VHDL Bonus Assignment 2
Write-Up Documentation

Aubrey McKinney

UIN: 01243380

ECE 341: Digital System Design

Spring 2025

18 February 2025

1. Starting with creating the simplest Boolean expression we must first create the circuit. Using the specified logic in the power point we can create the following circuit to create and verify our truth table:

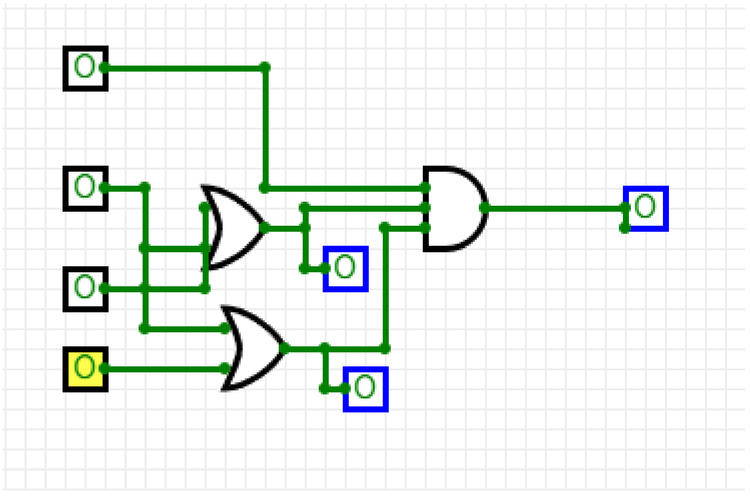


Figure 2.1: Ignition logic (Before Simplification).

Ignition Circuit Results						
Inputs				Outputs		
A (Key)	B (Park)	C (Brake)	D (Belt)	X (park or brake)	Y (park or belt)	Z
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	1	1	0
0	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	0	0	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Now that we have our truth table, we can create a K-map to derive our simplest expression:

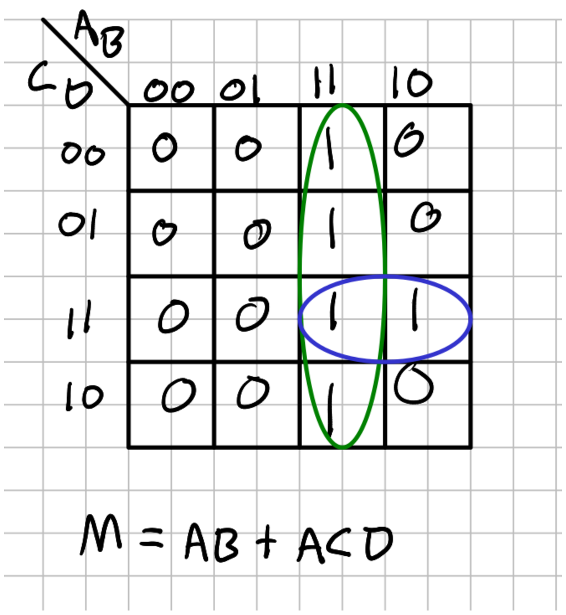
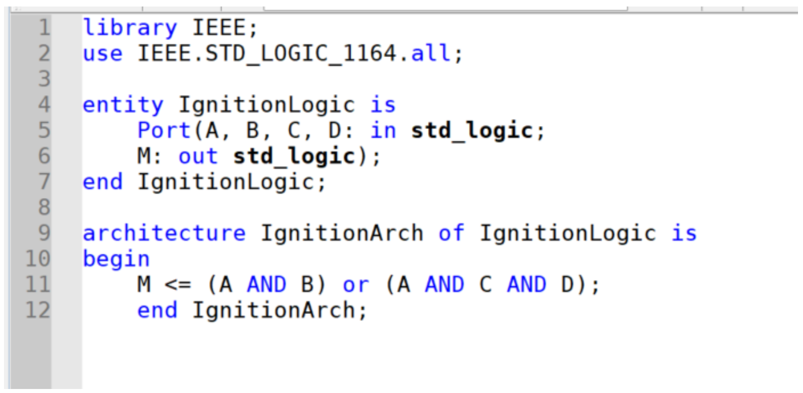


Figure 2.2: Ignition logic simplification.

Now that we have our simplified expression for the ignition logic, we can implement the simplest expression in Aldec.



