



## Paper submission to ETW 2001

# GRAAL: a Tool for Highly Dependable SRAMs Generation

|   |  |
|---|--|
|  | <p>Silvia CHIUSANO, Giorgio DI NATALE, Paolo PRINETTO<sup>1</sup><br/> <b>Politecnico di Torino</b><br/> Dipartimento di Automatica e Informatica<br/> Torino, Italy<br/> E-mail: {family name}@polito.it<br/> <a href="http://www.testgroup.polito.it">www.testgroup.polito.it</a></p>                |
|  | <p>Franco BIGONGIARI<br/> Aurelia Microelectronica<br/> Cascina (PI), Italy<br/> E-mail: <a href="mailto:franco.bigongiari@aurelia-micro.it">franco.bigongiari@aurelia-micro.it</a><br/> <a href="http://servermicro.aurelia-micro.it/index.htm">http://servermicro.aurelia-micro.it/index.htm</a></p> |

### Abstract<sup>2</sup>

*Commercial tools for the automatic insertion of testing structures so far cover the generation of highly dependable SRAMs only partially. This paper presents a tool to achieve proper reliability levels in systems based on memories. The tool allows the automatic insertion of BIST architectures for both OFF-line and ON-line memory testing. The set of algorithms and architectures supported by the tool is not limited, but it can be easily extended, to include innovative architectures and achieve the reliability requirements in any application.*

*Using the tool, the user can generate dependable memories trading-off in the design process the dependability properties and costs.*

**Keywords:**  
**MEMORY TESTING, ATE**

---

<sup>1</sup> Contact Author: Paolo PRINETTO  
Politecnico di Torino, Dipartimento di Automatica e Informatica  
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy  
Email: Paolo.Prinetto@polito.it  
Phone: +39-011-5647007 Fax: +39-011-5647099

<sup>2</sup> This work has been developed under the project GRAAL S167P, funded by the *Italian Ministry of the University and Technological and Scientific Research (MURST)*.

## 1. Introduction

Memories are key components in highly dependable applications as space, defense and automotive systems, and biomedical devices, since they play a crucial role in terms of *availability* and *serviceability*. They can appear in a big variety of sizes, technologies (SRAMs, DRAMs, Rambus,...), and packaging (IP cores, chips, dedicated boards). However, regardless of their actual implementation, memories proved to be among the most critical devices with respect to both *permanent* and *transient* faults, mainly due to the environmental stresses and interferences.

Strict requirements in terms of *fault latency* usually imply a continuous monitoring of memories, aiming at minimizing the time elapsed between the occurrence and the detection of a fault, while avoiding any interruption or degradation of the system performance. The adopted test strategies are usually based on *Built-In Self-Test* (BIST) techniques, where ON-line (Concurrent and Not-Concurrent) [1][2][3] and OFF-line [4][5] approaches often coexist. In particular, Concurrent ON-line BIST addresses transient faults, continuously testing the system, concurrently with its normal behavior. Not-Concurrent ON-line and OFF-line BIST, instead, are carried on during dedicated time frames, where the system behavior is suspended, and it is used to periodically run deeper tests, aiming at detecting faults left uncovered by the Concurrent ON-line test [6][7].

Commercial tools are nowadays available for the automatic insertion of the testing architectures in complex systems [8][9]. These tools aim at improving the dependability properties of the systems globally, facing with a wide range of different problems, like the complexity of today's System-on-Chips exceeding one million gates, the availability of multi-clock domains, the requirements for at-speed and IP cores testing. The memory testing is one of the aspects addressed by the tools, which typically propose a set of effective, but limited, testing solutions. They mainly allow the insertion of BIST logic for OFF-line testing, but few March-Tests are supported, and finally the ON-line testing is not targeted. Therefore, nevertheless their strong effectiveness in the majority of the applications, these tools do not guarantee the proper reliability levels in safety critical systems based on memories.

In this scenario, the present paper proposes a tool targeting the automatic generation of highly dependable SRAMs. The tool is named *GRAAL* and has been developed

under the project S167P founded by the *Italian Ministry of the University and Technological and Scientific Research (MURST)*.

GRAAL allows the automatic insertion of OFF-line and ON-line BIST architectures for memory testing. The set of architectures supported by the tool is not limited, but it can be easily extended to meet the dependability requirements or to include the most innovative and last developed testing algorithms and architectures. GRAAL is therefore complementary to the available commercial tools, since it targets aspects, which so far they partially covered, only.

The innovative aspects in GRAAL are the flexibility properties of the tool. GRAAL has been designed, developed and implemented targeting the following issues:

- *Independence*, from the technology, the system embedding the memory, and the EDA environment
- *Easy x-ability*, in terms of tool upgrade- and maintain-ability, and advanced design reusability
- *Trade-off* the dependability-cost in the design process.

