Development of Reliability Aware Intelligent Memory Manager in an SoC (SoC-RAIMM)

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

In

Electronics & Communication Engineering

(VLSI Design)

By

Soyeb A. Khanusiya

(10MECV29)

Department of Electronics & Communication Engineering

Institute of Technology

Nirma University

Ahmedabad-382 481

May 2012

Development of Reliability Aware Intelligent Memory Manager in an SoC (SoC-RAIMM)

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

In

Electronics & Communication Engineering

(VLSI Design)

By

Soyeb A. Khanusiya

(10MECV29)

Under the Internal guidance of

Prof. Amisha Naik

and

External guidance of

Mr. Deepak Baranwal

Department of Electronics & Communication Engineering

Institute of Technology

Nirma University

Ahmedabad-382 481

Declaration

This is to certify that

1. The thesis comprises of my orginal work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgement has been made in the text to all other material used.

Soyeb A. Khanusiya

Certificate

This is to certify that the Major Project entitled **"Development of Reliability Aware Intelligent Memory Manager in an SoC (SoC-RAIMM)"** submitted by **Soyeb A. Khanusiya**,towards the partial fulfilment of the requirements for the degree of Master of Technology in VLSI Design of Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: **Place:** Ahmedabad

Internal Guide

Prof. Amisha Naik
(Professor, EC)

External Guide

Mr. Deepak Baranwal
(Senior Engineering Specialist)
(ST Microelectronics)

HOD

Prof. A. S. Ranade
(Professor, EC)

Programm Co-ordinator

Dr. N. M. Devashrayee
(Professor, EC)

Acknowledgement

It is my immense pleasure to acknowledge the persons without whom this dissertation would be incomplete or even impossible.

I wish to thank first and foremost, Prof. A. S. Ranade, Head of Department, Electrical Engineering and Dr.N.M.Devashrayee, Program Coordinator, M.Tech - Electronics and Communication(VLSI Design), for allowing me to pursue my dissertation from ST Microelectronics, Greater Noida. I am thankful to ST Microelectronics for providing me the exposure to industry.

I owe my deepest thank to my external guide Mr. Deepak Baranwal (APG, ST Microelectronics) and Internal Guide Prof. Amish Naik, Nirma University, Ahmedabad to provide me the guidance throughout the dissertation. They are always there to support me whenever I need them.

I consider it an honor to work with such a team headed by Mr. Deepak Baranwal. I would like to thank my team members Mr. Digvijay Pratap Singh , Mr. Soyeb Khanusiya and Mr. Sandhesh for providing me the space to work with them. I thank all my team members for sharing their experience with me and helping me in situation where I need them whether it is technically or personally.

I am deeply obliged to all the faculties, Support staff, Lab assistants, of Nirma University, Ahmedabad for sharing their knowledge and experience with me. I cannot find words to express my gratitude to my dearest and nearest friends for their unconditional love and support. I am indebted to all of them for the golden time of my life which I had spent with them. Thanks a lot for being with me and making my life enjoyable.

Last but not the least I would like to acknowledge my parents without them I may not be able to acknowledge the above all personalities. It is their love, support and sacrifice because of which I am here at this stage of my life. I would also like to express my love to my elder brother Zulfikar for giving me the reason to live life.

- Soyeb A. Khanusiya

Abstract

As the geometries of the transistor reach the physical limits of operation, one of the main design challenges of System On Chips (SoCs) will be to provide the support against permanent and intermittent faults that can occur in the system. One of the most critical elements that affect the correct behavior of the system is the unreliable operation of on chip memories. In this dissertation report, we present a solution for the reliability of on chip memories of SoCs for sub nm technologies. This report is concentrating on the Reliability with respect to the Process, Voltage and Temperature variation. In sub nm technology this PVT variation effects are more visible. These variations is random in nature, the main challenge is to monitoring these random variation and taking some action to increasing SoC reliability. We introduce the concept of RAIMM (Reliability Aware Intelligent Memory Management) it allows to examine the effects of Process, Temperature and voltage (PVT) variation on system memories. It also allows to continuously reliability prediction for system memories and system level solutions for unreliable memories.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivations behind the RAIMM

Circuit designers have long been aware of the three primary categories of variation across a chip: Process Parameters for NMOS and PMOS devices, Supply Voltage, and Temperature – collectively referred to as 'PVT'. Each one varies independently, although there may be correlations between them. These variations could generally be different from within a chip, from chip-to-chip, and from wafer-to-wafer due to integrated circuit (IC) wafer processing techniques. However, the chip-to-chip variations and wafer-to-wafer variations would equally impact all components and branches of a single chip. Therefore, the greater concern for system performance is due to intra-chip PVT variation. The PVT variation could be time-varying during circuit operation. Typically, process variations tend to be static in nature but may drift slowly over time, while supply voltage and temperature variations are truly dynamic in nature and depend primarily on the layout of the power/ground grids, the floorplan of the blocks on the chip, and the activity levels of the blocks themselves. As a result, PVT variations must be expected throughout any chip and will affect latency, threshold voltage, clock skew and etc. Process variations can be further classified into two distinct categories based upon their degree of correlation across

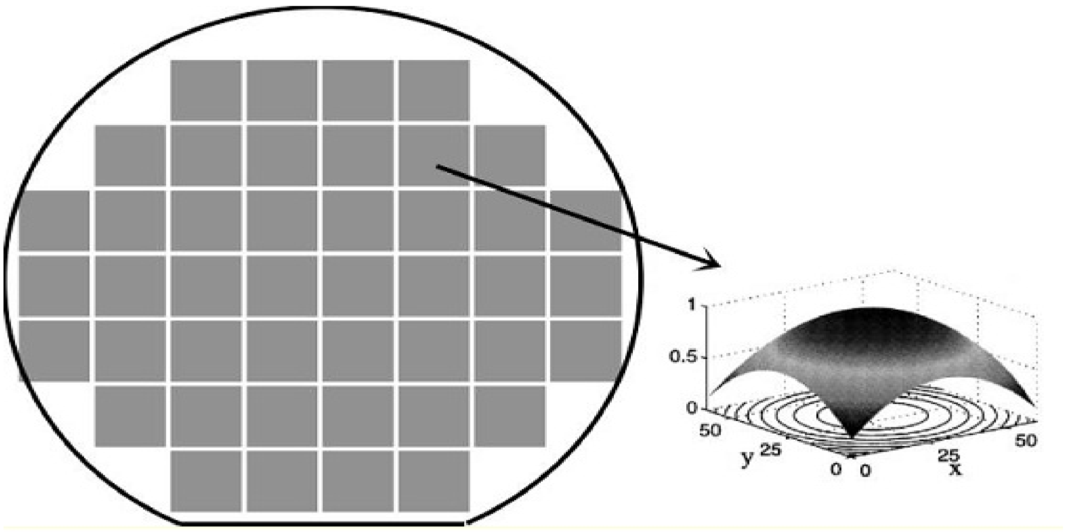


Figure 1.1: Process variation over one die of a wafer

the chip. Random variations are considered uncorrelated spatially and therefore they can be modeled as random variables at every location on the chip. Spatial variations are highly-correlated across the chip and therefore are modeled as a spatial pattern or gradient across the chip. Some circuits (e.g., small, matched, localized blocks) may be more affected by random variations, while others (e.g., large, global, dispersed blocks) may be more affected by spatial variations; so, both random and spatial variations must be considered in system timing analysis.

Vt variation with PVT:

Threshold Voltage (VT) variation is caused by two types of sources: systematic and random. Systematic sources include lithography-induced variations in channel length and width, which are deterministic and predictable. Random sources include as gate line-edge-roughness (LER), random-dopant-fluctuation (RDF), and gate work-function variation (WFV), which are non-deterministic and are projected to be the dominant sources of VT variation for transistor. The effect of random fluctuation is because of the number of impurity atoms, as one of the issues for continued transistor scaling. Analytical models and three-dimensional (3D) finegrid statistical device simulations were subsequently used to understand and predict RDF-induced VT variation in deep sub-micron devices. It is well known that RDF-induced VT is inversely proportional to (WL)0.5, where W and L are the transistor channel width and length, respectively.

Intra Die Variation:

By lowering supply voltage, the dynamic power decreases quadratically. On the other hand, the process variation sensitivity increases. PVT variation effects are more stringent in small size devices. Variations in channel length, channel width, oxide thickness, threshold voltage, line-edge roughness, and random dopant fluctuations are the sources of the inter-die and the intra-die variations in process parameters. One of this inter die effects is threshold voltage variations due to the changes of a single transistor (e.g. threshold voltage increases if temperature is reduced). However, intra-die variations may be different in one transistor to another (i.e. increase in Vth for one device on the other hand decrease in threshold voltage). An example of the systematic intra-die variation can be the change in the channel length of different transistors of a die that are spatially correlated. The RDF induced Vth variation is a classic example of the random intra-die variation. Fig.2 shows the Monte Carlo simulations for a single NMOS transistor. As it can be seen the drain current variation for lower supply voltages is higher. Fig 3 shows same for the PMOS transistor. As the supply voltage is reduced the process variation is increased. So with a small change in threshold voltage due to RDF, LER, aging effect, and etc, the drive current is changed significantly causing different behavior and in some circuits causing failure.

Fig 1.4 shows the effect of process variation on sub threshold current as it can be seen, process variation is dominant and the sigma is changed from 4.33nA to 4.71nA As shown in fig 1.5, the gate current is changed only by 0.5pA from - 40C to 125C while sub is changed by a value around 12 .5nA. So, we can see that at lower supply voltages, gate leakage is neglected and the process variations caused by this current has no significant effect on circuit evaluation. The effects of process variations on

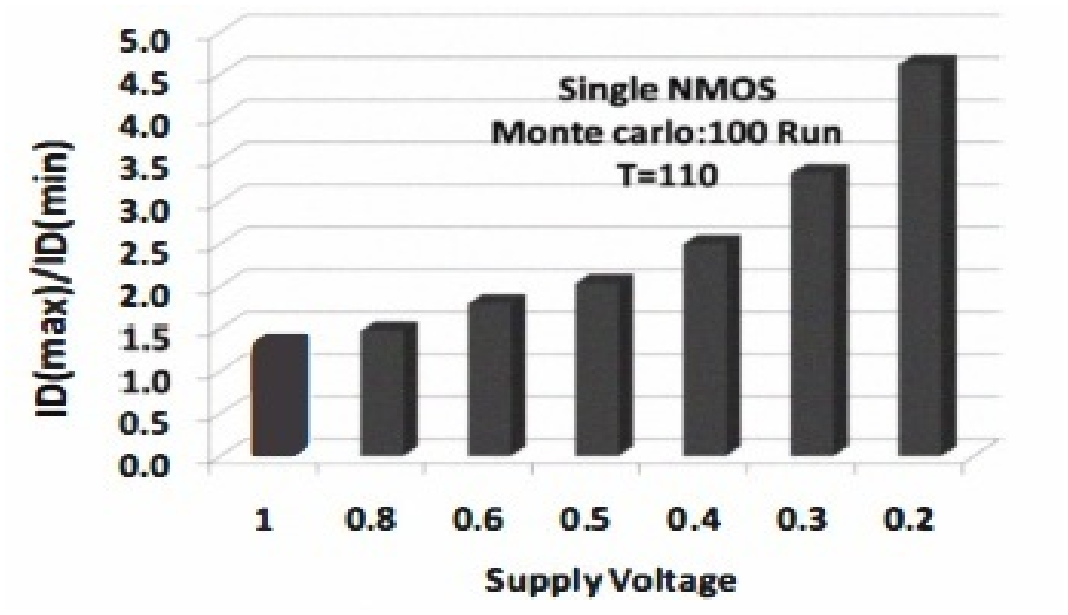


Figure 1.2: ID(on) for different supply voltage (nMOS)

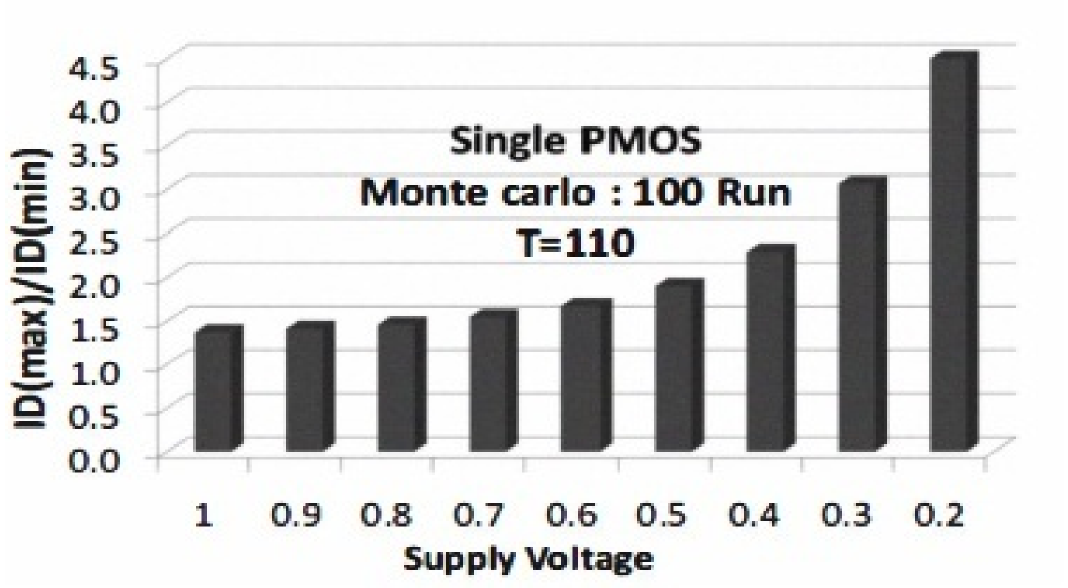


Figure 1.3: ID(off) for different supply voltage (pMOS)

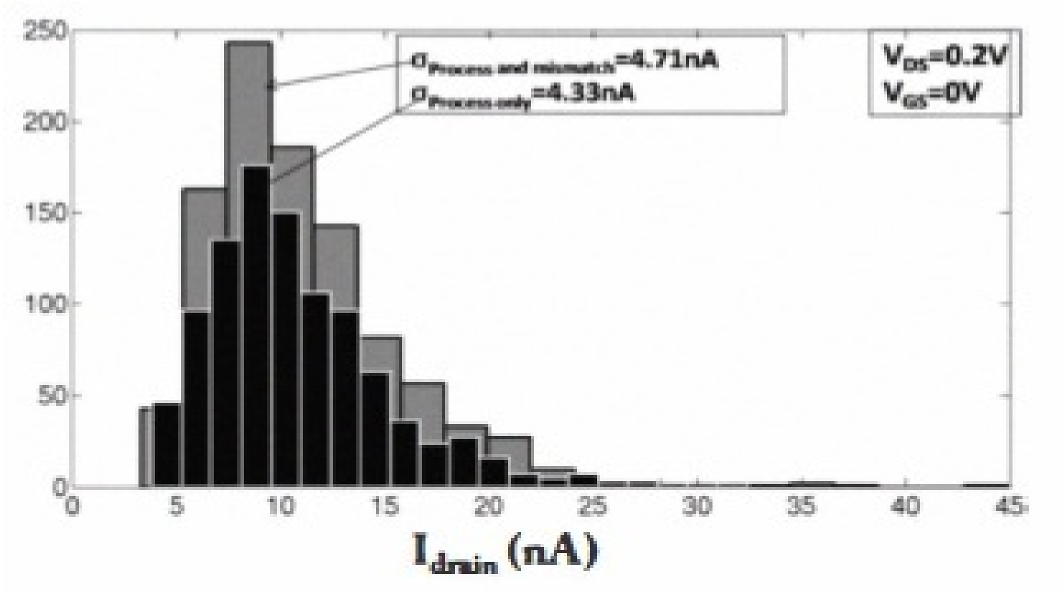


Figure 1.4: Idrain for Process variation

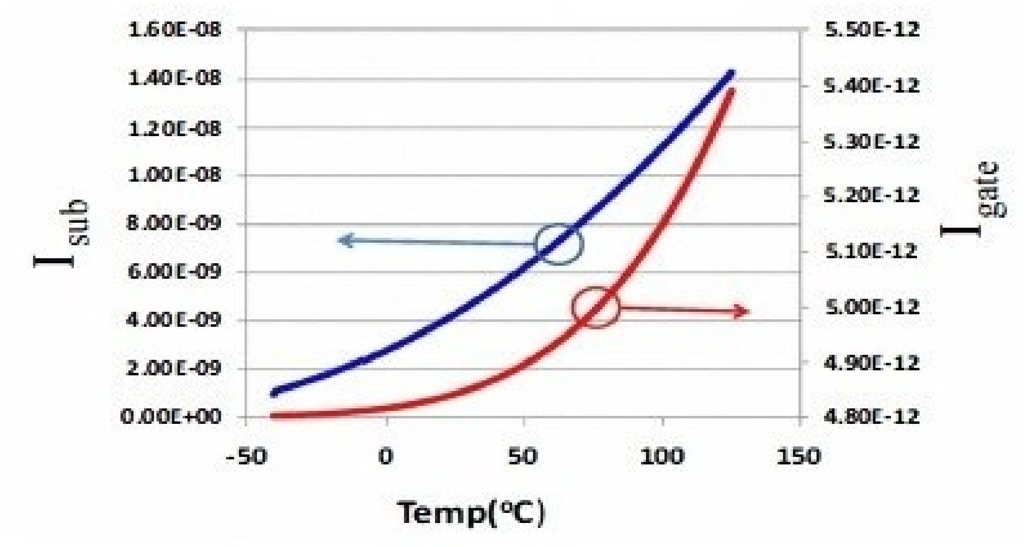


Figure 1.5: ISUB and IDATE w.r.t. Temperature Variation

6T-SRAM cause changes in SRAM reliability. In this case we used a SRAM cell specified for low power supply applications. It means that the SRAM cell is resized to work in lower supply voltages. If corresponding sizing for higher supply voltages is used for lower supply voltages, then these process variations causes some failures. So now reliability is more concerned for sub nm technologies.

1.1.1 Reliability Methodology

Adaptive Body Biasing (ABB)

Bidirectional adaptive body bias (ABB) is used to compensate for die-to-die parameter variations by applying an optimum pMOS and nMOS body bias voltage to each die which maximizes the die frequency subject to a power constraint.

Adaptive Voltage Management (AVM)

Detects temperature accurately by using two ring oscillators, sets supply voltage most suitable with a table look-up method corresponding to the process variation. Also, the AVM can supply the stable voltage with a local shift type regulator even at lower voltage. Thereby, this supply-voltage control system considering PVT variations can control the internal voltage corresponding to process and temperature variations and can realize a wide-operating-margin.

Temperature and Reliability Aware Memory design (TRAM)

Allow designers to examine the effects of frequency, supply voltage, power dissipation and temperature on reliability of memories. Fig 1.6 illustrates the overview of the TRAM which generates reliability values for memory blocks as a function of supply voltage and frequency while taking temperature into account. TRAM uses temperature estimates of the memory under design to generate a better estimate of the expected probability of error in the memory. TRAM computes the reliability with respect to probability of error/failure. Experimental results indicate that the

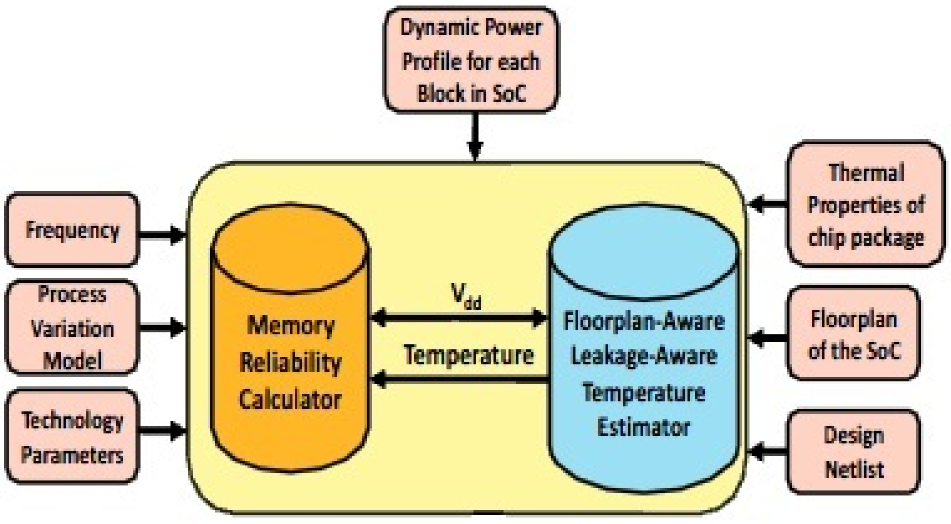

Dynamic Power Profile for each Block in SoC
Frequency
Process Variation Model
Technology Parameters
Memory Reliability Calculator
V_{dd}
Temperature
Floorplan-Aware Leakage-Aware Temperature Estimator
Thermal Properties of chip package
Floorplan of the SoC
Design Netlist

Figure 1.6: Overview of TRAM

probability of error is significantly higher at high temperature. Fig 1.7 indicates that an increase in Vdd fails to reduce probability of error at higher temperature. Increase in Vdd does not guarantee a reduction in probability of error because of the temperature effect. TRAM allows the designer to explore memory Vdd selection for the tradeoff between power and reliability in thermally aware environment and quantifies the effect of temperature on the probability of failures in memories. Thermal Aware Vdd selection using TRAM can reduce the power dissipation by up to 2.5X for a predefined limit of errors.

