A

# Project Report

On

# "FPGA-BASED DIGITAL CIRCUIT APPLICATIONS USING VERILOG"

**Submitted to**

**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES**
**RK VALLEY, KADAPA**

in partial fulfillment of the requirements for the award of the Degree of

**BACHELOR OF TECHNOLOGY**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**


**Submitted by**

**Y RAJESH R200012**

**K SATHVIKA R200527**

**P KARTHIK R200036**

**Under the Guidance of**

**Mr. K HARINATH REDDY sir, Assistant Professor**

**RGUKT RK VALLEY**




**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES**
**RKVALLEY,KADAPA 516330**


**2024-2025**

# RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
# RK VALLEY,KADAPA 516330

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

## DECLARATION

We hereby declare that the project report entitled **FPGA-Based Digital Circuit Applications using Verilog** submitted to the Department of **ELECTRONICS AND COMMUNICATION ENGINEERING** in partial fulfillment of requirements for the award of the degree of **BACHELOR OF TECHNOLOGY**. This project is the result of our own effort and that it has not been submitted to any other University or Institution for the award of any degree or diploma other than specified above.

**Y. RAJESH R200012**
**K.SATHVIKA R200527**
**P. KARTHIK R200036**

# RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
# RK VALLEY,KADAPA 516330

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

## CERTIFICATE

This is to certify that the project report entitled **FPGA-based digital circuit Applications using Verilog** a bonaflde record of the project work done and submitted by

**Y. RAJESH R200012**
**K SATHVIKA R200527**
**P KARTHIK R200036**

for the partial fulfment of the requirements for the award of B.Tech. Degree in **ELECTRONICS AND COMMUNICATION ENGINEERING**, RGUKT,RK VALLEY.

**GUIDE**
**Mr K.Harinath Reddy**
**Assistant Professor**
**RGUKT,RK VALLEY**
**Kadapa-516330**

**Head Of the Department**
**Mr.Y Arun kumar Reddy**
**Assistant Professor**
**RGUKT,RK VALLEY**
**Kadapa-516330**

## ACKNOWLEDGEMENT

We are thankful to our guide **Mr K HARINATH REDDY,** for his valuable guidance and encouragement. His helping attitude and suggestions have helped us in the successful completion of the project.

We would like to express our gratefulness and sincere thanks to **Y ARUN KUMAR REDDY**, Head of the Department of ELECTRONICS AND COMMUNICATION ENGINEERING, for his kind help and encouragement during the course of our study and in the successful completion of the project work. Successful completion of any project cannot be done without proper support and encouragement. We sincerely thanks to the management for providing all the necessary facilities during the course of study.

We would like to thank our parents and my friends who have the greatest contributions in all our achievements, for the great care and blessings in making us successful in all our endeavors.

**Y. RAJESH R200012**
**K. SATHVIKA R200527**
**P KARTHIK R200036**

# FPGA-BASED DIGITAL CIRCUIT APPLICATIONS USING VERILOG

## ABSTRACT :

This project focuses on the design and implementation of some of the applications of a digital circuit using Verilog hardware description language (HDL) and its deployment on an FPGA board. The design process begins with coding the logic in Verilog, followed by functional verification using simulation tools. After successful verification, synthesis is performed to convert the high-level HDL description into a gate-level representation. The next stage involves implementation, where the synthesized design is mapped, placed, and routed onto the FPGA architecture. Once implementation is complete, a bitstream file is generated and programmed onto the FPGA board for real-time testing. The final step is observing the output on the hardware to validate the design's functionality. This project provides hands-on experience with the FPGA design flow, bridging the gap between theoretical knowledge and practical hardware implementation. The use of Verilog and FPGA enhances the efficiency, flexibility, and real-time performance of digital circuit design.

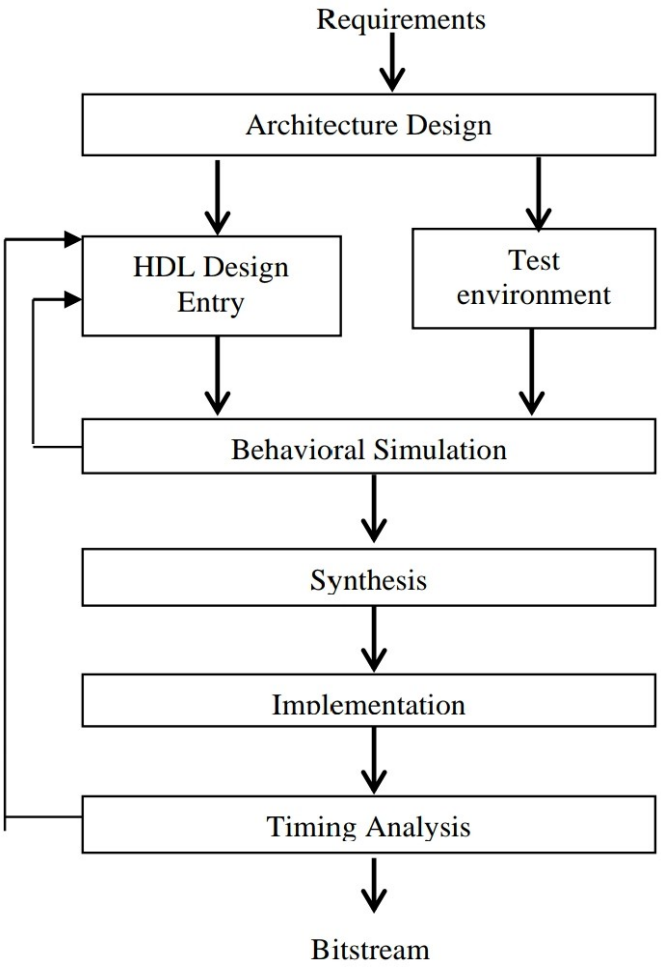



**Figure** : Schematic Showing design flow for **FPGA**

# Table of Contents

# I.Introduction

Field-Programmable Gate Arrays (FPGAs) have revolutionized digital circuit design by offering flexible, reconfigurable hardware platforms that enable rapid prototyping and the implementation of complex logic systems. Unlike traditional fixed-function integrated circuits, FPGAs allow designers to modify and optimize their designs post-fabrication, making them ideal for applications requiring adaptability, concurrency, and real-time performance.

This project aims to design and implement three fundamental digital systems—a digital clock, a stopwatch, and a scrolling display—using the Verilog Hardware Description Language (HDL) and deploy them on an FPGA development board. The **digital clock** will display real-time hours, minutes, and seconds, with user-adjustable settings. The **stopwatch** will support start, stop, and reset operations with millisecond-level precision. The **scrolling display** will present moving text or patterns across a 7-segment display, with adjustable scrolling speed.

The project follows the standard FPGA design flow: starting with Verilog coding, followed by functional simulation to verify logic correctness, synthesis to generate a gate-level netlist, implementation on a target FPGA (e.g., NEXYS A7), and finally, hardware programming for real-world testing. This process bridges theoretical digital design principles with practical application, reinforcing skills in logic development, timing analysis, and hardware-level debugging.

By integrating all three applications into a single FPGA-based system, this project showcases the efficiency, versatility, and real-time processing power of FPGAs. The hands-on experience gained lays a solid foundation for advanced FPGA-based designs in embedded systems, Internet of Things (IoT), and real-time signal processing.

# II.History of FPGA and Digital Circuits

## The Era of Discrete Logic Circuits

The foundation of digital electronics was built upon discrete logic components in the mid-20th century. Engineers constructed circuits using individual transistors, resistors, and diodes painstakingly assembled on printed circuit boards. These early systems could implement basic logic functions but suffered from significant limitations. Any design modification required physical rewiring, making iterations cumbersome and expensive. As systems grew more complex, the sheer number of components led to bulky designs with reliability challenges. Power consumption was another critical issue, with early transistor-based logic gates demanding substantial energy. The high cost of custom integrated circuit fabrication further restricted innovation to well-funded organizations, leaving smaller developers dependent on these inflexible discrete solutions.

## The Advent of Programmable Logic Devices

The 1970s brought the first wave of programmable solutions with the introduction of Programmable Array Logic (PAL) devices. These represented a significant improvement over discrete components by offering limited reprogrammability through fixed AND arrays with configurable OR planes. Manufacturers implemented them using erasable programmable read-only memory (EPROM) technology, allowing for field updates. However, these early programmable devices still faced constraints in logic capacity and input/output capabilities. Their primary applications remained relatively simple, serving as interface logic between subsystems or implementing basic state machines. While PALs and PLDs marked progress, they lacked the comprehensive reconfigurability that would later define modern FPGA technology.

## The FPGA Revolution Begins

A transformative breakthrough occurred in 1984 when Xilinx introduced the XC2064, the world's first commercial Field-Programmable Gate Array. This pioneering device featured three key innovations that would redefine digital

design: configurable logic blocks for implementing various digital functions, programmable interconnects that allowed flexible routing between components, and SRAM-based configuration that enabled field reprogrammability. Unlike previous solutions, Xilinx's architecture offered true post-manufacturing flexibility, allowing engineers to modify designs without physical changes to hardware. The higher logic density enabled more complex systems in single devices, while software-based development cycles dramatically accelerated the design process compared to traditional methods.

