Digital Design (CS F215) **17-09-2025**

Lab #5: Decoders and Encoders

Lecturer: Harikrishnan N B *Student:*

READ THE FOLLOWING CAREFULLY:

Honour Code for Students: I shall be honest in my efforts and make my parents proud.
Write the oath and sign it in your lab notebook. Use a dedicated notebook for all lab theory/design work.

Marks: 3M Theory + 17M Implementation = 20M

Instructions:

Step 1 - Theory (Notebook): Write all truth tables, K-maps and logic equations neatly.

Step 2 - Implementation (Verilog): Implement in *structural Verilog* and simulate with the provided testbenches. Do NOT use dataflow or behavioural modelling for this lab.

Important:

1. Ensure that your theory is checked by a TA before you leave the lab. You do not need to wait for a TA to check your theory before you start writing Verilog code.
2. Make sure to prefix any modules you include with `src/`. For example, if you're including `half_adder.v`, include it as `src/half_adder.v`
3. Ensure you're running all the scripts from the `Lab05` folder.
4. Autograder marks are final marks. Follow all submission guidelines carefully in README file.
5. Best of Luck ☺.

Submission Deadline: **03:40 PM** **Submission Portal**: quanta.bits-goa.ac.in

Background

Decoders in System Design:

Decoders are fundamental building blocks in digital systems. By transforming compact binary inputs into one-hot outputs, they allow for easy comparison, selection, and condition checking.
One-hot means that only one output/input line is active (logic 1) at a time, and all the other outputs/inputs are 0.

Q1. Decoder and Encoder Design with Enable Pins (5 marks)

Theory:

1. Design a **2-to-4 Decoder with Enable**:
   (a) Write its truth table showing how 2 input bits A_1, A_0 and an Enable input E generate 4 outputs D_0, D_1, D_2, D_3 such that only one output is active for each input combination when $E = 1$, and all outputs are 0 when $E = 0$. (0.5M)
   (b) Write the Boolean expression for each output bit D_i in terms of A_1, A_0 and E. (1M)
2. Design a **4-to-2 Priority Encoder with Enable**:
   (a) Write its truth table, ensuring that if multiple inputs $I_3 \dots I_0$ are active, the highest-order input has priority. Include an Enable input E and Valid bit V. The outputs (Y_1, Y_0 and V) are all 0 if E=0 and they are also 0 if all inputs , $I_3 \dots I_0$, are 0. (0.5M)
   (b) Write the Boolean expressions for outputs Y_1, Y_0 and V in terms of inputs $I_3 \dots I_0$ and E. (1M)

Implementation (Verilog code):

1. Write Verilog code for the 2:4 Decoder with Enable. (1M)
2. Write Verilog code for the 4:2 Priority Encoder with Enable. (1M)

Q2. Podium Permutations (15 marks)

During Zephyr finals night in the Auditorium, there are 4 winners standing on 4 podiums, each podium labeled 0–3. Each finalist wears a badge numbered 0–3.

The floor manager wants a quick system that can:

1. Check if the lineup is a valid permutation — each badge appears exactly once. (2M)
2. Report if the lineup has an odd number of **fixed points** (a fixed point is when the badge number matches the podium number). If the lineup is not valid, force `POI = 0`. (2M)
3. Build a simple **Index Map (IMAP)** — for each badge number (0–3), output the podium number where that badge is currently standing. IMAP must always be produced, even for invalid lineups (if the same

badge appears multiple times, assign the highest podium index to that badge; if a badge is missing, assign 0 by default). (11M)

I/O:

1. **Inputs:**
   - `N0[1:0], N1[1:0], N2[1:0], N3[1:0]` — the badge numbers of finalists at podiums 0–3.
   - Each `Ni` corresponds to **podium i**, and its value is the **badge number** at that podium.
   - Each `Ni` is two bits long, since badge number (value stored in `Ni`) $\in \{0, 1, 2, 3\}$, it requires two bits to represent.
2. **Outputs:**
   - `VALID` — 1 if the lineup is a valid permutation (all four badges appear exactly once).
   - `POI` — 1 if there are an odd number of fixed points.
   - `IMAP[7:0]` — for each badge i (0–3), {IMAP[2*i + 1], IMAP[2*i]} = podium number where that badge stands (since podium number $\in \{0, 1, 2, 3\}$, it requires two bits to represent).

Constraints:

- Structural only. Allowed: gate primitives (and, or, xor, xnor, not, nand, nor, buf) and module instantiation.
- Forbidden: assign, always/initial, case/if, loops, equality/relational/arithmetic operators (==, !=, >, <, +, -, etc.).
- Decoders and encoders are encouraged to simplify the design.

Example 1 (Valid Lineup):

```
Input:
N0 = 2, N1 = 0, N2 = 3, N3 = 1

Output:
VALID = 1
POI  = 0
IMAP  = [1, 3, 0, 2]
```

1. **Check validity**
   All four numbers 0–3 appear exactly once → VALID = 1.
2. **Check fixed points**
   None of the podiums have badge equal to its index ($2 \neq 0,\ 0 \neq 1,\ 3 \neq 2,\ 1 \neq 3$) → POI = 0.
3. **Build IMAP**

- Badge 0 is at podium 1 → IMAP[1:0] = 1
- Badge 1 is at podium 3 → IMAP[3:2] = 3
- Badge 2 is at podium 0 → IMAP[5:4] = 0
- Badge 3 is at podium 2 → IMAP[7:6] = 2

Example 2 (Invalid Lineup):

```
Input:
N0 = 2, N1 = 2, N2 = 1, N3 = 3

Output:
VALID = 0
POI  = 0
IMAP  = [0, 2, 1, 3]
```

1. **Check validity**
   Badge 2 is repeated, and badge 0 is missing → VALID = 0.
2. **Check fixed points**
   Since lineup is invalid, force POI = 0.
3. **Build IMAP**
   - Badge 0 missing → IMAP[1:0] = 0 (default)
   - Badge 1 is at podium 2 → IMAP[3:2] = 2
   - Badge 2 appears at podiums 0 and 1 → IMAP[5:4] = 1 (highest podium wins)
   - Badge 3 is at podium 3 → IMAP[7:6] = 3

Debugging Tips

- Always declare intermediate wires before use.
- Check carefully for missing commas/semicolons.
- Ensure all gate inputs/outputs are explicitly connected.
- If an output shows `X`, recheck wire connections.
- For debugging, use the `--verbose` flag in `RunTestCases`.
  > `./RunTestCases L5 <BITS_ID> --verbose`.