Dynamic reliability management (DRM) and Dynamic temperature management (DTM)

Proposed to recapture the system performance and reliability margin. Unlike DTM, which tends to avoid temperature violations to minimize the chances of reliability failure, DRM reacts to the current reliability state and then adjusts the voltage/frequency to recover unacceptable reliability loss or use available excess margin

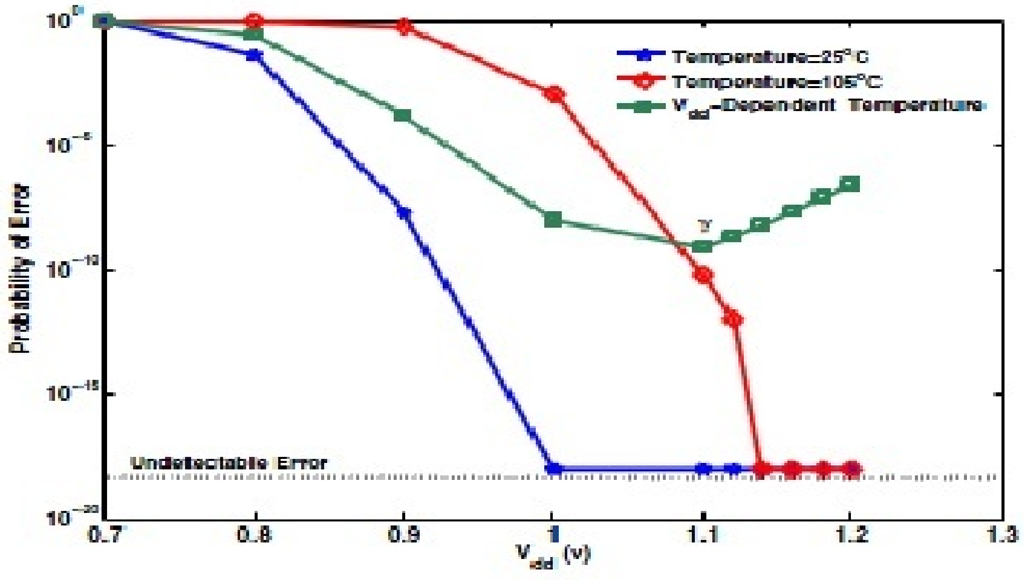


Figure 1.7: Probability of error for different temperature profiles

to boost performance by allowing operation at a higher supply voltage. DRM focuses on the process variation and temperature aware model for the oxide break-down (OBD) failure. Reliability Evaluation and prediction is achieved by the reliability assessment module. This module is software-based and takes into account both process and temperature variabilities for system reliability monitoring Thermal sensors provides accurate inputs to reliability models for projecting the degradation caused by various failure mechanisms. The projected failure probability is used to control the maximum voltage in the DVFS module. Management of reliability is achieved by the Dynamic voltage/frequency scaling (DVFS) module. This Module computes the deviation between the predicted overall lifetime reliability and target reliability. This Deviation is then used in the PID of the DVFS to adjust the system maximum voltage. To decide the reliability, DFM uses the Weibull Probability Distribution. F (t) = 1 - R (t). F (t) is cumulative distribution function of time to breakdown R (t) is Reliability Function.

The on-chip real time reliability monitoring and control can boost supply voltages beyond nominal values, which may enhance system performance by 28.7 perccent

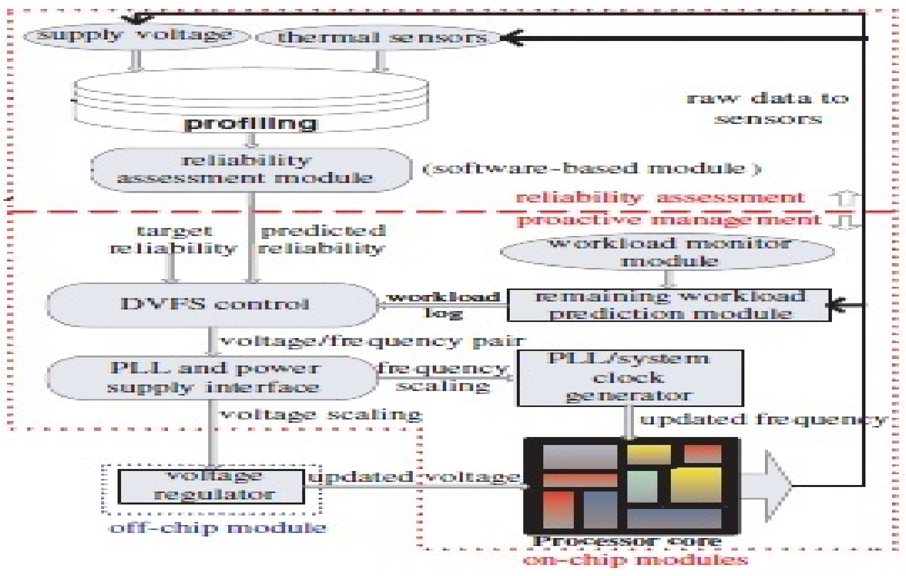


Figure 1.8: Block diagram of DRM

without violating pre-defined reliability constraints.

1.2 RAIMM Proposal

Memories are increasingly used in SoC's. As per new technology trends, SoC's worked at lower operating voltage. So as technology scaling increased, Soft Error Rate sensitively affects the performance of SoC's. Due to this SER, the reliability is going to be decreased. So we are concentrating on the MEMORY reliability of SoC's which are affected by Soft Error Rate (SER). To achieve this, we introduce the concept of RAIMM (Reliable Aware of Intelligence Memory Management). To increase the reliability of SRAM in SoC, we need to split the system memory into multiple blocks and place it in different area on die. There after every memory block connected with its individual PVT sensor for continuously update the values of PVT variation. For particular technology, the PVT variation values are fixed, for checking the reliability, RAIMM compares the values from PVT sensor with the reference values/lookup table to decide the reliability of the Memory. If the values

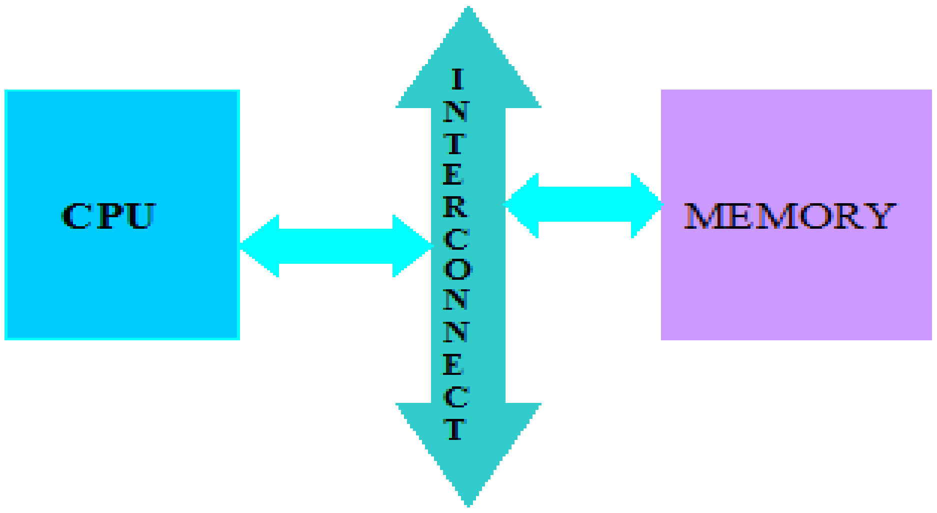


Figure 1.9: Basic model for data transfer between CPU and Memory

of PVT sensors are out of range, than the data is transferred to the memory which one is reliable. To achieve this task, Memory Manager is introduced in the RAIMM. Memory manager decides the reliability from the PVT values. For that, we have to make lookup table for P, V and T values individually. If memory is unreliable, This Memory Manager generates the output as interrupt for the CPU. For specific technology, Reliability ranges for Supply voltage, Temperature and process variation are decided by the statistical analysis. So lookup table/Ranking table compares the PVT sensor's output with these ranges for taking the decision in terms of reliable or unreliable condition. We are taking sum assumption for the sensors o Availability of suitable PVT sensors which can provide a overall memory operating conditions o Sensors are low-consumption so as not to worsen the local conditions

1.3 Features of RAIMM

1. Continuously monitoring process, voltage and temperature variation in the system memory. 2. Monitoring Reliability of the System Memory due to the process,

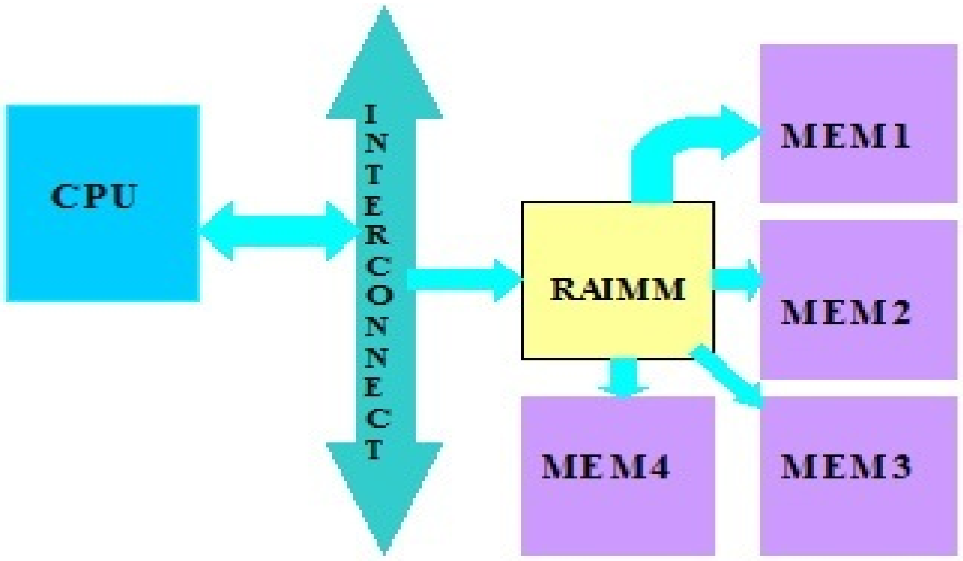


Figure 1.10: RAIMM Concept

voltage, temperature variation. 3. Generating system level interrupts for unreliable memory. 4. Memory profiling based on the accesses done to a particular memory module 5. Ranking of memories based on PVT condition and memory profiling. 6. Remapping of unreliable memory data into reliable memory.

Chapter 2

SystemC

2.1 Introduction

SystemC is a system design language that has evolved in response to a pervasive need for a language that improves overall productivity for the designers of electronic systems. Typically, today's systems contain the Application-specific hardware and software. Furthermore, the hardware and software are usually co-developed on a tight schedule, the systems have tight real-time performance constraints, and thorough functional verification is required to avoid expensive and sometimes catastrophic failures. SystemC offers real productivity gains by letting engineers design both the hardware and software components together as these components would exist on the final system, but at a high level of abstraction. This higher level of abstraction gives the design team a fundamental understanding early in the design process of the intricacies and interactions of the entire system and enables better system trade offs, better and earlier verification, and over all productivity gains through reuse of early system models as executable specifications.

What is SYSTEMC?

SystemC is a C++ class library and methodology that can effectively be used to create a model of a system consisting of software algorithm, hardware architecture

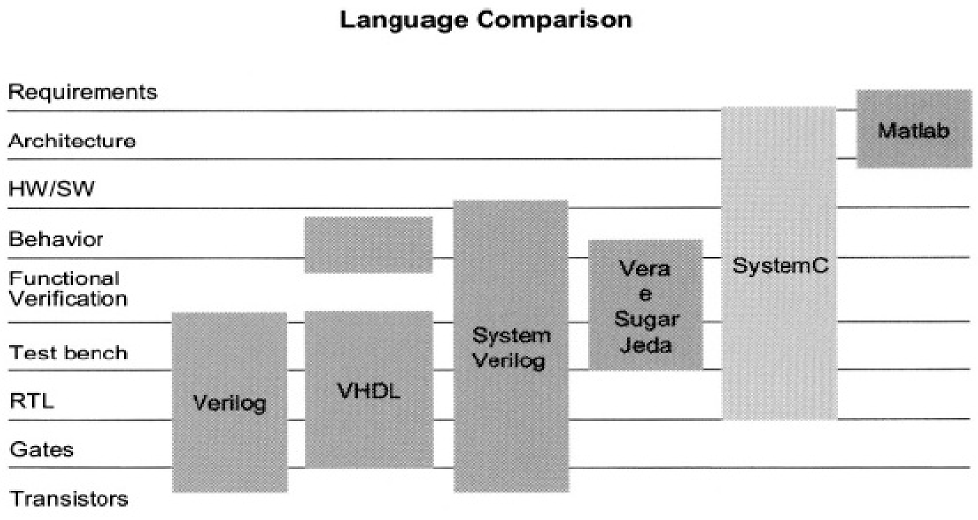


Figure 2.1: Language Comparison

and interfaces of Soc.

Why SYSTEMC?

C/C++ has no notion of time No event sequencing Concurrency But H/W is inherently concurrent H/W Data Types No 'Z'value for tri-state buses

SYSTEMC Class Library use for:

Cycle-Accurate model for Software Algorithm Hardware Architecture Interface of SoC (System-on-Chip) System-level designs Executable Specification

2.2 Language Comparison

SystemC is not a language, but rather a class library within a well established language, C++. SystemC is not a panacea that will solve every design productivity issue. However, when SystemC is coupled with the SystemC Verification Library, it does provide in one language many of the characteristics relevant to system design and modeling tasks that are missing or scattered among the other languages. Additionally, SystemC provides a common language for software and hardware, C++.

Several languages have emerged to address the various aspects of system Design. Although Ada and Java have proven their value, C/C++ is predominately used today for embedded system software. The hardware description languages (HDLs), VHDL and Verilog, are used for simulating and synthesizing digital circuits. Vera and e are the languages of choice for functional verification of complex application-specific integrated circuits (ASICs). SystemVerilog is a new language that evolves the Verilog language to address many hardware-oriented system design issues. Matlab and several other tools and languages such as SPW and System Studio are widely used for capturing system requirements and developing signal processing algorithms.

2.3 SYSTEMC Features

Hardware Type Communication

Signals and protocols

Notion of Time

Time Sequences Operation

Concurrency

Hardware and system are inherently coherent i.e. they operate in parallel

Reactivity

Hardware is inherently reactive, it responds to stimuli and is in constant interaction with its environment

Hardware Data Types

Bit type, Bit Vector Type, signed and unsigned int, fixed point types and multi value logic types. Integrated Simulation Kernel

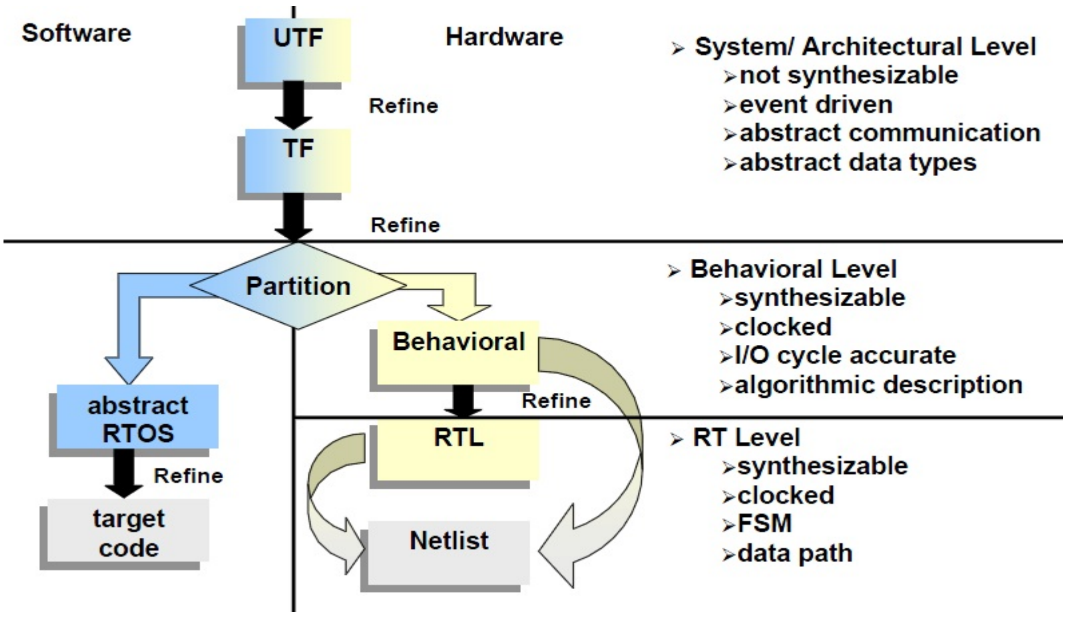


Figure 2.2: SystemC Design flow

2.4 SYSTEMC DESIGN FLOW

Design methods surrounding SystemC are currently maturing and vary widely. In the next few years, these methods will likely settle into a cohesive design methodology (with a few variants among certain industry segments). The resulting methodology will feel similar to the methodologies in use today, but at higher levels of abstraction. To some, the concept of using one unified language for hardware and software development appears revolutionary, but this concept is clearly an evolutionary path for those who frequently work in both domains. Although tools and language constructs exist in SystemC to support Register-transfer-level (RTL) modeling and synthesis, a major reason for using the language is to work at higher abstraction levels than RTL. SystemC's ability to model RTL designs enables support of design blocks generated by higher level (behavioral or graphical entry) synthesis tools or to support legacy design blocks.

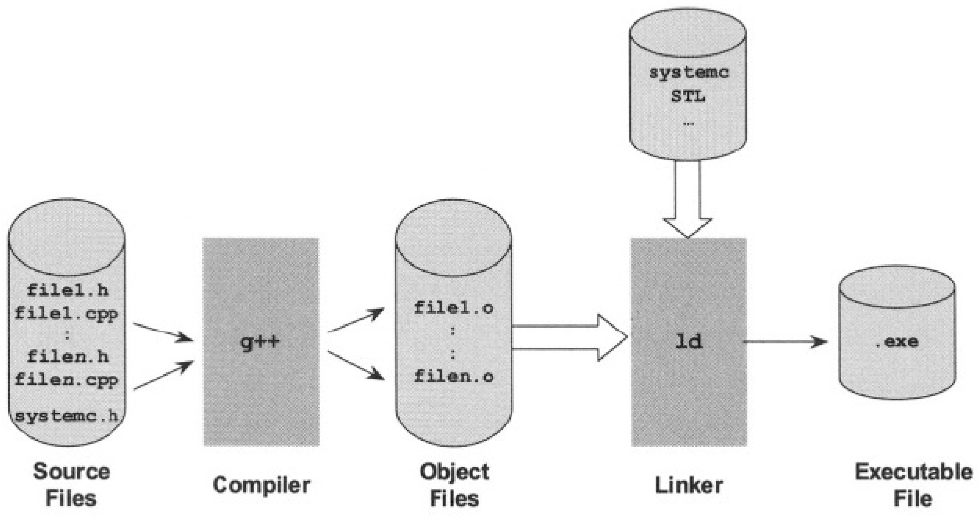


Figure 2.3: SystemC Compilation flow

2.5 SYSTEMC Compilation Flow

The flow for compiling a SystemC program or design is very traditional, and is illustrated in Figure for GNU C++. Most other compilers will be similar. The C++ compiler reads each of the SystemC code file sets separately and creates an object file (usual file extension of .o). Each file set usually consists of two files typically with standard file extensions. We use .h and .cpp as file extensions, since these are the most commonly used in C++. The .h file is generally referred to as the header file and the .cpp file is often called the implementation file. After creating the object files, the compiler (actually the loader or linker) will link your object files and the appropriate object files from the SystemC library (and other libraries such as the standard template library or STL). The resulting file is usually referred to as an executable, and it contains the SystemC simulation kernel and your design functionality. The compiler and linker need to know two special pieces of information. First, the compiler needs to know where the SystemC header files are located (to support include ¡systemc.h¿). Second, the linker needs to know

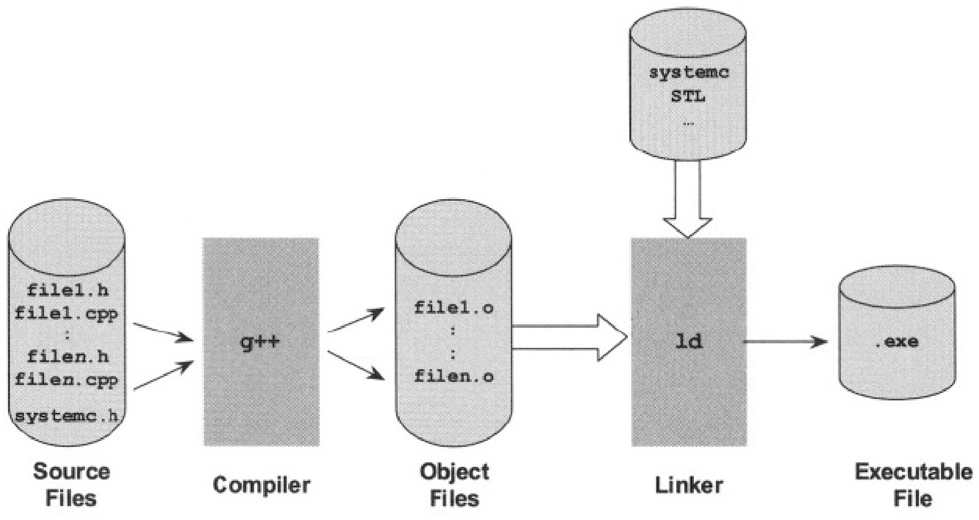


Figure 2.4: System of SystemC

where the compiled SystemC libraries are located. This is typically accomplished by providing an environment variable named SYSTEMC, and ensuring the Makefile.

