Week 1- Hardware Design & Verification

Section 1 – Combinational Logic

Objectives

- Design and implement three combinational circuits:
    1. **32-bit Adder**
    2. **Encoder** (8-to-3)
    3. **32-bit Barrel shifter**
- Develop a **self-checking testbench** for all circuits.

Tasks

1. Implement a 32-bit Adder

- Create a 32-bit adder module (`adder32.sv`)
- Inputs:
    - `a[31:0], b[31:0], cin`
- Outputs:
    - `sum[31:0], cout`

2. Implement an Encoder

- Design a **priority 8-to-3 encoder**:
- Inputs: `in[7:0]`
- Output: Encoded output (`out`)

3. Implement a Barrel Shifter

- Create a barrel shifter module (`barrel_shifter.sv`)
- Function: Logical left and right shift
- Inputs:
    - `data_in[31:0], shift_amt[4:0], dir` (0 = left, 1 = right)
- Outputs:
    - `data_out[31:0]`

4. Write Self-Checking Testbenches

Create separate self-checking testbenches for:

- `adder32_tb.sv`
- `encoder_tb.sv`
- `barrel_shifter_tb.sv`

Each testbench must:

- Apply both **directed** and **randomized** input vectors.
- Compute the **expected output** inside the testbench (not manually).
- Compare DUT output with expected results automatically.
- Print **PASS/FAIL** messages for each test case.
- End simulation with a summary (e.g., number of passed/failed tests).

Testbench Requirements:

- For the adder:
    - Check correctness for random 32-bit values of `a`, `b`, and `cin`.
    - Include edge cases:
        - `0 + 0`
        - Maximum values: `0xFFFFFFFF + 0xFFFFFFFF`
        - Overflow scenarios
- For the encoder:
    - Test all valid input patterns (exactly one bit high).
    - Test invalid patterns (multiple bits high) — should report as *invalid* or use priority_encoding rules.

Deliverables

- `adder32.sv`
- `encoder.sv`
- `barrel_shifter.sv`
- `adder32_tb.sv`
- `encoder_tb.sv`
- `barrel_shifter_tb.sv`
- Simulation results (log/waveform snapshots)