THE VLSI 'SILICON COMPILER' DESIGN PROCESS

R.M.P.West

IBM United Kingdom Laboratories Ltd
Hursley Park,
Winchester,
England SO21 2JN

Abstract

This paper describes a VLSI design process developed by a small team of logic designers and design tools specialists at the IBM Laboratory at Hursley in England. The design process consists of a large number of programs or design tools, many obtained from other IBM laboratories. Some of the "imported" programs were adapted and several other programs were specifically created. The design process has been used successfully to produce "first-time-right" VLSI integrated circuits (ICs) for several IBM projects, including IBM's Image Adapter/A.

The described design process may be summarised as follows:
A design, to be implemented in VLSI hardware, is partitioned into a number of smaller units. The design of each smaller unit is entered as text in a High Level Language as a technology independent behavioural description, which is then compiled for input to logic synthesis, where it is converted into an optimised design in the target VLSI technology. When all the smaller units to be integrated into a single VLSI IC have undergone this process, they are assembled together by a further run of logic synthesis. The resulting IC design, in the target technology, is then processed to produce the artwork and test patterns required for manufacture of the IC. In support of all these activities, a number of programs exist to allow the design team to perform simulation, timing analysis, and validation of the logic synthesis process.

Introduction

The IBM Laboratory at Hursley in England has amongst its responsibilities the development of display adapters and monitors for the IBM Personal System/2 product family.

Display adapters are complex logic and memory systems that are able to update text, graphics, and image data in a display buffer memory. They also control the refreshing of the display data from the display buffer memory to the electron guns of a cathode ray tube (CRT) or other display devices. Display adapters require a large amount of complex high-speed logic to perform the required functions. The size and complexity of this logic, together with commercial pressure to introduce new products rapidly, demand a logic design process that allows rapid, efficient, adaptable, and accurate design of application specific VLSI ICs.

This paper describes such a VLSI design process, also termed a methodology, that has been developed to satisfy these requirements. It begins with a description of the entire design cycle, from the point of view of a design team using the methodology. This is used to set the context, within the design methodology, for the various features of the methodology described in more detail later in the paper.

Description of the Design Cycle

The overall function of the hardware of the system being designed is partitioned into a number of Functional Islands (FIs), with the interfaces between the FIs clearly defined. If the overall function must be implemented using more than one IC, a decision on the partitioning between ICs can be made at the start or can be delayed until later in the design cycle. The FIs are constrained to a "manageable" size, so that a single designer can work on one or more FIs, and that a single FI can be handled rapidly by the design tools. The interfaces of each FI, in terms of its inputs and outputs, are listed in a FI Inter-Connect File (ICF).

A designer uses a text editor at a computer terminal to create the design of a given FI as a technology-independent behavioural description of the design expressed in text form using the syntax of a High Level Language (HLL). The HLL text generated is the HLL source for the FI, and together with the ICF for the FI forms the complete design source data for the FI. The HLL source for each FI may be compiled either to a behavioural model (MDL) for behavioural simulation or to a Technology-Independent Database (TID) for processing by logic synthesis.

At an early stage, a designer may choose to perform behavioural simulation on a single FI, in isolation, using simple simulation test-cases. This early simulation of a single FI is often used as an integral part of the development of the FI design. When the design of several FIs reaches a given level of functionality, the design team may begin behavioural simulation on the system or a subset thereof. Several FIs are simulated together, as a single functional unit, for verification of system or sub-system function and to check the FI connectivity. The design team usually attempts to reach this stage early in the design cycle, to obtain early verification of system functionality.

For system behavioural simulation, the FIs comprising the system are listed in a Hierarchy Content File (HCF). The HCF is used as input to a program (CREATE) that generates a system level ICF by linking together all the required ICFs. The system level ICF is then used in behavioural simulation to link together the MDLs for the FIs.

In order to perform logic synthesis on the FI, the HLL source for the FI is compiled into a Technology-Independent Database (TID) form. The TID for the FI is then processed through logic synthesis, where a number of transforms are used to convert and optimise the design to create a Technology-Dependent Database (TDD) for the FI in the target technology. Transforms, within logic synthesis, are also used to create control data for use in any subsequent timing analysis.

The designer will often run synthesis on the FI early in the design cycle, in order to obtain a sizing for the FI in the target technology. The designer may also run timing analysis and design rules checking on the TDD for the FI, in isolation, as an early warning of internal timing problems and design rule violations. At this stage, the designer may also run a synthesis validation program, in order to verify that the transforms, within logic synthesis, have not corrupted the design of the FI.