2.6 System of SystemC

2.6.1 TERMINOLOGY

Modules

Modules are the basic building block within SystemC to partition a design. Modules allow designers to break complex systems into smaller more manageable pieces. Modules help split complex designs among a number of different designers in a design group. Modules allow designers to hide internal d ta representation and algorithms from other modules. This forces designers to use public interfaces to other modules, and the entire system becomes easier to change and easier to maintain. For example, a designer can decide to completely change the internal data representation and implementation of a particular module. However, if the external interface and internal function remain the same, the users of the module do not know that the internals were changed. This allows designers to optimize the design locally.

Processes

Processes are the basic unit of execution within SystemC. Processes are called to emulate the behavior of the target device or system. Some processes behave just like functions, the process is started when called, and returns execution back to the calling mechanism when complete. Other processes are called only once at the beginning of simulation and are either actively executing or suspended waiting for a condition to be true. The condition can be a clock edge or a signal expression or combination of both. Processes have sensitivity lists, i.e. a list of signals that cause the process to be invoked, whenever the value of a signal in this list changes. Processes cause other processes to execute by assigning new values to signals in the sensitivity list of the other process.

Three types of SystemC processes are available:

o Methods When events (value changes) occur on signals that a process is sensitive to, the process executes. A method executes and returns control back to the simulation kernel.

o Threads Thread Process can be suspended and reactivated. The Thread Process can contain wait() functions that suspend process execution until an event occurs on one of the signals the process is sensitive to. An event will reactivate the thread process from the statement the process was last suspended. The process will continue to execute until the next wait().

o Clocked Threads Clocked threads can be used to create implicit state machines within design descriptions. An implicit state machine is one where the states of the system are not explicitly defined. Instead the states are described by sets of statements with wait() function calls between them. Clocked Thread Processes are a special case of a Thread Process. Clocked Thread Processes help designers describe their design for better synthesis results. Clocked Thread Processes are only triggered on one edge of one clock, which matches the way hardware is typically implemented with synthesis tools.

Ports

Ports of a module are the external interface that pass information to and from a module, and trigger actions within the module. Signals create connections between module ports allowing modules to communicate.

A port can have three different modes of operation.

o Input o Output o InOut

Events

Flexible, low-level synchronization primitive Used to construct other forms of synchronization

Channels

A container class for communication and synchronization They implement one or more interfaces

Interface

Specify a set of access methods to the channel

2.7 Simulation and Debugging Using SystemC

The scheduler in SystemC executes the following steps during simulation.

1. All clock signals that change their value at the current time are assigned their new values.

2. All SCMETHOD or SCTHREAD processes with inputs that have changed are executed. The entire body of SCMETHOD function processes is executed, while

SCTHREAD processes are executed until the next wait() statement suspends execution of the process. SCMETHOD or SCTHREAD processes are not executed in a fixed order.

3. All SCCTHREAD processes that are triggered have their outputs updated, and they are saved in a queue to be executed later in step 5. All outputs of SCMETHOD/SCTHREAD processes that were executed in step 1 are also updated.

4. Steps 2 and 3 are repeated until no signal changes its value.

5. All SCCTHREAD processes that were triggered and queued in step 3 are executed. There is no fixed execution order of these processes. Their outputs are updated at the next active edge (when step 3 is executed), and therefore are saved internally.

6. Simulation time is advanced to the next clock edge and the scheduler goes back to step 1.

Chapter 3

TLM (Transaction Level Modeling)

3.1 Introduction

In order to handle the ever increasing complexity of system on- chips (SoCs) and time-to-market pressures, the design abstraction has been raised to the system level in order to increase design productivity. This higher level of abstraction generated large interest in transaction-level modeling, synthesis, and verification.

In a transaction-level model (TLM), the details of communication among computation components are separated from the details of computation components. Communication is modeled by channels, while transaction requests take place by calling interface functions of these channel models.

Unnecessary details of communication and computation are hidden in a TLM and may be added later. TLMs speed up simulation and allow exploring and validating design alternatives at the higher level of abstraction.

Transaction-level modeling is an emerging concept without precise definitions. A working group of Open SystemC Initiative (OSCI) is currently defining a set of terminology for TLM and will eventually develop TLM standards. In reality, when engineers talk of TLM, they are probably talking about one or more of four different modeling styles that are discussed in the following section.

The underlying concept of TLM is to model only the level of detail that is needed by the engineers developing the system components and subsystem for a particular task in the development process. By modeling only the necessary details, design teams can realize huge gains in modeling speed thus enabling a new methodology. At this level, changes are relatively easy because the development team has not yet painted itself into a corner with low-level details such as a parallel bus implementation versus a serial bus implementation.

Using TLMs makes tasks usually reserved for hardware implementations practical to run on a model early in the system development process. TLM is a concept independent of language. However, to implement and refine TLM models, it is helpful to have a language like SystemC whose features support independent refinement of functionality and communication that is crucial to efficient TLM development.

In the transaction-level model (TLM), the details of communication among computation components are separated from the details of the implementation of computation components. Communication is modeled as channels and transaction requests take place by calling interface functions of these channel models. Unnecessary details of communication and computation are hidden in the TLM and may be worked out later. Transaction-level modeling enables speeding up simulation time, exploring and validating implementation alternatives at the higher level of abstraction. However, the current definition of TLM is too general and not well defined. Without clear definition of TLMs, not only the predefined TLMs cannot be easily reused, but also the usages of TLMs in the existing design practices, namely system synthesis, platform-based design, and component based design, cannot be systematically developed. Consequently, the advantages of TLM don't effectively benefit the users of the existing design practices. The TLM can be used in the existing three design practices:

- System synthesis (top down approach),

- Platform-based design (meet-in-the-middle approach),
- Component-based design (bottom-up approach).

System synthesis, starts design from the system behavior representing the design's functionality, generates system architecture from the behavior, and gradually reaches the implementation model by adding implementation details.

With comparison to the system synthesis, Platform-based design maps the system behavior to the predefined system architecture, rather than generating the architecture from the behavior. An example of platform-based design approach is to use VCC for design estimation/exploration and Co-Ware N2C for interface synthesis.

Unlike above two approaches, in order to produce the predefined platform, Component-based design assembles the existing computation components by inserting wrappers among them. Component-based design focuses on component reuse and wrapper generation. All of above three design practices fully/ partly cover design domain from the system behavior to the detailed system implementation, which exhibits great potential of employing TLMs

3.2 TLM BASED METHODOLOGY

- If the proper design language and techniques are used consistently throughout the flow, then the SAM can be reused and refined to develop the TLM.

1. Refinement of implementation features such as HW/SW partitioning; HW partitioning among ASICs, FPGAs, and boards; bus architecture exploration; coprocessor definition or selection; and many more 2. Development platform for system software 3. "Golden Model" for the hardware functional verification 4. Hardware micro-architecture exploration and a basis for developing Detailed hardware specifications In the near future, if EDA tools mature sufficiently, the TLM code may be refined to a behavioral synthesis model and be automatically converted to hardware from a higher-level abstraction than the current RTL synthesis flows. Today, the

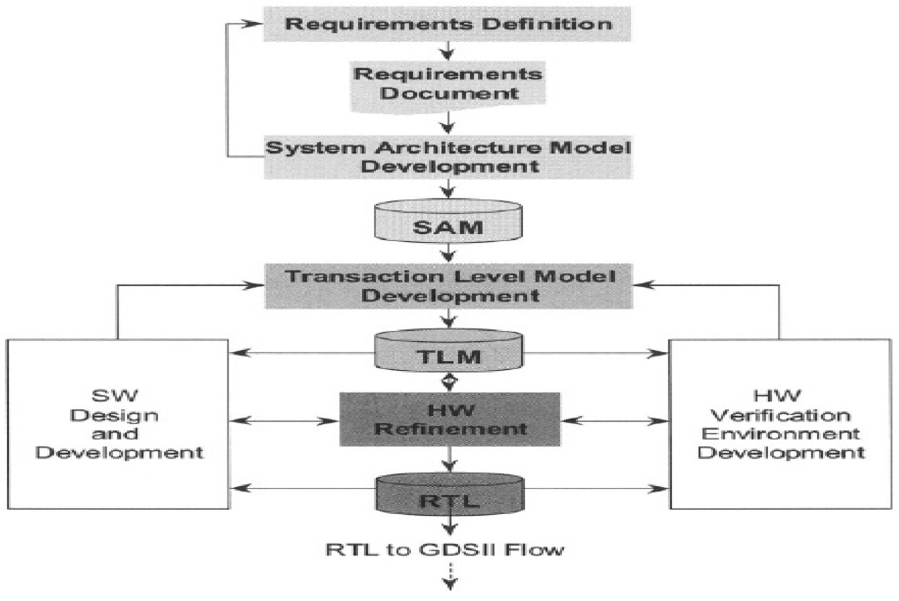

Requirements Definition
Requirements Document
System Architecture Model Development
SAM
Transaction Level Model Development
TLM
SW Design and Development
HW Refinement
HW Verification Environment Development
RTL
RTL to GDSII Flow

Figure 3.1: TLM based methodology

hardware refinement is likely done through a traditional paper specification and RTL development techniques, although the functional verification can now be performed via the TLM as outlined later in this chapter. At first, development of the TLM appears to be an unnecessary task.

The TLM creates benefits including: 1) Earlier software development 2) Earlier and better hardware functional verification test bench 3) Creates a clear and unbroken path from customer requirements to Detailed hardware and software specifications

The TLM has several goals

3.3 TLM TERMINOLOGY

TLM modeling infrastructure defines SoC design as a project that refers to IP models. Each project is hierarchically organized in several directories listed below:

- Component. Contain all TLM component models implemented for a given project. Each component holds its own directory, which can be hierarchically organized and may have inter-project dependencies.

- Platform. Represent the location where components and/or test benches are instantiated for creating the top netlist of a design.

- Software. Keep embedded software codes to be run on the TLM simulation platform of a given project.

- Devkit. Contain simulation facilities such as transaction recorders or helper functions that are not directly parts of TLM models, but useful for the project in certain perspectives or context.

- External. Store definitions of external libraries and tools required by the TLM simulation. These definitions may not strictly be related to TLM models. For example, a library containing a processor model is referred by a TLM platform to instantiate an ISS yet it remains an external item

3.4 MODELING API

The TLM uses layered approach, which separates a functional IP model clearly from its communication interfaces that exchange data with other models. The TLM layered approach offers high-level primitives tailored for the specific modeling needs of modeling engineers. In many cases of system modeling, several communication protocols of different semantics and content are required. Some cases may only require a simple communication protocol that requests a point-to-point connection to pass data from an initiator to a target. On the other hand, certain cases may need a more complex protocol that supports the following features:

1. Complex data structures with address, data, and byte-enable information. 2. Transaction routing onto the communication medium. 3. score-boarding capabilities for verification purposes.

To address such varying modeling requirements, the TLM layered approach has defined three complementary layers listed below:

1. Core TLM interface layer. 2. Protocol layer. 3. IP layer.

3.4.1 Core TLM Interface Layer

The core TLM interface layer is the foundation of TLM methodology. It defines a transactional level interface, and declares the related transactional level ports accordingly. Indeed, such layer is the minimum interface definition required for modeling a SoC at the transactional level. It gets ready a communication API that is capable of transporting a transaction from an initiator to a target module.

3.4.2 Protocol Layer

Various communication protocols can be defined on top of the core TLM interface layer. These protocols rely fully on the core TLM interface to transfer a transaction between two different points in a system. The semantics of the transactional transfer are refined by these protocols in terms of transaction payload and blocking/non-blocking transfer.

TLM modules communicate through ports. Initiator ports are defined as a specialization of sc_port on the master side, while target ports are defined as a specialization of sc_export on the slave side. For each TLM protocol, a set of methods is defined. These methods are also known as convenience functions. They are specified in the form of interfaces in the C++ abstract class.

- Initiator Port.

For initiator ports, the implementation of these methods is 1. Core TLM interface layer. Done once and for all by the protocol developer. It allows translating user-level operations (e.g. Read or write) into the appropriate core TLM interface calls.

- Target Module.

For target modules, the core TLM interface is implemented in a target base class that is done once and for all by the protocol developer. This implementation calls the methods of the protocol interface implemented in each target module. Therefore, the IP developer is responsible for the right implementation of the convenience functions. Since target modules inherit from the protocol interface, the compiler will definitely check for the existence of the implementation of these convenience functions. The examples of the TLM protocols developed by our efforts include: 1. TLM_TAC

TAC stands for Transaction Accurate Communication. It specifies a very abstract bus model with a blocking read/write API. It uses the bidirectional blocking core TLM interface, tlm_transport_if.

2. TLM_STBUS. This is a model of the STBus4 protocol at the packet level. This protocol definition includes the representation of all opcodes and the associated information of STBus. It uses the unidirectional blocking core TLM interface, tlm_blocking_put_if, and the non-blocking core TLM interface, tlm_nonblocking_put_if.

3. TLM_SYNCHRO. A TLM protocol particularly developed for purely functional simulations. System synchronizations are modeled without using sc signal. It uses the unidirectional blocking core TLM interface, tlm_blocking_put_if.

3.4.3 IP Layer

TLM IPs is modeled on top of a TLM protocol layer as functional modules. Communications between TLM IPs are established through the communication API defined in the protocol interface. A given TLM IP can instantiate as many ports as required as long as it is in accordance with the underlying protocol. Several ports based on the same protocol could be instantiated by an IP, for instance, a dual port memory IP. Each IP is aware of a set of protocols that it can support. An example of an IP supporting multiple protocols is the protocol converter. This IP receives transactions from a sub-system built on a particular protocol. It then converts the received transactions into another protocol, and reinitiates them to another sub-

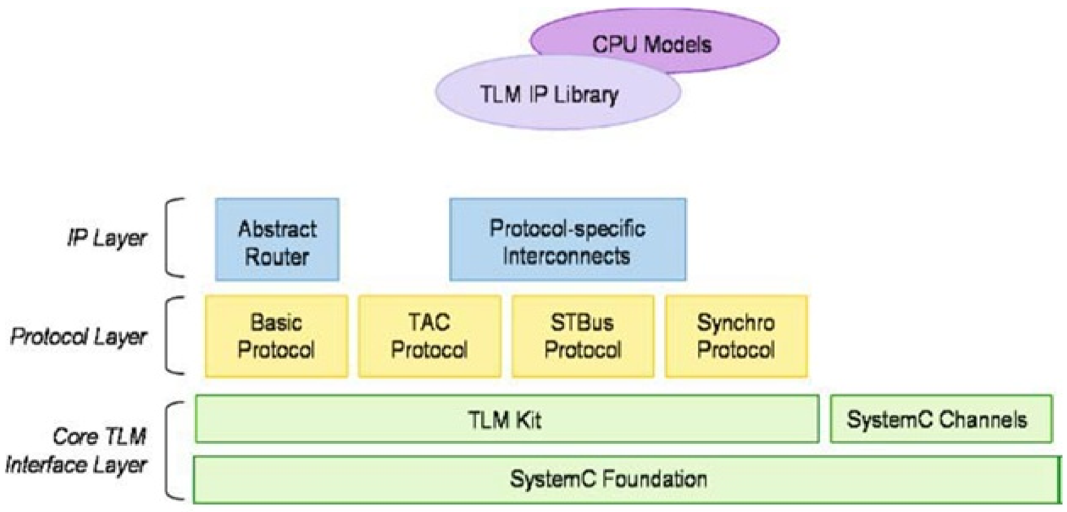


Figure 3.2

system as new transactions based on this protocol. Figure 3.2 illustrates the idea of the TLM layered approach in SystemC.

3.4.4 Other Terminology

1) Nonblocking: Means function implementations can never call wait() 2) Blocking: Means function implementations might call wait() 3) Unidirectional: data transferred in one direction 4) Bidirectional: data transferred in two directions 5) Write/Peek: Write potentially overwrites data and can never block. Peek reads most recent valid value. Write/Peek is similar to write/read to a variable or signal 6) Put/Get: Put queues data. Get consumes data. Put/Get is similar to writing/reading from a FIFO 7) Pop: A pop is equivalent to a get in which the data returned is simply ignored

3.5 INTERCONNECT MODELING

An interconnect or channel is a structure responsible for establishing communications among TLM modules. It can be modeled in various manners at the transactional level depending on the expected accuracy as well as the model used on the final platform. The minimum features of a TLM interconnect are the abilities to:

1. Decode addresses. 2. Route a transaction from an initiator to a target module.

This approach is sufficient to model untimed modules for fast functional simulations during the embedded software development. It is obviously insufficient for an architectural analysis. In that case, the topology (e.g. nodes and links between nodes of a network-on-chip) of the interconnect module will have to be modeled along with its arbitration policy and potential delays.

3.5.1 Interconnect Structure

In the untimed TLM, an interconnect is modeled as a hierarchical channel with communication ports and the implementation of one of the core TLM interfaces. Depending on the communication purposes, communication ports of a channel can be arranged differently as:

- Simple Transaction Router

If a channel simply routes transactions without considering arbitration, a single initiator port and a single target port will be sufficient. Each of these communication ports can be bound to one or more IP ports. Indeed, an initiator port and a target port of a router are centralized points to receive and transmit transactions collectively from all IP ports involved in the system communication. Figure illustrates the interconnect structure of a transaction router.

- Arbiter Channel

A sophisticated channel with arbitration ability is typically implemented for cases where the system topology and port notions are required for handling arbitration.

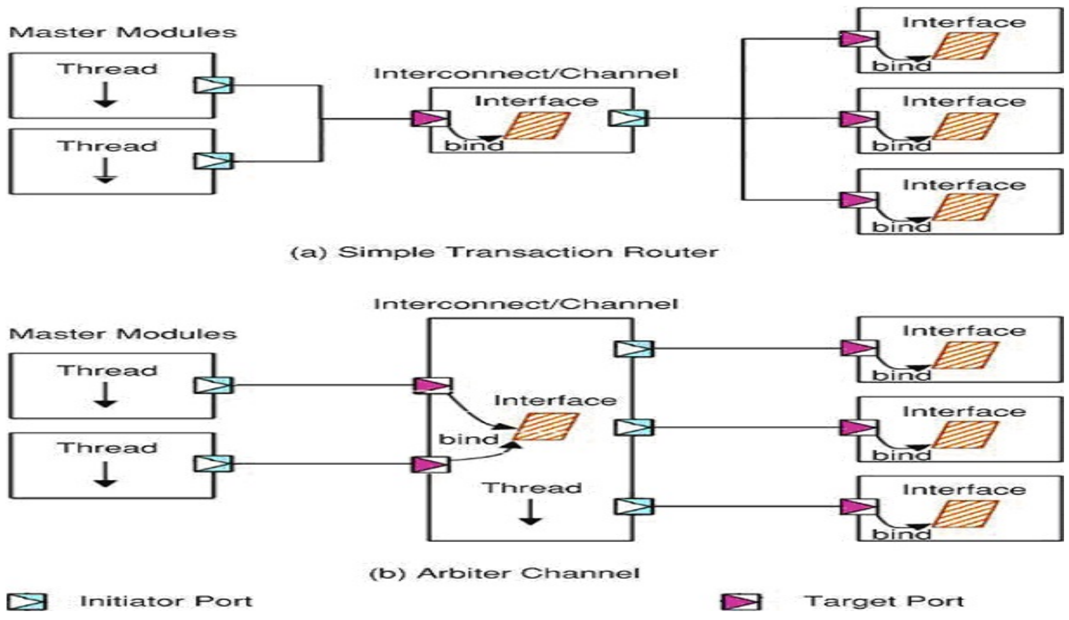


Figure 3.3: System level interconnect using TLM

In an arbiter channel, an initiator port or a target port will be particularly assigned for each IP port involved in the system Communication. The interconnect structure of an arbiter is depicted in Fig 3.3

Chapter 4

ARM926 BASED PLATFORM

4.1 Introduction

This System platform is a completely abstracted platform and different IPs is modeled in SystemC at higher abstraction level either transaction level models or timing annotated models. In this scenario, the platform gives a glimpse of platform SoC and a complete hardware + software prototype is available. There is an operating system booted on the platform and it permits to develop/plug and integrate device drivers for IPs.

4.2 ARM Based Platform for IP development-/verification

The ARM platform can be used to demonstrate platform architecture as well as illustrates the possibility to deploy the platform for RTL development/verification. The intent of the platform is to provide a preliminary platform with minimum required components which permits to run test software and permits to plug the RTL IP, corresponding verification IP and facilitates the RTL IP verification.

Such a platform could be a demo platform as well as it could help customer to do early evaluation of System with their own IPs. Customer may develop an abstracted IP model and perform all kind of analysis, start developing test benches for further phases of development. This could be a suitable platform for customer to perform hardware software partitioning and make explorations pertaining to make a better decision on what could go in software and the hardware portion of the IP. The ARM926 processor based system has been shown in Fig 17.

The approach helps for following of the activities in design cycle -

- Such platform which could be used for developing test vectors with an abstracted IP prototype. These test vectors could be further used for RTL verification with references already available.

