Lecture 22
EE 421 / CS 425
Digital System Design

Fall 2023

Shahid Masud

Topics

- **Special Features in FPGA**
- Sequential Implementation on CLB
- Memory
- Multipliers
- DSP Slices
- FIR and Symmetric Filters

- **Faults and Testing**
- Examples of Path Sensitization Method
- EXOR Method for Fault Generation
- What is Design for Testability?
- BIST and SCAN technique

Specialized Modules in FPGAs

- Dedicated Memory
  - Single Port and Dual Port Embedded Memory Blocks – Block RAM
- Dedicated Arithmetic Units
  - Adders, Multipliers, Multipliers – Accumulators, Fast Carry Logic
- Digital Signal Processing Blocks – DSP Slice
  - FFT Butterfly Modules, FIR / IIR Filters,
  - IP Core Libraries for Encryption, Video Compression, Cloud Applications, etc.
- Embedded Processors
  - PowerPC, Microblaze, NIOS, ARM, MIPS, etc.
- Content Addressable Memory (CAM)
  - used in Branch Prediction, Caches inside CPU
- More and more features keep appearing in new FPGA devices
  - High Speed Interfaces, Security Features, RISC-V Support, etc.

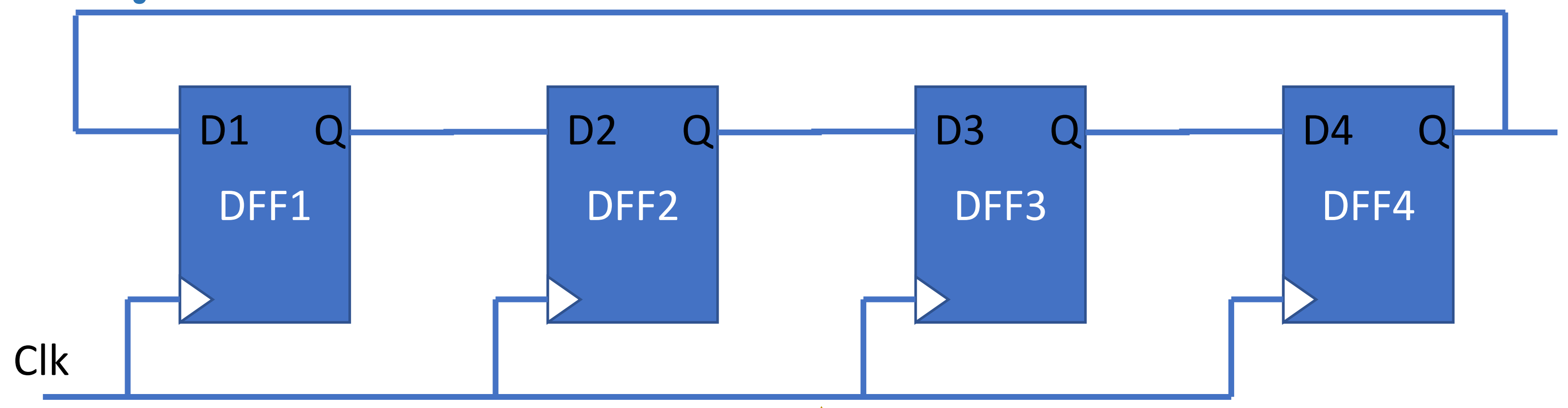
Implementation of memory in FPGA

Using LUT in CLBs – Distributed RAM

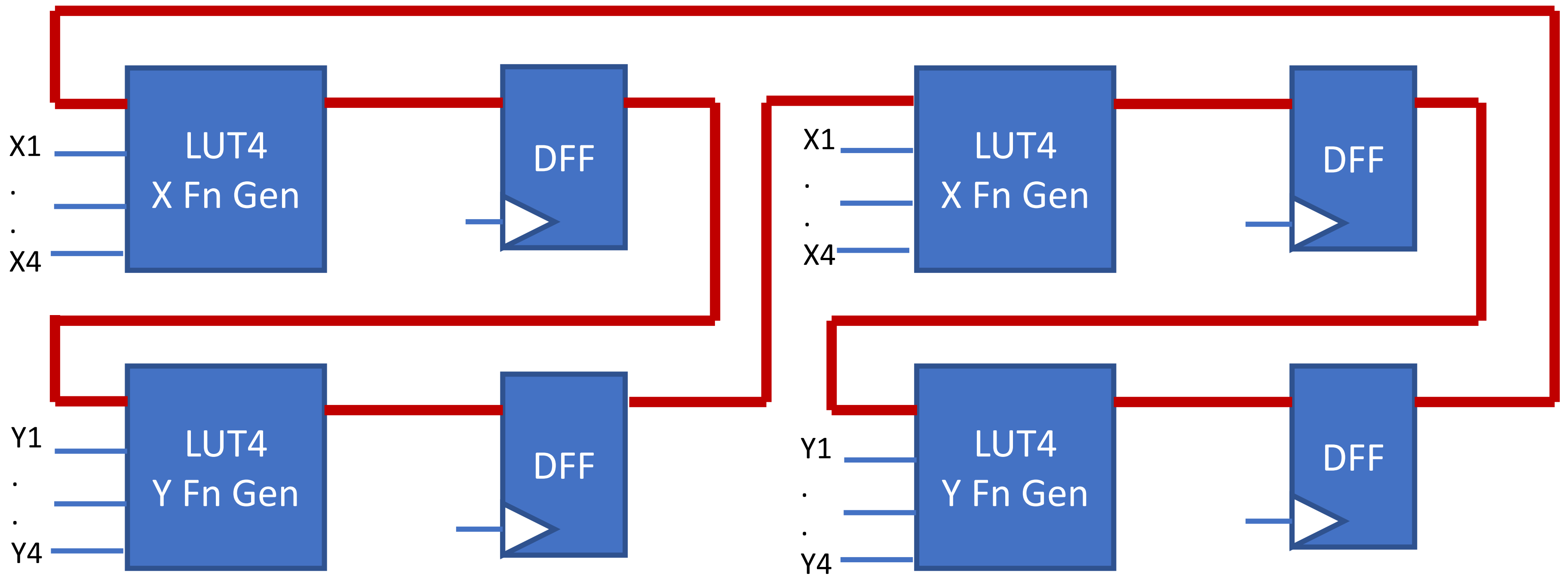
Instantiating Block RAMs

Provision of Dual port memory in modern FPGA

Sequential Circuits in FPGA



D1=Q4
D2=Q1
D3=Q2
D4=Q3



Multiplier Blocks – Xilinx Spartan-3AN

XILINX **Spartan-3AN FPGA Family: Introduction and Ordering Information**

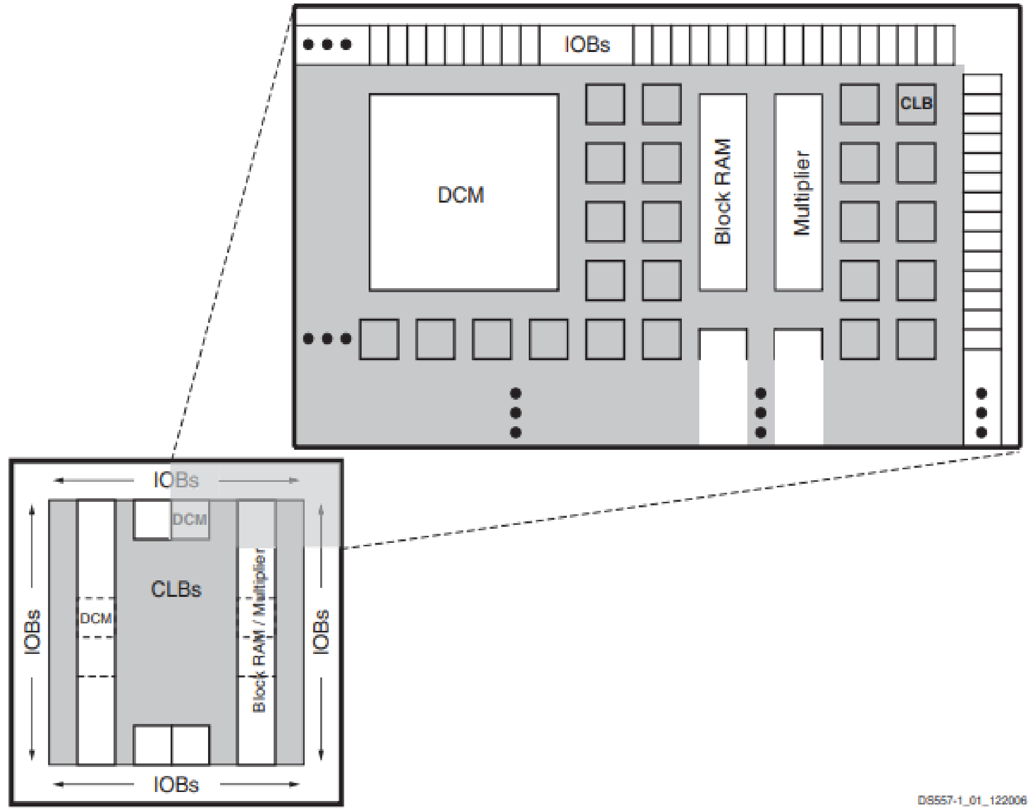
Architectural Overview

The Spartan-3AN FPGA architecture is compatible with that of the Spartan-3A FPGA. The architecture consists of five fundamental programmable functional elements:

- **Configurable Logic Blocks (CLBs)** contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches.
- **Input/Output Blocks (IOBs)** control the flow of data between the I/O pins and the internal logic of the device. IOBs support bidirectional data flow plus 3-state operation. They support a variety of signal standards, including several high-performance differential standards. Double Data-Rate (DDR) registers are included.
- **Block RAM** provides data storage in the form of 18-Kbit dual-port blocks.
- **Multiplier Blocks** accept two 18-bit binary numbers as inputs and calculate the product.
- **Digital Clock Manager (DCM) Blocks** provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

These elements are organized as shown in Figure 1. A dual ring of staggered IOBs surrounds a regular array of CLBs. Each device has two columns of block RAM except for the XC3S50AN, which has one column. Each RAM column consists of several 18-Kbit RAM blocks. Each block RAM is associated with a dedicated multiplier. The DCMs are positioned in the center with two at the top and two at the bottom of the device. The XC3S50AN has DCMs only at the top, while the XC3S700AN and XC3S1400AN add two DCMs in the middle of the two columns of block RAM and multipliers.

The Spartan-3AN FPGA features a rich network of traces that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.



Notes:

1. The XC3S700AN and XC3S1400AN have two additional DCMs on both the left and right sides as indicated by the dashed lines. The XC3S50AN has only two DCMs at the top and only one Block RAM/Multiplier column.

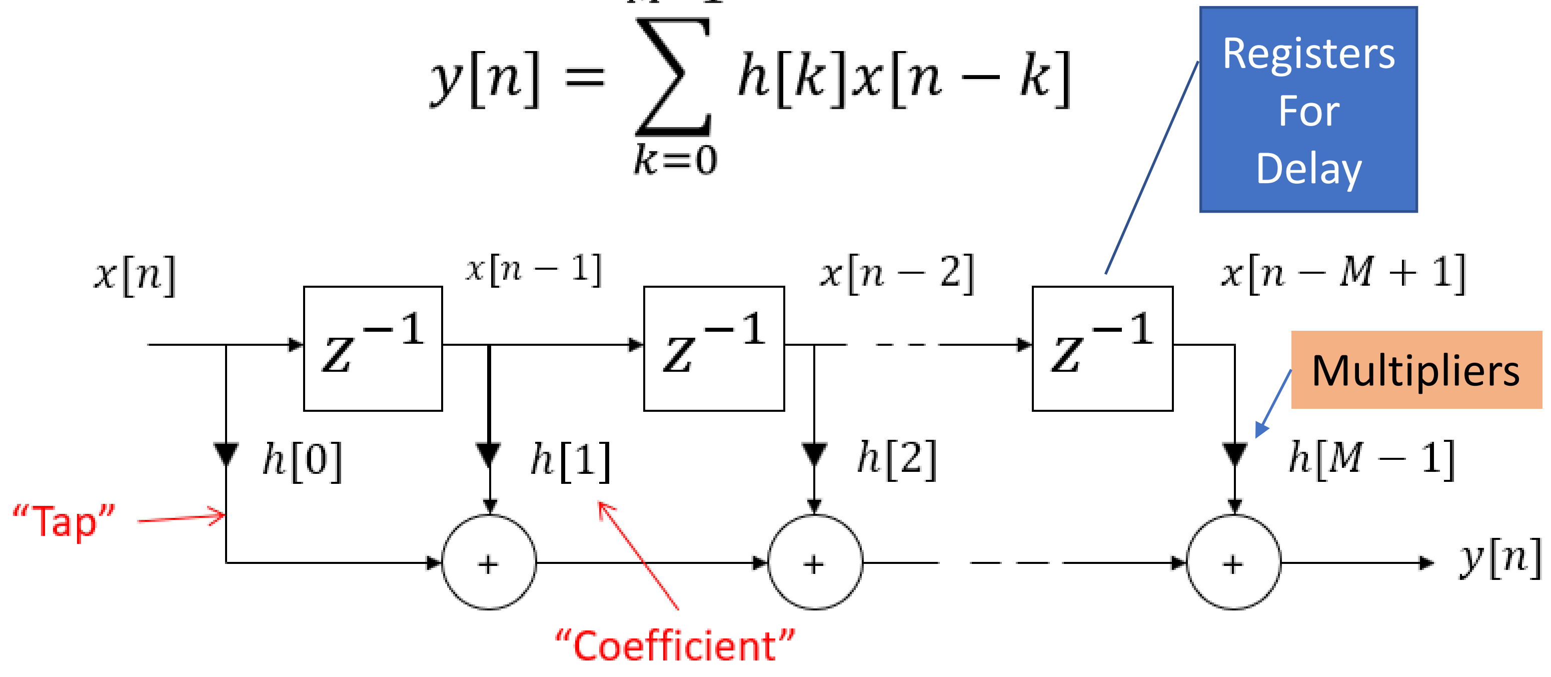
Figure 1: **Spartan-3AN Family Architecture**