When all the FIs for a single IC have been synthesised to TDDs, logic synthesis is used to link the TDDs according to the FI interfaces defined in a system level ICF for the IC. The system level ICF for the IC is generated by the CREATE program that links together the ICFs of all the FIs of the IC, as listed in an IC level HCF. A different set of transforms, within logic synthesis, is used to create a single TDD for the entire IC and the control data to be used in timing analysis. At this stage, technology-dependent timing analysis and design rules checking are run on the TDD for the entire IC. Any timing or design rule faults found are related to offending FIs, and necessitate updates to the HLL sources for those FIs. System behavioural simulation is performed to ensure that the required function has not been compromised by any update. Updated FIs are resynthesised and the TDD for the IC is rebuilt, so that timing analysis and design rules checking can be rerun.

As the design cycle proceeds, the design team spend less time on actual design and more time on simulation until, near the end, the entire effort of the design team is concentrated on simulation of the complete system. Behavioural simulation is used to verify the overall system function and is used extensively because of its speed and the large amount of simulation test data that can be handled.

When a TDD exists for an entire IC, the TDD can be compiled into a gate-level simulation model (GLM), which incorporates estimated technology-dependent gate delay information and timing rules. Gate-level simulation is necessarily slower than behavioural simulation, and therefore less simulation test data can be handled. The design team only performs gate-level simulation late in the design cycle, where it is used as a cross-check against behavioural simulation, as additional verification of the synthesis process, and to warn of any timing problems missed through assumptions made in timing analysis.

When the design team is satisfied with the performance and functionality of an entire IC, automatic test pattern generation and the physical design process can begin. The design team often attempts to reach this stage early in the design cycle, in order to obtain early warning of any problems, with timing, testability, and wireabilility of the design that might occur in the latter stages of the design process. Automatic test pattern generation and the physical design process can be run as parallel or independent activities. However, for the final generation of data to be sent to the IC manufacturing location, automatic test pattern generation must be run after the physical design process has been completed.

In the physical design process, the TDD for the IC is processed by automatic placement and wiring programs as the first step towards the creation of the physical mask artwork required for the manufacture of the IC. The timing analysis program can be used to generate a list of timing critical signals, with capacitance targets that the placement and wiring programs will attempt to meet. The design team may choose to modify the list of critical signals, prior to placement and wiring. The placement and wiring programs also allow a limited amount of manual intervention.

After placement and wiring, the capacitance and resistance values for all signal wiring are extracted. The extracted data is used to develop delay data for the IC. The delay data is used in timing analysis to ensure that no timing problems have been created by the physical design process. The delay data is also incorporated into the GLM, so that gate-level simulation may be rerun with physical delays. If any problems are found at this stage, manual updates can be made to the placement and wiring. If a large number of problems is found, then the list of critical signals is modified and the automatic placement and wiring programs are rerun. When satisfactory placement and wiring for the IC has been achieved, the physical mask artwork for the IC is generated and checked against technology ground rules.

At the end of the design process, the test pattern data and the physical mask artwork data are sent together to the IC manufacturing location.

Design Partitioning

The design of any system requires design partitioning at several different levels. At the highest level, a partition may be defined between system software and system hardware. At the hardware level, the design must be partitioned between elements of the design that may be implemented in "off the shelf" components already available and elements of the design that must be integrated in one or more application specific VLSI ICs. The scope of this paper covers the design of system hardware that must be integrated in one or more application specific VLSI ICs.

The design of system hardware is partitioned into a number of smaller units. These smaller units are termed Functional Islands (FIs). Each FI performs a distinct system function or set of system functions. The FIs are constrained to a "manageable" size, so that a single designer can work on one or more FIs, and that a single FI can be handled rapidly by the design tools. Sometimes it may be considered appropriate for more than one designer to work on a single FI, particularly if one of the designers has limited design experience.

The interfaces of each FI, in terms of its inputs and outputs, are clearly defined in a FI Inter-Connect File (ICF). If the design must be implemented using more than one IC, then each IC will be allocated a number of FIs, subject to a number of considerations including the minimisation of the complexity and size of the inter-IC interfaces, and the physical sizes of the ICs. The decision on the partitioning between ICs can be made at the start or can be delayed until later in the design cycle.