- This platform could be considered a reference platform for RTL development and seamlessly permits to write small s/w 'c/c++' test benches.

- A similar platform could be used for HW/SW partitioning. Facilitated with some tools available, it could be used to gather performance statistics in order to partition Software functionalities versus Hardware.

- Start early S/W driver development and perform S/W debugging. In this platform the S/W debugging capabilities are very good as well, it permits to debug the hardware also to a great extent.

- A setup could be established to have Instruction set simulator or Cycle accurate simulators plugged in the same platform and run the software as if running on the ARM boards.

The added advantage of such platforms are better simulation performance, good debugging capabilities with test benches as it permits to write C test benches. This platform will be highly recommended to customers who are interested in developing IPs which might have software along with hardware and Another platform in parallel will permit to perform various activities e.g. IP prototyping at

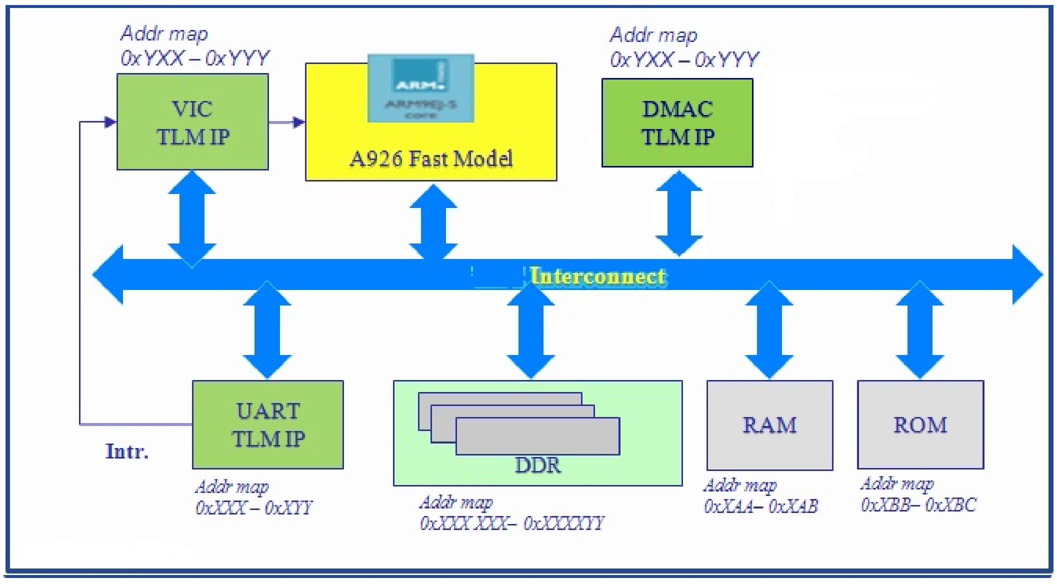


Figure 4.1: ARM926e processor based virtual platform

higher abstraction level, hardware software partitioning or developing a complete system prototype with their own IP. In these platforms there is always possibility of adding/removing the ip models very easily. In fact, we can start with minimal ip models needed to build the platform and could identify sequentially the IPs needed to integrate.

4.3 Components of the Platform

The current platform is based on TLM protocol. The available TLM IP models are

ARM926 o ARM926 core for running C software and interrupt handling.

DMAC PL080 o TLM TAC IP Model available. Full functionality available.

VIC o TLM TAC IP Model available. Full functionality available.

UART o TLM TAC IP Model available. Full functionality available

4.3.1 Arm926 - Overview

The ARM926 processor is a member of the ARM9 family of general-purpose microprocessors. The ARM926 processor is targeted at multi-tasking applications where full memory management, high performance, low die size, and low power are all important. The ARM926 processor supports the 32-bit ARM and 16-bit Thumb instruction sets, enabling you to trade off between high performance and high code density. The ARM926 processor includes features for efficient execution of Java byte codes, providing Java performance similar to JIT, but without the associated code overhead. The ARM926 processor supports the ARM debug architecture and includes logic to assist in both hardware and software debug. The ARM926 processor has a Harvard cached architecture and provides a complete high-performance processor subsystem, including:

- An ARM9 integer core
- A Memory Management Unit (MMU)
- Separate instruction and data AMBA AHB bus interfaces
- Separate instruction and data TCM interfaces.

The ARM926 processor provides support for external coprocessors enabling floating-point or other application-specific hardware acceleration to be added. The ARM926 processor implements ARM architecture v5TEJ. The ARM926 processor is a synthesizable macrocell. This means that you can optimize the macrocell for a particular target library, and that you can configure the memory system to suit your target application. You can individually configure the cache sizes to be any power of two between 4KB and 128KB. The tightly-coupled instruction and data memories are instantiated externally to the ARM926 macrocell, providing you with the flexibility of optimizing the memory subsystem for performance, power, and particular RAM type. The TCM interfaces enable nonzero wait state memory to be attached, in addition to providing a mechanism for supporting DMA.

Main features are:

- fmax 333 MHz (downward scalable).
- MMU.
- 16 Kbyte of instruction cache + 16 Kbyte of data cache.
- TCM memory available through customization.
- AMBA bus interface.
- Coprocessor interface through customization.
- JTAG and ETM9 (embedded trace macro-cell) for debug, large size version.

Vectored Interrupt Control (VIC_PL190)

Acting as an interrupt controller, the VIC determines the source that is requesting service and where its interrupt service routine (ISR) is loaded, doing that in hardware. In particular, the VIC supplies the starting address, or vector address, of the ISR corresponding to the highest priority requesting interrupt source.

Main features of the VIC are:

- Support for 32 standard interrupt sources.
- Generation of both fast interrupt request (FIQ) and interrupt ReQuest (IRQ), according to ARM system operation. IRQ is used for general interrupts, whereas FIQ is intended for fast, low-latency interrupt handling
- Support for 16 vectored interrupts (IRQ only).
- Hardware interrupt priority, where FIQ interrupt has the highest priority, followed by vectored IRQ interrupts (from vector 0 to vector 15), and then non-vectored IRQ interrupts with the lowest priority.
- Interrupt masking.

- Interrupts request status and raw interrupt status (prior to masking).
- Software interrupts generation.
- An AHB slave to connect to the CPU.

The interrupt inputs must be level sensitive, active HIGH, and held asserted until the interrupt service routine clears the interrupt. Edge-triggered interrupts are not compatible. The interrupt inputs do not have to be synchronous to HCLK

4.3.2 Universal asynchronous receiver/transmitter (UART)

A UART is responsible for performing the main task in serial communications with computers. The device changes incoming parallel information into serial data that can be sent on a communication line. The UART performs all tasks required for the communication, including timing, parity checking, and so forth. The UART is an APB module in the power bus subsystem; it supports standard asynchronous communication bits (start, stop, and parity), which are added prior to transmission and removed on reception. The UART can support serial data baud rate, dc up to UARTCLK_max_freq/16. It supports modem control functions CTS, DCD, DSR, RTS, DTR and RI.

If the interrupt is not currently being serviced, the highest priority request generates an IRQ interrupt. FIQ interrupts have the highest priority, followed by vectored interrupts (0-15) and, finally, none vectored interrupts.

The UART performs:

o Serial-to-parallel conversion on data received from a peripheral device o Parallel-to-serial conversion on data transmitted to the peripheral device.

The CPU reads and writes data and control/status information through the AMBA APB interface. The transmit and receive paths are buffered with internal FIFO memories enabling up to 16-bytes to be stored independently in both transmit and receive modes.

The UART can generate:

- Individually-maskable interrupts from the receive (including timeout), transmit, modem status and error conditions
- A single combined interrupt so that the output is asserted if any of the individual interrupts are asserted, and unmasked
- DMA request signals for interfacing with a Direct Memory Access (DMA) controller.

If a framing, parity, or break error occurs during reception, the appropriate error bit is set, and is stored in the FIFO. If an overrun condition occurs, the overrun register bit is set immediately and FIFO data is prevented from being overwritten. You can program the FIFOs to be 1-byte deep providing a conventional double-buffered UART interface. The modem status input signals Clear to Send (CTS), Data Carrier Detect (DCD), Data Set Ready (DSR), and Ring Indicator (RI) are supported. The output modem control lines, Request to Send (RTS), and Data Terminal Ready (DTR) are also supported.

Main features

- Separate 16x8 transmit and 16x12 receive first-in, first-out memory buffers (FIFOs)
- Programmable FIFO disabling for 1-byte depth
- Programmable baud rate generator
- Independent masking of transmit FIFO, receive FIFO, receive timeout, modem status, and error condition interrupts
- Support for direct memory access (DMA)
- False start bit detection

- Line break generation and detection
- Programmable hardware flow control
- Fully-programmable serial interface characteristics
- Data can be 5, 6, 7, or 8 bits.
- even, odd, stick, or no-parity bit generation and detection
- 1 or 2 stop bit generation

4.3.3 DMA Controller

The DMAC enables memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral transactions. Each DMA stream provides unidirectional serial DMA transfers for a single source and destination. For example, a bidirectional port requires one stream for transmit and one for receive. The source and destination areas can each be either a memory region or a peripheral, and you can access them through the same AHB master, or one area by each master.

Main features of the DMAC are:

- Each DMA channel can support a unidirectional transfer, with internal four-word FIFO per channel.
- 16 peripheral DMA request lines, where each peripheral connected to the DMAC can assert either a single DMA request or a DMA request (with programmable size to increase data transfer effectiveness).
- Hardware priority (0 the highest to 7 the lowest) for each DMA channel to manage requests from more than 1 channel at the same time.
- scatter or gather DMA support through the use of linked lists.
- An AHB slave acting as programming interface to access to DMA control registers:

- Two AHB masters for data transfer following a DMA request.
- 32-bit AHB master bus width, supporting 8, 16, and 32-bit wide transactions.
- support both big-endian and little-endian (default on DMAC reset).
- separate and combined DMA error and DMA count interrupt requests, with three interrupt request signals (DMACINTTC, DMACINTERR and DMACINTR).
- interrupt masking and raw interrupt status (prior to masking).

4.4 Working with Arm926 Environment

RealView Debugger

RVdebugger is Arm model debugger front end for software debugging. The rvdebugger is used to launch the ARM based platform. An input file can be programmed for rvdebugger which will run the simulations from platform environment and let user perform the configurations needed from the software side.

4.4.1 Debug Interfaces

Hardware Debug Interfaces

The RealView ICE Debug Interface is available if installed, which enables you to debug hardware through a RealView ICE unit.

Software Debug Interfaces

The following software Debug Interfaces are available:

- RealView ARMulator ISS, to connect to RealView ARMulator ISS (RVISS) simulated processors. RVISS is always installed with RealView Debugger.
- Instruction Set System Model (ISSM), to connect to simulated Cortex processors. ISSM is always installed with RealView Debugger.

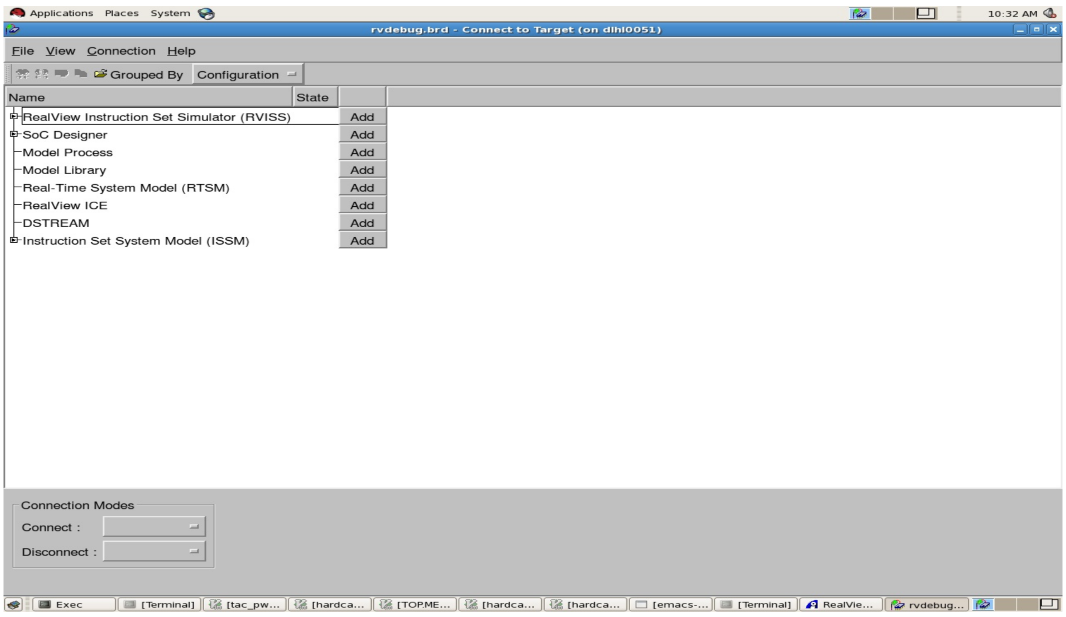


Figure 4.2: rvDebuger interface with arm processor

- SoC Designer, to connect to RealView SoC Designer models.
- Real-Time System Model (RTSM), to connect to models of real-time systems.
- The fig 18 shows the debug interfaces

4.4.2 Creating a Debug Configuration

To add a new Debug Configuration:

- Select Connect to Target... from the Target menu to display the Connect to Target window.
- Select Configuration from the Grouped by drop-down list.
- Click the Add button for the required Debug Interface. The configuration dialog box for the selected Debug Interface is displayed.

- Configure the Debug Interface targets as required: o RealView Instruction Set Simulator (RVISS) o Instruction Set System Model (ISSM) o Real-Time System Model (RTSM) o RealView ICE o SoC Designer.

- After you have configured and saved the Debug Interface configuration, a new Debug Configuration is added to that Debug Interface. RealView Debugger assigns a default name to the Debug Configuration according to the Target Interface.

4.4.3 Connecting to a target

Connecting to a RealView SoC Designer target

To connect to a model created with RealView SoC Designer:

- Open the required model in RealView SoC Designer Simulator.

- Start RealView Debugger.

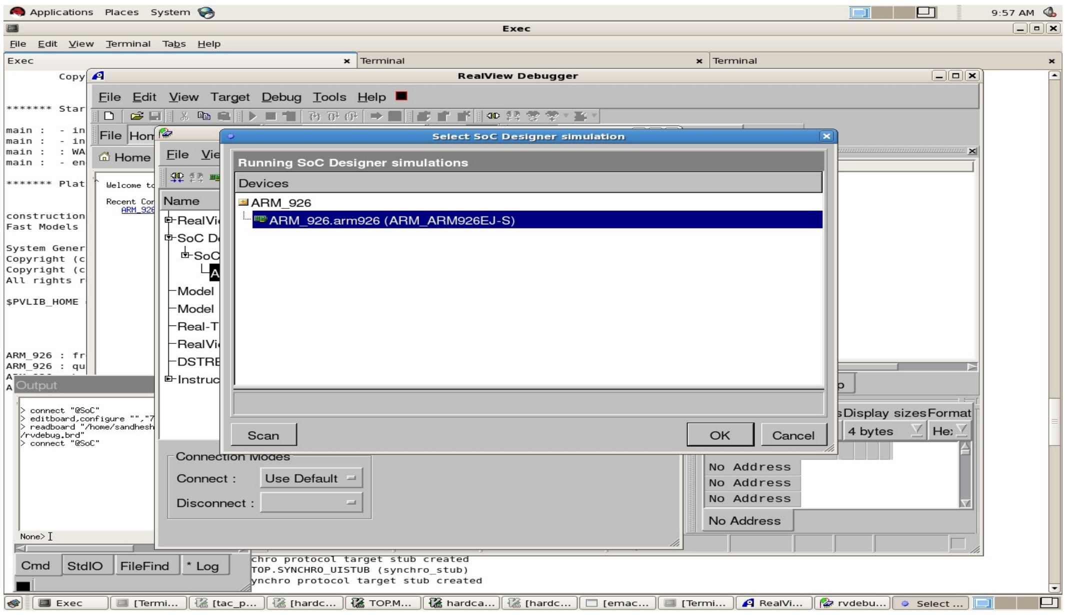
- In RealView Debugger, select Connect to Target... from the Target menu to display the Connect to Target window as shown in figure.

- Select Configuration from the Grouped by drop-down list as shown in fig 19 and 20.

- Expand the SoC Designer Debug Interface.

- Expand the required SoC Designer Debug Configuration. SoC Designer connections are named model.target[n] where: model is the name of the SoC Designer model. Target Identifies the target processor being modeled.

- Double-click on the target to connect.


Applications Places System
9:57 AM
Exec
File Edit View Terminal Tabs Help
Exec
Terminal
Terminal
RealView Debugger
File Edit View Target Debug Tools Help
Select SoC Designer simulation
Scan
OK
Cancel
Cmd StdIO FileFind * Log
Exec [Termi... [tac_p... [hardc... TOP.M... hardca... [hardc... [emac... [Termi... RealVi... rvdebu... Select ...

Running SoC Designer simulations

Devices

- ARM_926
  - ARM_926.arm926 (ARM_ARM926EJ-S)

Connection Modes

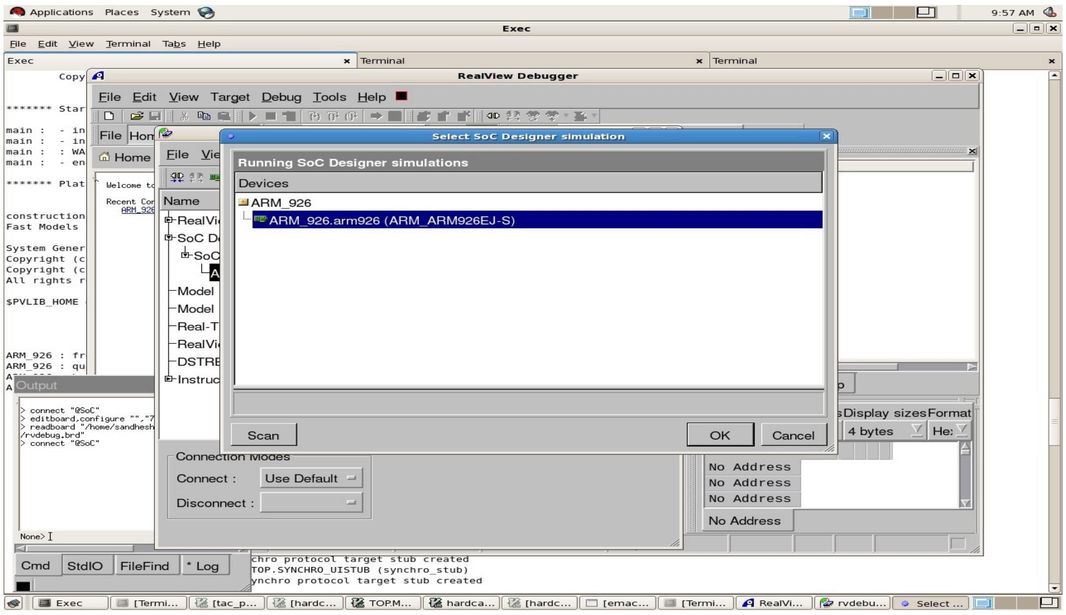
Connect : Use Default

Disconnect :

```
> connect "@SoC"
> editboard,configure "","7
> readboard "/home/sandhesh
/rvdebug.brd"
> connect "@SoC"
```

```
chro protocol target stub created
TOP.SYNCHRO_UISTUB (synchro_stub)
ynchro protocol target stub created
```

Figure 4.3: Configuring arm processor as a target


Running SoC Designer simulations

Devices

- ARM_926
  - ARM_926.arm926 (ARM_ARM926EJ-S)

Connection Modes

Connect : Use Default

Disconnect :

```
> connect "@SoC"
> editboard,configure "","7
> readboard "/home/sandhesh
/rvdebug.brd"
> connect "@SoC"
```

```
chro protocol target stub created
TOP.SYNCHRO_UISTUB (synchro_stub)
ynchro protocol target stub created
```

Figure 4.4: Connecting with arm

Chapter 5

RAIMM Implementation

5.1 About the RAIMM

The RAIMM (Reliability Aware Intelligent Memory Manager) is an intelligent memory manager implemented as a digital IP.

5.2 Features of RAIMM

- Continuously Monitoring of Reliability of the System Memory due to the process, voltage, temperature (PVT) variation
- It provide three state reliability output , Reliable , Less Reliable and Unreliable
- Memory profiling based on the accesses to memory.
- Ranking of memories based on PVT condition and memory profiling.
- Remapping of unreliable memory data into reliable memory.
- System level interrupts for unreliable memory.
- Dynamically change address object once data remapping done

- User has a flexibility to control interrupt generation, i.e. if user want to generate interrupt in less reliable region or unreliable region.

5.3 RAIMM Usage in SOCs

- In predicting memory failures in advance
- Provides a framework to safe-guard the application
- Provides the continuous status of memories to control:

1.DVFS 2.Power optimization 3.Selective code mapping in the memory blocks

5.4 Functional Description

The function of memory manager is to continuously update the reliability status of system memory with respect to PVT Variations and it provides a fail safe solution in case of memory is less reliable or unreliable. To detect the Process Voltage and temperature variation in system memory, PVT sensors is placed with the system memory; these sensors continuously sense the PVT variations in system memory and provides sensed data to memory manager. Using these PVT variation data memory manager compute the reliability. If memory manager detect system memory in less reliable or unreliable then it provides the system level solution. (I.e. for less reliable memory it remaps the data from less reliable memory to redundant memory, for unreliable memory it raises system level interrupt).

