**ELX304 – Electronic Systems**

**Individual Coursework Assignment**

**Digital Design**



**WERE VINCENT**

**12/09/2024**

1. **Task 1: Synchronous Design Problem**

**(a) Operational Limitations**

The given robot control system uses proximity sensors (S1, S2) to control the motor signals (Z1, Z2). However, operational limitations includes the following;

* ***State ambiguities-*** If the robot receives conflicting sensor signals (like S1 = 1 and S2 = 1 or both 0), the system may behave unpredictably.
* ***Non-optimized transitions-*** The system might include redundant states or transitions that are not necessary for efficient motor control.
* ***Lack of memory-*** No memory mechanism is provided to store previous states, making the control system react only to the current inputs, which may cause instability if the sensor data fluctuates quickly.

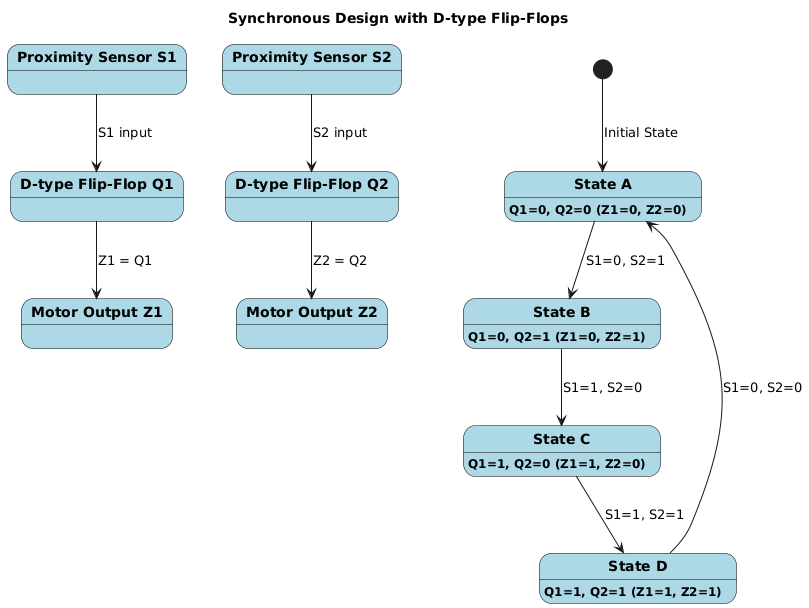
**(b) Minimal Synchronous Solution using D-Type Flip-Flops**

For the synchronous design,a state machine will be built that takes the proximity sensor inputs (S1, S2) and provides motor outputs (Z1, Z2). We need to use D-type flip-flops to store the states of the system and define the transition rules.

The steps are;

1. ***Define the States***

By assuming 4 possible states (A, B, C, D) as shown in the diagram from the brief. Each state corresponds to a specific behavior of the robot’s motors based on the sensor inputs.

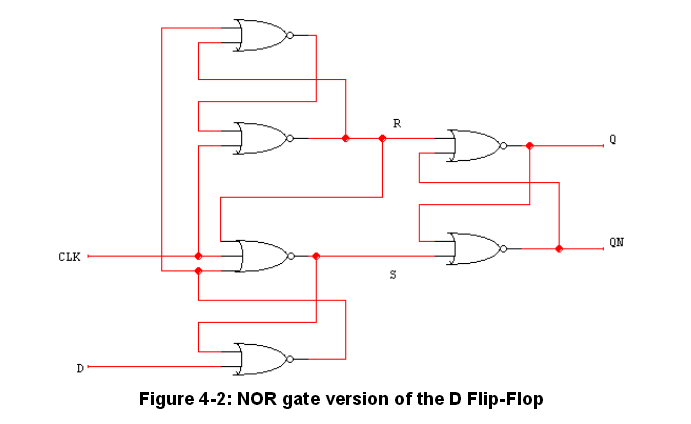


1. ***State Table-*** The design the state transition table with current states (Q1, Q2), inputs (S1, S2), and the next states (D1, D2).



