Skříňka-protokol

Zadáni:

Navrhněte Mealyho automat, který řeší otevírání skříňky na vlakovém nádraží. Cena otevření je 5 kč. Vhazované mince jsou 1,2 a 5 kč. Automat vrátí korunu zpět pouze pokud zákazník vhodil dříve již 4 koruny, a následně vhodil 2 korunu.

Teoretický rozbor :

Jazyk VHDL - Jazyk VHDL je speciální programovací jazyk pro popis hardwaru. Tento jazyk byl standardizován v roce 1987 a umožňuje popis a simulaci rozsáhlých číslicových systémů. Základní vlastnosti jazyka VHDL:

- Má bohaté vyjadřovací schopnosti.
- Je zde značná nezávislost číslicového systému popsaného tímto jazykem na cílovém prvku, v kterém bude tento systém použit.
- Je možné provádět simulaci chování číslicového systému.

Základním stavebním prvkem jazyka VHDL je tzv. entita, která popisuje vstupní a výstupní signály číslicového systému. Vlastní chování a funkci entity definuje tzv. architektura, která popisuje, jakým způsobem se zpracovávají vstupní signály a generují výstupní signály entity. Architektura je součástí entity a každá entita musí mít alespoň jednu architekturu. Více informací o jazyku VHDL lze získat v literatuře a na internetu.

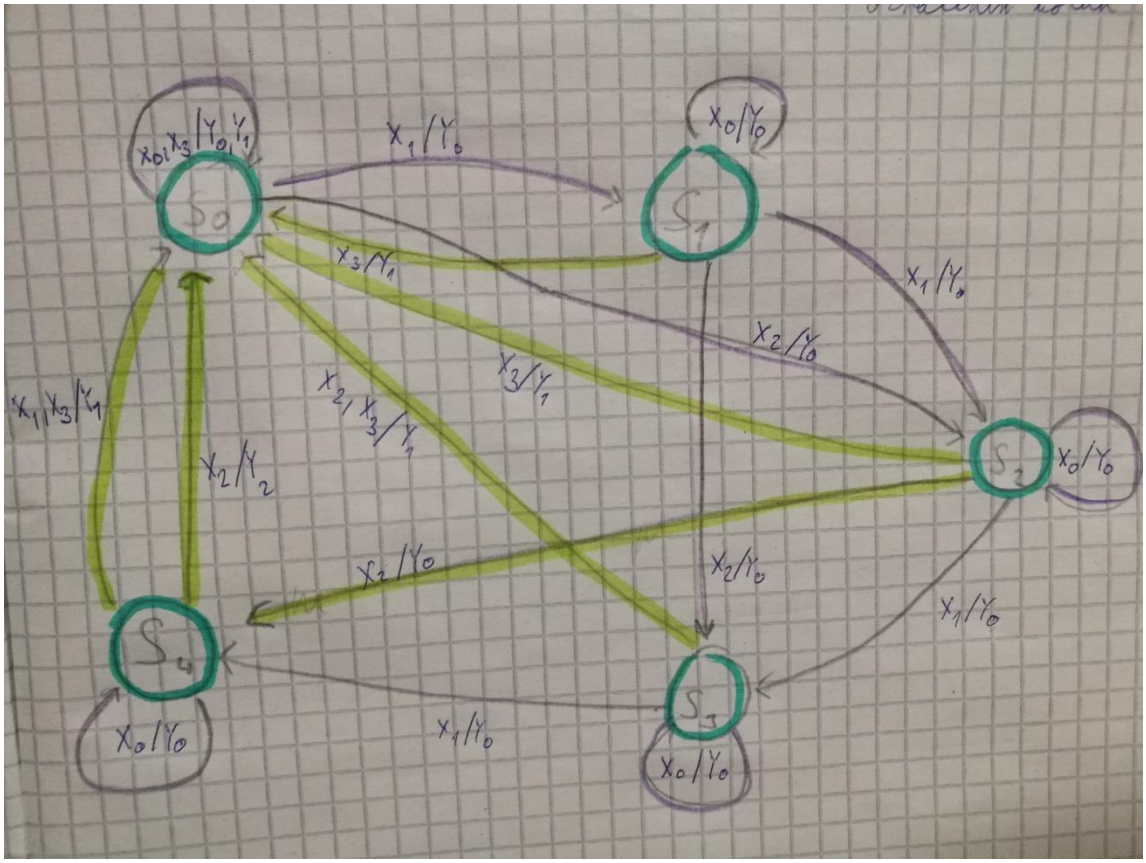
Definice stavů a jejich kódování:

	vstupní				výstupní			vnitřní		
	mince(2:0)				vyst(1:0)			Q1	Q2	Q3
	1kč	2kč	5kč		O(otev)	V(vrac)	S0	0	0	0
X0	0	0	0	Y1	0	0	S1	0	1	0
X1	1	0	0	Y2	1	0	S2	1	0	0
X2	0	1	0	Y3	1	1	S3	1	1	0
X3	0	0	1				S4	0	0	1

Popis Mealyho automatu:

Výstup je generován na základě příchozího vstupu i momentálního stavu, ve kterém se automat nachází. To znamená, že 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. Tímto Mealyho automat připomíná synchronní komunikaci: Nejen že reaguje na hranu vstupního signálu, ale jakmile ho zpracuje a dosáhne dalšího stavu, jednou vygeneruje výstupní hodnotu, puls výstupního signálu, a pak už žádný výstup neposkytuje; zase až do další vstupní hodnoty předložené ke zpracování. Totiž nejen, že jsou *stavy* Mealyho stroje podmnožinou kartézského součinu množiny (předešlých) stavů a *vstupní* abecedy, ale i jeho *výstupy* jsou podmnožinou kartézského součinu stavů a *výstupní* abecedy.

