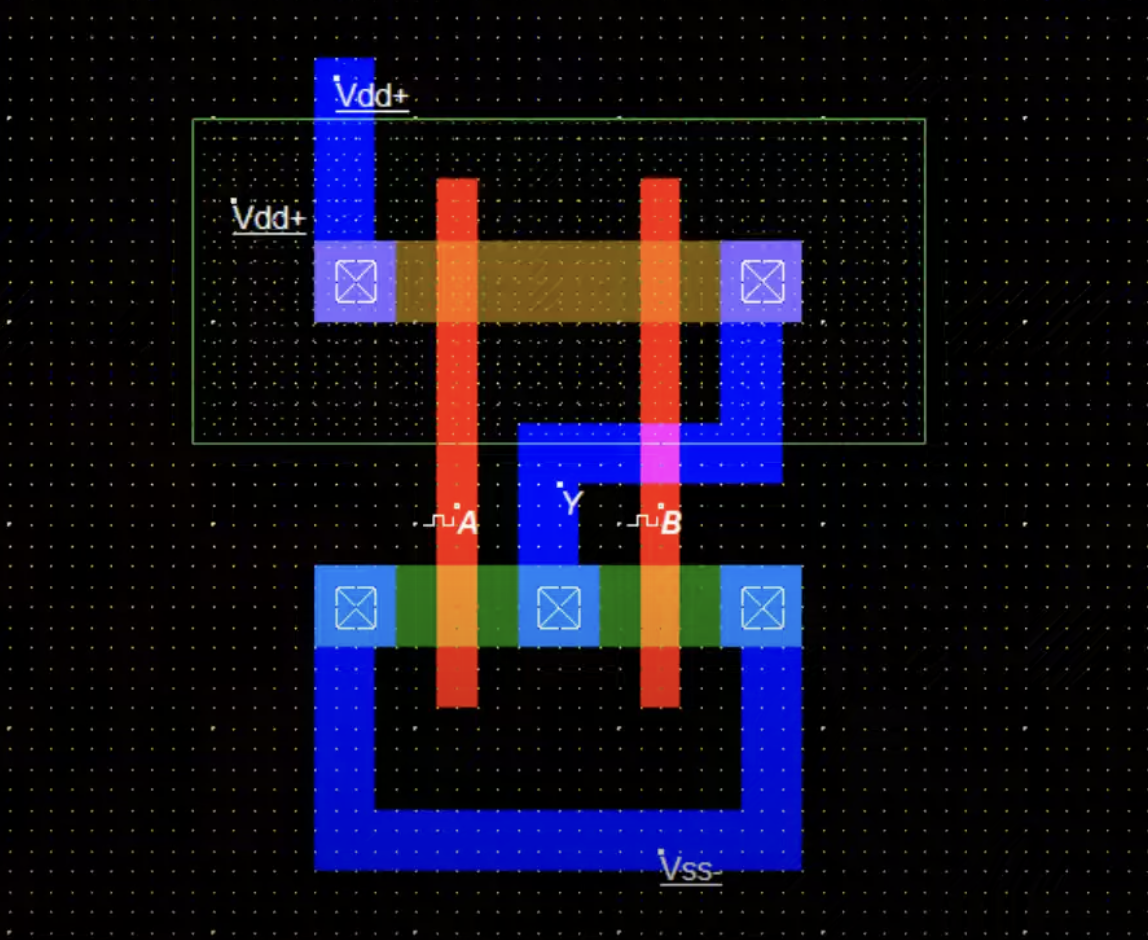
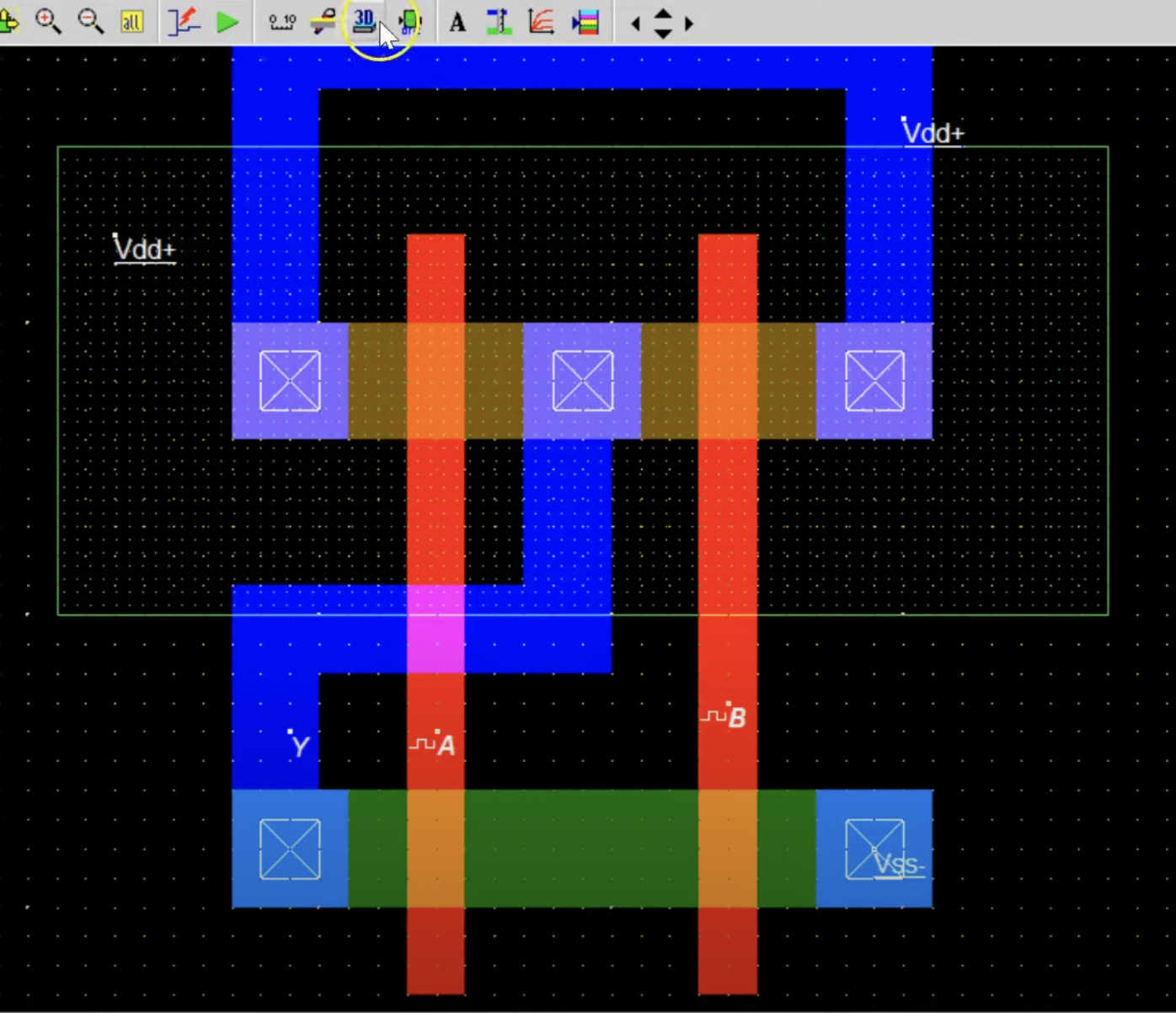
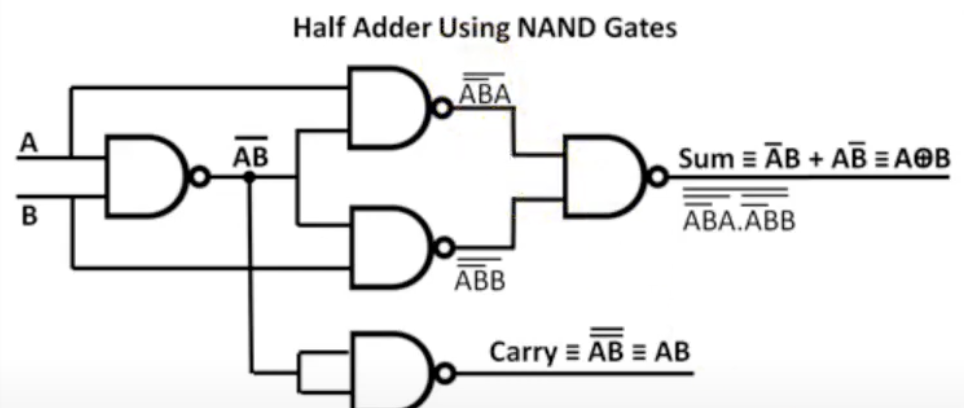
NOR

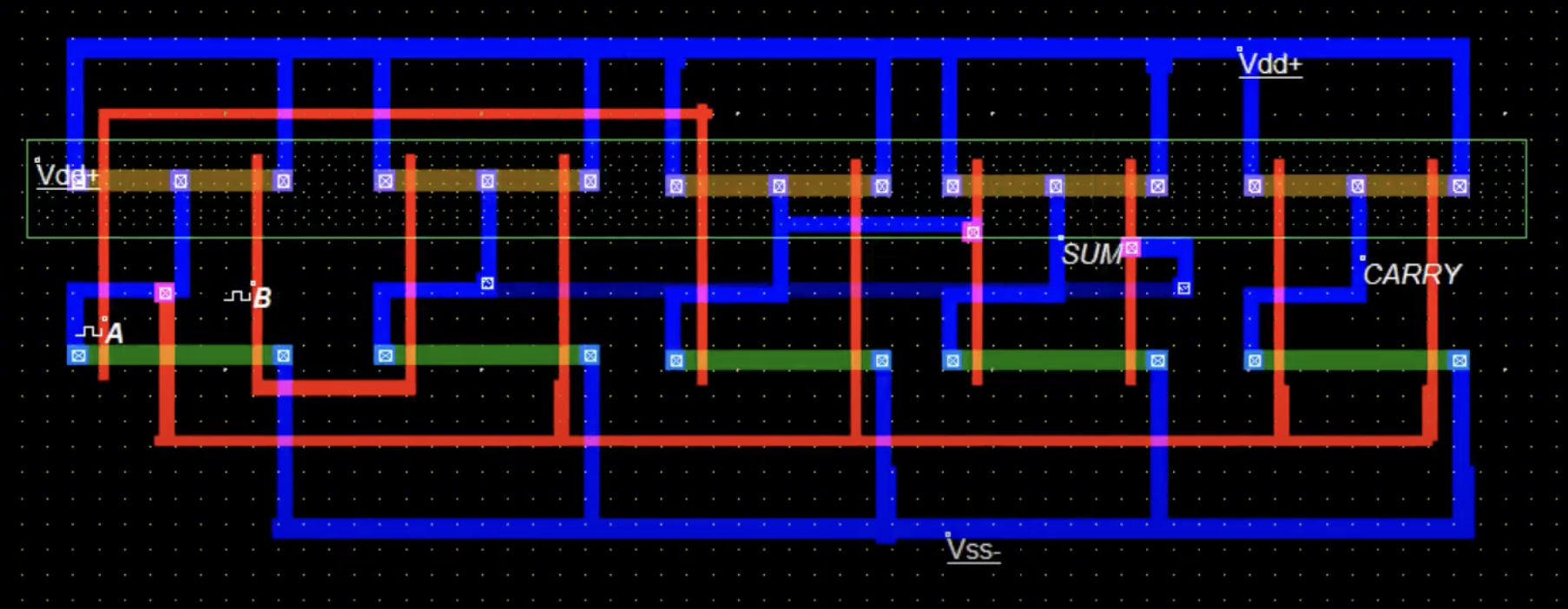


NAND

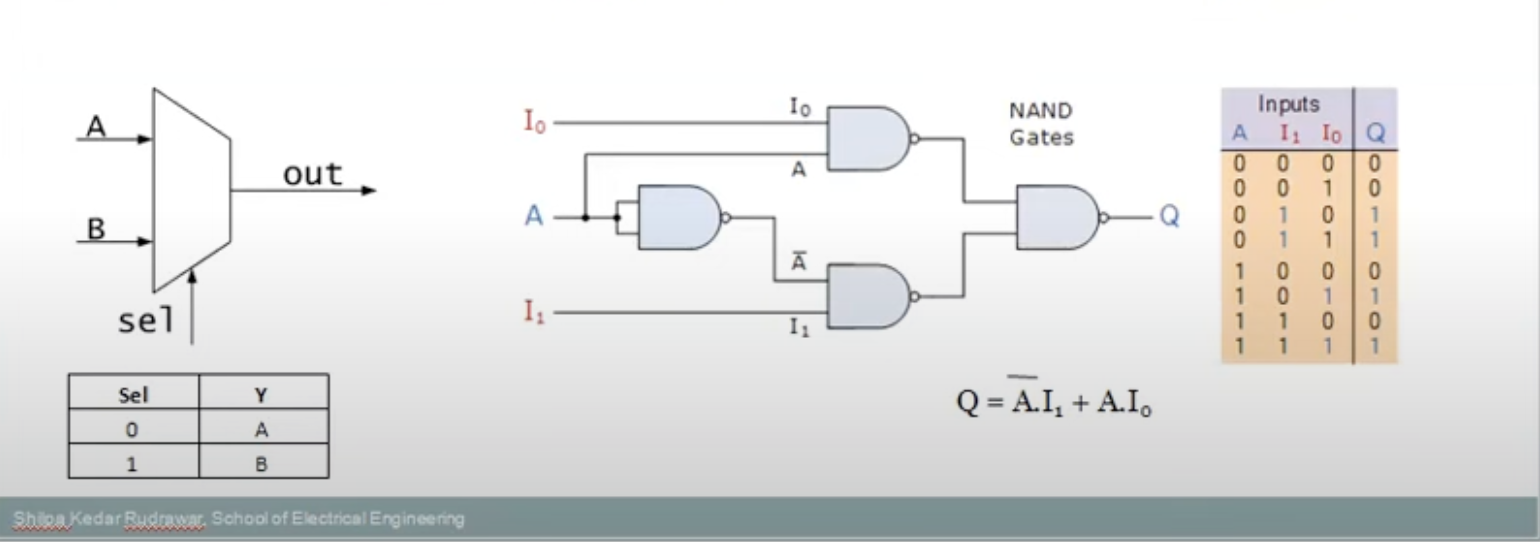


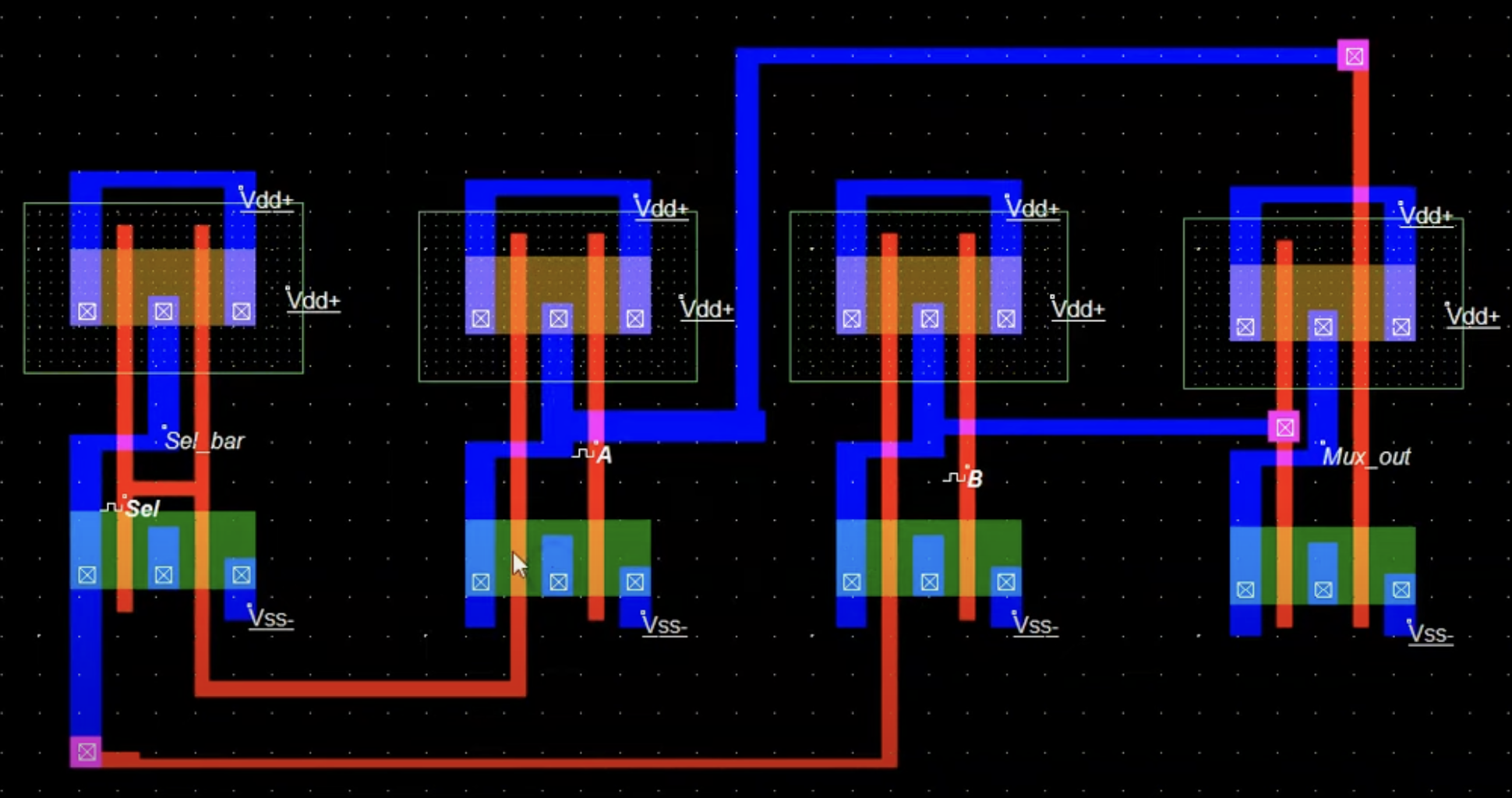
HALF ADDER





2:1 MUX





1. *--------------HALF ADDER---------------------*
2. **library** IEEE;
3. **use** IEEE.STD\_LOGIC\_1164.**ALL**;
5. **entity** HalfAdder **is**
6. **Port** ( A, B : **in** STD\_LOGIC;
7. Sum, Carry : **out** STD\_LOGIC);
8. **end** HalfAdder;
10. **architecture** Behavioral **of** HalfAdder **is**
11. **begin**
12. **process**(A, B)
13. **begin**
14. Sum <= A XOR B;    *-- XOR gives the sum bit*
15. Carry <= A AND B;  *-- AND gives the carry bit*
16. **end** **process**;
17. **end** Behavioral;



22. *------------------ALU--------------------------*
23. **library** IEEE;
24. **use** IEEE.STD\_LOGIC\_1164.**ALL**;
25. **use** IEEE.STD\_LOGIC\_unsigned.**ALL**;
27. **entity** FourBitALU **is**
28. **Port** ( A, B : **in** STD\_LOGIC\_VECTOR(3 **downto** 0);
29. opcode : **in** STD\_LOGIC\_VECTOR(2 **downto** 0);
