J.
HW TRS MTM, Internal document.
Antwerp January 22 1999.

HW-TRS, TLK-G for BBX, release 3.0.
Antwerp, February 1999.