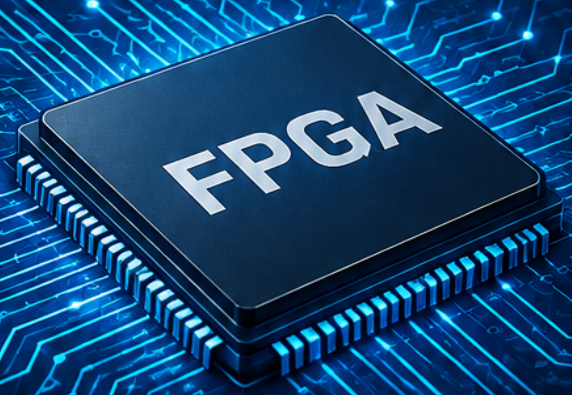
FPGA PROGRAMMING WITH VERILOG

A COMPLETE GUIDE
FROM DIGITAL LOGIC TO REAL-WORLD DESIGNS



DIGITAL LOGIC
Master the fundamentals that power every design

VERILOG MASTERY
Write clean, synthesizable code with confidence

INSIDE THE FPGA
Understand synthesis, architecture, and timing

REAL-WORLD DESIGNS
Build complete projects you can use and extend

BUILD PRACTICAL PROJECTS INCLUDING

- COMMUNICATION INTERFACES
- VIDEO DISPLAYS
- SIGNAL-PROCESSING PIPELINES
- SYSTEM-ON-CHIP ARCHITECTURES
- PRODUCTION-QUALITY CODE & TESTBENCHES

MATTHEW REYNOLDS

FPGA Programming with Verilog

A Complete Guide from Digital Logic to Real-World Designs

Steve Publications

This book is available at https://leanpub.com/fpgaprogrammingwithverilog

This version was published on 2026-08-21

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

A Complete Guide from Digital Logic to Real-World Designs

This book takes you from zero FPGA experience to designing sophisticated hardware systems in Verilog. You will learn the fundamental principles of digital logic, master the Verilog language, understand what your code becomes inside an FPGA, and build complete working projects including communication interfaces, video displays, signal-processing pipelines, and system-on-chip architectures. Every concept is explained with production-quality code examples, detailed testbenches, and practical guidance drawn from professional FPGA engineering practice.

Introduction: Hardware Is Different

You know how to program a computer. You write code in some language, the compiler translates it into instructions, and the processor executes those instructions one after another. The CPU fetches an instruction, decodes it, performs the operation, stores the result, then moves on to the next instruction. Time is linear. Execution is sequential.

Now imagine a different world. Instead of telling a processor what to do step by step, you describe a physical circuit that exists all at once. Every gate operates continuously and in parallel. There is no "next step" because everything happens simultaneously. The clock ticks, and every flip-flop updates its value at exactly the same instant. Data flows through combinational logic like water through pipes, following paths determined by the structure you designed.

This is hardware description. This is what Verilog does. And this is why learning FPGA design requires a fundamental shift in thinking.

What Is an FPGA?

An FPGA is a Field-Programmable Gate Array: a reconfigurable integrated circuit that lets you build custom digital logic after the chip has been manufactured. Inside the silicon, thousands of identical logic blocks are arranged in a grid and connected by programmable wiring channels. When you "program" an FPGA, you are not loading software onto it. You are configuring switches inside the chip to wire together logic elements into the specific circuit you designed.

The first commercially viable FPGA appeared in 1985 when Xilinx released the XC2064 with sixty-four configurable logic blocks [1]. Today's large FPGAs contain over fifty million equivalent gates, built-in processors, gigabytes of memory controllers, and high-speed transceivers capable of tens of gigabits per second. The market grew from fourteen million dollars in 1987 to an estimated nine billion dollars by 2020 [2].

Two companies have historically dominated: Xilinx (acquired by AMD in 2022 for approximately fifty billion dollars) and Altera (acquired by Intel in

2015, then spun out as an independent company again in February 2024). Other significant vendors include Lattice Semiconductor, Microchip (through its acquisition of Microsemi and Actel), Gowin, and Efinix [2].

Compared to a custom ASIC chip, an FPGA is slower, less power-efficient, and more expensive per function. A 2006 study found FPGAs required roughly forty times the area and twelve times the dynamic power of equivalent ASIC implementations, running at about one-third the speed [3]. But FPGAs offer something ASICs cannot: you can change the design after manufacturing. This makes them indispensable for prototyping, low-volume production, research and development, hardware acceleration, and any application where flexibility matters more than raw efficiency.

