ASIC Design Methodology Primer

Abstract

This application note provides an overview of the application-specific integrated circuit (ASIC) design process. Four major phases are discussed: design entry and analysis; technology optimization and floorplanning; design verification; and layout.

Introduction

The *ASIC Design Methodology Primer* provides an overview of the steps involved in application specific integrated circuit (ASIC) design. An ASIC is a collection of logic and memory circuits on a single silicon die. ASICs are used in a wide variety of products ranging from consumer products such as video games, digital cameras, automobiles and personal computers, to high-end technology products such as workstations and supercomputers. The ASIC market, with steady growth over the past decade and continued growth predicted for the next one, is expected to become a $50 billion market by the year 2000 (*Dataquest*, 12/16/96).

This primer is organized into three sections:

- The first section, **Basic Terminology**, defines key terms and the scope of this paper.
- The second and largest section, **Basic Methodology Walkthrough**, covers, at a high level, the four major phases of ASIC design, and is illustrated with real design examples. This discussion also identifies some of the major software vendors who offer ASIC design tools, and lists any process steps unique to IBM.
- The last section, **Design Challenges and Strategies** summarizes the specific strengths and capabilities IBM ASICs brings to the marketplace, and their resulting value to its customers.

Basic Terminology

ASICs are logic chips designed by the end-customers to perform a specific function and thereby meet the specific needs of their application. Customers implement their designs in a single silicon die by mapping their functions to a set of predesigned, preverified logic circuits provided by the ASIC vendor. These circuits are referred to as the ASIC vendor's **library**, and are described in the ASIC vendor's **databook**. These circuits range from the simplest functions, such as inverters, NANDs and NORs, flip-flops and latches, to more complex structures such as static memory arrays, adders, counters and phase-lock loops. Recently vendors have added some highly complex circuits to their ASIC libraries, such as microprocessors, Ethernet® functions, and peripheral component interconnect (PCI) controllers. These complex designs are referred to as **cores** and are fast becoming a major differentiator among ASIC vendors.

ASIC Vendor Selection Criteria

An ASIC designer, seeking to create a new design and select an appropriate ASIC vendor, should consider the following criteria:

- ASIC library content and characteristics:
  - Does the library contain the logic circuits needed to implement the design? Are the circuits fast enough? How many can fit on a single die?

ASIC Design Methodology Primer

- Design turn-around-time (TAT):
  - How long does the ASIC vendor take to fabricate, package, and test the part once the design is completed?
- Price of the die:
  - How much does the ASIC cost?
    This is an important factor to all designers, but is more crucial to some customers than others. Those in the consumer market may have this as their number one criteria when evaluating an ASIC vendor, whereas a high-end workstation customer may put performance or function ahead of price.
- Power consumption:
  - How much power does the ASIC consume?
    The importance of power utilization has greatly increased over the past several years, and surpasses the importance of cost in some cases, such as in battery-powered applications like cell phones and lap-top computers.
- Miscellaneous aspects:
  - Packaging options, reliability, supply assurance and second-source capabilities are absolutely critical to some customers, and of secondary importance to others.
- Design methodology.
  - Design methodology is the process that a designer must follow to implement a design in an ASIC vendor's library. The ease with which a designer can execute this process can affect time-to-market, design verification and reliability, and the cost of the overall design process. It is this aspect of the ASIC product, **design methodology**, that is the focus of this primer. This criteria is of importance to all ASIC customers.

Design Views

During the course of the design process, the design data exists in several different formats or views. As the design progresses, it becomes less abstract and more specific to, and optimized for, a particular technology. Each step in the design methodology serves a different purpose and requires unique tools.These views evolve through three major phases:

- In the initial phase the design is realized primarily as a technology-independent Hardware Description Language (HDL), a format very similar to a programming language, to describe the design's functionality.
- In the second phase the design is realized as a technology-dependent netlist that consists of a series of instances of circuits from the ASIC vendor's library, interconnected in a manner to implement the functionality described in the previous view.
- In the last phase the design is realized as a physical view, in which the logic circuits described in the previous view are physically placed on a piece of silicon, called a die, and interconnected by various layers of wiring.

Figure 1 on page 3 depicts these three design views.

HDL View

```
process(CLK)
 begin
   if (CLK = '1') and (not CLK'stable) then
     s_counter_output <= s_counter_input and not s_reset;
     s_ref_ctr_out <= s_load;
     end if;
end process;
```

Netlist View

```
begin
  U68: INVERT_A port map (Z => s_load, A => n265);
  U87 : NOR3_4 port map (Z => n275, A => COUNTb(3),
  B => COUNTb(4), C => COUNTb(0));
  U88 : NOR3_4 port map (Z => n275, A => COUNTb(3),
  B => COUNTb(4), C => COUNTb(0));
  s_ref_ctr_out_reg : D_F_LPH0001_4 port map (L2 =>
  s_ref_ctr_out, D => s_load, E => CLKb);
end SYN_refctr_rtl;
```

Physical View

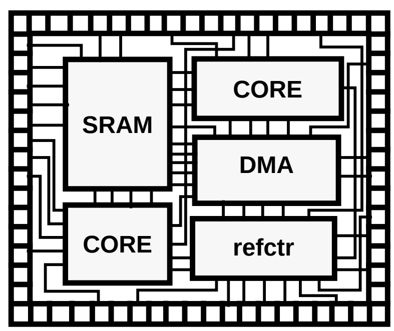

Figure 1. Design Views

Basic Methodology Walkthrough

There are four basic steps that an ASIC design must go through in order to create working silicon:

1. design entry and analysis
2. technology optimization and floorplanning
3. design verification
4. layout

Design Entry

The designer's first task is to describe the design's intended function. Typically this functionality is specified in a document, such as a functional specification, written in a natural language such as English in order to facilitate its development as well as to make it accessible for review by all project team members. Once the specification is finalized, the designer then translates the specification into a form that can be understood by software tools in order to direct the creation of silicon. The two principal design description methods are:

- Hardware Description Languages (HDLs), generally used for designs of 50 thousand gates or more; and,
- Schematic Capture, an older method, suitable only for sub-50k gate designs and generally less often used today.

The two dominant HDLs are Verilog® and VHDL. Both are entered using a text editor such as *vi* on a UNIX®-based workstation. Verilog and VHDL are languages much like programming languages, such as C or Pascal, but they have been designed specifically for describing hardware behavior. Verilog and VHDL are functionally equivalent. The choice of one over the other is driven primarily by the experience base of the design group, the tool set available to the designers to process the HDL, and, possibly, by organizational dictates, such as those of the US government, which requires that all designs be written in VHDL. Verilog dominates the US merchant ASIC market, whereas VHDL prevails in Europe, the US government, and some large US companies such as IBM.

HDLs allow designers to describe the function of their designs at a high level, often independent of the eventual implementation in silicon, much as a programmer can describe a function in the C language without knowing the specific compiler that will create the executable object code.

With schematic capture, graphical representations of the logic functions are placed on a computer screen and are manually connected by the designer. Schematic capture requires the designer to enter a much lower-level description of the design, implemented directly in the logic circuits available from the ASIC vendor, thereby sacrificing the flexibility of the higher-level description possible with HDLs. Schematic capture may still prevail for some time with very small ASICs (10–40k gates) or those containing analog functions. With the average size of an ASIC in the United States in 1996 exceeding 100k gates, the vast majority of customers will be using VHDL or Verilog as their design entry vehicle. (*Dataquest*).

Design Entry Examples

The following sections provide a brief look at some examples of HDL and a simple schematic.

Sample High-Level Hardware Description Language (HDL)

Figure 2 on page 5 contains a portion of a direct memory access (DMA) controller written in two different HDLs: VHDL and Verilog. Notice that though there are syntactical differences between the two languages (for example, VHDL's "entity DMA1..." versus Verilog's "module DMA1..."), the types of language statements and level of description are essentially equivalent. Both HDLs have execution control statements based on the state of a signal called CLK, and both propagate certain design values based on the status of CLK. The language statements are independent of any particular ASIC vendor's library and are at a level of abstraction above any particular logic circuit implementation; for example, such statements might be at a behavioral level or register transfer language (RTL) level. Whatever the level, an HDL can be implemented in several different ways, using different combinations of circuits from any one of a number of different ASIC vendors' libraries.

Sample Schematic

Figure 2 contrasts sharply with Figure 3 on page 5, which provides a schematic representation that is directly mapped into the logic circuits in an ASIC vendor's library. The schematic assembles circuits such as NOR3, AND2 and INVERT and includes explicit connections between inputs and outputs.The logic circuit implementation for this function is completely defined. Because the vast majority of ASIC designs done at IBM begin with an HDL description rather than schematic entry, this paper focuses primarily on HDL in the analysis phase.

