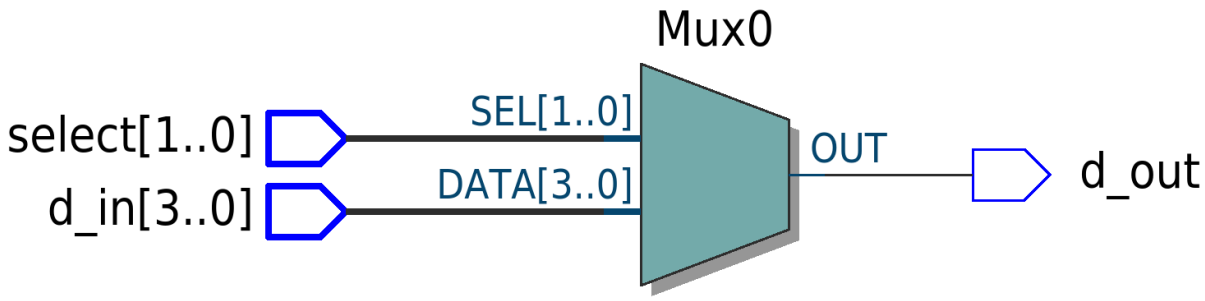
2. Ways of Coding a 4x1 multiplexer

(i) Mux using case:

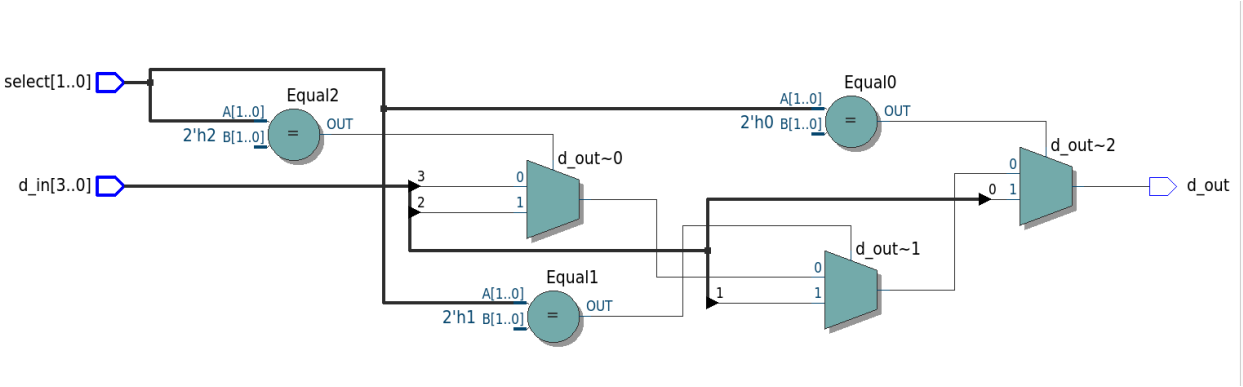
- **Design:**



- ❖ Designed a module named “mux_case” having ports: ‘d_in’ as 4bit input and ‘select’ as 2bit input, and ‘d_out’ as single bit output of multiplexer.
- ❖ Inside an always_comb block, wrote a purely behavioral combinational logic for 4to1 mux using ‘Case’ block.
- ❖ Wrote all the possible cases for select, with a default in case of invalid selection.

(ii) Mux using if-else:

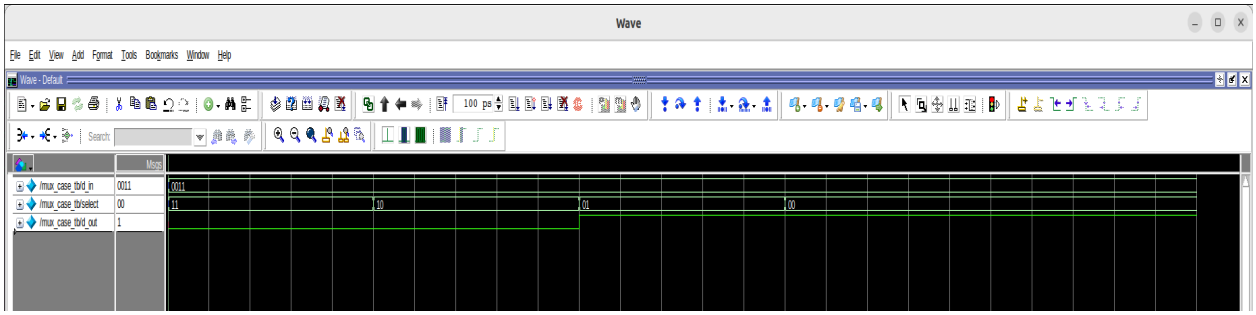
- **Design**



- ❖ Designed a module named “mux_ifelse” having the same ports, i.e. “d_in”, “select”, and “d_out” in multiplexer.
- ❖ Inside an ‘always_comb’ block, wrote the mux logic for every possible select value, using ‘if’, ‘else if’ and ‘else’ keywords.
- ❖ Wrote all cases of select with an else statement in case of invalid select.

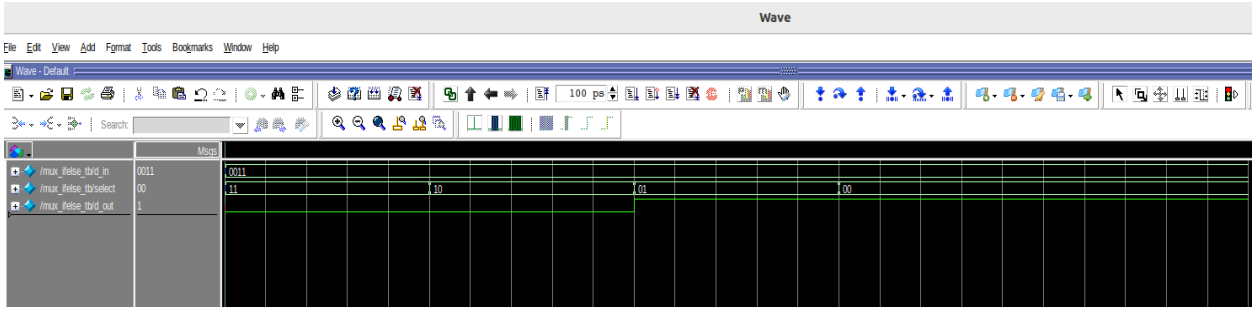
- **Verification**

(i) mux using case:




```
# Lets provide an input of '00_11' to mux
# select = 11, d_out = 0, expected = 0
# PASS
# select = 10, d_out = 0, expected = 0
# PASS
# select = 01, d_out = 1, expected = 1
# PASS
# select = 00, d_out = 1, expected = 1
# PASS
```

(ii) mux using if-else:




```
# Lets provide an input of '00_11' to mux
# select = 11, d_out = 0, expected = 0
# PASS
# select = 10, d_out = 0, expected = 0
# PASS
# select = 01, d_out = 1, expected = 1
# PASS
# select = 00, d_out = 1, expected = 1
# PASS
```

Explanation:

- This testbench verifies the mux_ifelse module by applying a fixed 4-bit input and cycling through all select lines.
- It uses a custom function to compare the output of the mux against the expected bit from d_in[select].
- The loop ensures all select combinations are tested, confirming correct conditional routing logic.
- Verifies the mux's behavior using a compact input pattern and systematic selection sweep.