## Evolution of FPGA Capabilities

Over subsequent decades, FPGA technology underwent remarkable transformations. The 1990s saw exponential growth in device capacity, with new architectures supporting thousands of gates and the emergence of hardware description languages like Verilog and VHDL. Competing vendors entered the market, driving innovation and expanding applications into telecommunications and networking. The 2000s marked the system integration phase, with FPGAs incorporating embedded processors, dedicated DSP blocks, and high-speed serial transceivers. This period also saw the introduction of non-volatile configuration options using flash memory. In recent years, FPGAs have evolved into heterogeneous computing platforms, integrating ARM processor cores, employing advanced 3D packaging technologies, and providing specialized acceleration for artificial intelligence workloads.

## Modern Applications and Future Directions

Today's FPGAs serve critical roles across diverse industries, from processing radar signals in aerospace to enabling 5G communications and powering machine vision systems in industrial automation. Their unique combination of flexibility and performance has made them indispensable in scientific research, including high-energy physics and radio astronomy. Looking ahead, the FPGA industry is focusing on several key areas: tighter integration with AI/ML accelerators, modular chiplet architectures, interfaces for quantum computing systems, enhanced security features, and ultra-low-power designs for edge applications. This continuous innovation ensures FPGAs remain at the forefront of digital system design, building on their rich history to power the next generation of technological advancements.

# Nexys A7 FPGA Development Board:

### A Comprehensive Platform for Digital Design

The Nexys A7 FPGA development board from Digilent represents a perfect balance between performance and accessibility, making it an ideal choice for both educational institutions and professional engineers. Built around Xilinx's Artix-7 family FPGA, this board provides a robust hardware platform that bridges the gap between theoretical concepts and practical implementation in digital system design.

### Core Architecture and Processing Capabilities

At the heart of the Nexys A7 lies the Xilinx Artix-7 XC7A100T-1CSG324C FPGA, featuring an impressive array of programmable resources. With 101,440 logic cells organized in configurable logic blocks (CLBs), designers can implement complex digital systems ranging from simple combinational circuits to complete microprocessor architectures. The FPGA incorporates 240 DSP slices optimized for high-speed mathematical operations, enabling efficient implementation of digital signal processing algorithms, while 4,860 Kb of block RAM supports memory-intensive applications. The inclusion of six clock management tiles allows for precise clock generation and distribution, critical for synchronous design implementations.

### Memory and Storage Subsystem

The board's memory architecture has been carefully designed to support a wide range of applications. The 256MB DDR3 SDRAM operates at 800MHz, providing sufficient bandwidth for data-intensive processing tasks such as video frame buffering or network packet processing. Non-volatile configuration storage is handled by a 16MB Quad-SPI Flash memory, which maintains the FPGA bitstream between power cycles. For additional storage needs, the onboard microSD card slot enables projects requiring large datasets or file systems, making it particularly useful for applications like digital audio processing, data logging, or embedded Linux implementations.

### Expansion and Peripheral Interfaces

One of the Nexys A7's most notable features is its extensive I/O capabilities and expansion options. The board includes four standard Pmod connectors, following Digilent's widely adopted Pmod interface specification. These connectors support a vast ecosystem of peripheral modules including sensors, displays, communication interfaces, and input devices. For user interaction, the board provides 16 slide switches and 5 push buttons for digital input, complemented by 16 discrete LEDs and two RGB LEDs for visual feedback. A 4-digit seven-segment display offers basic alphanumeric output capabilities, while the 12-bit VGA port enables direct video output for graphics applications.

**Connectivity and Communication Features**

The Nexys A7 excels in communication capabilities with its integrated 10/100 Ethernet PHY, enabling network-connected applications and Internet-of-Things (IoT) prototypes. The USB 2.0 interface supports both host and device modes, allowing the board to interact with peripherals like keyboards and mice or function as a USB device. A dedicated USB-UART bridge simplifies serial communication with host computers for debugging and data transfer. These features make the board particularly suitable for projects involving embedded networking, industrial communication protocols, or USB device development.

**Development Ecosystem and Software Support**

Designing for the Nexys A7 is supported by Xilinx's Vivado Design Suite, the industry-standard development environment for Artix-7 FPGAs. The board is fully compatible with both Verilog and VHDL hardware description languages, as well as higher-level design approaches using IP integrator tools. Digilent provides comprehensive documentation, reference designs, and tutorials that significantly reduce the learning curve for new users. The availability of academic pricing and the board's use in many university curricula have made it a popular choice for digital design courses covering topics from basic logic gates to advanced computer architecture.

**Applications and Project Potential**

The versatility of the Nexys A7 enables its use across numerous application domains. In academic settings, it serves as an excellent platform for teaching digital logic, computer organization, and FPGA design principles. Professional engineers leverage its capabilities for prototyping embedded systems, developing custom accelerators, and verifying communication protocols. The board's resources are particularly well-suited for projects involving real-time signal processing, embedded control systems, computer vision implementations, and custom processor development. Its combination of FPGA programmability and fixed peripherals creates an ideal environment for exploring hardware/software co-design concepts.

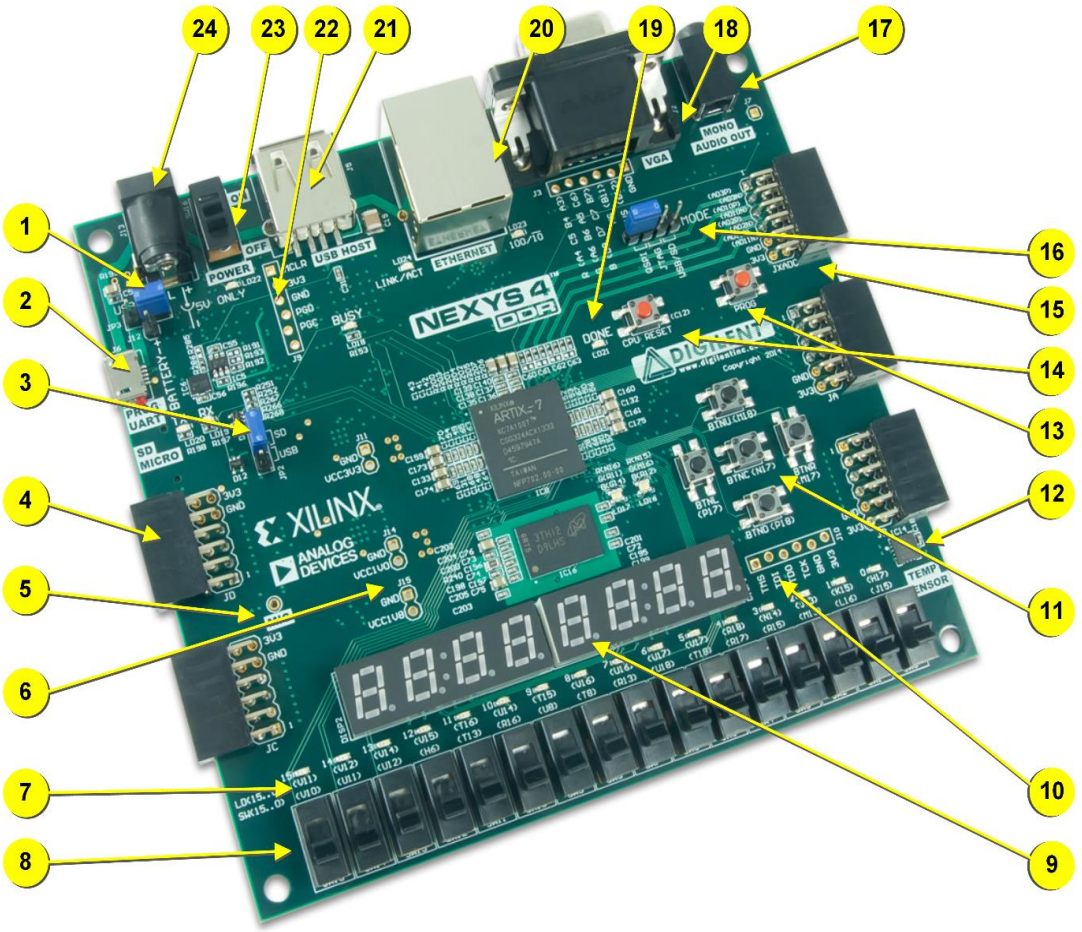



*Figure 1. Nexys4 DDR board features.*

| Callout | Component Description | Callout | Component Description |
|---|---|---|---|
| 1 | Power select jumper and battery header | 13 | FPGA configuration reset button |
| 2 | Shared UART/ JTAG USB port | 14 | CPU reset button (for soft cores) |
| 3 | External configuration jumper (SD / USB) | 15 | Analog signal Pmod port (XADC) |
| 4 | Pmod port(s) | 16 | Programming mode jumper |
| 5 | Microphone | 17 | Audio connector |
| 6 | Power supply test point(s) | 18 | VGA connector |
| 7 | LEDs (16) | 19 | FPGA programming done LED |
| 8 | Slide switches | 20 | Ethernet connector |
| 9 | Eight digit 7-seg display | 21 | USB host connector |
| 10 | JTAG port for (optional) external cable | 22 | PIC24 programming port (factory use) |
| 11 | Five pushbuttons | 23 | Power switch |
| 12 | Temperature sensor | 24 | Power jack |

A growing collection of board support IP, reference designs, and add-on boards are available on the Digilent website. See the Nexys4 DDR page at www.digilentinc.com for more information.

# III.Goals and Objectives

## Project Goals

The primary goals of this project are to design and implement three fundamental digital systems - a digital clock, a stopwatch, and a scrolling display - using Verilog HDL on an FPGA platform. Through this implementation, we aim to demonstrate the complete FPGA design flow, from initial concept and HDL coding to final hardware deployment and real-time validation. This project serves as a practical exploration of digital circuit design principles while highlighting the flexibility and capabilities of FPGA technology.

