Programable Logic Circuts
Lab 06

Sølve Kjelseth

September 5, 2025

0. Introduction

This is the sixth report in this course, detailing the completion of the sixth lab exercise. With the purpose is to investigate Finite State Machines (FSM).

Note: As always, the LaTeX code is open source, see my GitHub

0.0. Table of Contents

1. Part 1 **3**
1.0. Solving 3
1.1. Code 4
1.2. RTL 8
1.3. Results 10

2. Part 2 **11**
2.0. Solving 11
2.1. Code 11
2.2. RTL 14
2.3. Results 16

0.1. List of Figures

Code 1.0: TLE with the first type of one-hot 4
Code 1.1: State register first type 5
Code 1.2: TLE with the second type of one-hot 6
Code 1.3: State register second type 7
Figure 1.0: RTL of the TLE with the first type one-hot encoding 8
Figure 1.1: RTL of the State register with non zero reset 8
Figure 1.2: RTL of the TLE with the second type one-hot encoding 9
Figure 1.3: RTL of the State register with true zero reset 9
Code 2.0: TLE for the behavioural FSM 11
Figure 2.0: RTL of the TLE 14
Figure 2.1: State diagram 14

0.2. List of Tables

Table 1.0: State change table 3
Table 2.0: Bit encoding output with sequential type 15
Table 2.1: Bit encoding output with one-hot type 15

1. Part 1

This Part is about making a FSM using structural assignments, it is kind of divided into two parts with different types of one-hot encoding, one where the reset state, A, is `000000001`, and one where A is `000000000` but for any other state the last bit is `1` in addition to the encoding one-hot bit.

1.0. Solving

Solving this task was done by making table 1.0 and finding a boolean expression for each bit. Then the output was connected directly to the encoding to display it on the first 9 LED's. The final 10th LED was connected to the states that signifies on output of the system. For the the part with the different one-hot encoding only a few adjustments was needed.

Table 1.0: State change table

Current state										Input	New state									
Name	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	w	Name	Y8'	Y7'	Y6'	Y5'	Y4'	Y3'	Y2'	Y1'	Y0'
A	0	0	0	0	0	0	0	0	1	0	B	0	0	0	0	0	0	0	1	0
A	0	0	0	0	0	0	0	0	1	1	F	0	0	0	1	0	0	0	0	0
B	0	0	0	0	0	0	0	1	0	0	C	0	0	0	0	0	0	1	0	0
B	0	0	0	0	0	0	0	1	0	1	F	0	0	0	1	0	0	0	0	0
C	0	0	0	0	0	0	1	0	0	0	D	0	0	0	0	0	1	0	0	0
C	0	0	0	0	0	0	1	0	0	1	F	0	0	0	1	0	0	0	0	0
D	0	0	0	0	0	1	0	0	0	0	E	0	0	0	0	1	0	0	0	0
D	0	0	0	0	0	1	0	0	0	1	F	0	0	0	1	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0	0	E	0	0	0	0	1	0	0	0	0
E	0	0	0	0	1	0	0	0	0	1	F	0	0	0	1	0	0	0	0	0
F	0	0	0	1	0	0	0	0	0	0	B	0	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0	0	1	G	0	0	1	0	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0	0	B	0	0	0	0	0	0	0	1	0
G	0	0	1	0	0	0	0	0	0	1	H	0	1	0	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0	0	B	0	0	0	0	0	0	0	1	0
H	0	1	0	0	0	0	0	0	0	1	I	1	0	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0	0	B	0	0	0	0	0	0	0	1	0
I	1	0	0	0	0	0	0	0	0	1	I	1	0	0	0	0	0	0	0	0

1.1. Code

Code 1.0: TLE with the first type of one-hot

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity L06P01 is
    port(SW   : in  std_logic_vector(1 downto 0);
         KEY  : in  std_logic_vector(3 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end entity;


architecture structural of L06P01 is

    signal Vcc, w, z        : std_logic;
    signal NewState, State : std_logic_vector(8 downto 0);

begin

    Vcc               <= '1';
    w                 <= SW(1);
    LEDR(9)           <= z;
    LEDR(8 downto 0) <= State;

    process(State) is
    begin
        z           <= State(4) or State(8);
        NewState(0) <= '0';
        NewState(1) <= (not w) and (State(0) or State(5) or
                        State(6) or State(7) or State(8));
        NewState(2) <= (not w) and State(1);
        NewState(3) <= (not w) and State(2);
        NewState(4) <= (not w) and (State(3) or State(4));
        NewState(5) <= w and (State(0) or State(1) or
                        State(2) or State(3) or State(4));
        NewState(6) <= w and State(5);
        NewState(7) <= w and State(6);
        NewState(8) <= w and (State(7) or State(8));

    end process;


