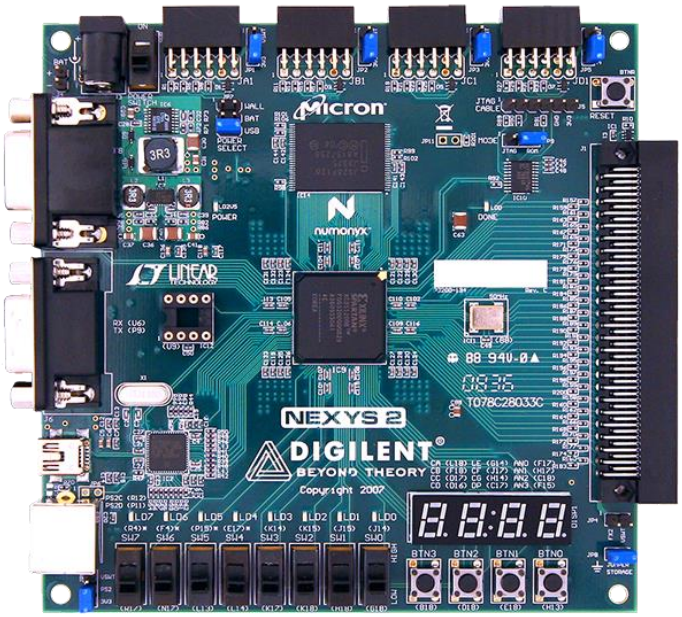
Project 01: Switch-Controlled LED Blinky



Prepared by: Muhammad Nuruddin bin Muktaruddin

Date: January 29, 2026

Platform: Digilent Nexys 2 (Spartan-3E FPGA)

Tools: Xilinx ISE 14.7, EDA Playground, Visual Studio, Python 3, Git

Table of Contents

1. Project Aim & Objectives

The primary goal of this project was to establish a complete **RTL-to-Bitstream** workflow for FPGA development. It moves beyond a simple "Hello World" by introducing **conditional logic** based on physical user input.

Specific Objectives:

- **Hardware Control:** Toggle an LED on/off using a physical slide switch.
- **Verification Strategy:** Use a **Python Golden Model** to verify logic *before* implementation.
- **Simulation:** Use UVM (Universal Verification Methodology) in EDA Playground to visualize timing.
- **Version Control:** Maintain a clean, professional Git repository, separating source code from build artifacts.

2. Process Flow (The Engineering Lifecycle)

The project followed a strict industry-standard lifecycle to ensure reliability.

1. **Requirement Analysis:** Defined the inputs (Switch 0) and outputs (LED 0).
2. **Golden Model (Python):** Wrote a Python script to calculate the expected "truth."
3. **RTL Design (VHDL):** Wrote the hardware description code.
4. **Verification (EDA Playground):** Simulated the VHDL against the Python model to catch logic errors.
5. **Synthesis (Xilinx ISE):** Compiled the VHDL into physical gates for the Spartan-3E chip.
6. **Pin Planning (UCF):** Mapped abstract VHDL signals to physical pins on the Nexys 2 board.
7. **Implementation:** Generated the .bit file and programmed the board.

3. Directory Structure & File Types

A clean folder structure is critical for collaboration and version control.

Extension	Name	Purpose
.vhd	VHDL Source	Describes the logic circuit behavior.
.ucf	User Constraints	Maps signals (e.g., clk) to physical FPGA pins.
.sv	SystemVerilog	Used for the Testbench (Verify the VHDL).
.py	Python Script	Used for the "Golden Model" (Math verification).
.do	Tcl Script	A macro script to automate the simulator commands.
.bit	Bitstream	The final binary machine code loaded onto the FPGA.

4. Code Analysis (Line-by-Line)

4.1 Hardware Design (rtl/blinky.vhd)

This is the code that lives inside the FPGA.

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; -- Standard signal definitions
use IEEE.NUMERIC_STD.ALL;    -- Allows math operations (unsigned)

entity blinky is
   Port ( clk : in  STD_LOGIC;                  -- 50MHz Clock Input
          sw  : in  STD_LOGIC_VECTOR (0 downto 0);  -- Switch Input
          led : out STD_LOGIC_VECTOR (0 downto 0)); -- LED Output
end blinky;

architecture Behavioral of blinky is
   signal counter : unsigned(25 downto 0) := (others => '0');
begin
   process(clk)
   begin
      if rising_edge(clk) then
         counter <= counter + 1; -- Count up every clock tick

         -- CONDITIONAL LOGIC
         if sw(0) = '1' then
            led(0) <= counter(25); -- Blink (Bit 25)
         else
            led(0) <= '0';       -- Force OFF
```

