CHAPTER 6

SEMI-CUSTOM ASIC DESIGN

6. Semi-Custom ASIC Design

Complex System-on-Chip (SoC) designs are fast becoming commonplace in today's applications. Hardware designers must overcome many complex and challenging issues regarding cost, time-to-market (TTM), performance, power, capacity, quality and IP integration. Due to the complexity and performance inherent in most SoC applications, designers, historically, were limited to cell-based Application Specific Integrated Circuit (ASIC) technology.

Cell-based ASIC technology offers the performance, power and capacity needed for modern SoC applications. However, the technical advancements in shrinking geometries have driven reticle cost through the roof. This exponential increase in reticle cost translates to excessive Non-Recurring Engineering (NRE) cost, making cell-based ASIC platforms too expensive for all but the highest volume applications (quarter million plus units per year).

Field Programmable Gate Array (FPGA) technologies, with greater capacity, performance and embedded IP, offer SoC designers another viable hardware platform. The programmability of the FPGA allows fast time-to-market, which is extremely critical to the success of a new product. However, the staggering per-unit cost of high-end FPGAs is prohibitive for all but the lowest volume applications (few thousand units per year).

The two extremes of cell-based NRE and FPGA per-unit cost have left a significant void in the market. This void has lead to a market demand for a cost-effect solution suitable for applications ranging from 5k to 1M units per year. In the past year that market demand has led to the emergence of a new, innovative, product called the Structured ASIC. Structured ASIC technology overcomes the two extremes by offering SoC designers a solution with the capacity and performance required for modern SoC applications but without the exorbitant NRE of the cell-based ASIC and exorbitant per-unit cost of FGPA technology.

Structured ASICs are a new breed of custom device that approach the performance of today's Standard Cell ASIC while dramatically simplifying the design complexity. Structured ASICs offer designers a set of devices with specific, customizable metal layers along with predefined metal layers, which can contain the underlying pattern of logic cells, memory, and I/O. By virtue of the predefined structures, the ASIC vendor can design any time-consuming task — such as test, signal integrity and IR drop — into the architecture.

Full custom ASIC chip is the most costly, and like standard cell ASICs, use a custom-designed mask for every layer in the chip. Unlike standard cells, designers of a full custom device have total control over the size of every transistor forming every logic gate, so they can "fine tune" each gate for optimum performance. Thus, a full custom ASIC performs electronic operations as fast as it is possible to do so, providing that the circuit design is efficiently architected.

Today, full custom ASICs represent a small percentage of the ASIC market because gate arrays, structured ASICs and standard cells turn circuit designs into working chips much faster and at much less cost. Such chips have greatly improved in speed over the years and provide the necessary performance for many applications. The speed advantage of a full custom ASIC is not as relevant as it was in the past. It is used primarily for devices such as microprocessors that must run as fast as possible and will be produced in huge quantities.

Figure 6-1 shows the design flow of a typical semi-custom design approach.

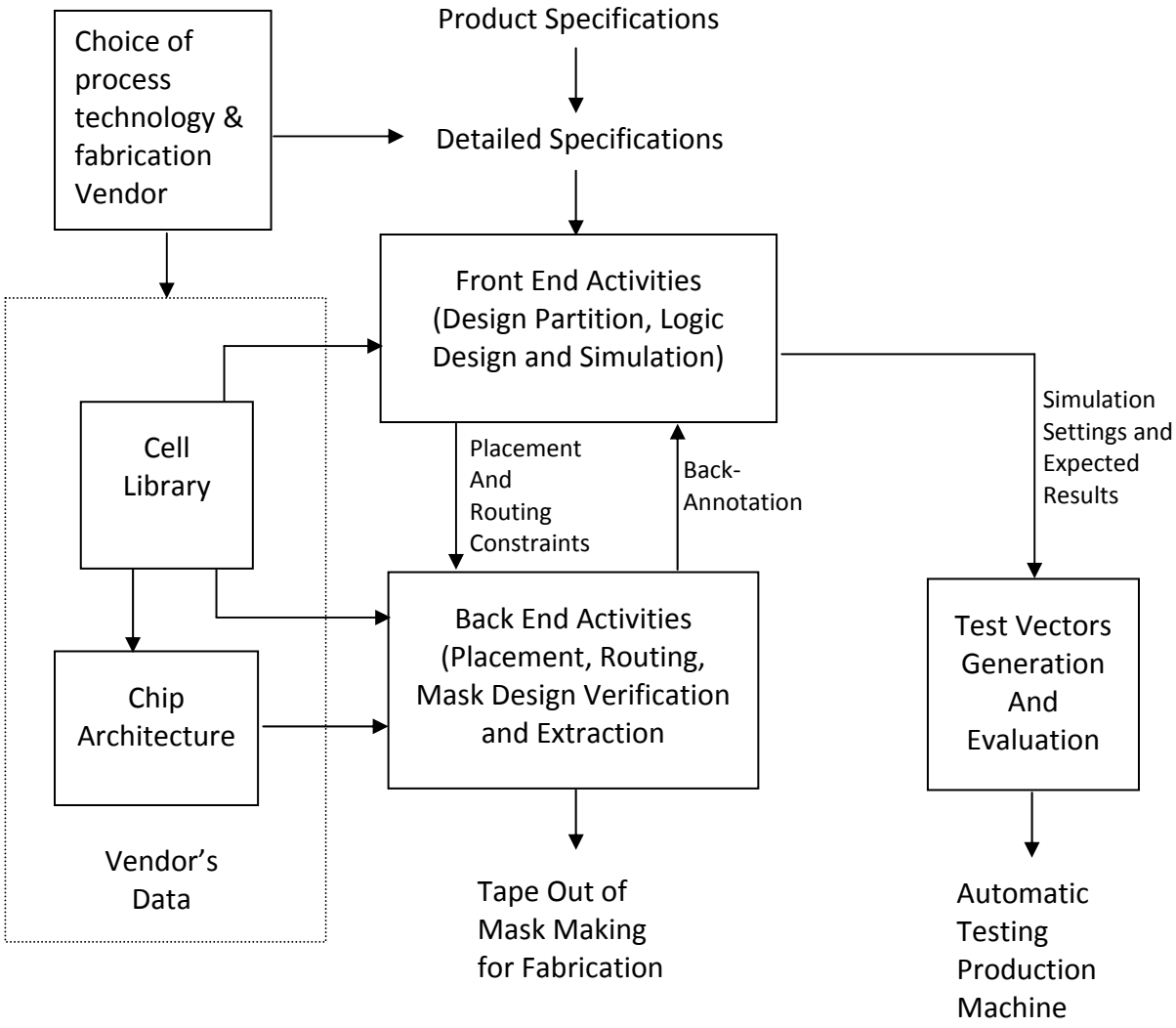


Figure 6-1 Design Flow for Semi-Custom Design Approach

6.1 Standard Cell Techniques

This technique relies on the existence of previously designed and fully characterized standard circuits and layout data held in a CAD database library. The cell libraries may consist of standard logic gates, I/O pads, macros, etc of a particular process technology. The cells are typically optimised full custom layouts, which minimize delays and area.

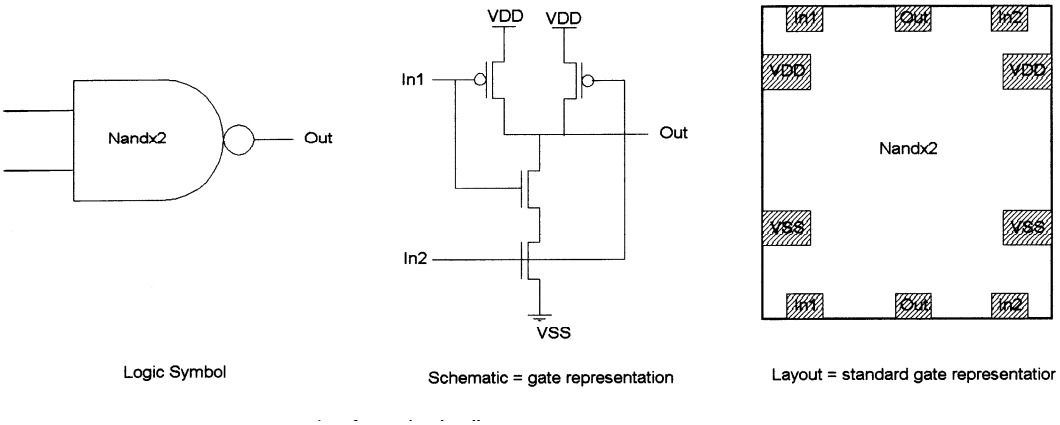


Figure 6-2 Database of a NAND2 standard cell

In the layout, only the access pins and power pins are visible for interconnects. The entire mask layout for the devices within gate is also available in the library but cannot be modified.

There are various sizes of the same device in the library. If a larger driver is required for the NAND2 gate, then another similar NAND2 gate with larger transistor size is selected from the library to meet the requirements.

The entire standard cell layouts have the same height. This enables the cells to be placed alongside each other in uniform rows across the chip. The width of the cells is variable to cater for different driver sizes and different types of gates.

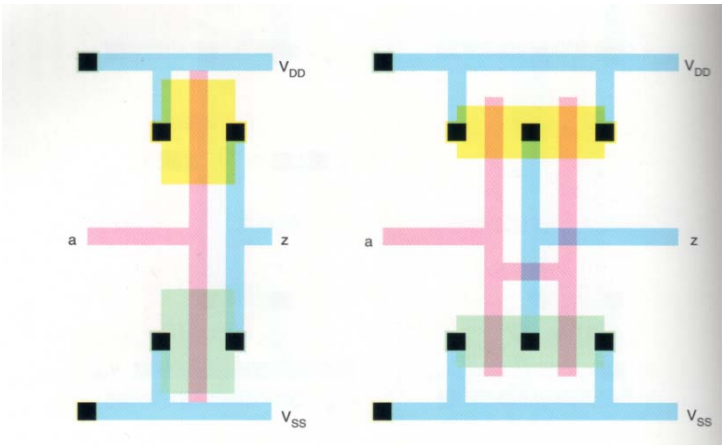


Figure 6-3 Sample layouts of same cell height

For standard cell technique, a full set of mask is required for fabrication purposes since the chip layout, number of I/Os and chip size is dependent on the specific design, which is fully controlled by the IC designer.

6.1.1 Cell Based Design Flow

Figure 6-4 shows an ASIC design flow, typically divided into two major phases namely, the front-end design and back-end design phase.