DSP Features in modern FPGA
Example FIR Filter Implementation

FIR Filter Design

- FIR system is easily implemented directly from convolution summation

$$y[n] = \sum_{k=0}^{M-1} h[k]x[n-k]$$

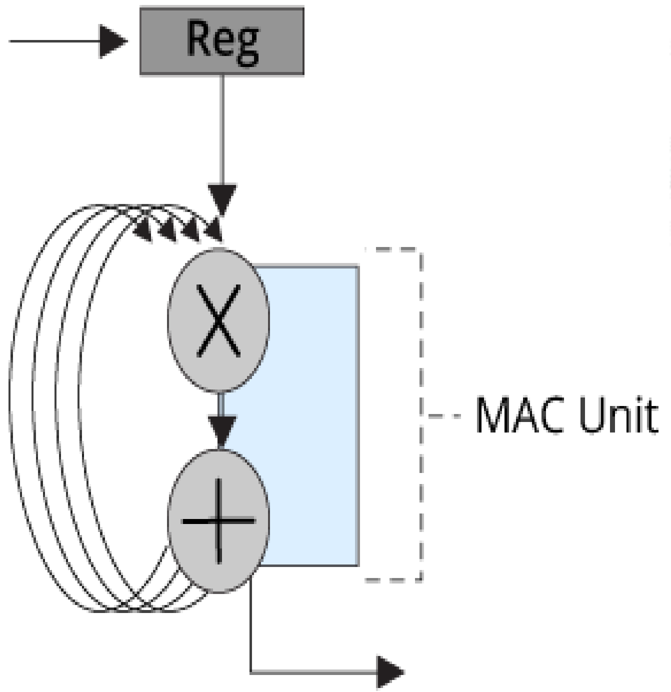


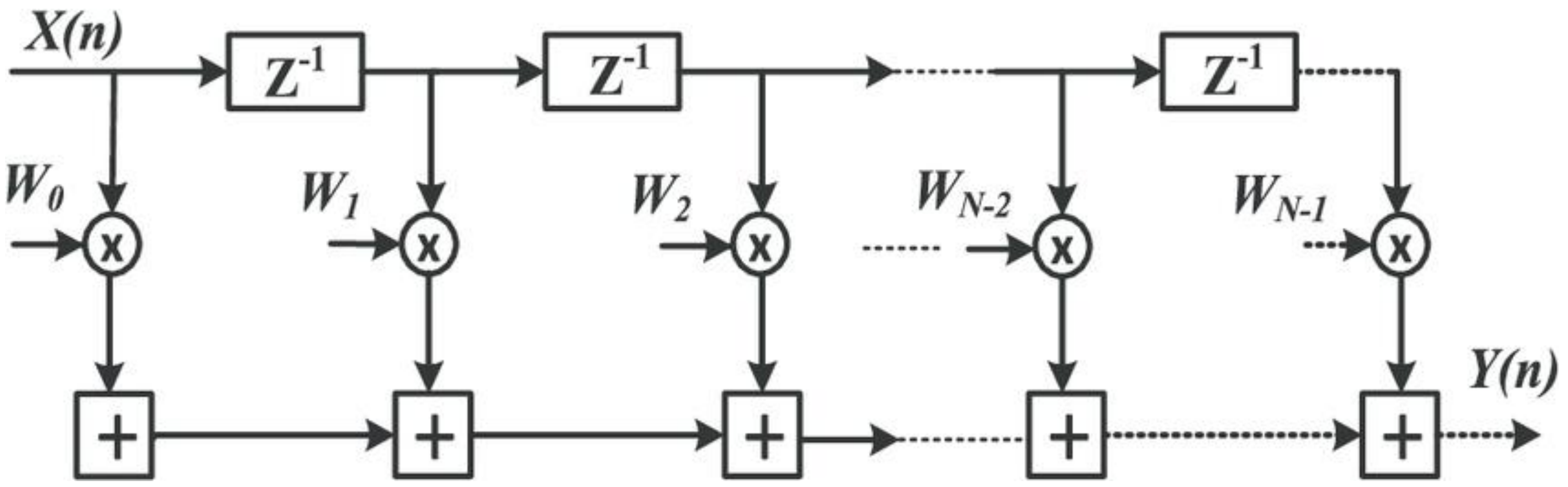
Copyright © 2019, Dan Boschen

Implementation of DSP Filters

Conventional DSP Device
(Von Neumann Architecture)

Implementation of FIR filters in Digital Signal Processing

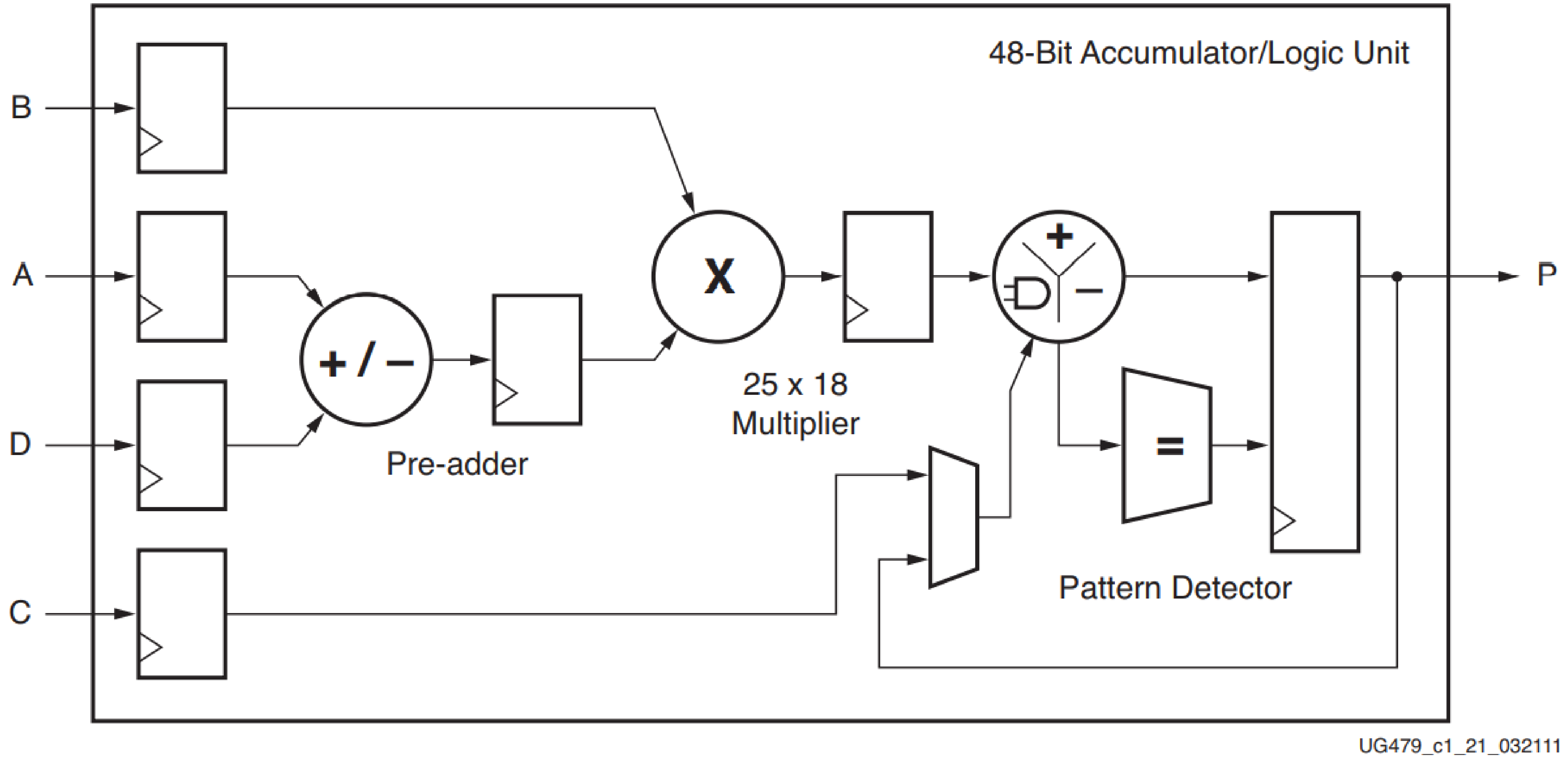




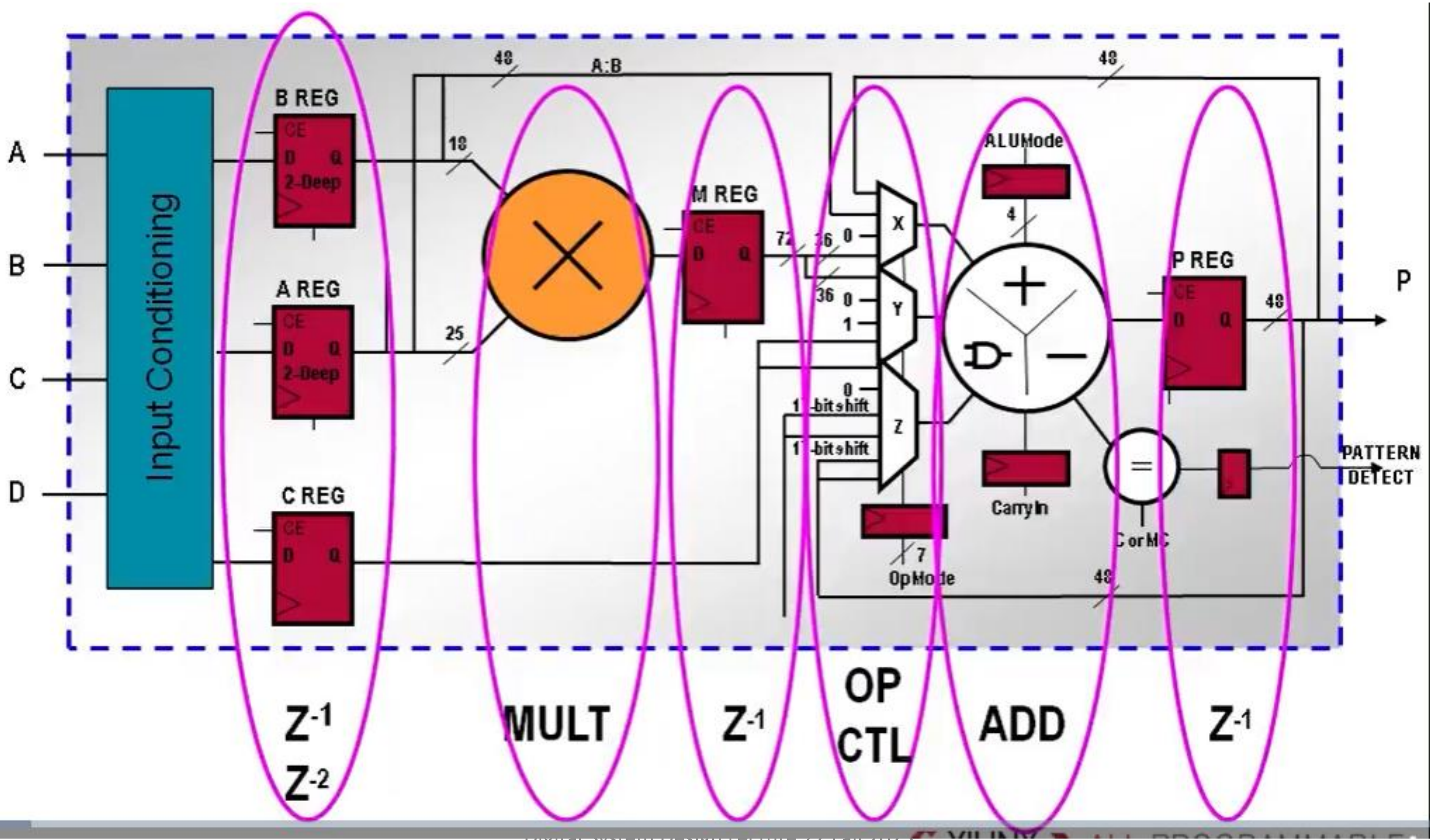
**FIF filter mapping on
Software Programmable Device**

