

Using Top-Down System Simulation (TDSS) to Develop a Wireless LAN System.

C. Michael Costa, Deputy Director, Advanced Business Development
Andrew Reynolds, System Analyst
Protocol, The Services Division of Zycad Corporation
100 Enterprise Drive, Suite 500
Rockaway, New Jersey 07866
(201) 366-8100

Abstract

Top-Down System Simulation (TDSS) is an effective means of defining, designing and verifying the requirements and capabilities of large/complex electronic systems. TDSS follows top-down analysis and design by allowing models of the system to be developed and simulated during each phase of the product development life cycle. Each model adds detail to the previous one and reflects a non-ambiguous specification of the system throughout the complete product development lifecycle.

This document will discuss the use of TDSS in the development of a wireless LAN system. The intention here is to provide the reader with a detailed understanding of the TDSS Methodology using the development of a wireless LAN system, as an example. As such, this document provides examples of the trade-offs and issues associated with applying TDSS to the development of a wireless LAN system.

1. THE NEED FOR SYSTEM SIMULATION

Time-to-market, product development cost and product quality issues are growing rapidly. In order to be a world-class electronic systems manufacturer in today's competitive world, companies need to couple smarter processes with proven design automation technology.

It is important throughout the development process that engineers consider the system in which their component is to be integrated. Normal development processes partition the design into hardware and software subsystems. Design engineers then proceed on their own and develop implementations for the individual subsystems they are responsible for, in too many cases oblivious to the system context. There may be several meetings (reviews) where the interfacing between hardware and software and the different hardware subsystems is discussed, but the complexity of the system being designed is just too complex to address within traditional forums.

The integration of hardware and software is not conducted until prototype hardware is available. At this point in the design it is very expensive to modify the hardware so software work-arounds are developed. This lowers the quality of the software and increases its complexity. For a number of reasons, integration of the different hardware subsystems is often unsuccessful as well. ASICs do not work on the circuit boards they were designed to work with and the costs associated with correcting these problems late in the design process is expensive. As Figure 1. below shows, the costs associated with correcting product flaws increases by an order of magnitude for each phase the development process advances. It is most economical to isolate and correct flaws during the design phase.

A mechanism for reflecting the design specification in an unambiguous form and a methodology for incrementally decomposing and continuously verifying specification compliance is needed to assure system design success. System simulation provides the necessary focus on the integration of hardware and software components in the resulting system. Problems in the design can be found earlier in the development when it is less expensive to correct them.

When one factors in costs, resource efficiency, product quality, customer perception, time-to-market and all the other related issues associated with successful products, system simulation forms a relatively low-cost, but sound foundation for continuous success.

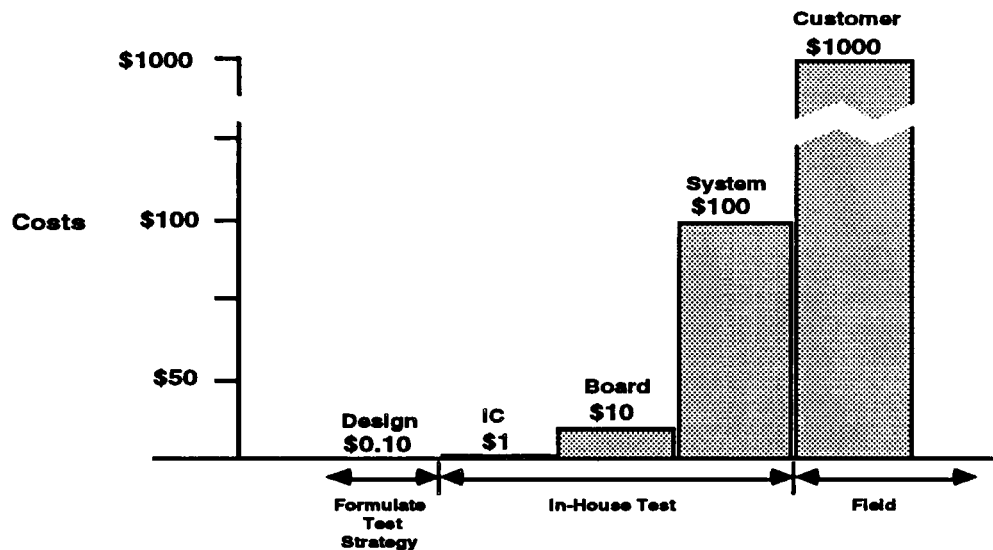


Figure 1. — Cost to Correct Errors

2. SYSTEM SIMULATION PROCESS

The following are the key traits of top-down design using system simulation:

1. Start system simulation at the highest level possible and continue incrementally downward,
2. Integrate system software and models of the hardware as early as possible,
3. Choose tools carefully to make the process "seamless",
4. Use fault simulation to verify test process against all possible manufacturing defects
5. Properly balance amount of modeling and simulation performed with time to market pressures,
6. Actively manage the effort.

There are different modeling techniques that are applicable during each phase of the top-down design process. The following paragraphs will describe the models and their use during each phase.

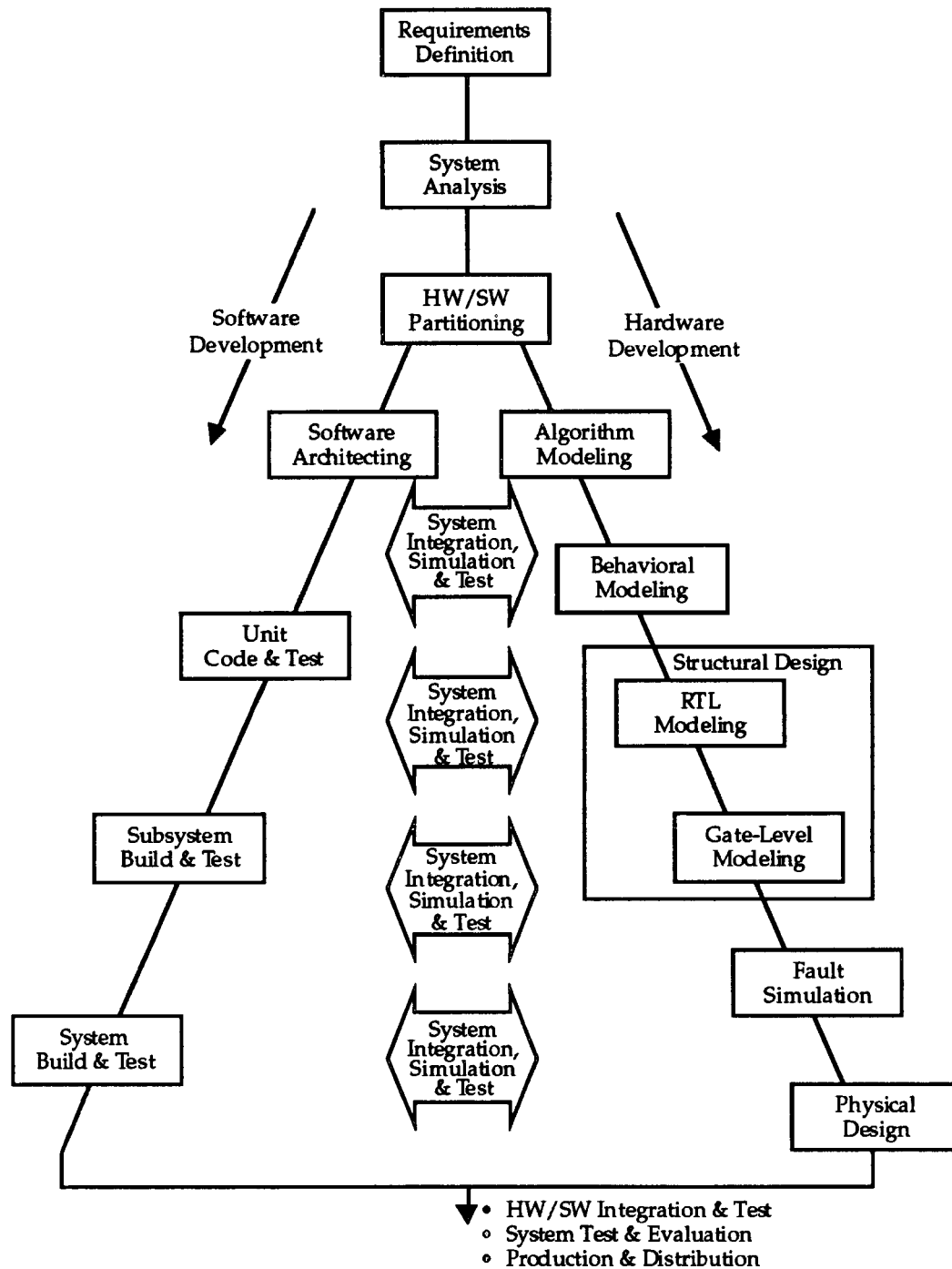


Figure 2. — Top-Down System Simulation (TDSS)