The granularity, created by partitioning the design into FIs, allows each FI to be handled independently and allows the grouping of FIs at a number of different levels. The FIs may be grouped at the entire system level or at any subsystem level, particularly at the level of the subsystem represented by a single IC. The FIs comprising a system or subsystem are listed in a Hierarchy Content File (HCF). HCFs at a number of different levels may be defined. The HCF is used as input to a program (CREATE) that generates a system level ICF by linking together all the required ICFs. System level ICFs are used in behavioural simulation and in logic synthesis.

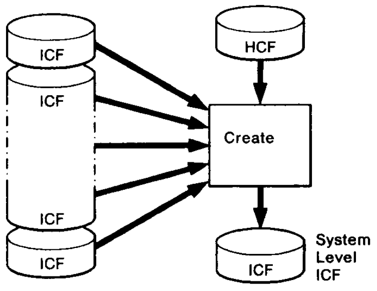

Figure 1. System level ICF generation

During the design cycle a number of different versions of each FI may be produced, and a number of different versions of each HCF may be generated to incorporate different versions of FIs. The different versions of FIs may result from changes in functional requirements and any problems found in simulation or timing analysis. All HLL sources, ICFs, HCFs and MDLs are maintained in a design library system and are made available to the entire design team. In the design library system, different versions of FIs are identified by FI name and version number, and different versions of HCFs and system level ICFs are identified by their system levels and version numbers. This means of identification is retained throughout the design process.

High Level Language Design Entry

The High Level Language (HLL) allows the designer to design systems in list form rather than in block form (as with graphical design entry tools). It provides an extremely fast, efficient, and unambiguous means of design entry. The designer is able to work with bus-wide entities, but can freely use substringing. Because the design is entered as text, comments and in-source documentation can be included (similarly to a programming language).

The text entered is the source specification of the design as a technology-independent behavioural description of the design and may be at a behavioural level or at a hardware-oriented level or a mixture of the two. The entire HLL specification of the design is, in fact, a behavioural description but the distinction between behavioural level and hardware-oriented level is made because many of the more complex HLL behavioural constructs do not have direct one-to-one mappings into VLSI hardware.

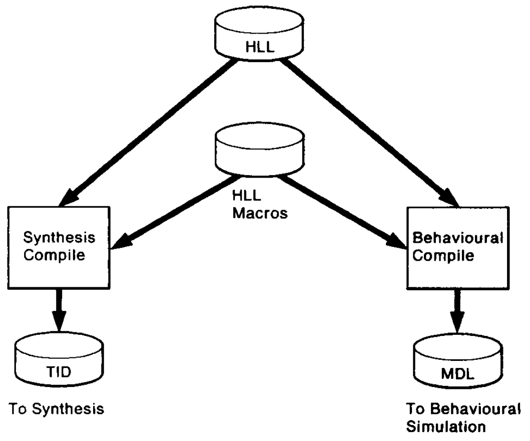

Figure 2. FI HLL compile

The HLL source text for a design may be compiled into a behavioural model (MDL) for behavioural simulation of the design. When the HLL source of the design is specified at the hardware-oriented level, it can be compiled to a TID, which is used as the input to logic synthesis. Because logic synthesis is able to operate on the TID to produce "optimal" logic in a target technology, the designer does not have to concentrate on the design at the gate level. This allows the designer to concentrate on the design of the system at a functional level.

The designer has available a library of standard HLL macros that can be used as shorthand for complex, yet commonly used, HLL constructs. Support for user-defined HLL macros is also provided. The syntax of the HLL also allows the designer to apply attributes to detailed elements of the design, in order to control the effects of logic synthesis and the generation of timing analysis control data. If the designer wishes to override logic synthesis in any area of the design, the HLL provides syntax for the explicit specification of logic gates in any target technology.

Because the HLL specification of a design is essentially technology independent, it can be used for logic synthesis into a number of different technologies. The HLL specification is also "portable" and can be used as part of a number of different ICs, including up-grades and follow-ons of any current IC.

Logic Synthesis

Logic synthesis is normally run on a mainframe system in batch mode. For small designs, however, it may be run interactively. Logic synthesis is an environment set up for the manipulation of a logic design. Within logic synthesis, a number of transforms operate sequentially on a logic design to convert it from one form to another. The transforms used by logic synthesis are available in a transform library. Logic synthesis also has access to technology data, which is integrated into the internal synthesis database, from a set of technology libraries.