    StateRegister: entity work.StateRegister(structural)
        port map(Clk      => KEY(3),
                 nRst     => SW(0),
                 Enable   => Vcc,
                 NewState => NewState,
                 State    => State);

end architecture;
```

Code 1.1: State register first type

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity StateRegister is
    port(Clk      : in  std_logic;
         nRst     : in  std_logic;
         Enable   : in  std_logic;
         NewState : in  std_logic_vector(8 downto 0);
         State    : out std_logic_vector(8 downto 0));
end entity;


architecture structural of StateRegister is

begin

    process(Clk) is
    begin
        if rising_edge(Clk) then
            if nRst = '0' then
                State <= "000000001";
            else
                if Enable = '1' then
                    State <= NewState;
                end if;
            end if;
        end if;
    end process;

end architecture;
```

The only thing changed for the second type is a few assignments in the TLE and the register uses a true zero reset instead of the last bit as `1`.

Code 1.2: TLE with the second type of one-hot

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity L06P01 is
    port(SW   : in  std_logic_vector(1 downto 0);
         KEY  : in  std_logic_vector(3 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end entity;


architecture structural of L06P01 is

    signal Vcc, w, z       : std_logic;
    signal NewState, State : std_logic_vector(8 downto 0);

begin

    Vcc              <= '1';
    w                <= SW(1);
    LEDR(9)          <= z;
    LEDR(8 downto 0) <= State;

    process(State) is
    begin
        z           <= State(4) or State(8);
        NewState(0) <= '1';
        NewState(1) <= (not w) and ((not State(0)) or State(5) or
                       State(6) or State(7) or State(8));
        NewState(2) <= (not w) and State(1);
        NewState(3) <= (not w) and State(2);
        NewState(4) <= (not w) and (State(3) or State(4));
        NewState(5) <= w and ((not State(0)) or State(1) or
                       State(2) or State(3) or State(4));
        NewState(6) <= w and State(5);
        NewState(7) <= w and State(6);
        NewState(8) <= w and (State(7) or State(8));

    end process;


    StateRegister: entity work.StateRegister(structural)
        port map(Clk      => KEY(3),
                 nRst     => SW(0),
                 Enable   => Vcc,
                 NewState => NewState,
                 State    => State);

end architecture;
```

Code 1.3: State register second type

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity StateRegister is
    port(Clk      : in  std_logic;
         nRst     : in  std_logic;
         Enable   : in  std_logic;
         NewState : in  std_logic_vector(8 downto 0);
         State    : out std_logic_vector(8 downto 0));
end entity;


architecture structural of StateRegister is

begin

    process(Clk) is
    begin
        if rising_edge(Clk) then
            if nRst = '0' then
                State <= "000000000";
            else
                if Enable = '1' then
                    State <= NewState;
                end if;
            end if;
        end if;
    end process;

end architecture;
```

