University of Guelph

ENGG*2410: Digital Design

Sequential Logic Design “Sequence Recognizer Circuit”

via VHDL

Group (# 8)

Layal Alzaydi (1041786)

Josiah Varghese (1043468)

Giuliana Lowry (1013417)

Mazen Hassan (0984322)

Problem Statement

The problem given is to design a sequence recognizer circuit based on a Moore machine using VHDL and schematic methods. The sequence recognizer should recognize ‘1101’ in any set of bit sequence. It should recognize the particular sequence even if it overlaps with another set of sequence. The output z should only equal to ‘1’ when the previous inputs to the circuit are ‘110’ and current input is a ’1’.

Assumptions and Constraints

Some of the assumptions and constraints used in this lab are as follows:

- Using both Schematic Capture and VHDL to implement the design
- Using a Moore Machine for the sequence recognizer
- Drawing the state diagram by hand
- Using an LED for the output of each Flip-Flop

System Overview & Justification of Design

(a) Give an overview of the system to be designed.

The purpose of this lab is to design and use basic operations of sequential logic using VHDL. In this lab, by using knowledge of state diagrams and sequential circuit design, the group has to create a “Sequence Recognizer” using Schematic Capture and VHDL. The Sequence Recognizer should be able to recognize a particular sequence of bits, in this case ‘1101’ and continue till there is no such occurrence of that particular sequence in the longer sequence. The circuit should recognize the particular sequence even if it overlaps with the previous set of the sequence (011**1101**1010100 - first occurrence , 011110**1101**0100 - second occurrence).

(b) Briefly explain how the system works and reasons behind the design.

The sequential recognizer circuit is designed using the VHDL design entry method based in a Moore Machine. The system reads through each bit in the long sequence and recognizes a particular sequence of ‘1101’ by using a states. The system is implemented by using the Moore machine method which outputs values determined by the current state. The Moore Machine method in the system works by going to current state determining if the bit corresponds to that state and if it's true goes to the next state. In cases in which it determines the input is not part of the sequence it resets or goes to the appropriate bit to continue reading the sequence (at State 2, if the bit is not ‘0’ it stays at the same state). By using states, the system stores the past

inputs this makes it possible to overlap/combine states into the fewest number needed. If the sequence is completed, the output is ‘1’ else it is a ‘0’;