**FIF filter mapping on
a configurable Hardware Device**

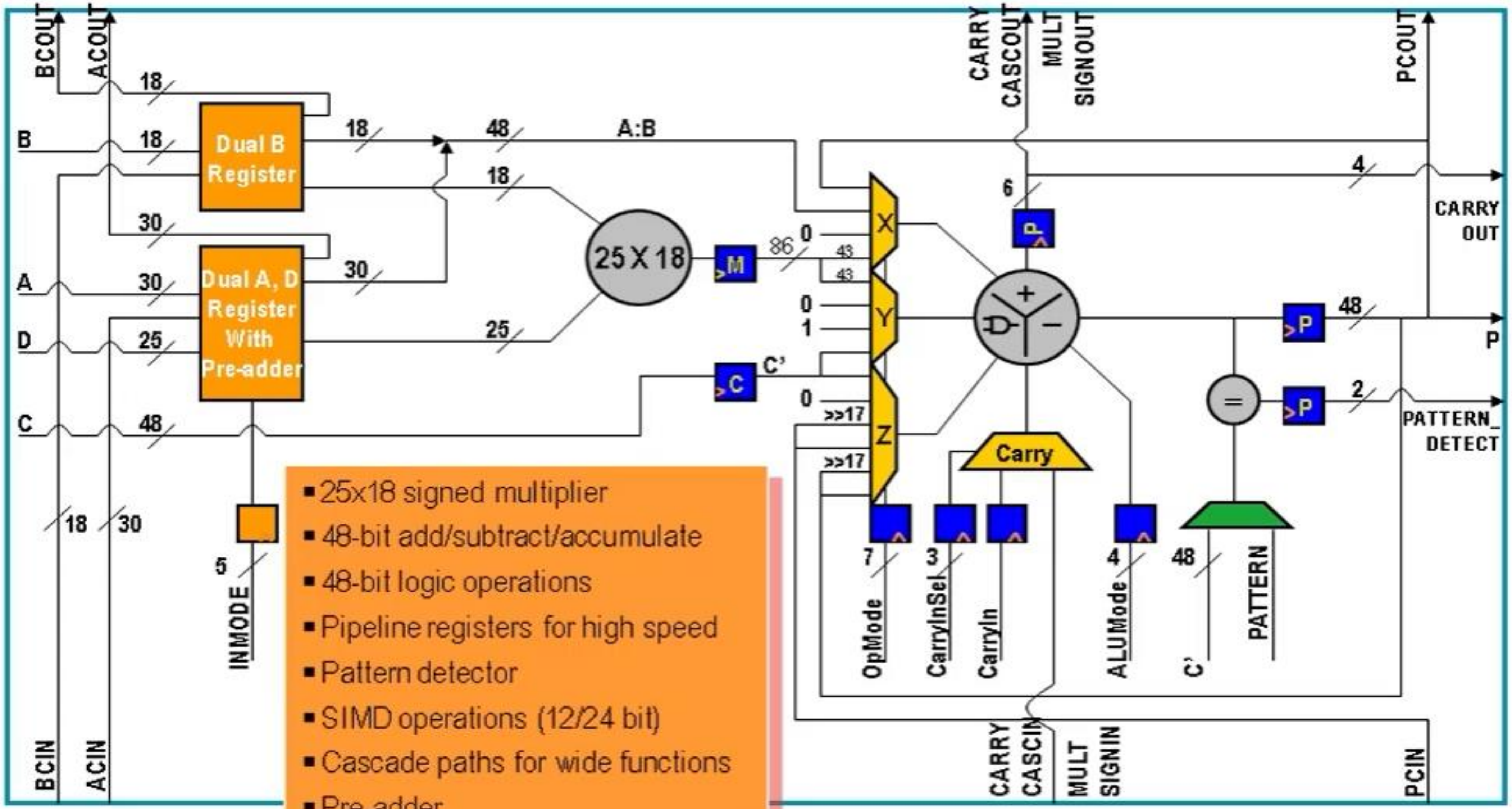
Basic Xilinx DSP48 Slice Architecture



DSP Slice Features



Example of DSP Slice and Features



- 25x18 signed multiplier
- 48-bit add/subtract/accumulate
- 48-bit logic operations
- Pipeline registers for high speed
- Pattern detector
- SIMD operations (12/24 bit)
- Cascade paths for wide functions
- Pre-adder

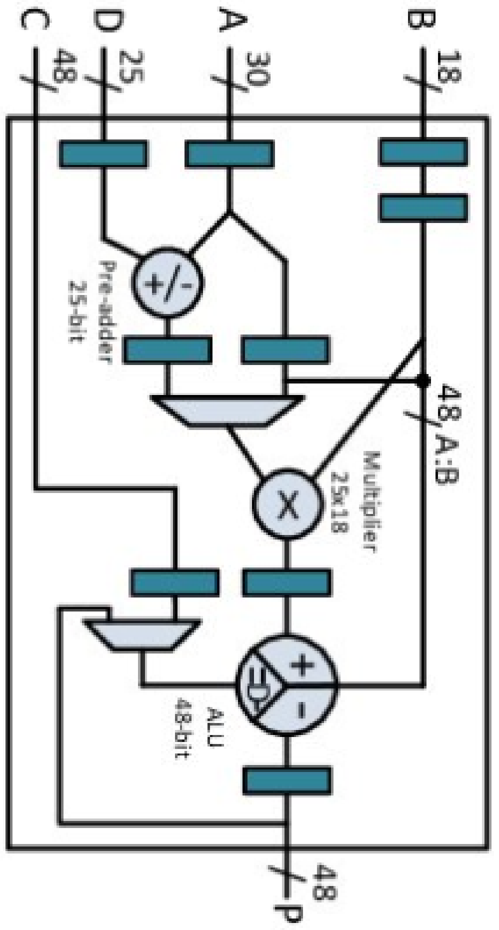
Xilinx DSP48 Slice Functionality

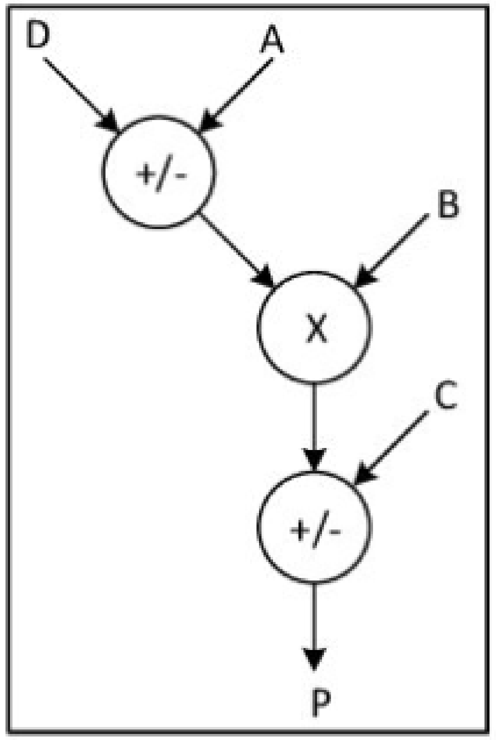
- 25 x 18 two's complement multiplier
- 48-bit Accumulator
- Power saving Pre-Adder for symmetric FIR filter implementation
- Single-Instruction-Multiple-Data (SIMD) arithmetic unit
- Dual 24-bit or Quad 12-bit Add/Sub/Acc
- Optional Logic Unit with 10 different operations on two operands
- Pattern Detector for convergent or symmetric rounding
- 96-bit wide Logic functions in conjunction with

Pattern Detector and Logic Unit

- Optional Pipelining and Dedicated Buses for Cascading

Mapping Add and Mult on DSP Slice





Figure

Caption

Fig. 4: Dataflow through the DSP48E1 primi-tive.

This figure was uploaded by Suhaib A Fahmy

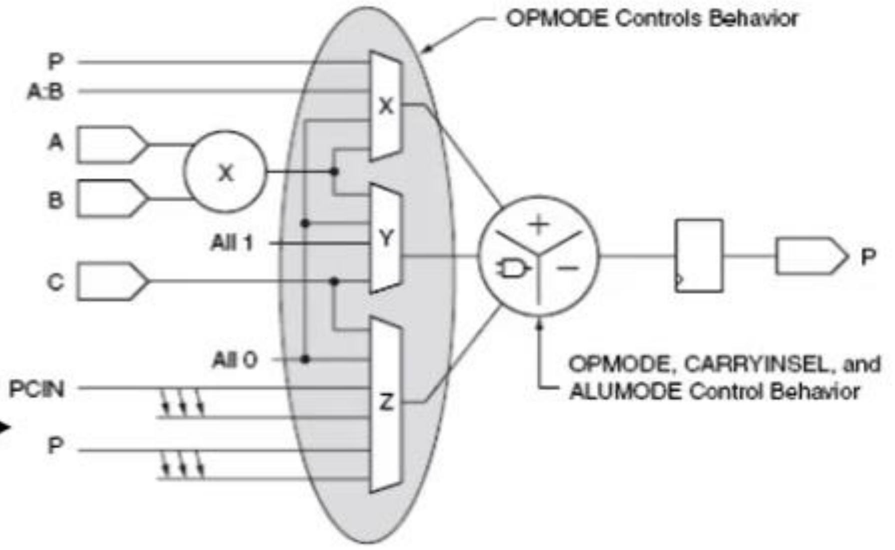
Content may be subject to copyright.

X, Y and Z Multiplexer

- Adder/subtractor operates on X, Y, Z and CIN operands
  - Table shows basic operations
- X, Y, and Z multiplexers allow for dynamic OPMODEs
- Multiplier output requires both X and Y multiplexers

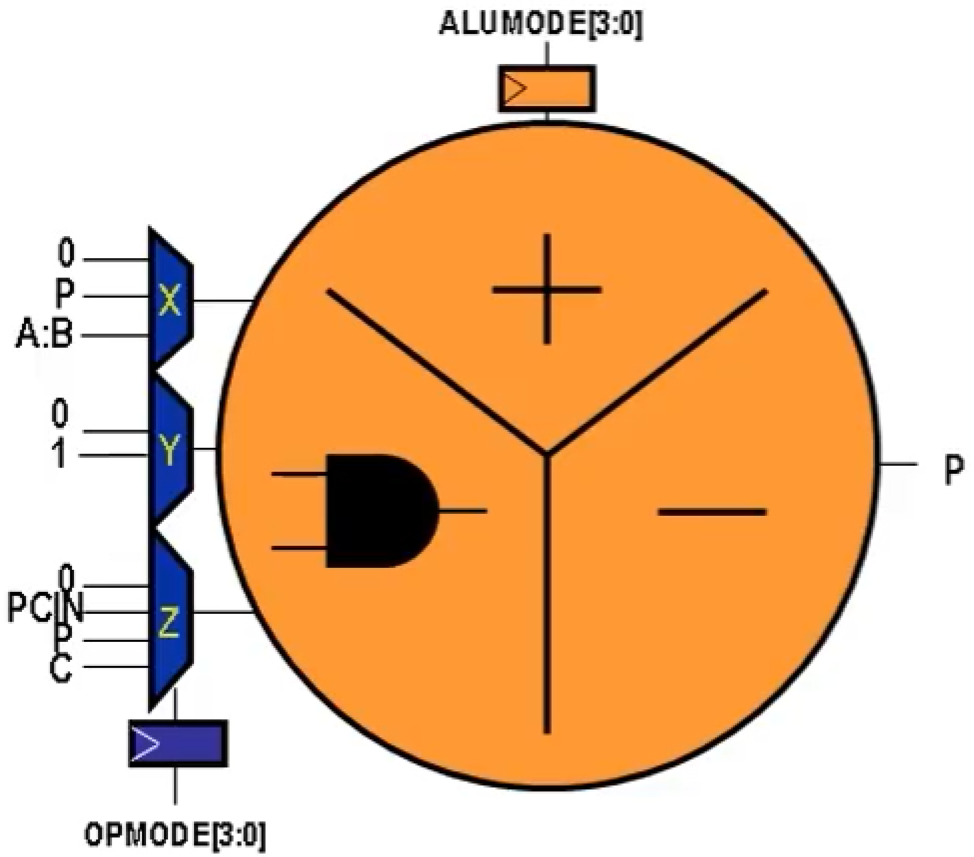
ALUMODE	Operation
0000	Z + X + Y + CIN
0001	-Z + (X + Y + CIN) – 1
0010	-Z – X – Y – CIN – 1
0011	Z – (X + Y + CIN)
Others	Logic Operations

Normal or 17-bit right shifted with MSB fill for multi-precision arithmetic



Two Input Logic Functions in DSP Slice

- 48-bit logic operations
  - XOR, XNOR, AND, NAND, OR, NOR, NOT



ALUMODEs

Logic Unit Mode	OPMODE[3:2]	ALUMODE[3:0]
X XOR Z	00	0100
X XNOR Z	00	0101
X XNOR Z	00	0110
X XOR Z	00	0111
X AND Z	00	1100
X AND (NOT Z)	00	1101
X NAND Z	00	1110
(NOT X) OR Z	00	1111
X XNOR Z	10	0100
X XOR Z	10	0101
X XOR Z	10	0110
X XNOR Z	10	0111
X OR Z	10	1100
X OR (NOT Z)	10	1101
X NOR Z	10	1110
(NOT X) AND Z	10	1111