DMA Controller

VHDL
RTL-level description

```
entity DMA1 is
  port(CLK         : IN STD_LOGIC;
       RESET       : IN STD_LOGIC;
                  ...
       FIFO_RESTART: BUFFER STD_LOGIC;
                     ...
--*process to create latches
architecture DATAFLOW of DMA1 is
   process
     begin
        wait until (CLK'EVENT and CLK='1');
        OUT_END1_L2 <= OUT_END1_SIG;
        OUT_END1_L1L2 <= OUT_END1_L2;
        OUT_END2_L2 <= OUT_END2_SIG;
        OUT_END2_L1L2 <= OUT_END2_L2;
           ...
     end process;
```

Verilog
RTL-level description

```
module DMA1(CLK, REST, FIFO_RESTART,...)
     input CLK;
     input RESET;
       ...
     output FIFO_RESTART;

//* process to create latches

 always

   begin : block_578
    @ (posedge CLK);
    OUT_END1_L2 <= OUT_END1_SIG;
    OUT_END1_L1L2 <= OUT_END1_L2;
    OUT_END2_L2 <= OUT_END2_SIG;
    OUT_END2_L1L2 <= OUT_END2_L2;
      ...
 endmodule;
```

Figure 2. DMA Controller with Two Different HDLs

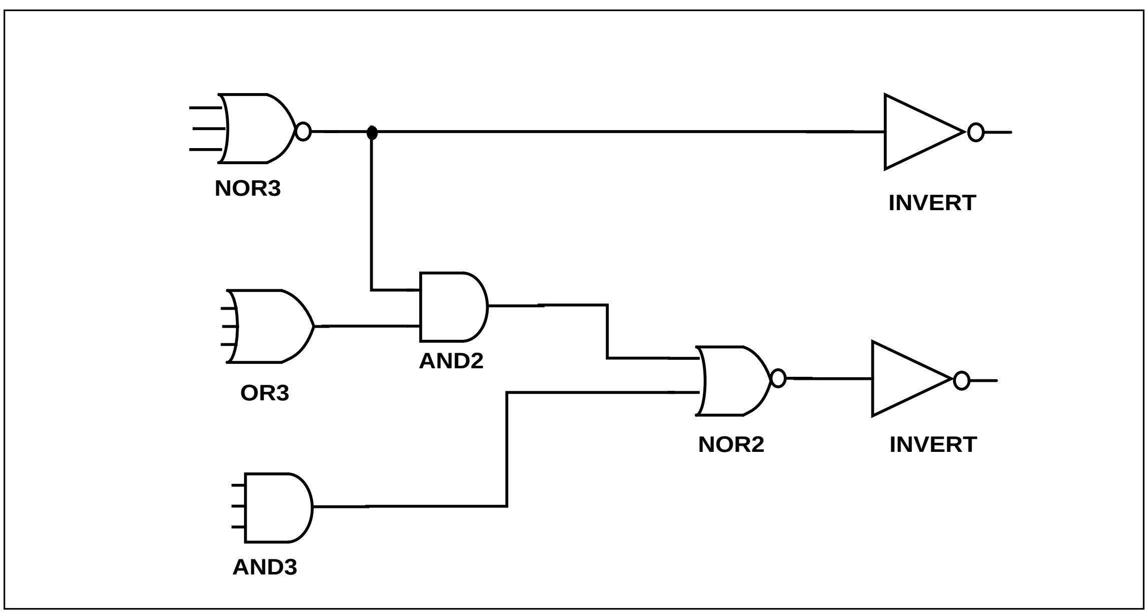

Figure 3. Schematic Representation of Logic Circuits in an ASIC Library

Design Analysis

After entering a design in an HDL, the designer begins the process of analyzing what was entered to determine if it correctly implements the intended function. The traditional method is through **simulation**, which evaluates how a design behaves. Simulation is a mature, well-understood process, and there are many simulators available that accept HDLs written in VDHL, Verilog, or increasingly, both languages. IBM ASICs supports many different simulators available from CAD vendors, such as Verilog-XL™ and Leapfrog™ from Cadence; VSS™ from Synopsys; and MTI™ from Model Technology, Inc.

A more recent addition to the design analysis phase is **power analysis**, with many new CAD tools coming to market in the last year. For a growing number of customers, the power consumption and dissipation of their designs are becoming critical factors. Early feedback on the power requirements of a design allows designers to make timely basic design trade-offs in order to achieve power targets. Because this analysis is at the architectural level and is technology-independent, the estimates may not be extremely accurate and may vary as much as 50% from the actual silicon implementation.

Simulation

Figure 4 represents the traditional simulation process. The VHDL or Verilog, which describes the design function, is read into the simulator tool along with a set of **input vectors** created by the designer. The simulator generates **output vectors** that are captured and evaluated against a set of expected values. If the output values match the expected values, then the simulation passes; if the output values differ, then the simulation is said to fail and the design needs to be corrected. Most simulators generate output in two forms: numerically, as 0's and 1's in a file for comparison purposes, and graphically, as waveforms that depict the transition of signals from 0 to 1 and vice-versa.

Note that the simulation at this level is technology-independent. There is usually little or no technology-specific information delivered by an ASIC vendor to support simulation at this phase. Exceptions include high-level behavioral models for large macros such as RAMs, ROMs, or complex cores.

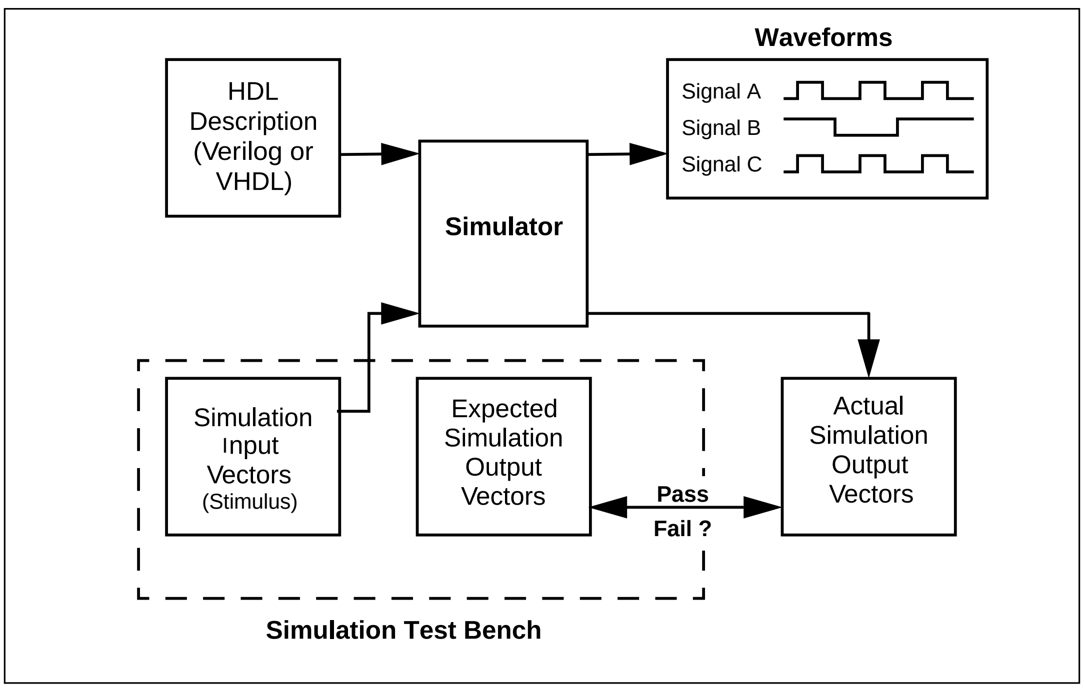

Figure 4. Traditional Simulation Process

Technology Optimization

Technology optimization takes a technology-independent description of a design, and maps it to a library of logic circuits provided by an ASIC vendor, thereby making the design technology-dependent. This phase seeks not just a correct mapping, but the most efficient one in terms of the customer requirements. The optimization process is divided into subprocesses: logic synthesis; test insertion; clock planning and insertion; and floorplanning.