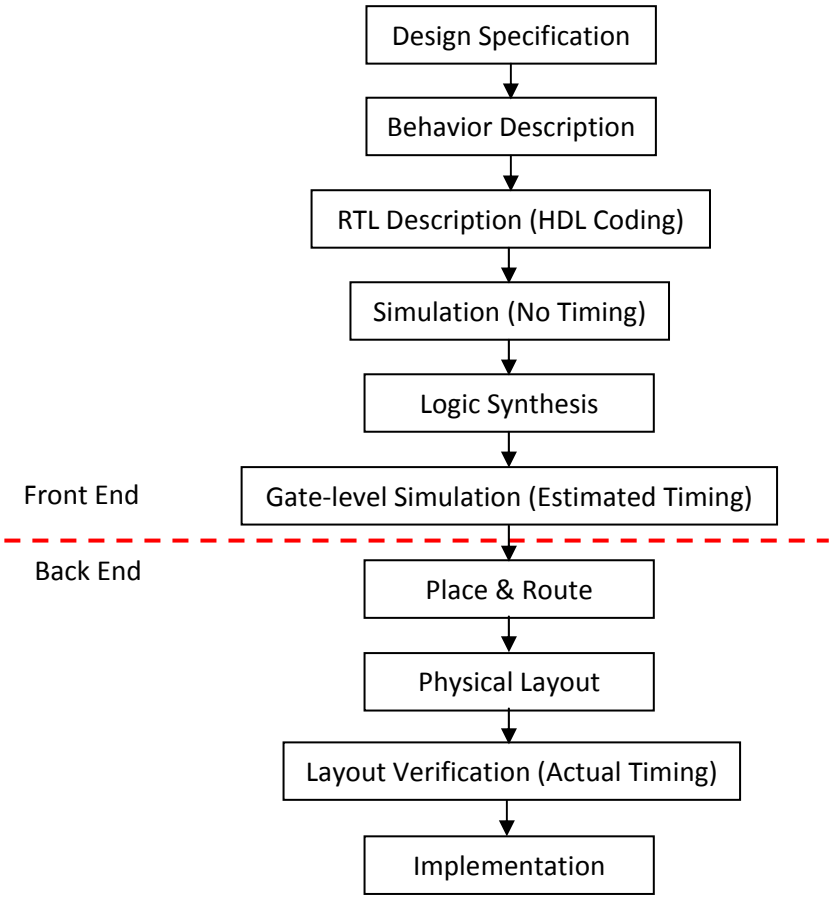


Figure 6-4 ASIC Design Flow

6.1.1.1 Frontend Design

Frontend design is the first phase where the logic for the chip is built using the Hardware Description Language (HDL). This logic realization is done using various steps in which the design is coded, verified and mapped to the actual gates using a process called synthesis.

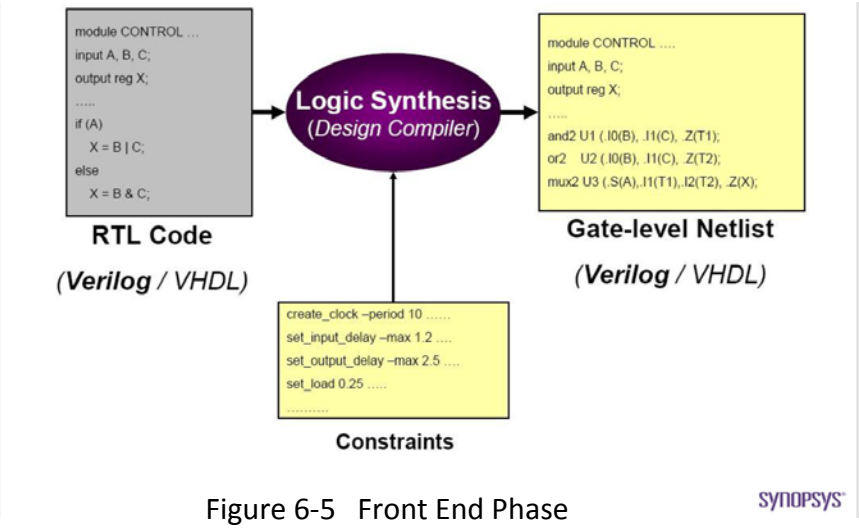


Figure 6-5 Front End Phase

(i) Coding

The design is expressed in the form of behavioural description codes using hardware description languages (HDL) such as Verilog and VHDL.

Figure 6-6 shows a simple example in Verilog codes. Refer to Page 23 for more examples.

```
// This line is a comment
module sample (clk, a, b, c, c_reg);
input clk, a, b;
output c;
output reg c_reg;

assign c = a || b; // Logical OR of the signals a and b

// output of the previous step is sent through a flop that
// is clocked by “clk”

always @(posedge clk)
      c_reg<= c;

endmodule
```

Figure 6-6 A sample block in Verilog HDL

The above code describes a design that has :
- 3 inputs “clk”, “a” and “b” and
- 2 outputs “c” and “c_reg”.

- output “c” is the logical OR of the “a” and “b”.
- output “c_reg” is a registered version of “c”. i.e. output “c” is given to the input of a flop that is clocked by “clk” and the “Q” output of the flop is connected to “c_reg”

(ii) Simulation

Once the coding is completed it has to be checked if the design is doing its expected function. This is called the functional verification.

Test bench using Verilog or VHDL can be developed to apply all possible stimuli at the input and check the output that is generated by the code.

For a huge chip, this may be a very involved job. Once the designer is sure that the code functions as expected, he will take the code through the synthesis process to convert it into gates.

(iii) Synthesis

During this phase, the design is target into equivalent gates and flops. The output from the synthesis phase is often referred as the ***netlist***, which represents the connectivity of the cells used to realise the logic.

The tool will read the verified code and map the logic functions into the relevant gates/flops based on target libraries provided by the silicon vendor, which depends on the type of technology the designer intended to use for the chip.

A sythesized netlist from a synthesis tool will look like the one shown below.

```
module sample ( clk, a, b, c, c_reg );
input clk, a, b;
output c, c_reg;

FD1 c_reg_reg ( .Q(c_reg), .D(c), .CP(clk) );
OR2 U9 ( .Z(c), .A(b), .B(a) );

endmodule
```

Figure 6-7 A synthesized netlist in Verilog HDL

The above representation is also in Verilog except that the file is having hard cells in them instead of the behavioural descriptions. The cell “OR2” represents a two-input OR gate from the technology library and the cell “FD1” represents a D-flip-flop. This flop’s D-input is same as the output “c” and output-Q of the flop is connected to “c_reg”. The flop is clocked by the signal “clk”. A pictorial representation of the logic is shown in Figure 6-8.

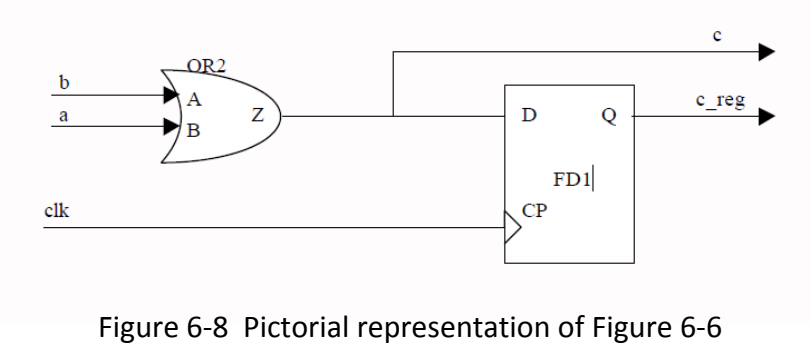


Figure 6-8 Pictorial representation of Figure 6-6

Since this netlist also represents the design, one can run the functional verification tests against this netlist. A Static Timing Analysis (STA) on the netlist which reveals any potential timing problems like a setup or hold violation in the design is then run. If there are some timing errors, those have to be fixed.

6.1.1.2 Backend Design

The second major phase in the ASIC design is commonly known as the backend where the cells in the netlist are placed on the die and then routed. The netlist, which has no violations is read into the placement tool and placed within the die area according to the guidelines given to the tool.

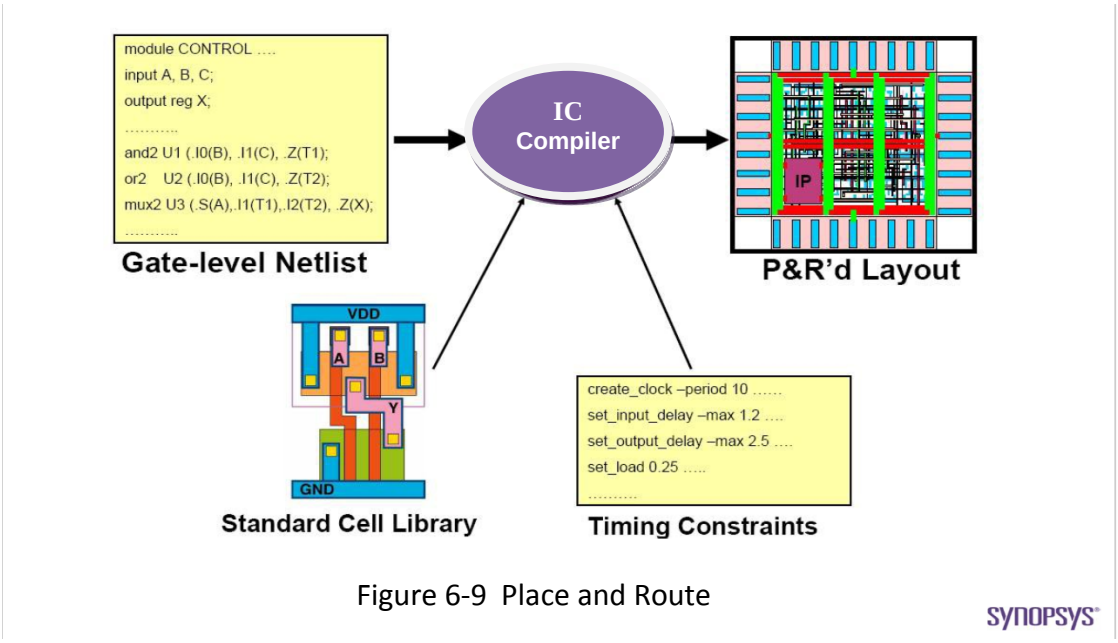


Figure 6-9 Place and Route

(i) Placement

Placement of cells depends on the connectivity and also on the distance between the cells so as not to cause any timing violation. Cells that are placed far apart will have a lengthy wire to connect them, and this will cause additional delay. Another reason for more delay is the fanout. When the fanout of a cell increases, this increases the capacitance load on the driving cell. This will also cause output signal to propagate slowly from one point to another.

The placement tool will be given a set of constraints that will indicate the timing requirements of the design. Hence the tool will try to place the cells in such a way that there are no timing deteriorations. In addition to the cells, the tool will place the IO pads also along the periphery of the die so that the pins can be connected to the pads using metal bonding at a later stage.