The transforms, to be applied to a logic design, are listed in a synthesis control file, which is termed the synthesis "scenario" The design methodology described uses two basic scenarios. The first scenario is the "FI scenario", which converts a TID into a TDD, with the logic optimised for the target technology. The second scenario is the "IC scenario", which links a number of TDDs for the FIs that make up an IC, to create a TDD for the IC.

In special circumstances, it is possible for a designer to modify a synthesis scenario by adding, removing, or altering specific transforms. This is usually undertaken by only the most experienced designers in order to "customise" the effect of logic synthesis for a specific design goal or to experiment with slightly different scenarios on the same design. Any perceived improvements may become permanently incorporated into the standard scenarios. Although the ability to alter the synthesis scenario sometimes yields improved results, it is potentially hazardous and hence the need for a synthesis validation program.

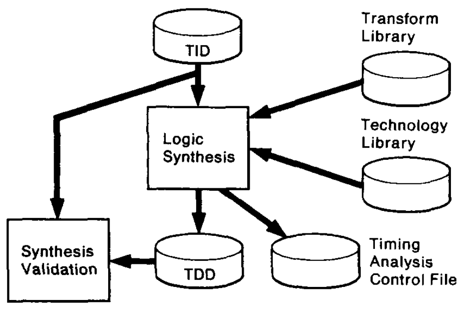

Figure 3. FI Synthesis "FI Scenario".

In the "FI scenario", a TID is read and converted to an internal synthesis database. The internal synthesis database contains a full representation of the design, including any synthesis and timing analysis control attributes from the HLL source. Firstly, a sequence of technology-independent transforms is applied to remove redundant or equivalent logic, to propagate logic constants, and to simplify various logic constructions. A second set of transforms is then applied to optimise the design into a form suitable for efficient implementation of the design in the target technology. These transforms mimic an "expert logic designer" by repetitively applying DeMorgan's Theorem, performing AND/OR/NOT optimisation, combining AND and OR functions into available AND-OR and OR-AND gates, and searching for logic constructs to map into efficient complex technology gates or macros. The goal of these "expert" transforms is to minimise the implementation of the design in the target technology. The design is then converted into the target technology and transforms are applied to repower any signals, whose fanout exceeds technology or design limits. After design minimisation and conversion into the target technology, the scenario may, optionally, include transforms to perform limited timing correction. The timing correction transforms calculate the delays in logic paths and attempt to restructure them to meet predefined timing targets. Finally, the internal synthesis database is output as a TDD.

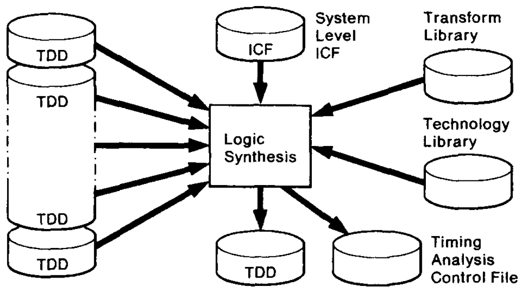

Figure 4. IC Synthesis "IC Scenario".

In the "IC scenario", the system level ICF for the IC is read in and converted to an internal synthesis database. At this stage the database consists of "black boxes" for each FI, interconnected according to the ICF. The next step is to read in the TDDs for each FI and insert them into the "black boxes". In this manner a single design database, containing all the FIs, is created. Because the volume of design data is now very large, only a few simple transforms are applied. These include transforms to repower any signals, whose fanout exceeds technology or design limits, and to repower and balance all clock signals in the design. Finally, the internal synthesis database for the entire IC is output as a TDD.

Within the logic synthesis scenarios, there are a number of transforms that monitor the progress of logic synthesis. These return information such as the number of signals, number and types of logic blocks, the size of the design in the target technology and the number of levels of logic between registers in the design. A number of transforms exist to handle any synthesis control attributes from the HLL source of FIs. These attributes may be used for a number of purposes, including identifying specific types of signals (e.g. clocks), identifying timing critical and non-critical signals, fixing signals at specific points in the logic, forcing specific logic implementations, temporarily hiding portions of logic from specific transforms and altering the rules that transforms may apply to specific logic signals and blocks.