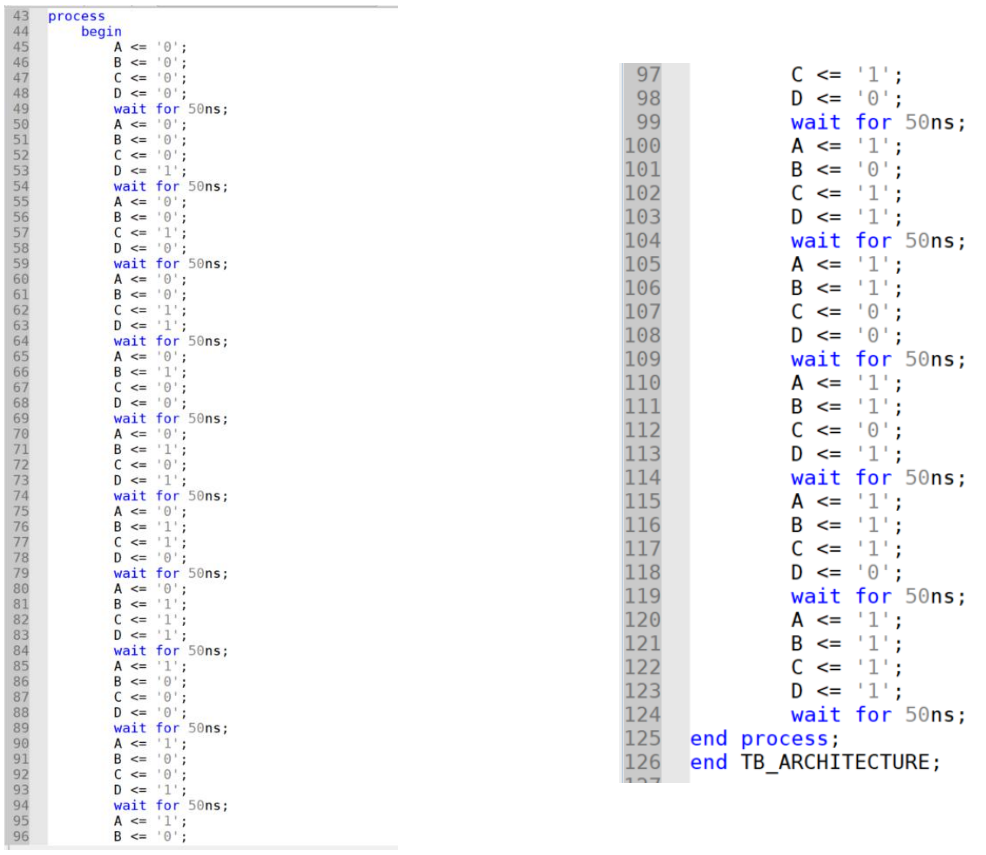
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity IgnitionLogic is
    Port(A, B, C, D: in std_logic;
    M: out std_logic);
end IgnitionLogic;

architecture IgnitionArch of IgnitionLogic is
begin
    M <= (A AND B) or (A AND C AND D);
    end IgnitionArch;
```

Figure 2.3: Ignition logic Assembly program.

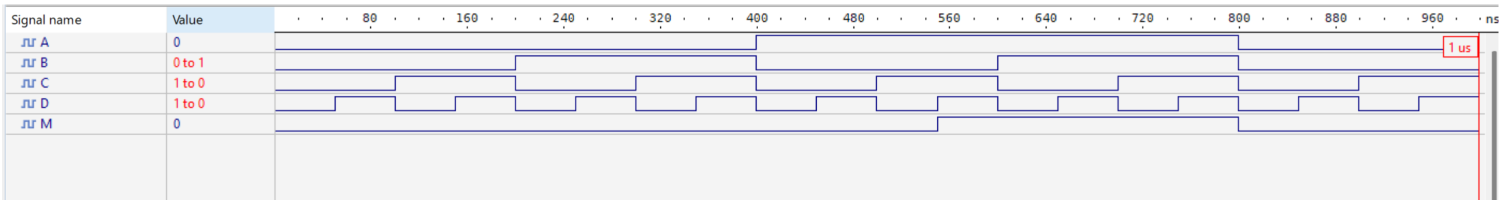
In order to test all of the different outputs against the simplified logic to ensure that it matches the truth table, we can implement the following test bench code.



```vhdl
process
    begin
        A <= '0';
        B <= '0';
        C <= '0';
        D <= '0';
        wait for 50ns;
        A <= '0';
        B <= '0';
        C <= '0';
        D <= '1';
        wait for 50ns;
        A <= '0';
        B <= '0';
        C <= '1';
        D <= '0';
        wait for 50ns;
        A <= '0';
        B <= '0';
        C <= '1';
        D <= '1';
        wait for 50ns;
        A <= '0';
        B <= '1';
        C <= '0';
        D <= '0';
        wait for 50ns;
        A <= '0';
        B <= '1';
        C <= '0';
        D <= '1';
        wait for 50ns;
        A <= '0';
        B <= '1';
        C <= '1';
        D <= '0';
        wait for 50ns;
        A <= '0';
        B <= '1';
        C <= '1';
        D <= '1';
        wait for 50ns;
        A <= '1';
        B <= '0';
        C <= '0';
        D <= '0';
        wait for 50ns;
        A <= '1';
        B <= '0';
        C <= '0';
        D <= '1';
        wait for 50ns;
        A <= '1';
        B <= '0';
        C <= '1';
        D <= '0';
        wait for 50ns;
        A <= '1';
        B <= '0';
        C <= '1';
        D <= '1';
        wait for 50ns;
        A <= '1';
        B <= '1';
        C <= '0';
        D <= '0';
        wait for 50ns;
        A <= '1';
        B <= '1';
        C <= '0';
        D <= '1';
        wait for 50ns;
        A <= '1';
        B <= '1';
        C <= '1';
        D <= '0';
        wait for 50ns;
        A <= '1';
        B <= '1';
        C <= '1';
        D <= '1';
        wait for 50ns;
end process;
end TB_ARCHITECTURE;
```

Figure 2.4: Ignition logic Assembly test bench.

Now that we have a test bench that tests for each one of the possible input combinations, we can generate a waveform and compare the results to our truth table we made previously.



Signal name	Value
A	0
B	0 to 1
C	1 to 0
D	1 to 0
M	0

Figure 2.5: Ignition logic Assembly test bench Waveform.

Comparing our waveform to our truth table, we can see that the wave form and the truth table only outputs a logic 1 for the logic inputs of ACD, AB, ABD, ABC, and ABCD. This concurs with our simplified logic so we can confirm with Aldec Assembly program that the simplified logical expression is:

$$M = (KEY)(PARK) + (KEY)(BRAKE)(BELT)$$

Or

$$M = AB + ACD$$