Why Verilog?

Verilog is one of two dominant hardware description languages (HDLs), the other being VHDL. Both are used professionally; neither is objectively superior. This book uses Verilog because:

- It has a C-like syntax that feels familiar to programmers, making the initial learning curve gentler.
- It is widely used in industry, particularly in North America and in companies working on processors, networking, and high-speed interfaces.
- Open-source toolchains like Yosys have excellent Verilog support, giving you freedom from proprietary vendor software if desired.

SystemVerilog extends Verilog with additional features for both design and verification. Where SystemVerilog constructs materially improve clarity or safety, this book introduces them alongside the base Verilog equivalents. The primary language throughout remains standard Verilog.

What This Book Will Do

This book is structured to build your understanding progressively:

Part I establishes the digital logic foundations you need before touching any code. You will learn how binary numbers work, how Boolean algebra maps

to physical gates, and how sequential circuits store state using flip-flops and clocks.

Part II teaches Verilog itself. You will write modules, describe combinational and sequential logic, understand the critical difference between blocking and nonblocking assignments, and most importantly, learn what hardware your code actually infers.

Part III opens the black box of FPGA internals so you understand what your design compiles into: look-up tables, flip-flops, routing fabric, block RAM, DSP slices, and clock management resources. You will also walk through the complete development flow from project creation to bitstream generation.

Part IV covers core design patterns that appear in virtually every FPGA system: finite-state machines for control logic, FIFOs and memories for data buffering, and techniques for managing timing, resets, and communication between different clock domains.

Part V builds practical hardware peripherals from scratch. You will implement UART, SPI, and I2C serial interfaces; drive seven-segment displays and VGA monitors; and apply digital signal processing concepts including fixed-point arithmetic and FIR filters.

Part VI brings everything together at a professional level. You will learn to integrate modules into complete systems, work with soft processors like MicroBlaze and Nios II, optimize designs for area and speed, debug hardware issues systematically, and adopt coding practices that serve you in real engineering environments.

How to Read This Book

Read sequentially. Each chapter assumes the concepts from earlier chapters. If you skip ahead to the Verilog code before understanding digital logic fundamentals, you will see patterns without understanding why they exist.

Study the code examples carefully. They are not toy snippets designed to illustrate a single feature in isolation. They are complete, synthesizable modules written to production standards with proper reset handling, parameterization, and documentation. Read them as if someone else wrote them and you need to understand what hardware they describe.

Pay attention to what is synthesized versus what is simulation-only. One of the most common beginner mistakes is writing Verilog that simulates correctly but either fails to synthesize or synthesizes into unintended hardware. This book flags these distinctions explicitly.

Try things on real hardware if you can. Nothing teaches FPGA design like seeing your code actually control LEDs, communicate with sensors, or generate video. Even a low-cost board like the Digilent Basys 3 (Xilinx Artix-7) or Terasic DE10-Lite (Intel Cyclone V) will let you verify everything in this book. If proprietary tools are a barrier, open-source toolchains like Yosys and Nextpnr work well with boards like the Lattice iCE40-based iCEstick [4].

A Note on Conventions

Throughout this book:

- Verilog code appears in monospace font blocks with syntax highlighting where possible.
- Hardware terms like “flip-flop,” “LUT,” and “DSP slice” are defined before first use.
- Design recommendations based on industry practice are marked as such; they reflect consensus guidance from vendor documentation, professional coding standards, and established engineering methodology [5].
- All code examples target synthesizable RTL unless explicitly noted as simulation or testbench code.

Let us begin.

Chapter 1: Bits, Numbers, and the Physical Layer

Before writing a single line of Verilog, you need to understand what digital circuits actually manipulate. This is not abstract mathematics. A "bit" in an FPGA is a voltage on a wire. A "number" is a pattern of voltages across multiple wires. Understanding the physical reality behind the abstraction is what separates someone who can copy code from someone who can design hardware.

What Is a Digital Signal?

A digital signal takes one of two valid values at any moment: logic 0 (LOW) or logic 1 (HIGH). In practice, these are voltage ranges, not exact voltages. A wire carrying approximately zero volts represents logic 0; a wire carrying approximately the supply voltage represents logic 1.