Logic Synthesis

Logic synthesis is the basic step that transforms the HDL representation of a design into technology-specific logic circuits. An ASIC vendor provides the logic circuits in a form called a “synthesis library”. As the synthesis tool breaks down high-level HDL statements into more primitive functions, it searches this library to find a match between the functions required and those provided in the library. When a match is found, the synthesis tool copies the function into the design (instantiates the circuit) and gives it a unique name (cell instance name). This process continues until all statements are broken down and mapped (synthesized) to logic circuits. There are potentially hundreds, or even thousands, of different combinations of logic circuits that can implement the same logical function. The combination chosen by a synthesis tool is determined by the synthesis constraints provided by the designer. These constraints define the design’s performance, power, and area targets. A design driven primarily by performance criteria may use larger, faster circuits than one driven to minimize area or power consumption. Synthesis has matured over the past 5–8 years in the merchant market and is used in virtually all ASIC design starts today.

The inputs to the logic synthesis process are the HDL design description (VHDL or Verilog), the design constraints, and the synthesis library provided by the ASIC vendor. The output of the synthesis process is a list of circuit instances interconnected in a manner that implements the logical function of the design. This list of interconnected circuit instances is called a **netlist** and can be written in several different formats or languages. The dominant netlist languages are VHDL, Verilog, and Electronic Design Interchange Format (EDIF). The interconnected circuits may also be graphically represented as schematics.

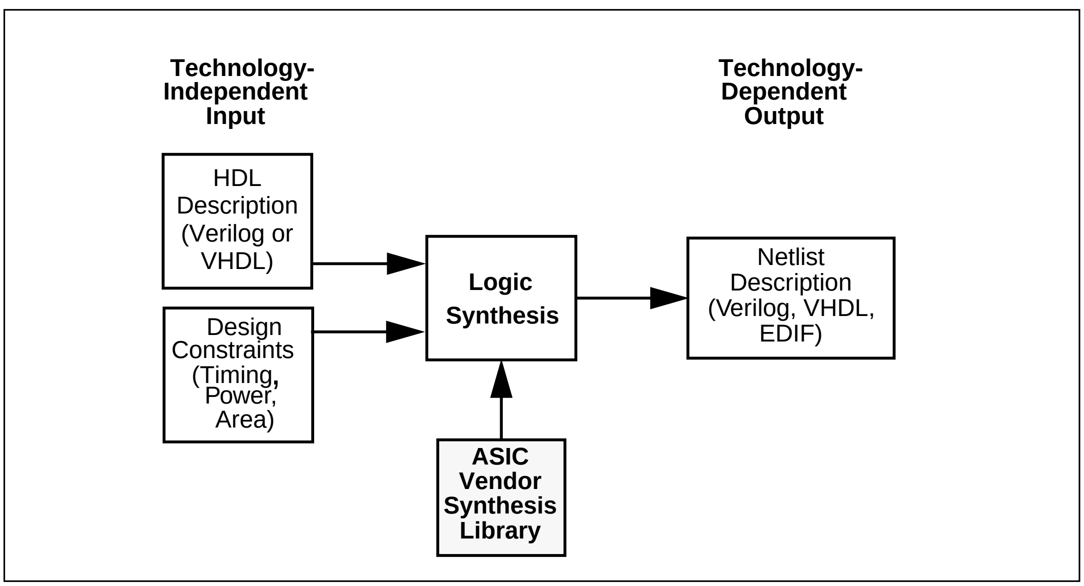

Figure 5. Logic Synthesis Process

The most popular synthesis tool in the external market, accounting for about 85% of the total synthesis seats, is Design Compiler™ by Synopsys. At IBM, IBM's BooleDozer™ is the tool of choice, accounting for approximately 90% of the internal synthesis seats.

Sample Synthesis Workflow

The next four figures depict the synthesis process. Figure 6 provides an overview of the process and indexes the three figures which follow it.

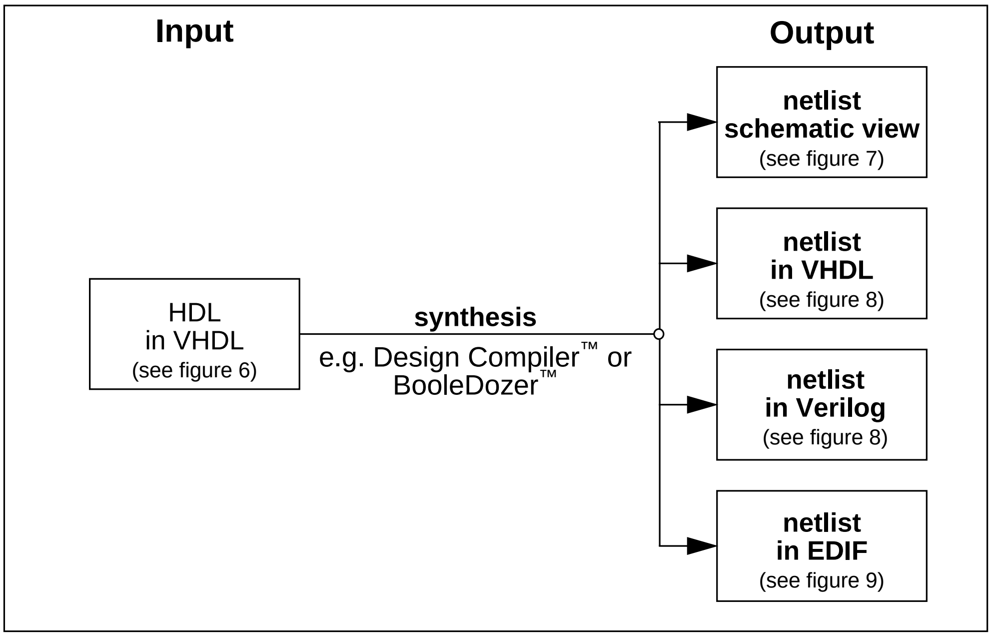

Figure 6. Synthesis Process with Various Possible Outputs

The HDL design description (in VHDL) shown below in Figure 7 is a technology-independent description of a counter function called **refctr**. Take note of the statements in the dotted box that assign the value of a signal **s_load** to a signal **s_ref_ctr_out** based on the status of **CLK**.

```
entity refctr is
    port (COUNT: in  std_ulogic_vector(5 downto 0);
          CLK: in std_ulogic;
          RESET: out std_ulogic);
          ....

architecture refctr_rtl of refctr is
    signal s_ref_ctr_out   : std_ulogic;
    signal s_load          : std_ulogic;
    s_next_ctr_val         : std_ulogic_vector(5 downto 0);
    s_counter_input        : std_ulogic_vector(5 downto 0);
    s_counter_output       : std_ulogic_vector(5 downto 0);
    s_reset                : std_ulogic_vector(5 downto 0);

begin
    s_reset(0) <= RESET;
          ....
process(CLK)
        begin
          if (CLK = '1') and (not CLK'stable) then
             s_counter_output <= s_counter_input and not s_reset;
             s_ref_ctr_out <= s_load;
          end if;
         end process;
          ....
end refctr_rtl;
```

Figure 7. Technology-Independent VHDL Source

ASIC Design Methodology Primer

Schematic View of refctr

Figure 8 depicts a post-synthesis schematic view of a section of **refctr**. Notice that the design was mapped to specific logic circuit functions, such as INVERT_A, NOR3_4 and D_F_LPH0001_4. These names correspond to circuit names found in the IBM *ASIC CMOS 5S Databook*, SA14-2203-03. Each circuit has a unique name, such as U87 for one instance of NOR3_4, and U88 for another instance of NOR3_4. The instance names U87 and U88 were generated by the synthesis tool as it mapped the HDL function into logic circuits such as NOR3_4.

Signals generated by the synthesis tool as it mapped the HDL to logic circuits appear with names such as n275 and n276. Signal names explicitly named in the HDL, such as sload and CLK, are retained. Notice that sload and CLK feed into a circuit that generates the signal s_ref_ctr_out, as described in the technology-independent source on the previous page (Figure 7).