Thanks to GRAAL easy upgradeability and maintainability, there is not a-priori limitation on the OFF-line and ON-line architectures that can be designed using GRAAL.

In the actual version, the tool deals with single port memory but it can be easily extended to multi-ports memories.

This paper is organized as follows: Section 2 overviews the tool summarizing how the achievement of the GRAAL targets has been addressed. Section 3, 4, and 5 briefly focus on three aspects which are key in guaranteeing the GRAAL flexibility, respectively the dependable memory architecture, the structure of the GRAAL database, and the design flow in GRAAL. Section 6 eventually draws some conclusions.

## **2. GRAAL overview**

To achieve the goals listed in Section 1, we mainly resort to two avenues of attack: define a proper structure for the tool and design the dependable memory architecture in a suitable way.

The GRAAL structure is sketched in Figure 1. The easily upgrade- and maintainability of the tool is obtained distinguishing two separate working environments, respectively for the actual designers and for the tool manager. Moreover, a proper library is defined to store the dependable architectures.

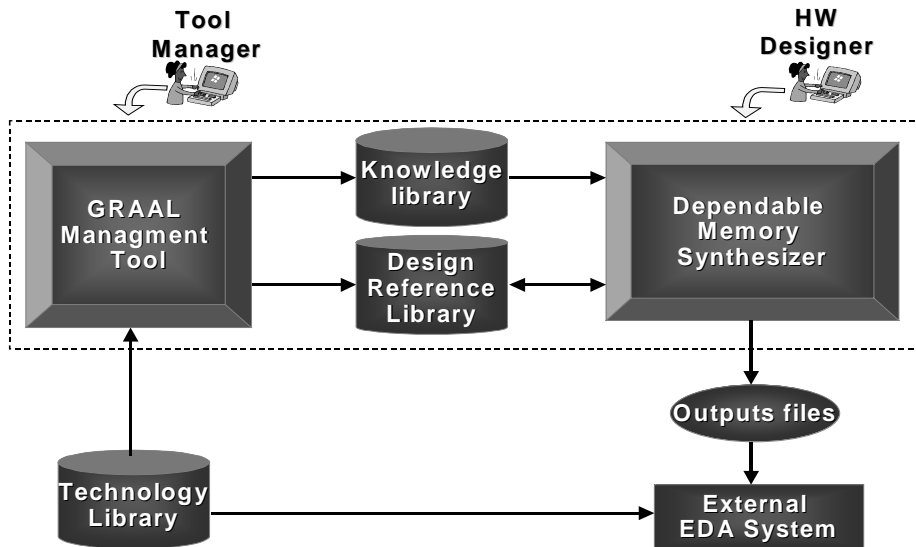
In Figure 1, the *Dependable Memory Synthesizer* (DMS) is the kernel of GRAAL and is the environment exploited by the user to select in the *Knowledge Library* the proper dependable SRAM architecture. The *Knowledge Library* stores the dependable SRAM architectures, and additional information to support the design flow.

The *GRAAL Management Tool* (GMT) is used by the tool manager to update the content of the *Knowledge Library*. Mainly, the tool manager inserts new architectures, removes some existing ones, and stores for the available ones their technology dependent characteristics. To achieve *technology independence*, the tool manager derives the proper information from a set of external libraries, named in Figure 1 *Technology Libraries*.

The *advanced design reusability*, today one of the main issue in the design flow, is achieved in GRAAL through a proper *Design Reference Library*, a repository of dependable SRAM architectures, results of previous projects or preferably used by the designer.

Finally, the GRAAL outputs are readable by the major *EDA environments*. The outputs include the synthesizable descriptions of the dependable architectures, in VHDL language, and some scripts to interface with commercial synthesis and testing tools. The scripts allow executing the synthesis process and performing the scan insertion in the architecture. Being not customized on any specific EDA environment, GRAAL is open to be widely used in the most common industrial design flow.

Concerning the dependable architecture, to make it *independent* from both the *embedding system* and the *target memory*, the tool wraps the SRAM by a *dependable memory collar*, in charge of interfacing the memory to the external environment and guaranteeing the required dependability levels. The collar has been designed using a strongly modular-based approach, and the user can select the suitable dependable architecture *trading-off* the dependability properties and the implementation costs.

Figure 1: *The GRAAL tool*

### 3. The Dependable Memory Collar (DMC)

The *dependable memory collar* (DMC) embeds the dependability logic, and the logic to interface with both the memory and the system environment in which the memory is located. It includes the *Built-In Self Test* (BIST) logic for both OFF-line and ON-line memory testing. To fulfill the target goals listed in Section 2, the DMC has been designed in order to achieve a strongly modular structure.