The beauty of digital design is that you do not need to care about the exact voltage. You only need to know which range it falls into. This abstraction is what makes complex systems possible: you reason in terms of zeros and ones, and the physical layer handles the voltages.

But the abstraction is not free. The physical layer imposes constraints that every FPGA designer must respect:

- Voltages must stay within defined ranges for signals to be recognized correctly.
- Signals take finite time to propagate through gates.
- Noise can corrupt signals if margins are too tight.
- Not all logic families are compatible with each other.

Binary, Hexadecimal, and Number Systems

Computers use binary (base-2) representation because it maps naturally to two-state physical systems: voltage high or low, switch on or off, magnetized north or south. Each position in a binary number represents a power of two.

An 8-bit binary number b7b6b5b4b3b2b1b0 has value:

Value = $b7\times2^{7 + b6\times2}6 + b5\times2^{5 + b4\times2}4 + b3\times2^{3 + b2\times2}2 + b1\times2^{1 + b0\times2}0$

For example, the binary pattern 01101011 equals: 0×128 + 1×64 + 1×32 + 0×16 + 1×8 + 0×4 + 1×2 + 1×1 = 107 in decimal.

Eight bits can represent values from 0 to 255. In general, N bits represent 2^N distinct values. Common widths:

- 8 bits (byte): 0 to 255
- 16 bits (half-word): 0 to 65535
- 32 bits (word): 0 to 4294967295
- 64 bits (double word): 0 to 18446744073709551615

Binary numbers are tedious to read and write. Engineers use hexadecimal (base-16) as a compact shorthand. Each hex digit represents four binary bits:

Hex	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101

Hex	Binary
E	1110
F	1111

The binary number 01101011 becomes 6B in hex. The pattern is immediate: group bits into fours from the right (0110 = 6, 1011 = B). Hexadecimal dominates FPGA documentation, tool output, and debug sessions. You will see values written as 0x6B or 6Bh depending on convention.

Signed Representations: Sign-Magnitude, Ones Complement, Twos Complement

Unsigned binary is straightforward: all bits contribute to the magnitude. But what about negative numbers? Three historical approaches exist; only one survives in modern hardware.

Sign-Magnitude

The most significant bit indicates sign (0 = positive, 1 = negative); remaining bits encode magnitude directly. For example, in a 4-bit system: +5 is 0101 and -5 is 1101.

Problems: two representations of zero (+0 = 0000, -0 = 1000). Arithmetic requires special handling for signs. Rarely used except in floating-point mantissa representation.

Ones Complement

Negative numbers are formed by inverting all bits of the positive value. For example, +5 in 4-bit is 0101; -5 becomes 1010.

Problems: still has two zeros (+0 = 0000, -0 = 1111). Addition requires an "end-around carry" step when the result generates a carry out of the sign bit. Used in some early computers but abandoned for practical reasons.

Twos Complement

Negative numbers are formed by inverting all bits and adding one. Equivalently, the value of an N-bit twos complement number is:

$$\text{Value} = -b_{N-1} \times 2^{(N-1)} + b_{N-2} \times 2^{(N-2)} + \ldots + b_1 \times 2^1 + b_0 \times 2^0$$

The sign bit has negative weight. For example, in 4-bit twos complement:

- 0101 = −0×8 + 1×4 + 0×2 + 1×1 = +5
- 1011 = −1×8 + 0×4 + 1×2 + 1×1 = −5

To negate a number: invert all bits, add one. The negative of 0101 (+5) is computed as: Invert: 1010, Add one: 1011 (−5). Verify: 1011 = −8 + 2 + 1 = −5. Correct.

Twos complement has three critical advantages [6]:

1. Only one representation of zero (all bits zero).
2. Addition and subtraction use the exact same hardware as unsigned arithmetic. The carry out of the sign bit is simply discarded.
3. The range is asymmetric: an N-bit twos complement number represents values from $-2^{(N-1)}$ to $2^{(N-1)}-1$. For 8 bits: −128 to +127. Note that −128 has no positive counterpart within 8 bits (its negation would be +128, which requires 9 bits).

This asymmetry matters. If you try to compute the negative of the most-negative number in twos complement, overflow occurs and you get the same number back. In 8-bit: −(−128) = −128. This is a real bug source in poorly designed arithmetic circuits.

Twos complement is universal in modern processors, FPGAs, and digital systems. When this book discusses signed numbers without qualification, it means twos complement.