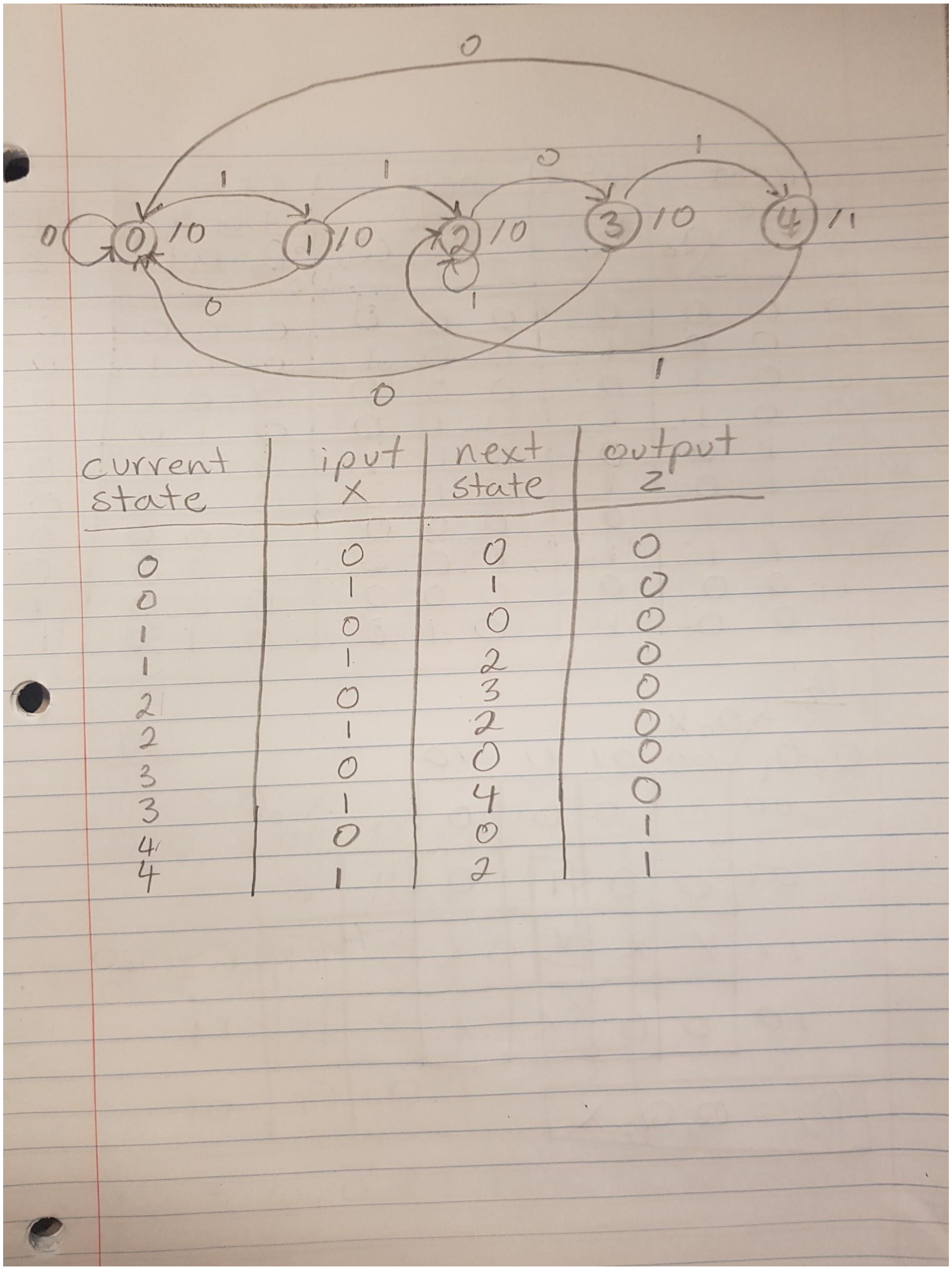
Procedures used to test the circuit

The group used Adept Software in order to implement any VHDL Code and schematic drawing onto the DIGILENT NEXYS 3 FPGA Circuit Board and tested them. Upon completion, there were no errors, and the board reflected the expected results.

Error Analysis

