UNIT 6: Subsystem Design Processes Illustration

Objectives: At the end of this unit we will be able to understand

- Design consideration, problem and solution
- Design processes
  - Basic digital processor structure
  - Datapath
  - Bus Architecture
- Design 4 – bit shifter
- Design of ALU subsystem
- 4 – bit Adder

General Considerations

- Lower unit cost
- Higher reliability
- Lower power dissipation, lower weight and lower volume
- Better performance
- Enhanced repeatability
- Possibility of reduced design/development periods

Some Problems

1. How to design complex systems in a reasonable time & with reasonable effort.
2. The nature of architectures best suited to take full advantage of VLSI and the technology
3. The testability of large/complex systems once implemented on silicon

Some Solution

Problem 1 & 3 are greatly reduced if two aspects of standard practices are accepted.

1. a) Top-down design approach with adequate CAD tools to do the job
   b) Partitioning the system sensibly
   c) Aiming for simple interconnections
   d) High regularity within subsystem
   e) Generate and then verify each section of the design
2. Devote significant portion of total chip area to test and diagnostic facility
3. Select architectures that allow design objectives and high regularity in realization

Illustration of design processes

1. Structured design begins with the concept of hierarchy
2. It is possible to divide any complex function into less complex subfunctions that is up to leaf cells

3. Process is known as top-down design

4. As a systems complexity increases, its organization changes as different factors become relevant to its creation

5. Coupling can be used as a measure of how much submodels interact

6. It is crucial that components interacting with high frequency be physically proximate, since one may pay severe penalties for long, high-bandwidth interconnects

7. Concurrency should be exploited – it is desirable that all gates on the chip do useful work most of the time

8. Because technology changes so fast, the adaptation to a new process must occur in a short time.

Hence representing a design several approaches are possible. They are:

- Conventional circuit symbols
- Logic symbols
- Stick diagram
- Any mixture of logic symbols and stick diagram that is convenient at a stage
- Mask layouts
- Architectural block diagrams and floor plans

**General arrangements of a 4 – bit arithmetic processor**

The basic architecture of digital processor structure is as shown below in figure 6.1. Here the design of datapath is only considered.

Data path is as shown below in figure 6.2. It is seen that the structure comprises of a unit which processes data applied at one port and presents its output at a second port.
Alternatively, the two data ports may be combined as a single bidirectional port if storage facilities exist in the datapath. Control over the functions to be performed is effected by control signals as shown.

Figure 6.2: Communication strategy for the datapath

Datapath can be decomposed into blocks showing the main subunits as in figure 3. In doing so it is useful to anticipate a possible floor plan to show the planned relative decomposition of the subunits on the chip and hence on the mask layouts.

Figure 6.3: Subunits and basic interconnection for datapath

Nature of the bus architecture linking the subunits is discussed below. Some of the possibilities are:

**One bus architecture:**

Sequence:
1. 1st operand from registers to ALU. Operand is stored there.
2. 2nd operand from register to ALU and added.
3. Result is passed through shifter and stored in the register
Two bus architecture:

Figure 6.5: Two bus architecture

Sequence:
1. Two operands (A & B) are sent from register(s) to ALU & are operated upon, result S in ALU.
2. Result is passed through the shifter & stored in registers.

Three bus architecture:

Figure 6.6: Three bus architecture

Sequence:
Two operands (A & B) are sent from registers, operated upon, and shifted result (S) returned to another register, all in same clock period.

In pursuing this design exercise, it was decided to implement the structure with a 2 – bus architecture. A tentative floor plan of the proposed design which includes some form of interface to the parent system data bus is shown in figure 6.7.

Figure 6.7: Tentative floor plan for 4 – bit datapath
The proposed processor will be seen to comprise a register array in which 4-bit numbers can be stored, either from an I/O port or from the output of the ALU via a shifter. Numbers from the register array can be fed in pairs to the ALU to be added (or subtracted) and the result can be shifted or not. The data connections between the I/O port, ALU, and shifter must be in the form of 4-bit buses. Also, each of the blocks must be suitably connected to control lines so that its function may be defined for any of a range of possible operations.

During the design process, and in particular when defining the interconnection strategy and designing the stick diagrams, care must be taken in allocating the layers to the various data or control paths. Points to be noted:

- Metal can cross poly or diffusion
- Poly crossing diffusion form a transistor
- Whenever lines touch on the same level an interconnection is formed
- Simple contacts can be used to join diffusion or poly to metal.
- Buried contacts or a butting contacts can be used to join diffusion and poly
- Some processes use 2\textsuperscript{nd} metal
- 1\textsuperscript{st} and 2\textsuperscript{nd} metal layers may be joined using a via
- Each layer has particular electrical properties which must be taken into account
- For CMOS layouts, p-and n-diffusion wires must not directly join each other
- Nor may they cross either a p-well or an n-well boundary

**Design of a 4-bit shifter**

Any general purpose n-bit shifter should be able to shift incoming data by up to n – 1 place in a right-shift or left-shift direction. Further specifying that all shifts should be on an end-around basis, so that any bit shifted out at one end of a data word will be shifted in at the other end of the word, then the problem of right shift or left shift is greatly eased. It can be analyzed that for a 4-bit word, that a 1-bit shift right is equivalent to a 3-bit shift left and a 2-bit shift right is equivalent to a 2-bit left etc. Hence, the design of either shift right or left can be done. Here the design is of shift right by 0, 1, 2, or 3 places. The shifter must have:

- input from a four line parallel data bus
- four output lines for the shifted data
- means of transferring input data to output lines with any shift from 0 to 3 bits

Consider a direct MOS switch implementation of a 4 X 4 crossbar switches shown in figure 6.8. The arrangement is general and may be expanded to accommodate n-bit inputs/outputs. In this arrangement any input can be connected to any or all the outputs. Furthermore, 16 control signals (sw\textsubscript{00} – sw\textsubscript{15}), one for each transistor switch, must be provided to drive the crossbar switch, and such complexity is highly undesirable.
An adaptation of this arrangement recognizes the fact that we couple the switch gates together in groups of four and also form four separate groups corresponding to shifts of zero, one, two and three bits. The resulting arrangement is known as a barrel shifter and a 4 X 4 barrel shifter circuit diagram is as shown in the figure 6.9.

The interbus switches have their gate inputs connected in a staircase fashion in groups of four and there are now four shift control inputs which must be mutually exclusive in the active state. CMOS transmission gates may be used in place of the simple pass transistor switches if appropriate. Barrel shifter connects the input lines representing a word to a group of output lines with the required shift determined by its control inputs (sh0, sh1, sh2, sh3). Control inputs also determine the direction of the shift. If input word has n – bits and shifts from 0 to n-1 bit positions are to be implemented.

To summaries the design steps

- Set out the specifications
- Partition the architecture into subsystems
- Set a tentative floor plan
- Determine the interconnects
- Choose layers for the bus & control lines
- Conceive a regular architecture
- Develop stick diagram
Produce mask layouts for standard cell
Cascade & replicate standard cells as required to complete the design

**Design of an ALU subsystem**

Having designed the shifter, we shall design another subsystem of the 4-bit data path. An appropriate choice is ALU as shown in the figure 6.10 below.