Boolean Values and Logic Levels

In hardware, the values 0 and 1 are not just numeric; they are logical. A signal either satisfies a condition (true, HIGH, logic 1) or does not (false, LOW, logic 0). Boolean algebra operates on these binary values using three fundamental operations:

- AND: Output is 1 only if all inputs are 1
- OR: Output is 1 if any input is 1
- NOT: Output is the opposite of the input

These three operations are sufficient to build any digital function. In Verilog, they appear as operators (& for bitwise AND, | for bitwise OR, ~ for bitwise NOT), but they also correspond directly to physical gates inside the FPGA.

Verilog uses a four-value logic system rather than pure binary:

- 0: Strong logic zero (driven LOW)
- 1: Strong logic one (driven HIGH)
- z: High impedance (floating, not driven by any source)
- x: Unknown or conflicting (multiple drivers disagree, or value is uninitialized in simulation)

The values z and x are crucial for correct design. A high-impedance state occurs on bidirectional buses when a device is not driving the line. An unknown state indicates either a simulation condition where the value has not been determined or, worse, a hardware conflict where two outputs drive the same net to different values.

In synthesis, x and z are treated differently:

- z maps to tri-state buffers on I/O ports (where allowed) or is flagged as an error internally depending on context.
- x in initialization indicates "do not care about this initial value," which synthesis tools may optimize aggressively. However, relying on x propagation for design correctness is dangerous and non-portable.

Voltage Thresholds, Noise Margins, and Real Hardware

The abstraction of "0" and "1" rests on precise voltage specifications. Different logic families define these differently. Understanding the distinction matters when interfacing an FPGA to external devices.

TTL Logic Levels

Transistor-Transistor Logic (TTL) was the dominant family before CMOS. Standard 5V TTL defines:

- Output LOW (VOL): 0 to 0.5V guaranteed
- Output HIGH (VOH): 2.7V to 5V guaranteed
- Input LOW threshold (VIL): 0 to 0.8V recognized as LOW
- Input HIGH threshold (VIH): 2.0V to 5V recognized as HIGH

The gap between output and input ranges is the noise margin:

- Low-level noise margin: VIL(max) - VOL(max) = 0.8V - 0.5V = 0.3V
- High-level noise margin: VOH(min) - VIH(min) = 2.7V - 2.0V = 0.7V

This means up to 0.3V of electrical noise on a LOW signal or 0.7V on a HIGH signal can be tolerated without misinterpretation [7].

CMOS Logic Levels

Complementary Metal-Oxide-Semiconductor (CMOS) uses voltage thresholds proportional to supply voltage. For standard 5V CMOS:

- Input LOW threshold: approximately 0 to 1.5V (about 30% of VDD)
- Input HIGH threshold: approximately 3.5V to 5V (about 70% of VDD)

CMOS provides significantly better noise margins than TTL at the same supply voltage and consumes far less static power because current flows primarily during switching transitions, not in steady state [8].

Modern FPGA I/O Standards

Modern FPGAs support numerous I/O standards to interface with various external devices:

- LVCMOS (Low-Voltage CMOS): Common standards at 3.3V, 2.5V, 1.8V, 1.5V, and 1.2V

- LVTTL (Low-Voltage TTL): TTL-compatible levels at 3.3V supply
- HSTL, SSTL: Differential and single-ended standards for high-speed memory interfaces
- LVDS: Low-voltage differential signaling for very high-speed data transmission

The key principle is compatibility: the driving device's output voltage range must fall within the receiving device's input recognition range. Connecting a 3.3V CMOS output to a 1.8V CMOS input without level shifting can damage the receiving device or cause unreliable operation.

FPGA pin constraints specify which I/O standard each pin uses. This configuration is baked into the bitstream and determines how internal logic levels are translated to external voltages. Getting this wrong is one of the most common causes of "my design works in simulation but not on hardware" problems.

Summary

This chapter established the physical foundation for everything that follows:

- Digital signals are voltage ranges, not mathematical abstractions.
- Binary and hexadecimal representations map directly to patterns of voltages on wires.
- Twos complement is the universal standard for signed arithmetic in digital hardware.
- Verilog's four-value logic (0, 1, z, x) models real physical conditions including floating buses and conflicts.
- Voltage thresholds and noise margins determine which devices can communicate reliably.