The proposed approach relies on a set of elementary building blocks, each one targeting different functionalities of the collar. The blocks are designed such as they can be developed, updated and debugged as a stand-alone, but at the same time easily *composed* to create a proper collar. Ad-hoc defined *rules* drive the building process, and collars with various characteristics can be generated composing the blocks in different ways. The elementary building blocks are named *Layers*.

#### 3.1. The Layers

Figure 2 sketches the DMS structure: the collar consists in a four-Layers hierarchy, in which the Layers are placed in an *onion skin-like* mode.

The DMC is first decomposed into three Layers: a middle Layer named *Test Layer* (TL), and two external Layers, named *System Interface Layer* (SIL) and the *Memory Interface Layer* (MIL). The TL contains all the dependability logic and therefore represents the kernel of the DMC; the remaining two Layers, the SIL and the MIL, interfaces the TL with a specific system environment and with a given SRAM, respectively.

The TL is then further sliced into two sub-Layers:

- The *Test Access Method Layer* (TAML), containing the BIST logic for OFF-line and/or Not-Concurrent ON-line memory testing.
- The *Encoding Layer* (EL), containing the BIST logic for Concurrent ON-line memory testing; it mainly relies on data encoding.

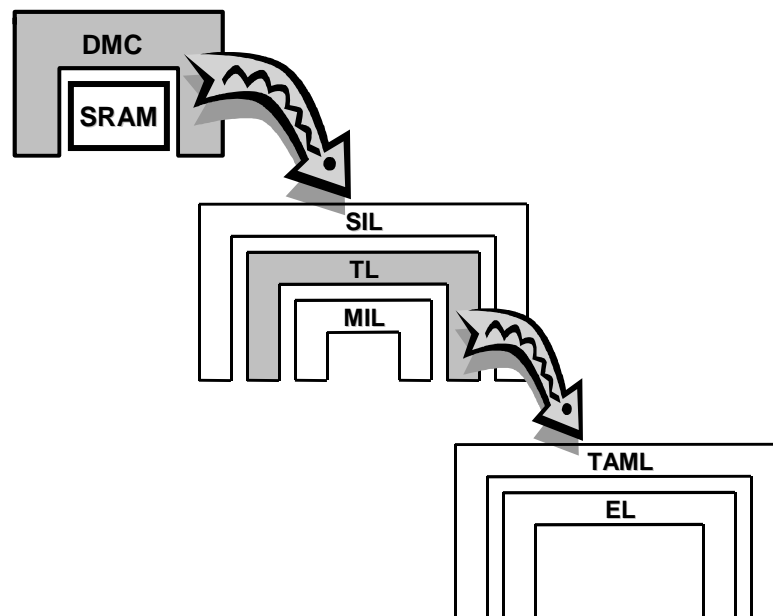


Figure 2: *The Dependable Memory Collar (DMC)*

Each Layer represents a sort of *black box*, which can be filled by different architectures, able to guarantee the functionality proper of the Layer, but differing in terms of *dependability* and *design properties*. The design properties are technology-dependent characteristics, as the area and the power consumption. The dependability properties are the reliability characteristics of the Layer, as the performed testing strategies, the detected faults, and the capability of correcting the memory data when affected by a fault.

As an example, for the TAML different architectures are obtained depending on the executed March-Test [10]; in the case of the EL, the implementations differ based on the performed encoding algorithm. Eventually, the MIL and the SIL are specific for each memory and system environment.

To support the onion-skin-like structure drawn in Figure 2, a proper protocol has been defined to exchange signals in the DMC. The communications occur between neighbors Layers and the input/output protocol is fixed a-priori and able to support all the different implementations of the Layers.

The proposed design solution achieves the following advantages. Being fixed the communication among the Layers, they can be developed, updated and debugged as a stand-alone. This aspects contributes to the easily upgrade- and maintain-ability of GRAAL.

Moreover, the DMW structure contributes to the independence from the system and the technology in GRAAL, since using the MIL and the SIL the TL becomes independent from the specific protocol required by both the memory and by the system environment.

The MIL and the SIL converts the TL input/output signals into the proper protocol required by the memory and the system environment, respectively. Therefore, the TL does not interface either with the *actual* memory and system, but with a *virtual* memory and system, which exchange signals using a fixed I/O protocol.

### 3.2. The DMC characterization

The DMS is created specifying the Layers implementations, and its characteristics are obtained *composing*, through a set of *composition rules*, the design and dependability properties of the instantiated architectures.

For the Layers in the TL, the architectures are chosen in order to fulfill the project goals. The MIL and the SIL mainly introduce some extra logic for interfacing and they are specific for each memory and system environment.