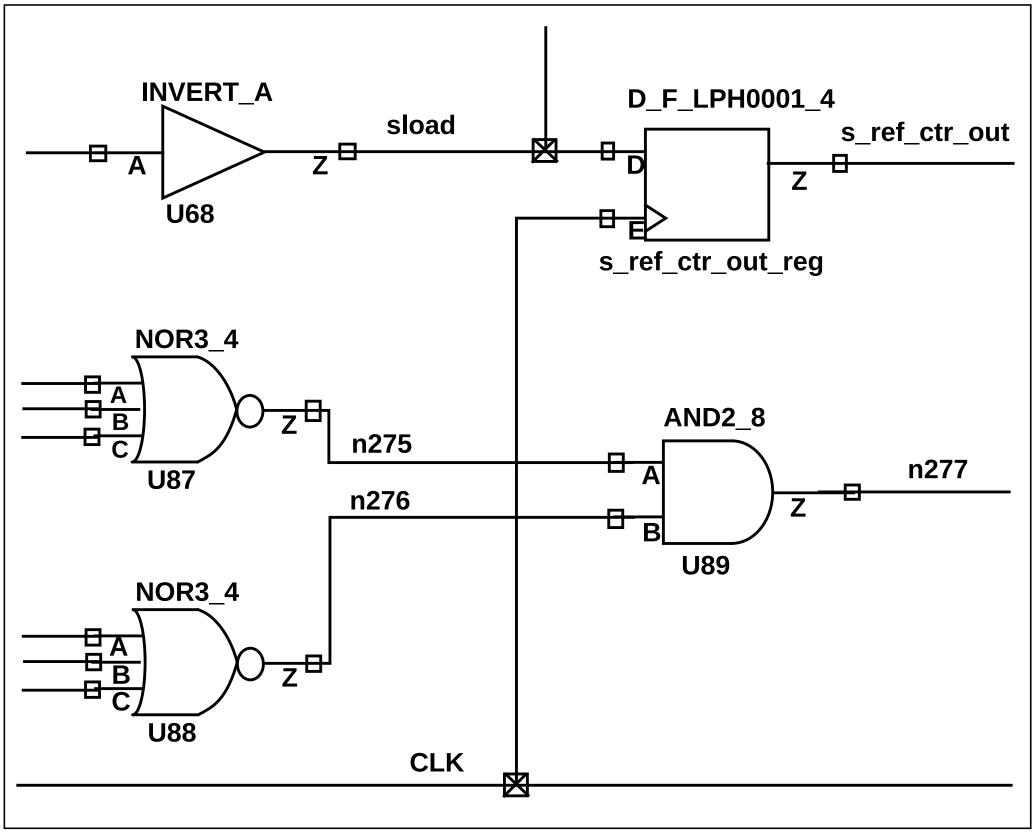

Figure 8. Netlist Schematic View of refctr

Netlist Gate-Level View of refctr (VHDL, Verilog)

Figure 9 contains the post-synthesis netlist of **refctr**, output in both VHDL and Verilog. The circuits described, along with net names and instance names are exactly the same. The difference is in the syntax of the description.

VHDL

```
entity refctr is
     ...
architecture SYN_refctr_rtl of refctr is
     ...
component INVERT_A
 port(Z : out std_logic; A : in std_logic);
end component;
component NOR3_4
 port(Z : out std_logic; A, B, C : in std_logic);
end component;
component AND2_8
 port(Z : out std_logic; A, B : in std_logic);
end component;
component D_F_LPH0001_4
 port(L2 : out std_logic; D, E : in std_logic);
end component;
     ...
begin
 U68 : INVERT_A port map (Z => s_load, A => n265);
 U87 : NOR3_4 port map (Z => n275, A => COUNT(3),
      B => COUNT(4), C => COUNT(0));
 U88 : NOR3_4 port map (Z => n276, A => COUNTb(5),
      B => COUNT(2), C => COUNT(1));
 s_ref_ctr_out_reg : D_F_LPH0001_4 port map (L2 =>
      s_ref_ctr_out, D => s_load, E => CLK);
end SYN_refctr_rtl;
```

Verilog

```
module refctr (COUNT, CLK, RESET, REF);
   ...
 INVERT_A U68 (.Z(s_load), .A(n265) );
 NOR_4 U87 (.Z(n275), .A(COUNT[3]),
   .B(COUNT[4]), .C(COUNT[0]) );
 NOR3_4 U88 (.Z(n276), .A(COUNT[5]),
   .B(COUNT[2]),.C(COUNT[1]) );
 AND2_8 U89 (.Z(n277),.A(n275),
   .B(n276) );
 D_F_LPH0001_4 s_ref_ctr_out_reg(
   .L2(s_ref_ctr_out), .D(s_load), .E(CLK));
   ...
endmodule;
```

Figure 9. Gate-Level Netlist View of refctr - VHDL/Verilog

ASIC Design Methodology Primer

Netlist Gate-Level View of refctr (EDIF)

The EDIF version of the netlist also contains the exact same information as the schematic, VDHL and Verilog versions in terms of the circuits and their connectivity. The difference is, again, syntactical. EDIF is also more verbose than either VHDL or Verilog, and the data volume of an EDIF netlist is a drawback; nonetheless, EDIF is an industry standard and is accepted by almost every electronic design automation (EDA) tool on the market.

EDIF

```
(cell refctr (cellType GENERIC)
   ...
(contents
   (instance U68
    (viewRef Netlist_representation
     (cellRef INVERT_A(libraryRef IBMCMOS5S_SC))
     )
 )
 (instance U87
  (viewRef Netlist_representation
   (cellRef NOR3_4(libraryRef IBMCMOS5S_SC))
   )
 )

 (instance U88
  (viewRef Netlist_representation
   (cellRef NOR3_4(libraryRef IBMCMOS5S_SC))
   )
 )
 (instance U89
  (viewRef Netlist_representation
   (cellRef AND2_8 (libraryRef IBMCMOS5S_SC))
   )
 )
 (instance s_ref_ctr_out_reg
  (viewRef Netlist_representation
   (cellRef D_F_LPH0001_4
    (libraryRef IBMCMOS5S_SC))
   )
 )
```

EDIF (continued)

```
(net s_load
  (joined (portRef A (instanceRef U74))
   (portRef D (instanceRef s_ref_ctr_out_reg))
   (portRef Z (instanceRef U68))
   )
 )
(net CLK
  (joined (portRef CLK)(portRef E (instanceRef
    (s_ref_ctr_out_reg))
    (portRef E (instanceRef s_counter_output_.....
      ...
 (net s_ref_ctr_out
   (joined (portRef D0 (instanceRef U90))
    (portRef L2 (instanceRef s_ref_ctr_out_reg))
   )
 )
 (net 275
   (joined (portRef A (instanceRef U89)) (portRef Z
     (instanceRef U87)))
 )
(net 276
   (joined (portRef B (instanceRef U89)) (portRef Z
     (instanceRef U88)))
 )
(net 277
   (joined (portRef SD instanceRef u90)) (portRef Z
     (instanceRef U89)))
   )
)))))
```

Figure 10. Gate-Level Netlist View of refctr - EDIF

Test Insertion

Test insertion, the step following logic synthesis, consists of inserting structures into the design to enable a complete and efficient manufacturing test. The IBM ASIC methodology requires that the test structures be inserted in a manner that is compliant with IBM's full-scan design-for-test (DFT) methodology. IBM is a recognized industry leader in DFT, and its incorporation into IBM ASIC flow is an important market differentiator. Compliance with the methodology offers customers significant advantages, such as high-quality test coverage (greater than 99% on average) and automatically-generated test patterns.

The design output by the logic synthesis phase is not automatically compliant with IBM's full-scan DFT. The sequential storage elements that the synthesis tool can select automatically from an ASIC vendor's library is limited to a flip-flop element that is not scan-based. The test insertion process replaces the non-scannable flip-flop with a scannable element from the IBM ASIC library, and then generates and connects the appropriate scan and test clocks. Figure 11 below depicts this insertion process.

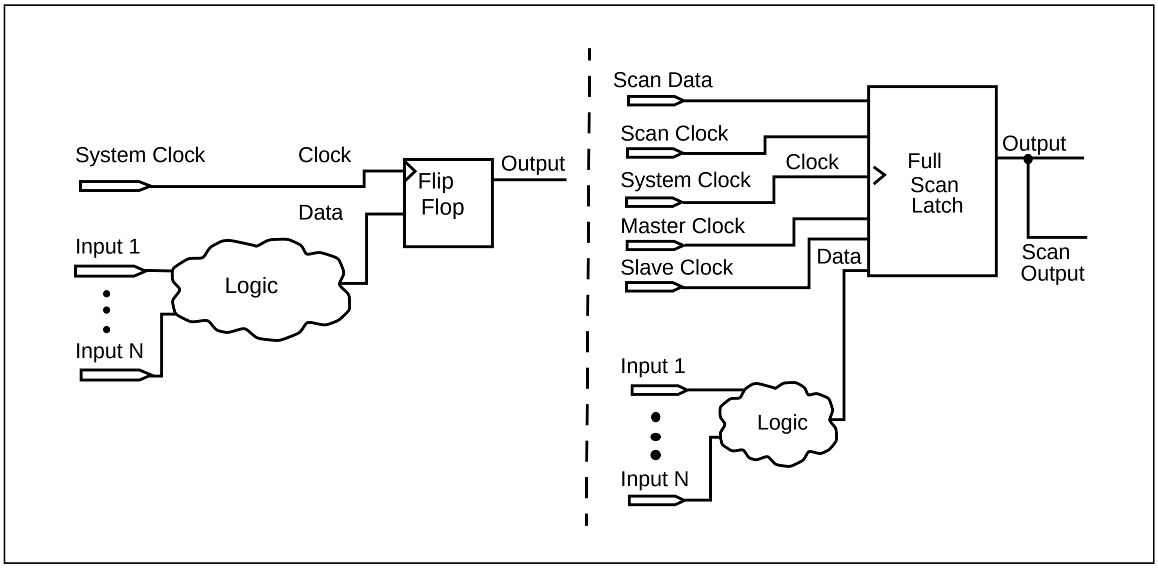

Figure 11. Test Insertion

Clock Planning and Insertion

The last phase of the technology optimization process is the planning and insertion of the **clock network**. Every ASIC design has at least one clock; many of the large and more complex ASIC designs have multiple clocks, in some cases, twenty or more. The manner in which the clock network is propagated throughout the design to the clocked circuits (such as latches, flip-flops and other logic circuits that need to be synchronized with the clock signal), can vary from vendor to vendor, and involves trade-offs amongst various design parameters:

- die area;
- delay through the clock network to the clocked circuits (**latency**);
- the variation in clock arrival time at the various clocked elements (**skew**); and,
- the power generated by the clock network as it switches.