![Figure 6.10: 4-bit data path for processor](image)

The heart of the ALU is a 4-bit adder circuit. A 4-bit adder must take sum of two 4-bit numbers, and there is an assumption that all 4-bit quantities are presented in parallel form and that the shifter circuit is designed to accept and shift a 4-bit parallel sum from the ALU. The sum is to be stored in parallel at the output of the adder from where it is fed through the shifter and back to the register array. Therefore, a single 4-bit data bus is needed from the adder to the shifter and another 4-bit bus is required from the shifted output back to the register array. Hence, for an adder two 4-bit parallel numbers are fed on two 4-bit buses. The clock signal is also required to the adder, during which the inputs are given and sum is generated. The shifter is unclocked but must be connected to four shift control lines.

**Design of a 4-bit adder:**

The truth table of binary adder is as shown in table 6.1

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A_k)</td>
<td>(B_k)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
As seen from the table any column k there will be three inputs namely A_k, B_k as present input number and C_{k-1} as the previous carry. It can also be seen that there are two outputs sum S_k and carry C_k.

From the table one form of the equation is:
\[
\text{Sum } S_k = H_k C_{k-1}' + H_k' C_{k-1}
\]
\[
\text{New carry } C_k = A_k B_k + H_k C_{k-1}
\]

Where
\[
\text{Half sum } H_k = A_k' B_k + A_k B_k'
\]

**Adder element requirements**

Table 6.1 reveals that the adder requirement may be stated as:
- If \( A_k = B_k \) then \( S_k = C_{k-1} \)
- Else \( S_k = C_{k-1}' \)

And for the carry \( C_k \)
- If \( A_k = B_k \) then \( C_k = A_k = B_k \)
- Else \( C_k = C_{k-1} \)

Thus the standard adder element for 1-bit is as shown in the figure 6.11.

![Adder element](image)

**Implementing ALU functions with an adder:**

An ALU must be able to add and subtract two binary numbers, perform logical operations such as And, Or and Equality (Ex-or) functions. Subtraction can be performed by taking 2’s complement of the negative number and perform the further addition. It is desirable to keep the architecture as simple as possible, and also see that the adder performs the logical operations also. Hence let us examine the possibility.

The adder equations are:
\[
\text{Sum } S_k = H_k C_{k-1}' + H_k' C_{k-1}
\]
\[
\text{New carry } C_k = A_k B_k + H_k C_{k-1}
\]

Where
\[
\text{Half sum } H_k = A_k' B_k + A_k B_k'
\]

Let us consider the sum output, if the previous carry is at logical 0, then
\[
S_k = H_k \cdot 1 + H_k'. 0
\]
\[
S_k = H_k = A_k' B_k + A_k B_k' \quad \text{An Ex-or operation}
\]

Now, if \( C_{k-1} \) is logically 1, then
\[
S_k = H_k \cdot 0 + H_k'. 1
\]
$S_k = H_k' – \text{An Ex-Nor operation}$

Next, consider the carry output of each element, first $C_{k-1}$ is held at logical 0, then
$C_k = A_k B_k + H_k \cdot 0$
$C_k = A_k B_k – \text{An And operation}$

Now if $C_{k-1}$ is at logical 1, then
$C_k = A_k B_k + H_k \cdot 1$

On solving $C_k = A_k + B_k – \text{An Or operation}$

The adder element implementing both the arithmetic and logical functions can be implemented as shown in the figure 6.12.

Figure 6.12: 1-bit adder element

The above can be cascaded to form 4-bit ALU.

**A further consideration of adders**

**Generation:**

This principle of generation allows the system to take advantage of the occurrences “$a_k=b_k$”. In both cases ($a_k=1$ or $a_k=0$) the carry bit will be known.

**Propagation:**

If we are able to localize a chain of bits $a_k a_{k+1}...a_{k+p}$ and $b_k b_{k+1}...b_{k+p}$ for which $a_k$ not equal to $b_k$ for $k$ in $[k,k+p]$, then the output carry bit of this chain will be equal to the input carry bit of the chain.

These remarks constitute the principle of generation and propagation used to speed the addition of two numbers.

All adders which use this principle calculate in a first stage.

$$p_k = a_k \text{ XOR } b_k$$
$$g_k = a_k \text{ b_k}$$
Manchester carry – chain

This implementation can be very performant (20 transistors) depending on the way the XOR function is built. The carry propagation of the carry is controlled by the output of the XOR gate. The generation of the carry is directly made by the function at the bottom. When both input signals are 1, then the inverse output carry is 0.

Figure-6.12: An adder with propagation signal controlling the pass-gate

In the schematic of Figure 6.12, the carry passes through a complete transmission gate. If the carry path is precharged to VDD, the transmission gate is then reduced to a simple NMOS transistor. In the same way the PMOS transistors of the carry generation is removed. One gets a Manchester cell.
The Manchester cell is very fast, but a large set of such cascaded cells would be slow. This is due to the distributed RC effect and the body effect making the propagation time grow with the square of the number of cells. Practically, an inverter is added every four cells, like in Figure 6.14.

Adder Enhancement techniques

The operands of addition are the addend and the augend. The addend is added to the augend to form the sum. In most computers, the augmented operand (the augend) is replaced by the sum, whereas the addend is unchanged. High speed adders are not only for addition but also for subtraction, multiplication and division. The speed of a digital processor depends heavily on the speed of adders. The adders add vectors of bits and the principal problem is to speed-up the carry signal. A traditional and non optimized four bit adder can be made by the use of the generic one-bit adder cell connected one to the other. It is the ripple carry adder. In this case, the sum resulting at each stage need to wait for the incoming carry signal to perform the sum operation. The carry propagation can be speed-up in two ways. The first—and most obvious—way is to use a faster logic circuit technology. The second way is to generate carries by means of forecasting logic that does not rely on the carry signal being rippled from stage to stage of the adder.
The Carry-Skip Adder

Depending on the position at which a carry signal has been generated, the propagation time can be variable. In the best case, when there is no carry generation, the addition time will only take into account the time to propagate the carry signal. Figure 6.15 is an example illustrating a carry signal generated twice, with the input carry being equal to 0. In this case three simultaneous carry propagations occur. The longest is the second, which takes 7 cell delays (it starts at the 4th position and ends at the 11th position). So the addition time of these two numbers with this 16-bits Ripple Carry Adder is $7k + k'$, where $k$ is the delay cell and $k'$ is the time needed to compute the 11th sum bit using the 11th carry-in.

With a Ripple Carry Adder, if the input bits $A_i$ and $B_i$ are different for all position $i$, then the carry signal is propagated at all positions (thus never generated), and the addition is completed when the carry signal has propagated through the whole adder. In this case, the Ripple Carry Adder is as slow as it is large. Actually, Ripple Carry Adders are fast only for some configurations of the input words, where carry signals are generated at some positions.

Carry Skip Adders take advantage both of the generation or the propagation of the carry signal. They are divided into blocks, where a special circuit detects quickly if all the bits to be added are different ($P_i = 1$ in all the block). The signal produced by this circuit will be called block propagation signal. If the carry is propagated at all positions in the block, then the carry signal entering into the block can directly bypass it and so be transmitted through a multiplexer to the next block. As soon as the carry signal is transmitted to a block, it starts to propagate through the block, as if it had been generated at the beginning of the block. Figure 6.16 shows the structure of a 24-bits Carry Skip Adder, divided into 4 blocks.