## Technical Objectives

**HDL Development**

- Design optimized Verilog modules for each application component:
- A 24-hour format digital clock with time-setting capability
- A precision stopwatch with start/stop/reset functionality and millisecond resolution
- A configurable scrolling display with adjustable speed control

**Functional Verification**

- Develop comprehensive testbenches for each module to verify:
- Correct timing behavior and display output
- Proper response to control inputs
- Boundary conditions and error handling

**FPGA Implementation**

- Complete the full hardware design flow including:
- Synthesis of Verilog code to gate-level netlist
- Place-and-route optimization for the target FPGA
- Timing analysis to ensure design constraints are met

**Hardware Validation**

- Deploy and test the design on physical FPGA hardware to:
- Verify real-time performance against specifications
- Validate display output accuracy
- Confirm reliable operation under continuous use

**Performance Optimization**

- Achieve design efficiency through:
- Resource utilization minimization
- Power consumption awareness
- Clock domain management

# IV.Problem Statement

Traditional implementations of digital systems such as clocks, stopwatches, and displays often rely on fixed-function hardware architectures, which are inherently rigid and resource-intensive to modify. While FPGAs offer a reconfigurable alternative to overcome these limitations, their flexibility introduces new challenges in achieving real-time performance, resource efficiency, and functional reliability—particularly when integrating multiple applications into a single design. This project confronts these challenges by developing a unified FPGA-based system that implements a 24-hour digital clock, a millisecond-precision stopwatch, and a user-configurable scrolling display. The core problem lies in balancing the competing demands of timing accuracy, hardware resource optimization, and seamless co-functionality across these applications, while adhering to FPGA design constraints such as clock domain management, power consumption, and I/O synchronization. Success requires rigorous verification at every design stage to ensure the system meets real-world operational requirements without compromising flexibility or scalability.

# Specifications:

**Digital Clock:**
Displays hours, minutes, and seconds in 24-hour format.
Includes reset and time-setting functionality.

**Stopwatch:**
Measures elapsed time with start, stop, and reset controls.
Displays minutes, seconds, and milliseconds.

**Scrolling Display:**
Scrolls a user-defined message horizontally on a 7-segment display or LED matrix.
Adjustable scrolling speed.

**General:**
Target FPGA: Xilinx NEXYS A7
Clock Frequency: 100 MHz (board clock).
Inputs: Push buttons for control.
Outputs: 7-segment displays .

# V.PROPOSED DESIGN:

The proposed system is architected using a modular hardware design approach in Verilog HDL, aimed at achieving functional separation and ease of debugging. The complete system consists of three main modules—Digital Clock, Stopwatch, and Scrolling Display—each developed as an independent unit and integrated within a top-level module for control and display.

## 1. Digital Clock

The Digital Clock Module is a synchronous timekeeping unit implemented in Verilog HDL. It uses a structured hierarchy of BCD counters and clock division logic to maintain and display the current time in a 24-hour format. Designed for deployment on an FPGA, the module is efficient, accurate, and well-suited for real-time digital systems.

**Timekeeping Counter Structure**

The digital clock is built upon eight 4-bit BCD counters, each representing the digits of:

- Milliseconds: ms0, ms1
- Seconds: s0, s1
- Minutes: m0, m1
- Hours: h0, h1

These counters operate in a cascading fashion, where overflow in a lower-order digit triggers an increment in the next higher-order digit. Each counter is driven by a clean, debounced, and synchronized clock signal derived from a high-frequency system clock.

**Clock Division and Pulse Generation**

A 32-bit counter serves as a clock divider to convert the 50 MHz FPGA system clock into a slower, human-readable time base—typically 1 Hz for seconds and optionally higher resolution (e.g., 100 Hz) for milliseconds. Once the counter reaches a preset terminal count (e.g., 50,000,000 for 1Hz), it resets and toggles a signal clk_seg, used to trigger BCD counter updates.
This ensures accurate, periodic time increments and enables precise real-time behavior.

**Cascading BCD Logic and Time Progression**

Each counter unit is constrained to its valid decimal range. The progression works as follows:

ms0 increments from 0 to 9, then resets and increments ms1.

ms1 resets after 9 and triggers s0.
s0 counts 0 to 9; s1 counts 0 to 5, forming a 0–59 second range.
Similarly, m0 and m1 create a 0–59 minute range.
h0 and h1 form the hour count, capping at 23 (when h1==2 && h0==3).
When the time reaches 23:59:59.99, the logic resets all counters back to 00:00:00.00, mimicking a natural 24-hour clock cycle.

**Control Inputs and Initialization**

The module supports reset and clear functionalities:

- Reset (rst): Maintains the current state (useful for pause or hold features).
- Clear (clr): Immediately resets all counters to zero, effectively restarting the clock.

These control signals enable flexible system integration and user control, essential in embedded or timed-event applications.

**Display Integration and Output Mapping**

The counter outputs are decoded using separate display modules, each translating a 4-bit BCD value into an 8-bit code for 7-segment display output. A time-multiplexing approach is employed using an anode selection signal (an) and a cycling selector counter (sel), allowing all 8 digits to be displayed with only a limited number of physical pins.
The display refresh rate is high enough to ensure flicker-free visualization due to persistence of vision.

**Design Efficiency and Reliability**

The module is fully synchronous and uses non-blocking assignments to maintain simulation accuracy and synthesis compatibility.
All internal transitions are synchronized with clk_seg, ensuring glitch-free operation.
Designed with FPGA constraints in mind, it avoids latches, metastability, and timing hazards.

**Expandability**

This digital clock module can be extended with:
AM/PM indicators for 12-hour format
Alarm functionality
Real-time clock synchronization via external I2C/SPI interfaces

## CODE:

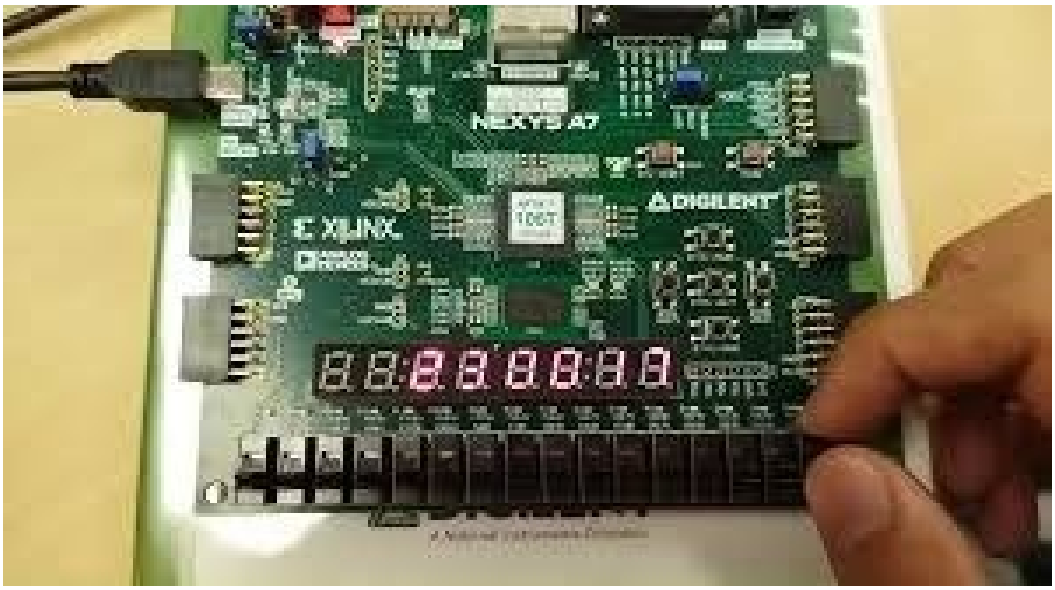
```
module digital_clk(clk,an,cn);
   input clk;                    // System clock input
   output reg [7:0] an;              // Anode control for 7-segment display
   output reg [7:0] cn;              // Cathode/segment output for 7-segment display

   reg clk_seg;                  // Divided clock for display multiplexing
   reg [31:0] count;                 // Counter for clock division

   initial begin
      clk_seg <= 0;
      count <= 0;
   end

   // Clock divider to generate slower clock for segment multiplexing
   always @(posedge clk) begin
      if(count == 100000) begin      // Approximate division for ~500 Hz
         count <= 0;
         clk_seg <= ~clk_seg;      // Toggle clk_seg at every terminal count
      end else
         count <= count + 1;
   end

   reg [2:0] sel = 3'd2;             // Selector to switch between digits

    wire [7:0] ca1, ca2, ca3, ca4, ca5, ca6, ca7, ca8;  // Segment outputs for each
digit

   // Time-division multiplexing logic for 6 digits
   always @(posedge clk_seg) begin
      case(sel)
         3'd2: begin an = 8'b1101_1111; cn = ca1; end // Hour tens
         3'd3: begin an = 8'b1110_1111; cn = ca2; end // Hour units
         3'd4: begin an = 8'b1111_0111; cn = ca3; end // Minute tens
         3'd5: begin an = 8'b1111_1011; cn = ca4; end // Minute units
         3'd6: begin an = 8'b1111_1101; cn = ca5; end // Second tens
         3'd7: begin an = 8'b1111_1110; cn = ca6; end // Second units
      endcase
      sel <= sel + 1;
   end

   wire clr;  // Clear input (not used here, but can be used in ring_count)
  // Time values from the ring counter (h1:h0 = hours, m1:m0 = minutes, s1:s0
= seconds)
```

```
    wire [3:0] h1, h0, m1, m0, s1, s0;

    ring_count utt(clk, clr, h1, h0, m1, m0, s1, s0); // Timekeeping module

    // Segment decoders for each digit
    display uut2(clk, h1, ca1); // Hour tens
    display uut3(clk, h0, ca2); // Hour units
    display uut4(clk, m1, ca3); // Minute tens
    display uut5(clk, m0, ca4); // Minute units
    display uut6(clk, s1, ca5); // Second tens
    display uut7(clk, s0, ca6); // Second units
endmodule

module ring_count(clk, rst, h1, h0, m1, m0, s1, s0);
    input clk, rst;
    output reg [3:0] h1, h0, m1, m0, s1, s0;

    reg clk_1hz;          // Slower 1Hz clock for time updates
    reg [31:0] c;         // Counter to generate 1Hz pulse from system clock

    initial begin
        clk_1hz <= 0;
        c <= 0;
    end

    // Generate 1Hz clock by dividing 50MHz system clock
    always @(posedge clk) begin
        if(c == 50000000) begin
            c <= 0;
            clk_1hz = ~clk_1hz;
        end else
            c <= c + 1;
    end

    // Time counter that updates every 1 second
    always @(posedge clk_1hz) begin
        // Reset when time reaches 23:59:59
        if(h1 == 2 && h0 == 3 && m1 == 5 && m0 == 9 && s1 == 5 && s0 == 9) begin
            h1 <= 1; h0 <= 1;
            m1 <= 5; m0 <= 9;
            s1 <= 5; s0 <= 1;
        end else begin
            s0 = s0 + 1;                // Increment second units
            if(s0 > 9) begin
```

```
            s0 <= 0;
            s1 <= s1 + 1;            // Carry to second tens

            if (s1 == 5) begin
               s0 <= 0;
               s1 <= 0;
               m0 <= m0 + 1;         // Carry to minute units

               if(m0 > 9) begin
                  s0 <= 0; s1 <= 0; m0 <= 0;
                  m1 <= m1 + 1;      // Carry to minute tens

                  if(m1 == 5) begin
                     s0 <= 0; s1 <= 0; m0 <= 0; m1 <= 0;
                     h0 <= h0 + 1;  // Carry to hour units

                     if(h0 > 9) begin
                        s0 <= 0; s1 <= 0; m0 <= 0; m1 <= 0; h0 <= 0;
                        h1 <= h1 + 1;  // Carry to hour tens

                        if(h1 == 2 && h0 == 3) begin
                           s0 <= 0; s1 <= 0; m0 <= 0; m1 <= 0; h0 <= 0; h1 <= 0;
                        end
                     end
                  end
               end
            end
         end
      end
   end
endmodule

module display(clk, value, ca);
   input clk;
   input [3:0] value;     // BCD digit to be displayed
   output reg [7:0] ca;   // Output for 7-segment pattern

   // Decode BCD digit to 7-segment display format
   always @(posedge clk) begin
      case(value)
         4'd0: ca = 8'b0000_0011;
         4'd1: ca = 8'b1001_1111;
         4'd2: ca = 8'b0010_0101;
         4'd3: ca = 8'b0000_1101;
```

```
            4'd4: ca = 8'b1001_1001;
            4'd5: ca = 8'b0100_1001;
            4'd6: ca = 8'b0100_0001;
            4'd7: ca = 8'b0001_1011;
            4'd8: ca = 8'b0000_0001;
            4'd9: ca = 8'b0000_1001;
        endcase
    end
endmodule
```