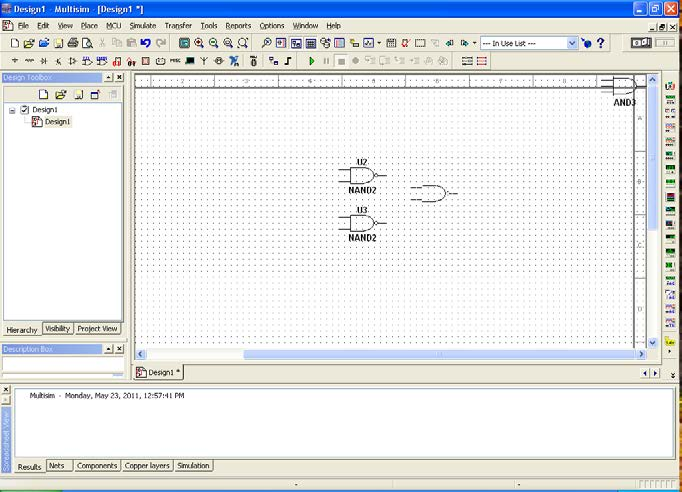
1. ***D Flip-Flop Logic***

* Use the Karnaugh maps (K-maps) to simplify the logic equations for the D inputs of the flip-flops (D1 and D2).
* Implement the transition logic using D flip-flops and logic gates (AND, OR, NOT).



1. **Simulation of the System**

* The Multisim project was set by starting a new file and configuring the workspace for my components. I ensured that the workspace was large enough to accommodate all the necessary parts like D flip-flops, logic gates, inputs, and outputs.
* Next, the components was selected as needed from the Multisim components library

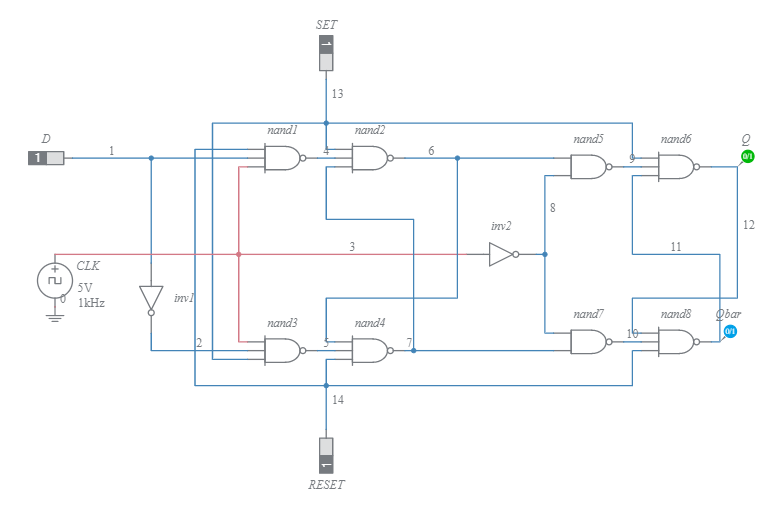


1. Two D flip-flops (74LS74) was added for storing the current states (Q1 and Q2).
2. The logic gates (AND, OR, NOT) were added to build the transition logic for the flip-flops.
3. The binary switches was also added to simulate the proximity sensors (S1 and S2), and LEDs to visualize the motor outputs (Z1 and Z2).

* Then wired the D flip-flops by;

1. Connecting the D input of the first flip-flop (Q1) to the output of the second flip-flop (D1 = Q2) based on the logic equation derived from the state transition table.
2. For the second flip-flop (Q2), the more complex logic equation D2 = S1 · ¬Q1 + S2 · ¬Q2 using AND, OR, and NOT gates were implemented. Then wired the gates to ensure that the inputs (S1, S2) were correctly applied to the gates.