```
        end if;
      end if;
   end process;
end Behavioral;
```

4.2 Constraints (constraints/nexys2.ucf)

This bridges the code to the physical world.

Plaintext

```
NET "clk" LOC = "B8";    # Pin B8 -> 50MHz Crystal Oscillator
NET "sw<0>" LOC = "G18";  # Pin G18 -> Switch 0
NET "led<0>" LOC = "J14"; # Pin J14 -> LED 0 (Yellow)
```

4.3 Automation Script (verification/run.do)

Used in EDA Playground to run the simulation automatically.

Tcl

```
vlib work              # Create "work" library
vcom ../rtl/blinky.vhd    # Compile VHDL Source
vlog +incdir+... testbench.sv # Compile Testbench
vsim -c top -L uvm_1_2    # Run Simulator (Command Line)
run -all               # Run until finished
```

4.4 Python Golden Model (verification/blinky_model.py)

This script calculates the expected behaviour using software logic.

Python Code:

```
class BlinkyFPGA:
   def __init__(self, simulation_mode=False):
      # State Variables (Registers)
      self.counter = 0
      self.sw = 0
      self.led = 0
```

```
        # PARAMETERIZATION:
        # Hardware uses Bit 25 (Slow). Simulation uses Bit 4 (Fast).
        if simulation_mode:
            self.target_bit = 4
        else:
            self.target_bit = 25

    def rising_edge_clk(self):
        """Emulates the behavior of the VHDL process(clk)"""
        # 1. Counter Logic: Increment by 1
        self.counter += 1

        # 2. Rollover Logic: Simulate 26-bit hardware limit
        if self.counter >= 67108864:
            self.counter = 0

        # 3. Switch Logic (The core verification target)
        if self.sw == 1:
            # If Switch is ON, LED takes value of the Target Bit
            # (Right shift by target_bit and mask with 1)
            self.led = (self.counter >> self.target_bit) & 1
        else:
            # If Switch is OFF, LED is forced to 0
            self.led = 0

# --- MAIN EXECUTION BLOCK ---
if __name__ == "__main__":
    # Create instance in Simulation Mode (Fast)
```

```
    dut = BlinkyFPGA(simulation_mode=True)
    dut.sw = 1 # Force switch ON

    # Run loop to print cycle-by-cycle behavior
    for i in range(35):
        dut.rising_edge_clk()
        print(f"{i:<10} | {dut.counter:<10} | {dut.led}")
```

Output:

```
>>> MODE: Simulation Speedup (Bit 4)
Cycle     | Counter    | LED
------------------------------
0         | 1          | 0
1         | 2          | 0
2         | 3          | 0
3         | 4          | 0
4         | 5          | 0
5         | 6          | 0
6         | 7          | 0
7         | 8          | 0
8         | 9          | 0
9         | 10         | 0
10        | 11         | 0
11        | 12         | 0
12        | 13         | 0
13        | 14         | 0
14        | 15         | 0
15        | 16         | 1 <-- LED Turns ON (Bit 4 High)
16        | 17         | 1
17        | 18         | 1
18        | 19         | 1
```

19 | 20 | 1

20 | 21 | 1

21 | 22 | 1

22 | 23 | 1

23 | 24 | 1

24 | 25 | 1

25 | 26 | 1

26 | 27 | 1

27 | 28 | 1

28 | 29 | 1

29 | 30 | 1

30 | 31 | 1

31 | 32 | 0 <-- LED Turns OFF (Bit 4 Low)

32 | 33 | 0

33 | 34 | 0

34 | 35 | 0

4.5 UVM Testbench (verification/testbench.sv)

The SystemVerilog testbench wraps around the VHDL to simulate it.

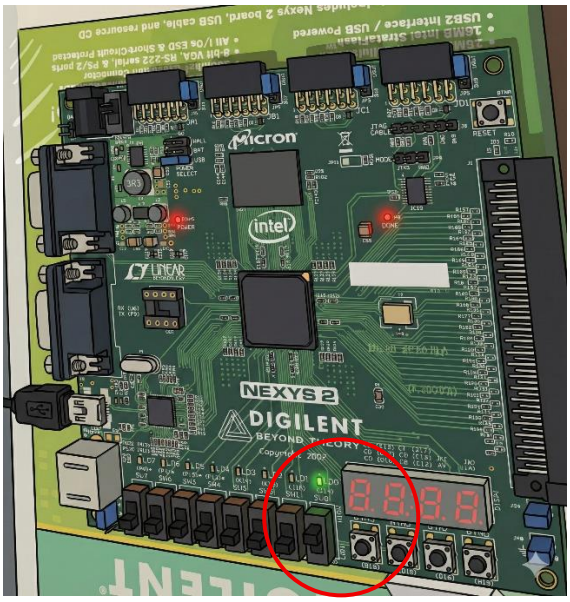
Code snippet:

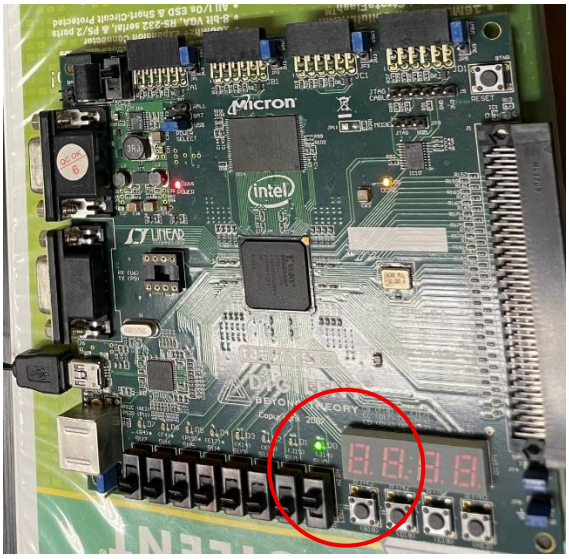
```
// 1. INTERFACE: Defines the wires connecting Testbench to DUT
interface blinky_if;
  logic clk;
  logic [0:0] sw;
  logic [0:0] led;
endinterface