The following sections describe the functions of the RAIMM:

- Reliability compute and Ranking Module
- Reliability Template
- Registers for each memory

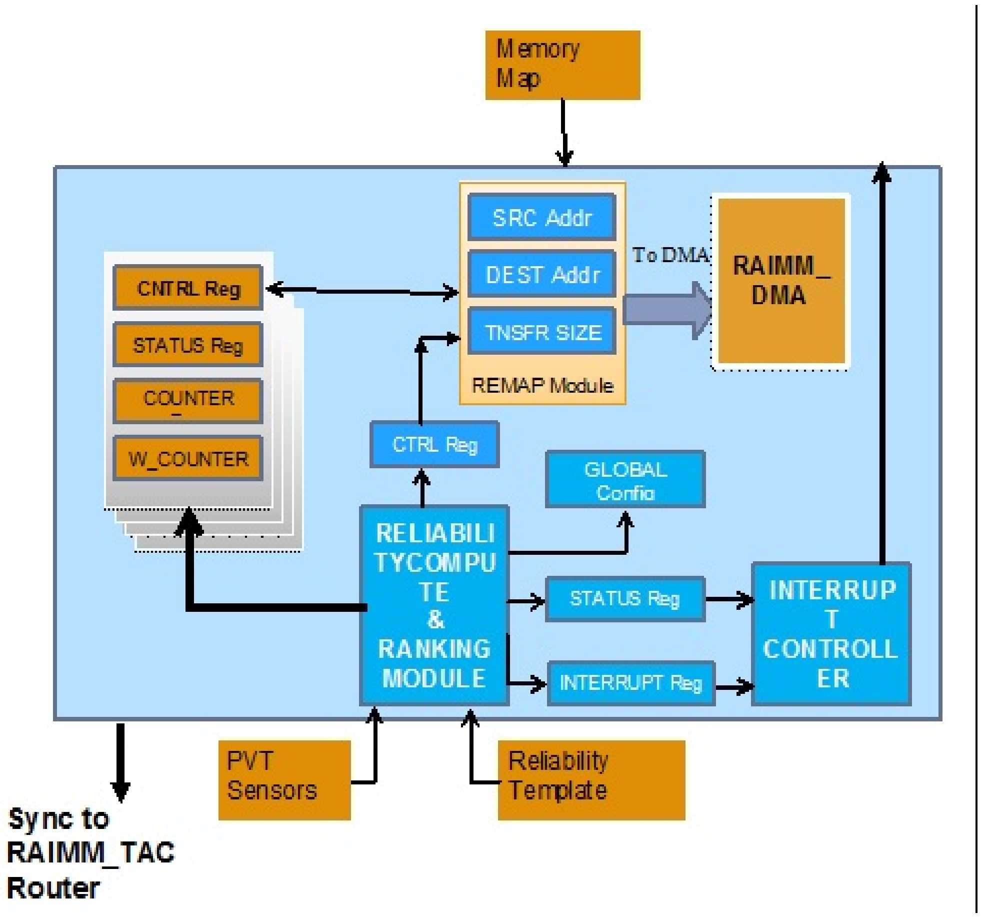


Figure 5.1: Functional block diagram of RAIMM

- Remap Module
- RAIMM Configuration Register
- RAIMM status register
- RAIMM Interrupt registers
- RAIMM Interrupt controller
- RAIMM_DMA

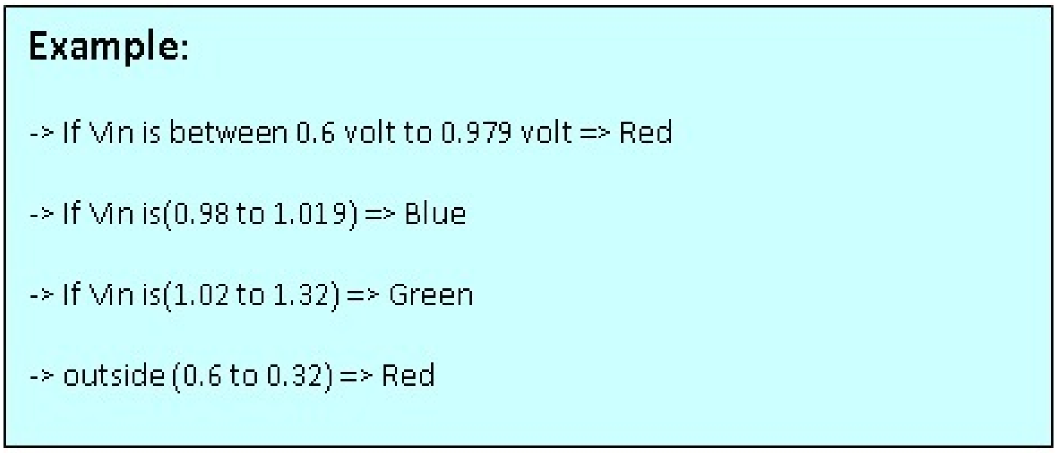
5.4.1 Reliability Compute and Ranking Module

The function of ranking module is dynamically checks the reliability status of system memory with respect to PVT variations. It has predefined PVT reference value including all corner cases for particular technology. The PVT sensors which is placed with the system memory is continuously (during run time) provides the PVT data to ranking module, Using mathematical computation between sensors data and predefine reference data ranking module decide the final reliability status of system memory. Reliability checking strategy:

- 2 stage computation to get reliability status of system memory
- In first stage, ranking module check the reliability status for individual Process , voltage and Temperature variation

First stage Voltage variation reliability checking process

- It decides the reliability of system memory with respect to voltage variatio
- Total Nine possible outputs in the form of color code for different voltage corner cases



Example:

-> If Vin is between 0.6 volt to 0.979 volt => Red

-> If Vin is(0.98 to 1.019) => Blue

-> If Vin is(1.02 to 1.32) => Green

-> outside (0.6 to 0.32) => Red

Figure 5.2: Voltage variation reliability checking process

- Predefine reference values taken form the Monte Carlo simulation and it changes according to technology o Possible outputs of voltage ranking table is R1,R2,R3,B1,B2,B3,G1 o R- red , B-blue , G- green

First stage Process variation reliability checking process

- Reliability of system memory with respect to process variation.
- Predefined reference values taken from the Monte Carlo simulation (40 nm technology).
- 6sigma - process variation simulated.
- It is divided into two stages I.First stage decide the reliability for separate nMOS and pMOS process variation II.Second stage decide the final reliability for process variation using the first stage outputs combinations
- Total Nine possible outputs in the form of color code

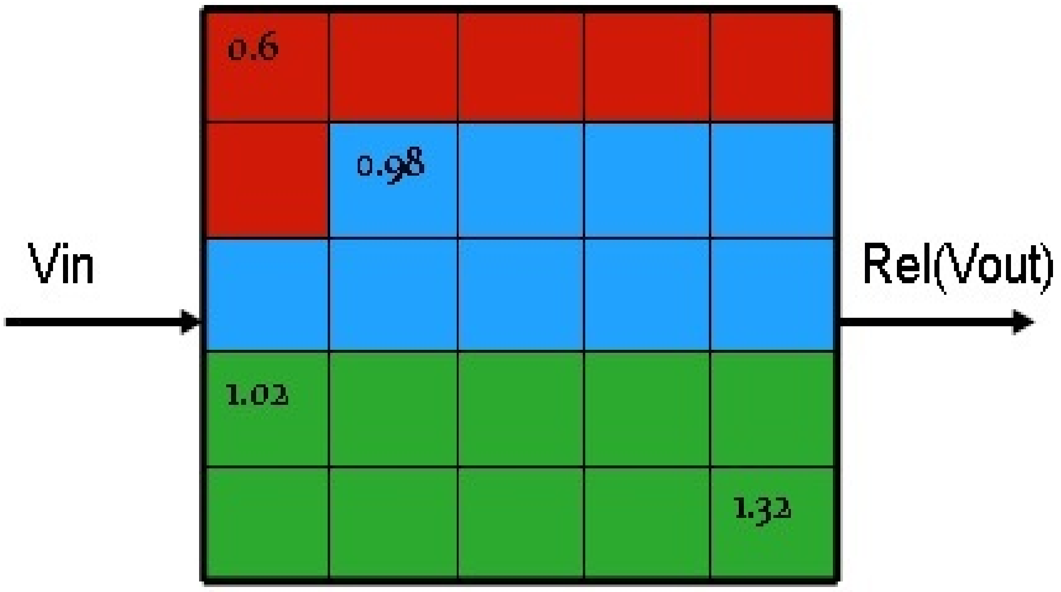
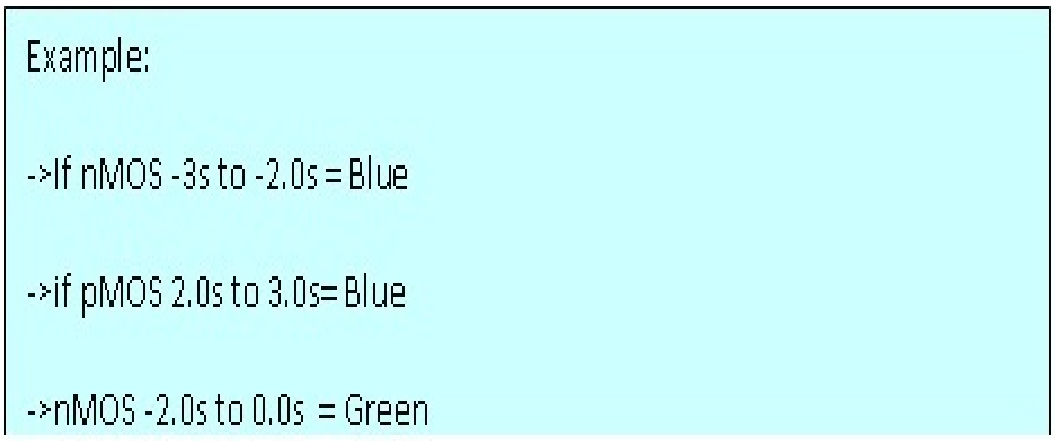


Figure 5.3: Voltage variation reliability checking process



Example:

->If nMOS -3s to -2.0s = Blue

->if pMOS 2.0s to 3.0s= Blue

->nMOS -2.0s to 0.0s = Green

Figure 5.4: Process variation reliability checking process

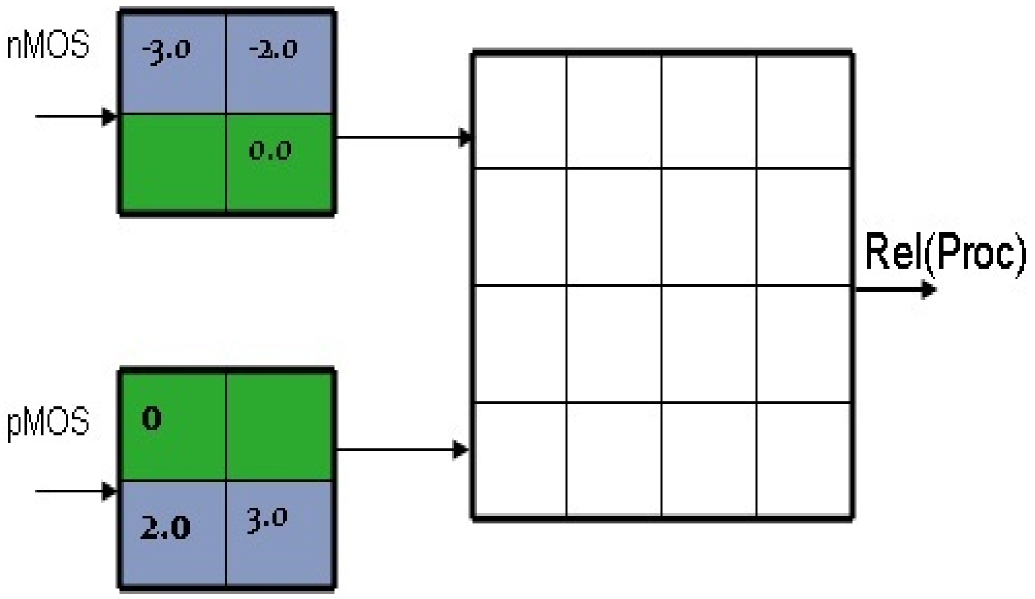


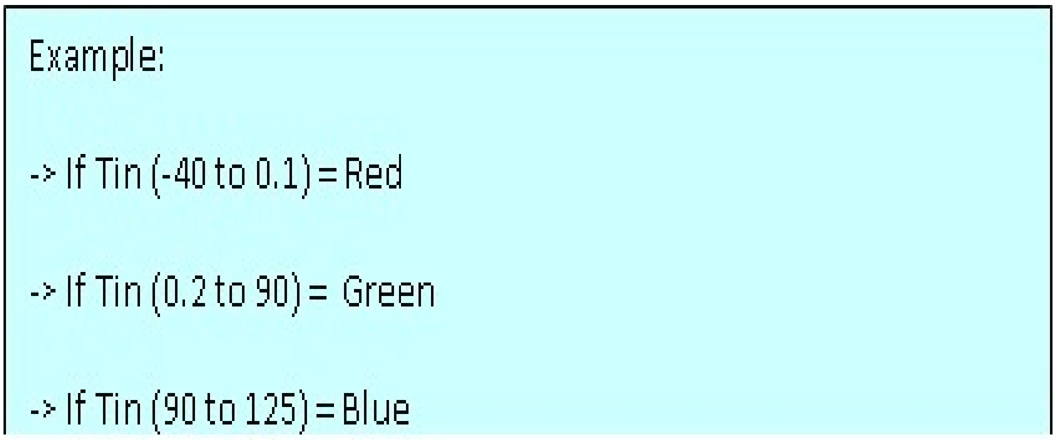
Figure 5.5: Process variation reliability checking process

First stage Temperature Variation reliability checking process

- It decides the reliability of system memory with respect to temperature variation
- Predefined reference values taken from the Monte Carlo simulation (40 nm technology)
- It includes all corner cases between -40C to 150C
- Nine possible outputs in the color code

Final Decision Ranking Table

- Decision Ranking table decides the final reliability with respect to PVT variation
- Total three outputs of decision ranking table o Reliable o Less Reliable o Unreliable



Example:

-> If Tin (-40 to 0.1) = Red

-> If Tin (0.2 to 90) = Green

-> If Tin (90 to 125) = Blue

Figure 5.6: Temperature Variation reliability checking process

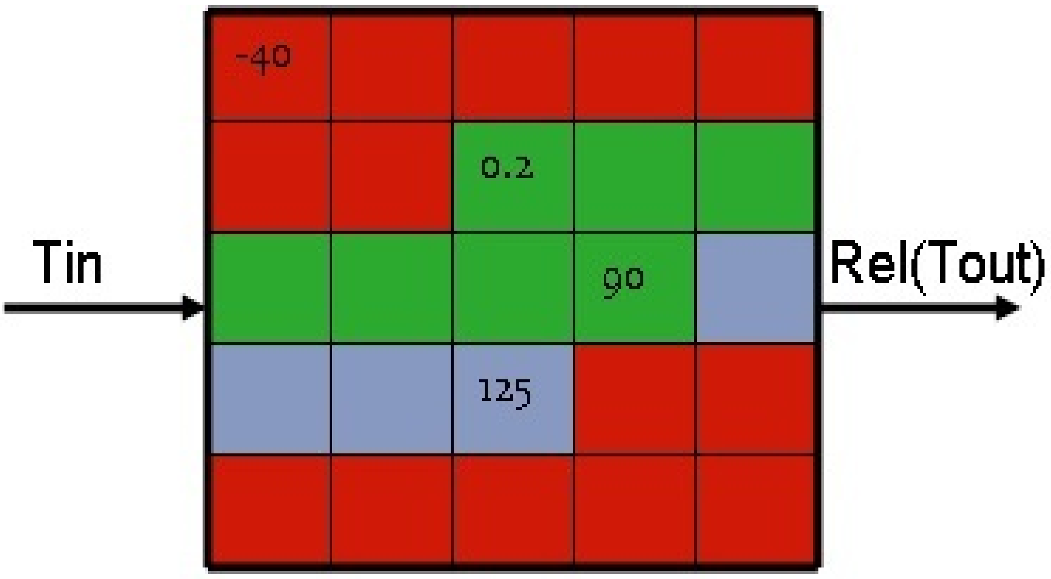
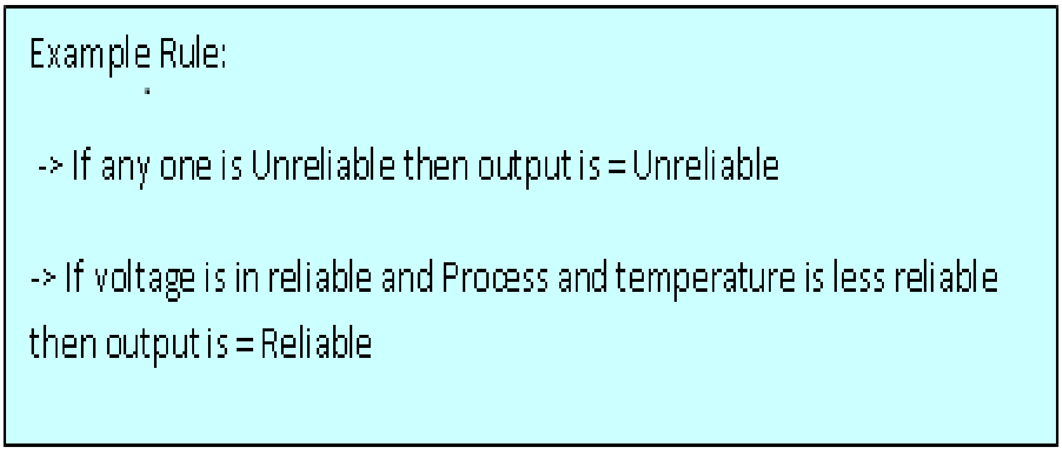


Figure 5.7: Temperature Variation reliability checking process


Example Rule:

-> If any one is Unreliable then output is = Unreliable

-> If voltage is in reliable and Process and temperature is less reliable then output is = Reliable

Figure 5.8: Decision Ranking Table

- Three dimensional ranking tables to cover all possible combinations of first stage outputs.

Flow Chart of Reliability Ranking Module

5.4.2 Reliability Template

As discuss above Ranking module uses the predefine PVT reference values which is taken from the Monte-Carlo simulation, but these values are different for different technologies (i.e. if user wants to move from one technology to other then user should have to make changes in design). To eliminate this issue, RAIMM provides the flexibility to designer is just make a changes in only one text file according to requirement, so no need to make changes in design. Reliability Template is text file it contains PVT reference values for particular technology and reliability ranges for that reference values. It provides this information to reliability checking process or ranking module.

5.4.3 Registers for each memory

For N number of memories N such a registers are required,

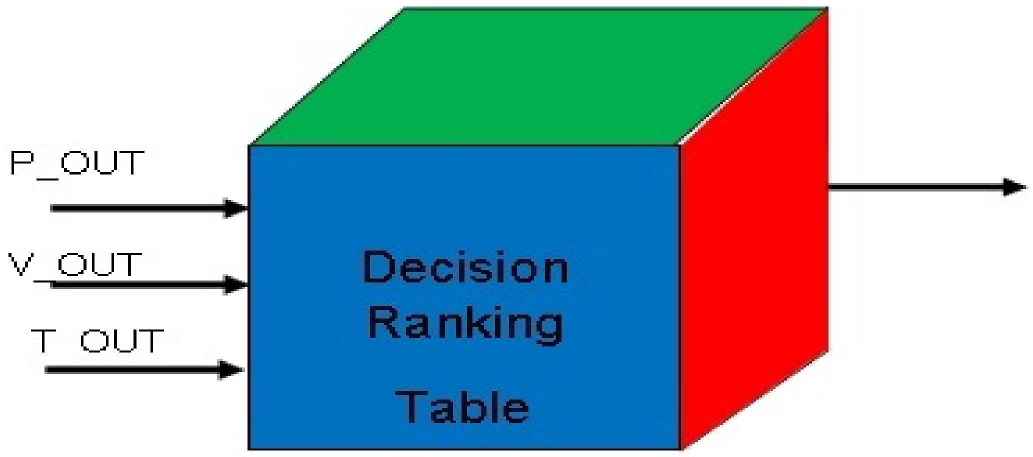


Figure 5.9: Decision Ranking Table

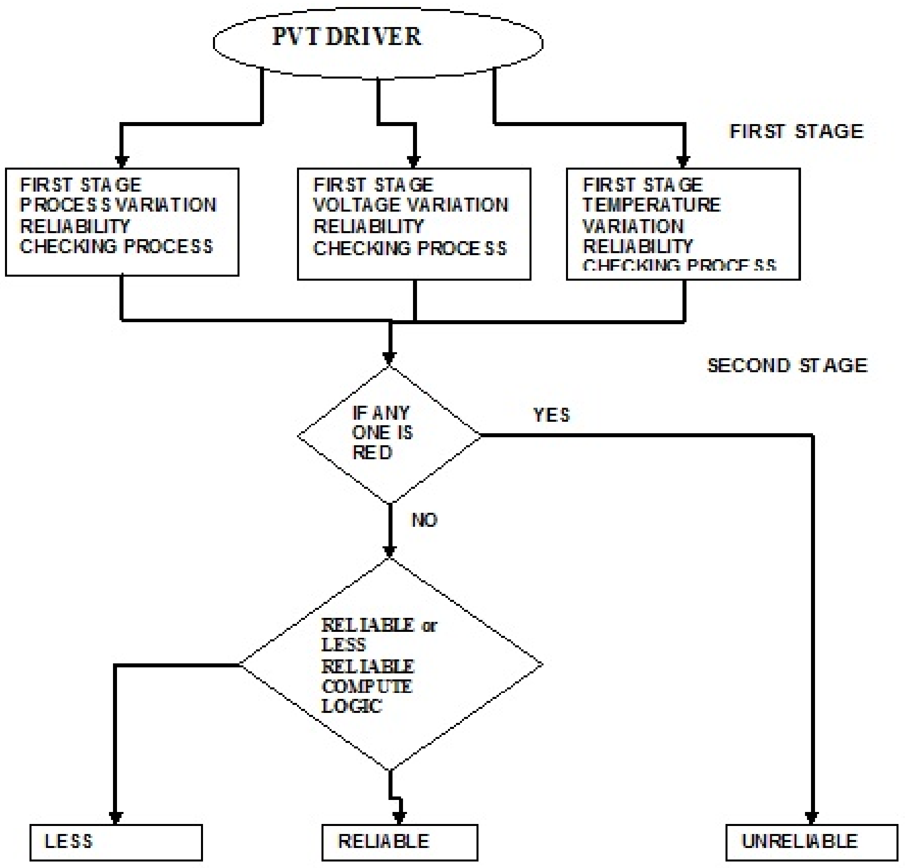

PVT DRIVER
FIRST STAGE
FIRST STAGE PROCESS VARIATION RELIABILITY CHECKING PROCESS
FIRST STAGE VOLTAGE VARIATION RELIABILITY CHECKING PROCESS
FIRST STAGE TEMPERATURE VARIATION RELIABILITY CHECKING PROCESS
SECOND STAGE
IF ANY ONE IS RED
YES
NO
RELIABLE or LESS RELIABLE COMPUTE LOGIC
LESS
RELIABLE
UNRELIABLE

Figure 5.10: Flow chart of RAIMM reliability ranking module

- Status Register
- Counter Register o Two read Counter Register o Two Write Counter Register
- Warning Counter Register
- Control Register

Status Register

It stores the reliability status of memory according to the reliability ranking module output.