An Implementation of FIR using RAM

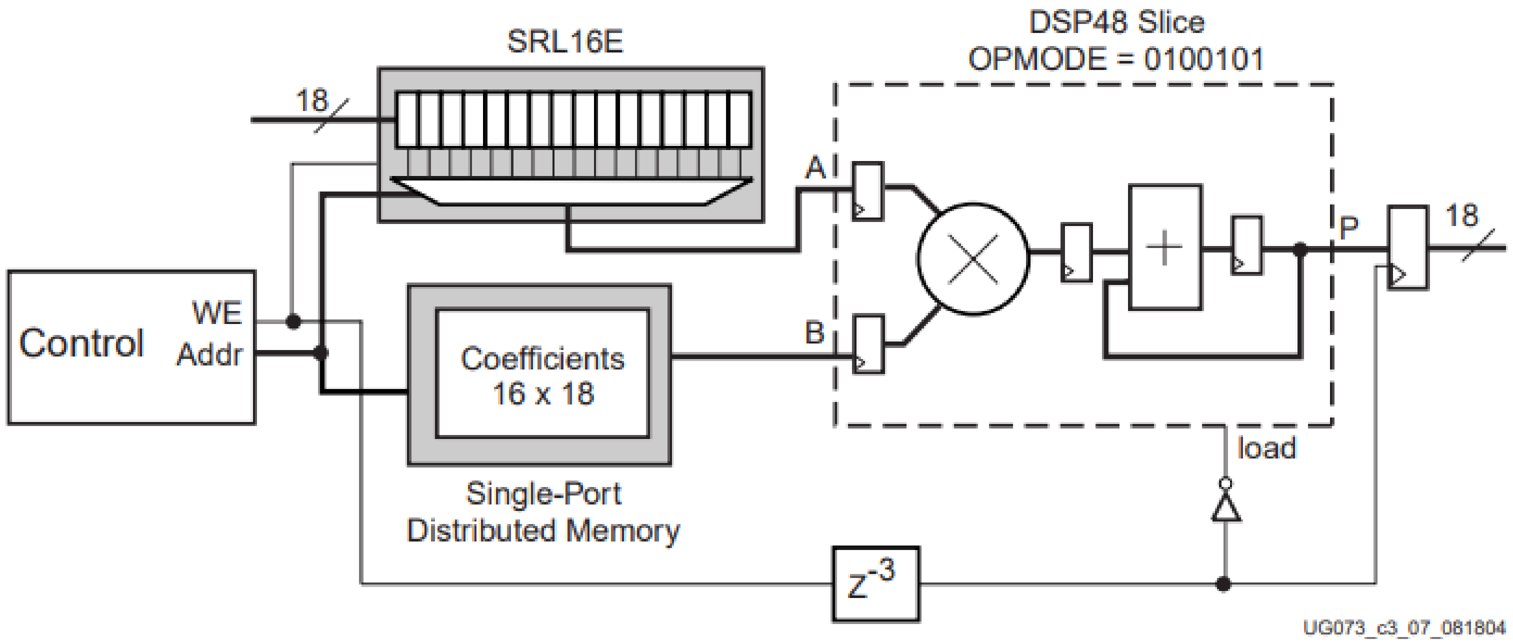


Figure 4-6: **Tap-Distributed RAM MAC FIR Filter**

MAC Engine for FIR Filter in FPGA

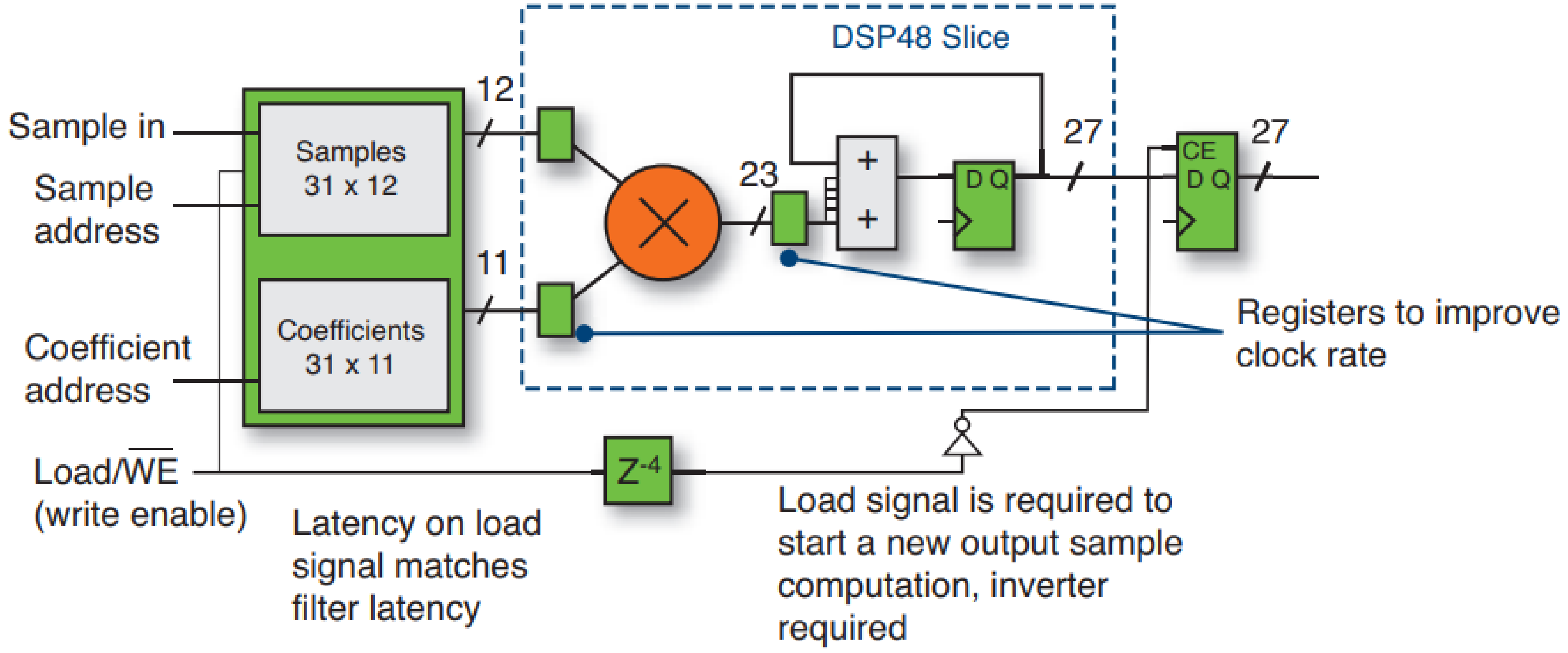

DSP48 Slice
Sample in
Sample address
Samples 31 x 12
Coefficients 31 x 11
Coefficient address
12
11
23
27
27
D Q
CE
D Q
Registers to improve clock rate
Load/$\overline{\text{WE}}$ (write enable)
Z^{-4}
Latency on load signal matches filter latency
Load signal is required to start a new output sample computation, inverter required

Figure 4 – MAC engine FIR filter in an FPGA

Symmetric FIR

Filter Coefficients are
Like Mirror values

Symmetric MAC FIR Filter

The HDL code provided in the reference design is for a single multiplier MAC FIR filter. other techniques can also be explored. This section describes how the symmetric nature of FIR filter coefficients can double the capable sample rate performance of the filter (assuming the same clock speed). By rearranging the FIR filter equation, the coefficients are exploited as follows:

$$(X0 \times C0) + (Xn \times Cn) \;\ldots \rightarrow (X0 + Xn) \times C0 \qquad (\text{if } C0 = Cn) \qquad \text{Equation 4-6}$$

Figure 4-7 shows the architecture for a symmetric MAC FIR filter.

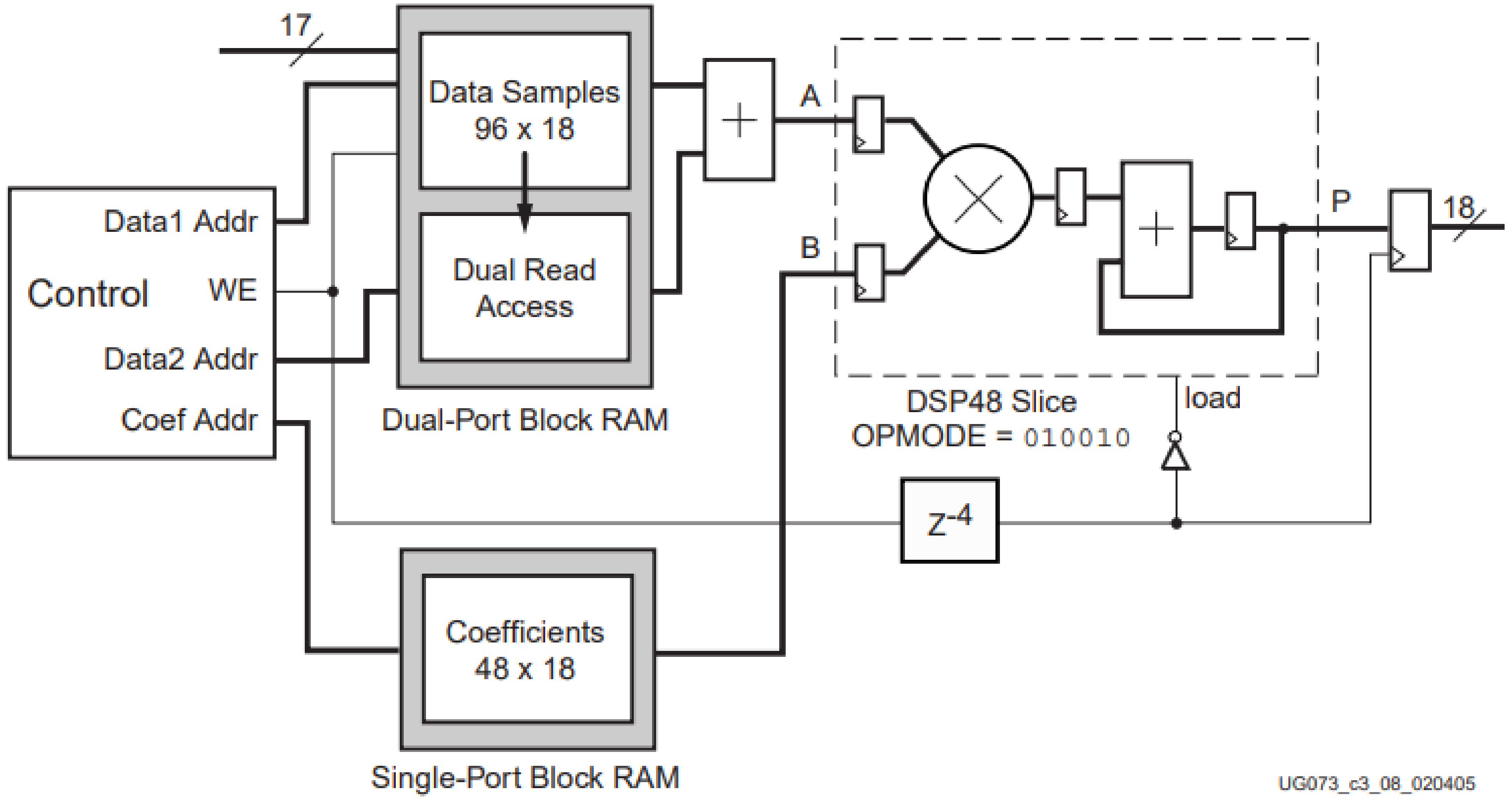
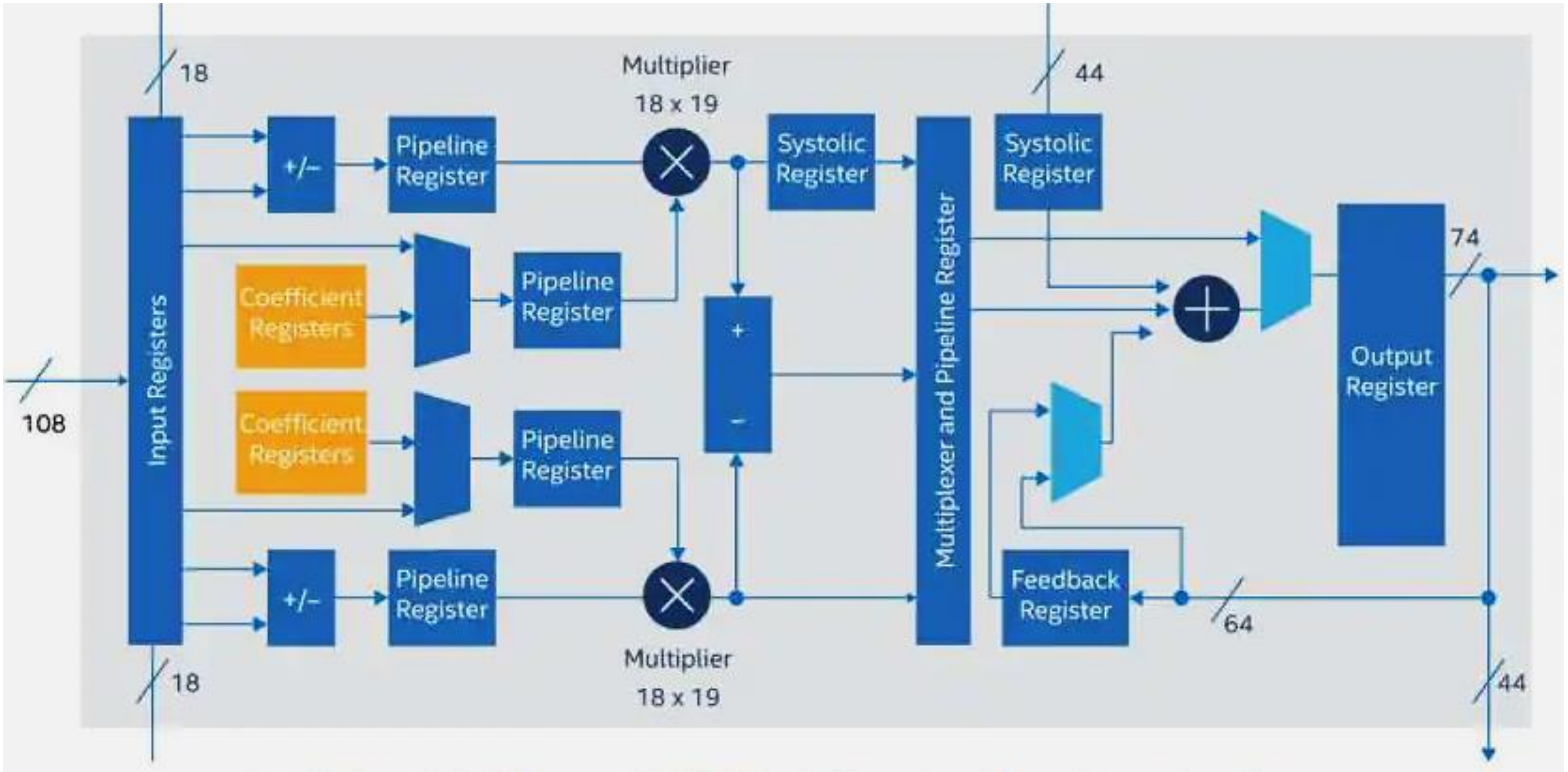