* Next, connected the clock signal to both flip-flops. The clock was set to drive the system, controlling when the flip-flops should transition to the next state. Lastly ensured the clock frequency was appropriate for a stable transition.
* Next connected the motor outputs i.e
* Z1 was connected directly to the output of Q1, and Z2 was connected to the output of Q2. Used LEDs as indicators to show when the motors were active, allowing me to visualize whether the motor was on (LED lit) or off (LED not lit).
* After wiring the inputs and outputs, binary switches were added to represent the proximity sensors (S1 and S2). These switches allowed me to toggle between different input combinations to simulate the presence of obstacles.
* Once the circuit was fully wired, the simulation was run. Started by toggling the binary switches (S1, S2) to simulate different obstacle detections and watched how the D flip-flops transitioned between states. The LEDs connected to Z1 and Z2 were connected to check if the motor control signals responded correctly based on the inputs.
* Throughout the simulation, Multisim's oscilloscope and logic analyzer tools were used to monitor the internal signals. This helped me verify that the flip-flops transitioned as expected and that the motor outputs (Z1, Z2) turned on and off according to the robot’s proximity sensor inputs.
* After verifying the system’s behavior, small adjustments was made to the timing of the clock or the logic gates, ensuring the transitions were smooth and the motor outputs were correct.



1. **Task 2 : Asynchronous Design Problem**

This second task involves designing a minimal race-free asynchronous solution for a robot that has a single proximity sensor (Z). The robot’s behavior is to rotate in a sequence: left-right-left, alternating directions based on the sensor signal. When an obstacle is detected (Z=1), the robot continues rotating in the same direction until the sensor signal is lost (Z=0), at which point it switches direction.

***Key Requirements***

1. *Single proximity sensor (Z)*

* *Z = 1:*- The robot detects an obstacle and keeps rotating in the current direction.
* *Z = 0:*- The robot switches direction (left to right or right to left).

1. *Race-free asynchronous design*

The circuit must be designed to avoid races or hazards (unintended transitions), ensuring stable operation during state changes.

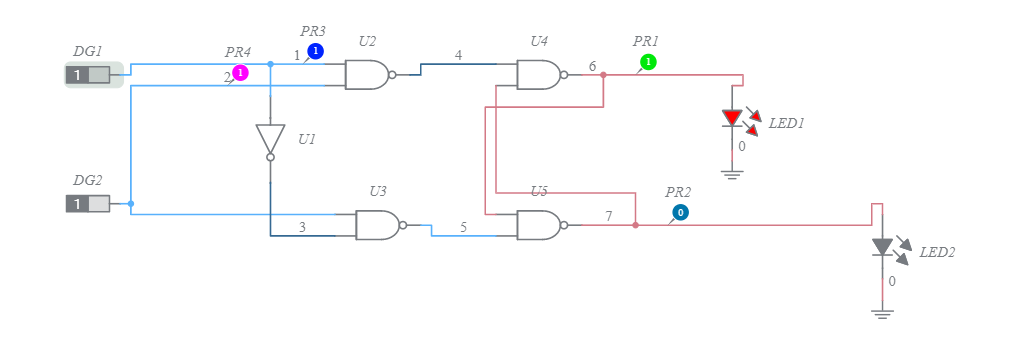
***State Table***

Based on the proximity sensor (Z), the state transition table looks like this;



***State Transition Diagram***

* When Z = 1, the robot continues rotating in its current state (no change in direction).
* When Z = 0, the robot switches to the opposite state (left to right or right to left).



**How the Circuit Works**

* ***When Z = 1***

The inputs to the SR latch do not change, so the robot remains in the current state (either left or right).

* ***When Z = 0***

1. If the robot is currently rotating right (R state), the Set input of the SR latch will be triggered, switching the robot to the left (L state).
2. If the robot is currently rotating left (L state), the Reset input of the SR latch will be triggered, switching the robot to the right (R state).

***Minimizing Race Conditions***

* To avoid race conditions, ensure that the transition logic is stable. This can be done by using de-bouncing circuits or ensuring that there is a clear delay between changing the input (Z) and the state switching, preventing any unwanted glitches.
* Using an SR latch helps eliminate hazards and races, since when there is a clear change in the input (Z = 0) , is when the latch only changes state .