## Digital clock




## 2. Stopwatch

The Stopwatch Module is architected using a precise and deterministic synchronous design in Verilog HDL. It integrates a clock division mechanism, cascaded BCD time counters, and display multiplexing logic. The module supports key stopwatch operations: start, pause, and reset/clear, responding to user control inputs with high reliability and low latency.

**Timekeeping Counters**

At its core, the stopwatch employs eight BCD (Binary-Coded Decimal) counters, each 4 bits wide, to represent:

Milliseconds (ms0, ms1)

Seconds (s0, s1)

Minutes (m0, m1)

Hours (h0, h1)

The up_counter module updates these values synchronously on a slower derived clock (clk_seg), created via a clock divider that toggles every 500,000 system clock cycles. This clock division allows for manageable update rates compatible with human time perception.

**Cascading Rollover Logic**

Each lower digit counter increments on every rising edge of the divided clock. When a digit exceeds its decimal limit (usually 9 for BCD), it rolls over to zero and increments the next significant digit:

- When ms0 > 9, ms1 is incremented.
- When both ms0 and ms1 overflow, s0 is incremented.
- This cascade continues through s1, m0, m1, h0, and finally h1.
- The logic prevents time overflow by resetting the counter if the stopwatch reaches 23:59:59.99.

**Control Inputs**

The stopwatch responds to three control inputs:

- opt (Start/Stop): When high, the stopwatch increments counters.
- clr (Clear): Asynchronously resets all counters to zero.
- rst (Reset/hold): Maintains the current state without updating counters.

This behavior emulates a basic FSM (Finite State Machine), although it's coded as conditional logic rather than an explicitly defined FSM block. The states are:

- Idle/Pause (opt=0): Counter state held, no change.
- Running (opt=1): Counters increment on each clock tick.
- Clear: Immediate reset regardless of state.

**Time Display Integration**

The dig_stp module drives an 8-digit 7-segment display using time-division multiplexing. Each digit is enabled one at a time via the an (anode) signal, while corresponding BCD counter values are decoded into 7-segment codes (ca0 to ca7) using display modules.

A selection counter sel cycles through digits every divided clock edge (clk_seg), ensuring all digits appear illuminated simultaneously due to persistence of vision.

**Design Features and Robustness**

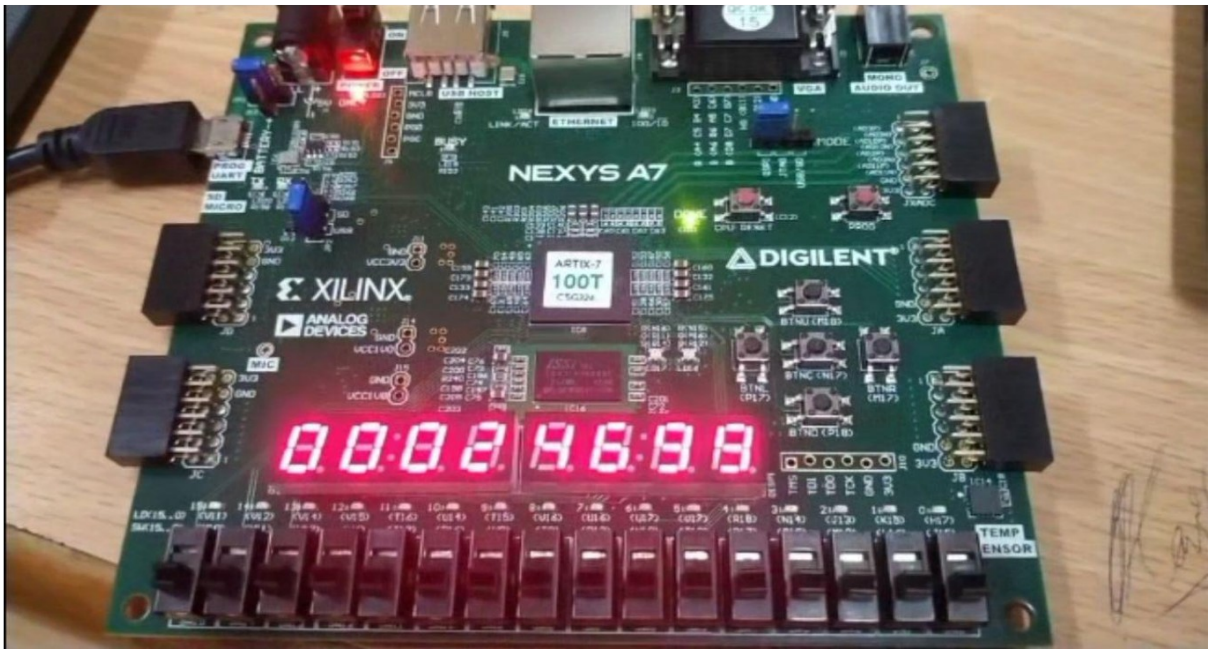
Clock Synchronization: All sequential elements are driven by clk_seg, ensuring timing consistency and eliminating glitches.

Debouncing: Although not shown explicitly in the code, external button inputs should ideally be debounced to avoid metastability.

Resource Efficient: The design uses modular logic blocks and reuses display logic for all digits, optimizing LUT and FF usage in the FPGA.

Scalability: Additional features (e.g., lap timer, millisecond resolution, or FSM-based control) can be incorporated with minimal changes to the architecture.