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

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
-- segment encoinputg
--      0
--     ---
--  5 |   | 1
--     ---   <- 6
--  4 |   | 2
--     ---
--      3

    with HEX SELect
   LED<= "1111001" when "001",   --1
         "0100100" when "010",   --2
         "0110000" when "011",   --3
                                 "0011001" when "100",   --4
         "1000000" when others; --0

end Behavioral;
```

Dělička:

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

Hlavní program vytah_main:

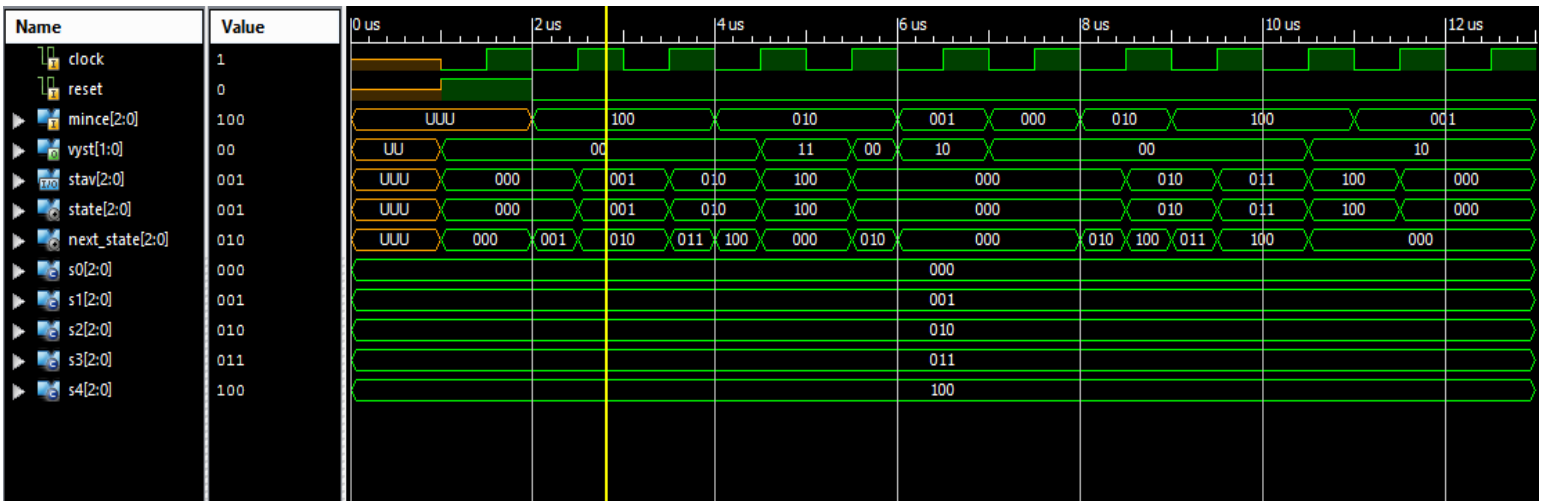
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity aut is
    Port ( clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           mince : in  STD_LOGIC_VECTOR (2 downto 0);
           vyst : out  STD_LOGIC_VECTOR (1 downto 0);
           stav : inout  STD_LOGIC_VECTOR (2 downto 0));
end aut;

architecture Behavioral of aut 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 (CLOCK, RESET)
```

```
begin
	if(reset='1') then
			state <= s0;

	elsif rising_edge(clock) then
		state<=next_state;
	end if;
end process;

OUTPUT_DECODE: process (state, 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 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";
```