A set of transforms, within logic synthesis, can also be used to create the timing analysis control files used by the separate timing analysis program. The timing analysis control files list all the signals and gates in the design and associate with each timing analysis control attributes. These attributes are either derived directly from the design or as received from the HLL source, via the TID, or both. The attributes include the identification of clock signals and clock gates, "don't care" signals to be excluded from timing analysis, timing adjust data, the cycle times of clock inputs, the arrival times for other inputs to the design, and the expected arrival times for outputs of the design. Where attributes can not be derived and are not supplied in the design database, default values are inserted.

Synthesis Validation

Synthesis Validation (see figure 3) is run on a mainframe system in batch mode. It is used to check for exact logical equivalence between the input to synthesis (usually a TID) and the output of synthesis (usually a TDD). The check ensures that none of the transforms, used in logic synthesis, has corrupted the design in any way.

The design is divided into logic cones, each with a single unique destination point and one or more source points. The destination points are either the outputs of the design or the inputs to registers within the design. The source points are either the inputs of the design or the outputs of registers within the design. Each source and destination point is called a "Stop Point". Extremely large logic cones can produce problems with the synthesis validation process and cause excessively long program run times. In this case, the designer may introduce additional "Stop Points&cdq, at equivalent points in both the TID and the TDD.

The synthesis validation program begins by associating every "Stop Point" in the TID with its equivalent in the TDD, on a one-to-one basis. Any failure at this stage is indicative of a major problem. After the "Stop Point" association process, the synthesis validation program checks for the exact Boolean equivalence of all logic cones. Any failure in Boolean equivalence checking is reported and indicates either a problem within the synthesis validation process or some corruption introduced by a transform in logic synthesis.

Synthesis validation is generally run when a new or updated transform has been used in logic synthesis and also as a final check of the logic synthesis used to produce a TDD. Any corruption problems found are traced to specific transforms and the transforms are corrected. In practice, very few such problems have been found.

Timing Analysis

Timing Analysis is run on a mainframe system in either batch or interactive mode. It is intended to give the designer quick, inexpensive timing information as early as possible during the design cycle and also after physical design has been completed. The designer has the choice of running timing analysis with "Worst Case" or "Best Case" timing data.

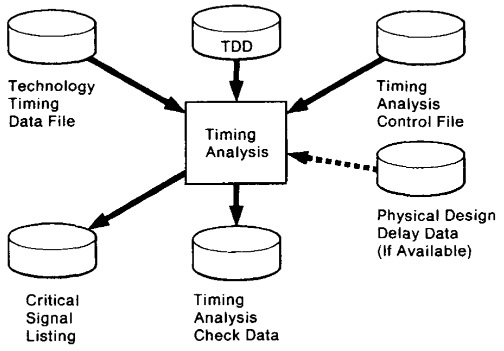
TDD
Technology Timing Data File
Timing Analysis Control File
Timing Analysis
Physical Design Delay Data (If Available)
Critical Signal Listing
Timing Analysis Check Data

Figure 5. Timing Analysis

The timing analysis program is supplied with data for the cycle times of all clock inputs to the design, the arrival times of all other inputs to the design, and the expected arrival times for all outputs of the design. The timing analysis program is also supplied with timing analysis control attributes associated with signals and gates in the design. The attributes include signal timing adjust data and the identification of "don't care" signals to be excluded from timing analysis. Data and attributes are either generated manually, or obtained from the timing analysis control files generated by logic synthesis, or allowed to take on default values.

The timing analysis program first calculates the input-to-output delays for all gates in the TDD. Delays are calculated for both logical polarities of output signal. The delay of a gate is a function of gate type, signal transition times, and the output load of the gate. Prior to physical design, the output loads are estimated as a function of output signal fanout. After physical design, the outputs loads are calculated as a function of actual output signal wiring. With the output loads determined, the gate delays and signal transition times are calculated using data supplied in a timing data file for the technology. If any signal transition times exceed a predetermined limit, then the timing analysis program outputs a warning that identifies any affected signals.

The design is divided into logic cones, each with a single unique destination point and one or more source points. The destination points are either the outputs of the design or the clock and data inputs to registers within the design. The source points are either the inputs of the design or the outputs of registers within the design. The timing analysis program uses the gate delay information to calculate the arrival times associated with all destination points in the design. The arrival time at a destination point, from a source point, is the sum of the source arrival time and the delays of all the gates between the source and destination points. For "Worst Case" analysis, the arrival time associated with a destination point is the latest arrival time at that point. For "Best Case" analysis, the arrival time associated with a destination point is the earliest arrival time at that point.