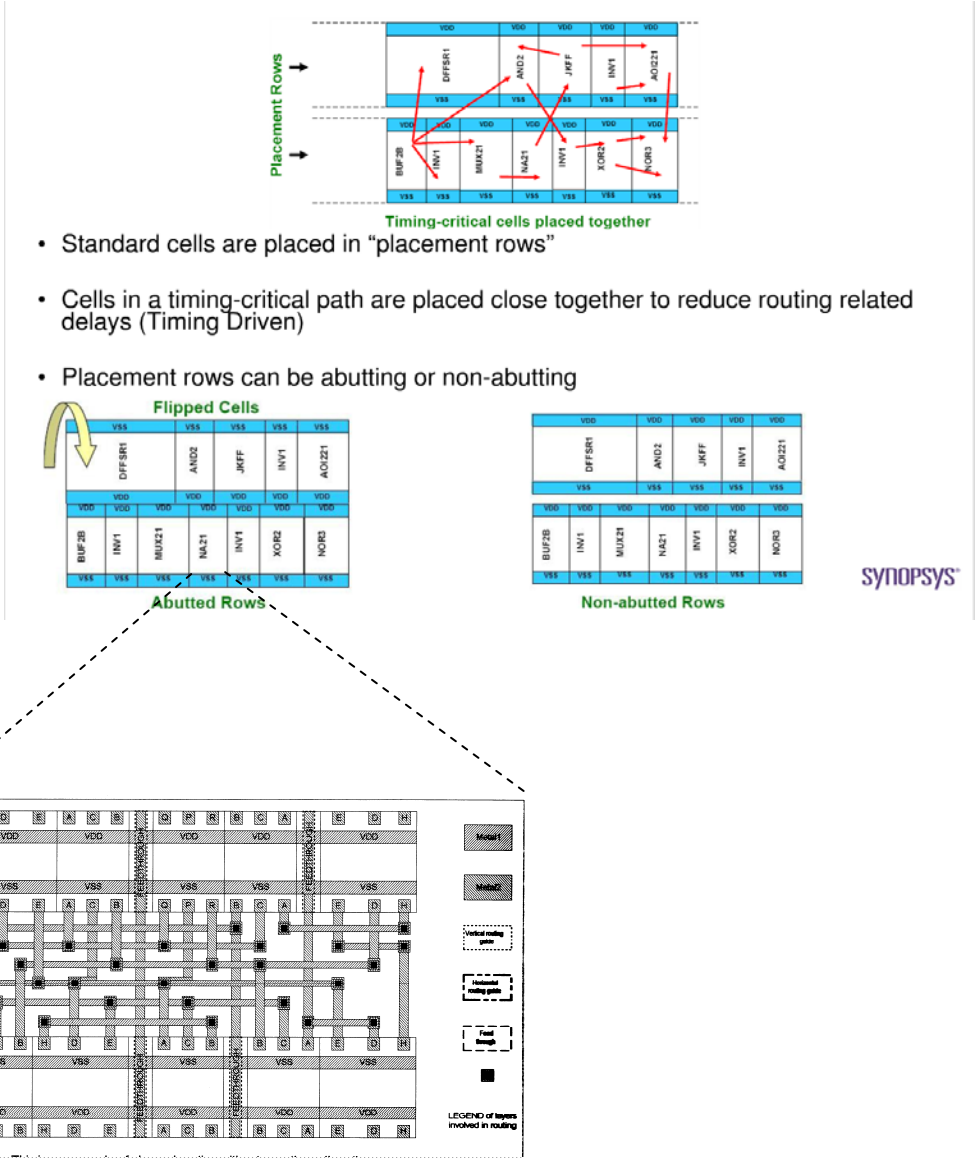


Figure 6-10 Concept of Placement

(ii) Routing

Once the cells and IO pads are placed satisfactorily, the design is given to the routing tool for routing. This tool will connect the cells according to the information in the netlist.

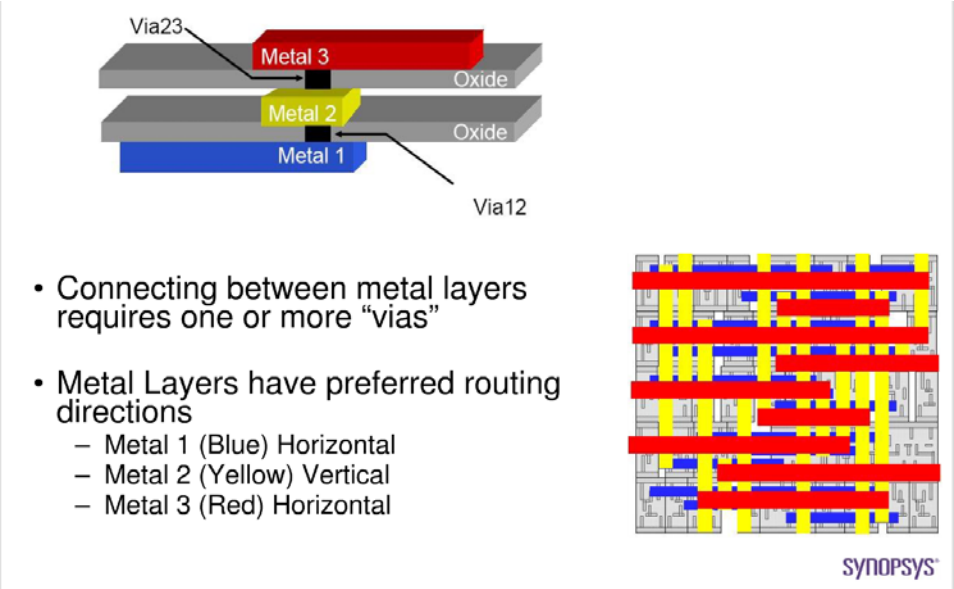


Figure 6-11 Concept of Routing

When the routing is completed the designer can extract the actual delay contributed by the cells and the wires. These timing numbers are to be fed to the STA tool once again to determine the conditions of violations based on the final routing. If there are any violations, they need to be fixed by performing some netlist changes or by improving the routing/placement or by doing both. This will take few iterations to converge on the timing.

When the backend phase is completed with no violations, the design is converted into a database that will be used to build the mask for the semiconductor processing. These data will be given to the processing units for manufacturing the chip.

6.2 Gate Array Design Techniques

A gate array or uncommitted logic array (ULA) is an approach to the design and manufacture of ASICs. A gate array circuit is a prefabricated silicon chip circuit with no particular function in which transistors, standard NAND or NOR logic gates, and other active devices are placed at regular predefined positions and manufactured on a wafer, usually called a *master slice*.

Circuit with a specified function is accomplished by adding a final surface layer or layers of metal interconnects to the chips on the master slice late in the manufacturing process, joining these elements to allow the function of the chip to be customised as desired. This layer is analogous to the copper layer(s) of a PCB.

Gate array master slices are usually prefabricated and stockpiled in large quantities regardless of customer orders. The design and fabrication according to the individual customer specifications may be finished in a shorter time compared with standard cell or full-custom design.

The gate array approach reduces the mask costs since fewer custom masks need to be produced. In addition manufacturing test tooling lead time and costs are reduced since the same test fixtures may be used for all gate array products manufactured on the same die size. Gate arrays were the predecessor of the more advanced structured ASIC.

An application circuit must be built on a gate array that has enough gates, wiring and I/O pins. Since requirements vary, gate arrays usually come in families, with larger members having more of all resources, but correspondingly more expensive.

The main drawbacks of gate arrays are their somewhat lower density and performance compared with other approaches to ASIC design. However this style is often a viable approach for low production volumes.

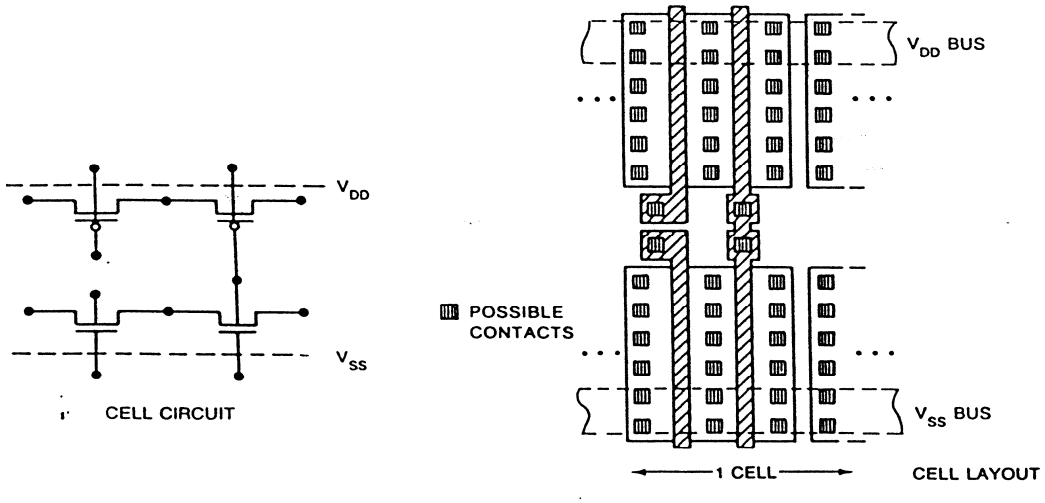


Figure 6-12 Single Cell Circuit and its Corresponding Layout

Gate Arrays are classified by the followings:

- the specifications of the cell on the chip (# of devices)
- the number of such cells per chip
- the number of I/Os per chip
- the fabrication process technology used
- the number of interconnection layers available for customization

Each die consists of rows of butting identical cells separated by wiring channels. I/O pads are around the perimeter of this active area.

6.2.1 Gate Array Design Flow

The Front End design process is similar to the standard cell technique. After all the schematics for the design are determined and simulated, they are converted to gate array macros. This process converts the schematic blocks to transistor interconnections that serve the same logic function.

The Back End design process involves routing connections of the cells to match the gate array macros. The placement of cells and routing channels are fixed, thus routing may not take the optimum path, resulting in slower performance.