2.1. Specification Development

Specification development occurs during the system analysis phase of the top-down design process. Specifications, like the system, are developed hierarchically and follow the functional decomposition of the system.

In the development of a wireless LAN system, specifications are developed for the operational, power, throughput, frequency spectrum, and topology requirements. In developing these requirements many issues lack sufficient information with which to completely define the requirements. To assist in completing these requirements various types and abstractions of models of the system may be developed. At higher levels of abstraction, these models can serve as "simulatable specifications" used to drive detailed (or structural) design. In order to support the structural decomposition process, tests need to be developed to ascertain that system requirements are being adhered to. A common way to achieve this objective is the development of a "test bench." The test bench includes a Device Under Test (DUT), a configuration of models representing all or most of the system, the simulator stimulus used to test for specification compliance, and a listing of expected results.

The following paragraphs will describe each model type, their roles and the development and application of test benches.

2.2. Behavioral Modeling

Behavioral modeling consists of three different model types — Algorithm, Performance and Architectural. Each behavioral model type represents a different aspect of the hardware system specification. The Algorithm model represents system function, the Performance model represents performance (e.g., through-put, latency, data rates, etc.) of the specification, and the Architectural model represents functional partitioning. In some circumstances, combinations of the behavioral modeling types can be combined within a single model. It is important to note that all three behavioral model types may not be needed in all circumstances. In general, the need to address each modeling type is proportionate with the system's algorithmic, performance and architectural complexity. If our wireless LAN system operated at FDDI speeds but the protocol processing and overall system architecture was simple, there might be a need to develop a performance model to ascertain that the system can process the data fast enough. However, because the system is not algorithmically or architecturally complex, there may be no need to develop algorithm or architectural models.

Behavioral models are useful to facilitate hardware prototype verification and as a mechanism to support early integration of hardware and software subsystems. This early hardware/software integration provides early visibility into system architecture, verification of system algorithms, and supports the development and verification of boot code, diagnostics and some of the application software. The benefit to using a behavioral model to execute the code versus an instruction set simulator is the presence of the other hardware components with which the code and processor interact. The software can be executed on the "virtual hardware" system before it has been fabricated. In the case of our wireless LAN system, a behavioral model could be used to process datagrams through the system's transceivers. In this case, the datagrams would serve as the simulator stimulus and the processing algorithms could be implemented with a combination of hardware and firmware (embedded software).

The following subsections provide further detail about each of the behavioral modeling types.

2.2.1. Algorithm Modeling

The algorithm model is an implementation of the functional requirements documented within the system specification. Its intent is to ensure that the functions of the system will work in concert with each other or within a given set of constraints (accuracy, clock rate).

For instance, within a wireless LAN system, an algorithm model could be used to determine how many parity bits (generated by hamming code) are needed to detect and correct bits in a packet given the worst case bit-error rate.

An algorithm model is developed using sequentially executed, procedural style code. The algorithm is implemented with a computer programming or hardware description language. The algorithm model does not address implicit or explicit structure, functional partitioning or correlation with a component level implementation.

2.2.2. Performance Modeling

Special constructs may be added to the model to get performance information needed to characterize the specifications. This information could be used to determine the following characteristics of our wireless LAN system:

- data rates (bit error rate, throughput),
- topology (transmission range, server interconnection), and
- processing load (directed at servers).

Consider the algorithm model used to determine the number of parity bits needed for detecting and correcting damaged packets. The performance model may address the process load needed to perform such operation or whether it would be a more efficient implementation to retransmit a damaged package based on a checksum (or other error detection scheme).