1.2. RTL

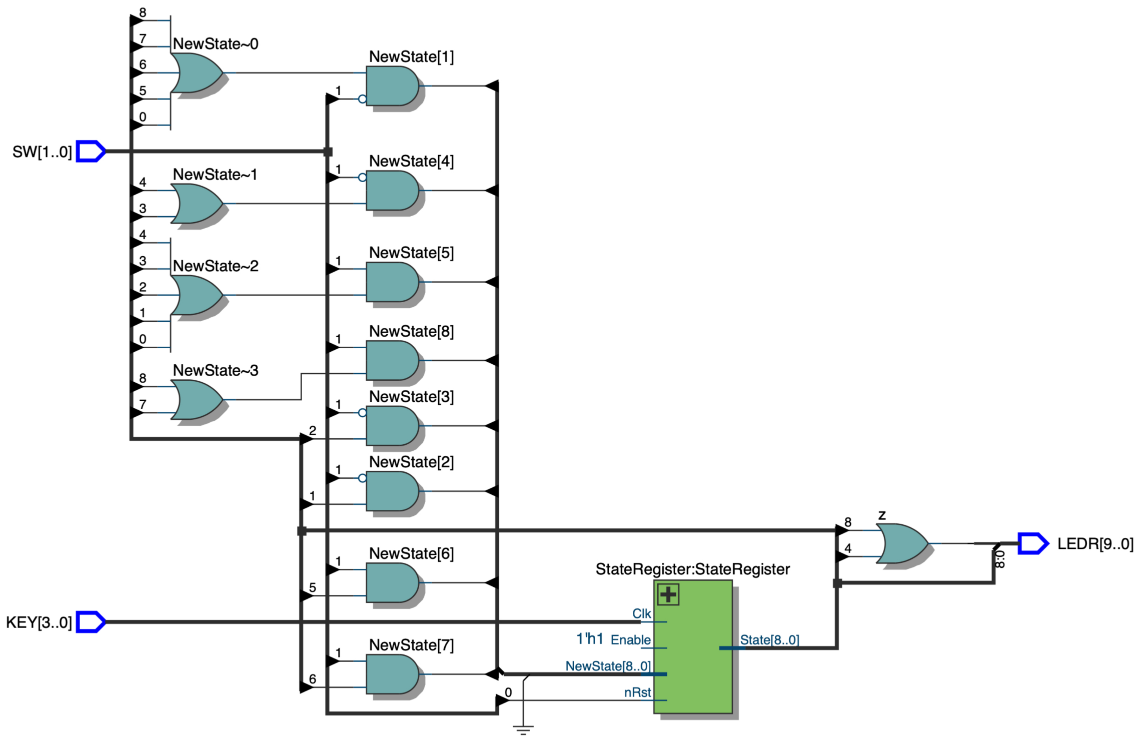

Figure 1.0: RTL of the TLE with the first type one-hot encoding

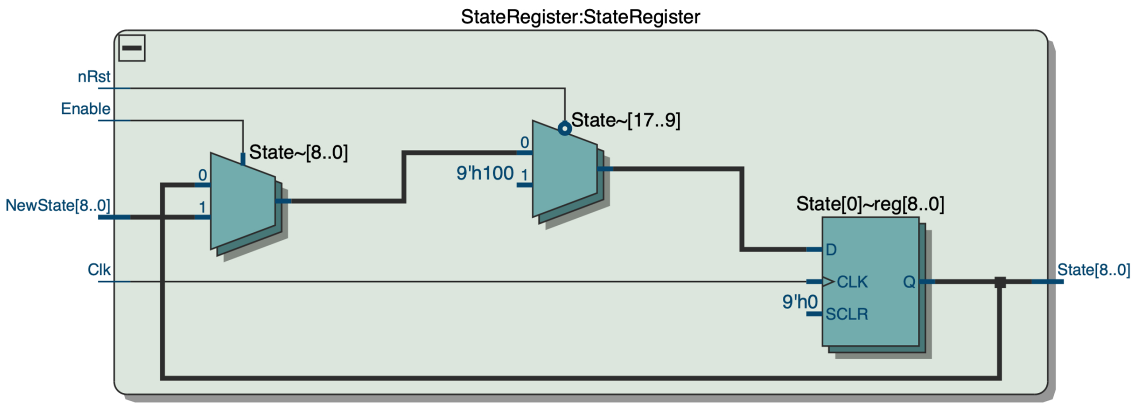

Figure 1.1: RTL of the State register with non zero reset

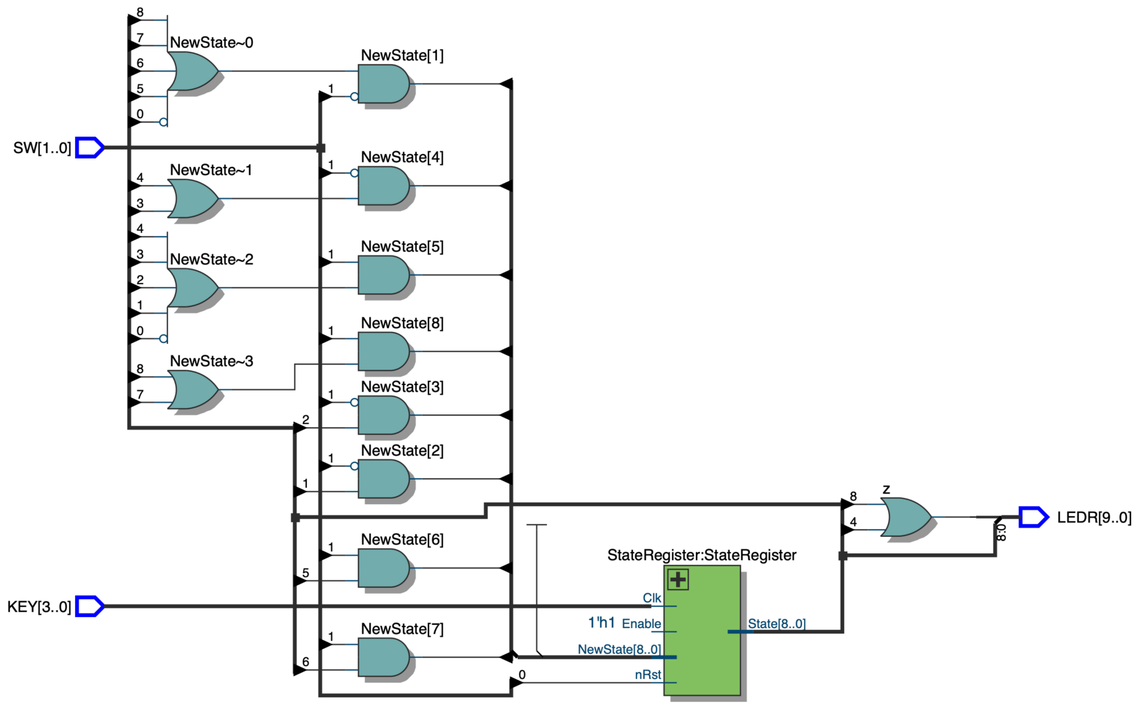

Figure 1.2: RTL of the TLE with the second type one-hot encoding

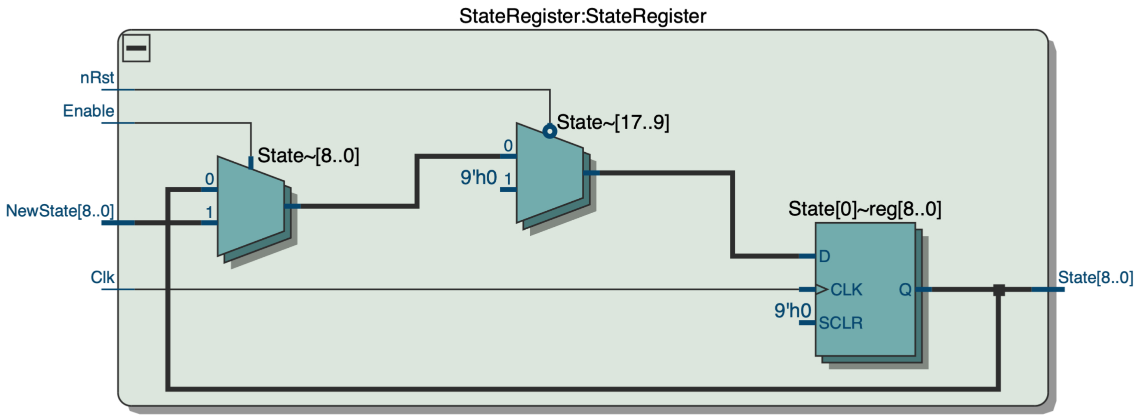

Figure 1.3: RTL of the State register with true zero reset

As clearly visible there is not much difference between these two types of encoding.

1.3. Results

Showing the function of the circuit is hard with photos, therefore videos is uploaded to YouTube. Here is the video showing the first encoding type and the second encoding type

In case linking is not working, the full URLs is below:

https://youtu.be/Wk256vCVhfY

https://youtu.be/uTf9eZYBo-c

2. Part 2

This Part is about making a FSM using behavioural architecture, and the encoding should be binary. The complier info for the encoding could be checked and then change the encoding to one-hot and check again.