Using the calculated arrival times, the timing analysis program performs a number of checks. It checks that the arrival times for the outputs of the design satisfy the expected output arrival times. At registers, it checks that the data input arrival times satisfy the set-up and hold time requirements with respect to the clock input arrival times. At clock gates, it checks that the arrival times of gating signals, with respect to the clocks, will not result in sporadic clock pulses at the clock gate outputs. If any check fails, then the timing analysis program outputs a warning that identifies any affected signals. If failures occur, then the designer must take the appropriate action to correct the problem.

The timing analysis program can be used to generate a list of timing critical signals, with capacitance targets that the placement and wiring programs will attempt to meet. The timing analysis program generates a listing of all arrival times and timing margins for all the checks it has made. Using the timing margins generated and the timing data file for the target technology, a program utility may be used to generate a list of timing critical signals, with capacitance targets. The program utility works backwards from the timing margins and calculates maximum signal loads that will satisfy the timing requirements. The calculated maximum signal loads are adjusted and distributed along signal paths within the logic. Signals and signal paths that cause timing problems or have low timing margins will be assigned lower maximum capacitance targets. The generation of capacitance targets is only useful if the number and size of any timing problems are small. Otherwise capacitance targets will be set too low for the placement and wiring programs to achieve.

Simulation

Simulation is the means by which a designer or a design team attempts to verify that a design correctly performs its required function. This is achieved by writing simulation test-cases, which provide the stimuli required to simulate functional operation of the design. The simulated response of the design to the applied stimuli is examined to verify that the correct function has been performed. It is the goal of simulation to check every function and combination of functions of the design in response to every possible set of functional stimuli. In practice, for large and complex designs, this goal is unachievable, and the selection of simulation test-cases must be based on "engineering judgement" and experience. Most simulation test-cases are written as "self-checking" test-cases, in that the simulated responses are tested against expected responses by the test-case. If a check fails, the test-case may generate a warning or may suspend the simulation.

The simulation environment provides a number of utilities to examine the results. These include the ability to graphically display signal waveforms internal to the design under simulation. If the simulation is being run interactively on a workstation, then the signal waveform display is continually updated as the simulation proceeds. This interactive display is extremely useful in the early stages of the design cycle, when designers are "debugging" their designs.

Two types of simulation are used in the design process, behavioural simulation and gate-level simulation. Simulation may be run either interactively on workstations or in batch mode on a mainframe. As the design cycle proceeds, the design team spend less time on actual design and more time on simulation until, near the end, the entire effort of the design team is concentrated on simulation of the complete system.

Behavioural simulation is in the form of event-driven simulation. It has no concept of logic timing or delays, unless some time-dependent behaviour has been built into the behavioural model. An event is a change in one or more inputs to the design, most commonly a transition of a clock input. Since a clock is generally a cyclical stream of events, this type of simulation is often termed "Cycle Simulation". When an event occurs, the logical state of the behavioural model is changed to reflect the effects of that event. The new logical state is a function of the previous logical state and the event causing the change of state. Once the new logical state of the behavioural model has been achieved, behavioural simulation processes the next event.

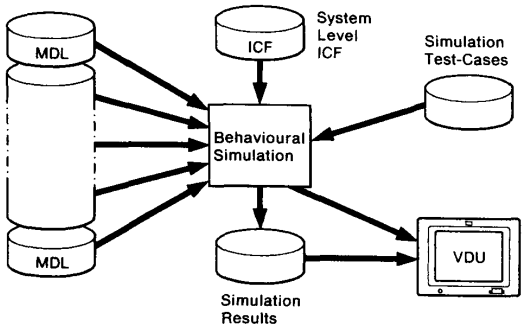

Figure 6. Behavioral Simulation

To perform behavioural simulation on a FI, the HLL source for that FI is compiled to a behavioural model (MDL). If a number of FIs are to be simulated together as a system or a subset thereof, the MDLs for the required FIs are linked together by a system level ICF to form the equivalent of a larger single behavioural model.

In gate-level simulation, each gate of the design is modelled by a behavioural model with the element of time built in. The time element is used to model the input-to-output delays of the gate. Gate-level simulation is event-driven simulation, differing from behavioural simulation in that every gate output is capable of causing an event at some time after an event has occurred at the gate inputs. Between the time of arrival of an event at a gate input and the time of any resulting event at a gate output, an event is said to be "pending" within the gate. An event at one or more of the inputs of the design causes the propagation of a sequence of events, dispersed in time, through the gates of the design. The effects of multiple events are analysed concurrently. The gate models may also contain internal checks that generate warnings based on in-built rules. Warnings may be generated in a number of circumstances including when undefined signal values are encountered, or when events of less than a required minimum duration occur, or when events occur in illegal or undesirable combinations or at times likely to cause hardware problems.