2.2.3. Architectural Modeling

Architectural modeling is performed to facilitate partitioning of the system into distinct functional components. This is where standard and custom components are identified and configured. Standard component models are pre-manufactured parts available from other suppliers and custom components are programmable devices (FPGA, PALs, etc.) or Application-Specific Integrated Circuits (ASICs). Standard component models can usually be purchased, in any number of forms, while custom components models usually need to be developed, preferably in bus functional form.

The following sections present additional detail about system partitioning, standard component models and bus functional models.

2.2.3.1. System Partitioning

The partitioning process usually starts with an architectural model. This process begins during the system preliminary design. The initial system algorithm model is decomposed into functional components. The system designer configures a number of different architectures, simulates each one to evaluate trade-offs and selects the best architecture to implement. This process is best done using an architectural model where parameters can be quickly modified to account for implementation as either hardware or software and then simulated. This method assumes that all software functions are modeled with parameters that account for applicable operating system interaction.

A wireless LAN system is typically composed of several functional elements such as the communications network, processor, mass storage (on- and off-line), operating system, and application programs. In the communications network there may be a mix of wired and wireless network components. An optimal configuration of network components can be identified through architectural modeling.

2.2.3.2. Standard Component and Bus Functional Modeling

As the system is decomposed into functional elements, the system designer identifies pre-existing, off-the-shelf (standard) components that can be used to implement specific system functions. Building a system level architectural model requires models of these components. Models are also needed for any custom components used in the system. In order to quickly understand how standard components and software interact with the system's custom components, the custom components of an architectural model need only address critical interface functions. Implementing more internal custom component functionality at this point is usually not necessary and would only serve to waste time. These interface functions include signal activity visible from the external pins of the custom component. This type of model is commonly referred to as a "bus functional behavioral model." Later, when it is decided that a custom component will either be used or further information is needed to make this decision, additional functionality will be added until the custom component model ultimately includes all of its intended functionality. This model is commonly referred to as a "full functional behavioral model." The full functional behavioral model will be discussed later.

Standard component models are available in a number of forms from a number of sources. Commercial merchants supply standard component models in bus functional, full functional, gate-level, and hardware modeling form. A hardware modeler is a specialized device in which the actual, manufactured component is placed, representing that component within a simulation. Gate-level models are available for use with hardware simulation accelerators. Hardware accelerators are specialized supercomputers used to greatly accelerate simulator execution time. If a standard component model is not available, trade-offs need to be conducted to arrive at another solution. In some cases, a bus functional model of the standard component may need to be developed.

Commercial merchants usually provide various standard component models used in wireless LAN systems.

2.3. Testbench Development

Once the behavioral modeling process starts, test benches can be developed to verify that system requirements are being met. After the behavioral modeling process is complete, the resulting suite of models and test benches can be used to drive structural design and verify that the incrementally decomposed design continues to meet system specification requirements. By providing the test benches early in the development, structural design engineers can be confident that the requirements are being met throughout the remainder of the development process.

Test benches provide an invaluable mechanism for testing complex systems. The test bench can be used to verify conditions which actual test stands (board and ASIC testers) may not be capable of supporting (reliably, if at all). It is this one benefit that makes system simulation a key to reducing risk.

Test benches can also be used to facilitate hardware verification, comparing design intent as represented by a detailed, structural model with its manufactured results. In order to support hardware verification, however, the test bench must rigidly adhere to applicable tester rules in order to ascertain appropriate DUT stimulation and observation.

2.4. Register Transfer Level (RTL) Modeling

Once a final algorithmic, performance and architectural configuration has been selected, the hardware and software engineers proceed to develop a structural design. Structural models commonly include register transfer level and gate-level models. Structural modeling is usually only needed for custom components. With today's synthesis tools, the need to manually generate a design at the gate-level has almost disappeared. When gate-level design is performed, it is almost always accomplished using an ASIC vendor's macro library. Much of the gate-level design task is to simply assemble applicable macros into a configuration which yields the correct design functionality. This paper assumes the use of synthesis tools for custom component development.

Through step-wise refinement, the behavioral models of custom components may be further decomposed to a level where synthesis can be applied. RTL models are commonly used as input to today's synthesis tools. RTL models should be developed for programmable components and ASICs. By the end of the preliminary design phase, the model begins to resemble what the completed product will look like.