The clocking methodology must comply with the DFT requirements in order to maintain the testability of the design.

IBM uses a **clock tree** or **repowering tree** method to propagate a clock signal to the hundreds, thousands, or tens of thousands of logic circuits that receive that clock signal. Before clock tree insertion, a design is said to have **idealized clocks**, meaning that all logic circuits receiving a given clock signal are driven in parallel from a single clock driver circuit. However, there are significant barriers to actually implementing a single circuit directly driving thousands of other circuits; these barriers include: routability; required circuit drive strength; management of clock latency and skew; and others. IBM's ClockPro tool allows a customer to input information on the characteristics and constraints for each clock network on the die. ClockPro automatically generates multiple valid clock trees, or levels of repowering circuitry, for each clock network, and generates for each such clock tree, the corresponding information on its cell area, latency, and fanout. This information allows the customer to select the optimum repowering network for each clock. The information from ClockPro can then be automatically added to the customer's design by IBM's BooleDozer-Lite tool.

Another important task accomplished during clock insertion is the introduction of **clock splitter** circuits into the design. The clock splitter, placed at the last stage of the repowering tree before the latches, generates the true and complement (master and slave) clock phases required for area-efficiency and high performance. The splitter also includes clock control logic required for DFT compliance, and can drive the optimum number of latches as chosen by the customer via the Clockpro™ tool. Figure 12 represents insertion process, including the clock splitter circuitry.

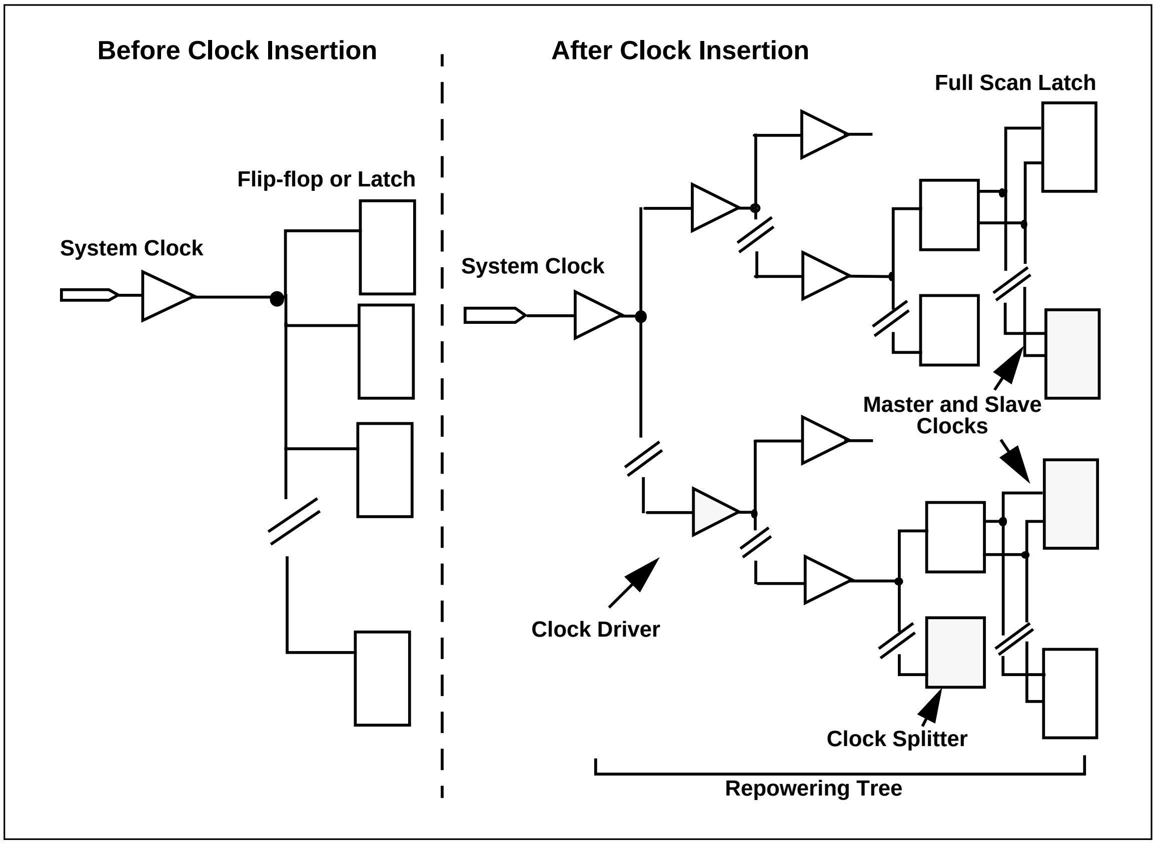

Figure 12. Clock Planning and Insertion

Floorplanning

Floorplanning is the process of placing groups of circuits on a die, and analyzing the effect of that placement in terms of design performance and routability. The need for floorplanning arose as circuits became smaller and the length of the wires that interconnect those circuits began to dominate design performance trade-offs. This is often referred to as one of the "deep-submicron" (>0.5 micron) design paradigms where interconnect delay dominates the delay through the individual circuits or gates. Integrating floorplanning into the prelayout portion of the methodology allows the designer to consider the physical implementation of the design during the logic design process. Trade-offs on design partitioning, I/O assignment, and macro location assignments can be made early on, avoiding costly design iterations between layout and synthesis.

By physically placing groups of logic on a die, more accurate estimates can be made of the wire lengths within the logic groups (shorter, faster nets) and the wires that interconnect the groups (longer, slower nets). More accurate estimation of wire lengths that interconnects the logic on chip translates into more accurate wire delay predictions, which greatly affects the overall design timing. The wire length estimates from floorplanning can be passed back to the synthesis tool and used to further optimize the selection of logic gates chosen to implement a function. The floorplan grouping information can also be passed directly to the ASIC vendor's detailed place and route tools. This can improve the turn-around-time through the design center for the layout of the die. Floorplanning also helps to monitor the actual size of a design which eliminates the discovery during the layout phase that a design has outgrown its target die size.

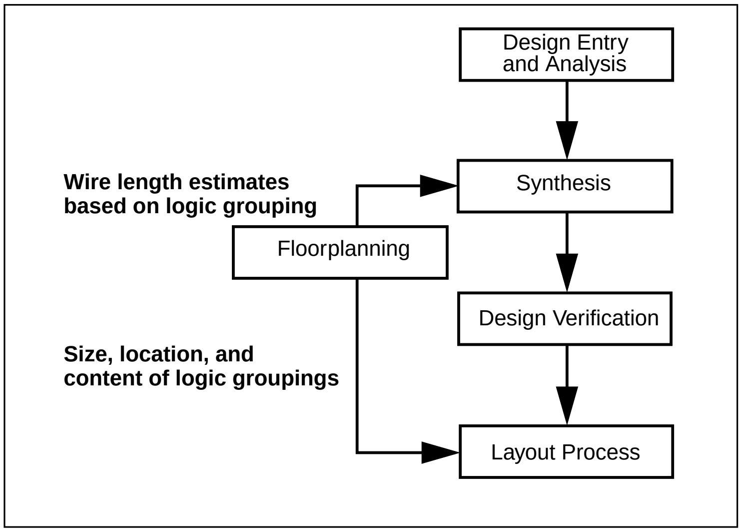

Figure 13. Prelayout Floorplanning

Design Verification

The design verification performed at this point in the design process ensures, through automated checking, that the design (1) is functionally correct, and (2) meets physical constraints in terms of performance, testability, power, and technology-specific electrical checks.