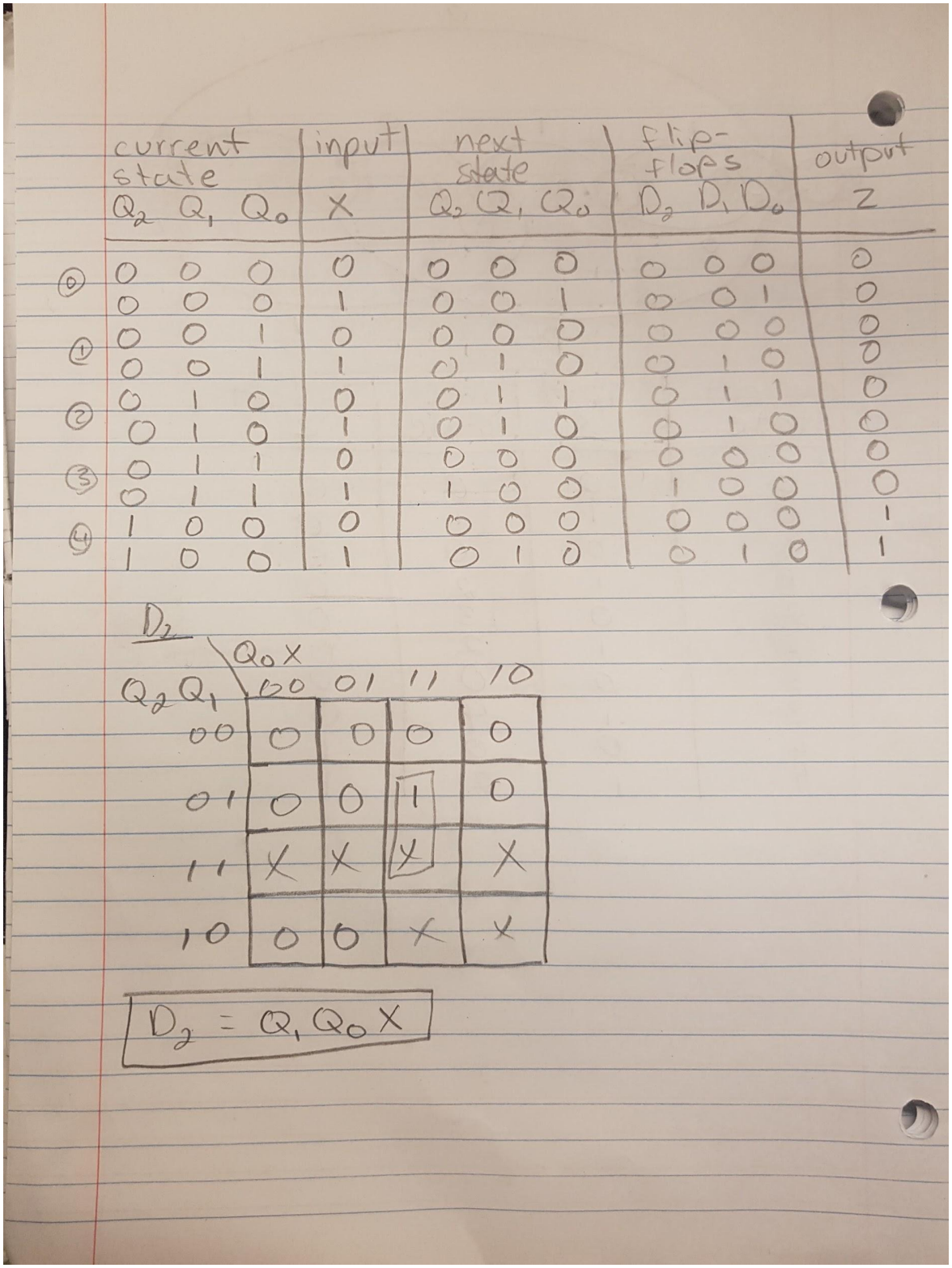
To conclude, the lab was generally error-free, however, the group made some errors while coding the test-bench Code. Once seeing the errors from the compiler and taking advice from the TA, the group decided to edit the portion of the code where the error existed. It then ran successfully.