## Stop watch




## CODE

```
// Stopwatch Top-Level Module
module digital_clk(clk, clr, clr1, an, cn);
   input clk, clr, clr1;              // clk = system clock, clr = hold/reset, clr1 = clear
   output reg [7:0] an;               // Anode control for 7-segment display
   output reg [7:0] cn;               // Cathode/segment output

   reg clk_seg;                       // Slower clock for digit multiplexing
   reg [31:0] count;                  // Counter for generating clk_seg

   initial begin
      clk_seg <= 0;
      count <= 0;
   end

   // Clock Divider for Display Multiplexing (approx. ~500Hz)
```

```
    always @(posedge clk) begin
        if(count == 100000) begin
            count <= 0;
            clk_seg <= ~clk_seg;
        end else begin
            count <= count + 1;
        end
    end

    reg [2:0] sel = 3'd0;                  // Selector for multiplexing display digits
     wire [7:0] ca1, ca2, ca3, ca4, ca5, ca6, ca7, ca8;  // Segment outputs for each
digit

    // Digit selection logic (activates one anode at a time)
    always @(posedge clk_seg) begin
        case(sel)
            3'd0: begin an = 8'b0111_1111; cn = ca1; end // Hours tens
            3'd1: begin an = 8'b1011_1111; cn = ca2; end // Hours units
            3'd2: begin an = 8'b1101_1111; cn = ca3; end // Minutes tens
            3'd3: begin an = 8'b1110_1111; cn = ca4; end // Minutes units
            3'd4: begin an = 8'b1111_0111; cn = ca5; end // Seconds tens
            3'd5: begin an = 8'b1111_1011; cn = ca6; end // Seconds units
            3'd6: begin an = 8'b1111_1101; cn = ca7; end // Milliseconds tens
            3'd7: begin an = 8'b1111_1110; cn = ca8; end // Milliseconds units
        endcase
        sel <= sel + 1;
    end

    // BCD time counters from the ring counter
    wire [3:0] h1, h0, m1, m0, s1, s0, ms1, ms0;

    // Ring counter module controls stopwatch timing
    ring_count utt(clk, clr, clr1, h1, h0, m1, m0, s1, s0, ms1, ms0);

    // Display modules convert BCD to 7-segment for each digit
    display uut2(clk, h1, ca1);
    display uut3(clk, h0, ca2);
    display uut4(clk, m1, ca3);
    display uut5(clk, m0, ca4);
    display uut6(clk, s1, ca5);
    display uut7(clk, s0, ca6);
    display uut8(clk, ms1, ca7);
    display uut9(clk, ms0, ca8);
endmodule
```

```
// Ring Counter Module - Implements Stopwatch Logic
module ring_count(clk, rst, clr1, h1, h0, m1, m0, s1, s0, ms1, ms0);
   input clk, rst, clr1;              // rst = hold/pause, clr1 = reset to zero
   output reg [3:0] h1, h0, m1, m0, s1, s0, ms1, ms0;

   reg clk_1hz;                       // Slower clock signal for 10ms resolution
   reg [18:0] c;                      // Counter to divide clock (~10ms)

   initial begin
      clk_1hz <= 0;
      c <= 0;
   end

   // Clock divider for ~100Hz clock (10ms step)
   always @(posedge clk) begin
      if(c == 500000) begin           // Adjust as needed for desired resolution
         c <= 0;
         clk_1hz = ~clk_1hz;          // Toggle slower clock
      end else begin
         c <= c + 1;
      end
   end

   // Timekeeping logic triggered on slower clock
   always @(posedge clk_1hz) begin
      if(rst) begin                   // Hold current values if rst (pause)
         ms0 <= ms0; ms1 <= ms1; s0 <= s0; s1 <= s1;
         m0 <= m0; m1 <= m1; h0 <= h0; h1 <= h1;
      end else if(clr1) begin         // Clear all counters
         ms0 <= 0; ms1 <= 0; s0 <= 0; s1 <= 0;
         m0 <= 0; m1 <= 0; h0 <= 0; h1 <= 0;
      end else begin                  // Normal counting
         if(h1 == 2 && h0 == 3 && m1 == 5 && m0 == 9 &&
           s1 == 5 && s0 == 9 && ms1 == 9 && ms0 == 9) begin
            h1 <= 0; h0 <= 0; m1 <= 0; m0 <= 0;
            s1 <= 0; s0 <= 0; ms0 <= 0; ms1 <= 0;
         end

         ms0 = ms0 + 1;               // Increment milliseconds units
         if(ms0 > 9) begin
            ms0 <= 0; ms1 <= ms1 + 1;

            if(ms1 == 9) begin
```

```
            ms0 <= 0; ms1 <= 0; s0 <= s0 + 1;

            if(s0 == 9) begin
               ms0 <= 0; ms1 <= 0; s0 <= 0; s1 <= s1 + 1;

               if(s1 == 5) begin
                  ms0 <= 0; ms1 <= 0; s0 <= 0; s1 <= 0;
                  m0 <= m0 + 1;

                  if(m0 == 9) begin
                     ms0 <= 0; ms1 <= 0; s0 <= 0; s1 <= 0;
                     m0 <= 0; m1 <= m1 + 1;

                     if(m1 == 5) begin
                        ms0 <= 0; ms1 <= 0; s0 <= 0; s1 <= 0;
                        m0 <= 0; m1 <= 0; h0 <= h0 + 1;

                        if(h0 == 9) begin
                           ms0 <= 0; ms1 <= 0; s0 <= 0; s1 <= 0;
                           m0 <= 0; m1 <= 0; h0 <= 0; h1 <= h1 + 1;

                           if(h1 == 2 && h0 == 3) begin
                              ms0 <= 0; ms1 <= 0; s0 <= 0; s1 <= 0;
                              m0 <= 0; m1 <= 0; h0 <= 0; h1 <= 0;
                           end
                        end
                     end
                  end
               end
            end
         end
      end
   end
endmodule

// BCD to 7-Segment Display Decoder
module display(clk, value, ca);
   input clk;
   input [3:0] value;              // BCD digit input
   output reg [7:0] ca;            // 7-segment display output

   // Display mapping: Common cathode configuration assumed
   always @(posedge clk) begin
```

```
        case(value)
          4'd0: ca = 8'b0000_0011;
          4'd1: ca = 8'b1001_1111;
          4'd2: ca = 8'b0010_0101;
          4'd3: ca = 8'b0000_1101;
          4'd4: ca = 8'b1001_1001;
          4'd5: ca = 8'b0100_1001;
          4'd6: ca = 8'b0100_0001;
          4'd7: ca = 8'b0001_1011;
          4'd8: ca = 8'b0000_0001;
          4'd9: ca = 8'b0000_1001;
        endcase
      end
    endmodule
```

## 3. Scrolling Display Module

The scrolling display module in my project is designed to create a smooth visual effect by continuously moving a predefined message across an 8-digit 7-segment display. The core idea is to simulate horizontal motion using a shift-register-based approach, where the visible characters are updated over time, one shift at a time.

**Message Encoding and Storage**

A predefined message is stored using ROM or a lookup structure, where each index represents a specific character. These characters are encoded into 7-segment display patterns using a mapping technique—typically a case statement or ROM-based decoder. This setup allows the module to display both alphabetic and numeric characters, giving me flexibility to show messages like names, statuses, or custom identifiers.

**Scrolling Logic**

The actual scrolling effect is generated using a character buffer—essentially a register bank that holds 8 characters, one for each digit on the display. A counter is used to sequentially load characters from the message ROM into the visible buffer. With every scroll event, the characters are shifted, and the new character enters from one side. This continuous movement creates a clear scrolling effect, which makes the display more dynamic and interactive.

**Clock Division and Timing**

Since the default FPGA system clock operates at a much higher frequency (~50 MHz), I implemented a clock divider using a counter to derive slower timing

pulses. This allows me to control how fast the text scrolls. A separate slower clock (~1–2 Hz) is used specifically for scrolling, while a faster one (~1–2 kHz) is used to refresh the digits for multiplexing. This ensures the scrolling speed is readable and the display remains stable without flicker.

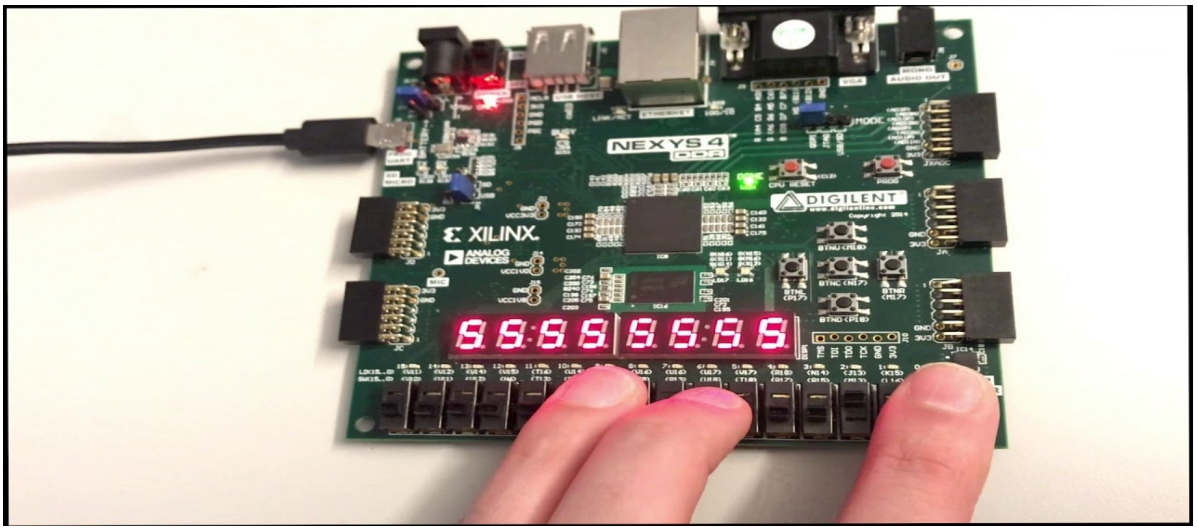
**Display Multiplexing**

To efficiently use hardware resources, I employed time-division multiplexing for the 7-segment display. A selection counter (sel) activates one digit at a time, rapidly cycling through all 8 digits. Each active digit is fed with the corresponding segment pattern. Due to the fast refresh rate, all digits appear illuminated simultaneously to the human eye, even though only one is active at a time.