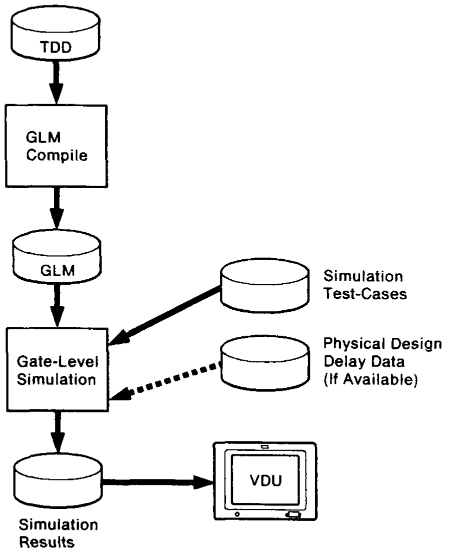

Figure 7. Gate-Level Simulation

To perform gate-level simulation on a design, the TDD for the design must be compiled into a gate-level simulation model (GLM). The GLM for a design incorporates technology-dependent gate delay information and timing rules. The behavioural models and rules for the gates of the design are obtained from a library of technology gate models. Prior to physical design, the gate delays are estimated as a function of output signal fanout. After physical design, the gate delays are calculated as a function of actual output signal wiring. Three types of gate delays are available: "Worst Case", "Best Case", or "Nominal". Gate-level simulation can be performed by selecting any of these three options applied to all gates of the design.

Gate-level simulation is necessarily slower than behavioural simulation. Behavioural simulation is used extensively, throughout the design cycle, because of its speed and, therefore large volumes of simulation test data can be handled. Behavioural simulation is run either on workstations or on a mainframe, and is used to verify the function of a system or any subset thereof. Gate-level simulation is normally run on a mainframe. The design team only performs gate-level simulation late in the design cycle, where it is used as a cross-check against behavioural simulation, as additional verification of the synthesis process, and to warn of any timing problems missed through assumptions made in timing analysis. Because gate-level simulation is slower than behavioural simulation, less simulation test data can be handled. The test data used for gate-level simulation is usually a subset of the simulation test data used in behavioural simulation.

Automatic Test Pattern Generation

Automatic test pattern generation is performed by a program that uses a number of different algorithms to generate test pattern data. The test pattern data is used in the IC manufacturing location to test and verify the fabrication of an IC design, and to separate faulty and fault-free ICs from a manufacturing batch.

In test pattern generation, an IC design is considered to be a large collection of nodes, any one of which may be stuck at either logical one or zero to simulate a fault condition in the fabrication of the IC. This approach is commonly known as "DC Stuck-Fault Testing". To observe a stuck fault within an IC design, the inputs of the logic must be stimulated with a pattern that will make the effect of a stuck fault statically observable at one or more outputs of the IC.

The test pattern generation program uses a number of different algorithms to derive patterns to make stuck faults observable. When a pattern has been derived that makes a stuck fault observable, that stuck fault is said to have been tested. In some cases, a single pattern tests a large number of stuck faults, but in other cases, a large number of patterns may be needed to test a single stuck fault. The derived patterns do not necessarily bear any relation to actual functional patterns, but are developed by the algorithms on the basis of the logic present and the stuck faults being tested.

The test pattern generation program is run incrementally, with run after run being performed until the required number of stuck faults has been tested. The percentage of all the stuck faults, associated with the design, which have been tested is commonly known as the "test coverage", and is considered to be a measure of the "testability" of the design. It is a requirement of the IC manufacturing location that a given level of test coverage be achieved or exceeded. The design team often attempts to start test pattern generation early in the design cycle, in order to obtain early warning of any problems with the testability of the design. Any problems with the testability of the design will necessitate updates to the design, which may involve restructuring the design or providing additional non-functional test paths in the design. For the final generation of data to be sent to the IC manufacturing location, automatic test pattern generation must be run after the physical design process has been completed.