Functional Verification

Designs, as we have seen, are functionally verified before synthesis using simulation. Now, after synthesis, the design is **resimulated** to ensure that its function has not been corrupted by the synthesis process. As synthesis tools have matured, the likelihood of introducing functional errors during synthesis has been drastically reduced. Nonetheless, it is still advisable to verify the technology-mapped version of the design.

The traditional verification method is to resimulate the gate-level version of the design. The process is straight-forward. The gate-level version of a design should produce the exact same functional results as the pre-synthesis version of the design, given the same set of stimulus (input vectors). Unfortunately, as designs exceed 100,000 gates, the elapsed time required to rerun simulation vectors becomes prohibitive. Designs of up to one million gates can take weeks or more of simulation time to complete functional verification. Because of the inefficiency of this method for large designs, **formal verification**, also referred to as **Boolean equivalency checking**, is recommended as an alternative.

ASIC Design Methodology Primer

Figure 14 illustrates the traditional verification process of repeating simulation after synthesis. The simulation inputs include a gate-level, technology-dependent version of the design, and the ASIC vendor simulation library. The simulation library contains a model for each circuit in the library that describes the circuit function (invert, AND, etc.).

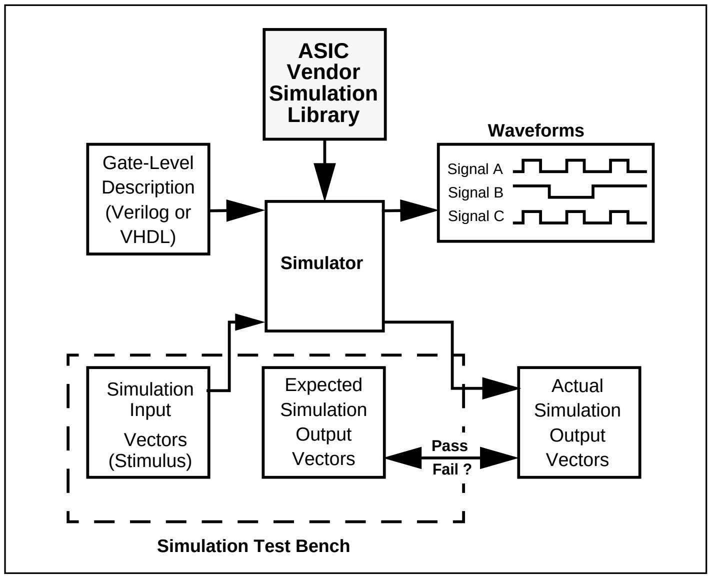

Figure 14. Gate-Level Simulation

Formal Verification

Formal verification achieves the same purpose as gate-level simulation, which is to guarantee that the function of the design was not altered or corrupted by the synthesis process. The method, however, is very different. A formal verification tool breaks a design down into a set of Boolean or logical expressions. This process is repeated on a second version of a design, and the logical expressions are compared for equivalence. The comparison of the two designs is exhaustive, and not driven by evaluating different design states created by input vectors. There are no input or output vectors required.

This method of design verification, while relatively new to the merchant market, has been used successfully within IBM for many years. Verification of a 500,000 gate design through formal verification can occur in approximately three hours, as compared to the hundreds of hours required by simulation. Robust tools are becoming available in the merchant market; IBM ASICs supports Chrysalis Symbolic Design, Inc.'s Design VERIFYer.® Comparisons can be done against both technology-dependent and technology-independent versions of a design.

In addition, formal verification is also extremely useful for comparing two technology-dependent versions of a design for equivalence. A recommended use is to compare the post-test insertion version of the design against the netlist from logic synthesis, or the post-layout version of a design against the prelayout version. A technology file from the ASIC vendor is required to help the formal verification tool understand the function of technology-dependent features such as master/slave clocks used for DFT support. Figure 15 shows some of the ways in which formal verification can be applied.

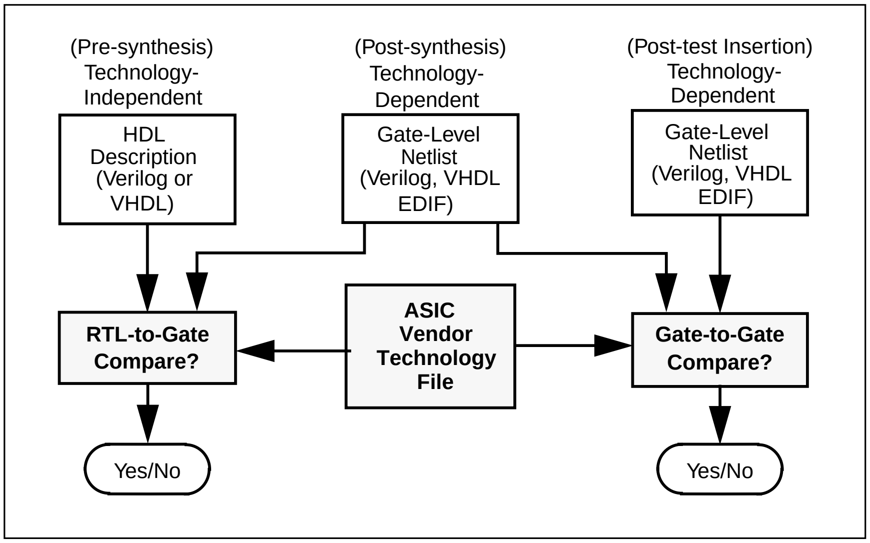

Figure 15. Formal Verification

Testability Verification

The purpose of this step to ensure that the design, as implemented by a specific set of circuits, can be tested on the manufacturing floor. The traditional method for testability verification is also based on gate-level simulation. A subset of the functional test patterns (input vectors) is applied to the actual silicon on the manufacturing testers. Those parts that yield the expected values for the vectors applied are said to pass and are shipped to the customer as good die; those that do not pass, are said to fail. Then the ASIC vendor and the customer determine where the problem is on the die, and how to correct it.

This test verification method suffers from the same problems as functional verification based on gate-level simulation; namely, the time required to develop and run an exhaustive functional pattern set is prohibitive for large designs. The quality of the test coverage also becomes a problem with functional pattern testing because defects in areas of the die not tested by the pattern set can go undetected. As stated before, the IBM ASIC design methodology is based on a full-scan design-for-test (DFT) methodology. IBM provides the customer with the TestBench™ suite of test tools, which includes the Test Structure Verification (TSV) program.

TSV analyzes the gate-level version of a design for compliance with test requirements. Designs that are compliant require no further action on the part of the customer to support manufacturing test. Test patterns are generated by IBM as part of the normal ASIC processing. A design must comply with all TSV requirements to pass IBM's sign-off requirements. TSV noncompliances are noted as errors or warnings. Errors can affect the ability to generate the manufacturing pattern and must be fixed by the customer. Warnings will not affect test pattern generation, but may affect overall test coverage, and therefore are also communicated to the customer. The resulting test coverage achieved on a DFT-compliant design can be very high (99.5%); most designs achieve 99% coverage or greater.

IBM's world-leading test methodology is one of the primary differentiators between IBM and other ASIC suppliers. TSV analyzes the testability of the entire design, and solves the problem of failure to detect manufacturing defects because of the lack of appropriate patterns. This method, which facilitates the automatic generation of test patterns, overcomes the difficulty of creating high-quality, high-coverage tests for large designs; TSV has been proven many times on designs with more than 500,000 gates. By relieving the customer of the time-consuming task of generating manufacturing test vectors, many customers find significant savings in time-to-market. IBM's TSV program provides an integrated test verification methodology that can handle the self-test circuitry provided on all IBM ASIC RAMs and ROMs, the isolation required for embedded core functions, and boundary scan requirements.

Timing Verification

The purpose of timing verification is to determine if a design, once mapped to a specific technology library of circuits, meets the specified performance target. Once again, the traditional method is based on gate-level simulation, and once again run-time and design coverage issues make this method impractical for large designs. IBM ASICs advocates the use of static timing analysis for timing verification and requires the use of its EinsTimer™ static timing analyzer for timing sign-off. IBM is the first ASIC vendor to support static timing analysis for sign-off, and delivers the EinsTimer static timing analysis tool as part of the ASIC design kit. This tool has been proven to handle extremely large (over 1 million gate) designs with complex timing characteristics such as multiple clock domains and clock gating.