30. result : **out** STD\_LOGIC\_VECTOR(3 **downto** 0));
31. **end** FourBitALU;
33. **architecture** Behavioral **of** FourBitALU **is**
34. **signal** temp\_result : STD\_LOGIC\_VECTOR(4 **downto** 0); *-- To handle overflow*
35. **begin**
37. **process**(A, B, opcode)
38. **begin**
39. **case** opcode **is**
40. **when** "000" => *-- ADD*
41. temp\_result <= ('0' & A) + ('0' & B);
42. **when** "001" => *-- SUBTRACT*
43. temp\_result <= ('0' & A) - ('0' & B);
44. **when** "010" => *-- AND*
45. temp\_result <= A AND B;
46. **when** "011" => *-- OR*
47. temp\_result <= A OR B;
48. **when** "100" => *-- XOR*
49. temp\_result <= A XOR B;
50. **when** "101" => *-- NAND*
51. temp\_result <= A NAND B;
52. **when** "110" => *-- NOR*
53. temp\_result <= A NOR B;
54. **when** "111" => *-- NOT*
55. temp\_result <= not A;
56. **when** **others** =>
57. temp\_result <= (**others** => '0');
58. **end** **case**;
60. result <= temp\_result(3 **downto** 0); *-- Extract the 4-bit result*
61. **end** **process**;
63. **end** Behavioral;





70. *-------------------COUNTER--------------------------*
71. **library** IEEE;