![Figure 6.15: Example of Carry skip adder](image)
Optimization of the carry skip adder

It becomes now obvious that there exist a trade-off between the speed and the size of the blocks. In this part we analyze the division of the adder into blocks of equal size. Let us denote $k_1$ the time needed by the carry signal to propagate through an adder cell, and $k_2$ the time it needs to skip over one block. Suppose the $N$-bit Carry Skip Adder is divided into $M$ blocks, and each block contains $P$ adder cells. The actual addition time of a Ripple Carry Adder depends on the configuration of the input words. The completion time may be small but it also may reach the worst case, when all adder cells propagate the carry signal. In the same way, we must evaluate the worst carry propagation time for the Carry Skip Adder. The worst case of carry propagation is depicted in Figure 6.17.

The configuration of the input words is such that a carry signal is generated at the beginning of the first block. Then this carry signal is propagated by all the succeeding adder cells but the last which generates another carry signal. In the first and the last block the block propagation signal is equal to 0, so the entering carry signal is not transmitted to the next block. Consequently, in the first block, the last adder cells must wait for the carry signal, which comes from the first cell of the first block. When going out of the first
block, the carry signal is distributed to the 2\textsuperscript{nd}, 3\textsuperscript{rd} and last block, where it propagates. In these blocks, the carry signals propagate almost simultaneously (we must account for the multiplexer delays). Any other situation leads to a better case. Suppose for instance that the 2\textsuperscript{nd} block does not propagate the carry signal (its block propagation signal is equal to zero), then it means that a carry signal is generated inside. This carry signal starts to propagate as soon as the input bits are settled. In other words, at the beginning of the addition, there exist two sources for the carry signals. The paths of these carry signals are shorter than the carry path of the worst case. Let us formalize that the total adder is made of N adder cells. It contains M blocks of P adder cells. The total of adder cells is then

\[ N = M \cdot P \]

The time \( T \) needed by the carry signal to propagate through \( P \) adder cells is

\[ T = k_1 \cdot P \]

The time \( T' \) needed by the carry signal to skip through \( M \) adder blocks is

\[ T' = k_2 \cdot M \]

The problem to solve is to minimize the worst case delay which is:

\[
T_{\text{worst case}} = 2 \cdot \frac{P}{M} \cdot k_1 + (M - 2) \cdot k_2
\]

\[
T_{\text{worst case}} = 2 \cdot \frac{N}{M} \cdot k_1 + (M - 2) \cdot k_2
\]

The Carry-Select Adder

This type of adder is not as fast as the Carry Look Ahead (CLA) presented in a next section. However, despite its bigger amount of hardware needed, it has an interesting design concept. The Carry Select principle requires two identical parallel adders that are partitioned into four-bit groups. Each group consists of the same design as that shown on Figure 6.18. The group generates a group carry. In the carry select adder, two sums are generated simultaneously. One sum assumes that the carry in is equal to one as the other assumes that the carry in is equal to zero. So that the predicted group carry is used to select one of the two sums.

It can be seen that the group carries logic increases rapidly when more high-order groups are added to the total adder length. This complexity can be decreased, with a subsequent increase in the delay, by partitioning a long adder into sections, with four groups per section, similar to the CLA adder.
Prof. Roopa Kulkarni, GIT, Belgaum.

**Optimization of the carry select adder**

- Computational time
  \[ T = K_1n \]

- Dividing the adder into blocks with 2 parallel paths
  \[ T = K_1n/2 + K_2 \]

- For a n-bit adder of M-blocks and each block contains P adder cells in series
  \[ T = PK_1 + (M - 1) K_2 \]

  \( n = M \cdot P \) minimum value for \( T \) is when \( M = \sqrt{\frac{K_1n}{K_1}} \)

**The Carry Look-Ahead Adder**

The limitation in the sequential method of forming carries, especially in the Ripple Carry adder arises from specifying \( c_i \) as a specific function of \( c_{i-1} \). It is possible to express a carry as a function of all the preceding low order carry by using the recursivity of the carry function. With the following expression a considerable increase in speed can be realized.

\[
C_i = G_i + G_{i+2}P_{i-1} + G_{i+3}P_{i-1} + \ldots + G_0P_1P_2\ldots P_{i-1} + C_0P_0P_1P_2\ldots P_{i-1}
\]

Usually the size and complexity for a big adder using this equation is not affordable. That is why the equation is used in a modular way by making groups of carry (usually four bits). Such a unit generates then a group carry which give the right predicted information to the next block giving time to the sum units to perform their calculation.

\[
\pi = P_0P_1P_2P_3
\]

\[
\gamma = g_3 + P_3g_2 + P_3P_2g_1 + P_3P_2P_1g_0
\]

\[
c_A = \gamma + \pi \cdot c_0
\]

Figure-6.19: The Carry Generation unit performing the Carry group computation
Such unit can be implemented in various ways, according to the allowed level of abstraction. In a CMOS process, 17 transistors are able to guarantee the static function (Figure 6.20). However this design requires a careful sizing of the transistors put in series.

The same design is available with less transistors in a dynamic logic design. The sizing is still an important issue, but the number of transistors is reduced (Figure 6.21).

![Figure 6.20: Static implementation of the 4-bit carry lookahead chain](image)

![Figure 6.21: Dynamic implementation of the 4-bit carry lookahead chain](image)

Figure 6.22 shows the implementation of 16-bit CLA adder.
Multipliers

Introduction

Multiplication can be considered as a series of repeated additions. The number to be added is the multiplicand, the number of times that it is added is the multiplier, and the result is the product. Each step of the addition generates a partial product. In most computers, the operands usually contain the same number of bits. When the operands are interpreted as integers, the product is generally twice the length of the operands in order to preserve the information content. This repeated addition method that is suggested by the arithmetic definition is slow that it is almost always replaced by an algorithm that makes use of positional number representation.

It is possible to decompose multipliers in two parts. The first part is dedicated to the generation of partial products, and the second one collects and adds them. As for adders, it is possible to enhance the intrinsic performances of multipliers. Acting in the generation part, the Booth (or modified Booth) algorithm is often used because it reduces the number of partial products. The collection of the partial products can then be made using a regular array, a Wallace tree or a binary tree.

Serial-Parallel Multiplier

This multiplier is the simplest one, the multiplication is considered as a succession of additions.

if \( A = (a_n a_{n-1} \ldots a_0) \) and \( B = (b_n b_{n-1} \ldots b_0) \)

The product \( A.B \) is expressed as:

\[
A.B = A.2^n \cdot b_n + A.2^{n-1} \cdot b_{n-1} + \ldots + A.2^0 \cdot b_0
\]

The structure of Figure 6.23 is suited only for positive operands. If the operands are negative and coded in 2’s complement:

1. The most significant bit of \( B \) has a negative weight, so a subtraction has to be performed at the last step.
2. Operand $A.2^k$ must be written on $2N$ bits, so the most significant bit of $A$ must be duplicated. It may be easier to shift the content of the accumulator to the right instead of shifting $A$ to the left.

![Figure-6.23: Serial-Parallel multiplier](image)

**Braun Parallel Multiplier**

The simplest parallel multiplier is the Braun array. All the partial products $A.bk$ are computed in parallel, and then collected through a cascade of Carry Save Adders. At the bottom of the array, the output of the array is noted in Carry Save, so an additional adder converts it (by the mean of carry propagation) into the classical notation (Figure 6.24). The completion time is limited by the depth of the carry save array, and by the carry propagation in the adder. Note that this multiplier is only suited for positive operands. Negative operands may be multiplied using a Baugh-Wooley multiplier.

![Figure 6.24: A 4-bit Braun Array](image)
Baugh-Wooley Multiplier

This technique has been developed in order to design regular multipliers, suited for 2’s-complement numbers.