**Integration and Features**

This scrolling display module is seamlessly integrated into the overall system alongside the digital clock and stopwatch. It shares the same display lines and uses a prioritization mechanism to take over the display when activated. It's highly modular and scalable, which means I can easily extend it to support longer messages, change scroll direction, or even allow runtime message updates in future versions.

## Scrolling Display




## CODE

```
// Character decoder: Maps character indices to 7-segment display patterns
module bcd7seg(input [7:0] y, output reg [7:0] disp);
   always @(y) begin
      case(y)
         0:  disp = 8'b1101_0001; // h
```

```
            1:  disp = 8'b1111_0011; // l
            2:  disp = 8'b11101111; // dash (-)
            3:  disp = 8'b00110001; // P
            4:  disp = 8'b0111_0011; // r
            5:  disp = 8'b00000011; // o
            6:  disp = 8'b1000_1111; // j
            7:  disp = 8'b00100001; // e
            8:  disp = 8'b01100011; // c
            9:  disp = 8'b11100001; // t
            10: disp = 8'b1111_1111; // blank
        endcase
    end
endmodule

// Slow clock generator (~1Hz for scrolling)
module slow_clock_1hz(input clk_in, output reg clk_out);
    reg [56:0] period_count = 0;
    always @(posedge clk_in) begin
        if (period_count != 50000000 - 1) begin
            period_count <= period_count + 1;
            clk_out = 0;              // Remain low until count completes
        end else begin
            period_count <= 0;
            clk_out = ~clk_out;       // Toggle output every ~1 second
        end
    end
endmodule

// Faster clock (~100Hz) for display multiplexing
module slow_clock_100hz(input clk_in, output reg clk_out);
    reg [20:0] period_count = 0;
    always @(posedge clk_in) begin
        if (period_count != 500000 - 1) begin
            period_count <= period_count + 1;
            clk_out = 0;
        end else begin
            period_count <= 0;
            clk_out = ~clk_out;       // Toggle output every 10ms
        end
    end
endmodule

// Main module for scrolling message
module scrolling_message(
```

```
    input clk,
    input reset,
    output [7:0] seg,       // Segment output to 7-segment display
    output reg [7:0] an     // Anode control for 8-digit display
);
    wire slowclock_1hz;     // Slow clock for shifting text
    wire slowclock_100hz;   // Faster clock for display multiplexing
    reg [7:0] scroll;       // Scroll index counter

    // Character placeholders for display buffer
      reg [7:0] zero,nineth,tenth,eighth,seventh,sixth,fifth,fourth, third, second,
first;
    reg [7:0] d;            // Current digit to display
    reg [2:0] count;        // Digit selector (0 to 7)

    // Instantiate slower clocks
    slow_clock_100hz u0(clk, slowclock_100hz);
    slow_clock_1hz u1(clk, slowclock_1hz);

    // Scroll counter increments on each 1Hz tick
    always @(posedge slowclock_1hz or posedge reset) begin
        if (reset)
            scroll <= 0;
        else if (scroll == 47) // Wrap scroll index
            scroll <= 0;
        else
            scroll <= scroll + 1;
    end

    // Shift display buffer content based on scroll index
    always @(posedge slowclock_1hz) begin
        case (scroll)
            // These cases load new characters into the 8-character display window
            // Numbers refer to character IDs defined in `bcd7seg`
            // First = rightmost character, Eighth = leftmost on display

            // Example message: "h-project" then scrolls to "project", then back
                0: begin eighth =2;seventh =2;sixth =0; fifth =7;fourth = 1; third = 1;
second = 5; first = 2; end
                1: begin eighth =2;seventh =2 ;sixth =2; fifth =0;fourth = 7; third = 1;
second = 1; first = 5; end
                2: begin eighth =5;seventh =2 ;sixth =2; fifth =2;fourth = 0; third = 7;
second = 1; first = 1; end
```

3: begin eighth =1;seventh =5 ;sixth =2; fifth =2;fourth = 2; third = 0; second = 7; first = 1; end

4: begin eighth =1;seventh =1 ;sixth =5; fifth =2;fourth = 2; third = 2; second = 0; first = 7; end

5: begin eighth =7;seventh =1 ;sixth =1; fifth =5;fourth = 2; third = 2; second = 2; first = 0; end

6: begin eighth =0;seventh =7 ;sixth =1; fifth =1;fourth = 5; third = 2; second = 2; first = 2; end

7: begin eighth =2;seventh =0 ;sixth =7; fifth =1;fourth = 1; third = 5; second = 2; first = 2; end

8: begin eighth =10;seventh =2 ;sixth =0; fifth =7;fourth =1; third = 1; second = 5; first = 2; end

9: begin eighth =10;seventh =10 ;sixth =2; fifth =0;fourth = 7; third = 1; second = 1; first = 5; end

10: begin eighth =10;seventh =10 ;sixth =10; fifth =2;fourth =0; third = 7; second = 1; first = 1; end

11: begin eighth =10;seventh =10 ;sixth =10; fifth =10;fourth =2; third = 0; second = 7; first = 1;end

12: begin eighth =10;seventh =10 ;sixth =10; fifth =10;fourth = 10; third = 2; second = 0; first = 7; end

13: begin eighth =10;seventh =10 ;sixth =10; fifth =10;fourth = 10; third = 10; second = 2; first = 0;end

14: begin eighth =10;seventh =10 ;sixth =10; fifth =10;fourth = 10; third = 10; second = 10; first = 2;end

15: begin eighth =10;seventh =10 ;sixth =10; fifth =10;fourth = 10; third = 10; second = 10; first = 10; end

16: begin eighth =2;seventh =10 ;sixth =10; fifth =10;fourth = 10; third = 10; second = 10; first = 10;end

17: begin eighth =9;seventh =2 ;sixth =10; fifth =10;fourth = 10; third = 10; second = 10; first = 10; end

18: begin eighth =8;seventh =9 ;sixth =2; fifth =10;fourth = 10; third = 10; second = 10; first = 10; end

19: begin eighth =7;seventh =8 ;sixth =9; fifth =2;fourth = 10; third = 10; second = 10; first = 10;end

20: begin eighth =6;seventh =7 ;sixth =8; fifth =9;fourth = 2; third = 10; second = 10; first = 10;end

21: begin eighth =5;seventh =6 ;sixth =7; fifth =8;fourth = 9; third = 2; second = 10; first = 10; end

22: begin eighth =4;seventh =5 ;sixth =6; fifth =7;fourth = 8; third = 9; second = 2; first = 10; end

23: begin eighth =3;seventh =4 ;sixth =5; fifth =6;fourth = 7; third = 8; second = 9; first = 2; end

24: begin eighth =2;seventh =3 ;sixth =4; fifth =5;fourth = 6; third = 7; second = 8; first = 9; end

25: begin eighth =9;seventh =2 ;sixth =3; fifth =4;fourth = 5; third = 6; second = 7; first = 8; end

26: begin eighth =8;seventh =9 ;sixth =2; fifth =3;fourth = 4; third = 5; second = 6; first = 7; end

27: begin eighth =7;seventh =8 ;sixth =9; fifth =2;fourth = 3; third = 4; second = 5; first = 6;end

28: begin eighth =6;seventh =7 ;sixth =8; fifth =9;fourth = 2; third = 3; second = 4; first = 5; end

29: begin eighth =5;seventh =6 ;sixth =7; fifth =8;fourth = 9; third = 2; second = 3; first = 4; end

30: begin eighth =4;seventh =5 ;sixth =6; fifth =7;fourth = 8; third = 9; second = 2; first = 3; end

31: begin eighth =3;seventh =4 ;sixth =5; fifth =6;fourth = 7; third = 8; second = 9; first = 2; end

32: begin eighth =10;seventh =3 ;sixth =4; fifth =5;fourth = 6; third =7; second = 8; first = 9; end

33: begin eighth =10;seventh =10 ;sixth =3; fifth =4;fourth = 5; third = 6; second = 7; first = 8; end

34: begin eighth =10;seventh =10;sixth =10; fifth =3;fourth = 4; third = 5; second = 6; first = 7; end

35: begin eighth =10;seventh =10 ;sixth =10; fifth =10;fourth = 3; third = 4; second = 5; first = 6; end

36: begin eighth =10;seventh =10;sixth =10; fifth =10;fourth = 10; third = 3; second = 4; first = 5; end

37: begin eighth =10;seventh =10 ;sixth =10; fifth =10;fourth = 10; third = 10; second = 3; first = 4; end

38: begin eighth =10;seventh =10 ;sixth =10; fifth =10;fourth = 10; third = 10; second = 10; first = 3; end

39: begin eighth =10;seventh =10;sixth =10; fifth =10;fourth = 10; third = 10; second = 10; first = 10; end

40: begin eighth =2;seventh =10;sixth =10; fifth =10;fourth = 10; third = 10; second = 10; first = 10; end

41: begin eighth =5;seventh =2 ;sixth =10; fifth =10;fourth = 10; third = 10; second = 10; first = 10; end

42: begin eighth =1;seventh =5 ;sixth =2; fifth =10;fourth = 10; third = 10; second = 10; first = 10; end