Appendices



current state	iput X	next state	output Z
0	0	0	0
0	1	1	0
1	0	0	0
1	1	2	0
2	0	3	0
2	1	2	0
3	0	0	0
3	1	4	0
4	0	0	1
4	1	2	1

Figure 1.1 State diagram (top) and state table (bottom)



	current state Q_2 Q_1 Q_0	input X	next state Q_2 Q_1 Q_0	flip-flops D_2 D_1 D_0	output Z
0	0 0 0	0	0 0 0	0 0 0	0
	0 0 0	1	0 0 1	0 0 1	0
1	0 0 1	0	0 0 0	0 0 0	0
	0 0 1	1	0 1 0	0 1 0	0
2	0 1 0	0	0 1 1	0 1 1	0
	0 1 0	1	0 1 0	0 1 0	0
3	0 1 1	0	0 0 0	0 0 0	0
	0 1 1	1	1 0 0	1 0 0	0
4	1 0 0	0	0 0 0	0 0 0	1
	1 0 0	1	0 1 0	0 1 0	1

D_2

Q_2Q_1 \ Q_0X	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	0	0	X	X

$D_2 = Q_1 Q_0 X$

Figure 1.2 State table with flip flops (top), K-map and boolean equation for D2 flip flop (bottom)

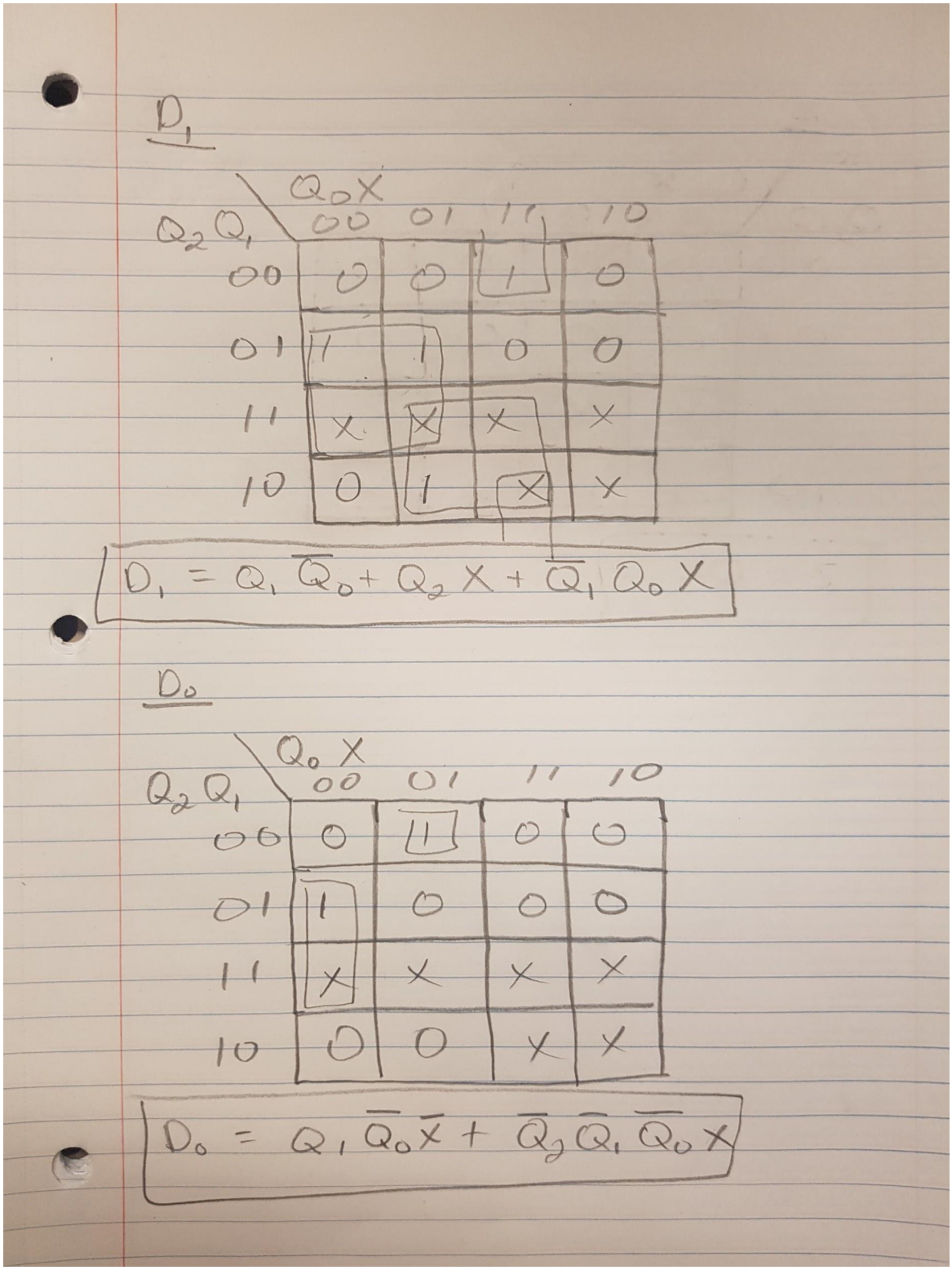


Figure 1.3 K-map and boolean equation for D1 flip flop (top), K-map and boolean equation for D0 flip flop (bottom)