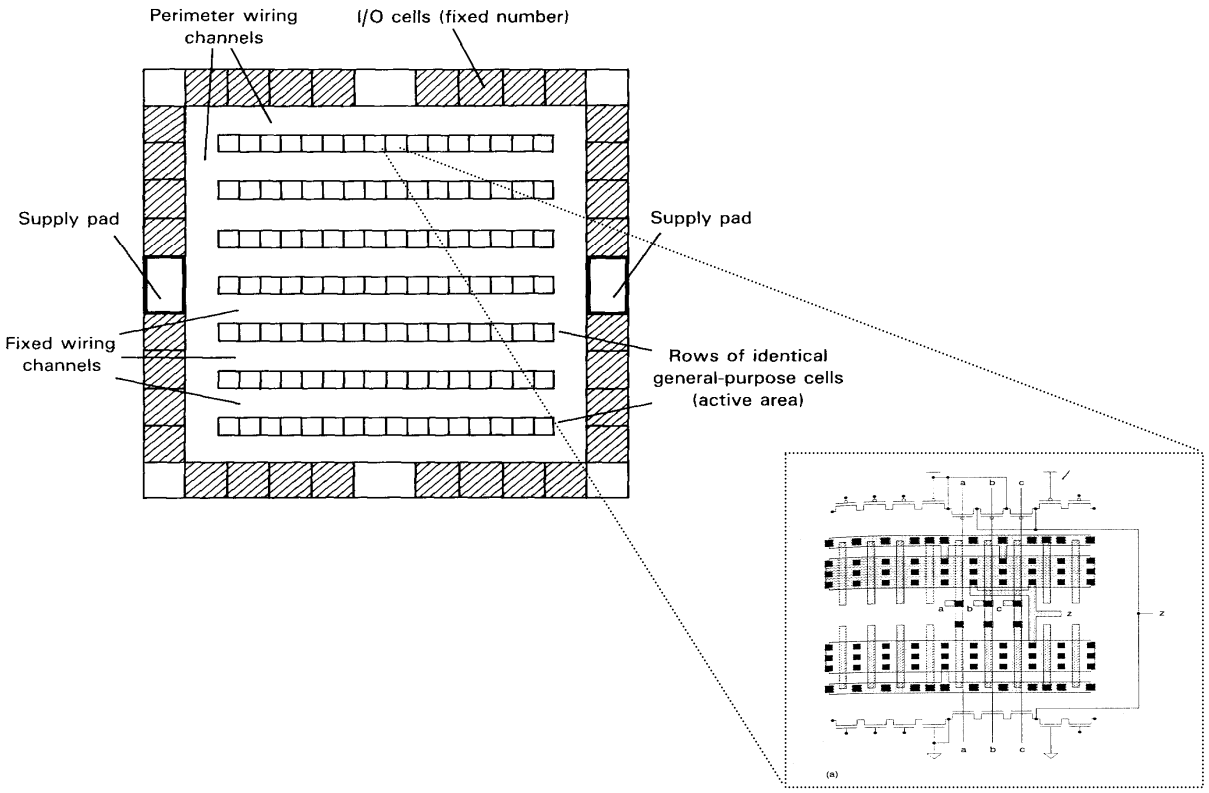


Figure 6-13 Floor plan of Gate Array IC

6.3 Field Programmable Devices

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an ASIC.

FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, and the low non-recurring engineering costs relative to an ASIC design offers advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

The most compelling advantages of FPGAs are low startup cost, low financial risk, and, because the end user programs the device, quick manufacturing turnaround and easy design changes.

Figure 6-14 shows the internal structure of an FPGA. There are three key parts of a FPGA structure, namely logic blocks, interconnect, and I/O blocks. The I/O blocks form a ring around the outer edge of the part. Each of these provide individually selectable input, output, or bi-directional access to one of the general purpose I/O pins on the exterior of the FPGA package. Inside the ring of I/O blocks lies a rectangular array of logic blocks. Programmable interconnect wirings connect logic blocks to logic blocks and I/O blocks to logic blocks.

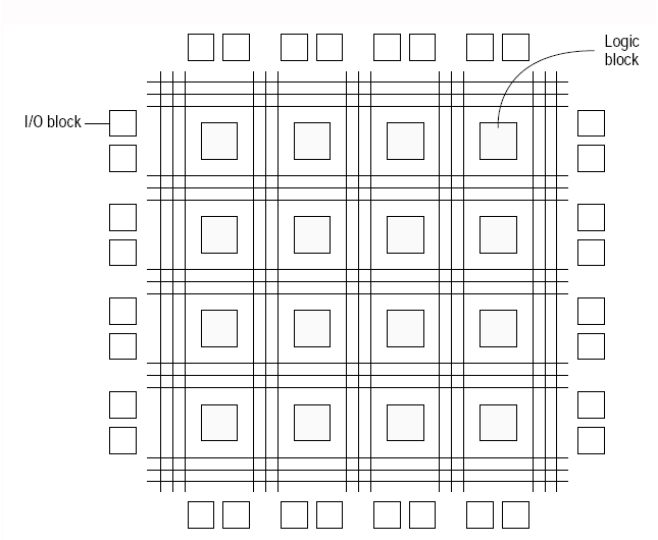


Figure 6-14 Internal structure of an FPGA

6.3.1 FPGA Design Flow

The process of creating digital logic is through a description of the hardware's structure and behaviour written in a high-level hardware description language (usually VHDL or Verilog. The code is then compiled and downloaded prior to execution. Schematic capture is also an option for design entry, but it has become less popular as designs have become more complex and the language-based tools have improved.

6.3.1.1 Design Entry (Coding)

During design entry, hardware designers must think-and program-in parallel. All of the input signals are processed in parallel, as they travel through a set of execution engines – each one a series of macrocells and interconnections – toward their destination output signals. Therefore, the statements of a hardware description language create structures, all of which are “executed” at the very same time.

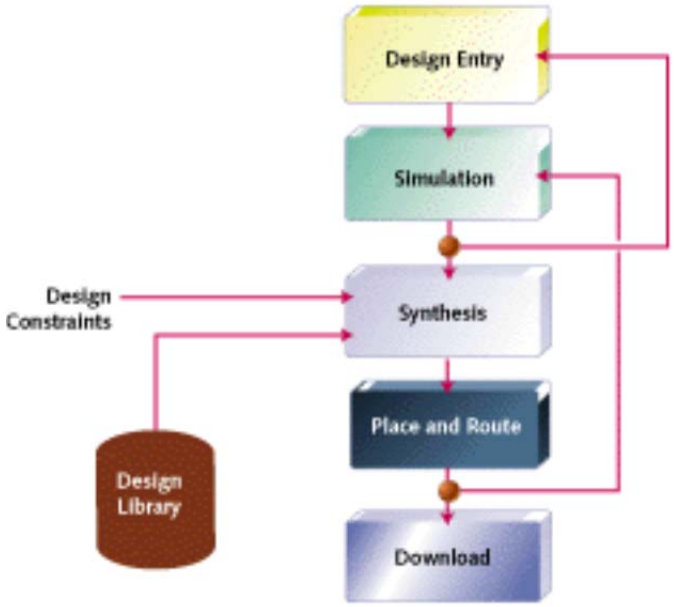


Figure 6-15 FPGA Design Flow

6.3.1.2 Simulation (Verification)

Typically, the design entry step is followed or interspersed with periods of functional simulation. A simulator is used to execute the design and confirm that correct outputs are produced for a given set of test inputs. Although problems with the size or timing of the hardware may still crop up later, the logic functionality is ensured before the next stage of development.

6.3.1.3 Synthesis

During this step, a representation of the hardware based on the functionally correct design will be generated. The result is a netlist which is device independent. It is usually stored in a standard format called the Electronic Design Interchange Format (EDIF).

6.3.1.4 Place & Route

This step involves mapping the logical structures described in the netlist onto actual macrocells, interconnections, and input and output pins. The result of the place & route process is a bitstream, which is the binary data that must be loaded into the FPGA or CPLD to cause that chip to execute a particular hardware design.

6.3.1.5 Device Programming

Once the bitstream for a particular FPGA or CPLD has been created, it can be downloaded it to the device. The details of this process are dependent upon the chip's underlying process technology.

6.3.2 Programming Technologies

Programmable logic devices are like non-volatile memories in that there are multiple underlying technologies. In fact, exactly the same set of names is used: PROM (for one-time programmable), EPROM, EEPROM, and Flash.

6.3.2.1 Non-Volatile Technologies

PROM and EPROM-based logic devices can only be programmed with the help of a separate piece of lab equipment called a device programmer.

One common approach used is the fusible link method. Figure 6-16(a) shows the construction of a fusible link. It is normally open until programmed to close. To program it, the programmer applies a high voltage to the link, this causes the dielectric to breakdown, thus forming a connection as shown in Figure 6-16(b).

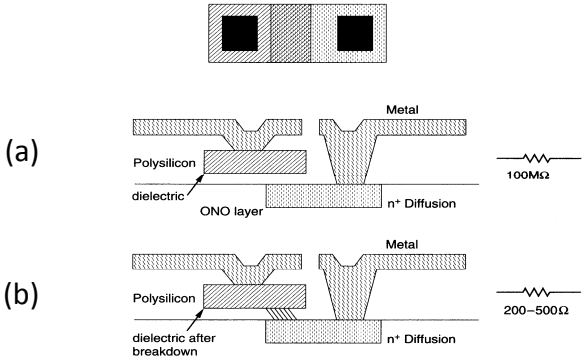


Figure 6-16 Fusible Link

EEPROM or Flash technology devices allow re-programming. Additional circuitry that's required to perform device (re)programming is provided within the FPGA or CPLD silicon as well. This makes it possible to erase and reprogram the device internals via a JTAG interface or from an on-board embedded processor. (Note, however, that because this additional circuitry takes up space and increases overall chip costs, a few of the programmable logic devices based on EEPROM or Flash still require insertion into a device programmer.

Figure 6-17 shows a floating gate MOS transistor. The floating gate (Gate 1) is unconnected and surrounded by extremely high impedance oxide. In the original state, the floating gate has no charge on it and has no effect on the circuit operation. Hence, it effectively functions like a normal transistor.

To program it, the programmer applies a high voltage to the non-floating gate (Gate 2) at each location where a logical link is not wanted. This causes a temporary breakdown in the insulating material and allows a negative charge to accumulate on the floating gate. The negative charge prevents the transistor from turning "on" when a HIGH signal is applied to the non-floating gate; the transistor is effectively disconnected from the circuit.

The charges at the floating gate can be "erased" by exposing the transistor to UV light.

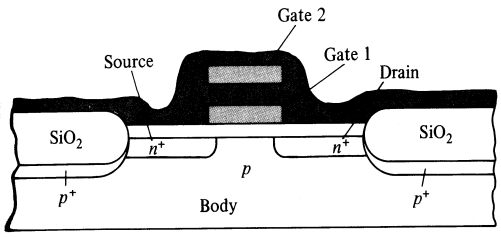


Figure 6-17 UV Erasable Floating Gate

6.3.2.2 Volatile Technologies

In addition to non-volatile technologies, there are also programmable logic devices based on SRAM technology. In such cases, the contents of the device are volatile. This has both advantages and disadvantages.

One advantage is that the content of the logic device can be manipulated on-the-fly so that the hardware design could be upgraded as easily as software.

The obvious disadvantage is that the internal logic must be reloaded after every system or chip reset. That means you'll need an additional memory chip of some sort in which to hold the bitstream.