72. **use** IEEE.STD\_LOGIC\_1164.**ALL**;
73. **use** IEEE.STD\_LOGIC\_unsigned.**ALL**;
75. **entity** UpDownCounter **is**
76. **Port** ( clk : **in** STD\_LOGIC;
77. rst : **in** STD\_LOGIC;
78. up\_down : **in** STD\_LOGIC;
79. count : **out** STD\_LOGIC\_VECTOR(3 **downto** 0));
80. **end** UpDownCounter;
82. **architecture** Behavioral **of** UpDownCounter **is**
83. **signal** counter : STD\_LOGIC\_VECTOR(3 **downto** 0) := "0000";
84. **begin**
86. **process**(clk, rst)
87. **begin**
88. **if** rst = '1' **then**
89. counter <= "0000"; *-- Reset the counter to 0*
90. **elsif** rising\_edge(clk) **then**
91. **if** up\_down = '1' **then**
92. *-- Up counter*
93. counter <= counter + 1;
94. **else**
95. *-- Down counter*
96. counter <= counter - 1;
97. **end** **if**;
98. **end** **if**;
99. **end** **process**;
101. count <= counter;
103. **end** Behavioral;





110. *--------------Seven Segment------------------*
111. **library** IEEE;
112. **use** IEEE.STD\_LOGIC\_1164.**ALL**;
113. **use** IEEE.STD\_LOGIC\_unsigned.**ALL**;
115. **entity** SevenSegmentDisplay **is**
116. **Port** ( clk : **in** STD\_LOGIC;