Counter Register

- Two read counter:

First read counter counts the number of read transactions but it control through the prescaler and second counter increment whenever first counter gives control signal I.e. if first read counter counts are equals to prescaler then it gives control signal to the second read counter.

- Two write counter:

First counter count the number of write transactions but it control trough the prescaler and second counter increment counter whenever first counter gives control signal. i.e. if first counter counts are equals to prescaler then gives count signal to second counter.

Warning Counter Register

It Counts the warning in terms of reliability, for example whenever memory goes in to the less reliable region warning counter increment.

5.4.4 Remap Module

This module responsible for source address, destination address and transfer size decision for RAIMM_DMA. When memory goes in to the less reliable region immediately remap module is enabled and it take a less reliable memory address as a source address. For destination address remap module search the available redundant memory, if redundant memory is available then it takes redundant memory address as a destination address. Once both addresses are decided remap module calculates the transfer size and it send these information to RAIMM_DMA.

5.4.5 RAIMM Configuration Register

To enable/disable RAIMM.

5.4.6 RAIMM Status Register

32-bit register, it stores the reliability status of each memory. Two bit for each memory to store less reliable and unreliable status. User programmable bits for interrupt control

5.4.7 RAIMM Interrupt Controller

This module generates a system interrupt when memory is in less reliable or unreliable region. RAIMM provides the flexibility to user for generate interrupt in less reliable or unreliable condition. Suppose user wants to generate interrupt only in less reliable condition then user have to set the less reliable programmable bit of status register and interrupt register or user wants to generate interrupt in unreliable region then user have to set unreliable programmable bit of status and interrupt register.

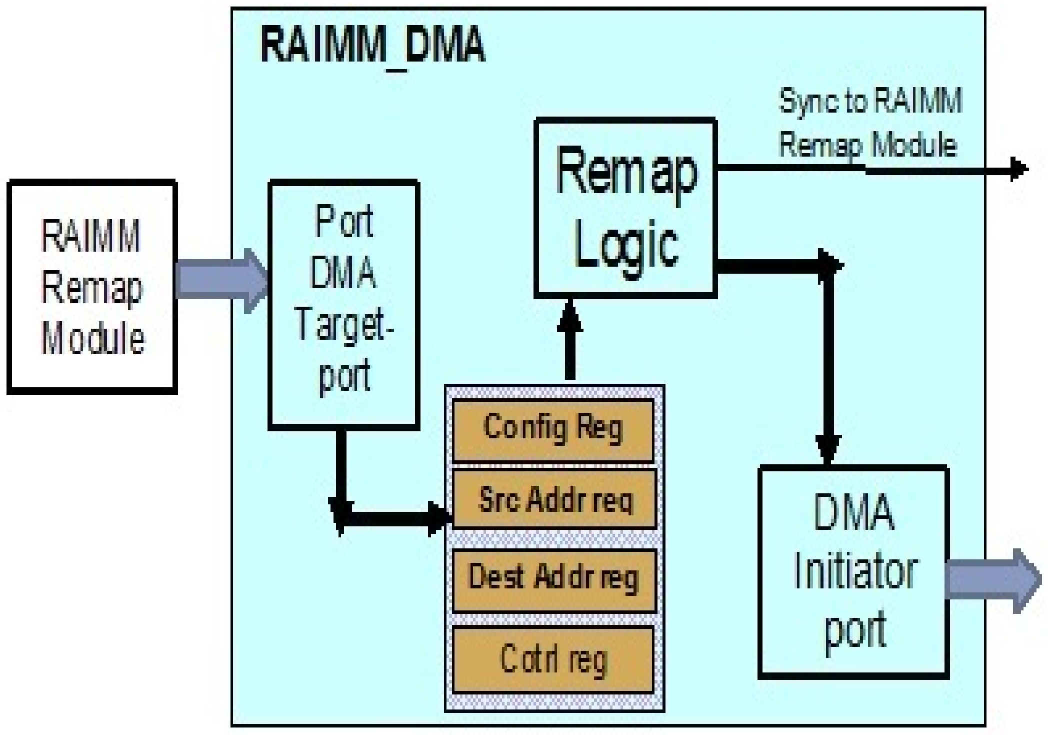


Figure 5.11: Block diagram of RAIMM_DMA

5.4.8 RAIMM_DMA

RAIMM_DMA is a simple DMA controller performs only memory to memory data moving operation. Initially it configure trough the software but once application is start then RAIMM_DMA control and configure through hardware(RAIMM) only. RAIMM remap module provides the source address, destination address, transfer size and DMA enable bit to DMA. Once DMA enabled bit is set, DMA start data remapping operation from source address to destination address. When DMA remapping operation is complete it send a TRANSFER_COMPLETE sync signal to remap module to update the address map object.

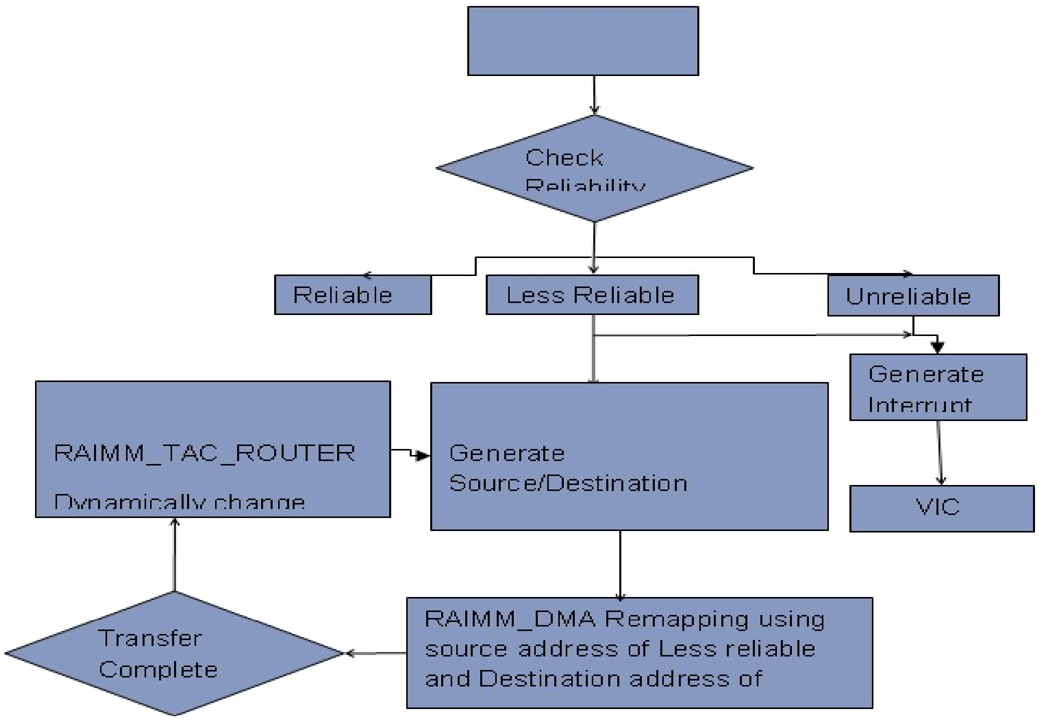


Figure 5.12: RAIMM functional flowchart

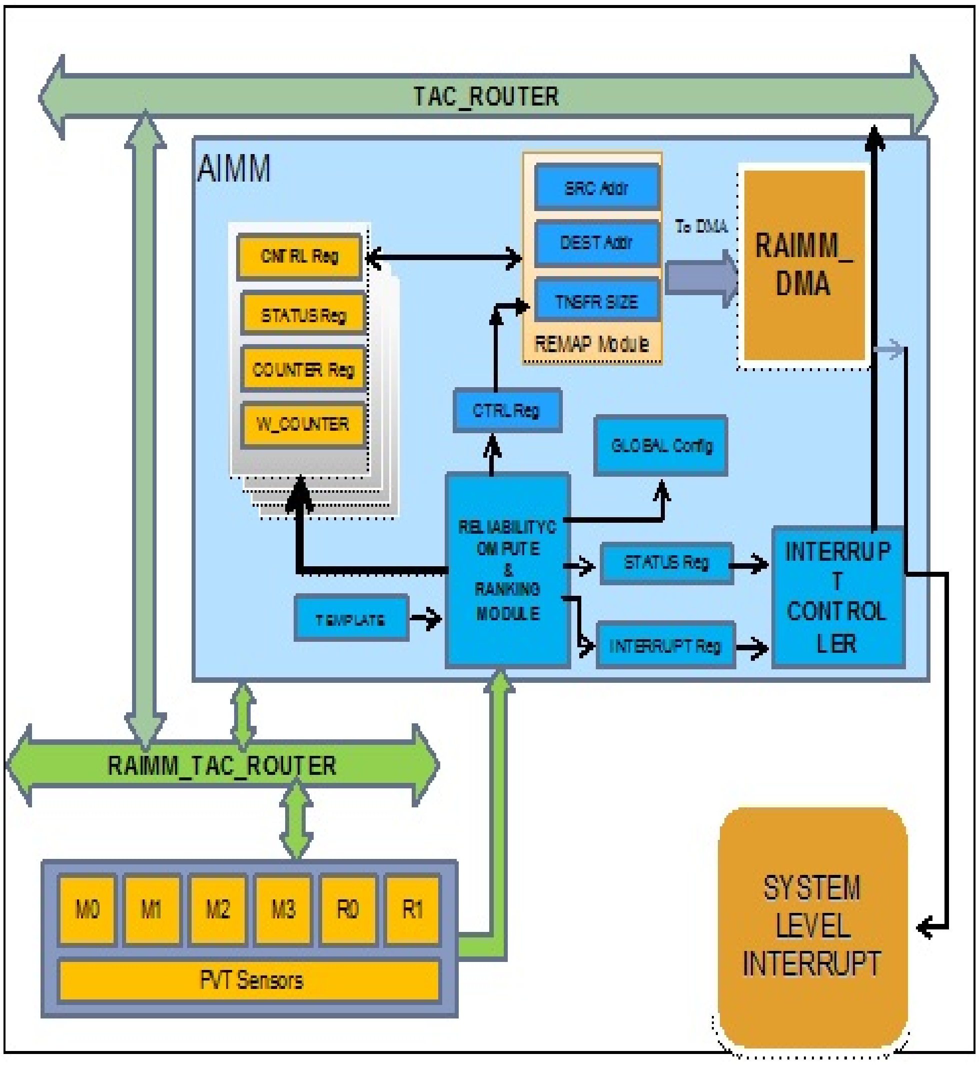


Figure 5.13: RAIMM system level connectivity

RAIMM Functional flow chart

5.5 System Connectivity

5.5.1 RAIMM top level view with platform

5.6 Software Consideration

You must take into account the following software considerations when programming the RAIMM

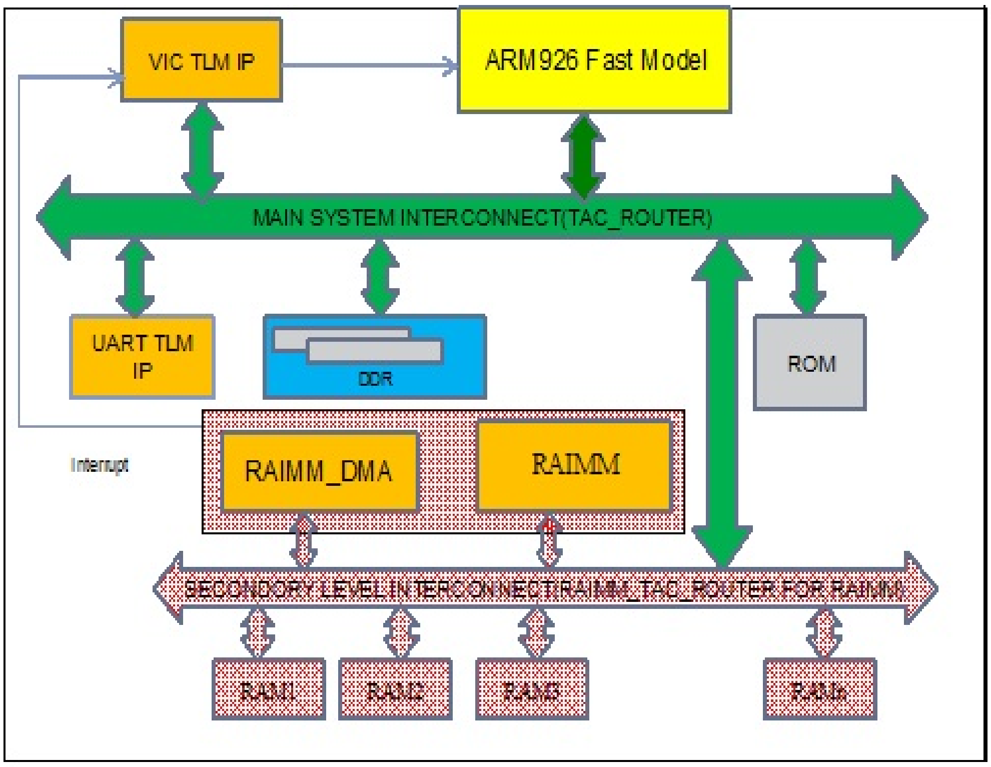

VIC TLM IP
ARM926 Fast Model
MAIN SYSTEM INTERCONNECT(TAC_ROUTER)
UART TLM IP
DDR
ROM
Interupt
RAIMM_DMA
RAIMM
SECONDORY LEVEL INTERCONNECT(RAIMM_TAC_ROUTER FOR RAIMM)
RAM1
RAM2
RAM3
RAMn

Figure 5.14: Top level view of platform with RAIMM

- Initially you must have to provide the address information of each registers to the RAIMM address information registers.
- Initially you must have to provide the information of available redundant memory.

5.7 PROGRAMMERS Model

5.7.1 Register fields

The following applies to the registers used in the RAIMM:

- You must not access reserved or unused address locations because this can result in unpredictable behavior of the device.
- You must write reserved or unused bits of registers as zero, and ignore them on read unless otherwise stated in the relevant text.
- A system or power-on reset resets all registers bits to logic 0 unless otherwise stated in the relevant text.
- All registers support read/write accesses unless otherwise stated in the relevant text. A write updates the contents of a register and a read returns the contents of the register.
- You can only access registers defined in this document using word reads and word writes, unless otherwise stated in the relevant text.

5.7.2 List of RAIMM Register

- RAIMM Global registers

  RAIMM_STATUS Status register

  RAIMM_INTR Interrupt register

RAIMM_CONFIG Configuration register

RAIMM_PRESCALER Prescaler register

RAIMM_SOURCEADDR Source Address register

RAIMM_DESTADDR Destination Address register

RAIMM_TRANSFERSIZE Transfer Size register

RAIMM_srcaddM0 contains the address information of memory block zero.

RAIMM_srcaddM1 contains the address information of memory block one.

RAIMM_srcaddM2 contains the address information of memory block two.

RAIMM_srcaddM3 contains the address information of memory block three.

RAIMM_srcaddM4 contains the address information of memory block four.

RAIMM_srcaddM5 contains the address information of memory block five.

RAIMM_USEM0 - indicates memory block zero is redundant or not.

RAIMM_USEM1 - indicates memory block one is redundant or not.

RAIMM_USEM2 - indicates memory block two is redundant or not.

RAIMM_USEM3 - indicates memory block three is redundant or not.

RAIMM_USEM4 - indicates memory block four is redundant or not.

RAIMM_USEM5 - indicates memory block five is redundant or not.

- RAIMM Local registers for each memory(i.e. if N number of memory, N number of such registers required)

  RAIMM_RDCNTR Two Read transaction count register

  RAIMM_WRCNTR Two Write Transaction count register

  RAIMM_WRNCNTR Warning count register

- RAIMM_DMA registers

  RAIMM_DMA Configuration Register

RAIMM_DMA Source Address Register

RAIMM_DMA Destination Address Register

RAIMM_DMA Control Register

5.7.3 RAIMM Global registers description

STATUS Register

- To store the reliability status of each memory blocks
- 32 bit register
- Bit for each memory block from LSB [Bit 0: less reliable status, Bit 1: Unreliable status]
- Field Description, HW Write access,

  Based on memory reliability status output of PVT ranking module.

  -Bit [1-0] : Memory block zero

  -Bit [3-2] : Memory block one

  -Bit [5-4] : Memory block two

  -Bit [7-6] : Memory block three

  -Bit [9-8] : Memory block four

  -Bit [11-10]: Memory block five

- User Controllable bit: it is for to generate interrupt in less reliable or unreliable status of memory.

  -Bit [18-17]: Memory block zero

  -Bit [20-19]: Memory block one

  -Bit [22-21]: Memory block two

-Bit [24-23]: Memory block three

-Bit [26-25]: Memory block four

-Bit [28-27]: Memory block five

-Bit [31] : Reset the Register

RAIMM Interrupt Register

- To store the reliability status of each memory blocks
- 32 bit register
- Bit for each memory block from LSB[Bit 0: less reliable status, Bit 1: Unreliable status
- Field Description
- Bit set through the hardware, based on memory reliability status output of PVT ranking module.

  -Bit [1-0] : Memory block zero

  -Bit [3-2]: Memory block one

  -Bit [5-4]: Memory block two

  -Bit [7-6]: Memory block three

  -Bit [9-8]: Memory block four

  -Bit [11-10]: Memory block five
- User Controllable bit: it is for to generate interrupt in less reliable or unreliable status of memory.

  -Bit [18-17] : Memory block zero

  -Bit [20-19]: Memory block one

  -Bit [22-21]: Memory block two

Table 5.1: d ddf

-Bit [24-23]: Memory block three

-Bit [26-25]: Memory block four

-Bit [28-27]: Memory block five

-Bit [31]: Reset the Register

RAIMM Configuration Register

- 32 bit register
- Enabled/Disabled RAIMM
- Field Description:

  Bit [0]: 1-Enabled memory manager. 0 -Disabled memory manager

RAIMM Prescaler Register:

- 32 bit register
- To Control Read/Write transaction counter overflow
- Field Description:

  Bit [0-10]: User Programmable, User can fix Prescaler value.

RAIMM Source Address Register

- To store the address of less-reliable or unreliable memory
- 32 bit register
- Field Description:

  Bit [0-31]: To store source address

RAIMM Destination Address Register

- To store the address of available redundant memory
- 32 bit register
- Field Description:

  Bit [0-31]: To store source address

RAIMM Transfer Size Register

- To store the transfer size, data size which is in the less-reliable or unreliable memory
- 32 bit register
- Field Description:

  Bit [0-10]: To store transfer size

RAIMM_srcadd[M0-M5] Register

- 32 bit register
- It contain the initial address information of memory, N registers required for N number of memory
- Field Description:

  Bit [0-31]: To store the address information[M0-M5]

RAIMM_USE[M0-M5] Register

- 32 bit register
- It contains the information of redundant memory, N registers required for N number of memory.
- Initially configure through the software, initially software should provide the information of redundant memory.
- Field Description:

  Bit [0-31]: To store the address information[M0-M5]

5.7.4 RAIMM Local registers for each memory

Two Read Counter Registers

- To count no of read transaction to the each memory block.
- N such registers for N memory
- Both are 32 bit registers
- Prescaler register control the First read counter register and first read counter register control the second read counter register.
- Field Description is same for both read counter register:

  -Bit [30-0] - Count the number of read transactions

  -Bit [31] - Reset the Register

Two Write Counter Registers

- To count no of write transactions to the each memory blocks.
- N such registers for N memory
- Both are 32 bit registers
- Prescaler register control the first write counter register and first write counter register control the second read counter register.
- Field Description is same for both write counter register:

  -Bit [30-0] - Count the number of read transactions

  -Bit [31] - Reset the Register

Warning counter register

- To count no of warning when memory is less reliable or unreliable.
- N such registers for N memories
- 32 bit registers
- Field Description:

  -Bit [30-0] - To Count number of warnings

  -Bit [31] - Reset the Register

5.8 RAIMM Verification

5.8.1 Overview

- The additional logic for the verification of the RAIMM.
- The tests for the signals are system specific and enable to write the necessary test cases.