6.3.3 Applications

Many times a CPLD or FPGA will be used in a prototype system. A small device may be present to allow the designers to change a board's glue logic more easily during product development and testing. A large device may be included to allow prototyping of a system-on-a-chip design that will eventually find its way into an ASIC. Either way, the basic idea is the same: allow the hardware to be flexible during product development. When the product is ready to ship in large quantities, the programmable device will be replaced with a less expensive, though functionally equivalent, hard-wired alternative.

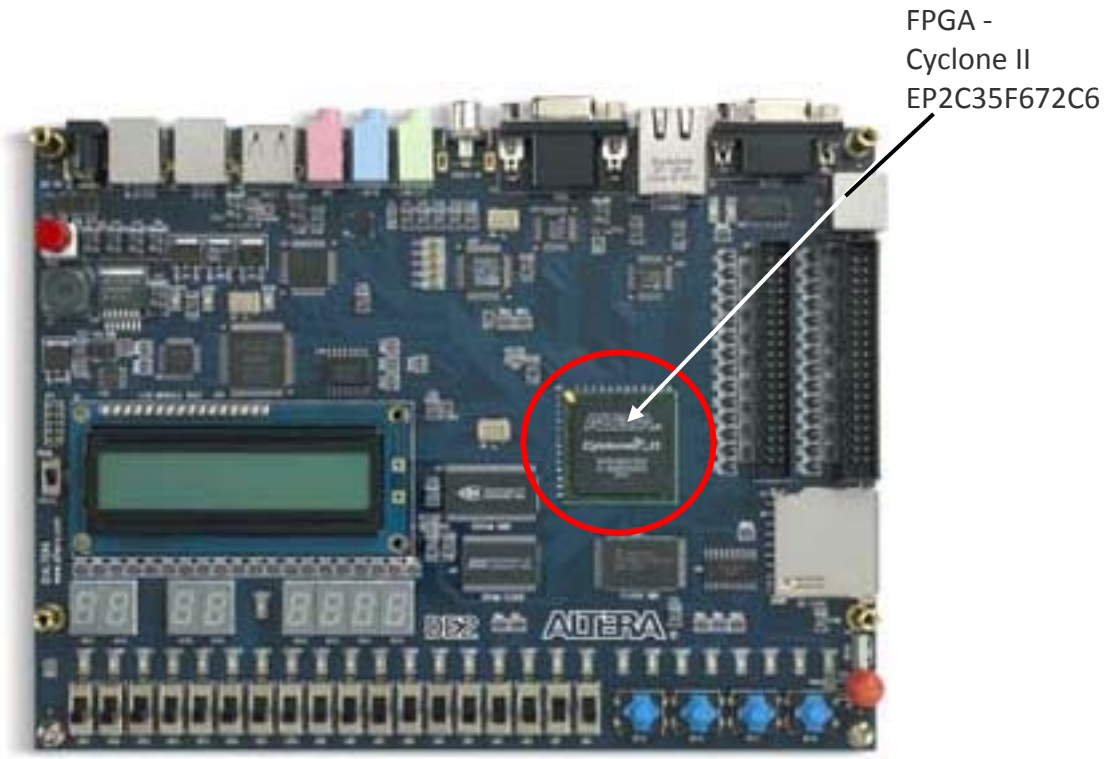


Figure 6-18 Altera DE 2 Board

6.4 Structured ASIC

Structured ASIC is a type of application specific IC (ASIC) chip. Its performance approaches that of Standard Cell ASIC while dramatically simplifying the design complexity. Structured ASICs offer designers a set of devices with specific, customizable metal layers along with predefined metal layers, which can contain the underlying pattern of logic cells, memory, and I/O. By virtue of the predefined structures, the ASIC vendor can design any time-consuming task — such as test, signal integrity and IR drop — into the architecture.

Unlike a standard cell, which requires all the masking stages in the transistors and metallization layers to be fabricated, structured ASICs require only one or two masks to tie the blocks together. In some design, only the vias need to be made conductive in order to complete the chip.

In addition to functional IP, Structured ASIC products also contain large amounts of high-performance, initializable macros. Figure 6-19, shown below, illustrates the architecture of a Structured ASIC.

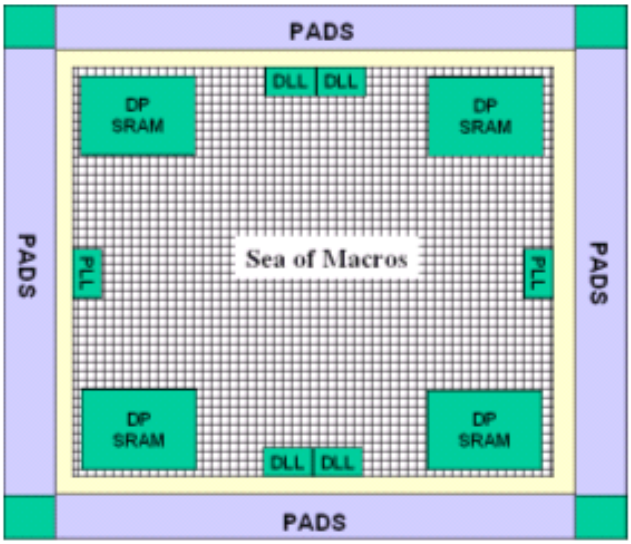


Figure 6-19 Structured ASIC Block Diagram

The manufacturing cycle for a Structured ASIC is considerable shorter than that required for a cell-based ASIC as fewer layers need to be processed. This significantly lowers the NRE cost.

With the capacity and performance optimization inherent in Structured ASIC technology, SoC applications in the millions of gates can operate at speeds up to 200 MHz, making Structured ASIC technology robust enough to meet the demand of modern SoC applications. That, coupled, with the cost savings afforded by the hybrid nature of the technology, easily demonstrate how Structured ASIC technology can fill the void left by cell-based ASIC and FPGA technologies.

Figure 6-20, illustrates the metal utilization of a structured ASIC. The base architecture, up through M2, contains the 'sea-of-macros', and a portion of the embedded IP, both comprised of high-performance Deep Sub-micron (DSM) transistors. Some of the embedded IP will utilize layers in the base as well as some or all of the programmable layers, depending on the complexity and performance requirements of the IP. The programmable metal layers, M3 to M5, are a more mature technology and are used to route the custom applications. Soft, synthesizable, IP will also be routed in these programmable metal layers.

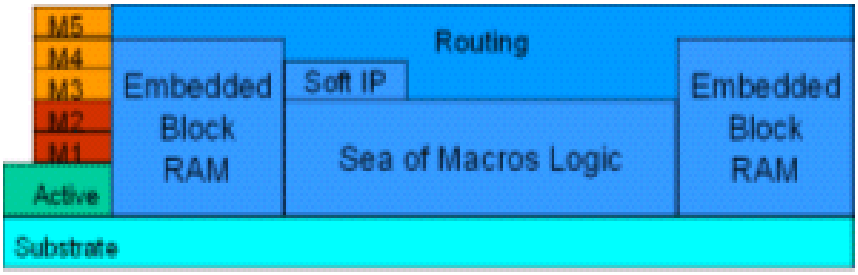


Figure 6-20 Structured ASIC Metal Utilization

6.5 Technical Comparisons

6.5.1 Standard Cell and Full Custom Design

Standard Cell	Full Custom Design
Cells are designed to interface horizontally with any other cell in the library	Design to interface only with transistors within its own block
Cell height is fixed/or in multiples for macro cells	Cell height is variable and customizable and independent of all other cells
Transistor sizes for the gates are fixed and is selected from library data	All transistor sizes are customized for maximum performance
Floor planning, placement and routing are done with aid of EDA software	Placement and routing of devices are usually done manually
Power lines in the cell rows are calculated for average dissipation	All power lines are designed to sustains a pre-defined current loading
Emphasis on the ability to use EDA tools efficiently	Considerable design experience required
Short design cycle	Long design cycle
For complex design up the several hundred thousand gates	For small design (up to several hundred devices) where emphasis is on performance (e.g. analogue functions)

6.5.2 Standard Cell and Gate Array Approach

Standard Cell	Gate Array
Designer controls placement and routing of cells	Placement is fixed and routing iterations are greatly reduced. Routing may not take optimum path; results in slower performance
The number of gates required depends on the design	Fixed number of cells/gates per chip resulting in unused cells
ROMs and other custom blocks can be included (block layout height fit into the multiple the standard cell's height)	Difficult to include macro blocks such as ROMs, PLAs in design
The number of I/Os pads depends on the design	The number of I/O pads are fixed and may results in unused pads
The chip size is variable and depends on the designer's skill to compact the design through placement and routing control	The chip size is fixed (depends on the number of cells per chips and its I/O pads). Routing channels are fixed
Full set of mask is required for fabrication of the IC	Only the interconnects mask layers (e.g. metal1, via, metal2, etc) are required for fabrications
More costly fabrication cost and need longer lead time	Less lead time since only interconnections are processed
Testing cost is higher since test fixtures are customized for the specified design	Testing cost is lower since test fixtures are used for multiple designs

6.5.3 Comparison of Various Design Approaches

	Off -the-shelf Devices			Vendor Customized	
	SSI and MSI IC	**PLDs**	**PGA**	**Gate Array**	**Standard Cell**
Capacity (# of gates)	10 to few hundreds	100s to few thousands	1000s to ten thousands	1000s to ten thousands	10,000s to hundred thousands
Speed	Medium To Fast	Slow to Medium	Slow to Medium	Slow to Fast	Medium to Fast
Functionality Defined by user	No	Yes	Yes	Yes	Yes
Time to customized	-	Seconds	Seconds	Depends upon many factors, up to months	
User programmable	No	Yes	Yes	No	No

6.5.4 Cell-Based vs FPGA vs Structured ASIC

	Cell-Based	FPGA	Structured ASIC
Production Volume	¼ of a Million units	Few Thousand units	5K to 1M units
Architecture	Sea-of-Gates	Sea-of-Macros with programmability	Sea-of-Macros
Performance	Good	Good	Good
Cost (NRE)	High	Low	Moderate
Time-to-Market	Fast	Fastest	Faster
Design Cycle	Long	Short	Moderate

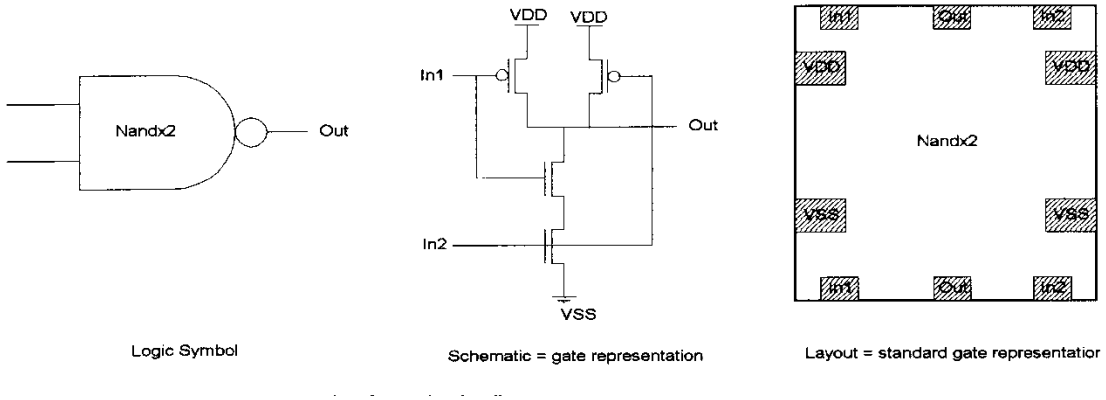
Review Questions

Question 1

Briefly describe the semi-custom ASIC design techniques :
(a) Standard Cell Techniques.
(b) Gate Array Techniques

Solution to Q1

(a) This technique relies on the existence of previously designed and fully characterized standard circuits and layout data held in a CAD database library. The cell libraries consists standard logic gates, I/O pads, etc of a particular process technology, which are provided by the fabrication vendor.



