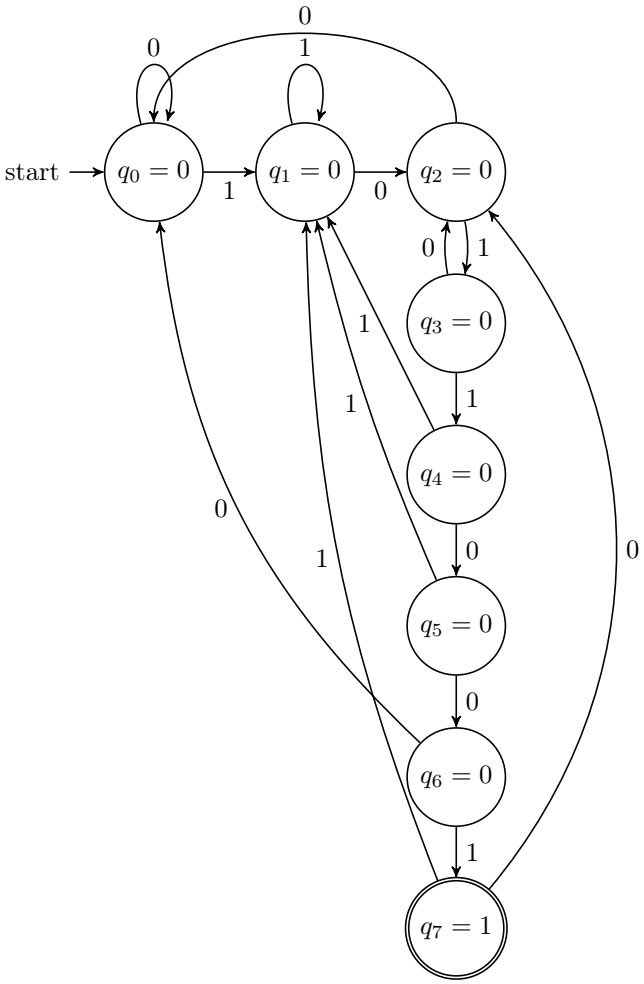
1 Moore Diagram

The personal bit-pattern to be detected by the state-machine in this assignment is $(1001101)_2$. This is interpreted as MSB $\rightarrow$ LSB by convention, and since the RS/EIA-232 standard dictates that LSB be transmitted first, the bit sequence will be in the 'reverse' order (ie. first 1, then 0, then 1 etc...).

A moore machine's outputs are defined by its current state while its inputs define the state-transitions. In this case there is only one input and one output, the current bit being detected and a green indicator LED respectively. The LED will remain LOW until a complete bit-sequence is detected whereupon it is set HIGH.

The state-machine should be able to handle the overlap of two correct sequences, although in this case the only possible overlap is that of the '1's that bookend the sequence, drastically reducing the complexity. To clarify, the only valid overlapping sequence is |101100|1|011001| with the shared '1' in the middle.

The final diagram with the behaviour described is shown below.



2 VHDL Implementation

To implement the Moore-machine above, a CASE/IF structure was used to determine the current state and subsequently transition state depending on the data input.

Clocking on an external signal (a pushbutton on the DE1 board) required additional circuitry in order to detect a rising edge without using the FPGA's internal *rising_edge(CLOCK)* command, which may cause timing instabilities if used in this manner.

```
1  library ieee;
2  use ieee.STD_LOGIC_1164.all;
3
4  entity serial_pattern_match is
5          port (CLOCK_50 : in std_logic; -- 50MHz internal clock.
6          btn : in std_logic; -- Manual clock signal using external PUSHBUTTON's rising edge. Inverted LOGIC.
7          reset : in std_logic; -- RESET signal tied to external PUSHBUTTON. Inverted LOGIC.
8          data : in std_logic; -- Current DATA bit set using external switch.
9
10         LEDG7 : out std_logic; -- Indicates successful pattern detection when HIGH.
11         LEDR : out std_logic_vector(7 downto 0)); -- Indicates current STATE
12 end entity;
13
14 architecture rtl of serial_pattern_match is
15         type stateType is (U,Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0); -- 8 states are needed to implement the Moore-machine.
16         signal state : stateType;
17         signal prevBtn : std_logic; -- Used to detect rising edge on 'btn'.
18                 -- Since we can't edge-trigger directly on a non-internal-clock signal,
19         -- we use a synchronous flip-flop to detect if a rising edge has occured since last internal-clock cycle.
20 begin
21         ---------------------------------------------------------------------------------------------
22         -- Process: proc_moore_machine
23         -- Description: Moore-machine implementation of algorithm detecting a specific pattern in a data-stream.
24         --                                  Pattern is (in chronological order ->): 1011001.
25
26         -- Input(s)            : CLOCK_50, btn, reset, data
27         -- Output(s)            : LEDG7, LEDR
28         -- Internal Signals     : state, prevBtn
29         ---------------------------------------------------------------------------------------------
30         proc_moore_machine:process(CLOCK_50, reset)
31         begin
32                 if reset = '0' then -- asynchronous reset.
33                         state <= Q0; -- load inital state.
34                         LEDR <= (others => '0'); -- set all indicator LEDs LOW.
35                         LEDG7 <= '0';
36                 elsif (prevBtn = '0' AND btn = '1') then  -- if rising_edge(btn) then
37                         case state is
38                                 when Q0 =>
39                                         LEDR <= "00000001";
40                                         if data = '1' then
41                                                 state <= Q1;
42                                                 LEDR <= "00000010";
43                                         else
44                                                 -- stay in current state.
45                                         end if;
46                                 when Q1 =>
47                                         if data = '0' then
48                                                 state <= Q2;
49                                                 LEDR <= "00000100";
50                                         else
51                                                 -- stay in current state.
52                                         end if;
53                                 when Q2 =>
54                                         if data = '1' then
55                                                 state <= Q3;
56                                                 LEDR <= "00001000";
57                                         else
58                                                 state <= Q0;
59                                         end if;
60                                 when Q3 =>
61                                         if data = '1' then
```

```
62                                              state <= Q4;
63                                              LEDR <= "00010000";
64                                      else
65                                              state <= Q2;
66                                      end if;
67                              when Q4 =>
68                                      if data = '1' then
69                                              state <= Q1;
70                                      else
71                                              state <= Q5;
72                                              LEDR <= "00100000";
73                                      end if;
74                              when Q5 =>
75                                      if data = '1' then
76                                              state <= Q1;
77                                      else
78                                              state <= Q6;
79                                              LEDR <= "01000000";
80                                      end if;
81                              when Q6 =>
82                                      if data = '1' then
83                                              state <= Q7;
84                                              LEDR <= "10000000";
85                                              LEDG7 <= '1'; -- Transition to last state means full sequence detected!
86                                      else
87                                              state <= Q0;
88                                      end if;
89                              when Q7 =>
90                                      if data = '1' then
91                                              state <= Q1;
92                                              LEDG7 <= '0';
93                                              LEDR <= "00000010";
94                                      else
95                                              state <= Q2;
96                                              LEDG7 <= '0';
97                                              LEDR <= "00000100";
98                                      end if;
99                              when others => -- when 'U'
100                                     -- do nothing
101                     end case;
102             end if;
103     end process;
104
105     ---------------------------------------------------------------------------------------------
106     -- Process: proc_rising_edge_detect
107     -- Description: Saves last PUSHBUTTON state.
108     --              Allows detection of state change through comparison with current state.
109
110     -- Input(s)               : btn
111     -- Internal Signals       : prevBtn
112     ---------------------------------------------------------------------------------------------
113     proc_rising_edge_detect:process(CLOCK_50)
114     begin
115             if rising_edge(CLOCK_50) then
116                     prevBtn <= btn;
117             end if;
118     end process;
119 end architecture;
```