In the next chapter, we will build on this foundation to explore Boolean algebra and combinational logic: how simple gates combine to implement arbitrary logical functions, and how those functions map from mathematical expressions to physical circuits.

Chapter 2: Boolean Algebra and Combinational Logic

Digital circuits compute by combining signals through logic gates. Each gate implements a simple Boolean function. By connecting gates together, you build complex functions that perform arithmetic, control data flow, decode addresses, and implement protocols. Understanding how these gates work mathematically is essential for writing Verilog that produces efficient, correct hardware.

The Basic Logic Gates

Seven fundamental gates form the basis of combinational logic:

NOT (Inverter) One input, one output. Output is the opposite of the input.

- Symbol: Y = NOT A or Y = A' or Y = ~A
- Truth table: A=0 gives Y=1; A=1 gives Y=0

AND Two or more inputs. Output is 1 only if all inputs are 1.

- Symbol: Y = A AND B or Y = A·B or Y = AB
- Truth table (2-input): 00□0, 01□0, 10□0, 11□1

OR Two or more inputs. Output is 1 if any input is 1.

- Symbol: Y = A OR B or Y = A+B
- Truth table (2-input): 00□0, 01□1, 10□1, 11□1

NAND NOT-AND. Output is 0 only if all inputs are 1; otherwise 1.

- Symbol: Y = NOT(A AND B) or Y = (AB)'

- NAND is universal: any Boolean function can be built using only NAND gates.

NOR NOT-OR. Output is 1 only if all inputs are 0; otherwise 0.

- Symbol: Y = NOT(A OR B) or Y = (A+B)'
- NOR is also universal, like NAND.

XOR (Exclusive OR) Two inputs. Output is 1 if exactly one input is 1.

- Symbol: Y = A XOR B or Y = A □ B
- Truth table: 00□0, 01□1, 10□1, 11□0
- Key property: A XOR A = 0; A XOR 0 = A; A XOR 1 = NOT A

XNOR NOT-XOR. Output is 1 if both inputs are equal.

- Symbol: Y = NOT(A XOR B) or Y = A □ B
- Used for equality comparison: outputs 1 when two bits match.

In FPGA design, you rarely instantiate individual gates explicitly in Verilog. Instead, you describe the logical function and let synthesis map it to the FPGA's look-up tables (LUTs), which we will cover in Chapter 7. However, understanding these primitives is essential because they are what your code compiles into conceptually, and they form the vocabulary for reasoning about logic behavior.

Boolean Algebra Laws and Identities

Boolean algebra provides rules for manipulating logical expressions. These rules let you simplify circuits (fewer gates means less area and faster operation), transform between equivalent forms, and prove that two designs behave identically.

Fundamental Laws:

- Commutative: A+B = B+A; AB = BA
- Associative: (A+B)+C = A+(B+C); (AB)C = A(BC)
- Distributive: A(B+C) = AB+AC; A+BC = (A+B)(A+C)

Identity and Annihilation:

- A+0 = A; A·1 = A (identity elements)
- A+1 = 1; A·0 = 0 (annihilation)

Complement:

- A+A' = 1; A·A' = 0
- Double negation: (A')' = A

Idempotent:

- A+A = A; A·A = A (unlike arithmetic, adding a value to itself does not double it)

Absorption:

- A+AB = A; A(A+B) = A

De Morgan's Laws (critically important):

- (AB)' = A' + B' (NOT of AND equals OR of NOTs)
- (A+B)' = A' · B' (NOT of OR equals AND of NOTs)

De Morgan's laws are used constantly in hardware design. They let you convert between NAND-based and NOR-based implementations, which matters when optimizing for specific gate types or understanding timing behavior. For example, a NAND gate with inverted inputs behaves like a NOR gate: ((A')(B'))' = A+B.

Truth Tables and Canonical Forms

A truth table exhaustively lists the output of a function for every possible combination of inputs. For N inputs, there are 2^N rows. Truth tables completely specify combinational functions and serve as the starting point for synthesis from behavioral descriptions.

Consider a function F(A,B,C) that outputs 1 when at least two inputs are 1:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

From a truth table, you can derive two canonical (standard) forms:

Sum of Products (SOP): List all input combinations where output is 1. Each combination becomes a product term (AND), and all terms are ORed together. These individual terms are called minterms.

For the example above: F = A'BC + AB'C + ABC' + ABC

Product of Sums (POS): List all input combinations where output is 0. Each combination becomes a sum term (OR), and all terms are ANDed together. These are maxterms.