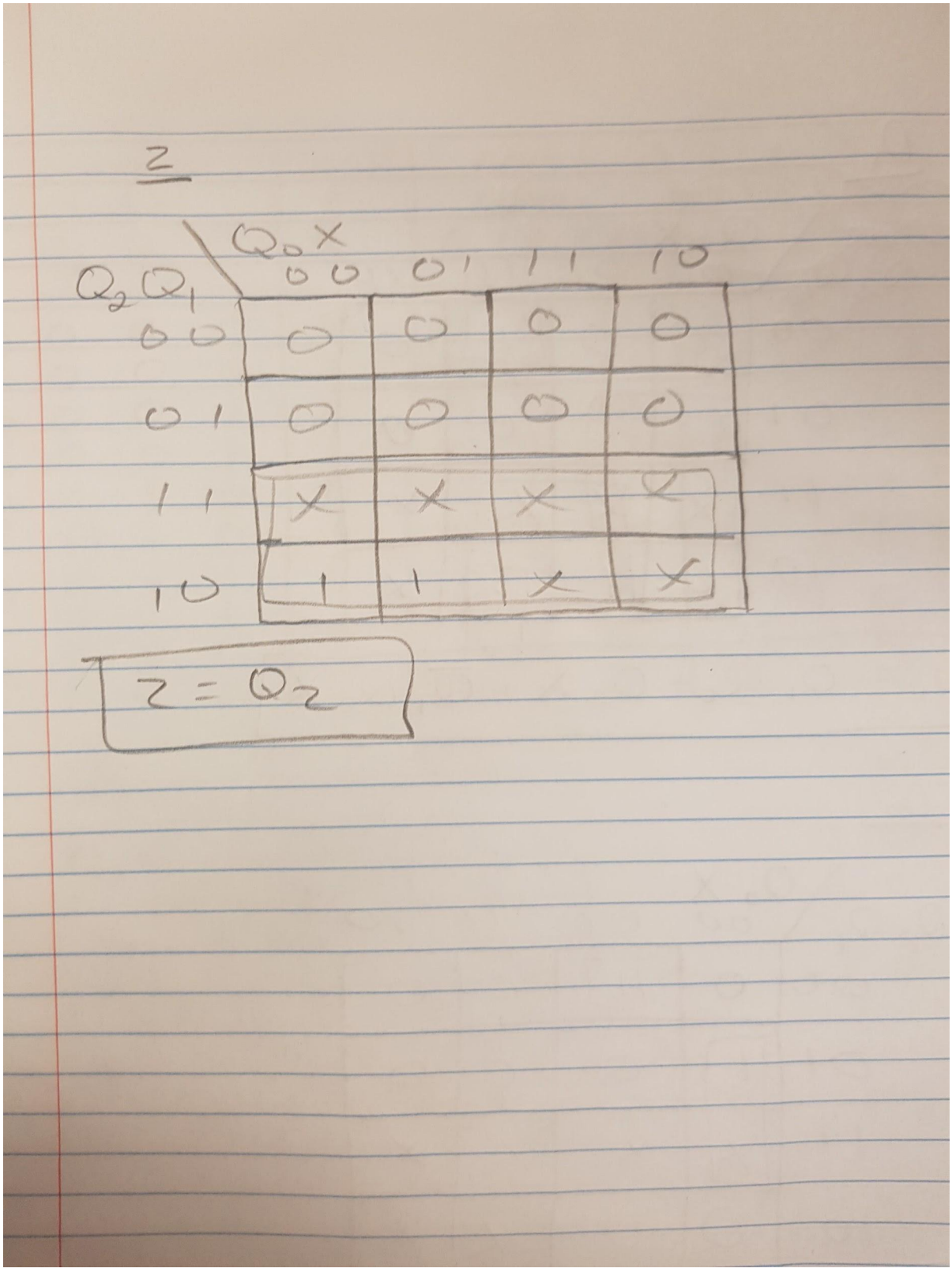


Figure 1.4 K-map and boolean equation for output Z

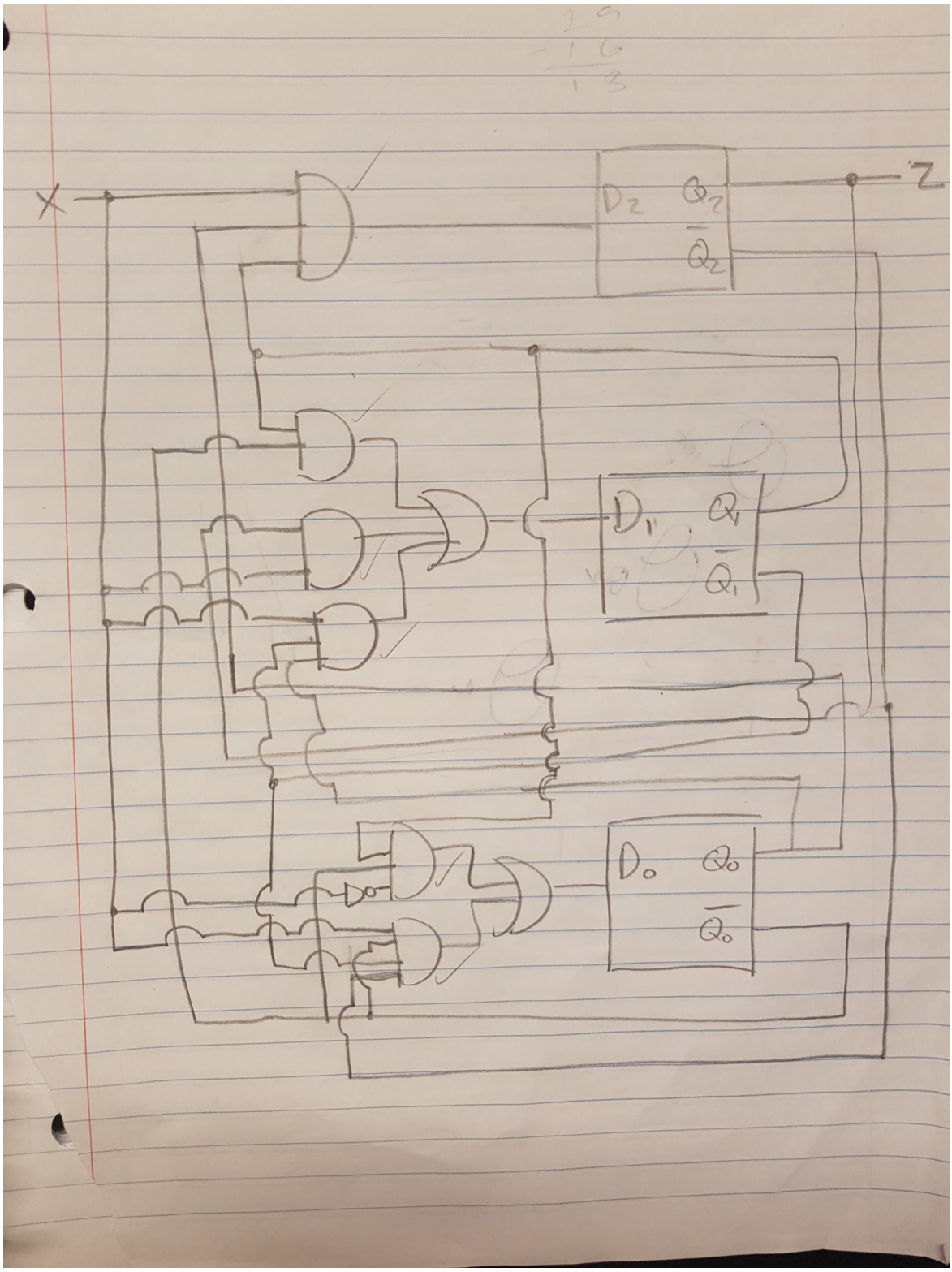


Figure 1.5 Circuit diagram for sequence finder using three D flip flops with input X and output Z

VHDL code

```
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  entity Lab7 is
24      Port ( reset, clk, X : in  STD_LOGIC;
25             Z : out  STD_LOGIC);
26  end entity Lab7;
27
28  architecture Behavioral of Lab7 is
29      type statetype is (state0, state1, state2, state3, state4);
30      signal present_state, next_state: statetype:= state0;
31
32  begin
33  output_process: process(present_state) is
34  begin
35     case present_state is                -- depending upon the current state only
36        when state0 =>                    -- set output signals
37           z <= '0';
38        when state1 =>
39           z <= '0';
40        when state2 =>
41           z <= '0';
42        when state3 =>
43           z <= '0';
44        when state4 =>
45           z <= '1';
46     end case;
```

```
46     end case;
47  end process output_process;
48
49  next_state_process: process(present_state,x) is
50  begin
51     case present_state is               -- depending upon the current state and input
52        when state0 =>                   -- set next state
53           if x = '1' then
54              next_state <= state0;
55           else
56              next_state <= state1;
57           end if;
58
59        when state1 =>
60           if x = '1' then
61              next_state <= state2;
62           else
63              next_state <= state0;
64           end if;
65
66        when state2 =>
67           if x = '1' then
68              next_state <= state2;
69           else
70              next_state <= state3;
71           end if;
72
73        when state3 =>
74           if x = '1' then
```

```
72
73          when state3 =>
74             if x = '1' then
75                next_state <= state4;
76             else
77                next_state <= state0;
78             end if;
79
80          when state4 =>
81             if x = '1' then
82                next_state <= state2;
83             else
84                next_state <= state0;
85             end if;
86       end case;
87    end process next_state_process;
88
89    clk_process: process is
90    begin
91        wait until (rising_edge(clk));        -- wait until the rising edge
92        if reset = '1' then                   -- check for reset and initialize state
93          present_state <= statetype'left;
94        else
95          present_state <= next_state;
96        end if;
97    end process clk_process;
98    end architecture behavioral;
99
100   --end Behavioral;
```