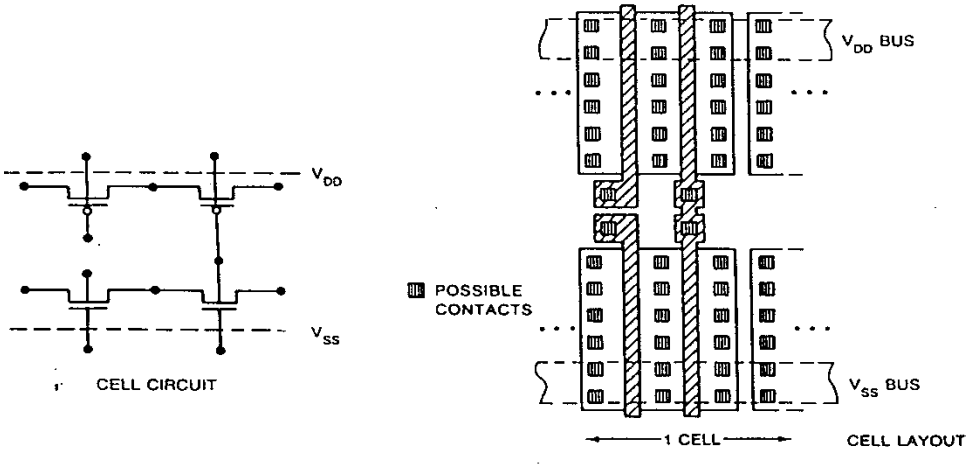
Database of a NAND2 standard cell

In the layout, only the access pins and power pins are visible for interconnects. The entire mask layout for the devices within gate is also available in the library but cannot be modified.

There are various sizes of the same device in the library. If a larger driver is required for the NAND2 gate, then another similar NAND2 gate with larger transistor size is selected from the library to meet the requirements.

The entire standard cells layout has the same height. This enables the cells to be placed alongside each other in uniform rows across the chip. The width of the cells is variable to cater for different driver sizes and different types of gates.

(b) Gate Array design techniques rely on the availability of wafers that have been fully processed up to, but not including the final chip interconnections. These items are kept as stock items by the vendors and each die on the wafer contains an array of identical general-purpose cells.



Single Cell Circuit and its Corresponding Layout

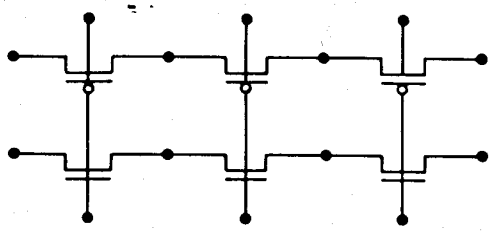
Gate Arrays are classified by the followings:

- the specifications of the cell on the chip (# of devices)
- the number of such cells per chip
- the number of I/Os per chip
- the fabrication process technology used
- the number of interconnection layers available for customization

Each die consists of rows of butting identical cells separated by wiring channels. I/O pads are around the perimeter of this active area.

Question 2

Implement a 2 inputs "NOR" logic gate on the Site schematic and Mask layout as shown in Figure 6-21.



SITE SCHEMATIC

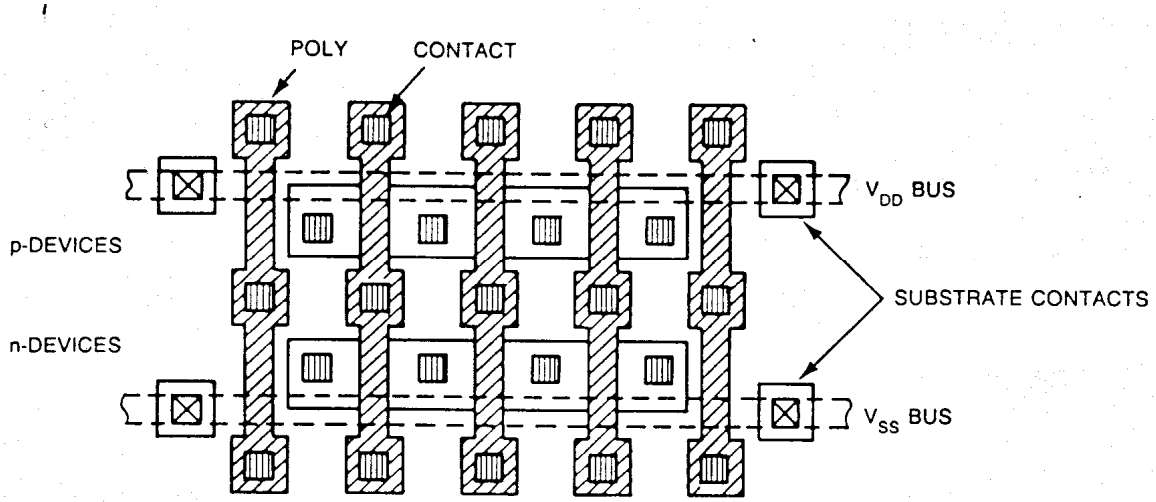
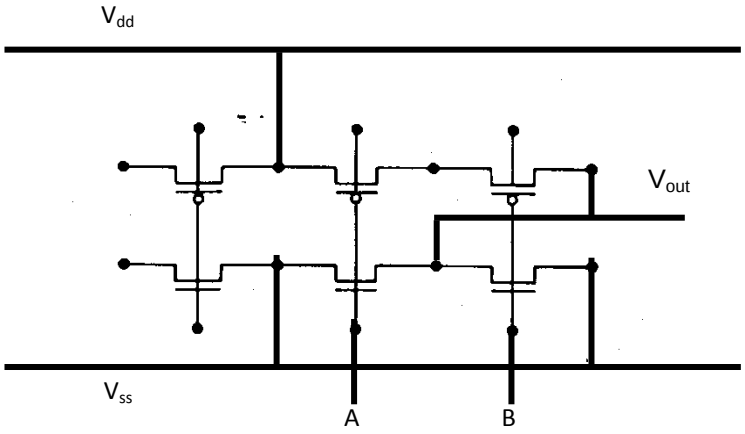
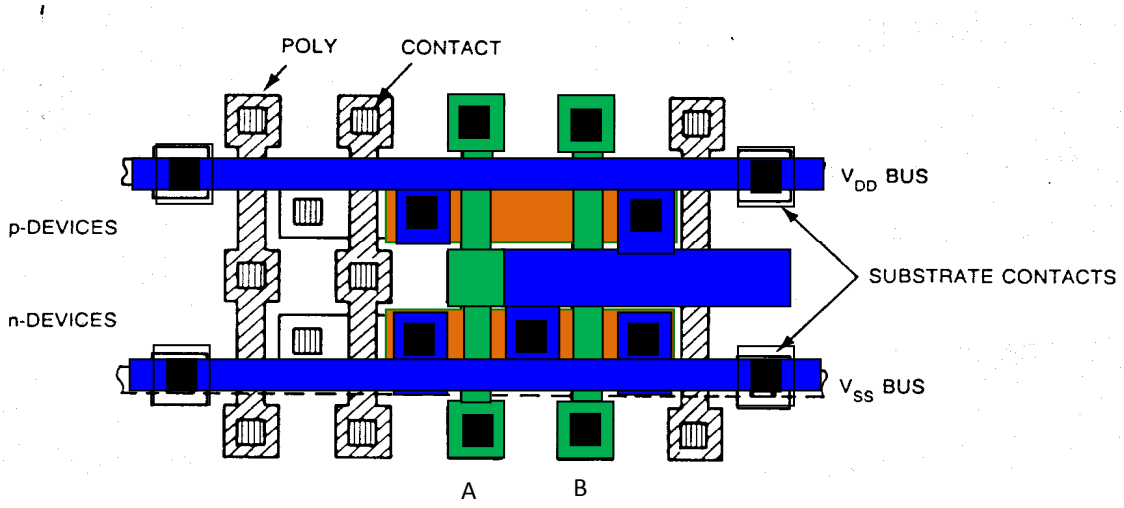


Figure 6-21

Solution 2

Implementation of a 2 inputs “NOR” gate using a Gate Array Cell.





Question 3

List the advantages Standard Cells technique has over Full Custom Design approach.

