Protokol Skříňka

Zadání:

Navrhněte Mealyho automat, který řeší otevírání skříňky na vlakovém nádraží. Cena je 5Kč, vhazované mince jsou 1,2 a 5-koruna. Automat vrací 1Kč pouze v případě, že zákazník vhodil 4Kč a následně 2-korunovou minci. Výstupy, jestli se skříňka otevře, nebo vrátí i peníze, se bodou zobrazovat na LED diodách.

Teoretický rozbor Spartan a VHDL:

Spartan je programovatelné hradlové pole, což je typ logického integrovaného obvodu, který je vyroben tak, aby mohl být naprogramován kdykoliv. Obsahuje pole programovatelných logických obvodů (PLD), logických bloků, umožňuje je navzájem propojit a tím vytvořit takřka libovolné číslicové zařízení. Mikroprocesor je víceúčelové programovatelné zařízení, které na vstupu akceptuje digitální data, zpracuje je pomocí instrukcí uložených v paměti a jako výstup zobrazí výsledek. Mikroprocesor představuje příklad sekvenčního logického obvodu, který pro uložení dat používá dvojkovou soustavu.

VHDL je programovací jazyk, který slouží pro popis hardwaru. Používá se pro návrh a simulaci digitálních integrovaných obvodů, například programovatelných hradlových polí nebo různých zákaznických obvodů. Umožňuje návrh jak logických tak i sekvenčních struktur a jeho hlavní výhodou je jeho univerzálnost.

Definice stavů a jejich kódování:

Vstupní proměnné:

		btn0	btn1	btn2
X0	(0 Kč)	0	0	0
X1	(1 Kč)	1	0	0
X2	(2 Kč)	0	1	0
X3	(5 Kč)	0	0	1

Vnitřní proměnné:

	q2	q1	q0
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	1	0	0

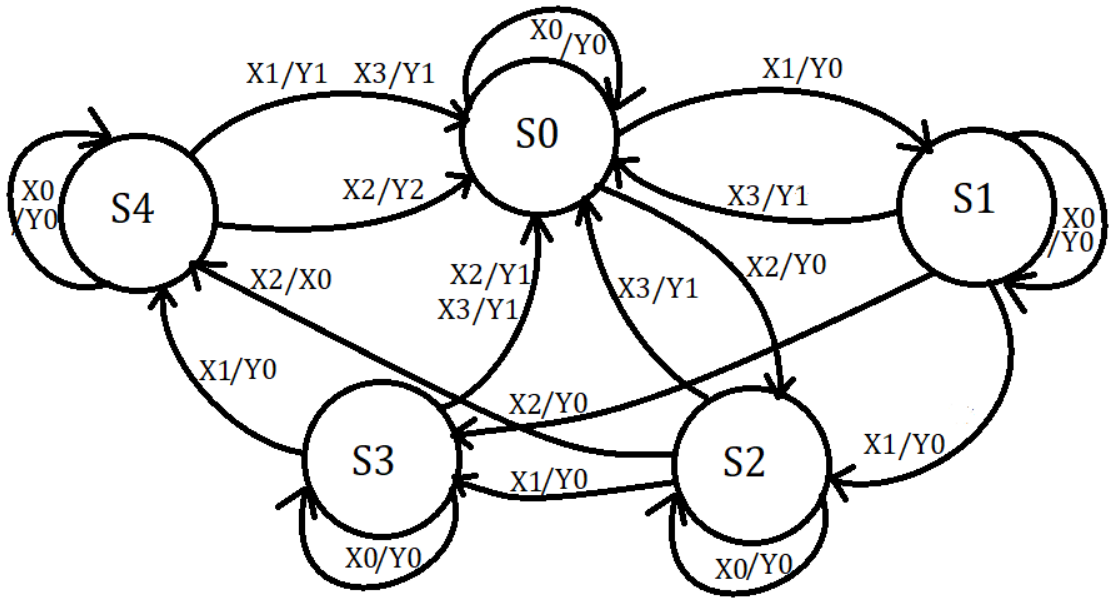
Výstupní proměnné:

	S0	S1
Y0	0	0
Y1	1	0
Y2	1	1

Popis automatu Mealy:

Mealyho automat se označuje jako konečný automat s výstupem. Výstup je generován na základě příchozího vstupu i momentálního stavu. Jeho stavový diagram automatu má ke každému přechodu přiřazenu nejen vstupní hodnotu, kterou je přechod aktivován, ale i výstupní hodnotu, která je při aktivaci přechodu vygenerována.