Let us consider 2 numbers A and B:

\[ A = (a_{n-1} \ldots a_0) = -a_{n-1} \cdot 2^{n-1} + \sum_{0}^{n-2} a_i \cdot 2^i \]

\[ B = (b_{n-1} \ldots b_0) = -b_{n-1} \cdot 2^{n-1} + \sum_{0}^{n-2} b_i \cdot 2^i \]

The product A.B is given by the following equation:

\[ A \cdot B = a_{n-1} \cdot b_{n-1} \cdot 2^{2n-2} + \sum_{0}^{n-2} \sum_{0}^{n-2} a_i \cdot b_i \cdot 2^{i+j} - a_{n-1} \sum_{0}^{n-2} b_i \cdot 2^{n+i-1} - b_{n-1} \sum_{0}^{n-2} a_i \cdot 2^{n+i-1} \]

We see that subtraction cells must be used. In order to use only adder cells, the negative terms may be rewritten as:

\[ -a_{n-1} \sum_{0}^{n-2} b_i \cdot 2^{i+n-1} = a_{n-1} \cdot \left( -2^{2n-2} + 2^{n-1} \sum_{0}^{n-2} b_i \cdot 2^{i+n-1} \right) \]

By this way, A.B becomes:

\[ A \cdot B = a_{n-1} \cdot b_{n-1} \cdot 2^{2n-2} + \sum_{0}^{n-2} \sum_{0}^{n-2} a_i \cdot b_j \cdot 2^{i+j} + b_{n-1} \left[ -2^{2n-2} + 2^{n-1} \sum_{0}^{n-2} a_i \cdot 2^{i+n-1} \right] + a_{n-1} \left[ -2^{2n-2} + 2^{n-1} + \sum_{0}^{n-2} b_i \cdot 2^{i+n-1} \right] \]
The final equation is:

\[ A \cdot B = -2^{2n-1} + (a_{n-1} + \overline{b_{n-1}} + a_{n-1} \cdot b_{n-1}) \cdot 2^{2n-2} \]

\[ + \sum_{i=0}^{n-2} \sum_{j=0}^{n-2} a_i b_j \cdot 2^{i+j} + (a_{n-1} + b_{n-1}) \cdot 2^n \]

\[ + \sum_{i=0}^{n-2} b_{n-1} \cdot x_i \cdot 2^{i+n-1} + \sum_{i=0}^{n-2} a_{n-1} \cdot b_i \cdot 2^{i+n-1} \]

because:

\[ - (b_{n-1} + a_{n-1}) \cdot 2^{2n-2} = -2^{2n-1} + (a_{n-1} + \overline{b_{n-1}}) \cdot 2^{2n-2} \]

A and B are n-bits operands, so their product is a 2n-bits number. Consequently, the most significant weight is 2n-1, and the first term \(-2^{2n-1}\) is taken into account by adding a 1 in the most significant cell of the multiplier. The implementation is shown in figure 6.25.

![Figure-6.25: A 4-bit Baugh-Wooley Multiplier](image)

**Booth Algorithm**

This algorithm is a powerful direct algorithm for signed-number multiplication. It generates a 2n-bit product and treats both positive and negative numbers uniformly. The idea is to reduce the number of additions to perform. Booth algorithm allows in the best case n/2 additions whereas modified Booth algorithm allows always n/2 additions.
Let us consider a string of \( k \) consecutive 1s in a multiplier:
\[
..., i+k, i+k-1, i+k-2, ..., i, \ i-1, ...
..., 0, \ 1, \ 1, ..., \ 1, \ 0, ...
\]
where there is \( k \) consecutive 1s.

By using the following property of binary strings:
\[
2^{i+k} - 2^i = 2^{i+k-1} + 2^{i+k-2} + ... + 2^{i+1} + 2^i
\]
the \( k \) consecutive 1s can be replaced by the following string
\[
..., i+k+1, \ i+k, i+k-1, i+k-2, ..., i+1, \ i, \ i-1, ...
..., 0, \ 1, \ 0, \ 0, ..., \ 0, \ -1, \ 0, ...
\]
\( k-1 \) consecutive 0s Addition Subtraction

In fact, the modified Booth algorithm converts a signed number from the standard
2’s-complement radix into a number system where the digits are in the set \( \{-1,0,1\} \). In
this number system, any number may be written in several forms, so the system is called
redundant.

The coding table for the modified Booth algorithm is given in Table 1. The
algorithm scans strings composed of three digits. Depending on the value of the string, a
certain operation will be performed.

A possible implementation of the Booth encoder is given on Figure 6.26.

Table-1: Modified Booth coding table

<table>
<thead>
<tr>
<th>BIT</th>
<th>OPERATION</th>
<th>M is multiplied by</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Y_{i+1} )</td>
<td>( Y_i )</td>
<td>( Y_{i-1} )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
To summarize the operation:

- Grouping multiplier bits into pairs
  - Orthogonal idea to the Booth recoding
  - Reduces the num of partial products to half
  - If Booth recoding not used have to be able to multiply by 3 (hard: shift+add)

- Applying the grouping idea to Booth
  - Modified Booth Recoding (Encoding)
    - We already got rid of sequences of 1’s no multiplication by 3
    - Just negate, shift once or twice

Wallace Trees

For this purpose, Wallace trees were introduced. The addition time grows like the logarithm of the bit number. The simplest Wallace tree is the adder cell. More generally, an n-inputs Wallace tree is an n-input operator and \( \log_2(n) \) outputs, such that the value of the output word is equal to the number of “1” in the input word. The input bits and the least significant bit of the output have the same weight (Figure 6.27). An important property of Wallace trees is that they may be constructed using adder cells. Furthermore, the number of adder cells needed grows like the logarithm \( \log_2(n) \) of the number n of input bits. Consequently, Wallace trees are useful whenever a large number of operands are to add, like in multipliers. In a Braun or Baugh-Wooley multiplier with a Ripple Carry Adder, the completion time of the multiplication is proportional to twice the number n of bits. If the collection of the partial products is made through Wallace trees, the time for getting the result in a carry save notation should be proportional to \( \log_2(n) \).

Figure-6.27: Wallace cells made of adders
Figure 6.28 represents a 7-inputs adder: for each weight, Wallace trees are used until there remain only two bits of each weight, as to add them using a classical 2-inputs adder. When taking into account the regularity of the interconnections, Wallace trees are the most irregular.

Figure-6.28: A 7-inputs Wallace tree

To summarize the operation:

The Wallace tree has three steps:

- Multiply (that is - AND) each bit of one of the arguments, by each bit of the other, yielding $n^2$ results.
- Reduce the number of partial products to two by layers of full and half adders.
- Group the wires in two numbers, and add them with a conventional adder.

The second phase works as follows.