Table 1 and Table 2 list some composition rules for design and dependability characteristics, respectively. In the former case, the rules are mainly arithmetic operators: as an example, the DMC area is computed summing (rule “Sum”) the areas contributes of the architectures.

| Property          | Type of Value | Composition Rule |
|-------------------|---------------|------------------|
| Area              | Integer       | Sum              |
| Area Overhead     | Percentage    | Sum              |
| Power Consumption | Real          | Sum              |

Table 1: *Composition rules for the design properties*

In the latter case, the rules are typically logic operators: as an example, the DMC detects shorts faults if at least one of the available architectures (rule “Or”) detects those faults.

| Property         | Type of Value | Composition Rule |
|------------------|---------------|------------------|
| Stuck-at Covered | Boolean       | OR               |
| Short Covered    | Boolean       | OR               |

Table 2: *Composition rules for the dependability properties*

## 4. The Knowledge Library

The DMC structure described in Section 3 maps in the Knowledge Library as three Layer-Libraries, respectively for the MIL, TAML, and EL, each Library storing the set of possible architectures for the corresponding Layer. The SIL only is not defined a priori but is designed by the user supported by the tool.

### 4.1. The TAML and EL Libraries

To make the Knowledge Library easily upgradeable, the libraries for the two Layers in the TL have been structured in a two-levels hierarchy.

The basic idea is to decompose the Layer architectures in elementary blocks, distinguishing between the Layer *structures*, and the *implementations* of the instanced components. The *structure* is the RT-level description of the architecture; in the structure the components and the interconnections are instantiated, but the components are still as black boxes and no a specific implementation is assigned to them.

The proposed approach relies on the fact that often architectures have an equivalent *structure*, but they diversify for the actual implementations of the available compo-

nents. As an example, the architectures for March-Test execution share the same RT-level structure, but the actual descriptions of the BIST Controller and a Background Pattern Generator depend on the performed March-Test. In a similar way, the same RT-level description characterizes the architectures for ON-line error detection, but the available encoding module is implemented differently based on the selected data-encoding algorithm [11].

Based on the proposed approach, the three Layers Libraries have been organized as sketched in Figure 3; each of them consists of:

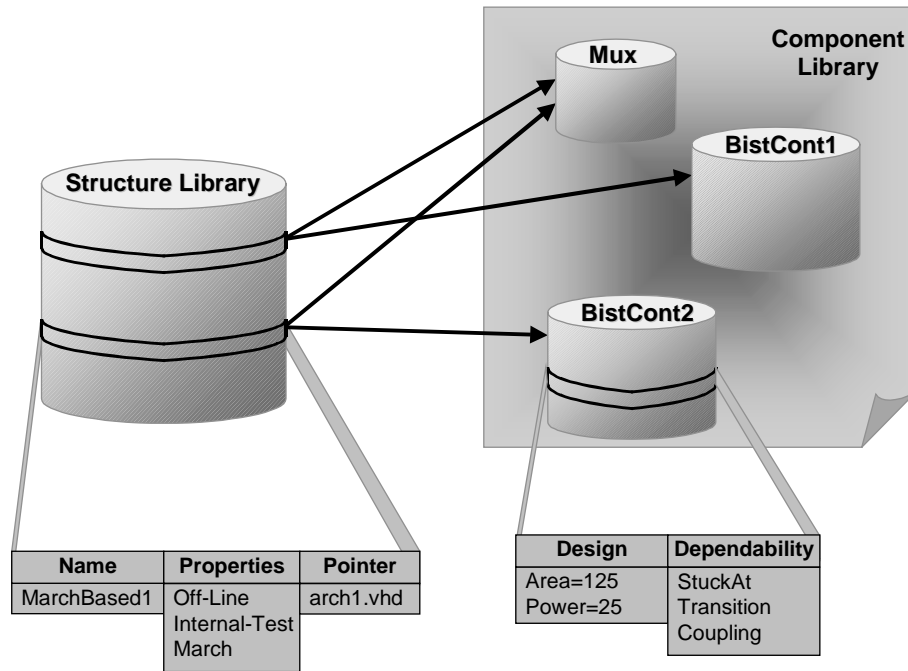
- A *Structures-library*, storing the RT-level descriptions of the architectures.
- A set of *Component-library*, one for each component instantiated in the Structures-library, and storing the possible implementations the component.

Each entry in the Structures-library corresponds to a different structure; it includes the pointer to the VHDL RT-level description, and for each instanced component the links to the proper Component-library. Moreover, it contains the list of dependability properties characterizing the structure, like whether it performs OFF-line memory testing, Not concurrent On-line memory testing, memory data detection or memory data correction.

Each entry in the Component-library corresponds to a different implementation of the component; it contains the pointer to the VHDL description, and the list of design and dependability properties of the implementation. As an example, in the case of the BIST Controller, the dependability properties are the list of faults detected by the implemented March Test; for the Encoder component, the list includes the faults detected via the implemented encoding algorithm.