Figure 4-7: **Symmetric MAC FIR Filter**

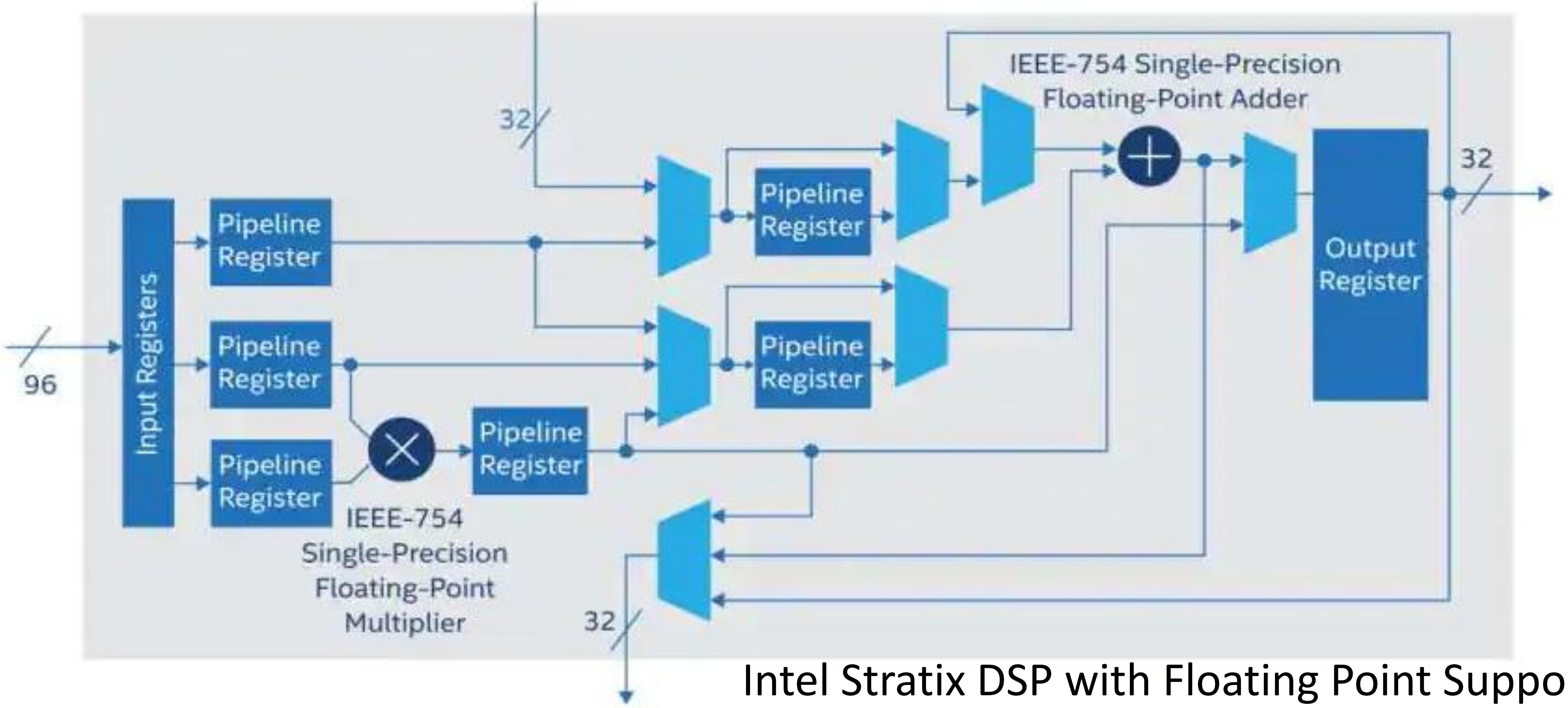
Intel Stratix Slice

Intel Stratix DSP Block – Fixed Point



Intel® Stratix® 10 Device DSP Block: Standard-Precision Fixed Point

Intel Stratix DSP Slice with Floating Point



Intel Stratix DSP with Floating Point Support

Intel® Stratix® 10 Device DSP Block: Single-Precision Floating Point

Further Reading

- https://www.xilinx.com/video/fpga/7-series-dsp-resources.html

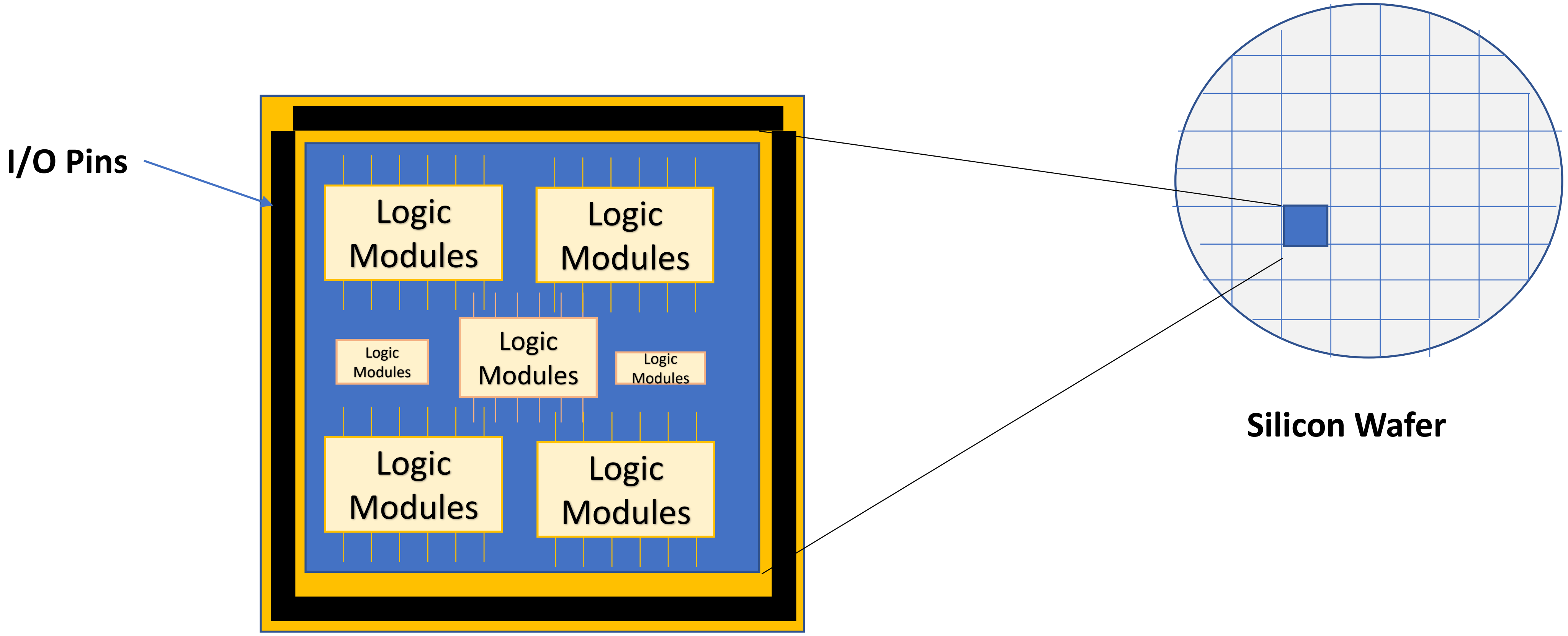
Topics: Faults and Testing, Examples of path sensitization method for fault tests, EX-OR method truth table, EX-OR method Boolean expression, testing of sequential elements using Scan cells

Faults and Testing

Types of Circuit Failure

- The domain of hardware related failure
- Permanent Failure: Incorrect behaviour at all times
- Intermittent Failure: Occurs randomly for finite time duration
- Transient Failure: Occurs in presence of certain environmental conditions such as high temperature, radiation, etc.
- Reasons for Failure: Wafer defects, impurities in clean room, mask mis-alignment, process imperfections, vibrations in equipment

Faults and Failure in Logic Circuit Chip



Number of internal inputs and outputs is much more than the number of physical I/O pins available

Production testing

- Detection of permanent errors caused by manufacturing defects. Involves two major steps:
  - Test Generation, and
  - Fault Simulation
- Failure modes are called 'Faults'
- Set of vectors generated to detect 'Faults' is called 'Fault-Simulation'
- 'Fault-Models' consider the logic effects that result from the physical faults in a circuit
- When a circuit fails to behave correctly, implies that the logic realized is different from logic that was specified for design

Chip Level Faults

Chip Level Fault Type	Degradation Fault	Open Circuit	Short Circuit
Leakage or Short between package leads	Yes		Yes
Broken or missing wire bonding		Yes	
Surface contamination or moisture	Yes		
Metal migration, stress peeling		Yes	Yes
Metallization		Yes	Yes

Gate Level Faults

Gate Level Fault Type	Degradation Fault	Open Circuit	Short Circuit
Contact Open		Yes	
Gate to Source short circuit	Yes		Yes
Field Oxide Parasitic Device	Yes		Yes
Gate Oxide Flaw, Spiking	Yes		Yes
Mask Misalignment	Yes		Yes

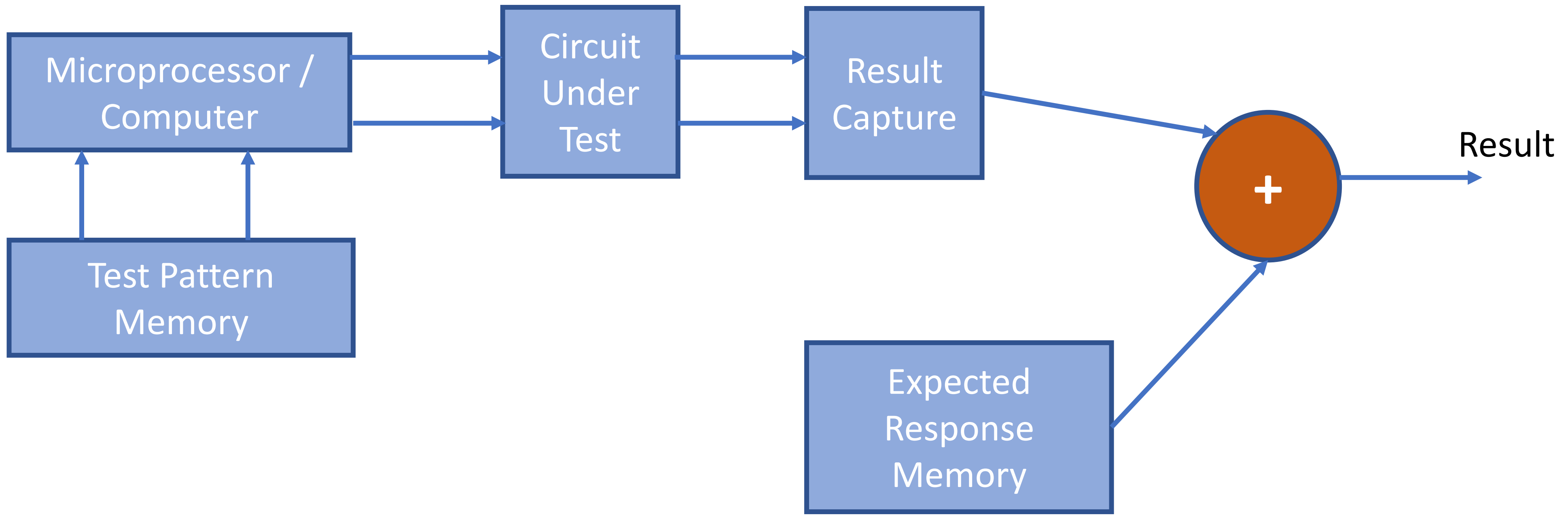
Fault Types

- Stuck Faults: A signal line is shorted to supply or ground permanently
- Bridging Faults: Short circuits in the interconnects between transistors in a logic cell are called bridging faults
- Bridging faults are detected by measuring the quiescent current through the CMOS logic circuit. It takes more time to detect Bridging faults whereas Stuck-At faults are easier to locate.
- Problem: The fault sites are typically located in the middle of the logic circuit and their inputs or outputs are not directly accessible from i/o pins.
- There are maximum 100s of i/o pins vs the number of gates and their interconnects is in millions

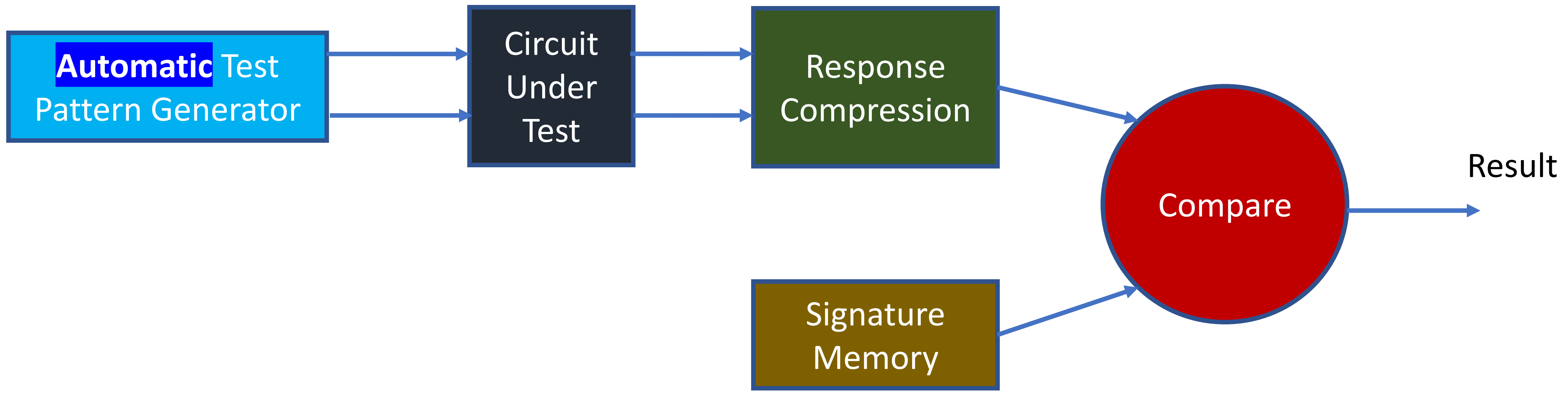
Un-Testable Faults

- Redundant Logic
- Un-Controllable Nets
- Un-Observable Nets that cannot be sensitized through I/O pins

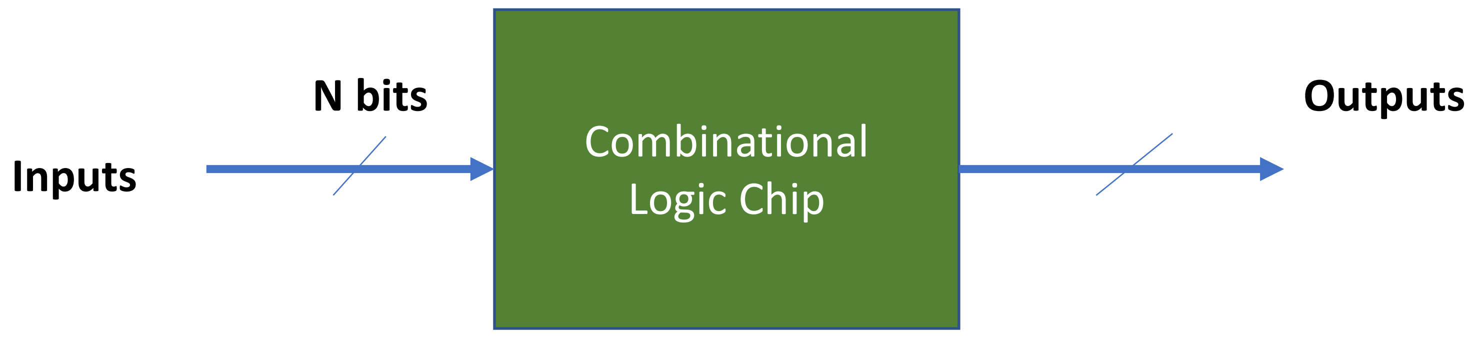
Typical Test Setup



Automated Test Setup



Exhaustive Testing



Requires 2^N test vectors to exhaustively test all input combinations

Exhaustive Testing compares correct and faulty outputs for each input combination
This is a very slow approach

Single Stuck-At Fault Models

Stuck-At Fault Model: Assumes that there is just one stuck-at fault in circuit under test. Hope that single fault removal will remove multiple faults as well.