The first run of the program begins with a list of all stuck faults in the design. When the program generates a pattern that makes a stuck fault observable, that stuck fault is removed from the list so that no further attempt is made to make that fault observable. A list of stuck faults remaining and the patterns generated are output at the end of each run, along with the current test coverage figure. Each subsequent run uses the list of remaining stuck faults from the previous run and adds to the patterns generated. In special cases, the program can be used to incorporate and analyse manually generated test patterns.

If the design follows the rules of Level Sensitive Scan Design (LSSD), then the task of test pattern generation is made far easier, since all LSSD registers in the design can be considered to be both inputs and outputs of the design. In a well structured LSSD design, the test pattern generation program will obtain very high test coverage, approaching or even reaching 100% of all stuck faults tested.

Physical Design

Physical design is performed by a number of programs that convert the TDD for an IC into physical mask artwork required to manufacture the IC. The first stage of physical design is the automatic placement program, which places the gates of the IC design onto the IC image. The placement program is the most crucial phase of physical design, since it can have dramatic effects on the wireability and the timing performance of the design. The next stage is the automatic wiring program which attempts to route all the signal wiring between the gates. Any signals which are not successfully routed are wired manually. The next stage is a program which generates the physical mask artwork for the entire IC. The final stage is an artwork checking program which verifies that no violations of the physical ground rules for the technology have occurred.

Automatic Placement

In the first stage of physical design, an automatic placement program places the gates of the IC design onto a grid of legal locations on the IC image. The program takes a heuristic approach to the placement of gates, it continually changes the locations of gates and evaluates its own progress by assigning a score to each unique placement configuration. Initially, the program interchanges gate locations almost at random and slowly reduces the number of interchanges with a strategy to continually improve the score. The score is a function of a number of parameters, including estimates of wiring congestion in any direction, densities of logic gate input/outputs, and estimates of signal wiring length and capacitance. The weighting for any parameter, used to determine the score, may be changed by overriding the default values with a placement control file. The program also attempts to meet any supplied wiring capacitance targets for timing critical signals. It is also possible to confine the placement of specific gates, input receivers and output drivers to either fixed locations or predefined areas of the IC image.

At the end of the automatic placement program, data on the final placement configuration is listed and may be assessed by the design team. From the data, the design team is able to estimate whether the automatic wiring program will be able to successfully route the required signal wiring for the generated placement configuration. By using the estimates of wiring capacitance, it is possible to obtain an estimate of the output loads for all the gates in the design. This output load data can be supplied to the timing analysis program, to obtain a better estimate of the likely timing performance of the design. If the data suggests that any wiring or timing problems are likely to occur, then the automatic placement program may be rerun with revised control data in an attempt to resolve the problems.

Automatic Wiring

In the next stage of physical design, when the design team is satisfied with the placement configuration, an automatic wiring program is used to route the signal wiring between the gates placed on the IC image. The program routes the signal wiring of the design on a wiring grid, compatible with the physical design rules for the technology. The program attempts to route each signal along the shortest possible path on the wiring grid, whilst avoiding areas reserved for the wiring of power supplies and the wiring within the gates of the design. The program also attempts to meet any supplied wiring capacitance targets for timing critical signals. As more and more wires are routed, wiring path conflicts begin to occur. Wiring path conflicts are resolved by choosing alternative wiring paths and, where necessary, rewiring signals that are causing path conflicts. At the end of the wiring program, any signals for which the wiring is incomplete can be wired manually on a graphics screen. Provided that the placement configuration is of good quality, then very few signals will require manual intervention.

When the wiring is complete, the output loads for all the gates in the design can be obtained. The output load data is used to develop delay data, which is then used by the timing analysis program and in gate-level simulation. Thereby the performance of the IC can be analysed with physical delays. If small timing problems are found, the automatic wiring program may be rerun with revised wiring capacitance targets, or the signal wiring can be manually updated. In the unlikely event of major timing problems being found, then manual updates to both the placement and wiring may be attempted, or the entire physical design process may have to be restarted.

Artwork Generation and Checking

In the next phase of physical design, when successful placement and wiring have been achieved, a program is run to generate the physical mask artwork for the IC. The mask artwork for the gates of the design is obtained from physical data in the technology gate library. Signal and power supply wiring is converted from lines on the wiring grid into polygons, whose width and spacing are defined by technology ground rules.

As a final check on the physical design process, a program is run to verify that the generated mask artwork does not violate any physical ground rule for the technology. No ground rule violations should be found since the placement and wiring programs are designed to operate within the technology ground rules.