117. rst : **in** STD\_LOGIC;
118. seg : **out** STD\_LOGIC\_VECTOR(6 **downto** 0);
119. anode : **out** STD\_LOGIC);
120. **end** SevenSegmentDisplay;
122. **architecture** Behavioral **of** SevenSegmentDisplay **is**
123. **signal** counter : INTEGER **range** 0 **to** 9 := 0;
124. **signal** display\_pattern : STD\_LOGIC\_VECTOR(6 **downto** 0);
126. **begin**
128. **process**(clk, rst)
129. **begin**
130. **if** rst = '1' **then**
131. counter <= 0;
132. display\_pattern <= "0000000"; *-- Reset to display pattern for 0*
133. **elsif** rising\_edge(clk) **then**
134. **if** counter < 9 **then**
135. counter <= counter + 1;
136. **else**
137. counter <= 0;
138. **end** **if**;
140. **case** counter **is**
141. **when** 0 => display\_pattern <= "0000001"; *-- 0*
142. **when** 1 => display\_pattern <= "1001111"; *-- 1*
143. **when** 2 => display\_pattern <= "0010010"; *-- 2*
144. **when** 3 => display\_pattern <= "0000110"; *-- 3*
145. **when** 4 => display\_pattern <= "1001100"; *-- 4*
146. **when** 5 => display\_pattern <= "0100100"; *-- 5*
147. **when** 6 => display\_pattern <= "0100000"; *-- 6*