Stuck at 0 / Stuck at 1 (SA0/SA1) faults: Only two types of logical faults assumed in the model at gate level.

Observability: The degree to which one can observe a node at the output pins of an IC package.

Given that only a limited number of nodes could be directly observed, alternative methods such as JTAG are used to observe all outputs with some delays.

Controllability: Measure of the ease of setting the node to ‘1’ or ‘0’ state. Easiest would be directly settable by an input pin on IC package

Fault Coverage

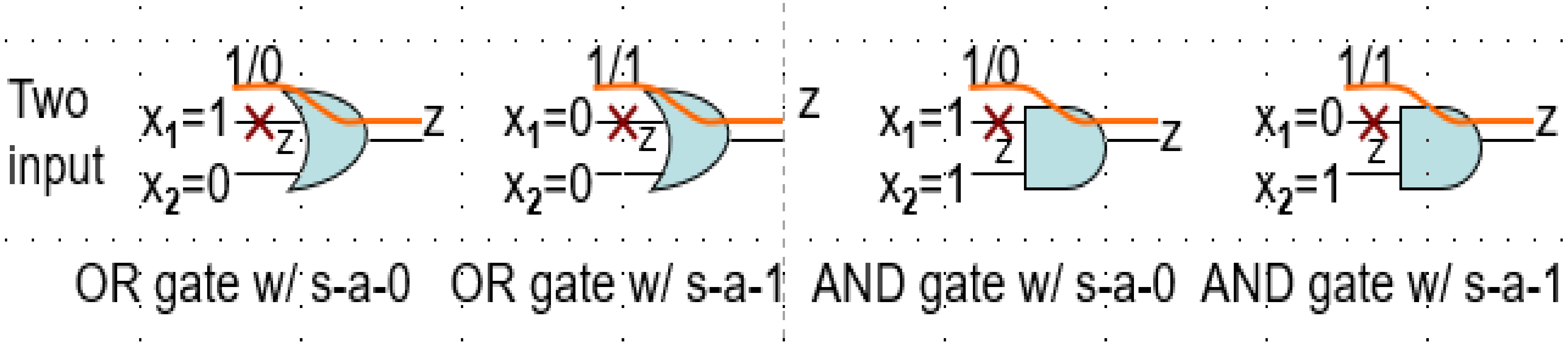
- % Fault Coverage = $\frac{No.of\ nodes\ when\ set\ to\ 1\ or\ 0\ result\ in\ detection\ of\ fault}{total\ number\ of\ nodes\ in\ the\ circuit}$

- KN Cycles are needed; K = no. of nodes in the circuit
- N/2 cycles are needed to detect each fault
- N = length of test sequence
- In turn, every node is tested for SA0 and SA1 sequentially i.e. Sequential Fault Grading

Fault Representation

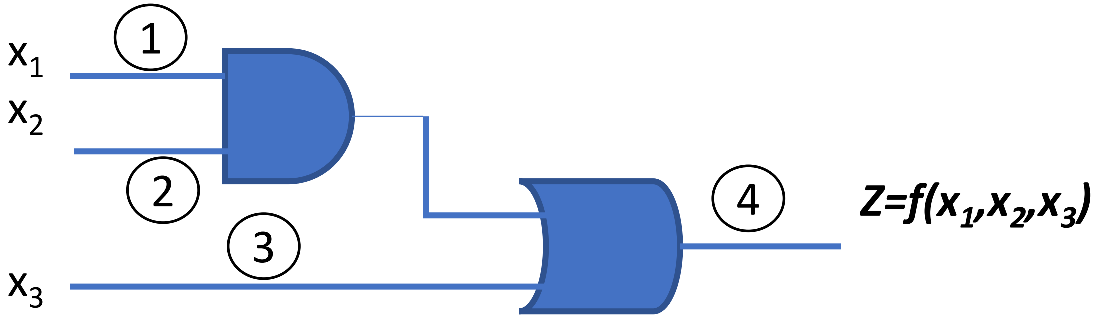
- **$f(x_n) = f(x_1, x_2, x_3, \ldots\ldots, x_n)$** represents a fault-free circuit
- **$f^{p/d}\ (x_n)$** represents the same circuit with fault p/d;
- Where p is a wire label, d is ‘0’ or ‘1’ representing SA0 or SA1 respectively
- n is the no. of input variables

Different Faults and Input vectors



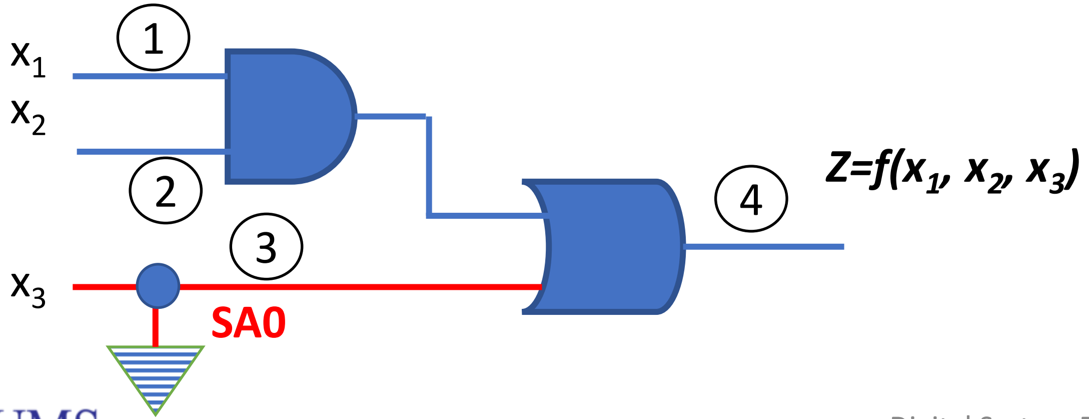
How to observe different faults at outputs of gates?

Example of fault representation



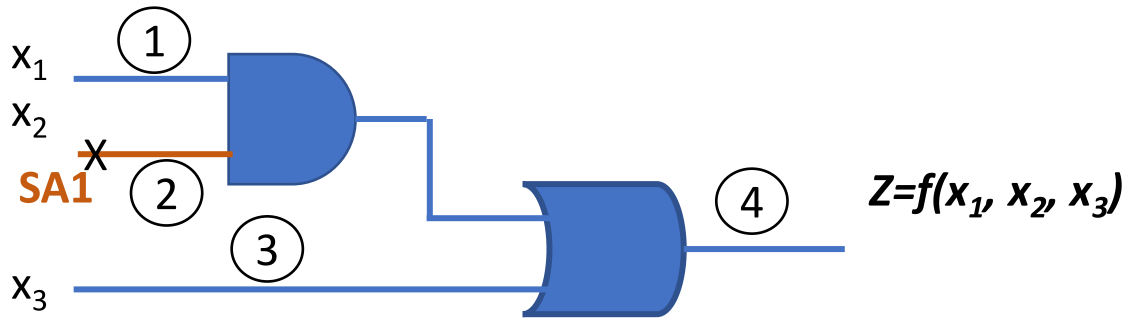
Stuck at 0 (SA0) fault at node 3:

$f^{3/0}(x_3) = x_1.x_2$



Stuck at 1 (SA1) fault at node 2:

$f^{2/1}(x_2) = x_1 + x_3$



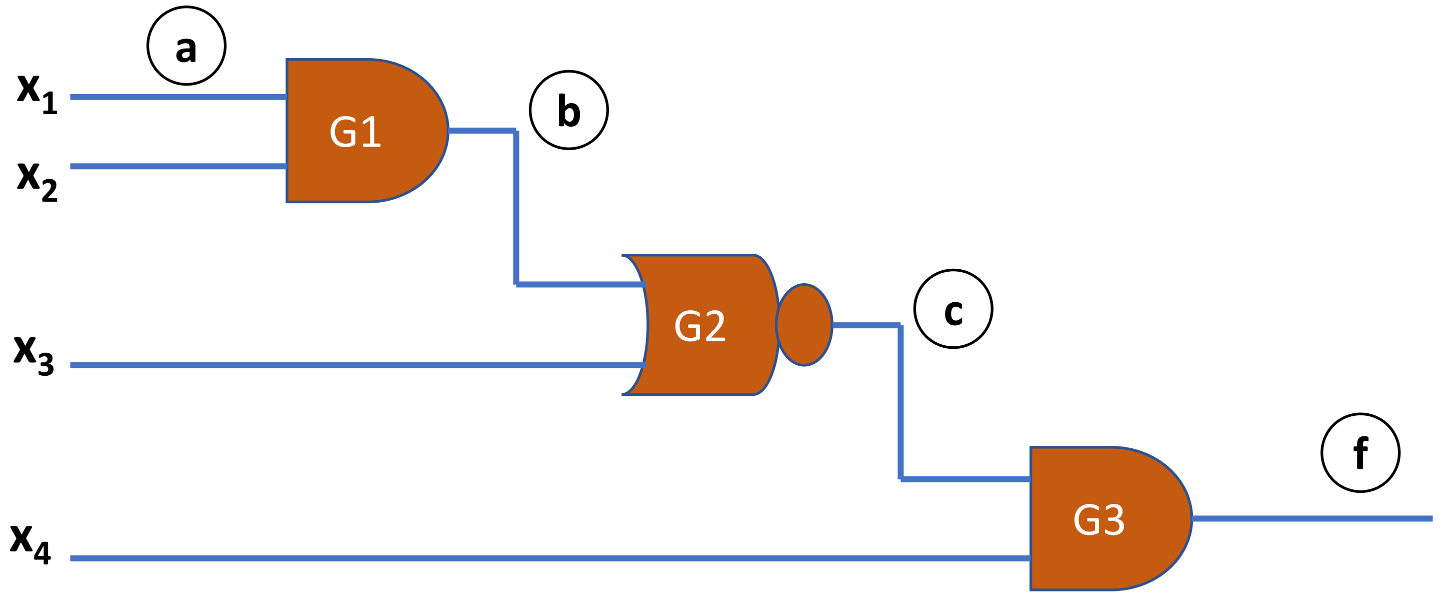
Path Sensitization

- Purpose is to sensitize the path so that inputs can help observe effects of SA0 or SA1 faults at the outputs
- In multi-level circuits, one set of test vector can act as test for faults in several paths

Three Steps in Path Sensitization Method

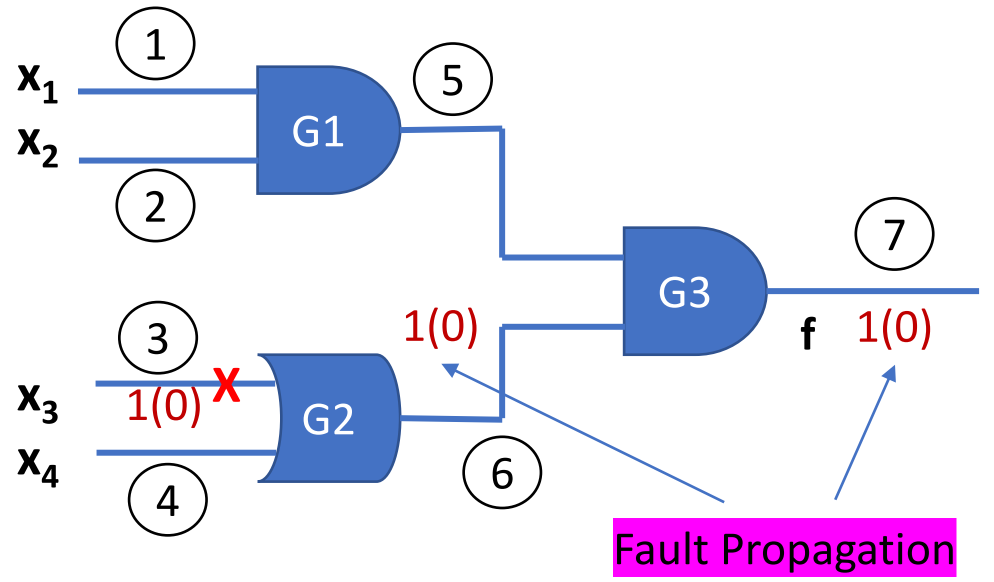
- **Fault Excitation:** Which vector to be induced to detect the suspected SA0 or SA1 fault at the suspicious path
- **Fault Propagation:** Identify path/s through which fault can be propagated to the observable output
- **Back tracking:** Move back from output towards all inputs and assign appropriate test values