Some portions of the model remain behavioral because they consist of analog, standard component or "low-risk" circuitry. In the wireless LAN, the radio frequency (RF) portion of the transceiver would remain behavioral. The RF component would model the signal propagation, bit error rate, and timing. A/D converters would be modeled functionally at the behavioral level using real number (analog) signals and modulo arithmetic functions to produce binary, octal or hexadecimal (digital) signals. "Low-risk" circuitry is functionality that is either already well-defined at the structural level (ASIC vendor core macro or megafunction), not complex enough to warrant decomposition (e.g., a memory), or pre-existing circuitry.

Once the RTL models have been developed, the system test bench should be executed using the RTL models in place of the applicable behavioral versions. If the results of the new test bench configuration are sufficiently identical to the original test bench results, the system is considered to be in compliance with system requirements.

As the hardware system is incrementally decomposed, software should be run on the system model so that it can be verified before continuing to the next level of design. However, only a minimal amount of diagnostics and application software can be executed in a reasonable amount of time (less than a day) due to software simulator speed (approx. 0.25 machine cycles per second). Simulator performance will continue to decline as the design transforms from behavioral to gate-level. A hardware simulation accelerator can help boost simulation performance. Current accelerators support RTL (using synthesis software) or gate-level models. By the third calendar quarter of 1993, accelerators will be available to boost behavioral, RTL and gate-level simulation performance.

2.5. Execute Application Software on Multi-Level System Model

By the end of the critical design phase both the hardware and software comprise fairly accurate representations of the system. The hardware is represented by a mixed-level model consisting of all the components. A mixed-level model is one where components are modeled at all levels of abstraction (behavioral, RTL and gate-level) or are implemented using different modeling technologies (software, hardware and accelerated).

The software system developed may now be executed on a system model that accurately reflects the system being built. With simulation accelerators, emulators, and networked software simulators, a sufficient amount of code can be executed to verify the functionality of the applications and operating system (if any).

This step is extremely valuable since the interaction of the hardware and software tends to be the major area of problems once integration begins. With system simulation the integration occurs much earlier and continues throughout the remainder of the development.

2.6. Scan insertion, ATPG and Fault Simulation

The topic of "Design For Testability" (DFT) is beyond the scope of this paper, but this subject should be considered a very important aspect of product development. We will devote three paragraphs to the topic of test, before returning to our TDSS discussion. DFT, in overly simplistic terms, is the designing-in of testability. That is, to undertake the design process such that all nodes and devices of the system are controllable and observable from the system's external signal boundaries (e.g., connector pins). DFT, therefore, facilitates post-manufacturing test to ascertain that the product meets requirements and it also supports field test, problem isolation and repair.

Various trade-offs are usually made to justify the return-on-investment associated with supporting complete system testability. All too often, the costs associated with complete system testability are deemed prohibitive when compared to the return. In many cases, a more complete accounting of product life cycle costs would clearly justify the development of systems with far greater testability. It is usually very easy to justify DFT for custom components, particularly ASICs, because the developer is usually personally responsible for correct operation of fabricated components. The only guarantee an ASIC foundry is often willing to make is that the tests the developer supplies will pass. The large non-recurring engineering fees associated with today's large ASICs make clear the advantage of complete and thorough tests.

A number of schemes are known to improve ASIC testability. The more often used schemes include full SCAN, partial SCAN, LSSD, T-Cells and BIST. Each scheme has its advantages and disadvantages. The most popular scheme is called SCAN. The SCAN scheme involves the use of special storage elements (e.g., registers) that support the clocking of a serial test pattern (for circuit control) and sampling the internal state of these elements at certain clock boundaries (for circuit observation). The SCAN storage elements are connected to form a serial SCAN chain, complete with a single clock signal, data signal and control signal. The difference between partial and full SCAN is the degree to which this scheme is applied. In partial SCAN, only those storage elements that are not inherently controllable and observable are converted to SCAN type elements. In the full SCAN approach all storage elements are converted to the SCAN type and connected in a chain.