In the Layer Library, each *structure* and each *component implementation* is *uniquely* described by the list of its dependability properties.

The Layer architecture is fully characterized once first a structure is fixed and then an implementation for the components instantiated in the structure is selected. The dependability and design properties for the architecture are obtained composing those of the available components, and the evaluation process is driven by the composition rules. The TL is fully characterized once the architectures of its two Layers have been specified.

Figure 3: *The TAML and EL Libraries*

## 4.2. The MIL Library

Each entry in the MIL library corresponds to a different type of memory. It stores the technology for memory implementation, the range of possible word size and cells number supported by the technology, and the list of design properties, parametric in the memory size. Moreover, it contains the pointer to the VHDL description of the MIL. For each different memory a specific MIL is available.

## 5. The Dependable Memory Synthesizer (DMS)

The DMS is the tool in GRAAL exploited by the user to design the DMC; this tool therefore represents the kernel of GRAAL.

The design process is carried on characterizing the Layers in the DMC one at the time; the user interacts with the tool and drives the process in order to achieve the target requirements for the dependable RAM architecture.

Figure 4 sketches the basic design flow. The process is mainly structured into three phases, which run sequentially: the *Memory Selection* first, to state the target memory,

the *TL Designing* then, to fully specify the Layers in the TL, and the *SIL Designing* finally, to interface the TL with the external environment. The three phases outputs the VHDL descriptions of the MIL, TL, and SIL, respectively.

The execution of the three phases relies on content of the Knowledge Library for the needed information; the TL Designing phase is further supported by the Design Reference Library.

Before ending the design process, the *Windup Phase* collects the MIL, TL, and SIL VHDL files and provides the final synthesizable VHDL description of the DMC. Moreover, it generates a report summarizing the DMC characteristics, and some *scripts* to interface with commercial synthesis and testing tools. The scripts allow the synthesis of the dependable architecture and the automatic insertion of scan chains in the DMC.

The DMS tool has a strongly modular structure, since the phases are independent, and exchange information through the *Project Database*. The database is filled up during the design flow and at the end its content summarizes the characteristics of the DMC.

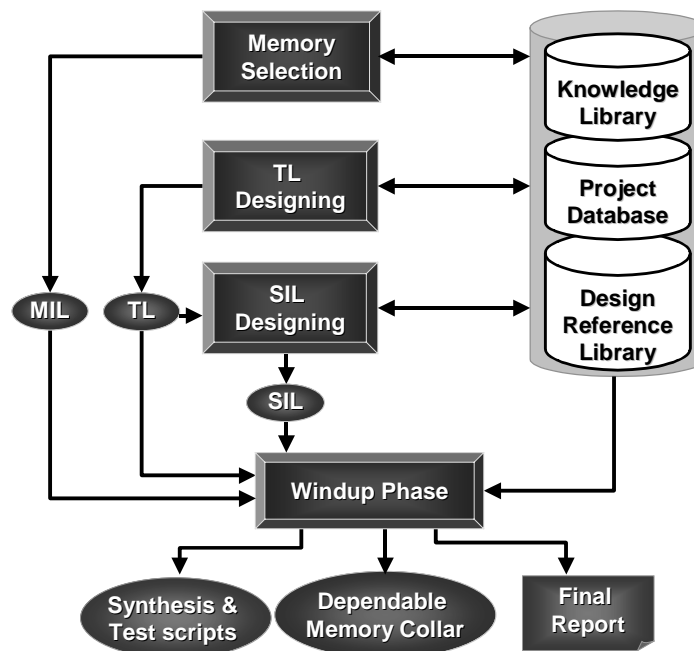


Figure 4: *The DMS Design Flow*

## 5.1. The DMS Main Console

The DMS tool has been implemented in standard C, with a graphical front-end for Microsoft Windows®. Figure 5 shows the DMS main console; it includes three sections: one *displayer* for all the graphical user interfaces (GUIs) activated through the design flow (on the left side), one *monitor* for the DMC properties (on the bottom-right), and finally one *tracer* showing the actual step in the design flow (on the top-right).

The *monitor* shows the target and actual values for all the design and dependability properties of the DMC, allowing to verify the consistency between them. The target values are typically stated by the user at the beginning of the project, while the actual values are automatically updated during the whole design flow based on the adopted solutions.

The *tracer* shows the whole Layers-hierarchy in the DMC. The Layers are colored differently based on their status: the *white* color states not-characterized Layers i.e., they still are black boxes; the *light grey* partially characterized Layers, i.e., their structures are specified; the *dark grey* fully specified Layer, i.e., specific architectures are assigned to them.