Solution to Q3

Standard Cell	**Full Custom Design**
Emphasis on the ability to use EDA tools efficiently	Considerable design experience required
Short design cycle	Long design cycle
For complex design up the several hundred thousand gates	For small design (up to several hundred devices) where emphasis is on performance (e.g. analogue functions)

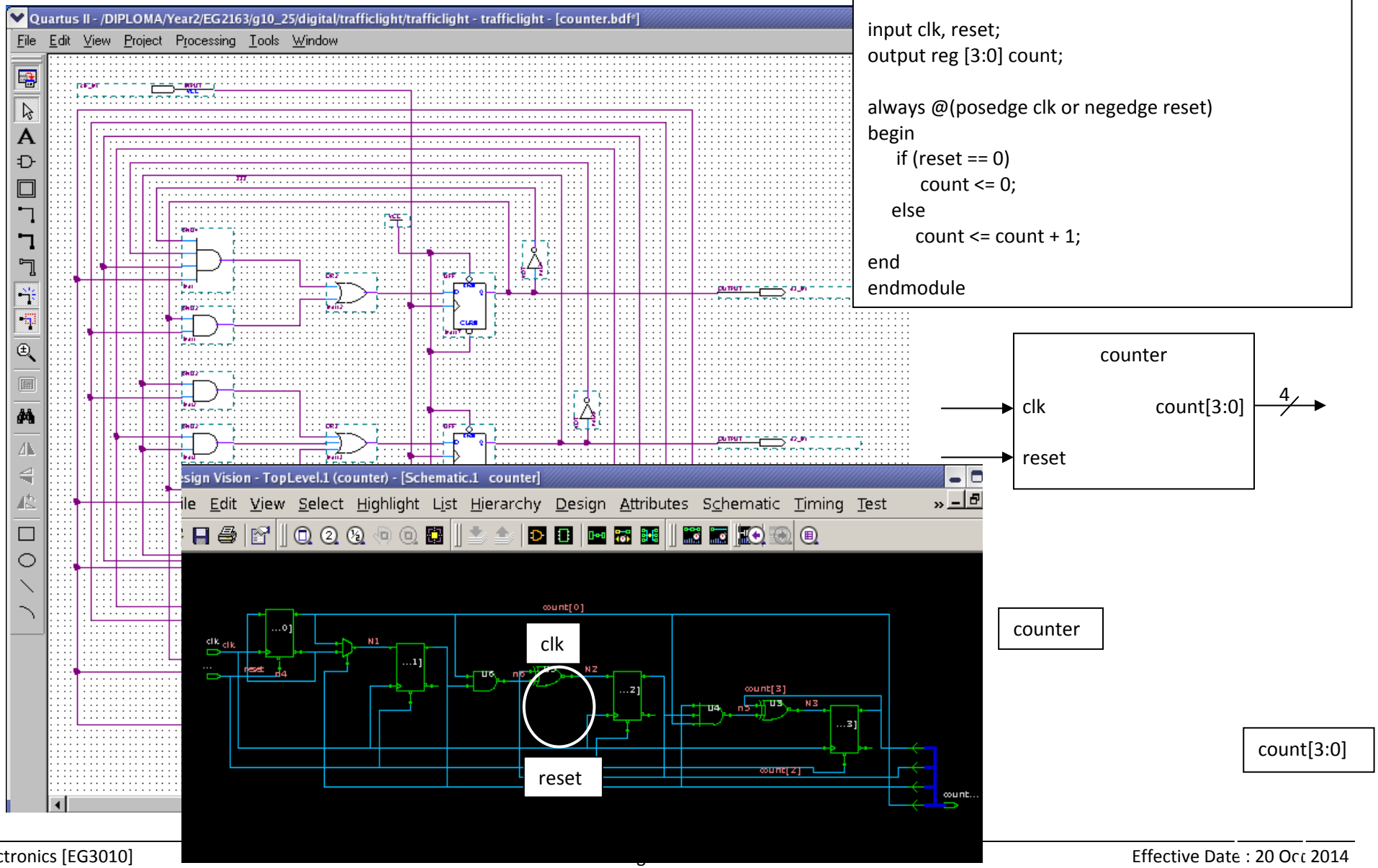
Question 4

List the advantages Standard Cells technique has over the Gate Array approach.

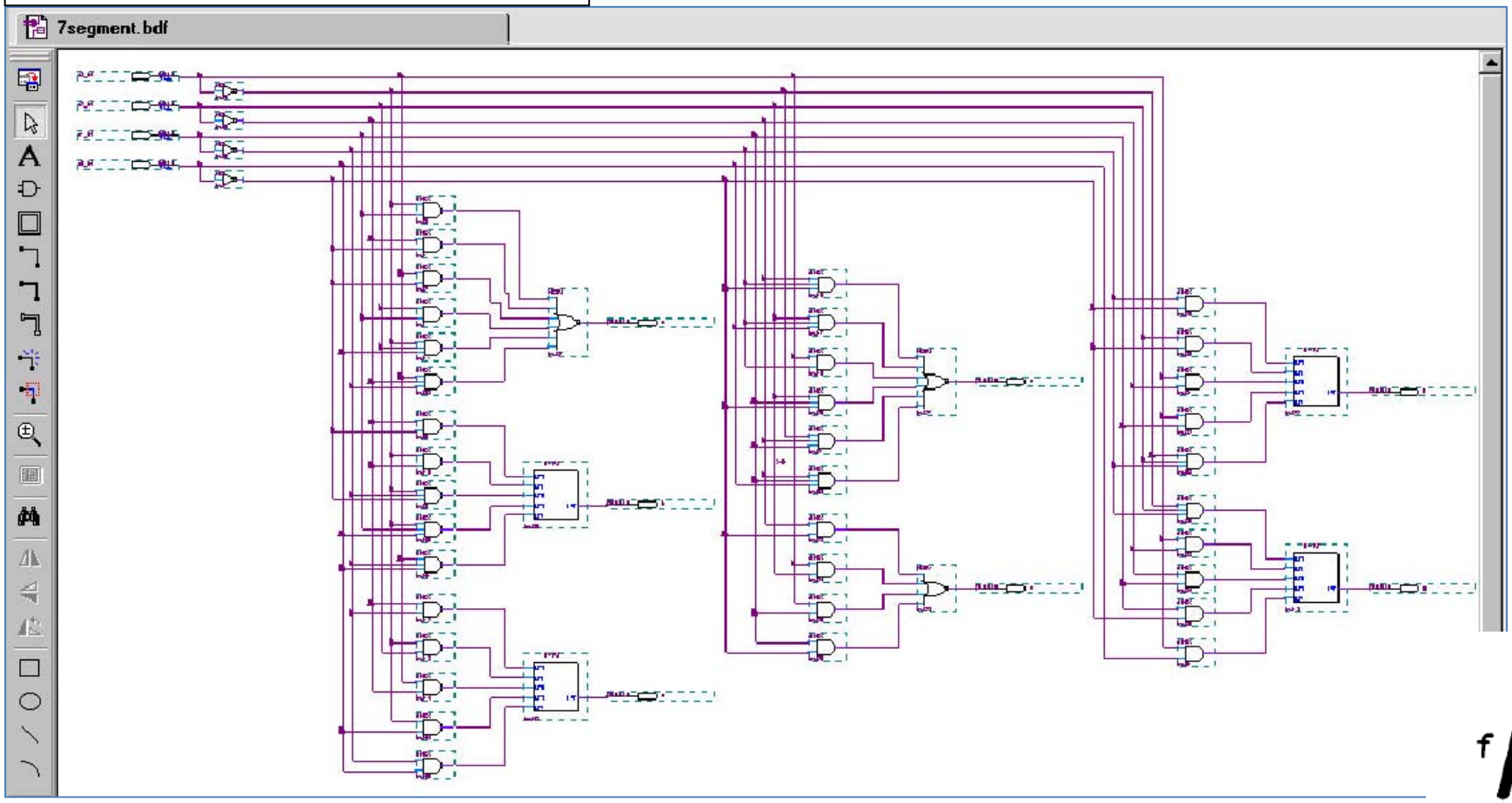
Solution to Q4

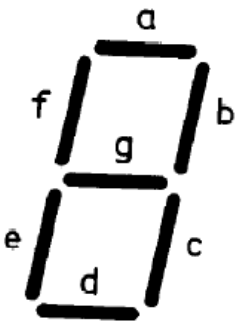
Standard Cell	**Gate Array**
The number of gates required depends on the design	Fixed number of cells/gates per chip resulting in unused cells
The number of I/Os pads depends on the design	The number of I/O pads are fixed and may results in unused pads
The chip size is variable and depends on the designer's skill to compact the design through placement and routing control	The chip size is fixed (depends on the number of cells per chips and its I/O pads). Routing channels are fixed
Full set of mask is required for fabrication of the IC	Only the interconnects mask layers (e.g. metal1, via, metal2, etc) are required for fabrications
More costly fabrication cost and need longer lead time	Less lead time since only interconnections are processed
Testing cost is higher since test fixtures are customized for the specified design	Testing cost is lower since test fixtures are used for multiple designs

```
module counter (clk, reset, count);

input clk, reset;
output reg [3:0] count;

always @(posedge clk or negedge reset)
begin
    if (reset == 0)
        count <= 0;
    else
        count <= count + 1;
end
endmodule
```



Schematic of a 7-Segment Decoder (active low)