// 2. DRIVER: The "Hand" that flips switches
class blinky_driver extends uvm_driver;
```

```
  // ... (UVM registration code) ...

  virtual task run_phase(uvm_phase phase);
    // TEST SCENARIO
    vif.sw = 0;   // Start with Switch OFF
    #2000;       // Wait 2000 nanoseconds
    vif.sw = 1;   // Turn Switch ON
    #5000;       // Wait 5000 nanoseconds
  endtask
endclass

// 3. TOP MODULE: The physical connection
module top;
  // Generate Clock: Toggle every 10ns (50 MHz)
  logic clk;
  initial begin
    clk = 0;
    forever #10 clk = ~clk;
  end

  // Instantiate Interface
  blinky_if vif();
  assign vif.clk = clk;

  // Instantiate DUT (Device Under Test) - Connecting VHDL to SV
  blinky dut (
    .clk(vif.clk),
    .sw(vif.sw),
```

```
  .led(vif.led)
 );

 // Start UVM Test
 initial begin
  uvm_config_db#(virtual blinky_if)::set(null, "*", "vif", vif);
  run_test("blinky_test");
 end
endmodule
```

Output: (As in red circle)





5. Software & Tools Settings

5.1 EDA Playground (Simulation)

- **Limitation:** Cloud simulators are slow. Simulating 1 second of real time (50 million cycles) takes hours.
- **Workaround:** We implemented a **"Simulation Speedup"** parameter.
    - **Hardware Mode:** Uses Bit 25 (Slow blink).
    - **Simulation Mode:** Uses Bit 4 (Fast blink, only 16 cycles).
- **Tools Used:** Aldec Riviera-PRO (simulator) and UVM 1.2 (verification library).

5.2 Xilinx ISE 14.7 (Implementation)

- **Role:** Synthesizes the VHDL into hardware gates.
- **Setting:** Configured for **Spartan-3E (XC3S500E-4FG320)**.
- **Crucial Step:** Manually adding the source files from the clean 01_Blinky directory instead of creating them inside the project folder.

6. Troubleshooting & Lessons Learned

Throughout Project 1, several critical engineering lessons were learned:

1. **The "Ghost Folder" Issue:**
    - *Mistake:* Created nested folders like 00_Logbook/00_Logbook.
    - *Fix:* Flattened the directory structure. Always check file paths before committing.
2. **Git "Trash" Files:**
    - *Mistake:* The repository was initially cluttered with .ise, .ncd, and .html files.
    - *Fix:* Created a .gitignore file to automatically exclude build artifacts.
3. **Naming Mismatch (Critical):**
    - *Mistake:* The UCF file used old names (clk_50mhz, led_out) while the new VHDL used standard names (clk, led). This caused synthesis failure.
    - *Fix:* Updated UCF to match VHDL port names exactly.
4. **Python Path Issue:**
    - *Mistake:* Windows "App Execution Alias" prevented Python from running in VS Code.
    - *Fix:* Used the py launcher or installed Python directly from the Microsoft Store.