Orientovaný graf:



Tabulky přechodů mezi vnitřními stavy v závislosti na vstupních stavech, tabulky výstupů:

	X0/Y	X1/Y	X2/Y	X3/Y
S0	S0/Y0	S1/Y0	S2/Y0	S0/Y1
S1	S1/Y0	S2/Y0	S3/Y0	S0/Y1
S2	S2/Y0	S3/Y0	S4/Y0	S0/Y1
S3	S3/Y0	S4/Y0	S0/Y1	S0/Y1
S4	S4/Y0	S0/Y1	S0/Y2	S0/Y1

VHDL moduly – programy:

Dělička:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity delicka is
    Port ( CLK_in : in  STD_LOGIC;
           CLK_out : out  STD_LOGIC);
end delicka;

architecture Behavioral of delicka is

begin

    process (CLK_in)
        variable i : integer range 0 to 15000000 ;
begin
    if rising_edge(CLK_in) then
        if i=0 then CLK_out <= '1' ;
                    i := 9843000 ;
        else
                CLK_out <= '0' ;
                i := i - 1 ;
         end if ;
     end if ;
    end process;

end Behavioral;
```

Dekodér:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity dekoder is
    Port ( HEX : in  STD_LOGIC_VECTOR (2 downto 0);
           LED : out  STD_LOGIC_VECTOR (6 downto 0)
              );
end dekoder;


architecture Behavioral of dekoder is

begin


    with HEX SELect
   LED<= "1111001" when "0001",   --1
         "0100100" when "0010",   --2
         "0110000" when "0011",   --3
         "0011001" when "0100",   --4
         "1000000" when others;   --0

end Behavioral;
```

Skřínka:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity automat1 is
    Port (
           clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           mince : in  STD_LOGIC_VECTOR (2 downto 0);
           vyst : out STD_LOGIC_VECTOR (1 downto 0);
           stavy : out STD_LOGIC_VECTOR (2 downto 0));
end automat1;

architecture Behavioral of automat1 is
  signal state, next_state : STD_LOGIC_VECTOR (2 downto 0);

   constant s0 : STD_LOGIC_VECTOR (2 downto 0) := "000";
   constant s1 : STD_LOGIC_VECTOR (2 downto 0) := "001";
   constant s2 : STD_LOGIC_VECTOR (2 downto 0) := "010";
   constant s3 : STD_LOGIC_VECTOR (2 downto 0) := "011";
   constant s4 : STD_LOGIC_VECTOR (2 downto 0) := "100";
-------------------------------------------------------------
begin

   SYNC_PROC: process (clk)
   begin
      if rising_edge (clk)
         then if (rst='0')
              then state <= next_state;
             else state <= s0;
         end if;
      end if;
   end process SYNC_PROC;


-------------------------------------------------------------
OUTPUT_DECODE: process (mince)
   begin

      case (state) is
         when s0 =>
            if    (mince = "000") then vyst <= "00";
            elsif (mince = "100") then vyst <= "00";
            elsif (mince = "010") then vyst <= "00";
            elsif (mince = "001") then vyst <= "10";
            else vyst <= "00";
            end if;
         when s1 =>
            if    (mince = "000") then vyst <= "00";
            elsif (mince = "100") then vyst <= "00";
            elsif (mince = "010") then vyst <= "00";
            elsif (mince = "001") then vyst <= "10";
            else vyst <= "00";
            end if;
         when s1 =>
            if    (mince = "000") then vyst <= "00";
            elsif (mince = "100") then vyst <= "00";
            elsif (mince = "010") then vyst <= "00";
            elsif (mince = "001") then vyst <= "10";
            else vyst <= "00";
            end if;
         when s2 =>
            if    (mince = "000") then vyst <= "00";
            elsif (mince = "100") then vyst <= "00";
            elsif (mince = "010") then vyst <= "00";
            elsif (mince = "001") then vyst <= "10";
            else vyst <= "00";
            end if;
         when s3 =>
            if    (mince = "000") then vyst <= "00";
            elsif (mince = "100") then vyst <= "00";
            elsif (mince = "010") then vyst <= "10";
            elsif (mince = "001") then vyst <= "10";
            else vyst <= "00";
            end if;
         when s4 =>
            if    (mince = "000") then vyst <= "00";
            elsif (mince = "100") then vyst <= "10";
            elsif (mince = "010") then vyst <= "11";
            elsif (mince = "001") then vyst <= "10";
            else vyst <= "00";
            end if;

         when others => NULL;
      end case;

      end process OUTPUT_DECODE;