- Take any three wires with the same weights and input them into a full adder.
- The result will be an output wire of the same weight and an output wire with a higher weight for each three input wires.
- If there are two wires of the same weight left, input them into a half adder.
- If there is just one wire left, connect it to the next layer.
Unit 7: Memory

Objectives: At the end of this unit we will be able to understand
- System timing consideration
- Storage / Memory Elements
  - dynamic shift register
  - 1T and 3T dynamic memory
  - 4T dynamic and 6T static CMOS memory
- Array of memory cells

System timing considerations:
- Two phase non-overlapping clock
- \( \phi_1 \) leads \( \phi_2 \)
- Bits to be stored are written to register and subsystems on \( \phi_1 \)
- Bits or data written are assumed to be settled before \( \phi_2 \)
- \( \phi_2 \) signal used to refresh data
- Delays assumed to be less than the intervals between the leading edge of \( \phi_1 \) & \( \phi_2 \)
- Bits or data may be read on the next \( \phi_1 \)
- There must be at least one clocked storage element in series with every closed loop signal path

Storage / Memory Elements:
The elements that we will be studying are:
- Dynamic shift register
- 3T dynamic RAM cell
- 1T dynamic memory cell
- Pseudo static RAM / register cell
- 4T dynamic & 6T static memory cell
- JK FF circuit
- D FF circuit

Dynamic shift register:

Circuit diagram: Refer to unit 4(ch 6.5.4)

Power dissipation
- static dissipation is very small
- dynamic power is significant
- dissipation can be reduced by alternate geometry

Volatility
- data storage time is limited to 1msec or less
3T dynamic RAM cell:

Circuit diagram

![3T Dynamic RAM Cell](image)

**Working**
- RD = low, bit read from bus through T1, WR = high, logic level on bus sent to Cg of T2, WR = low again
- Bit level is stored in Cg of T2, RD=WR=low
- Stored bit is read by RD = high, bus will be pulled to ground if a 1 was stored else 0 if T2 non-conducting, bus will remain high.

**Dissipation**
- Static dissipation is nil
- Depends on bus pull-up & on duration of RD signal & switching frequency

**Volatility**
- Cell is dynamic, data will be there as long as charge remains on Cg of T2

1T dynamic memory cell:

Circuit diagram

![1T Dynamic RAM Cell](image)
Working

- Row select (RS) = high, during write from R/W line Cm is charged
- data is read from Cm by detecting the charge on Cm with RS = high
- cell arrangement is bit complex.
- solution: extend the diffusion area comprising source of pass transistor, but Cd<<Cgchannel
- another solution: create significant capacitor using poly plate over diffusion area.
- Cm is formed as a 3-plate structure
- with all this careful design is necessary to achieve consistent readability

Dissipation

- no static power, but there must be an allowance for switching energy during read/write

Pseudo static RAM / register cell:

Circuit diagram

Figure 7.3: nMOS pseudo-static memory Cell

Figure 7.4: CMOS pseudo-static memory Cell
Working

- dynamic RAM need to be refreshed periodically and hence not convenient
- static RAM needs to be designed to hold data indefinitely
- One way is connect 2 inverter stages with a feedback.
- say $\phi_2$ to refresh the data every clock cycle
- bit is written on activating the WR line which occurs with $\phi_1$ of the clock
- bit on $C_g$ of inverter 1 will produce complemented output at inverter 1 and true at output of inverter 2
- at every $\phi_2$, stored bit is refreshed through the gated feedback path
- stored bit is held till $\phi_2$ of clock occurs at time less than the decay time of stored bit
- to read RD along with $\phi_1$ is activated

Note:

- WR and RD must be mutually exclusive
- $\phi_2$ is used for refreshing, hence no data to be read, if so charge sharing effect, leading to destruction of stored bit
- cells must be stackable, both side-by-side & top to bottom
- allow for other bus lines to run through the cell

4T dynamic & 6T static memory cell:

Circuit diagram

![Circuit Diagram](image)

Figure 7.4: Dynamic and static memory cells
Working

• uses 2 buses per bit to store bit and bit’
• both buses are precharged to logic 1 before read or write operation.
• write operation
• read operation

Write operation

• both bit & bit’ buses are precharged to VDD with clock $\phi_1$ via transistor T5 & T6
• column select line is activated along with $\phi_2$
• either bit or bit’ line is discharged along the I/O line when carrying a logic 0
• row & column select signals are activated at the same time => bit line states are written in via T3 & T4, stored by T1 & T2 as charge

Read operation

• bit and bit’ lines are again precharged to VDD via T5 & T6 during $\phi_1$
• if 1 has been stored, T2 ON & T1 OFF
• bit’ line will be discharged to VSS via T2
• each cell of RAM array be of minimum size & hence will be the transistors
• implies incapable of sinking large charges quickly
• RAM arrays usually employ some form of sense amplifier
  • T1, T2, T3 & T4 form as flip-flop circuit
  • if sense line to be inactive, state of the bit line reflects the charge present on gate capacitance of T1 & T3
  • current flowing from VDD through an on transistor helps to maintain the state of bit lines
Unit 8: Testability

Objective: At the end of this unit we will be able to understand
- Design for testability (DFT)
- DFT methods for digital circuits:
  - Ad-hoc methods
  - Structured methods:
    - Scan
    - Level Sensitive Scan Design
    - Boundary scan
    - Other Scan Techniques

Definition:
Design for testability (DFT) refers to those design techniques that make test generation and test application cost-effective.

Some terminologies:
Input / output (I/O) pads
- Protection of circuitry on chip from damage
- Care to be taken in handling all MOS circuits
- Provide necessary buffering between the environments On & OFF chip
- Provide for the connections of power supply
- Pads must be always placed around the peripheral

Minimum set of pads include:
- VDD connection pad
- GND(VSS) connection pad
- Input pad
- Output pad
- Bidirectional I/O pad

Designer must be aware of:
- nature of circuitry
- ratio/size of inverters/buffers on which output lines are connected
- how input lines pass through the pad circuit (pass transistor/transmission gate)

System delays
Buses:
- convenient concept in distributing data & control through a system
- bidirectional buses are convenient
- in design of datapath
- problems: capacitive load present
• largest capacitance
• sufficient time must be allowed to charge the total bus
• clock $\phi_1$ & $\phi_2$

Control paths, selectors & decoders
1. select registers and open pass transistors to connect cells to bus
2. Data propagation delay bus
3. Carry chain delay

Faults and Fault Modeling

A fault model is a model of how a physical or parametric fault manifests itself in the circuit Operation. Fault tests are derived based on these models

Physical Faults are caused due to the following reasons:

- Defect in silicon substrate
- Photolithographic defects
- Mask contamination and scratches
- Process variations and abnormalities
- Oxide defects

Physical faults cause Electrical and Logical faults

Logical Faults are:

- Single/multiple stuck-at (*most used*)
- CMOS stuck-open
- CMOS stuck-on
- AND / OR Bridging faults

Electrical faults are due to short, opens, transistor stuck on, stuck open, excessive steady state currents, resistive shorts and open.

Design for Testability

Two key concepts
- Observability
- Controllability

DFT often is associated with design modifications that provide improved access to internal circuit elements such that the local internal state can be controlled (controllability) and/or observed (observability) more easily. The design modifications can be strictly physical in nature (e.g., adding a physical probe point to a net) and/or add active circuit elements to facilitate controllability/observability (e.g., inserting a multiplexer into a net). While controllability and observability improvements for internal circuit elements definitely are important for test, they are not the only type of DFT
What can we do to increase testability?

- increase observability
  - add more pins (?!)
  - add small "probe" bus, selectively enable different values onto bus
  - use a hash function to "compress" a sequence of values (e.g., the values of a bus over many clock cycles) into a small number of bits for later read-out
  - cheap read-out of all state information

- increase controllability
  - use muxes to isolate submodules and select sources of test data as inputs
  - provide easy setup of internal state

Testing combinational logic

The solution to the problem of testing a purely combinational logic block is a good set of patterns detecting "all" the possible faults.