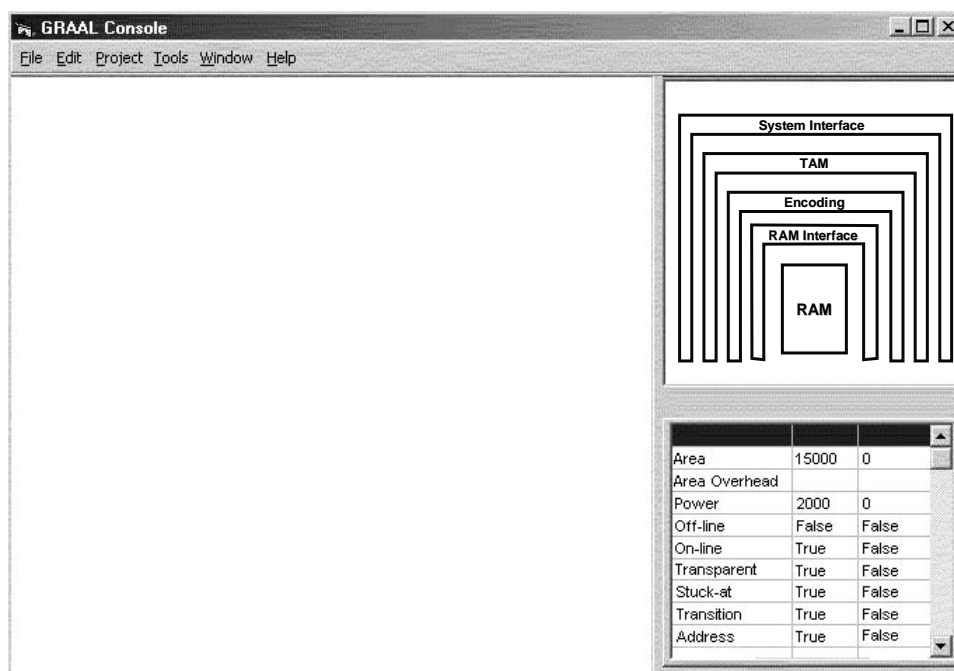


Figure 5: The DMS Main Console

## 5.2. The Design Flow

The present section describes the design phases, showing how they are implemented in the DMS.

### 5.2.1. Phase 1: Memory Selection

Figure 6 shows the GUI for the memory selection; it allows efficiently accessing the MIL Library to select the target SRAM.

The user specifies the technology (*Technology*) and the memory size (*Word Size* and *Cells Number*), and then selects among the memories available for that technology the one (*Memory*) having suitable design properties (*Estimated Parameters*). The design properties (e.g., area, power consumption) are run time evaluated by the tool.

At the end, the memory characteristics are stored in the Project Database, and the MIL VHDL description for the target memory is provided.

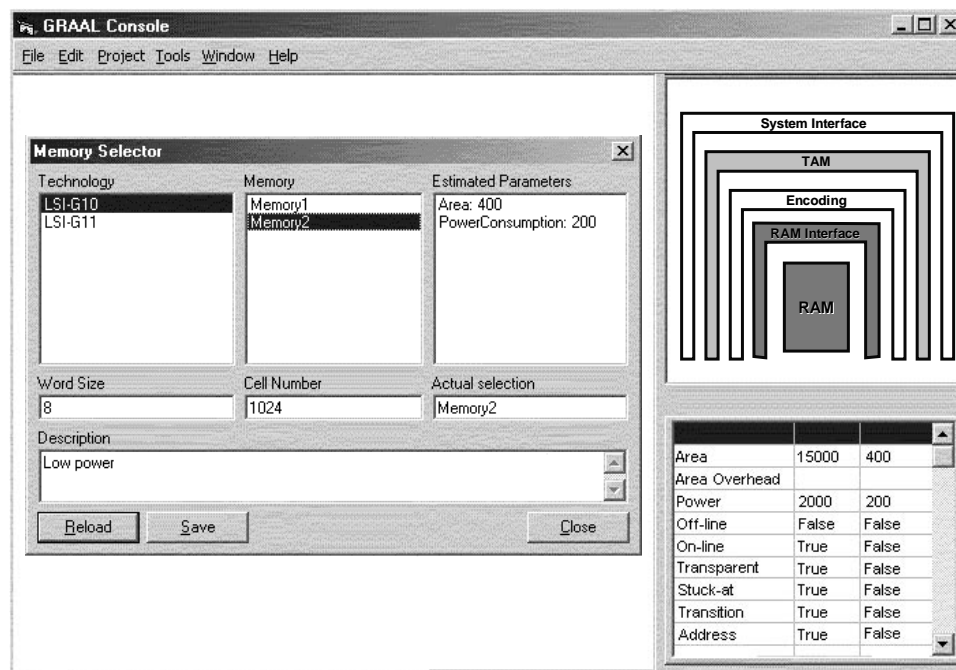


Figure 6: *Memory Selection GUI*

### 5.2.2. Phase 2: TL Designing

The present phase is further structured in two steps, named *Structures Selection* and *Components characterization*. In the former, based on the target dependability

requirements the user fixes the functionality to include in the TL, i.e., it selects the proper structure for each Layer in the TL. In the latter, the user completes the Layers characterization, selecting a specific implementation for the instantiated components.