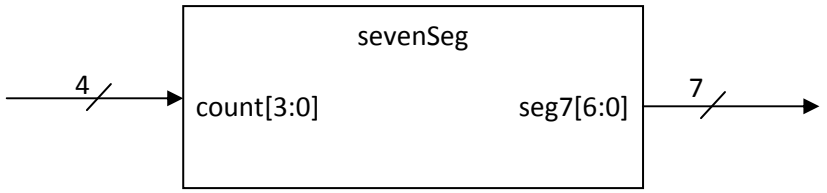




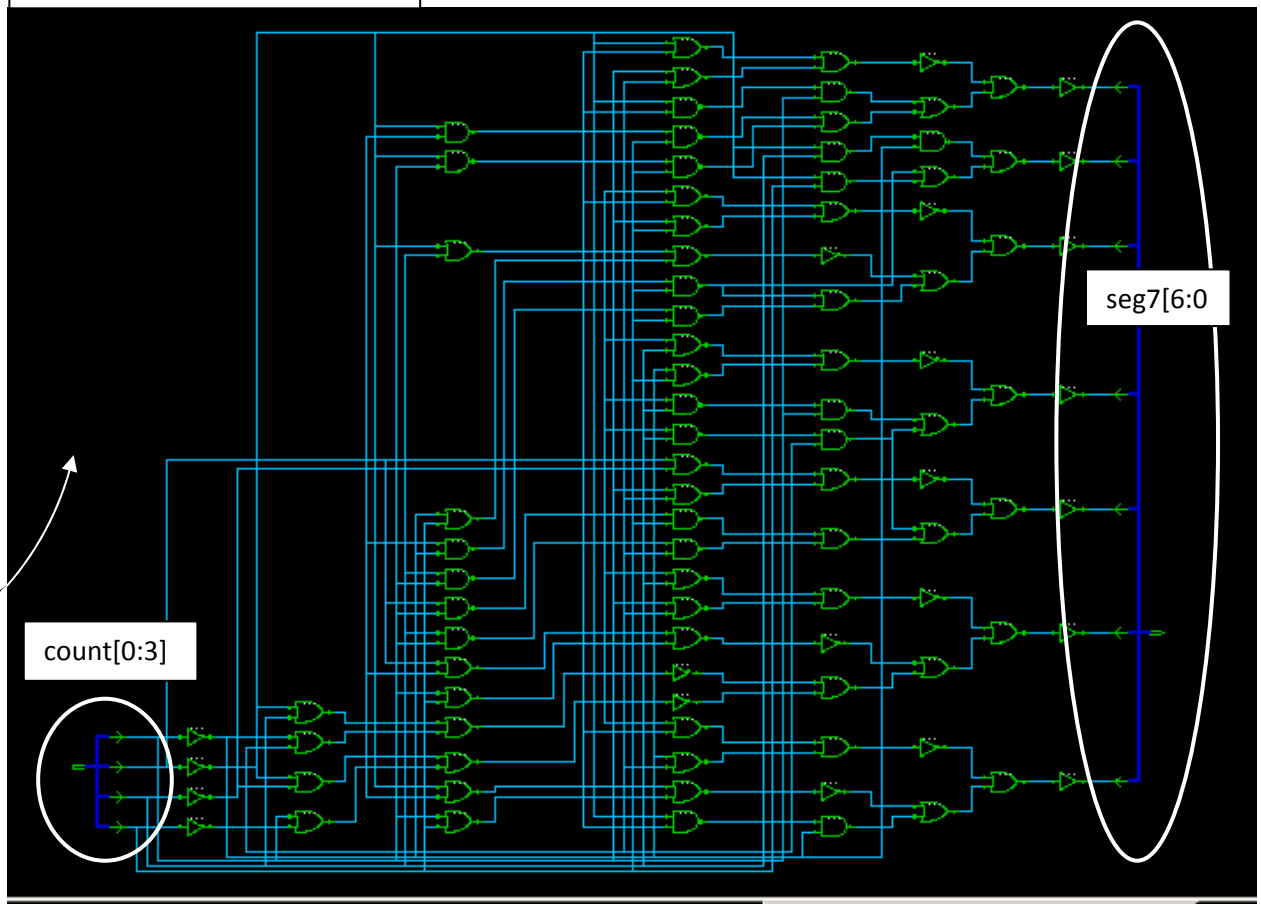
Verilog Codes - sevenSeg

```
module sevenSeg (count, seg7);

input [3:0] count;
output reg [6:0] seg7;

always @(count)
begin
   case (count)
      4'b0000: seg7 = 7'b0000001;  // a b c d e f g
      4'b0001: seg7 = 7'b1001111;
      4'b0010: seg7 = 7'b0010010;
      4'b0011: seg7 = 7'b0000110;
      4'b0100: seg7 = 7'b1001100;
      4'b0101: seg7 = 7'b0100100;
      4'b0110: seg7 = 7'b0100000;
      4'b0111: seg7 = 7'b0001111;
      4'b1000: seg7 = 7'b0000000;
      4'b1001: seg7 = 7'b0001100;
      4'b1010: seg7 = 7'b0001000;
      4'b1011: seg7 = 7'b1100000;
      4'b1100: seg7 = 7'b0110001;
      4'b1101: seg7 = 7'b1000010;
      4'b1110: seg7 = 7'b0110000;
      4'b1111: seg7 = 7'b0111000;
   endcase
end

endmodule
```

Block Diagram - sevenSeg

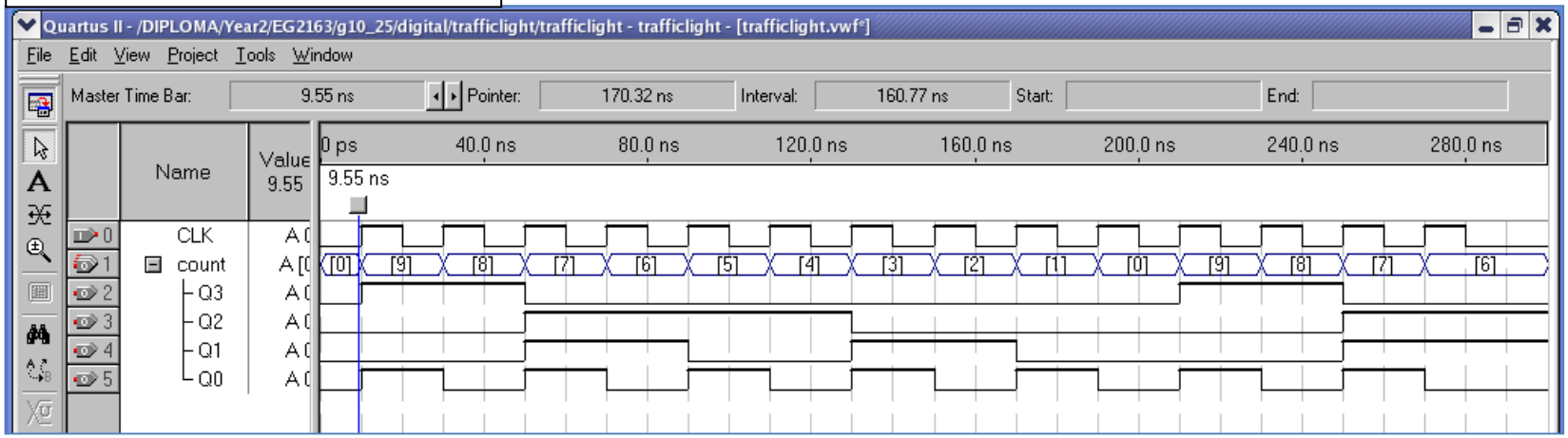


Logic Synthesized - sevenSeg



Logic synthesis is the process of converting design codes such as Verilog or VHDL into logic gates for implementation.

Simulation waveform – 9 to 0 Counter



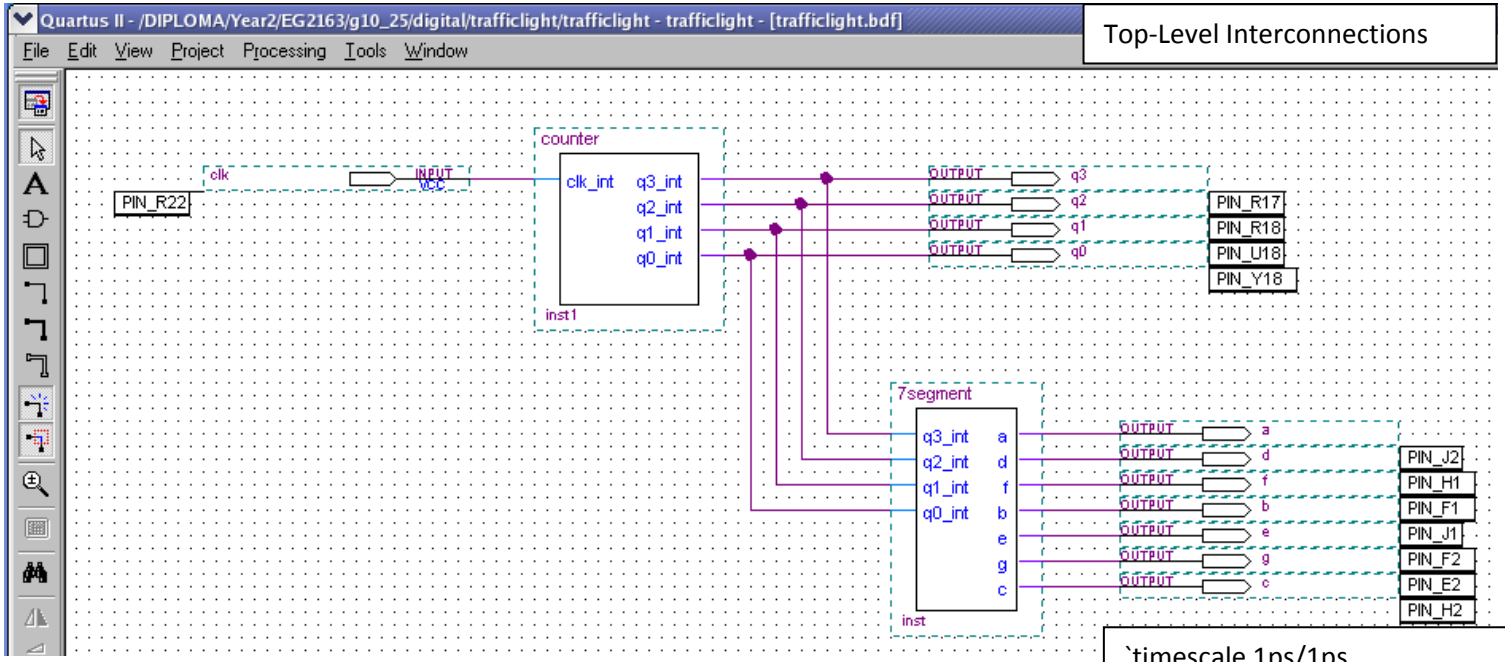
Test Bench

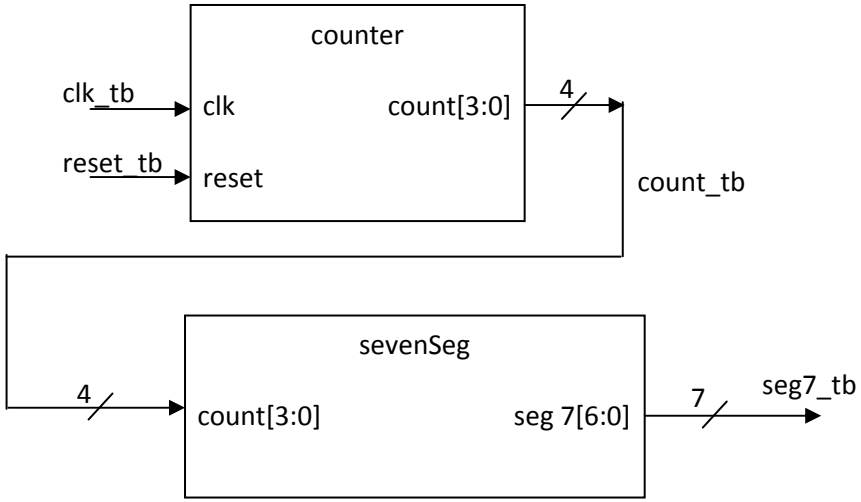
```
`timescale 1ps/1ps
module counter_tb;
reg clk_tb, reset_tb;
wire [3:0] count_tb;

counter u1 (.clk(clk_tb), .reset(reset_tb), .count(count_tb));

initial
begin
   #0   clk_tb = 0;
        reset_tb = 1;
   #10   reset_tb = 0;
   #50   reset_tb = 1;
   #800 $stop;
end
always
   #5  clk_tb = ~clk_tb;
endmodule
```

Test benches provide all possible inputs to the design circuits during functional simulation.

Functional simulation invokes the simulator to verify that the design performs according to specification for all possible input conditions.

Top-Level Interconnections





Test Bench

```
`timescale 1ps/1ps
module counter_tb;
reg clk_tb, reset_tb;
wire [3:0] count_tb;
wire [6:0] seg7_tb ;

counter u1 (.clk(clk_tb), .reset(reset_tb), .count(count_tb));
sevenSeg u2 (.count(count_tb), .seg7(seg7_tb));

initial
begin
   #0   clk_tb = 0;
        reset_tb = 1;
   #10   reset_tb = 0;
   #50   reset_tb = 1;
   #800 $stop;
end
always
   #5  clk_tb = ~clk_tb;
```