F = (A+B+C)(A+B+C')(A+B'+C)(A'+B+C)

Both forms implement the same function. SOP tends to be more natural for most designs and maps directly to two-level AND-OR gate networks. Modern synthesis tools automatically choose between forms based on optimization goals, but understanding both helps you reason about logic structure.

Karnaugh Maps and Minimization

Writing a function as the full sum of minterms is correct but wasteful. Boolean algebra lets us simplify: A'BC + ABC = BC(A'+A) = BC·1 = BC. Doing this manually for complex functions is error-prone. Karnaugh maps (K-maps) provide a visual method for minimizing logic expressions.

Maurice Karnaugh introduced K-maps in 1953 as an improvement on Edward Veitch's earlier chart [9]. They arrange truth table values in a grid where adjacent cells differ by exactly one input variable (Gray code ordering). Groups of adjacent 1s can be combined into simpler terms.

For our three-variable example:

```
  BC
00  01  11  10
```

A=0 0 0 1 0 A=1 0 1 1 1

Grouping the four 1s in a rectangle covering cells (0,11), (1,11), (1,01), and (1,10):

- The group of two at column 11 spans both A values: term is BC
- The group of two at row A=1, columns 01 and 11: term is AB
- The group of two at row A=1, columns 11 and 10: term is AC

Wait, we can do better. Looking again:

- Group the four cells forming an L-shape cannot be done as one rectangle. Instead:
- Group (A=1,B=1) covers cells (1,11) and (1,10): term AB
- Group (A=1,C=1) covers cells (1,01) and (1,11): term AC
- Group (B=1,C=1) covers cells (0,11) and (1,11): term BC

But cell (1,11) is covered three times. That is allowed; overlapping groups are fine. The minimal SOP is: F = AB + AC + BC

This matches our intuition: "at least two inputs are 1" means any pair being 1 satisfies the condition. Three terms of two literals each, compared to the original four terms of three literals each.

K-maps work well for up to four or five variables. Beyond that, they become unwieldy and computer algorithms (Quine-McCluskey, Espresso) take over. Modern synthesis tools use sophisticated minimization algorithms automatically; you rarely draw K-maps in professional design. But understanding the concept helps you predict how your logic will be optimized and recognize when a function can be simplified.

Don't Care Conditions: Sometimes certain input combinations never occur or their output does not matter. These are marked as "don't care" (X or d) on the K-map and can be treated as either 0 or 1, whichever helps create larger groups. For example, in BCD-to-seven-segment decoding, input values 1010 through 1111 never occur for valid BCD digits, so their outputs are don't cares that enable simpler logic.

Combinational Hazards and Glitches

A combinational hazard is a transient glitch on an output caused by different propagation delays along different paths within the circuit. Even if your Boolean expression is algebraically correct, the physical implementation may produce brief incorrect values during input transitions.

Consider the function F = A·B + A'·C implemented with AND-OR gates. When B=C=1 and A transitions from 0 to 1:

- The term A·B changes from 0 to 1 (after the delay through the first AND gate)
- The term A'·C changes from 1 to 0 (after the delay through the NOT gate plus second AND gate)

If the NOT gate is slower than the direct path, there is a brief moment when both terms are 0 before A·B rises. The output F momentarily glitches to 0 even though it should remain at 1 throughout. This is a static-1 hazard (output should stay at 1 but briefly drops).

Hazards fall into categories:

- Static-1: Output should remain 1 but briefly goes to 0
- Static-0: Output should remain 0 but briefly goes to 1
- Dynamic: Output transitions multiple times when it should transition once

Whether hazards matter depends on what the output drives:

- If the output feeds a flip-flop sampled by a clock, hazards that resolve before the clock edge are harmless. The flip-flop samples only the stable value.
- If the output feeds another combinational block, hazards can propagate and amplify.
- If the output controls asynchronous behavior (like enabling a device), hazards can cause real malfunctions.

To eliminate static hazards in SOP implementations, add consensus terms that cover transitions between adjacent minterms. For F = A·B + A'·C, adding the term B·C covers the transition when both B and C are 1: F_hazard_free = A·B + A'·C + B·C

This extra term does not change the steady-state function but ensures that during the A transition, at least one product term remains high.