The first idea to test an N input circuit would be to apply an N-bit counter to the inputs (controllability), then generate all the 2N combinations, and observe the outputs for checking (observability). This is called "exhaustive testing", and it is very efficient... but only for few-input circuits. When the input number increases, this technique becomes very time consuming.

Sensitized Path Testing

Most of the time, in exhaustive testing, many patterns do not occur during the application of the circuit. So instead of spending a huge amount of time searching for faults everywhere, the possible faults are first enumerated and a set of appropriate vectors are then generated. This is called "single-path sensitization" and it is based on "fault oriented testing".
The basic idea is to select a path from the site of a fault, through a sequence of gates leading to an output of the combinational logic under test. The process is composed of three steps:

- **Manifestation**: gate inputs, at the site of the fault, are specified as to generate the opposite value of the faulty value (0 for SA1, 1 for SA0).
- **Propagation**: inputs of the other gates are determined so as to propagate the fault signal along the specified path to the primary output of the circuit. This is done by setting these inputs to "1" for AND/NAND gates and "0" for OR/NOR gates.
- **Consistency**: or justification. This final step helps finding the primary input pattern that will realize all the necessary input values. This is done by tracing backward from the gate inputs to the primary inputs of the logic in order to receive the test patterns.

**Example 1** - SA1 of line 1 (L1): the aim is to find the vector(s) able to detect this fault.

**Manifestation**: L1 = 0, then input A = 0. In a fault-free situation, the output F changes with A if B, C and D are fixed: for B, C and D fixed, L1 is SA1 gives F = 0, for instance, even if A = 0 (F = 1 for fault-free).

**Propagation**: Through the AND-gate: L5 = L8 = 1, this condition is necessary for the propagation of the "L1 = 0". This leads to L10 = 0. Through the NOR-gate, and since L10 = 0, then L11 = 0, so the propagated manifestation can reach the primary output F. F is then read and compared with the fault-free value: F = 1.
• **Consistency:** From the AND-gate: \(L5 = 1\), and then \(L2 = B = 1\). Also \(L8 = 1\), and then \(L7 = 1\). Until now we found the values of \(A\) and \(B\). When \(C\) and \(D\) are found, then the test vectors are generated, in the same manner, and ready to be applied to detect \(L1 = SA1\). From the NOT-gate, \(L11 = 0\), so \(L9 = L7 = 1\) (coherency with \(L8 = L7\)). From the OR-gate \(L7 = 1\), and since \(L6 = L2 = B = 1\), so \(B + C + D = L7 = 1\), then \(C\) and \(D\) can have either 1 or 0.

These three steps have led to four possible vectors detecting \(L1 = SA1\).

**Example 2** - \(SA1\) of line8 (\(L8\)) : The same combinational logic having one internal line \(SA1\)

**Manifestation** : \(L8 = 0\)

**Propagation:** Through the AND-gate: \(L5 = L1 = 1\), then \(L10 = 0\) Through the NOR-gate: we want to have \(L11 = 0\), not to mask \(L10 = 0\).

**Consistency:** From the AND-gate \(L8 = 0\) leads to \(L7 = 0\). From the NOT-gate \(L11 = 0\) means \(L9 = L7 = 1\), \(L7\) could not be set to 1 and 0 at the same time. This incompatibility could not be resolved in this case, and the fault "\(L8 SA1\)" remains undetectable.

**D – Algorithm:**

Given a circuit comprising combinational logic, the algorithm aims to find an assignment of input values that will allow detection of a particular internal fault by examining the output conditions. Using this algorithm the system can either be said as good or faulty. The existence of a fault in the faulty machine will cause a discrepancy between its behavior and that of the good machine for some particular values of inputs. The D-algorithm provides a systematic means of assigning input values for that particular design so that the discrepancy is driven to an output where it may be observed and thus detected. The algorithm is time-intensive and computing intensive for large circuits.
Practical design for test guidelines

Practical guidelines for testability should aim to facilitate test processes in three main ways:

- facilitate test generation
- facilitate test application
- avoid timing problems

These matters are discussed as below:

**Improve Controllability and Observability**

All "design for test" methods ensure that a design has enough observability and controllability to provide for a complete and efficient testing. When a node has difficult access from primary inputs or outputs (pads of the circuit), a very efficient method is to add internal pads accessing this kind of node in order, for instance, to control block B2 and observe block B1 with a probe.

![Optimize Controllability and Observability](image)

**Figure 8.1 Improve Controllability and Observability**

It is easy to observe block B1 by adding a pad just on its output, without breaking the link between the two blocks. The control of the block B2 means to set a 0 or a 1 to its input, and also to be transparent to the link B1-B2. The logic functions of this purpose are a NOR-gate, transparent to a zero, and a NAND-gate, transparent to a one. By this way the control of B2 is possible across these two gates.

Another implementation of this cell is based on pass-gates multiplexers performing the same function, but with less transistors than with the NAND and NOR gates (8 instead of 12).

The simple optimization of observation and control is not enough to guarantee a full testability of the blocks B1 and B2. This technique has to be completed with some other techniques of testing depending on the internal structures of blocks B1 and B2.
Use Multiplexers

This technique is an extension of the precedent, while multiplexers are used in case of limitation of primary inputs and outputs.

In this case the major penalties are extra devices and propagation delays due to multiplexers. Demultiplexers are also used to improve observability. Using multiplexers and demultiplexers allows internal access of blocks separately from each other, which is the basis of techniques based on partitioning or bypassing blocks to observe or control separately other blocks.

![Use Multiplexers](image1)

Figure 8.2: Use multiplexers

Partition Large Circuits

Partitioning large circuits into smaller sub-circuits reduces the test-generation effort. The test-generation effort for a general purpose circuit of n gates is assumed to be proportional to somewhere between n^2 and n^3. If the circuit is partitioned into two sub-circuits, then the amount of test generation effort is reduced correspondingly.

![Partition Large Circuits](image2)

Figure 8.3: Partition Large Circuits

Logical partitioning of a circuit should be based on recognizable sub-functions and can be achieved physically by incorporating some facilities to isolate and control
clock lines, reset lines and power supply lines. The multiplexers can be massively used to separate sub-circuits without changing the function of the global circuit.

**Divide Long Counter Chains**

Based on the same principle of partitioning, the counters are sequential elements that need a large number of vectors to be fully tested. The partitioning of a long counter corresponds to its division into sub-counters.

The full test of a 16-bit counter requires the application of \(2^{16} + 1 = 65537\) clock pulses. If this counter is divided into two 8-bit counters, then each counter can be tested separately, and the total test time is reduced 128 times \((2^7)\). This is also useful if there are subsequent requirements to set the counter to a particular count for tests associated with other parts of the circuit: pre-loading facilities.

![Figure 8.4: Divide Long Counter Chains](image)

**Initialize Sequential Logic**

One of the most important problems in sequential logic testing occurs at the time of power-on, where the first state is random if there were no initialization. In this case it is impossible to start a test sequence correctly, because of memory effects of the sequential elements.

![Initialize Sequential Logic](image)
The solution is to provide flip-flops or latches with a set or reset input, and then to use them so that the test sequence would start with a known state.

Ideally, all memory elements should be able to be set to a known state, but practically this could be very surface consuming, also it is not always necessary to initialize all the sequential logic. For example, a serial-in serial-out counter could have its first flip-flop provided with an initialization, then after a few clock pulses the counter is in a known state.

Overriding of the tester is necessary some times, and requires the addition of gates before a Set or a Reset so the tester can override the initialization state of the logic.