Path Sensitization – how to



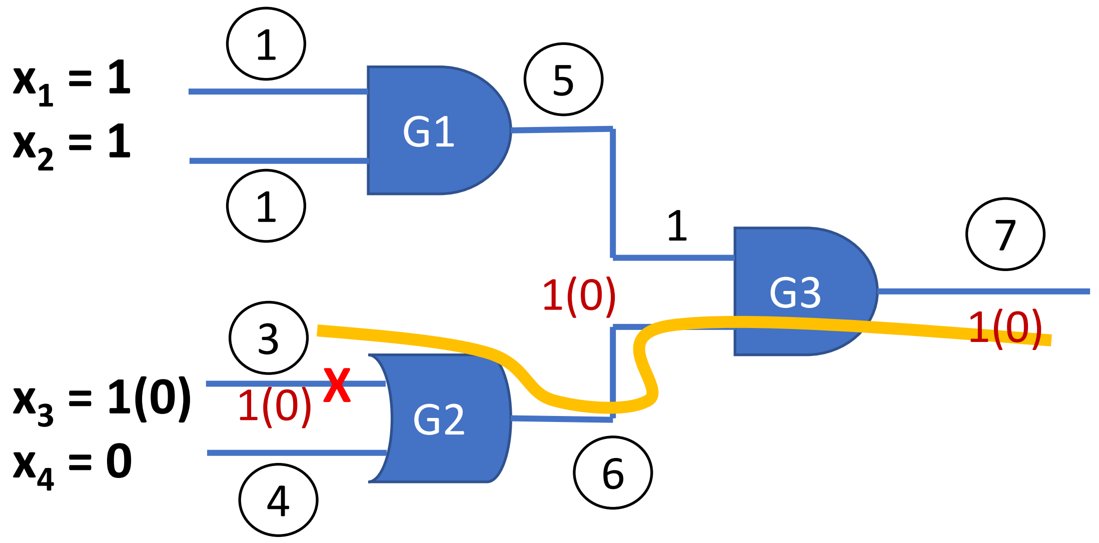
To sensitize a path through an input of AND gate or NAND gate, all other inputs must be set to ‘1’
To sensitize a path through an input of OR gate or NOR gate, all other inputs must be set to ‘0’

Path Sensitization – Example 1



Purpose: To detect SA0 fault at wire 3 connected to input of OR gate
Input x_3 is selected opposite to Stuck-At fault (eg. SA0), written as **x_3=1(0)**
This is fault generation or excitation

Test Vectors for Example 1



Sensitized path = 3 → 6→ 7

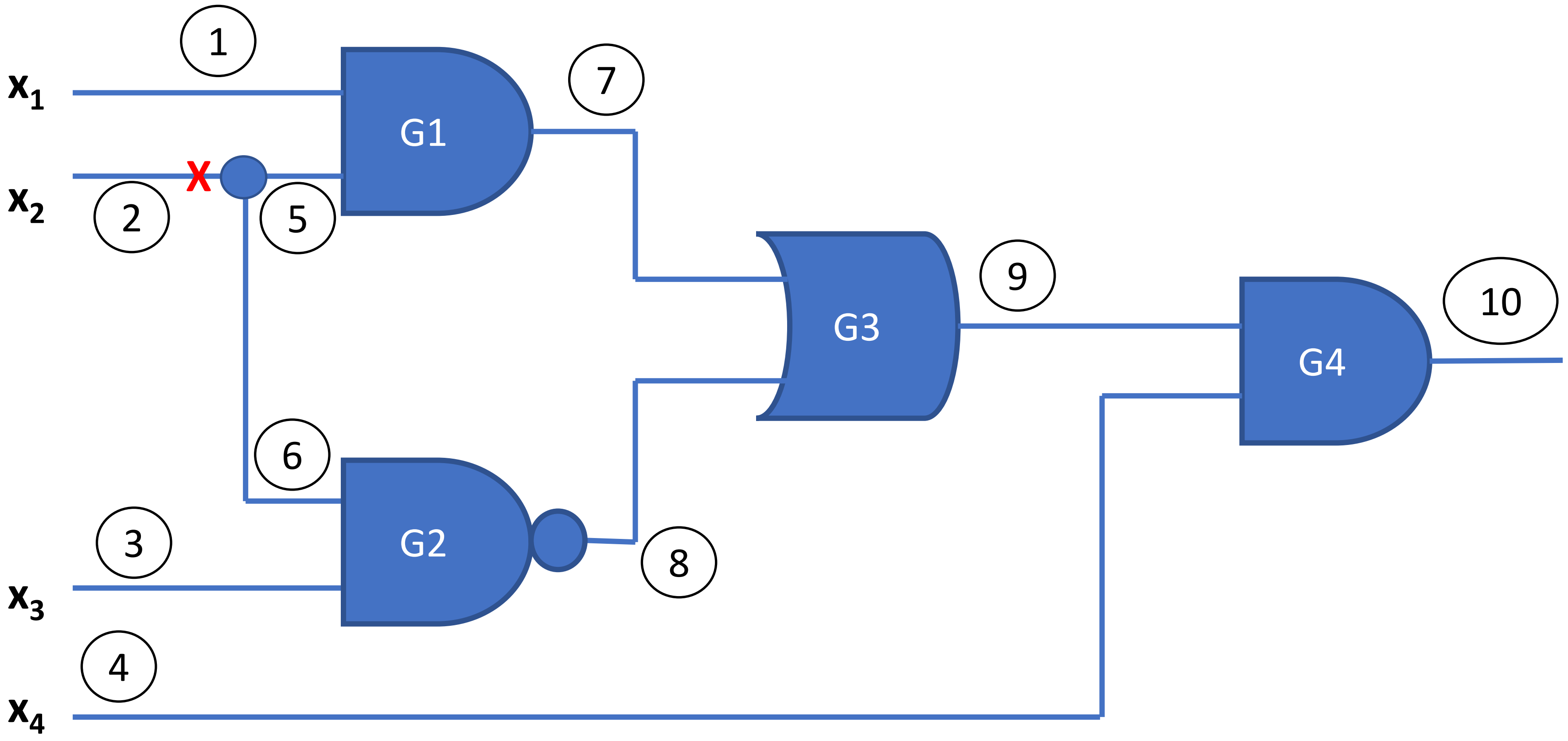
Back tracing reveals inputs to all gates to ensure fault propagation

Required test vector to detect SA0 at wire 3 is **“1110”**

Path Sensitization – Example 2

To detect SA1 fault at wire 2

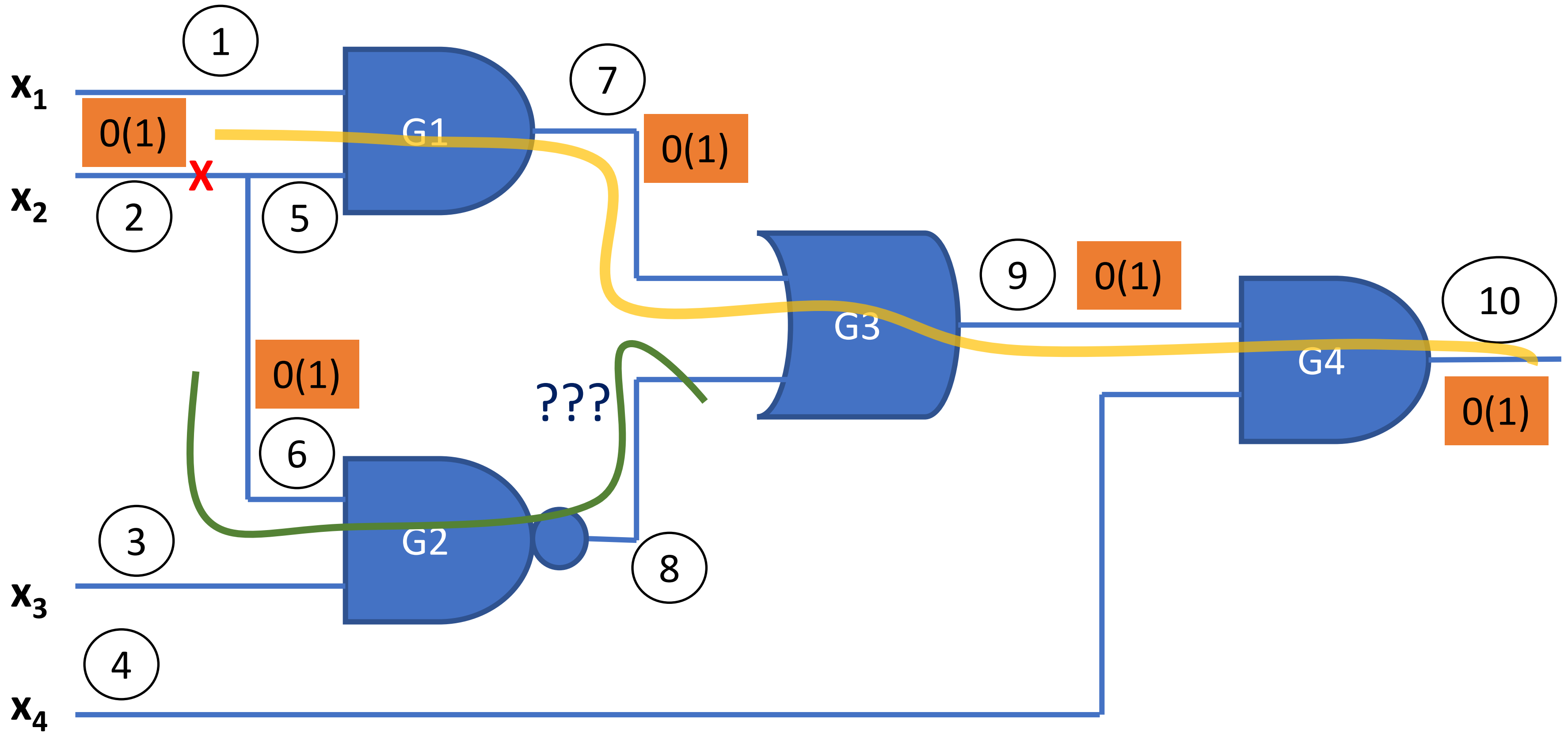
Two possible paths can be excited



continued

Detect SA1 at path 2

Fault propagation by supplying input x_2=0(1)

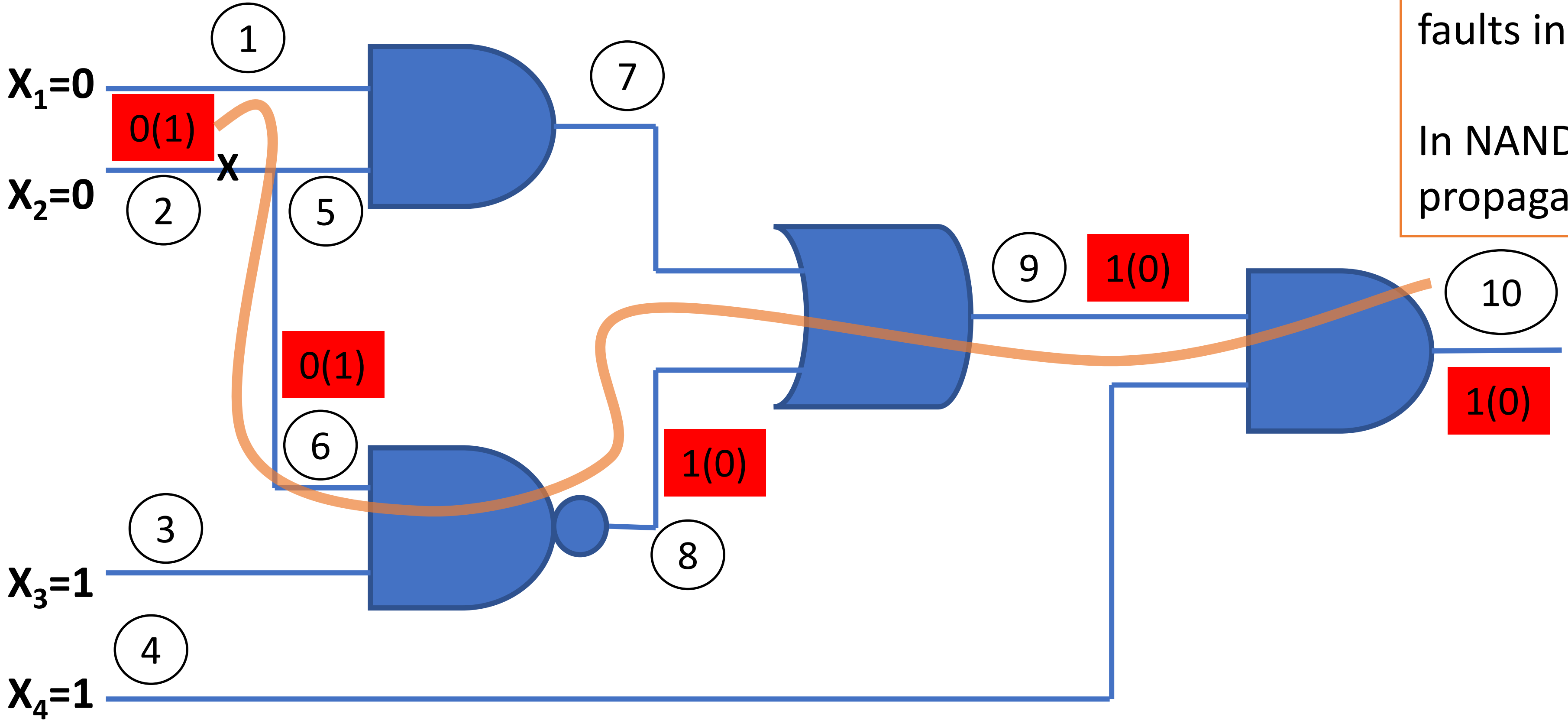


Case 1:
Back tracking reveals:
First Selected path = 2→7 → 9 → 10

But path 8=1 due to path 2 input x_2
This is not correct to have '1' at Path 9. Hence '0' cannot be justified at line 8 and line 2 simultaneously