Static timing analysis allows all paths on a die (under best- and worst-case conditions) to be examined in a single timing run. This method, while relatively new to the merchant market, has been practiced successfully within IBM for over 15 years. The move away from timing simulation toward static timing analysis is the industry trend, with news of other ASIC vendors supporting static timing becoming more common. Static timing on a large design (example 860,000 gates) can be achieved in two to three hours as compared to the many days or weeks required to get equivalent coverage (if possible) using delay simulation.

To yield meaningful information about a design using static timing, careful attention must be given to the development of the **timing assertions** (those files that define expected arrival times, clock phase relationships, false paths, etc.) that drive the analysis tool. While development of a meaningful assertion set requires an up-front designer investment, once completed, these assertions are used to drive the layout process and allow for timing correction without customer intervention. This translates into another significant time-to-market advantage, and is an important differentiator for IBM in the ASIC marketplace.

Power Estimation

Verifying the amount of power a design consumes and dissipates is the purpose of power estimation. The traditional method has been primarily pen and paper calculations using technology information provided by the ASIC vendor and switching information supplied by the customer. This method is inadequate in terms of scope and accuracy for today's power-conscious designs. As a result, many new power estimation tools are under development and are just beginning to enter the merchant market.

Prelayout Technology Checks

A final set of technology- and library-specific design verification checks is usually provided by the ASIC vendor. These checks verify a variety of ASIC vendor requirements; examples include, verifying all input pins on each circuit in the design are used (connected to another circuit); verifying all circuits that communicate to tester equipment are located in the required I/O slots, etc. To perform these checks, IBM provides the CMOSChks tool as part of the ASIC design kit. Passing these checks without error is one of the prelayout sign-off requirements. Customers can find and resolve errors in the design before the design enters the layout process. The earlier these types of errors are discovered, the faster they can be corrected without adverse impact on the turn-around-time for layout.

Layout

The last major section on ASIC design methodology examines the physical implementation of the design in silicon. This process is traditionally performed by the ASIC vendor at **ASIC design centers**. These design centers may be located at the actual silicon foundry site or at satellite locations.

The layout process entails the physical placement of the logic circuits on a die (placing), and the interconnection of those circuits using wire (routing). Most ASIC vendors support two to three different layers of wiring on a die; some are beginning to claim the capability of up to five in future technologies. IBM ASICs has successfully executed five levels of wiring in production since 1993. IBM's leading-edge interconnect technology is a significant differentiator in the ASIC market place. More levels of wiring translate into the ability to interconnect more circuits on a given die, allowing for denser, more integrated designs.

Traditionally, designs were placed and routed by the ASIC vendor, and then retimed to see if the original performance target was achieved. If the performance target was missed, the customer had to change the logic. This often meant resynthesizing blocks of logic and then repeating the design verification and place and route steps. Multiple iterations through this process to achieve timing closure could add weeks or even months on top of the original schedule. With floorplanning, earlier analysis can be done by the customer, and the design reoptimized before entering the layout process.

Floorplanning can be considered part of the layout process, part of the technology optimization process, or both, and can be performed by either the customer or the ASIC vendor. Floorplanning straddles the traditional **front-end** (that is, the logic design process) and the **back-end** (the physical design process), and helps to yield optimum results from both. Because early, pre-layout, floorplanning is an essential ingredient to successfully designing deep-submicron ASICs, floorplanning has already been discussed in the Technology Optimization section on page 7.

The layout process proper involves two additional steps:

- **Place and Route**, the process of determining the placement and interconnect of each circuit on a die
- **Back-annotation,** the process of extracting timing information from one design step for analysis in an earlier step, such as using post-layout delay information in simulation

Place and Route

While floorplanning deals primarily with the placement and interconnect analysis of groups or clusters of logic, place and route deals with the placement and interconnect of each circuit on a die. On today's large die, containing over 1 million logic gates, this is comparable to solving a jigsaw puzzle with hundreds of thousands of pieces. From the customer's perspective, the end result must not only fit in the area allocated, but must also meet the performance targets, and all this must be achieved on schedule. In addition, the ASIC vendor has more criteria for success in terms of technology constraints (for example, no electromigration) and testability (that is, test-related circuits are in the correct locations).

In addition to support for floorplanning, the IBM place and route methodology has the significant advantage of being **timing-driven**; that is, the placement algorithms in the IBM tools take the performance constraints of the design into account as the circuits are placed. Most layout tools place circuits in the most efficient manner from an area standpoint. Adjustments to that placement to improve timing are largely a manual process, and may require the customer to change the actual logic. The IBM layout tools are driven by the same timing assertion files developed by the customer for static timing sign-off with the EinsTimer static timing analysis tool. The placement tools work to create a layout that has the most efficient area utilization *and* meets the timing assertions. If after placement, nets remain that do not meet the specified timing, a series of automated placement optimization routines are run, varying the drive strength of logic circuits and relocating clock driver cells, until timing closure is achieved.

Because the assertions completely describe the performance targets of the design, the optimization can be performed without intervention from the customer and without requiring resynthesis of the design.

ASIC Design Methodology Primer

This is a significant methodology differentiator for IBM ASICs and translates into real time-to-market savings for the customer. This method can handle a range of design sizes from 50,000 to over 2 million gates, and it accommodates a variety of placement approaches:

- a **flat approach**, placing all circuits on the die simultaneously
- a **partitioned approach**, placing circuits using the grouping and preplacement information from floorplanning
- a **hierarchical approach**, individually placing routing sections of the die and interconnecting these sections with global wires

The ability to use up to five levels of metal allows the place and route system to achieve the most area-efficient (dense) placements, utilizing circuit area on the die that other layout systems cannot.

Figure 16 below depicts the flow during IBM place and route, and highlights the fact that timing-driven design allows timing closure to occur at the ASIC vendor rather than having to return this task to customer for solution.

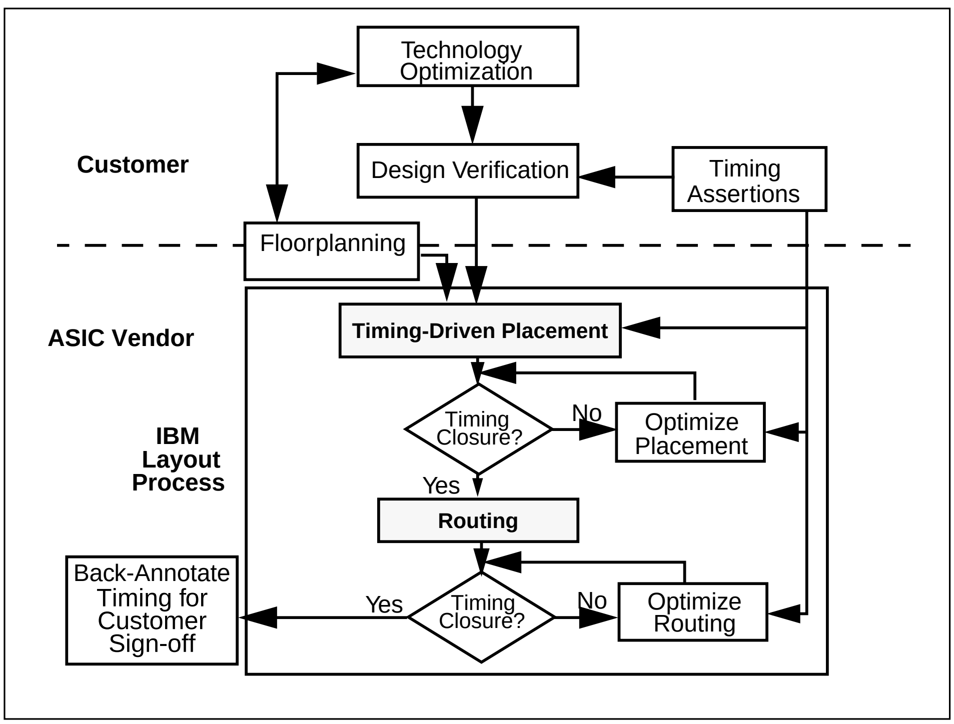

Figure 16. Place and Route Flow Diagram

Timing Back-Annotation

Timing back-annotation is the process of extracting timing information from one design step for analysis in an earlier step. For example, the classic use of back-annotation is the use of post-layout delay information in simulation. Because once a design is placed and routed, the actual distance between logic circuits on the die is known, the corresponding wire delay can be calculated with great accuracy. This delay information is then extracted from the layout and is written in a form that the simulator can understand. The industry standard for this type of delay information is the Standard Delay File (SDF).

The SDF can be read into a simulator for post-layout gate-level timing verification. This process is orders of magnitude slower that gate-level simulation without timing, and impractical for today's large designs. Therefore, IBM takes an alternative approach utilizing the EinsTimer static timing analyzer for both pre-layout and post-layout timing sign-off.