Figure 2 - VHDL code of the sequence detector

Schematic

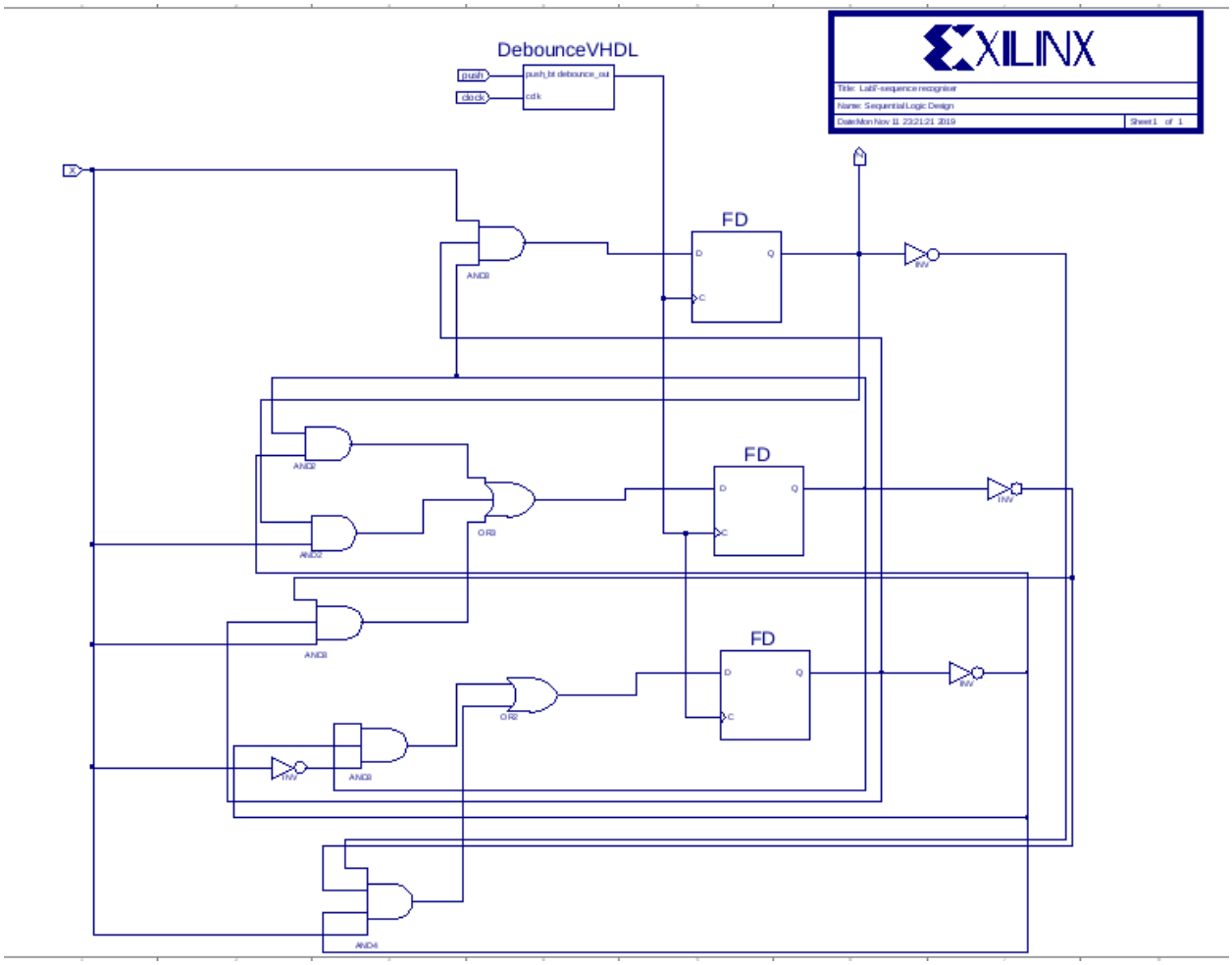


Figure 3 - Schematic Capture of the sequence detector

VHDL for the debounce used in schematic

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23  use IEEE.NUMERIC_STD.ALL;
24
25
26  entity DebounceVHDL is
27      Port ( push_bt : in  STD_LOGIC;
28             cclk : in  STD_LOGIC;
29             debounce_out : out  STD_LOGIC);
30  end DebounceVHDL;
31
32  architecture Behavioral of DebounceVHDL is
33  signal d1, d2, reset, cout : std_logic;
34  signal count : std_logic_vector(20 downto 0);
35
36  begin
37  reset <= d1 xor d2;
38
39  FF: process(cclk)
40  begin
41  if(cclk'event and cclk = '1') then
42  d1 <= push_bt;
43  d2 <= d1;
44  if(cout = '1') then
45  debounce_out <= d2;
46  end if;
47  end if;
48  end process;
49
```