2.0. Solving

Solving this task was done by making a new variable type for the different states then making a State variable and setting it to the different states using a case when system. This was done three times, one case when for changing state, here a custom procedure was implemented, not necessary but increases readability. Then a case when for the output, which is only on for the two ”last” states. Then a final output case when is only made for displaying the current state encoding on the first 4 LED’s. `type is "sequential"` forces binary encoding. This was simply replaced with `type is "one-hot"` for checking compilation with this different encoding.

2.1. Code

Code 2.0: TLE for the behavioural FSM

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity L06P02 is
    port(SW   : in  std_logic_vector(1 downto 0);
         KEY  : in  std_logic_vector(3 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end entity;


architecture behavioural of L06P02 is

    type TaskState is (A, B, C, D, E, F, G, H, I);
    attribute syn_encoding : string;
    attribute syn_encoding of TaskState : type is "sequential";
    signal State             : TaskState;

    signal w, z              : std_logic;
    signal ShowEncoding      : std_logic_vector(3 downto 0);

begin

    w                 <= SW(1);
    LEDR(9)           <= z;
    LEDR(8 downto 4) <= (others => '0');
    LEDR(3 downto 0) <= ShowEncoding;

    process(KEY(3)) is

        procedure ChangeState(wantedValue : std_logic;
                              toState     : TaskState;
                              elseState   : TaskState) is
        begin
```

```vhdl
            if w = wantedValue then
                State <= toState;
            else
                State <= elseState;
            end if;
        end procedure;

    begin
        if rising_edge(KEY(3)) then
            if SW(0) = '0' then
                State <= A;    -- Reset
            else
                case State is -- Next state logic

                    when A =>
                        ChangeState(wantedValue => '0',
                                    toState     => B,
                                    elseState   => F);

                    when B =>
                        ChangeState(wantedValue => '0',
                                    toState     => C,
                                    elseState   => F);

                    when C =>
                        ChangeState(wantedValue => '0',
                                    toState     => D,
                                    elseState   => F);

                    when D =>
                        ChangeState(wantedValue => '0',
                                    toState     => E,
                                    elseState   => F);

                    when E =>
                        ChangeState(wantedValue => '0',
                                    toState     => E,
                                    elseState   => F);

                    when F =>
                        ChangeState(wantedValue => '1',
                                    toState     => G,
                                    elseState   => B);

                    when G =>
                        ChangeState(wantedValue => '1',
                                    toState     => H,
                                    elseState   => B);

                    when H =>
                        ChangeState(wantedValue => '1',
                                    toState     => I,
                                    elseState   => B);

                    when I =>
                        ChangeState(wantedValue => '1',
                                    toState     => I,
                                    elseState   => B);

                end case;
            end if;
        end if;
    end process;

    process(State)
    begin
        case State is -- Output logic
            when E      => z <= '1';
            when I      => z <= '1';
            when others => z <= '0';
        end case;
    end process;