In this second approach, the post-layout wire delays (also known as parasitics) are extracted from the placed and routed design and then written out as RC and CAP files (resistance and capacitance values associated with the individual wire segments). The RC and CAP information is then read into EinsTimer for the postlayout timing analysis. IBM supports the creation and use of the SDF format for customers who chose to do this type of analysis, but does not support delay simulation for timing sign-off.

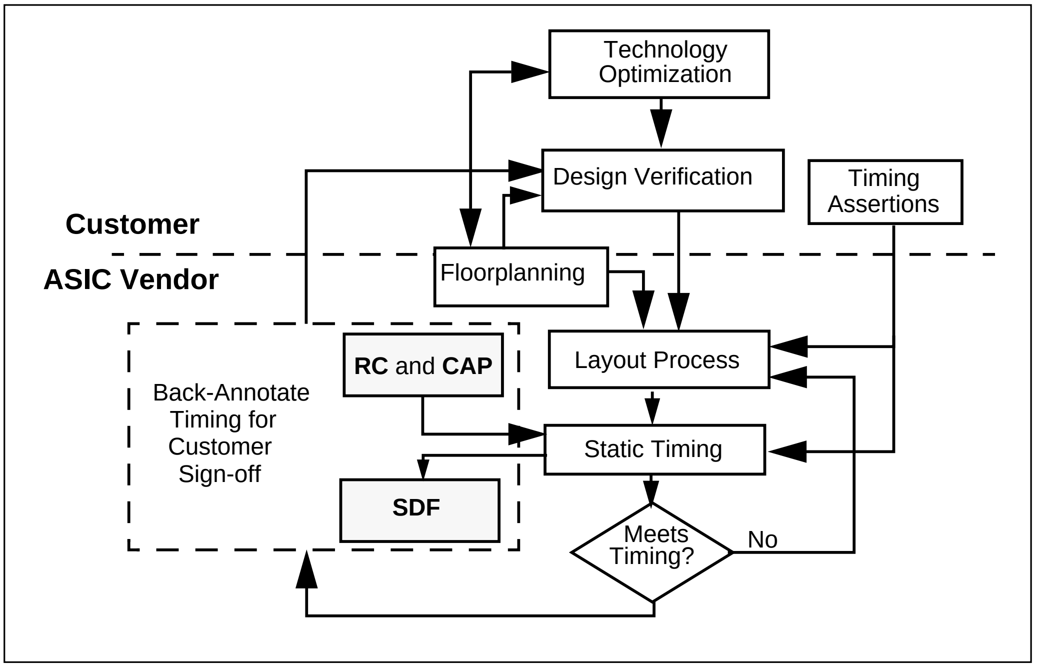

Figure 17. Timing Back-Annotation Process

ASIC Design Methodology Primer

Summary of Steps: The ASIC Methodology Flow

The flow chart in Figure 18 is a graphical summary of the IBM ASIC Design Methodology. The highlighted boxes represent the sign-off points that the design must go through as it progresses to silicon. The first sign-off point is the **Initial Design Review**. At this checkpoint IBM and the customer meet to agree on roles and responsibilities (for example, who will do floorplanning), as well as to document all the technical characteristics about a design. This review makes possible the development of accurate schedules, die sizes, and resource demands.

When logic and design verification is completed, the designer and the ASIC Design Center take the design through the **Release-to-Layout** (RTL) sign-off procedure. Final netlist and assertion files are identified, time- and date-stamped, and rerun through the key sign-off tools (TSV tool for testability; EinsTimer for timing; and CMOSChks for wiring).

To exit the RTL sign-off, the design is required to time without any negative slacks (that is, all nets must meet timing assertions), as well as without any transition or capacitance violations. This is essential to enabling the timing-driven-layout and placement-based optimization routines to achieve timing closure with minimum intervention by the logic designer. The design must also be free of TSV errors, which prevent automatic generation of test patterns. Designs with warnings are allowed to proceed to layout if the customer agrees to accept the lower test coverage that results from the flag-triggering condition.

A final suite of checking programs is run after layout to ensure all technology-specific requirements are met by the design layout, and that final timing is achieved. At this point, test patterns are automatically generated and shipped to manufacturing with the final design.

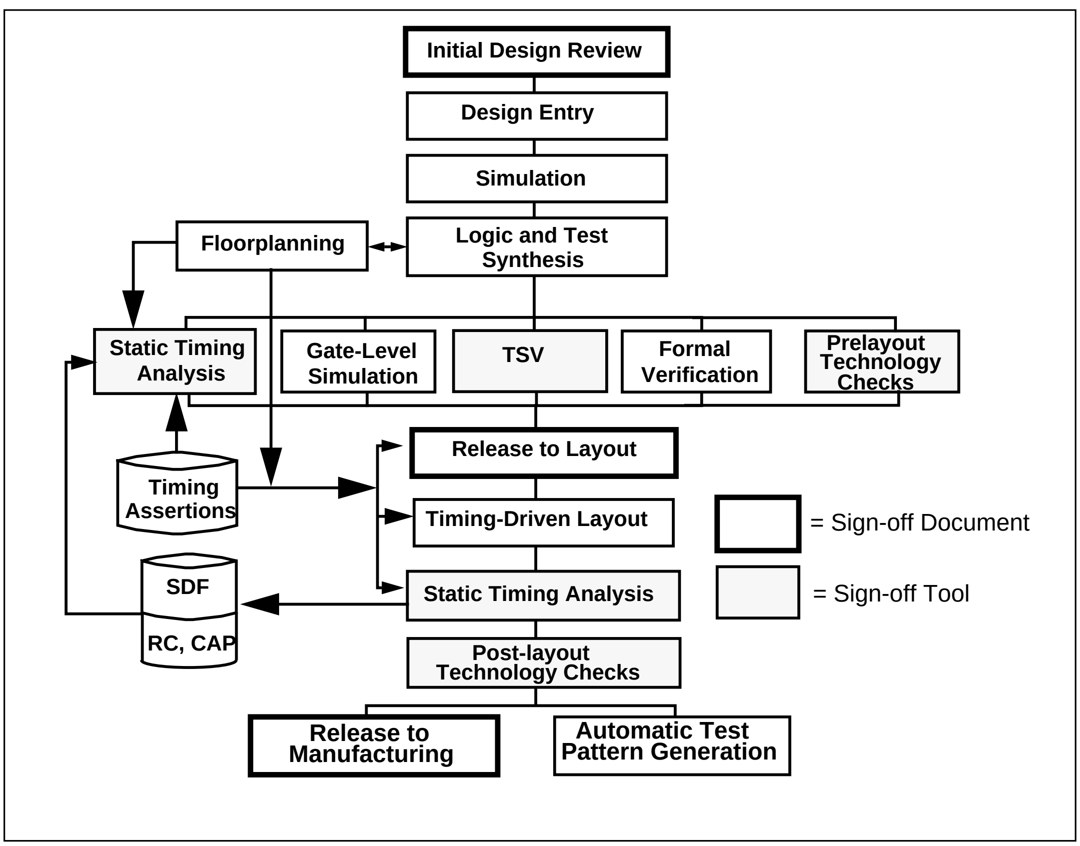

Figure 18. ASIC Methodology Flow

ASIC Design Methodology Primer

Design Challenges and Strategies

The last two years have seen a steady rise in the number of ASICs in the 400,000 gate range and above. In 1995, the one-million-gate ASIC became a reality. With the complexity inherent in such large designs came the problem of prohibitively long development and verification times. Older methods of verifying ASIC timing and testability through gate-level simulation are not practicable given today's time-to-market requirements. Increasingly the layout, or back-end, portion of the methodology has been driven not by gate delay, but by wire delay. To overcome this difficulty front-end timing requirements must be tightly coupled with the layout implementation; this helps to minimize costly iterations through the design and layout process. In order to implement such design strategies, alternative verification strategies are needed. IBM is leading the industry in defining and successfully executing an alternative verification and sign-off strategy based on test structure verification and static timing analysis. IBM's rigorous design verification and checking in the Release-to-Layout and Release-to-Manufacturing procedures along with timing-driven placement and placement-based optimization tools help solve these design challenges.

Summary

IBM's ASIC methodology eliminates the need for lengthy gate-level timing simulation by utilizing static timing analysis. By using design-for-test methodologies and utilizing IBM's test structure verification tools, designers can avoid discovering test coverage problems late in the design cycle. Automatic test pattern generation saves valuable time by eliminating the need to generate the functional patterns manually. Taken together, IBM's ASIC methodology results in shorter design cycles that get the customer's silicon to market faster.