Avoid Asynchronous Logic

Asynchronous logic uses memory elements in which state-transitions are controlled by the sequence of changes on the primary inputs. There is thus no way to determine easily when the next state will be established. This is again a problem of timing and memory effects.

Asynchronous logic is faster than synchronous logic, since the speed in asynchronous logic is only limited by gate propagation delays and interconnects. The design of asynchronous logic is then more difficult than synchronous (clocked) logic and must be carried out with due regards to the possibility of critical races (circuit behavior depending on two inputs changing simultaneously) and hazards (occurrence of a momentary value opposite to the expected value).

Non-deterministic behavior in asynchronous logic can cause problems during fault simulation. Time dependency of operation can make testing very difficult, since it is sensitive to tester signal skew.

Avoid Logical Redundancy

Logical redundancy exists either to mask a static-hazard condition, or unintentionally (design bug). In both cases, with a logically redundant node it is not possible to make a primary output value dependent on the value of the redundant node. This means that certain fault conditions on the node cannot be detected, such as a node SA1 of the function F.
Another inconvenience of logical redundancy is the possibility for a non-detectable fault on a redundant node to mask the detection of a fault normally-detectable, such as input C in the second example, masked by a SA1 of a redundant node.

**Avoid Delay Dependent Logic**

Automatic test pattern generators work in logic domains, they view delay dependent logic as redundant combinational logic. In this case the ATPG will see an AND of a signal with its complement, and will therefore always compute a 0 on the output of the AND-gate (instead of a pulse). Adding an OR-gate after the AND-gate output permits to the ATPG to substitute a clock signal directly.

**Avoid Clock Gating**

When a clock signal is gated with any data signal, for example a load signal coming from a tester, a skew or any other hazard on that signal can cause an error on the output of logic.
This is also due to asynchronous type of logic. Clock signals should be distributed in the circuit with respect to synchronous logic structure.

**Distinguish Between Signal and Clock**

This is another timing situation to avoid, in which the tester could not be synchronized if one clock or more are dependent on asynchronous delays (across D-input of flip-flops, for example).

**Avoid Self Resetting Logic**

The self resetting logic is more related to asynchronous logic, since a reset input is independent of clock signal.

Before the delayed reset, the tester reads the set value and continues the normal operation. If a reset has occurred before tester observation, then the read value is erroneous. The solution to this problem is to allow the tester to override by adding an OR-gate, for example, with an inhibition input coming from the tester. By this way the right response is given to the tester at the right time.
Figure 8.10: Avoid Self Resetting Logic

Use Bused Structure

This approach is related, by structure, to partitioning technique. It is very useful for microprocessor-like circuits. Using this structure allows the external tester the access of three buses, which go to many different modules.

Figure 8.11: Use Bused Structure

The tester can then disconnect any module from the buses by putting its output into a high-impedance state. Test patterns can then be applied to each module separately.

Separate Analog and Digital Circuits

Testing analog circuit requires a completely different strategy than for digital circuit. Also the sharp edges of digital signals can cause cross-talk problem to the analog lines, if they are close to each other.
If it is necessary to route digital signals near analog lines, then the digital lines should be properly balanced and shielded. Also, in the cases of circuits like Analog-Digital converters, it is better to bring out analog signals for observation before conversion. For Digital-Analog converters, digital signals are to be brought out also for observation before conversion.

**Ad-Hoc DFT Method**

- Good design practices learnt through experience are used as guidelines:
  - Avoid asynchronous (unclocked) feedback.
  - Make flip-flops initializable.
  - Avoid redundant gates. Avoid large fan-in gates.
  - Provide test control for difficult-to-control signals.
  - Avoid gated clocks.
  - Avoid delay dependant logic.
  - Avoid parallel drivers.
  - Avoid monostable and self-resetting logic.

**Design Reviews**

- Manual analysis
  - Conducted by experts
- Programmed analysis
  - Using design auditing tools
- Programmed enforcement
  - Must use certain design practices and cell types.

Objective: Adherence to design guidelines and testability improvement techniques with little impact on performance and area.
Disadvantages of ad-hoc DFT methods:

- Experts and tools not always available.
- Test generation is often manual with no guarantee of high fault coverage.
- Design iterations may be necessary.

Scan Design Techniques

The set of design for testability guidelines presented above is a set of ad hoc methods to design random logic in respect with testability requirements. The scan design techniques are a set of structured approaches to design (for testability) the sequential circuits.

The major difficulty in testing sequential circuits is determining the internal state of the circuit. Scan design techniques are directed at improving the controllability and observability of the internal states of a sequential circuit. By this the problem of testing a sequential circuit is reduced to that of testing a combinational circuit, since the internal states of the circuit are under control.

Scan Path

The goal of the scan path technique is to reconfigure a sequential circuit, for the purpose of testing, into a combinational circuit. Since a sequential circuit is based on a combinational circuit and some storage elements, the technique of scan path consists in connecting together all the storage elements to form a long serial shift register. Thus the internal state of the circuit can be observed and controlled by shifting (scanning) out the contents of the storage elements. The shift register is then called a scan path.

The storage elements can either be D, J-K, or R-S types of flip-flops, but simple latches cannot be used in scan path. However, the structure of storage elements is slightly different than classical ones. Generally the selection of the input source is achieved using a multiplexer on the data input controlled by an external mode signal. This multiplexer is integrated into the D-flip-flop, in our case; the D-flip-flop is then called MD-flip-flop (multiplexed-flip-flop).
The sequential circuit containing a scan path has two modes of operation: a normal mode and a test mode which configure the storage elements in the scan path.

As analyzed from figure 8.13, in the normal mode, the storage elements are connected to the combinational circuit, in the loops of the global sequential circuit, which is considered then as a finite state machine.

In the test mode, the loops are broken and the storage elements are connected together as a serial shift register (scan path), receiving the same clock signal. The input of the scan path is called scan-in and the output scan-out. Several scan paths can be implemented in one same complex circuit if it is necessary, though having several scan-in inputs and scan-out outputs.

A large sequential circuit can be partitioned into sub-circuits, containing combinational sub-circuits, associated with one scan path each. Efficiency of the test pattern generation for a combinational sub-circuit is greatly improved by partitioning, since its depth is reduced.

Before applying test patterns, the shift register itself has to be verified by shifting in all ones i.e. 111...11, or zeros i.e. 000...00, and comparing.

The method of testing a circuit with the scan path is as follows:

1. Set test mode signal, flip-flops accept data from input scan-in
2. Verify the scan path by shifting in and out test data
3. Set the shift register to an initial state
4. Apply a test pattern to the primary inputs of the circuit
5. Set normal mode, the circuit settles and can monitor the primary outputs of the circuit
6. Activate the circuit clock for one cycle
7. Return to test mode
8. Scan out the contents of the registers, simultaneously scan in the next pattern
Level sensitivity scan design (LSSD)

![Diagram of Level sensitivity scan design](image.png)

The level-sensitive aspect means that the sequential network is designed so that when an input change occurs, the response is independent of the component and wiring delays within the network (Figure 8.14).

The scan path aspect is due to the use of shift register latches (SRL) employed as storage elements. In the test mode they are connected as a long serial shidt register. Each SRL has a specific design similar to a master-slave FF. It is driven by two non-overlapping clocks which can be controlled readily from the primary inputs to the circuit. Input D1 is the normal data input to the SRL; clocks CK1 and CK2 control the normal operation of the SRL while clocks CK3 and CK2 control scan path movements through the SRL. The SRL output is derived at L2 in both modes of operation, the mode depending on which clocks are activated.