With that said, let's return to our TDSS discussion. Once the functionality of the custom component structural models has been tested for correct function and performance, fault simulation is performed to grade the fault coverage of the existing test vectors. The fault coverage provides the designer with a measure indicating the degree to which the test stimulus can be used to test for manufacturing defects. A fault coverage of 100% indicates that the design is completely observable and controllable using existing test stimulus. In many cases, the design needs to be altered to increase observability and controllability (collectively referred to as "visibility") and the test stimulus needs to be augmented to increase their fault coverage. One way to increase design visibility is to insert scan logic. Unlike the simulation realm, a manufactured part does not allow unlimited visibility into the internal nodes and devices within the part — visibility is restricted to the external signal boundaries. Many libraries consist of parts with and without scan logic. However, when the system is initially designed, the ability to test the manufactured component should be considered and design in (DFT).

Our discussions of DFT tend to be applicable within the digital electronics realm. Applying SCAN design principles with analog electronics is not possible. In a wireless LAN system, testability of the RF circuitry would need to be addressed in other ways (e.g., feedback circuitry). The concept of increasing design visibility to enhance testability is, however, universal because

the problems associated with stimulating and verifying circuitry from external boundaries (e.g., component pins and board connectors) still apply.

Depending on the foundry, an Automatic Test Pattern Generation (ATPG) tool may exist. This is extremely important for generating compact, efficient vectors for detecting manufacturing flaws. These vectors are generated for the structural components to get the maximum fault coverage. The combination of design alteration (e.g., inserting SCAN) and developing additional fault test vectors (e.g., using ATPG tools) usually suffice to increase the fault coverage to acceptable levels. In many cases, however, changing the design has functional, timing and/or performance ramifications — this is one of the reasons why testability should be designed-in using DFT techniques.

2.7. ASIC and PCB Layout

Once fault coverage has reached adequate levels, physical CAD tools are used to perform floor-planning, block placement, layout, routing, packaging, wire bonding, parasitic delay calculation, design rule checking, electrical rule checking, etc.

The information generated by physical CAD tools is back-annotated into the model used to direct manufacturing (e.g., the lowest level model used prior to manufacturing) so that it reflects the attributes of the post-layout model accurately. The functional test suite is then simulated to verify the layout did not induce problems into the circuit. If so, the problems would be corrected, all simulations re-run and layout performed again on the affected parts of the system.

Once the model passes functional tests using post-layout data, the model is released to the foundry for manufacturing. During manufacturing, the test suite (functional and fault) is used to test the finished ASICs and/or PCBs. After the system has been distributed to the customer, the design models and test suites can be used to isolate problems and/or verify design changes.

This concludes the TDSS discussion. The rest of the process lies in the physical world.

3. TDSS PROCESS MANAGEMENT FLOW

Top-down design with system simulation allows for the management of next generation system design. It establishes definitive results that are more visible by management and simplifies measurement and verification of development progress.

TDSS helps rectify design issues in-line with development of the system. It enables team design where portions of the system can be developed by many groups or companies. It also enables verification of specification compliance before fabrication.

TDSS facilitates total electronic system quality and improves the development and manufacturing process. Figure 3. below shows the relationship between the various phases of the TDSS process and associated management verification points (reviews). It is important to note that the System Integration, Simulation & Test steps can be undertaken independent of the reviews. PDR stands for Preliminary Design Review and CDR stands for Critical Design Review.