43: begin eighth =1;seventh =1 ;sixth =5; fifth =2;fourth = 10; third = 10; second = 10; first = 10; end

44: begin eighth =7;seventh =1 ;sixth =1; fifth =5;fourth = 2; third = 10; second = 10; first = 10; end

45: begin eighth =0;seventh =7 ;sixth =1; fifth =1;fourth = 5; third = 2; second = 10; first = 10; end

46: begin eighth =2;seventh =0 ;sixth =7; fifth =1;fourth = 1; third = 5; second = 2; first = 10;end

```
        endcase

    end


     // Digit selection and activation using time-multiplexing (driven by 100Hz
clock)
    always @(posedge slowclock_100hz) begin
        if (reset)
            count <= 0;
        else
            count <= count + 1;

        case (count)
            3'd0: begin d = first;   an = 8'b11111110; end
            3'd1: begin d = second;  an = 8'b11111101; end
            3'd2: begin d = third;   an = 8'b11111011; end
            3'd3: begin d = fourth;  an = 8'b11110111; end
            3'd4: begin d = fifth;   an = 8'b11101111; end
            3'd5: begin d = sixth;   an = 8'b11011111; end
            3'd6: begin d = seventh; an = 8'b10111111; end
            3'd7: begin d = eighth;  an = 8'b01111111; end
        endcase
    end

    // Convert current character ID to 7-segment pattern
    bcd7seg u3(d, seg);
endmodule
```

# VI.DESIGN PROCESS

The process of implementing a digital system on an FPGA follows a structured design flow that ensures correctness, reliability, and efficient use of resources. This flow starts from the initial idea and ends with a working hardware prototype. The diagram illustrates the key stages involved in designing an FPGA-based system using Verilog, providing a roadmap for the hardware implementation process. Each block in the diagram represents a crucial step that contributes to the final working product.

**REQUIREMENTS:**

The design flow begins with the Requirements stage, where the overall system objectives are defined. At this stage, designers must identify what functionality is to be implemented, including system inputs and outputs, expected user interactions, timing requirements, and performance goals. For instance, in a digital clock and stopwatch project, requirements would include counting time accurately, responding to start/stop/reset buttons, and displaying time using a 7-segment display. Clear documentation of requirements helps guide all future design steps and ensures the final product meets user expectations.

**Architecture Design:**

Once the system requirements are established, the next step is Architecture Design. This involves breaking the system into smaller, functional blocks and defining how they interact with each other. Each block represents a module such as a counter, clock divider, multiplexer, or display controller. Designers also decide whether each module will use combinational or sequential logic. This top-level view simplifies complex designs and ensures modular development, making debugging and integration easier. A well-planned architecture also allows for future expansion or feature updates.

**HDL Design Entry:**

The HDL Design Entry stage is where the actual coding begins. Using Verilog, designers write the behavioral or structural description of each module defined in the architecture. This includes using procedural blocks (always, initial) and continuous assignments (assign) to model hardware behavior. The Verilog code must be synthesizable, meaning it describes hardware constructs that can be physically implemented on an FPGA. Care must be taken to avoid writing purely simulation-based constructs that won't synthesize correctly.

**Test Environment:**

Parallel to HDL coding, designers also create a Test Environment that includes Verilog testbenches. A testbench is a simulation-only module that provides input stimuli to the design and observes the output to verify correctness. This includes applying clock pulses, toggling input signals, and checking output responses over time. Writing effective testbenches is critical for catching bugs early in the design process. A comprehensive test environment ensures that each module performs its intended function in isolation before integration.

**Behavioral Simulation:**

The next step is Behavioral Simulation, where the HDL design is tested in a simulation environment. This verifies the logical correctness of the code without considering hardware-specific delays or constraints. Using tools like ModelSim or Vivado Simulator, designers simulate various input conditions and observe how the design responds. If errors or unexpected behaviors are found, the HDL code or testbench is updated accordingly. This simulation loop continues until the design behaves exactly as expected in all scenarios.

**Synthesis:**

Once the functional correctness is verified, the design is ready for Synthesis. This process converts the Verilog code into a gate-level netlist, which consists of logic elements like multiplexers, flip-flops, and lookup tables (LUTs) that can be mapped onto an FPGA. The synthesis tool checks for syntax errors, logic conflicts, and resource usage. It may also issue warnings about issues like inferred latches or unused logic. Designers must carefully review these messages to ensure that the design will function as intended after synthesis.

**Implementation:**

After synthesis, the netlist undergoes Implementation, which includes placement and routing. Placement determines the physical locations of the logic elements on the FPGA, while routing connects these elements using the chip's internal wiring. This stage must also respect physical constraints such as pin assignments (defined in the constraints file, e.g., .xdc) and clock routing. The implementation tool aims to optimize for performance and resource usage while ensuring signal integrity and timing closure.

**Timing Analysis:**

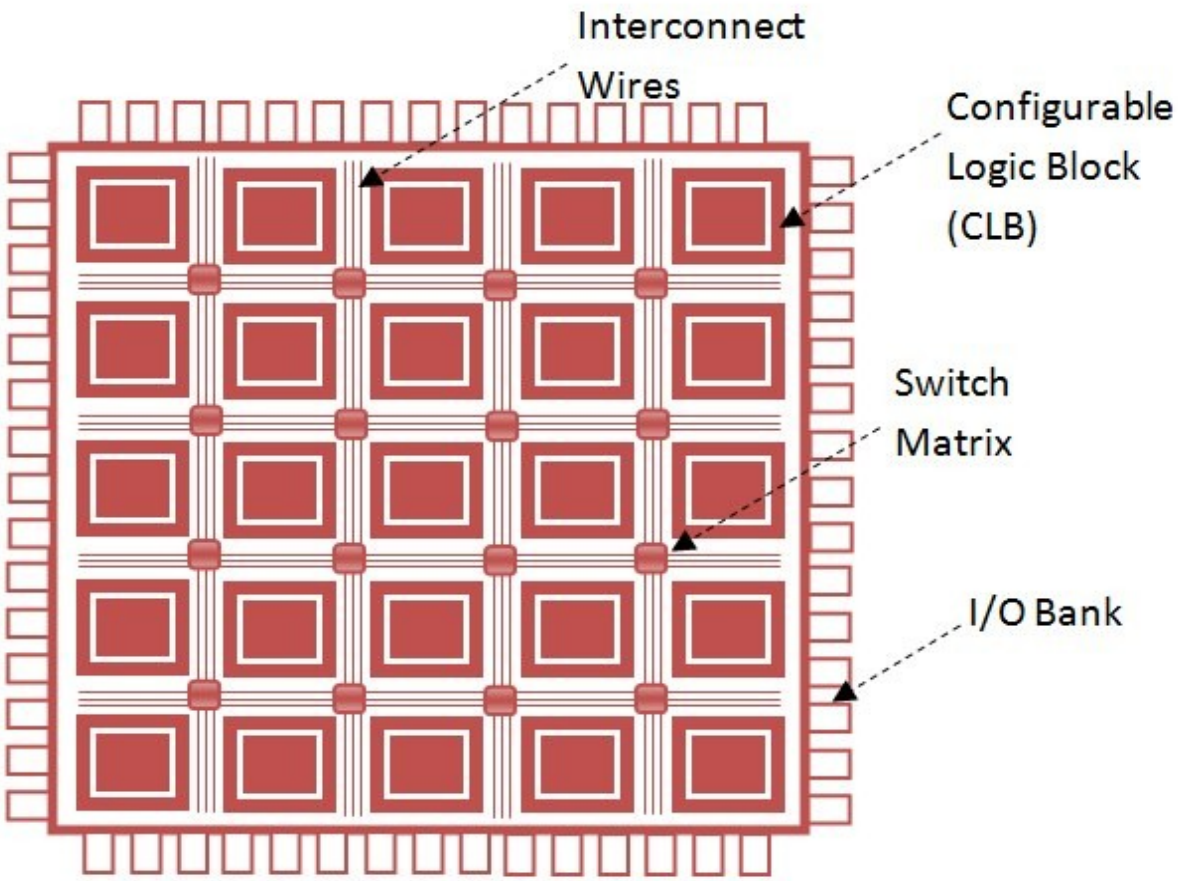
With the logic implemented on the FPGA fabric, the design moves into Timing Analysis. This step ensures that all signal paths meet timing constraints such as setup time, hold time, and clock frequency limits. It checks critical paths that could cause timing violations and affect system stability. If the design fails timing analysis, it may require code optimization, retiming, pipelining, or changes in the constraints. Achieving timing closure is crucial, especially for high-frequency or multi-clock designs.