      NEXT_STATE_DECODE: process (state, mince)
      begin

      case (state) is
      when s0 =>
            if (mince = "000") then next_state <= s0; stavy <="000";
```

```
89              elsif (mince = "100") then next_state <= s1; stavy <="001";
90              elsif (mince = "010") then next_state <= s2; stavy <="010";
91              elsif (mince = "001") then next_state <= s0; stavy <="000";
92                      else next_state <= s0;
93                      end if;
94
95        when s1 =>
96              if (mince = "000") then next_state <= s1; stavy <="001";
97              elsif (mince = "100") then next_state <= s2; stavy <="010";
98              elsif (mince = "010") then next_state <= s3; stavy <="011";
99              elsif (mince = "001") then next_state <= s0; stavy <="000";
100                     else next_state <= s1;
101                     end if;
102
103       when s2 =>
104             if (mince = "000") then next_state <= s2; stavy <="010";
105             elsif (mince = "100") then next_state <= s3; stavy <="011";
106             elsif (mince = "010") then next_state <= s4; stavy <="100";
107             elsif (mince = "001") then next_state <= s0; stavy <="000";
108                     else next_state <= s2;
109                     end if;
110
111       when s3 =>
112             if (mince = "000") then next_state <= s3; stavy <="011";
113             elsif (mince = "100") then next_state <= s4; stavy <="100";
114             elsif (mince = "010") then next_state <= s0; stavy <="000";
115             elsif (mince = "001") then next_state <= s0; stavy <="000";
116                     else next_state <= s3;
117                     end if;
118
119       when s4 =>
120             if (mince = "000") then next_state <= s4; stavy <="100";
121             elsif (mince = "100") then next_state <= s0; stavy <="000";
122             elsif (mince = "010") then next_state <= s0; stavy <="000";
123             elsif (mince = "001") then next_state <= s0; stavy <="000";
124                     else next_state <= s4;
125                     end if;
126
127   when others => NULL;
128   end case;
129   stavy <= state;
130
131      end process NEXT_STATE_DECODE;
132
133
134
135    end Behavioral;
136
```

Simulace:

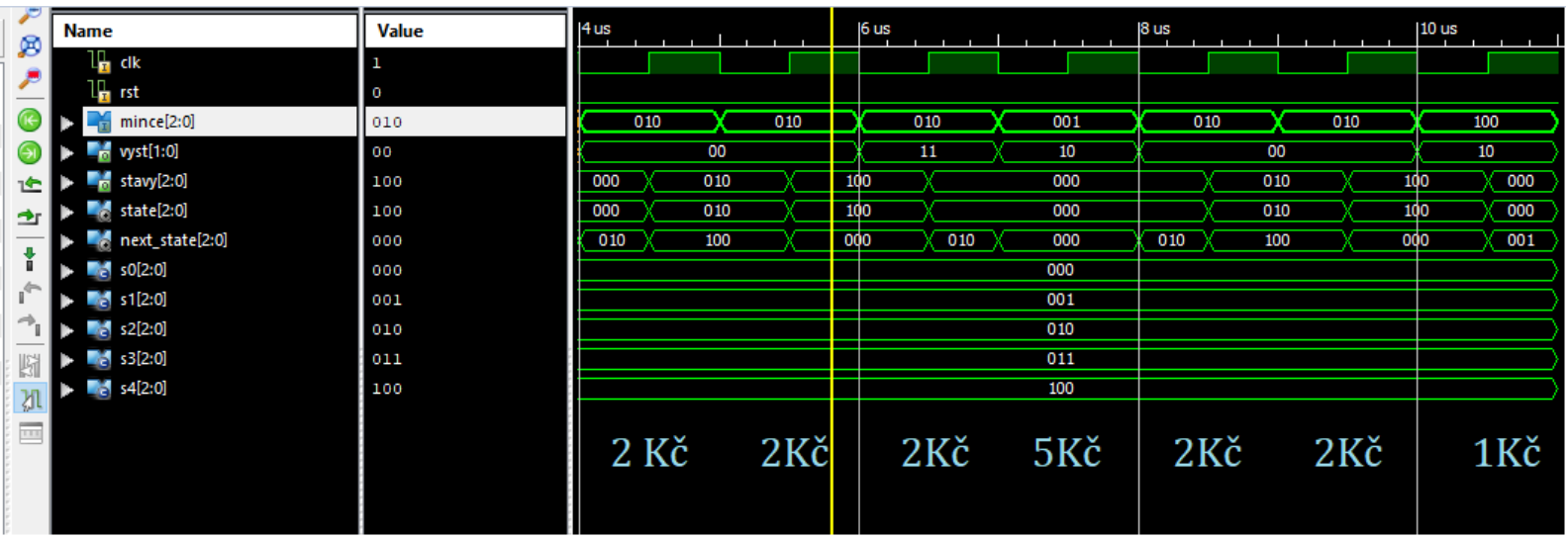
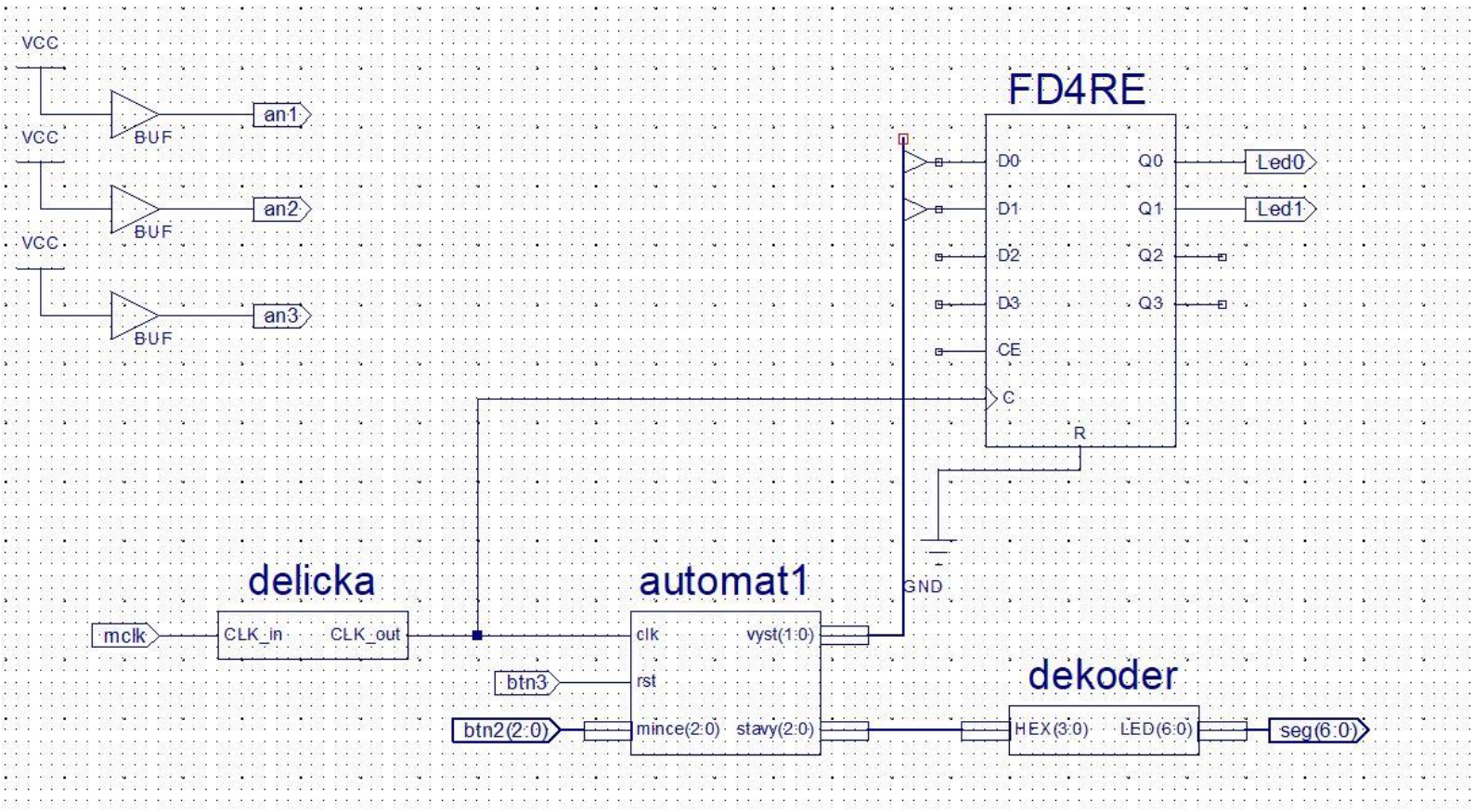


Schéma:


VCC
VCC
VCC
BUF
BUF
BUF
an1
an2
an3
FD4RE
D0
D1
D2
D3
CE
C
R
Q0
Q1
Q2
Q3
Led0
Led1
GND
delicka
mclk
CLK_in
CLK_out
automat1
clk
rst
vyst(1:0)
btn3
btn2(2:0)
mince(2:0)
stavy(2:0)
dekoder
HEX(3:0)
LED(6:0)
seg(6:0)

Piny:

