Digital Design (CS F215) **12-09-2025**

Lab #5 Practice Problem: Encoders and Decoders

Lecturer: Harikrishnan N B *Student:*

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. In this lab we explore their power in a real-world inspired "voting circuit" where decisions are made based on majority input values.

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. Warm-up: Decoder Basics

1. Implement 2:4 decoder in `decoder_2to4.v` which takes two inputs - `A[1:0]` and an enable signal `E` and produces an output `O[3:0]`. All the bits in the output will be 0 whenever `E = 0`.

2. Implement 3:8 decoder in `decoder_3to8.v` using 2:4 decoder modules. The module should take two inputs - `A[2:0]` and an enable signal `E` and produces an output `O[7:0]`. All the bits in the output will be 0 whenever `E = 0`.

3. Implement a full adder in `fullAdder.v` using a `3:8 decoder` and two OR gates. The module should take the usual inputs and produce the usual outputs.

4. Test your decoder and full adder by creating appropriate testbenches and testing against a truth table.

Q2. CC Lab Circular Roll-Call

During CC attendance, the TA writes down the roll numbers of the next **4** students entering the lab as a short list: `[S0, S1, S2, S3]`.

The prof wants to see if the roll numbers form part of a circular roll-call — where one student's number is immediately followed by its next roll number in order, either going forward (clockwise) or backward (counter-clockwise) in the cycle $0 \to 1 \to 2 \to 3 \to 0$.

You are given:

- the **4** roll numbers in order.
- an **anchor** number to start watching for.
- and a **mode**: 0 means look for a forward step (`anchor`, `anchor + 1 mod 4`), and 1 means look for a backward step (`anchor`, `anchor - 1 mod 4`)

Your task is to:

1. Scan all 4 adjacent pairs `(S0,S1), (S1,S2), (S2,S3), (S3,S0)` (wrapping around).
2. Detect the first pair that shows (`anchor`, `anchor ± 1`) depending on the mode.
3. Report the index (0–3) of the pair's starting position, and the second roll number in that pair.

I/O:

1. **Inputs:**
   - `S0[1:0], S1[1:0], S2[1:0], S3[1:0]` — 2-bit roll numbers in order.
   - `PAT[1:0]` — the anchor roll number.
   - `MODE` — 0 = forward, 1 = backward.
2. **Outputs:**
   - `MATCH[3:0]` — bit i is 1 if window (Si, S(i+1) mod 4) matches the target pair.
   - `ANY` — 1 if any match exists.
   - PAR — parity (XOR) of MATCH bits (odd=1).
   - `FIRST[1:0]` — start index of the match; 00 if none.
   - `Y[1:0]` — the second roll number of the matching pair; 00 if none.

Extra Rules:

- It is guaranteed that there are 0 or 1 matches (never more). So `MATCH` is either all zeros or one hot (only one bit is 1; the rest are 0).
- The second roll number must be `00` if no match exists.

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/encoders are encouraged.

Example 1:

```
Input:
S0 = 2, S1 = 3, S2 = 1, S3 = 0
PAT = 2
MODE = 0

Output:
MATCH = 0001
ANY   = 1
PAR   = 1
FIRST = 00
Y     = 3
```

1. **Work out the target pair** Since `PAT` $= 2$ and `MODE` $= 0$ (which means clockwise), the target pair we are looking for is: (2,3) because in a clockwise roll-call, the number after 2 should be 3.

2. **List all four adjacent pairs**
   We now write down the four possible adjacent pairs of roll numbers from the array:
   - Pair starting at index $0 \rightarrow (S_0, S_1) = (2, 3)$
   - Pair starting at index $1 \rightarrow (S_1, S_2) = (3, 1)$
   - Pair starting at index $2 \rightarrow (S_2, S_3) = (1, 0)$
   - Pair starting at index $3 \rightarrow (S_3, S_0) = (0, 2)$ (note the wrap-around)

3. **Compare each pair to the target pair** $(2, 3)$
   - The first pair $(2, 3)$ **matches** the target exactly.
   - The second pair $(3, 1)$ does not match because the first element is 3, not 2.
   - The third pair $(1, 0)$ does not match because the first element is 1, not 2.
   - The fourth pair $(0, 2)$ does not match because the first element is 0, not 2.

   So, only the pair starting at **index 0** is a match.

4. **Work out the outputs**
   - `MATCH[3:0]`: only the pair at index 0 matched, so this should be `0001`.
   - `ANY`: since at least one match was found, `ANY` $= 1$.
   - `PAR`: there was exactly one match (an odd number), so `PAR` $= 1$.
   - `FIRST`: the first (and only) match was at index 0, so `FIRST` $=$ `00`.
   - `Y`: in the matching pair $(2, 3)$, the second roll number is 3. So `Y` $= 3$.

Example 2:

```
Input:
S0 = 1, S1 = 0, S2 = 3, S3 = 2
PAT = 2
MODE = 1
```

```
Output:
MATCH = 1000
ANY   = 1
PAR   = 1
FIRST = 11
Y     = 1
```

Q3. Circuit Design using Multiplexers (Optional Question for FUN)

$$Z1 = ABC' + AB'C + A'BC$$
$$Z2 = AB + AC + BC$$

Theory:

1. Implement the functions using only three 2:1 multiplexers.

Implementation:

1. Implement 2:1 multiplexer in `mux_2to1.v`.
2. Implement your design in `mux_design.v`.
3. Verify outputs by creating a testbench and testing against a truth table.