This situation is 'Inconsistent' hence Some other path is thus required

Continued – final test vectors



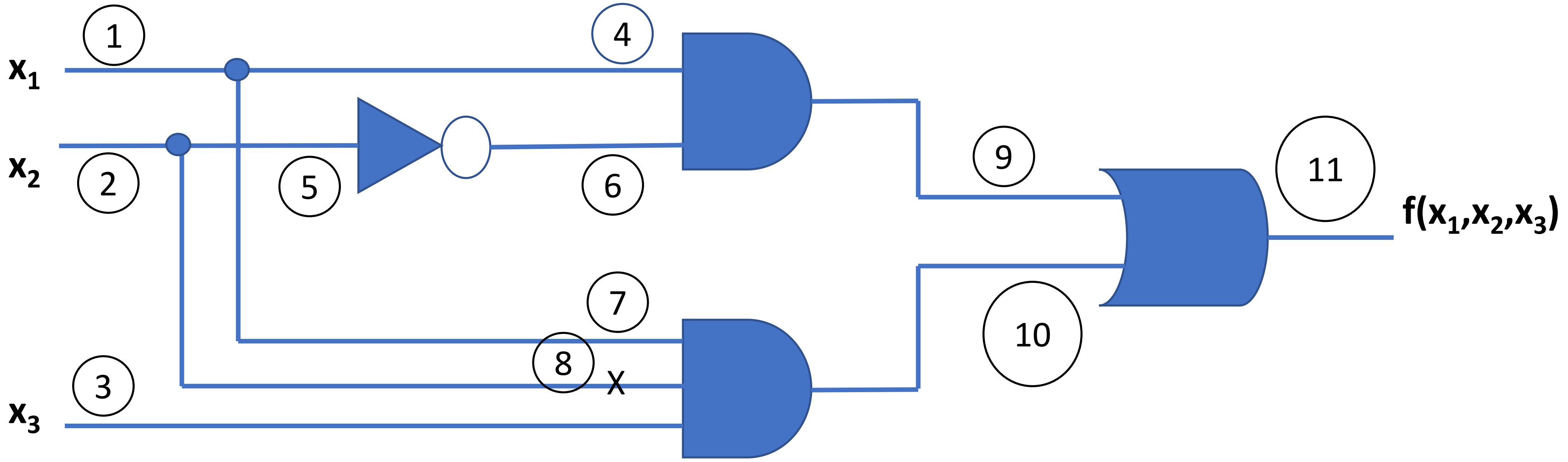
Selected path = 2 → 6 → 8 → 9 → 10

Test Vector = "0011"

This test vector can also reveal other faults in wires 6, 8 and 9

In NAND and NOR, reverse fault is propagated

Untestable Fault



Look at SA1 fault on path 8

This fault cannot be distinguished (sensitized) by changing inputs x1 to x3

Mathematically:

An untestable fault exists when $f^{8/1} \oplus f^{8} = 0$

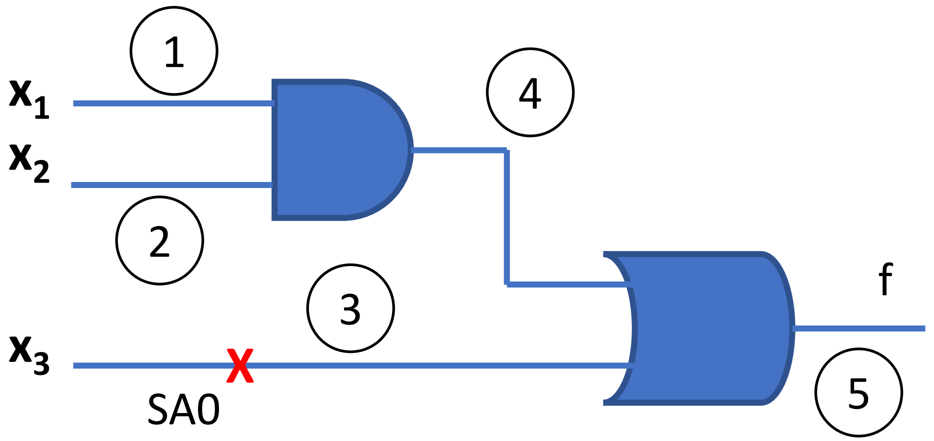
This condition means it is not possible to test this path

EXOR Method for Fault Test Generation

Define faulty and fault-free circuit as Boolean expressions
The Exclusive OR of the two functions should be ZERO if they are same i.e. NO FAULT
The Exclusive OR of the two functions should be ONE if there is fault, means different fault propagation

Premise:
Faulty Circuit must produce a different response from a fault-free circuit

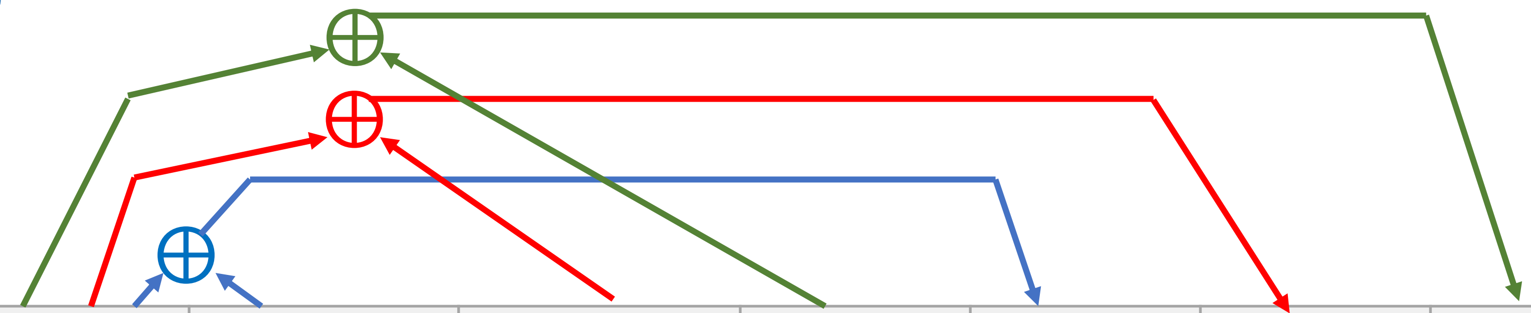
Example SA0 fault on wire 3, input to OR gate in circuit below:



EXOR Method – The steps involved

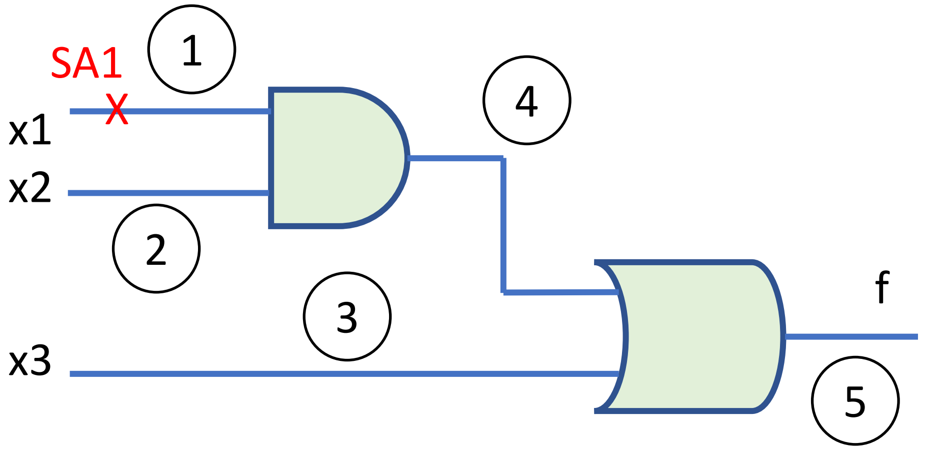
- Step 1: Construct truth table of fault-free ‘f’ and faulty ‘ $f^{p/d}$ ’
- Step 2: Compute $f \oplus f^{p/d}$ for each row of the truth table
- Step 3: Tests (input vectors) for fault p/d are indicated by the ones in the columns corresponding to $f \oplus f^{p/d}$
- Step 4: By expressing f and $f^{p/d}$ in Boolean algebra, an expression that gives all tests for p/d can be determined

Draw Fault Table – shows a set of faults and a set of inputs



Tests (inputs)			f output	$f^{1/0}$	$f^{2/1}$	$f^{3/0}$	$f \oplus f^{1/0}$	$f \oplus f^{2/1}$	$f \oplus f^{3/0}$
x_1	x_2	x_3		SA0 at 1	SA1 at 2	SA0 at 3			
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	1
0	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	1
1	0	0	0	0	1	0	0	1	0
1	0	1	1	1	1	0	0	0	1
1	1	0	1	0	1	1	1	0	0
1	1	1	1	1	1	1	0	0	0

EXOR method using Boolean Algebra



$F^{1/0} = f \oplus f^{1/0}$

$= (x1.x2 + x3) \oplus (x3)$

$= x1.x2.x3'$

$\Longrightarrow$ test vector = 110

Similarly, test for 3/0 is:

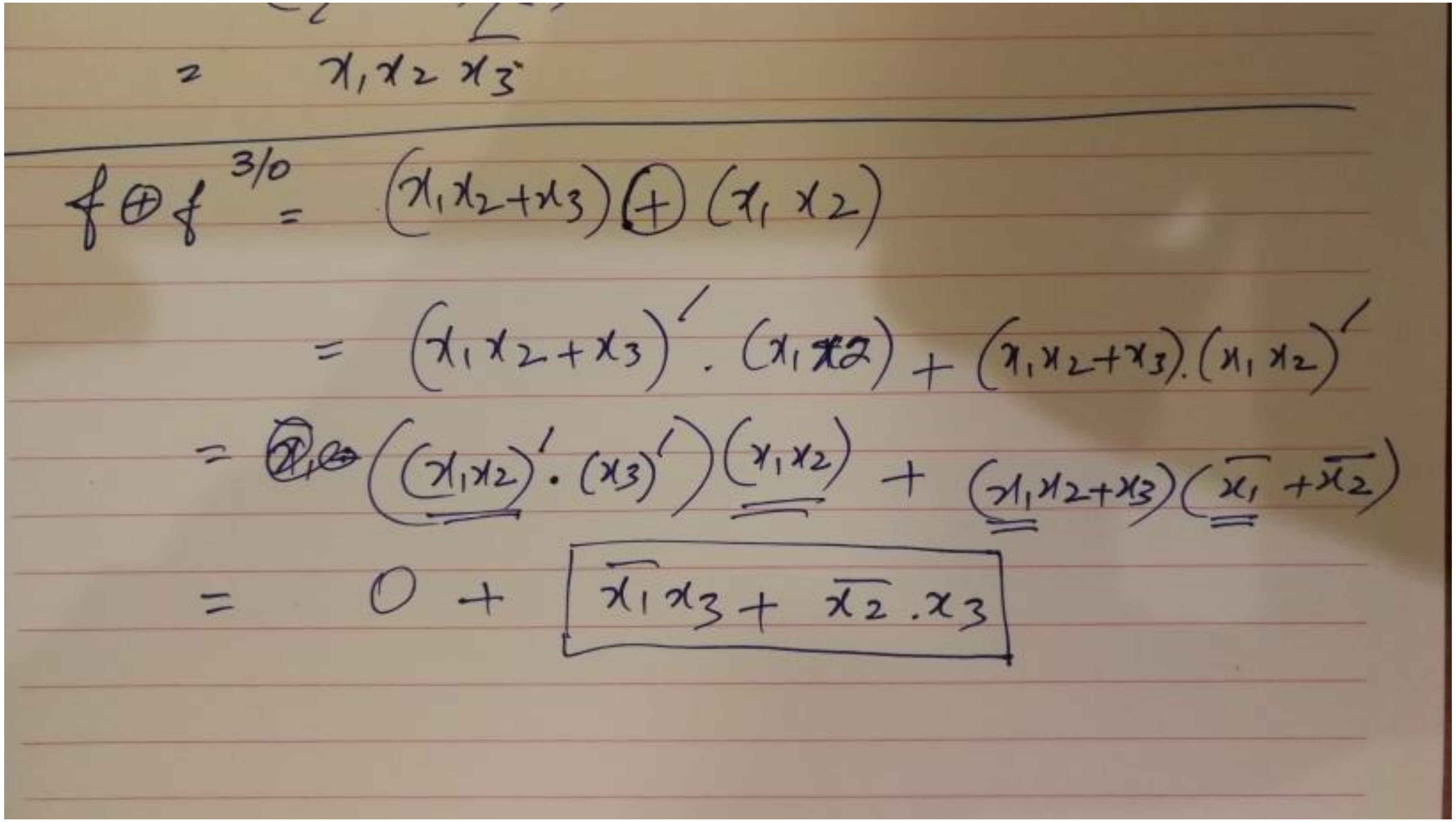
$F^{3/0} = (x1.x2 + x3) \oplus (x1.x2)$

$= (x1' + x2').x3$

$= (x1'.x3 + x2'.x3)$

Thus tests are: 001, 011, 101

Solution



$= x_1 x_2 x_3$

$f \oplus f^{3/0} = (x_1 x_2 + x_3) \oplus (x_1 x_2)$

$= (x_1 x_2 + x_3)' \cdot (x_1 \cancel{x_2}) + (x_1 x_2 + x_3) \cdot (x_1 x_2)'$

$= \cancel{x_1}\,\big((x_1 x_2)' \cdot (x_3)'\big)(x_1 x_2) + (x_1 x_2 + x_3)(\bar{x}_1 + \bar{x}_2)$

$= 0 + \boxed{\bar{x}_1 x_3 + \bar{x}_2 . x_3}$