```
# clock pins for Basys2 Board
NET "mclk" LOC = "B8"; # Bank = 0, Signal name = MCLK

# Pin assignment for DispCtl
# Connected to Basys2 onBoard 7seg display
NET "seg<0>" LOC = "L14"; # Bank = 1, Signal name = CA
NET "seg<1>" LOC = "H12"; # Bank = 1, Signal name = CB
NET "seg<2>" LOC = "N14"; # Bank = 1, Signal name = CC
NET "seg<3>" LOC = "N11"; # Bank = 2, Signal name = CD
NET "seg<4>" LOC = "P12"; # Bank = 2, Signal name = CE
NET "seg<5>" LOC = "L13"; # Bank = 1, Signal name = CF
NET "seg<6>" LOC = "M12"; # Bank = 1, Signal name = CG
#NET "dp" LOC = "N13"; # Bank = 1, Signal name = DP

NET "an3" LOC = "K14"; # Bank = 1, Signal name = AN3
NET "an2" LOC = "M13"; # Bank = 1, Signal name = AN2
NET "an1" LOC = "J12"; # Bank = 1, Signal name = AN1
#NET "an0" LOC = "F12"; # Bank = 1, Signal name = AN0

# Pin assignment for LEDs
#NET "Led<7>" LOC = "G1" ; # Bank = 3, Signal name = LD7
#NET "Led<6>" LOC = "P4" ; # Bank = 2, Signal name = LD6
#NET "Led<5>" LOC = "N4" ;  # Bank = 2, Signal name = LD5
#NET "Led<4>" LOC = "N5" ;  # Bank = 2, Signal name = LD4
#NET "Led<3>" LOC = "P6" ; # Bank = 2, Signal name = LD3
#NET "Led<2>" LOC = "P7" ; # Bank = 3, Signal name = LD2
NET "Led<1>" LOC = "M11" ; # Bank = 2, Signal name = LD1
NET "Led0" LOC = "M5" ;  # Bank = 2, Signal name = LD0

# Pin assignment for SWs
#NET "sw7" LOC = "N3";  # Bank = 2, Signal name = SW7
#NET "sw6" LOC = "E2";  # Bank = 3, Signal name = SW6
#NET "sw5" LOC = "F3";  # Bank = 3, Signal name = SW5
#NET "sw4" LOC = "G3";  # Bank = 3, Signal name = SW4
#NET "sw3" LOC = "B4";  # Bank = 3, Signal name = SW3
#NET "sw2" LOC = "K3";  # Bank = 3, Signal name = SW2
#NET "sw1" LOC = "L3";  # Bank = 3, Signal name = SW1
#NET "sw0" LOC = "P11";  # Bank = 2, Signal name = SW0

NET "btn3" LOC = "A7";  # Bank = 1, Signal name = BTN3
NET "btn<2>" LOC = "M4";  # Bank = 0, Signal name = BTN2
NET "btn<1>" LOC = "C11"; # Bank = 2, Signal name = BTN1
NET "btn<0>" LOC = "G12"; # Bank = 0, Signal name = BTN0

## Pin assignment for PS2
#NET "ps2c"    LOC = "B1"   | DRIVE = 2  | PULLUP ; # Bank = 3, Signal name = PS2C
#NET "ps2d"    LOC = "C3"   | DRIVE = 2  | PULLUP ; # Bank = 3, Signal name = PS2D
```

Zhodnocení:

Úkolem bylo vytvořit funkční skříňku, která se bude otevírat a otevírat i vracet korunu podle toho, kolik jsme vložili korun do skříňky. Budou se nám rozsvicovat ledky podle našich výstupů. S tímto úkolem jsem měl problém, kvůli kterému se mi skříňka otevírala předem, naštěstí jsem měl pouze špatně nastavený clock. Program jsme celý dělali doma a byl celkem obtížný.