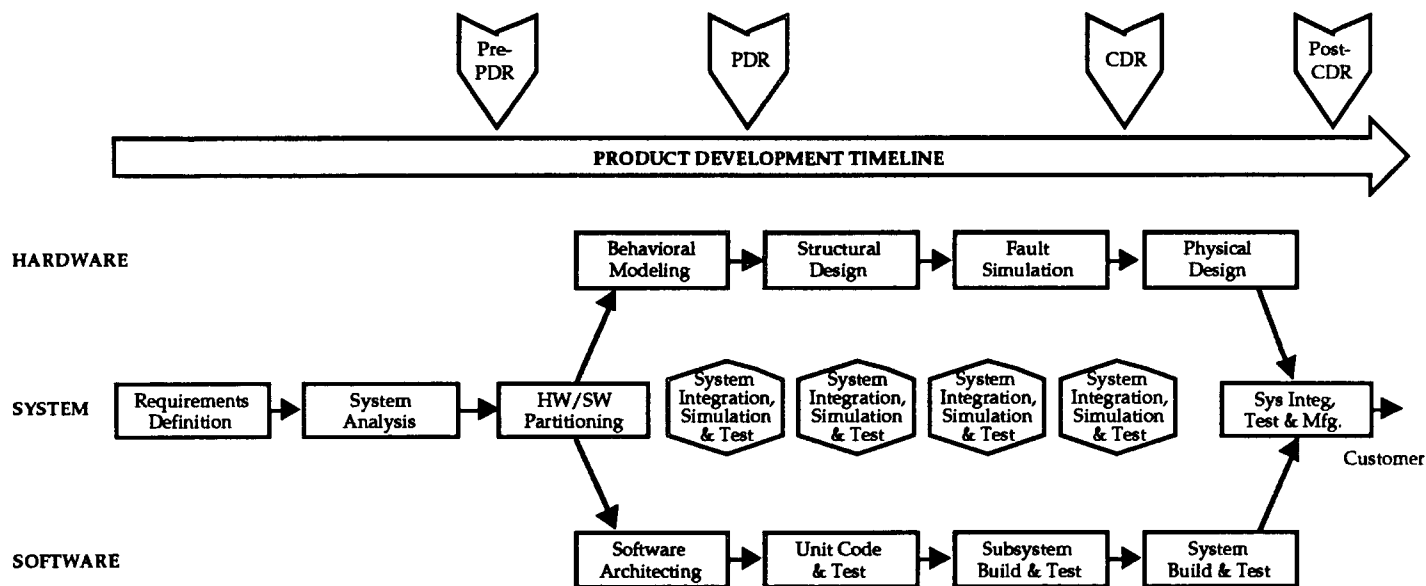


Figure 3. — TDSS Management Task/Time Line

Management is provided various points along the development process in which progress can be measured. The following describes some of the information that management can use to measure progress at each stage of the process.

3.1. Pre-Preliminary Design Review (PRE-PDR)

- Architectural/Performance analysis
- Architectural & performance models
- Algorithm modeling and simulation
- Machine executable specifications

3.2. PDR

- Full functional behavioral models of components, ASICs and PCBs
- PCB and multi-PCB behavioral simulations
- Software execution on behavioral models of hardware

3.3. Critical Design Review (CDR)

- Full functional gate-level models of components, ASICs and PCBs
- PCB and multi-PCB simulations
- Software execution on pre-manufacturing level (structural and behavioral) models of hardware
- Fault grading
- Diagnostic and Fault Simulation
- Test pattern generation

3.4. Post-CDR

- Simulations of system integration and software execution
- Hardware verification to specification
- Hardware integration support
- Manufacturing support
- Simulation of design for test support circuitry
- Upgrade analysis, design and development
- Hardware and software maintenance

4. CONCLUSIONS

TDSS is a process that uses readily available Electronic Design Automation (EDA) technology to greatly improve product quality, development and manufacturing efficiency, and time-to-market. The expertise to establish and maintain a TDSS environment is not difficult to build. With organizational dedication and commitment, TDSS has helped electronic systems manufacturers become and remain successful. In fact, many of the most successful manufacturers competing in the world electronics systems market today use the TDSS process in some measure — and they continue to expand its use.

We have attempted to apply the TDSS methodology to a generic wireless LAN system development project. In so doing, we have provided examples of how certain aspects of this methodology could be applied. Being more specific than that is not possible without detailed knowledge of the requirements associated with a particular system development project. This is because different systems possess different characteristics and associated risks. The TDSS methodology, therefore, needs to be tuned or optimized for each application.

Applying this methodology to the development and support of any specific project requires an analysis of the requirements of the system and customizing baseline TDSS principles to adequately address each design's specific requirements. The methodology described above is thorough but it is important to note that every step of the process is NOT needed in all cases. The particular requirements associated with each product development project should be analyzed carefully and used to develop a TDSS methodology that adds emphasis where risks are likely to exist and reduce emphasis where risks do not exist. For instance, the decision on whether or not a RTL model is needed to add accuracy to a subsystem behavioral model can only be made if you first understand the value the added accuracy provides in comparison with the cost of developing and supporting the RTL model. For these reasons, each product development project should include a TDSS Plan.

The most pervasive themes of TDSS are: early and complete adoption; methodical design decomposition; early discovery and resolution of problems; and predictable results. Companies that adopt this process today are fortunate that others have proven the value of this technology. In many cases, most companies already have a majority of the required EDA tools. The remaining ingredients are commitment and determination.