- These tests are controlled by registers.
- Driver is used to provide the PVT values of each memory, through these values RAIMM check the reliability level of each memory

5.9 Driver

- Driver is used to provide the PVT values for each memory.
- RAIMM decides the reliability of each memory using these values.
- Reliability Ranges are decided by the Template i.e. specification.(input_specs.txt file)
- This template has three levels of Reliability

  1 Reliable i.e. Green

  2 Less Reliable i.e. Blue

  3 Unreliable. i.e. Red

Example: Voltage: Green 1.21 1.32 Blue 1.01 1.20 Red 0.8 1.00

This is an example of the template for voltage. User can modify this ranges as per application required. RIMM decides the reliability using these ranges. Green indicates Reliable; same as Blue and Red indicates Less Reliable and Unreliable respectively. So, if voltage is in the range of 1.21 to 1.32, RAIMM take it as a Reliable. Out of these three ranges, Voltage is considered as an unreliable condition. Same is required for nMOS Process, pMOS Process and Temperature.

Constraints (input_specs.txt): User has to define ranges in the order of Temperature, Voltage, nMOS_Process and pMOS_Process and also in same manner. Test Cases:

" To provide the PVT ranges for each Memory. " Each memory block/region required separate Test Cases.

Example: Temperature: Reliable Voltage: Reliable nMOS_Process: Less_Reliable pMOS_Process: Less_Reliable

This is an example of Test Cases for only one memory block/region (test_case_driver1.txt file). These types of test cases are required for all memory block/region.

- Driver takes these ranges to generate the values for the verification purpose.

- Test cases can also be provided in the following manner: E.g. Voltage: Reliable Less_Reliable

So in this case, Driver generates the PVT ranges using both conditions. Constraints (test_case_driverx): User has to define ranges in the order of Temperature, Voltage, nMOS_Process and pMOS_Process and also in same manner.

5.9.1 Registers

For Verification purpose, following registers are required to be enable/disable. 1) Memory Enable Register:

Used to provide memories which are available as a Redundant. Registers:

- RAIMM_USEM0
- RAIMM_USEM1 Reset value : 0x0
- RAIMM_USEM2 Set value : 0x1
- RAIMM_USEM3
- RAIMM_USEM4
- RAIMM_USEM5

By default, M4 and M5 are provided as a Redundant Memory than it is dynamically changed as per Reliable levels.

Note: If redundant memory is in unreliable condition, than RAIMM uses only System memory, if anyone is free and in reliable condition which is totally dependent on Driver Test Cases. If System Memory is not free, RAIMM doesn't work.

2) Source Address Registers: To provide the predefined source address of each memory. Data is transfer to source address of memory which one is enabled by Memory Enable Register. Registers:

- RAIMM_srcaddM0
- RAIMM_srcaddM1
- RAIMM_srcaddM2
- RAIMM_srcaddM3
- RAIMM_srcaddM4
- RAIMM_srcaddM5

3) Transfer Size Register: To provide the size of data to transfer the data of Unreliable/Less Reliable Memory to the Reliable memory. RAIMM_DMA_SIZE

5.9.2 RAIMM Verification Strategy

RAIMM Verification was done successfully for below test scenario

Test:

- Total 4 system memories (M0-M3)

  Total 2 redundant memories (M4-M5)

  Application code execution from M0 memory

  Code is copying content of M0 Memory to M3 memory

- Failure(Less reliability) is introduced in M0 memory (rest all memories should remain reliable).

- The RAIMM chooses any of the redundant memory modules from two redundant memory modules.
- RAIMM select source address as M0 memory address and Destination Address as available redundant memory address.
- RAIMM_DMA is configured with source address and destination address to remap content of M0 memory to selected redundant memory.
- Once Remapping is done RAIMM change the address of M0 memory with address of selected redundant memory

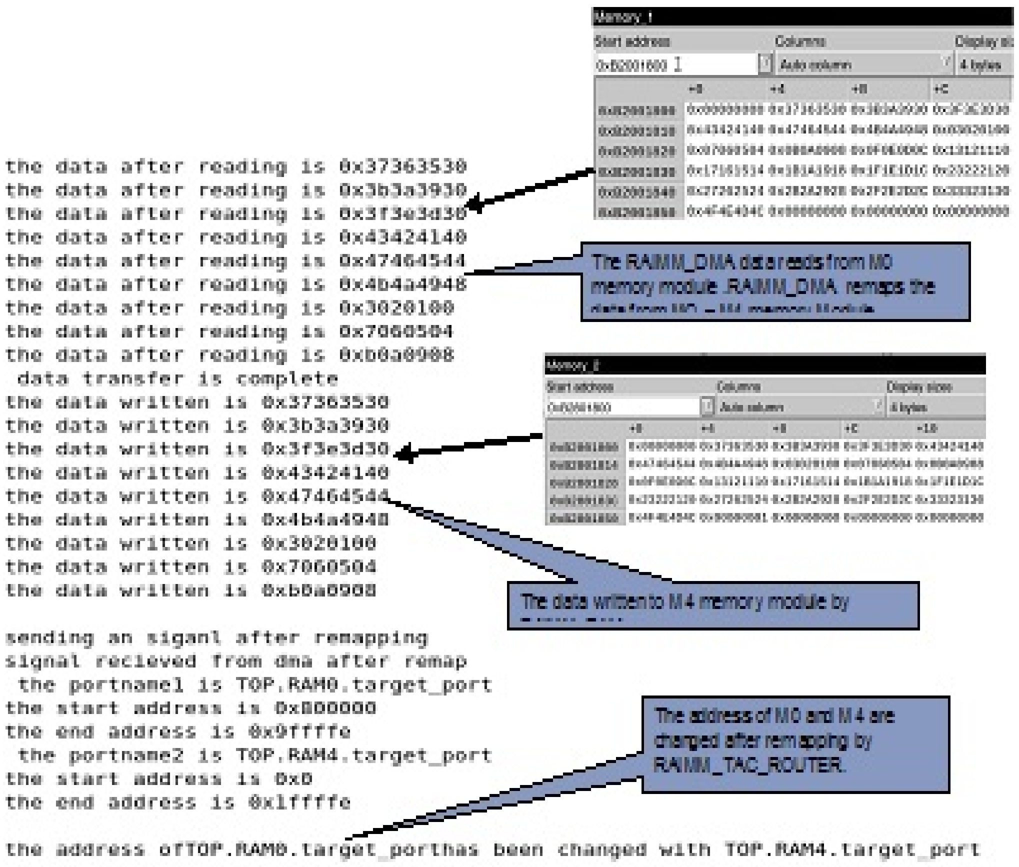
  M0 memory address is M4 memory address, M0 is converted as a redundant memory

  M4 memory address is M0 memory address, M4 is converted ad a memory address is system memory
- It is verified that test is executed successfully
- Latency is estimated

RAIMM Verification Results

Latency of RAIMM for Remapping

- Considering 200 MHz clock. Each clock of 5ns.
- RAIMM total execution time T: 0.005234010(1046802 clock cycles)
- RAIMM_DMA start time for remapping T: 0.000001345(269 clock cycles)
- RAIMM_DMA end time for remapping T: 0.005234010(1046802 clock cycles)
- Latency = end_time - start_time
- Latency = 0.005232665 seconds.(1046533 clock cycles)


The RAIMM_DMA data reads from M0 memory module .RAIMM_DMA remaps the
The data written to M 4 memory module by
The address of M0 and M 4 are changed after remapping by RAIMM_TAC_ROUTER.

Memory_1

Start address: 0x82001800 · Columns: Auto column · Display size: 4 bytes

	+0	+4	+8	+C
0x82001800	0x00000000	0x37363530	0x3B3A3930	0x3F3E3D30
0x82001810	0x43424140	0x47464544	0x4B4A4948	0x03020100
0x82001820	0x07060504	0x0B0A0908	0x0F0E0D0C	0x13121110
0x82001830	0x17161514	0x1B1A1918	0x1F1E1D1C	0x23222120
0x82001840	0x27262524	0x2B2A2928	0x2F2E2D2C	0x33323130
0x82001850	0x4F4E4D4C	0x00000000	0x00000000	0x00000000

Memory_2

Start address: 0x82001800 · Columns: Auto column · Display size: 4 bytes

	+0	+4	+8	+C	+10
0x82001800	0x00000000	0x37363530	0x3B3A3930	0x3F3E3D30	0x43424140
0x82001814	0x47464544	0x4B4A4948	0x03020100	0x07060504	0x0B0A0908
0x82001828	0x0F0E0D0C	0x13121110	0x17161514	0x1B1A1918	0x1F1E1D1C
0x8200183C	0x23222120	0x27262524	0x2B2A2928	0x2F2E2D2C	0x33323130
0x82001850	0x4F4E4D4C	0x00000001	0x00000000	0x00000000	0x00000000

```
the data after reading is 0x37363530
the data after reading is 0x3b3a3930
the data after reading is 0x3f3e3d30
the data after reading is 0x43424140
the data after reading is 0x47464544
the data after reading is 0x4b4a4948
the data after reading is 0x3020100
the data after reading is 0x7060504
the data after reading is 0xb0a0908
 data transfer is complete
the data written is 0x37363530
the data written is 0x3b3a3930
the data written is 0x3f3e3d30
the data written is 0x43424140
the data written is 0x47464544
the data written is 0x4b4a4948
the data written is 0x3020100
the data written is 0x7060504
the data written is 0xb0a0908

sending an siganl after remapping
signal recieved from dma after remap
 the portname1 is TOP.RAM0.target_port
the start address is 0x800000
the end address is 0x9ffffe
 the portname2 is TOP.RAM4.target_port
the start address is 0x0
the end address is 0x1ffffe

the address ofTOP.RAM0.target_porthas been changed with TOP.RAM4.target_port
```

Figure 5.15: RAIMM Verification Result with latency

- The latency above shows the time required by RAIMM_DMA to remap 2MB of data from one memory module to other module

5.9.3 RAIMM Verification with UART Interrupt operation

- Test:

  - Total 4 system memories (M0-M3)

  - Total 2 redundant memories (M4-M5)

  - Application code execution from M0 memory

  - UART configured to generate system level interrupt

  - UART interrupt service routine address point to M0 memory Address

  - Code is copying content of M0 Memory to M4(Redundant) memory after selected count of UART interrupt

- Initially RAIMM configured.

- UART system level interrupt generated successfully.

- Failure(Less reliability) is introduced in M0 memory (rest all memories should remain reliable)

- RAIMM warning counter register count the number of interrupt from UART. Once counter reach at selected number then it immediately enable the RAIMM Remap module to remap the content of less reliable memory to reliable redundant memory.

- The RAIMM chooses any of the redundant memory modules from two redundant memory modules.

- RAIMM select source address as M0 memory address and Destination Address as available redundant memory address.

- RAIMM_DMA is configured with source address and destination address to remap content of M0 memory to selected redundant memory.
- Once Remapping is done RAIMM change the address of M0 memory with address of selected redundant memory

  - M0 memory address is M4 memory address, M0 is converted as a redundant memory

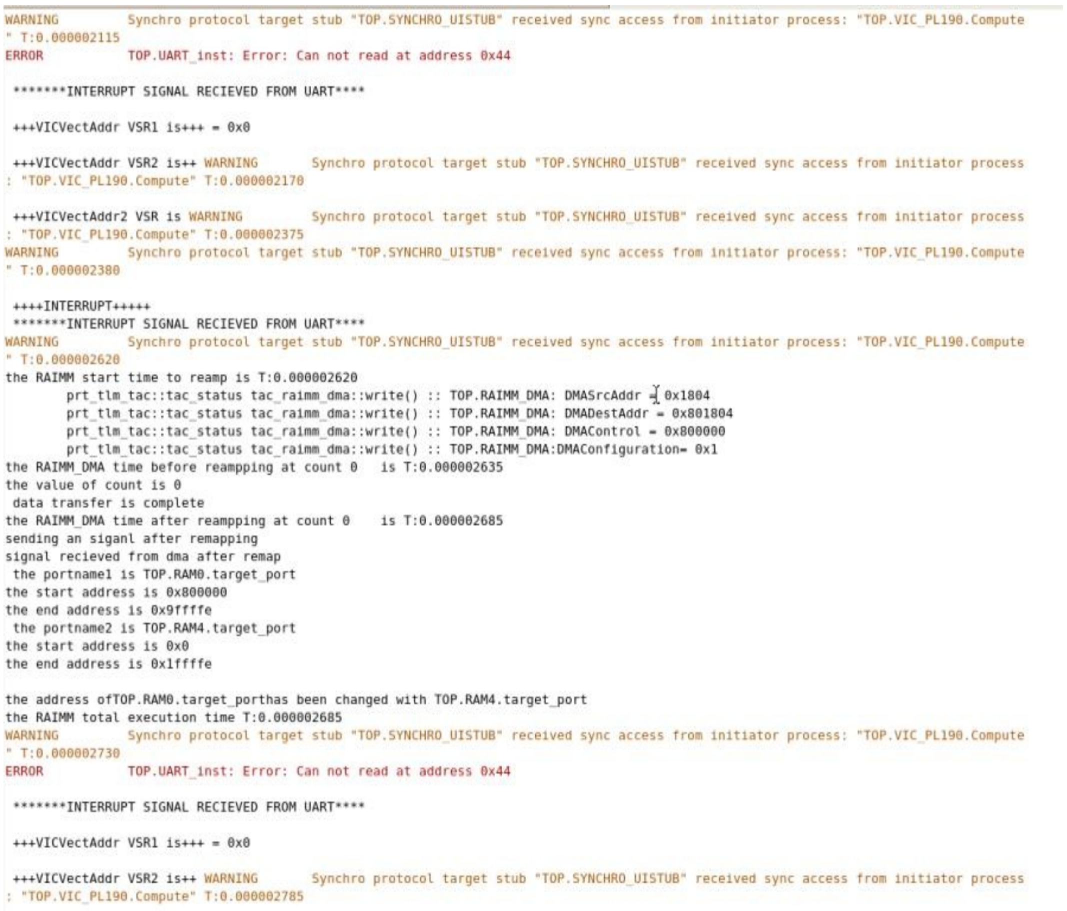
  - M4 memory address is M0 memory address, M4 is converted ad a memory address is system memory
- It is verified that test is executed successfully
- Latency is estimated

RAIMM Verification Results with UART interrupt operation

5.10 Methodology for Implementing RAIMM Transaction level modeling

5.10.1 TLM Methodology

Transaction-level modeling (TLM) is a high-level approach to modeling digital systems here details of communication among modules is separated from the details of the implementation of functional units or of the communication architecture. Communication mechanisms such as buses or FIFOs are modeled as channels, and are presented to modules using SystemC interface classes. Transaction requests take place by calling interface functions of these channel models, which encapsulate low-level details of the information exchange. At the transaction level, the emphasis is more on the functionality of the data transfers - what data are transferred to and from what locations - and less on their actual implementation that is, on the actual protocol used for data transfer. This approach makes it easier for the system-level



```
WARNING         Synchro protocol target stub "TOP.SYNCHRO_UISTUB" received sync access from initiator process: "TOP.VIC_PL190.Compute
" T:0.000002115
ERROR           TOP.UART_inst: Error: Can not read at address 0x44

 *******INTERRUPT SIGNAL RECIEVED FROM UART****

 +++VICVectAddr VSR1 is+++ = 0x0

 +++VICVectAddr VSR2 is++ WARNING       Synchro protocol target stub "TOP.SYNCHRO_UISTUB" received sync access from initiator process
: "TOP.VIC_PL190.Compute" T:0.000002170

 +++VICVectAddr2 VSR is WARNING         Synchro protocol target stub "TOP.SYNCHRO_UISTUB" received sync access from initiator process
: "TOP.VIC_PL190.Compute" T:0.000002375
WARNING         Synchro protocol target stub "TOP.SYNCHRO_UISTUB" received sync access from initiator process: "TOP.VIC_PL190.Compute
" T:0.000002380

 ++++INTERRUPT+++++
 *******INTERRUPT SIGNAL RECIEVED FROM UART****
WARNING         Synchro protocol target stub "TOP.SYNCHRO_UISTUB" received sync access from initiator process: "TOP.VIC_PL190.Compute
" T:0.000002620
the RAIMM start time to reamp is T:0.000002620
        prt_tlm_tac::tac_status tac_raimm_dma::write() :: TOP.RAIMM_DMA: DMASrcAddr = 0x1804
        prt_tlm_tac::tac_status tac_raimm_dma::write() :: TOP.RAIMM_DMA: DMADestAddr = 0x801804
        prt_tlm_tac::tac_status tac_raimm_dma::write() :: TOP.RAIMM_DMA: DMAControl = 0x800000
        prt_tlm_tac::tac_status tac_raimm_dma::write() :: TOP.RAIMM_DMA:DMAConfiguration= 0x1
the RAIMM_DMA time before reampping at count 0   is T:0.000002635
the value of count is 0
 data transfer is complete
the RAIMM_DMA time after reampping at count 0    is T:0.000002685
sending an siganl after remapping
signal recieved from dma after remap
 the portname1 is TOP.RAM0.target_port
the start address is 0x800000
the end address is 0x9ffffe
 the portname2 is TOP.RAM4.target_port
the start address is 0x0
the end address is 0x1ffffe

the address ofTOP.RAM0.target_porthas been changed with TOP.RAM4.target_port
the RAIMM total execution time T:0.000002685
WARNING         Synchro protocol target stub "TOP.SYNCHRO_UISTUB" received sync access from initiator process: "TOP.VIC_PL190.Compute
" T:0.000002730
ERROR           TOP.UART_inst: Error: Can not read at address 0x44

 *******INTERRUPT SIGNAL RECIEVED FROM UART****

 +++VICVectAddr VSR1 is+++ = 0x0

 +++VICVectAddr VSR2 is++ WARNING       Synchro protocol target stub "TOP.SYNCHRO_UISTUB" received sync access from initiator process
: "TOP.VIC_PL190.Compute" T:0.000002785
```

Figure 5.16: RAIMM Verification Results with UART interrupt

designer to experiment, for example, with different bus architectures (all supporting a common abstract interface) without having to recode models that interact with any of the buses, provided these models interact with the bus through the common interface. The right TLM abstraction level often depends on the application domain and purpose of running the simulation in the first place. Some applications require cycle accuracy - for example to analyze detailed cache behavior. Some might even require linking in RTL models as part of the development flow, while other applications (typically software development tasks) only need functional accuracy. At the same time, if modeling is performed at lower level of abstraction model, it would directly affect the simulation performance of the complete platform and could be not at all usable so identifying the application and deployment need to be done cautiously. The TLM methodology starts with the first level of conceiving idea about a system or SoC. The main objectives of the methodology is as follows -

- SystemC IP prototyping/System level platform for functional verification.

  - Understanding the RAIMM IP specification and modeling in TLM or any other abstraction level as per the requirements.

- Testbench and test vectors generations with System Level Platform.

  - First level of test bench available and test vectors could be generated with the platform.

- Start Early Software Development with TLM Platform.

  - Once a prototype available, AN S/W development could be started without waiting for real board or H/W platform.

- Platform/tests availability for RTL verification.

  - Second step will provide test inputs and test bench for this step. All test vectors could be reused.

- S/W Validation on System Level Platform.