During the present phase, the main design and dependability properties of the DMC are fixed. When concluded, the Project Database is updated with these values, and the tool outputs the VHDL descriptions of the Layers structures and of the instantiated components.

#### **5.2.2.1. Structures Selection**

This process is carried on in an interactive way, considering the Layers one at the time. For the target Layer, the user sets the desiderata *dependability properties* and the tool uses this information to find out in the Knowledge Library the Layer *structure* characterized by such properties.

Figure 6 reports the GUI: on the right, the whole list of all the dependability properties characterizing the structures of the Layer is showed; sideways, the set of possible structures is displayed.

The user identifies a specific structure out of the available ones in one of the two following ways. He can directly select a specific structure, and in this case the tool marks the matching dependability properties in green and in red the others.

As an alternative, the user can select one at a time, the target dependability properties. Once a property is selected, the tool marks in green the matching structures and in red the others.

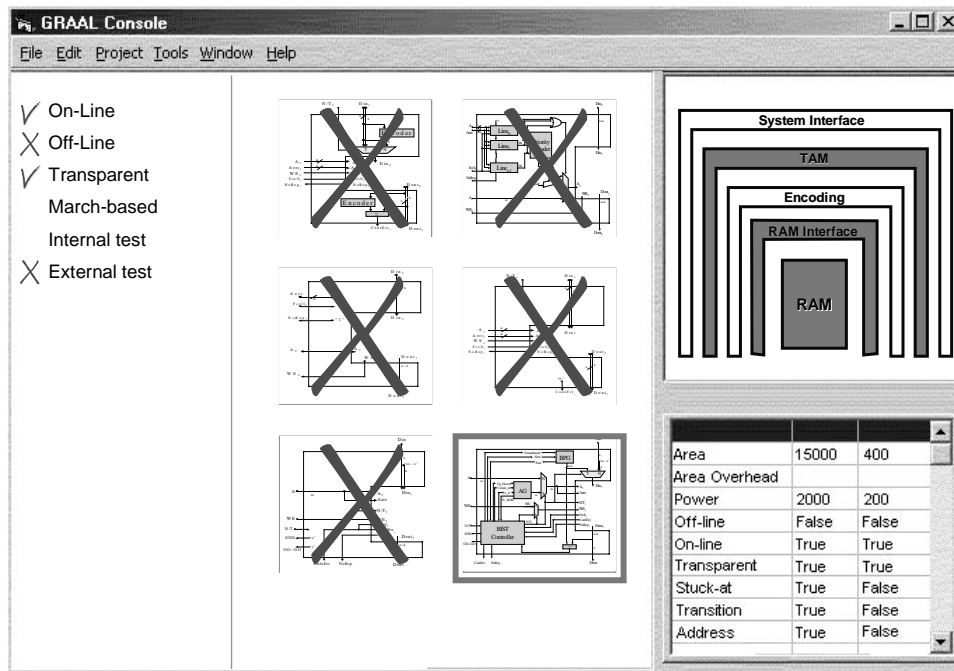


Figure 6: Structures Selection GUI

#### 5.2.2.2. Components Characterization

Using an approach similar to the one for selecting the Layer structure, the user completes the TL characterization fixing an implementation for all the instantiated components.

The components are considered one at a time. The tool shows the whole list of all the dependability properties characterizing the implementation of the component and the set of its possible implementations. The user either selects a specific implementation directly or states the target dependability requirements.

Besides, during the present phase the user can rely on the *Design Reference Library*, and select one of the components implementations stored in it.

Going on with the characterization process, the design and dependability properties of the TL are modified according to the properties of the adopted solutions. Typically during the present phase, the user evaluates the consistency between the target and actual values of the DMC properties. Based on the result of the comparison, the user can either select a different implementation for some/all the instanced components or re-execute Phase 2 and selects different structures for the TL.

### 5.2.3. Phase 3: SIL Designing

The user interacts with the tool to describe the SIL, based on the TL functionalities and on the communication protocol of the external environment. The SIL is described in VHDL language.

The tool provides a VHDL template in which the whole set of TL input/output signals is listed; the user links these signals with the ones coming from/going to the external environment. The GUI is shown in Figure 7, and contains both a VHDL editor to easily modify the template, and a VHDL compiler to verify its correctness.