148. **when** 7 => display\_pattern <= "0001111"; *-- 7*
149. **when** 8 => display\_pattern <= "0000000"; *-- 8*
150. **when** 9 => display\_pattern <= "0000100"; *-- 9*
151. **when** **others** => display\_pattern <= "1111111"; *-- All segments off for invalid input*
152. **end** **case**;
153. **end** **if**;
154. **end** **process**;
156. seg <= display\_pattern;
157. anode <= '1'; *-- Anode control signal (common anode display)*
159. **end** Behavioral;





166. *-----------------FIFO-------------------------*
167. **library** IEEE;
168. **use** IEEE.STD\_LOGIC\_1164.**ALL**;
169. **use** IEEE.STD\_LOGIC\_unsigned.**ALL**;
171. **entity** FourBitFIFO **is**
172. **Port** ( clk : **in** STD\_LOGIC;
173. rst : **in** STD\_LOGIC;
174. wr\_en : **in** STD\_LOGIC;
175. rd\_en : **in** STD\_LOGIC;
176. data\_in : **in** STD\_LOGIC\_VECTOR(3 **downto** 0);
177. data\_out : **out** STD\_LOGIC\_VECTOR(3 **downto** 0);
178. empty : **inout** STD\_LOGIC;
179. full : **inout** STD\_LOGIC);
180. **end** FourBitFIFO;
182. **architecture** Behavioral **of** FourBitFIFO **is**