Advantages:
- Circuit operation is independent of dynamic characteristics of the logic elements
- ATP generation is simplified
- Eliminate hazards and races
- Simplifies test generation and fault simulation

Boundary Scan Test (BST)

Boundary Scan Test (BST) is a technique involving scan path and self-testing techniques to resolve the problem of testing boards carrying VLSI integrated circuits and/or surface mounted devices (SMD).

Printed circuit boards (PCB) are becoming very dense and complex, especially with SMD circuits, that most test equipment cannot guarantee good fault coverage.

BST (figure 8.15) consists in placing a scan path (shift register) adjacent to each component pin and to interconnect the cells in order to form a chain around the border of the circuit. The BST circuits contained on one board are then connected together to form a single path through the board.

The boundary scan path is provided with serial input and output pads and appropriate clock pads which make it possible to:
• Test the interconnections between the various chip
• Deliver test data to the chips on board for self-testing
• Test the chips themselves with internal self-test

![Boundary Scan Test (BST)](image)

Figure 8.15: Boundary Scan Test (BST)

The advantages of Boundary scan techniques are as follows:

• No need for complex testers in PCB testing
• Test engineers work is simplified and more efficient
• Time to spend on test pattern generation and application is reduced
• Fault coverage is greatly increased.

Other scan techniques:

Partial Scan Method

- **A subset of flip-flops is scanned.**

- **Objectives:**
  - Minimize area overhead and scan sequence length, yet achieve required fault coverage
  - Exclude selected flip-flops from scan:
    - Improve performance
    - Allow limited scan design rule violations
  - Allow automation:
    - In scan flip-flop selection
    - In test generation
  - Shorter scan sequences – reduce application time
Random Access Scan Method

- The scan function is implemented like a random-access memory (RAM)
- All flip-flops form a RAM in scan mode
- A subset of flip-flops can be included in the RAM if partial scan is desired
- In scan mode, any flip-flop can be read or written

Procedure:
- Set test inputs to all test points
- Apply the master reset signal to initialize all memory elements
- Set scan-in address & data, then apply the scan clock
- Repeat the above step until all internal test inputs are scanned
- Clock once for normal operation
- Check states of the output points
- Read the scan-out states of all memory elements by applying the address

Built-in Self Test

Objectives:
1. To reduce test pattern generation cost
2. To reduce volume of test data
3. To reduce test time

Built-in Self Test, or BIST, is the technique of designing additional hardware and software features into integrated circuits to allow them to perform self-testing, i.e., testing of their own operation (functionally, parametrically, or both) using their own circuits, thereby reducing dependence on an external automated test equipment (ATE).

BIST is a Design-for-Testability (DFT) technique, because it makes the electrical testing of a chip easier, faster, more efficient, and less costly. The concept of BIST is applicable to just about any kind of circuit, so its implementation can vary as widely as the product diversity that it caters to. As an example, a common BIST approach for
DRAM's includes the incorporation onto the chip of additional circuits for pattern generation, timing, mode selection, and go-/no-go diagnostic tests.

Advantages of implementing BIST include:
1) Lower cost of test, since the need for external electrical testing using an ATE will be reduced, if not eliminated
2) Better fault coverage, since special test structures can be incorporated onto the chips
3) Shorter test times if the BIST can be designed to test more structures in parallel
4) Easier customer support and
5) Capability to perform tests outside the production electrical testing environment. The last advantage mentioned can actually allow the consumers themselves to test the chips prior to mounting or even after these are in the application boards.

Disadvantages of implementing BIST include:
1) Additional silicon area and fab processing requirements for the BIST circuits
2) Reduced access times
3) Additional pin (and possibly bigger package size) requirements, since the BIST circuitry need a way to interface with the outside world to be effective and
4) Possible issues with the correctness of BIST results, since the on-chip testing hardware itself can fail.

Techniques are:
- compact test: signature analysis
- linear feedback shift register
- BILBO
- self checking technique

**Compact Test: Signature analysis**

Signature analysis performs polynomial division that is, division of the data out of the device under test (DUT). This data is represented as a polynomial P(x) which is divided by a characteristic polynomial C(x) to give the signature R(x), so that

\[ R(x) = \frac{P(x)}{C(x)} \]

This is summarized as in figure 8.16.
Linear feedback shift register (LFSR):

An LFSR is a shift register that, when clocked, advances the signal through the register from one bit to the next most-significant bit. Some of the outputs are combined in exclusive-OR configuration to form a feedback mechanism. A linear feedback shift register can be formed by performing exclusive-OR (Figure 8.16) on the outputs of two or more of the flip-flops together and feeding those outputs back into the input of one of the flip-flops.

LFSR technique can be applied in a number of ways, including random number generation, polynomial division for signature analysis, and n-bit counting. LFSR can be series or parallel, the differences being in the operating speed and in the area of silicon occupied; Parallel LFSR being faster but larger than serial LFSR.

Built-in logic block observer (BILBO):

BILBO is a built-in test generation scheme which uses signature analysis in conjunction with a scan path. The major component of a BILBO is an LFSR with a few gates (Figure 8.17).

A BILBO register (built-in logic block observer) combines normal flipflops with a few additional gates to provide four different functions. The example circuit shown in the applet realizes a four-bit register. However, the generalization to larger bit-widths should be obvious, with the XOR gates in the LFSR feedback path chosen to implement a good polynomial for the given bit-width.
When the A and B control inputs are both 1, the circuit functions as a normal parallel D-type register.

When both A and B inputs are 0, the D-inputs are ignored (due to the AND gate connected to A), but the flipflops are connected as a shift-register via the NOR and XOR gates. The input to the first flipflop is then selected via the multiplexer controlled by the S input. If the S input is 1, the multiplexer transmits the value of the external SIN shift-in input to the first flipflop, so that the BILBO register works as a normal shift-register. This allows to initialize the register contents using a single signal wire, e.g. from an external test controller.

If all of the A, B, and S inputs are 0, the flipflops are configured as a shift-register, again, but the input bit to the first flipflop is computed by the XOR gates in the LFSR feedback path. This means that the register works as a standard LFSR pseudorandom pattern generator, useful to drive the logic connected to the Q outputs. Note that the start value of the LFSR sequence can be set by shifting it in via the SIN input.

Finally, if B and S are 0 but A is 1, the flipflops are configured as a shift-register, but the input value of each flipflop is the XOR of the D-input and the Q-output of the previous flipflop. This is exactly the configuration of a standard LFSR signature analysis register.

Because a BILBO register can be used as a pattern generator for the block it drives, as well provide signature-analysis for the block it is driven by, a whole circuit can be made self-testable with very low overhead and with only minimal performance degradation (two extra gates before the D inputs of the flipflops).
Figure 8.17: BIST – BILBO
Self-checking techniques:

It consists of logic block and checkers should then obey a set of rules in which the logic block is ‘strongly fault secure’ and the checker ‘strongly code disjoint’. The code use in data encoding depends on the type of errors that may occur at the logic block output. In general three types are possible:

- Simple error: one bit only affected at a time.
- Unidirectional error: multiple bits at 1 instead of 0 (or 0 instead of 1)
- Multiple errors: multiple bits affected in any order.

Self-checking techniques are applied to circuits in which security is important so that fault tolerance is of major interest. Such technique will occupy more area in silicon than classical techniques such as functional testing but provide very high test coverage.