In modern FPGA design, hazards in pure combinational logic feeding registered outputs are usually ignored because:

1. The register samples only after signals stabilize (assuming proper timing).
2. Synthesis tools can add hazard-eliminating terms automatically when requested.

However, hazards become critical in asynchronous circuits and clock-domain crossing scenarios, topics we will cover later. Understanding them now prevents subtle bugs later.

Summary

This chapter covered the mathematical foundation of combinational logic:

- Seven basic gates (NOT, AND, OR, NAND, NOR, XOR, XNOR) implement all digital functions.
- Boolean algebra laws provide rules for simplifying and transforming expressions.
- Truth tables completely specify functions; canonical forms (SOP/POS) derive directly from them.
- Karnaugh maps visualize minimization for small functions; tools handle larger ones automatically.
- Hazards are transient glitches caused by unequal path delays; they matter in specific contexts and can be eliminated with consensus terms.

In the next chapter, we will introduce state into our circuits. So far everything has been combinational: outputs depend only on current inputs. Sequential circuits use memory elements to create systems where outputs depend on both current inputs and past history. This is where clocks, flip-flops, and real computation begin.

Chapter 3: Sequential Logic, Memory, and State

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Latches versus Flip-Flops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

The Clock Signal and Synchronous Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Registers and Shift Registers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Counters and Timing Sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Metastability and Setup/Hold Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Setup and Hold Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Metastability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 4: Hardware Thinking and Verilog Basics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Hardware Description versus Programming

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Your First Verilog Module

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Ports, Directions, and Data Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Port Directions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Wire versus Reg

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Vector Widths

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Number Literals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Continuous Assignment and the assign Statement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

What Synthesis Actually Means

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 5: Combinational Logic in Verilog

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Logical, Bitwise, and Reduction Operators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

The always_comb Block

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Conditional Assignment and if/else

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Case Statements and Priority Encoders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Functions for Combinational Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 6: Sequential Logic in Verilog

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

The always_ff Block and Clocking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Blocking versus Nonblocking Assignments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Registers and Register Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Counters and Timers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Reset Styles: Synchronous versus Asynchronous

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 7: Inside an FPGA: What Your Code Becomes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

The Configurable Logic Block

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Look-Up Tables and Function Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Routing Fabric and Timing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Embedded Memory Blocks and DSP Slices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Clock Management Resources and I/O Banks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 8: The Complete FPGA Development Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Setting Up Your First Project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Simulation and Testbenches

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Synthesis: From RTL to Netlist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Implementation, Place-and-Route, and Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Bitstream Generation and Board Programming

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 9: Finite-State Machines: The Heart of Control Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

What Is a Finite-State Machine?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

State Encoding Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Moore versus Mealy Machines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

The Three-Process FSM Style

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

FSM Timing, Reset, and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 10: Memories, FIFOs, and Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Inferred Combinational versus Registered Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Using Block RAM Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Read-Only Memory and Initialization Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

First-In First-Out Buffers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Memory-Mapped Address Decoding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 11: Timing, Resets, and Clock-Domain Crossing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Setup Time, Hold Time, and Critical Path

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Timing Constraints and Reports

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Reset Strategies and Power-On Initialization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Metastability and Synchronizers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Clock-Domain Crossing Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 12: Serial Communication Protocols

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Debouncing and Input Conditioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Universal Asynchronous Receiver Transmitter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Serial Peripheral Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Inter-Integrated Circuit Protocol

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 13: Display Interfaces and Video Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Seven-Segment Displays and Multiplexing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

VGA Timing and Signal Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Generating Video Pixels at 25 MHz and Beyond

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 14: Digital Signal Processing on FPGAs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Fixed-Point Number Representation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Multipliers and DSP Slices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Pipelining for Throughput

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Finite Impulse Response Filters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Signal Processing Design Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 15: System Integration and Soft Processors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Bus Protocols and Memory-Mapped I/O

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Interconnecting Multiple Modules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Soft Processor Cores

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Hardware-Accelerated Computation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Building a Complete System-on-Chip

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 16: Optimization, Debugging, and Verification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Area versus Speed Tradeoffs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Timing Closure Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

In-System Debugging Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Verification Methodologies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Common Bugs and How to Avoid Them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Chapter 17: Professional RTL Design Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Synthesis-Friendly Coding Style

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Parameterization and Reusability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Portability Across Vendor Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Version Control and Project Organization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

From Hobbyist to Professional Engineer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Conclusion: What You Now Know

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

Glossary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fpgaprogrammingwithverilog.