John Croft
19930814-7959

3 VHDL Simulation

The above VHDL code was simulated and tested using the following input bitstream:

$$10010\overbrace{1011001011001}^{\text{2 overlapping patterns}}10000111\overbrace{1011001}^{\text{1 pattern}}$$

where bits are read in from left to right.

The results of this can be seen in figure 1, which shows the various state transitions in the Moore-machine as well as an indication of the three successful pattern matches on signal *LEDG7*.

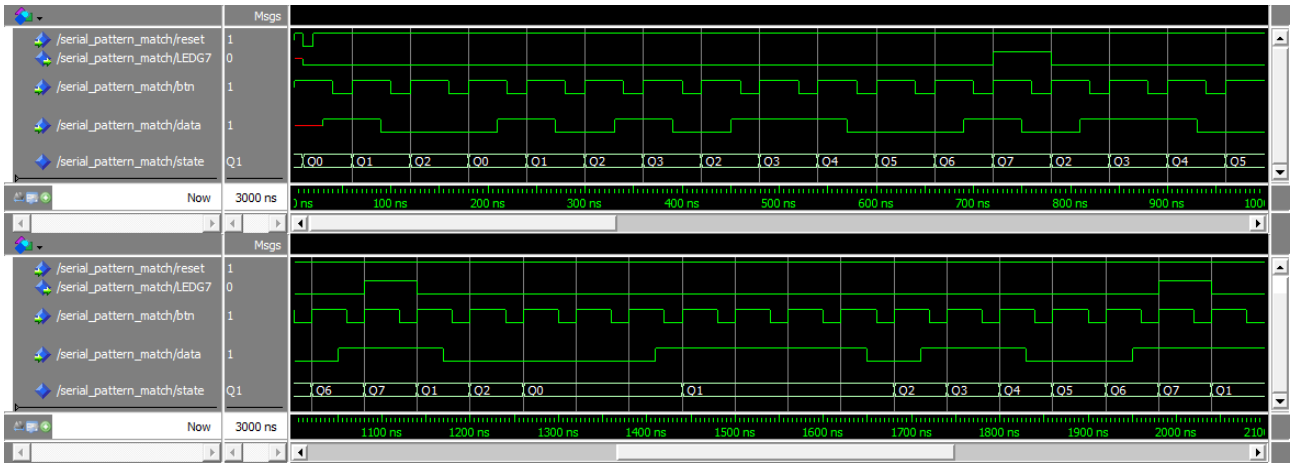


Figure 1: *VHDL code simulation in modelsim. Note that irrelevant signals have been omitted.*

4 Pin Table for DE1

Name	Location	DE1 Name
CLOCK_50	PIN_L1	CLOCK_50
data	PIN_L22	SW[0]
btn	PIN_R22	KEY[0]
reset	PIN_T21	KEY[3]
LEDG7	PIN_Y21	LEDG[7]
LEDR[7]	PIN_U18	LEDR[7]
LEDR[6]	PIN_Y18	LEDR[6]
LEDR[5]	PIN_V19	LEDR[5]
LEDR[4]	PIN_T18	LEDR[4]
LEDR[3]	PIN_Y19	LEDR[3]
LEDR[2]	PIN_U19	LEDR[2]
LEDR[1]	PIN_R19	LEDR[1]
LEDR[0]	PIN_R20	LEDR[0]