    process(State) is
    begin
        case State is -- Encoding output hardcoded sequential
            when A => ShowEncoding <= "0000";
            when B => ShowEncoding <= "0001";
            when C => ShowEncoding <= "0010";
            when D => ShowEncoding <= "0011";
            when E => ShowEncoding <= "0100";
            when F => ShowEncoding <= "0101";
            when G => ShowEncoding <= "0110";
            when H => ShowEncoding <= "0111";
            when I => ShowEncoding <= "1000";
        end case;
    end process;
end architecture;
```

2.2. RTL

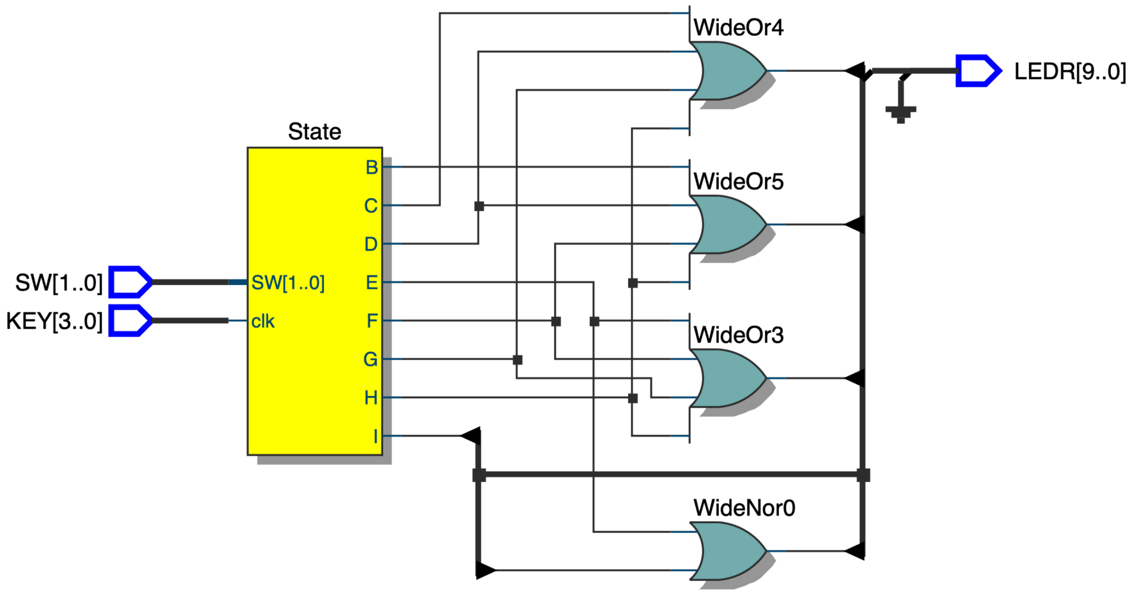

Figure 2.0: RTL of the TLE

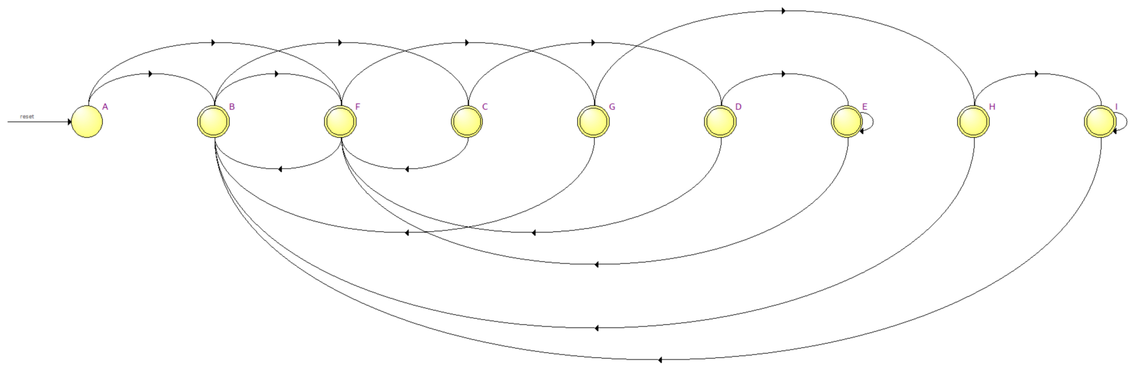

Figure 2.1: State diagram

Below the State diagram the used encoding shows up in a table. Instead of supplying a screenshot with bad quality, it was re-typed in excel with the same information and then imported here. The full screenshots from the FSM views is found on the GitHub

Table 2.0: Bit encoding output with sequential type

State	Encoding			
Name	3	2	1	0
A	0	0	0	0
B	0	0	0	1
C	0	0	1	0
D	0	0	1	1
E	0	1	0	0
F	0	1	0	1
G	0	1	1	0
H	0	1	1	1
I	1	0	0	0

Table 2.1: Bit encoding output with one-hot type

State	Encoding								
Name	8	7	6	5	4	3	2	1	0
A	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	1	1
C	0	0	0	0	0	0	1	0	1
D	0	0	0	0	0	1	0	0	1
E	0	0	0	0	1	0	0	0	1
F	0	0	0	1	0	0	0	0	1
G	0	0	1	0	0	0	0	0	1
H	0	1	0	0	0	0	0	0	1
I	1	0	0	0	0	0	0	0	1

2.3. Results

Showing the function of the circuit is hard with photos, therefore a video is uploaded to YouTube. Here is the video showing the behavioural state with the hardcoded output equal to the `type is "sequential"` encoding

In case linking is not working, the full URL is below:

https://youtu.be/ZBt7R3HuPg0