183. **type** MemoryArray **is** **array** (0 **to** 7) **of** STD\_LOGIC\_VECTOR(3 **downto** 0);
184. **signal** fifo\_memory : MemoryArray := (**others** => "0000");
185. **signal** read\_ptr, write\_ptr : integer **range** 0 **to** 7 := 0;
186. **begin**
188. **process**(clk, rst)
189. **begin**
190. **if** rst = '1' **then**
191. read\_ptr <= 0;
192. write\_ptr <= 0;
193. fifo\_memory <= (**others** => "0000");
194. **elsif** rising\_edge(clk) **then**
195. *-- Write operation*
196. **if** wr\_en = '1' and not full = '1' **then**
197. fifo\_memory(write\_ptr) <= data\_in;
198. write\_ptr <= write\_ptr + 1;
199. **end** **if**;
201. *-- Read operation*
202. **if** rd\_en = '1' and not empty = '1' **then**
203. data\_out <= fifo\_memory(read\_ptr);
204. read\_ptr <= read\_ptr + 1;
205. **end** **if**;
206. **end** **if**;
207. **end** **process**;
209. empty <= '1' **when** read\_ptr = write\_ptr **else** '0';
210. full <= '1' **when** ((write\_ptr + 1) mod 8) = read\_ptr **else** '0';
212. **end** Behavioral;




218. *-----------------UNIVERSAL SHIFT REGISTER-----------------------*
219. **library** IEEE;