```
35
36  begin
37  reset <= d1 xor d2;
38
39  FF: process(cclk)
40  begin
41  if(cclk'event and cclk = '1') then
42  d1 <= push_bt;
43  d2 <= d1;
44  if(cout = '1') then
45  debounce_out <= d2;
46  end if;
47  end if;
48  end process;
49
50  CNTR: process(cclk, reset)
51  begin
52  if(reset='1') then
53  count <= (others=>'0');
54  elsif (cclk'event and cclk='1') then
55  if (cout = '0') then
56  count <= count + 1;
57  end if;
58  end if;
59  end process;
60
61  cout <= count(20);
62
63  end Behavioral;
64
```

Figure 4 - VHDL code for the debounce used in the schematic capture

Simulation

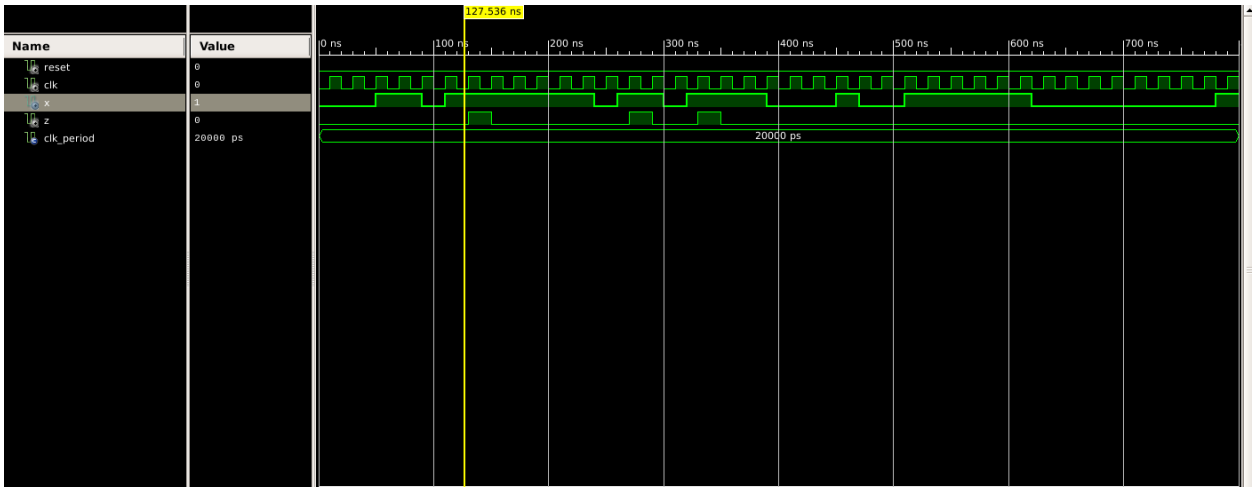


Figure 5 - Simulation

Test Bench

```
27  --------------------------------------------------------------------------------
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY Lab7_TB IS
36  END Lab7_TB;
37
38  ARCHITECTURE behavior OF Lab7_TB IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT Lab7
43      PORT(
44           reset : IN  std_logic;
45           clk : IN  std_logic;
46           X : IN  std_logic;
47           Z : OUT  std_logic
48          );
49      END COMPONENT;
50
51
52     --Inputs
53     signal reset : std_logic := '0';
54     signal clk : std_logic := '0';
55     signal X : std_logic := '0';
56
```

```
52      --Inputs
53      signal reset : std_logic := '0';
54      signal clk : std_logic := '0';
55      signal X : std_logic := '0';
56
57      --Outputs
58      signal Z : std_logic;
59
60      -- Clock period definitions
61      constant clk_period : time := 20 ns;
62
63  BEGIN
64
65      -- Instantiate the Unit Under Test (UUT)
66      uut: Lab7 PORT MAP (
67              reset => reset,
68              clk => clk,
69              X => X,
70              Z => Z
71            );
72
73      -- Clock process definitions
74      clk_process :process
75      begin
76          clk <= '0';
77          wait for clk_period/2;
78          clk <= '1';
79          wait for clk_period/2;
80      end process;
81
```

```
83    -- Stimulus process
84    stim_proc: process
85    begin
86
87     wait for 50 ns;
88       --1101
89       x <= '1';
90       wait for clk_period;
91       x <= '1';
92       wait for clk_period;
93       x <= '0';
94       wait for clk_period;
95       x <= '1';
96       wait for clk_period;
97
98
99        -- hold reset state for 100 ns.
100      wait for 50 ns;
101      --Overlap condition
102      x<= '1';
103      wait for clk_period*3;
104      x<= '0';
105      wait for clk_period;
106      x<= '1';
107      wait for clk_period*2;
108      x <= '0';
109      wait for clk_period;
110      x <='1';
111      wait for clk_period;
112
```

```
113
114         -- hold reset state for 100 ns.
115       wait for 50 ns;
116
117       --Does not work condition
118       x<= '0';
119       wait for clk_period*3;
120       x<= '1';
121       wait for clk_period;
122       x<= '0';
123       wait for clk_period*2;
124       x<= '1';
125       wait for clk_period;
126
127        -- hold reset state for 100 ns.
128       wait for 50 ns;
129
130       --Does not work condition
131       x <= '1';
132       wait for clk_period*2;
133       x <= '0';
134       wait for clk_period*8;
135       x <= '1';
136       wait for clk_period;
137
138     wait;
139   end process;
140
141 END;
```

Figure 6 - Test bench for the sequence detector

The test-bench has 4 base cases, the first was to detect a 1101 case, while the second has the overlap case. Moreover there was 2 cases that did not include a 1101 sequence