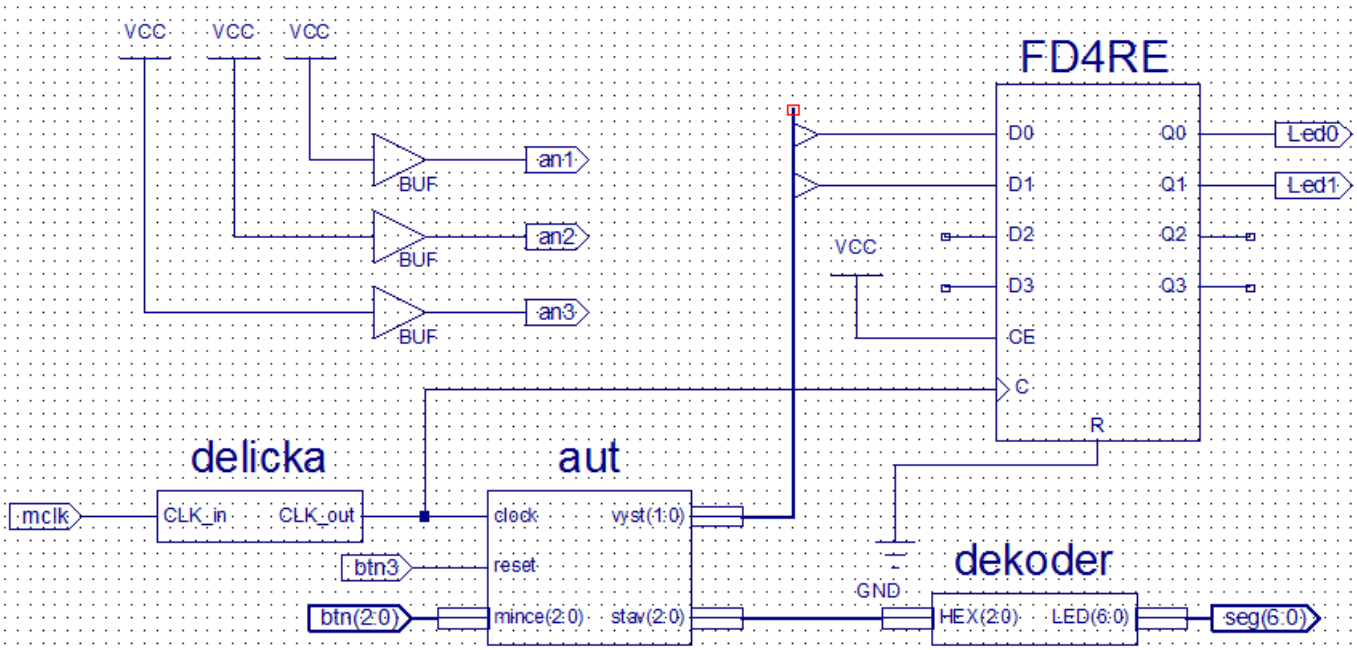
```
                                        else vyst<= "00";
                                        end if;

                        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;
                        stav <= state;
                end process OUTPUT_DECODE;

NEXT_STATE_DECODE: process (state, mince)
begin

case (state) is
        when s0 =>
                if (mince = "000") then next_state <= s0; stav <="000";
                elsif (mince = "100") then next_state <= s1; stav <="001";
                elsif (mince = "010") then next_state <= s2; stav <="010";
                elsif (mince = "001") then next_state <= s0; stav <="000";
        else next_state <= s0;
        end if;

        when s1 =>
    if (mince = "000") then next_state <= s1; stav <="001";
                elsif (mince = "100") then next_state <= s2; stav <="010";
                elsif (mince = "010") then next_state <= s3; stav <="011";
                elsif (mince = "001") then next_state <= s0; stav <="000";
        else next_state <= s1;
        end if;

        when s2 =>
    if (mince = "000") then next_state <= s2; stav <="010";
                elsif (mince = "100") then next_state <= s3; stav <="011";
```

```
                elsif (mince = "010") then next_state <= s4; stav <="100";
                elsif (mince = "001") then next_state <= s0; stav <="000";
        else next_state <= s2;
        end if;

        when s3 =>
    if (mince = "000") then next_state <= s3; stav <="011";
                elsif (mince = "100") then next_state <= s4; stav <="100";
                elsif (mince = "010") then next_state <= s0; stav <="000";
                elsif (mince = "001") then next_state <= s0; stav <="000";
        else next_state <= s3;
        end if;

        when s4 =>
                if (mince = "000") then next_state <= s4; stav <="100";
                elsif (mince = "100") then next_state <= s0; stav <="000";
                elsif (mince = "010") then next_state <= s0; stav <="000";
                elsif (mince = "001") then next_state <= s0; stav <="000";
        else next_state <= s4;
        end if;

when others => NULL;
end case;
stav <= state;

end process NEXT_STATE_DECODE;

end Behavioral;
```

Simulace:



Celkové schéma:



Výpis pinů:

```
# 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 "Led<0>" 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 "btn2" LOC = "M4";  # Bank = 0, Signal name = BTN2
NET "btn1" LOC = "C11"; # Bank = 2, Signal name = BTN1
NET "btn0" 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:

Za úkol jsme dostali vytvořit v programu Xilinx program skříňky, který se zavírá, otevírá a vrací peníze. Jako první jsem pomocí nápovědy vytvořil děličku, následně dekodér, a podle nápovědy Mealyho automatu jsem vytvořil i hlavní program skříňky. U každého modulu jsem zkontroloval syntaktické chyby, a po opravě jsem vytvořil schématické značky. Nasimuloval jsem skříňku i v simulačním prostředí, a fungoval bez problému. Tento program jsem nedělal ve škole, tudíž jsem ho nemohl otestovat na přípravku. Program sám o sobě je jeden z náročnějších, a vyžaduje znalosti programovacího jazyka a zkušenosti.

Šimon Nieslanik 4.C