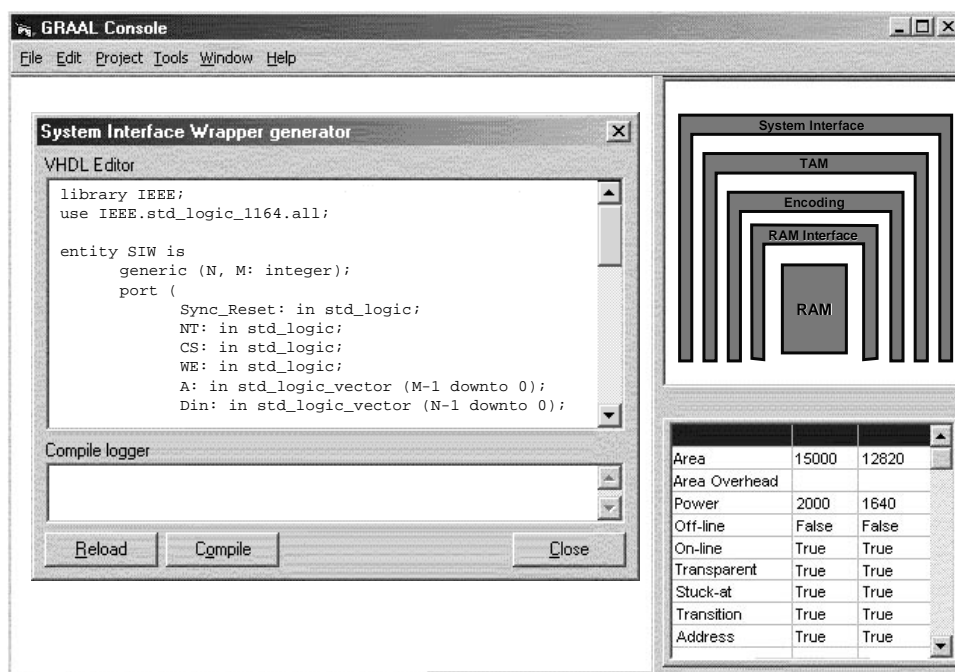


Figure 7: The SIL Designing GUI

### 5.2.4. Phase 4: Windup Phase

This phase collects the VHDL description of the *SIL*, *TL Structures*, *TL Components*, and the *MIL*, and produces the final VHDL of the whole dependable memory collar. Moreover, the tool generates scripts to synthesize the final architecture, and the automatically insert scan chains in the DMC, with commercial tools.

## 6. Conclusions

This paper presented a tool named GRAAL for the automatic generation of highly dependable memories. A proper structure has been defined for both the tool and the dependable memory, in order to achieve a very high degree of flexibility.

The easy upgrade- and maintain-ability of the tool makes it open to support always the most advanced testing strategies and architectures; the design flow allows the user to identify the proper architectures trading-off the dependability-cost. GRAAL can be complementary to the available commercial tools for inserting testing structures, since it targets aspects, which so far they partially covered, only.

## 7. References

- [1] H. Al-Asaad, B. T. Murray, J. P. Hayes, *Online BIST for Embedded Systems*, IEEE Design & Test of Computers, October 1998, pp. 17-24
- [2] R. Karri, M. Nicolaidis, *Online VLSI Testing*, IEEE Design & Test of Computers, October 1998, pp. 12-16
- [3] A. Steininger, C. Scherrer, *On the Necessity of On-line-BIST in Safety-Critical Applications - A Case-Study*, Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (FTCS-99), 1999, pp. 208-215
- [4] B. F. Cockburn, *Tutorial on Semiconductor Memory Testing*, Journal of Electronic Testing: Theory and Applications, 5, pp. 321-336, Kluwer Academic, Boston, MA (USA), 1994
- [5] A. J. Van De Goor, C. A. Verruijt, *An Overview of Deterministic Functional RAM Chip Testing*, ACM Computing Surveys, Vol. 22, No. 1, pp. 5-33, March 1990
- [6] P. Bardel, W. McAnney, J. Savir, *Built-in Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, New York, 1987
- [7] M. Abramovic, M. Breuer, A. Friedman, *Digital System Testing and Testable Design*, Computer Science Press, New York, 1990
- [8] LogicVision®, *memBIST-IC User Manual*, January 1999
- [9] Mentor Graphics Corporation®, *MBISTArchitect Manual*, 1999
- [10] A.J. Van de Goor, *Testing semiconductor memories: theory and practice*, Wiley, Chichester (UK), 1991
- [11] F. J. Macwilliams, N. J. A. Sloane, *The Theory of Error-Correcting Codes II*, North-Holland Mathematical Library, Vol. 16