PROJECTS

Must do projects for everyone (implementing complete vlsi design flow)

1. **Top-down approach (must for digital vlsi)**

- Design a full adder circuit in Verilog (hint: keep in mind that it should be synthesizable)
    - Write a test bench covering all the possibility to verify it.
    - Design the above circuit in c modelling (write its c code) and use it to verify your testbench outputs.
    - Now, take a paper and draw the hardware for verified rtl code you have written.
    - Try to minimize the area and power restructuring the design.
    - Now code back the optimised design in Verilog. And again verify it with the c modelling outputs.
    - Synthesize the final verified rtl code and implement the complete PnR flow. Generate its final layout.
    - Do the design rule check (DRC) and layout vs schematic (LVS) for the design.
- **Tools to use**:
    - RTL design/Verification– vivado or eda playground
    - Synthesys – Yosys
    - PnR – open road
    - LVS - netgen

2. **Bottom – Up approach (must for analog and backend)**

- Design PMOS and NMOS, analyze all the characteristics of Mosfets
    a. Id vs Vgs
    b. Id vs Vds
    c. Voltage transfer characteristics.
    - By changing the load, compare all the above graphs
    - Using the above pmos and nmos design a cmos inverter.
    - Analyse all the graphs as above – for Cmos
    - Now, design all the gates – AND, OR, NAND ,NOR, XOR and XNOR by using your PMOS, NMOS and CMOS.
    - Analyse all the above characteristics for each gate.
    - Take a capacitor as a output load and analyse the rise time (time the capacitor takes to charge) and fall time (time taken by capacitor to discharge).
    - After this analyse the logical efforts for these gates, specially compare nand and nor in terms of speed and power.
    - Now, design a half adder – for the best possible power and speed (based on the data you have analysed above) using the gates designed above then design a full adder using the half adders.

- **Tool to use**
  - Design and characteristics analysis – cadence virtuoso.
  - LTSpice

Analog VLSI

a. **LEVEL 1**
   i. Design mosfets (nmos and pmos) with different libraries and analyse the characteristics.
   ii. Design a simple opamp circuit and play with gain and bandwidth.
   iii. Design low pass, high pass, bandpass bandstop filters.

b. **LEVEL 2**
   i. Design PLL - phase-locked loop
   ii. Low noise amplifier

c. **LEVEL 3**
   i. Design a heterodyne receiver
   ii. Design VCO - https://ieeexplore.ieee.org/document/10119339 Simulate and document its performance

Digital VLSI

a. Frontend
   i. Design
      1. **LEVEL 1**
         a. Implement Calculator – add, subtract, multiply, divide - Gate level modelling
         b. Multi bit Barrel shifter
         c. BCD counter with up and down counting.
      2. **LEVEL 2**
         a. Traffic light controller using FSM approach
         b. Vending machine/washing machine using FSM aproach
         c. Synchronous FIFO
         d. Booth multiplier.
         e. Code converter using datapath and control path properly. http://lnkiy.in/codeconverterpdf
      3. **LEVEL 3**
         a. RISC V – implementation of 5 staged pipelined RISC-V processor. Also implement hardware detection if possible.
         b. MIPS – implementation of 5 staged pipelined MIPS Processor. Also implement hardware detection if possible.
         c. Half precision floating point adder and multiplier

ii. Verification

1. **LEVEL 1**
    a. Write proper testbench to verify vending machine
2. **LEVEL 2**
    a. Test bench for Synchronous FIFO
3. **LEVEL 3**
    a. Do UVM verification for AXI interface
    b. Formal verification for RISC v processor.
    c. Half precision floating point adder and multiplier

b. Backend

i. Physical design

1. **LEVEL 1**
    a. Take simple nand gate and implement its layout.
2. **LEVEL 2**
    a. Take any wrapper from openroad Designs like swerv_wrapper, Do the Floorplaning and adjust the
        i. PLACE_DENSITY = default
        ii. PLACE_DENSITY = default – 40%

    Analyze both the timings using OpenTime.

3. **LEVEL 3**
    a. RISC v processor – RTL to GDS
        i. Do the floorplanning according to the hierarchy, have atleast 3 set of floor plans as experiments.
        ii. Analyse the placement (utilization and congestion).
        iii. Spread out the design with different place_density.
        iv. At cts, analyse the clock structure and try reducing the latency without effecting timing and skew.
        v. Annalyse the timing in OpenTime (Do Sizing , adding buffers)
        vi. Clean up drc and lvs on the experiment with best results.

Tools :

- RTL design/Verification– vivado or eda playground
- Synthesis – Yosys
- PnR – open road
- LVS – netgen
- Spice netlist simulation – ngspice
- OpenTime – Timing Analysis

EMBEDDED

1. **LEVEL 1**
    a. Smart street lights using IoT – motion sensor street lights
    b. Traffic light control system – railway crossing traffic light control
    c. Smoke detector – sensor beeps when smoke is detected
    d. Vehicle speed check and accident-avoidance system – check vehicle distance from the front vehicle continuously and buzz the buzzer if the distance is less than the set values. Mainly on highways to avoid crash.
2. **LEVEL 2**
    a. Robotic Arm - allows controlling a robotic arm by hand movements
3. **LEVEL 3**
    a. Take any 2 microcontrollers (LPC2148, Atmega (Arduino) etc) and communicate between the using
        i. SPI
        ii. I2C
        iii. UART
        iv. CAN (you can use MCP2515 can controller)
    b. Analyze which protocol will suit for:
        i. Single master single slave
        ii. Single Master multiple slave
        iii. Multiple master multiple slave
    c. Choose an appropriate protocol to implement the below project:
        i. Develop a complete system for a smart vehicle with several ECUs - Electronic Control units (slaves) interacting with one master node.
        ii. Here these ECUs can be Airbag system, Obstacle avoidance system, Fuel system, Anti theft system, Temperature system.
        iii. Each ECUs can have its own microcontroller connected to an appropriate sensor. Like for the airbag system there will be a sensor to detect the impact, based on the impact value the ECU decides if the Air bag should be opened or not. If opened ECU sends data to the master node about the impact, the master can then send the location and call an ambulance.
        iv. Basically each slave node collects data from their sensors and send it to the master and the master takes appropriate call.

(Hint: check which protocol used in today's automobiles)