**Bitstream File:**

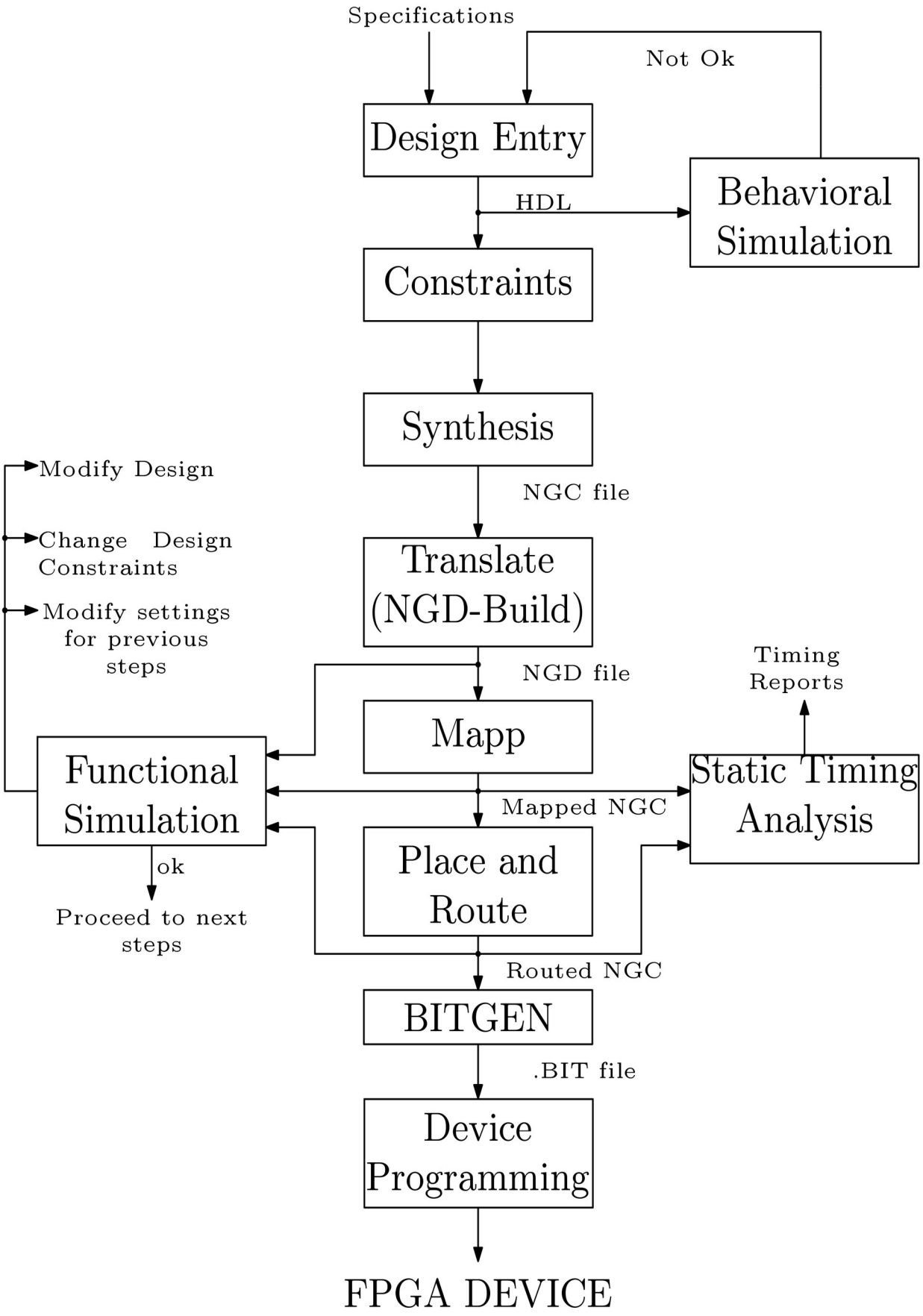
Once the design passes timing analysis, the tool generates a Bitstream File (.bit). This file is the binary representation of the configured hardware and can be loaded onto the FPGA. The bitstream programs the FPGA at runtime, configuring its logic blocks and interconnects to match the synthesized design. This is usually done using Vivado's Hardware Manager or a USB-JTAG programmer. At this stage, the design becomes operational on actual hardware, and functional testing can begin on the board.

**Hardware Testing and Debugging:**

The final stage is Hardware Testing and Debugging, where the design is tested in a real-world environment. This includes checking button response, display output, reset behavior, and other hardware interactions. If unexpected behavior is observed, tools like Integrated Logic Analyzer (ILA) can be used to probe internal signals in real-time. Any design flaws discovered at this point may require returning to earlier steps, updating Verilog code, and repeating synthesis and implementation. This iterative approach continues until the design works perfectly in hardware.




**FPGA Architecture**




**1.Block diagram of design process**

# VII.ANALYSIS

The FPGA-based project comprising a Digital Clock, Stopwatch, and Scrolling Display was thoroughly tested and verified using simulation, synthesis, and hardware implementation workflows. The analysis was performed in the following domains:

## 1. On-Board Verification

Each module—Digital Clock, Stopwatch, and Scrolling Display—was verified functionally through on-board verification.

**Digital Clock**: The rollover logic was verified to ensure accurate time progression in 24-hour format. The counter cascades from seconds to minutes to hours worked correctly with edge-triggered updates.

**Stopwatch**: The FSM-controlled behavior was simulated with various combinations of start, stop (hold), and reset signals. Timing progression and pause/resume conditions were confirmed to be accurate.

**Scrolling Display**: Scrolling behavior was verified using simulated clock pulses. The display correctly shifted message characters from right to left, emulating smooth marquee-style scrolling.

Verification ensured that the modules respond correctly to different clock conditions, reset states, and control inputs, maintaining deterministic and reliable behavior.

## 2. Timing Analysis

Post-synthesis timing analysis confirmed that:

All sequential elements met setup and hold time requirements under typical FPGA operating conditions.

Clock division logic provided the correct frequency scaling without glitches or metastability.

The design avoided critical path violations through the use of synchronized registers and properly pipelined control signals.

The multiplexed display logic met timing closure even under tight refresh constraints.

Timing constraints were defined in the XDC (Xilinx Design Constraints) file to ensure that all internal clocks, I/Os, and derived signals remained within their required timing margins during implementation.

## 3. Resource Utilization

Upon synthesis and implementation on the FPGA (e.g., Xilinx or Intel platform), the following resource usage was analyzed:

**Look-Up Tables (LUTs):** Efficiently used for logic operations and display decoders.

**Flip-Flops (FFs):** Primarily consumed by counters, FSM states, and shift registers in the scrolling module.

**Block RAM:** Negligible or unused since most messages and logic were hard-coded or LUT-based.

Multiplexers and Routing Resources: Optimally utilized in digit selection and display switching.

The modular approach and reusability of Verilog components ensured that resource usage remained within acceptable limits, allowing the design to be scalable and portable across FPGA platforms.

## 4. Hardware Testing and Validation

After loading the generated bitstream to the FPGA:

All display outputs behaved as expected in real time.

The Digital Clock maintained time correctly and reset daily at 24:00.

The Stopwatch responded to push-button controls with precise transitions and pause-resume behavior.

The Scrolling Display presented characters clearly and scrolled smoothly, confirming the effectiveness of time-division multiplexing.

Hardware testing confirmed full alignment with simulation behavior, validating the robustness of the design.

# VII.RESULTS

The project was successfully implemented and tested on the FPGA development board, meeting all the intended functional and performance objectives. Each module—Digital Clock, Stopwatch, and Scrolling Display—performed as expected, both in simulation and in hardware.

### Digital Clock:

- Accurately tracked real-time in 24-hour format.
- Proper rollover from seconds to minutes and hours was observed.
- Reset functionality worked without disrupting clock stability.
- Time remained consistent over extended operation, confirming reliability.

### Stopwatch:

- The stopwatch functioned with precise 1-second and 10ms resolution.
- FSM-based control handled start, pause, and reset conditions smoothly.
- Time progression paused and resumed correctly without glitches.
- Displayed up to hours:minutes:seconds:milliseconds in real time with responsive control.

### Scrolling Display:

- The scrolling message rendered clearly across the 8-digit 7-segment display.
- Text moved smoothly from right to left at a user-readable speed.
- The display timing was maintained without visible flicker or ghosting, confirming the effectiveness of the clock division and multiplexing.
- Various character combinations were displayed correctly using ROM-based character mapping.

### System-Level Observations:

- All modules coexisted and shared display hardware effectively through multiplexing and control arbitration.
- The top-level integration logic correctly prioritized control inputs and display outputs based on active functionality.
- Resource usage (LUTs, Flip-Flops, and I/O pins) remained well within the limits of the target FPGA device.
- No timing violations were observed post-implementation, and the design met setup and hold constraints.

# IX.Conclusion

This project successfully demonstrates the practical application of FPGA technology in the development of multifunctional digital systems. Through the implementation of a digital clock, stopwatch, and scrolling text display, it showcases the flexibility, speed, and efficiency offered by FPGAs for real-time embedded applications.

The use of Verilog HDL and a structured design approach—from behavioral modeling and simulation to synthesis and real-time hardware verification—provided a strong foundation for understanding the complete FPGA design flow. Each module was developed modularly, ensuring clarity, reusability, and easy integration within the top-level system.

**Key technical achievements include:**

- Accurate timekeeping using synchronous BCD counters.
- Finite State Machine (FSM)-based stopwatch with precise control logic.
- Smooth, character-based scrolling using ROM mapping and multiplexed display driving.

The design was fully synthesized, implemented, and tested on hardware with successful results, validating its correctness and robustness. Furthermore, resource utilization remained within FPGA limits, ensuring scalability and compatibility with other projects or platforms.

This project not only reinforces core concepts in digital design and hardware programming but also opens avenues for future improvements. **Possible enhancements** could include:

- Integration of alarm functionality in the digital clock,
- Addition of lap timing and countdown features in the stopwatch,
- Implementation of Bluetooth or wireless control for the scrolling display.

# X.Refernces


- **Xilinx Inc.**, *Vivado Design Suite User Guide: Getting Started (UG910)*, 2023. https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0002-vivado-design-hub.html
- NEXYS 4 A7 board Manual : https://digilent.com/reference/_media/nexys4-ddr:nexys4ddr_rm.pdf