220. **use** IEEE.STD\_LOGIC\_1164.**ALL**;
222. **entity** UniversalShiftRegister **is**
223. **Port** ( clk : **in** STD\_LOGIC;
224. load : **in** STD\_LOGIC;
225. shift\_left : **in** STD\_LOGIC;
226. shift\_right : **in** STD\_LOGIC;
227. data\_in : **in** STD\_LOGIC\_VECTOR(3 **downto** 0);
228. data\_out : **out** STD\_LOGIC\_VECTOR(3 **downto** 0));
229. **end** UniversalShiftRegister;
231. **architecture** Behavioral **of** UniversalShiftRegister **is**
232. **signal** shift\_register : STD\_LOGIC\_VECTOR(3 **downto** 0) := "0000";
233. **begin**
235. **process**(clk)
236. **begin**
237. **if** rising\_edge(clk) **then**
238. **if** load = '1' **then**
239. shift\_register <= data\_in; *-- Parallel load*
240. **else**
241. **if** shift\_left = '1' **then**
242. shift\_register <= '0' & shift\_register(3 **downto** 1); *-- Left shift*
243. **elsif** shift\_right = '1' **then**
244. shift\_register <= shift\_register(2 **downto** 0) & '0'; *-- Right shift*
245. **end** **if**;
246. **end** **if**;
247. **end** **if**;
248. **end** **process**;
250. data\_out <= shift\_register;
252. **end** Behavioral;