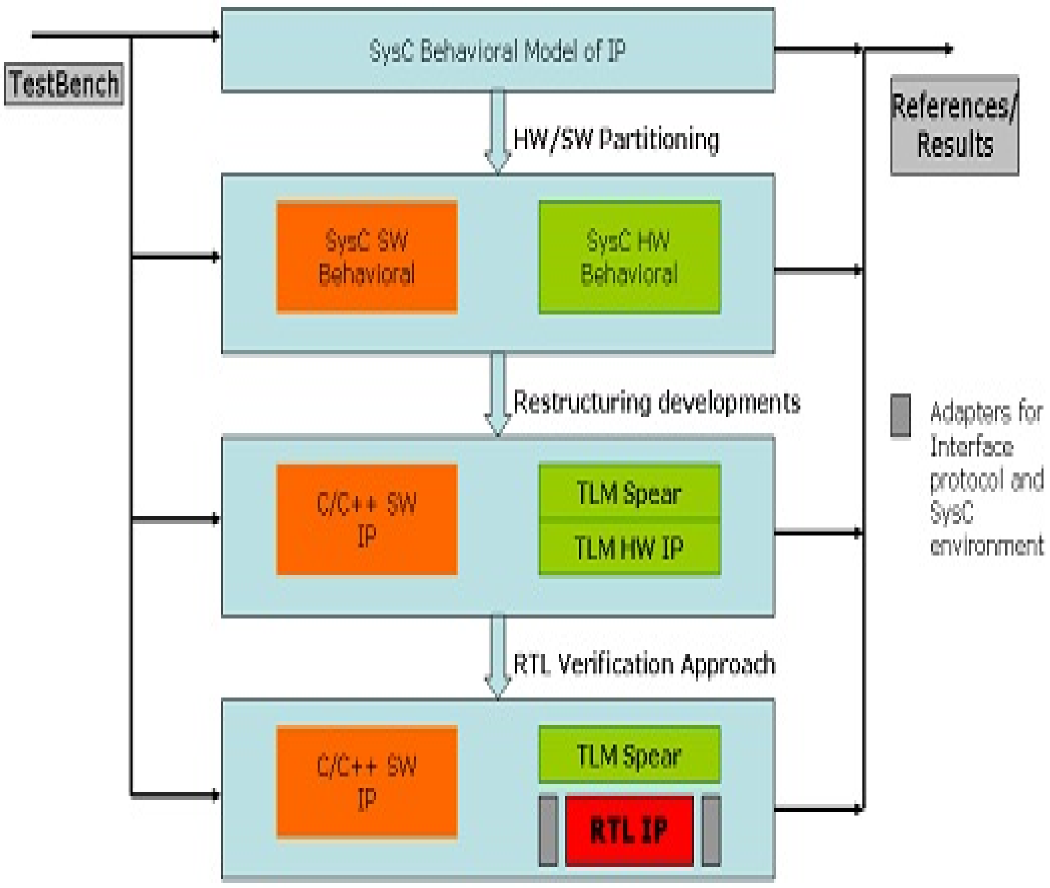


Figure 5.17: Generic Design Flow with TLM Methodology

- This platform may be used for Real S/W validation. Although there had been simulation performance issues identified when RTL and SystemC plugged together but there are some methods to improve this bottleneck

5.10.2 PLATFORM

This proposal is basically to have an ARM based abstracted Platform. The intent of the platform is to provide a preliminary platform with minimum required components which permits to run test software and permits to plug the RAIMM IP, which facilitates the IP verification.

The approach helps for following of the activities in design cycle -

- This platform which is used for developing test vectors with an abstracted IP prototype. These test vectors could be further used for RTL verification with references already available.

- Start early S/W driver development and perform S/W debugging. In this platform the S/W debugging capabilities are very good as well, it permits to debug the hardware also to a great extent.

- A similar platform could be used for HW/SW partitioning of RAIMM Ip. Facilitated with some tools available, it could be used to gather performance statistics in order to partition Software functionalities versus Hardware.

This System platform is a completely abstracted platform and IP is modeled in SystemC at higher abstraction level either transaction level models or timing annotated models. In this scenario, the platform gives a glimpse of SoC and a complete hardware + software prototype is available.

The added advantage of platform are better simulation performance, good debugging capabilities with test benches as it permits to write C test benches. This platform will be highly recommended in developing IPs which might have software along with hardware and Another platform in parallel will permit to perform various activities e.g. IP prototyping at higher abstraction level, hardware software partitioning or developing a complete system prototype with their own IP.

Such a platform could be a demo platform as well as it could help to do early evaluation of System with their own IPs. We may develop an abstracted IP model and perform all kind of analysis, start developing test benches for further phases of development. This could be a suitable platform to perform hardware software partitioning and make explorations pertaining to make a better decision on what could go in software and the hardware portion of the IP.

5.10.3 Components of the Platform

The current platform is based on TLM TAC (Transaction Accurate Channel) protocol. The available TLM IP models are

ARM926

- ARM926 core for running C software and interrupt handling.

DMAC PL080

- TLM TAC IP Model available. Full functionality available.

VIC

- TLM TAC IP Model available. Full functionality available.

UART

- TLM TAC IP Model available. Full functionality available.

RAIMM

- TLM IP Model.

ROUTER

- The router is the interconnect backbone in the platform known as Transaction Accurate Channel (TAC).
- The TAC is responsible for routing the transaction from one IP to other IP, IP to Memories.
- The router in the platform acts as decoder, the decoder decodes the transaction of each IP then routes according to the address.
- Once the decoder decodes the transaction, the router will have start address and size of the target IP. If the transaction address information doesn't matches the entries of memory map the router returns an error.
- Each of the hierarchy or sub-hierarchy of the TAC channel needs to have a memory map. The memory map is for the accessible registers and memory regions.
- The platform consists of two routers, one known as MAIN SYSTEM INTERCONNECT (TAC_ROUTER) and other is dedicated router for the integration of RAIMM IP known as SECONDRY LEVEL INTERCONNECT (RAIMM_TAC_ROUTER).
- The TAC_ROUTER is responsible routing the transactions from its bind Ips to other Ips or RAIMM_TAC_ROUTER.

RAIMM_TAC_ROUTER

- NEED OF RAIMM_TAC_ROUTER
- The RAIMM_TAC_ROUTER was mainly added for better performance of RAIMM IP.

- The RAIMM_TAC_ROUTER is connected to split memories and dedicated DMA.
- The RAIMM_TAC_ROUTER has additional variables which indicate about the reliability and redundancy of the memories. These variables changes according to the reliability of the memories.
- The RAIMM_TAC_ROUTER initializes the split memories into useable memories and redundant memories. The initialization is according user specification.
- The RAIMM_TAC_ROUTER tracks all the transaction coming to memories and to count number of transaction to the memory modules, the transaction count is helpful in judging the memory modules.
- The RAIMM_TAC_ROUTER helps in remapping by dynamically changing the start address of the memory modules.
- The RAIMM_TAC_ROUTER is responsible for dynamically changing the start, end address of two memory modules i.e. redundant and unreliable memory module used for remapping.
- The RAIMM_TAC_ROUTER changes the address only after getting and signal from RAIMM_DMA.

Dynamically Changing the Address of Memory Modules

- The platform contains the map file parser which creates an object for each entry in map file.
- The RAIMM_TAC_ROUTER uses the object created by map file parser to dynamically change the address of memory modules.
- Whenever the RAIMM_TAC_ROUTER gets a signal from the RAIMM_DMA the router exchanges the address of the memory modules.

- After remapping the RAIMM_TAC_ROUTER also changes the status of the memory modules by disabling the unreliable memory and sets the remapped redundant memory as currently in use.
- For eg if M0 data is copied to M5 then start address of M0 and M5 will be changed as shown in table. The router also changes the status of the memory modules.

5.10.4 LAUCHING OF PLATFORM

- The rvdebugg is used to launch the ARM based platform.
- The ARM926 fast model simulator has been deployed in the platform.
- Once the platform is launched the driver (PVT sensors) acts as stimulus for the operation RAIMM IP.
- An input file can be programmed for rvdebugg which will run the simulations from platform environment and let user perform the configurations needed from the software side.

5.11 RAIMM Results

5.11.1 Simulation results of RAIMM ranking module

5.11.2 RAIMM - Ranking Module Behavior

5.11.3 RAIMM Simulation Results

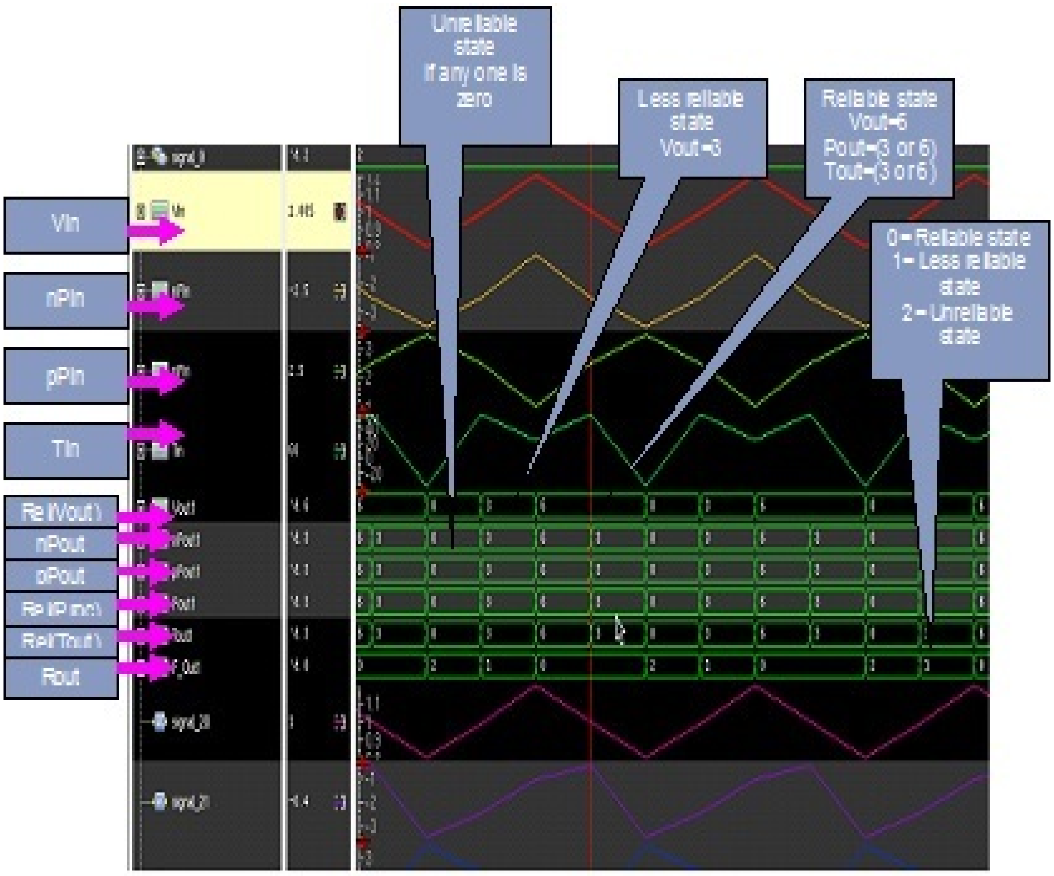

Unreliable state If any one is zero
Less reliable state Vout=3
Reliable state Vout=6 Pout=(3 or 6) Tout=(3 or 6)
0=Reliable state 1=Less reliable state 2=Unreliable state
VIn
nPIn
pPIn
TIn
Rel(Vout)
nPout
pPout
Rel(Pinc)
Rel(Tout)
Rout

Figure 5.18: Simulation results of RAIMM ranking module

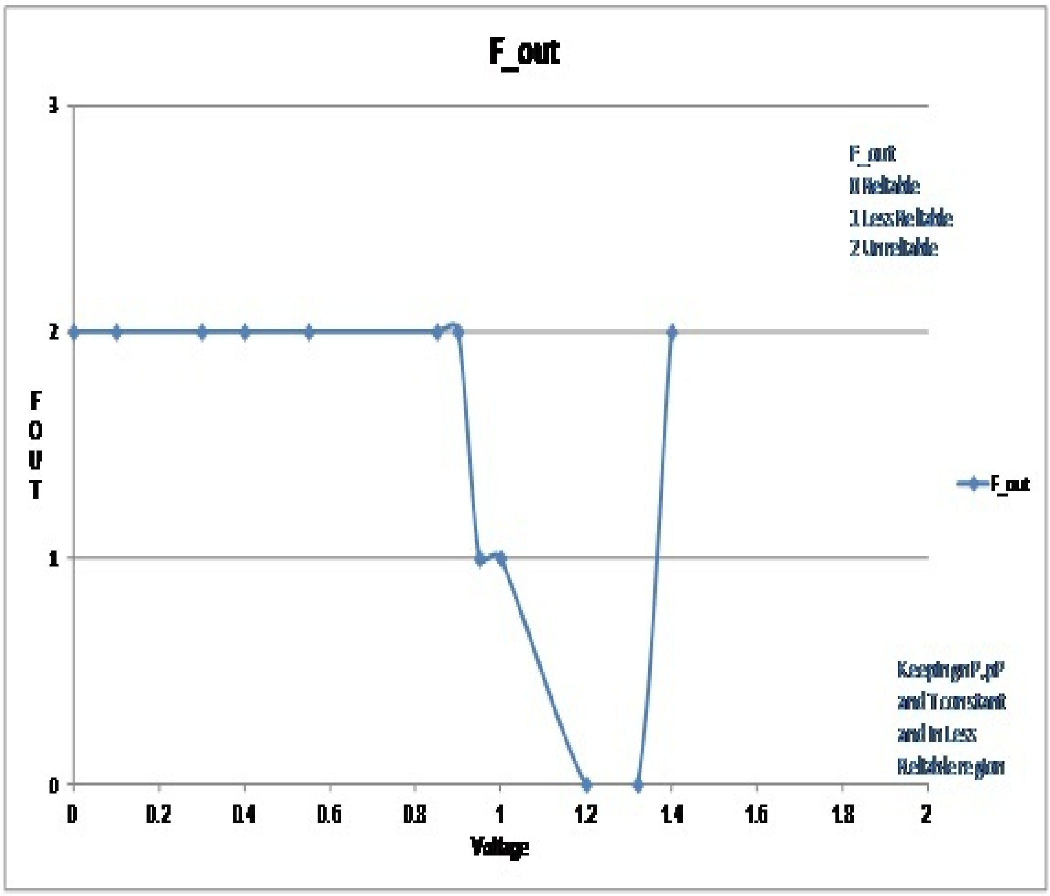
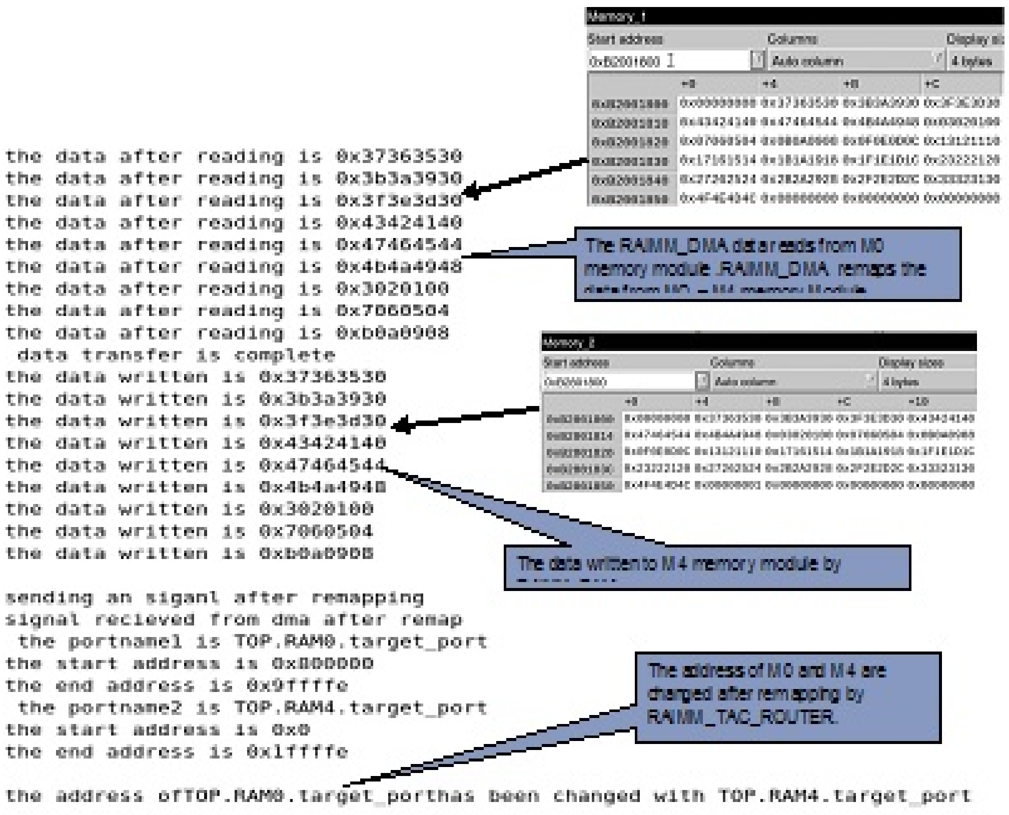


Figure 5.19: Voltge vs Reliabiliy


The RAIMM_DMA data reads from M0 memory module .RAIMM_DMA remaps the data from M0 → M4 memory Module
The data written to M 4 memory module by
The address of M0 and M 4 are changed after remapping by RAIMM_TAC_ROUTER.

Memory_1

Start address: 0x82001800 — Columns: Auto column — Display size: 4 bytes

	+0	+4	+8	+C
0x82001800	0x00000000	0x37363530	0x3B3A3930	0x3F3E3D30
0x82001810	0x43424140	0x47464544	0x4B4A4948	0x03020100
0x82001820	0x07060504	0x0B0A0908	0x0F0E0D0C	0x13121110
0x82001830	0x17161514	0x1B1A1918	0x1F1E1D1C	0x23222120
0x82001840	0x27262524	0x2B2A2928	0x2F2E2D2C	0x33323130
0x82001850	0x4F4E4D4C	0x00000000	0x00000000	0x00000000

Memory_2

Start address: 0x82001800 — Columns: Auto column — Display size: 4 bytes

	+0	+4	+8	+C	+10
0x82001800	0x00000000	0x37363530	0x3B3A3930	0x3F3E3D30	0x43424140
0x82001814	0x47464544	0x4B4A4948	0x03020100	0x07060504	0x0B0A0908
0x82001828	0x0F0E0D0C	0x13121110	0x17161514	0x1B1A1918	0x1F1E1D1C
0x8200183C	0x23222120	0x27262524	0x2B2A2928	0x2F2E2D2C	0x33323130
0x82001850	0x4F4E4D4C	0x00000000	0x00000000	0x00000000	0x00000000

```
the data after reading is 0x37363530
the data after reading is 0x3b3a3930
the data after reading is 0x3f3e3d30
the data after reading is 0x43424140
the data after reading is 0x47464544
the data after reading is 0x4b4a4948
the data after reading is 0x3020100
the data after reading is 0x7060504
the data after reading is 0xb0a0908
 data transfer is complete
the data written is 0x37363530
the data written is 0x3b3a3930
the data written is 0x3f3e3d30
the data written is 0x43424140
the data written is 0x47464544
the data written is 0x4b4a4948
the data written is 0x3020100
the data written is 0x7060504
the data written is 0xb0a0908

sending an siganl after remapping
signal recieved from dma after remap
 the portname1 is TOP.RAM0.target_port
the start address is 0x800000
the end address is 0x9ffffe
 the portname2 is TOP.RAM4.target_port
the start address is 0x0
the end address is 0x1ffffe

the address ofTOP.RAM0.target_porthas been changed with TOP.RAM4.target_port
```

Figure 5.20: RAIMM Simulation Results

Chapter 6

Conclusion

6.1 Conclusion

Purpose of RAIMM is used for increasing the reliability of SoC in sub nm technology. RAIIM is continuously predicting the reliability of system memories, and if any memory is goes in unreliable zone then it provides the system level solution.

References

[1] V. Chandra and R. Aitken, "Impact of Technology and Voltage Scaling on the Soft Error Susceptibility of Nanoscale CMOS," IEEE Intl. Symposium on Defect and Fault Tolerance of VLSI Systems, 2008.

[2] R. Rodriguez et al, "The impact of gate oxide breakdown on SRAM stability," IEEE Electron Device Letters, pp. 559-561, Sep. 2002

[3] E. H. Cannon et al, "The impact of aging effects and manufacturing variation on SRAM soft-error rate," IEEE Transactions on Device and Materials Reliability, Vol. 8, No. 1, pp. 145-152, Mar 2008

[4] Chandra et al, "Impact of Voltage Scaling on Nanoscale SRAM Reliability", DATE'09

[5] Pilo et al, "An SRAM Design in 65nm and 45nm Technology Nodes Featuring Read and Write-Assist Circuits to Expand Operating Voltage", 2006 Symposium on VLSI Circuits

[6] Bathen and Dutt, E-RoC: " Embedded RAIDs-on-Chip for Low Power Distributed Dynamically Managed Reliable Memories ", DATE'2011

[7] Jeff Mueller M.B. "Techniques for Clock Skew Reduction over Intra-Die Process, Voltage, and Temperature Variations".

[8] Changhwan Shin "Advanced MOSFET Designs and Implications for SRAM Scaling"

[9] James W. Tschanz ”Adaptive Body Bias for Reducing Impacts of Die-to-Die and Within-Die Parameter Variations on Microprocessor Frequency and Leakage”

[10] Takayuki GYOHTEN ”An On-Chip Supply-Voltage Control System Considering PVT Variations for Worst-Case less Lower Voltage SoC Design”

[11] Farshad Moradi, Oag Wisland. ”Process Variations in Sub-Threshold SRAM Cells in 65nm CMOS” 22nd International Conference on Microelectronics (ICM 2010).

[12] Amin Khajeh, Aseem Gupta. ”TRAM: A Tool for Temperature and Reliability Aware Memory Design”

[13] Ashok K. Sharma ”Semiconductor Memories: Technology, Testing and Reliability” IEEE Press.

[14] Cheng Zhuo ”Process Variation and Temperature-Aware Reliability Management”

[15] David C Black ”SystemC: From the Ground Up” Kluwer Academic Publisher

[16] Daniel Gajaski ”Transaction level Modeling: An Overview”

[17] IEEE standard SystemC: Language Reference manual

[18] www.systemc.org

[19] ”SystemC Version 2.0 User’s Guide”

[20] Sudeep Pasricha ”Transaction level modeling of SoC with SystemC

[21] Lukai Cai ”Transaction Level Modeling in System Level Design”

[22] ”UART: Technical Reference Manual”, ARM Limited.

[23] ”DMAC: Technical Reference Manual”, ARM Limited.

[24] ”VIC: Technical Reference Manual”, ARM Limited.