

VAJA 5: AVTOMATI KONČNIH STANJ, UART

Primer realizacije MOORE-ovega avtomata v VHDL-u

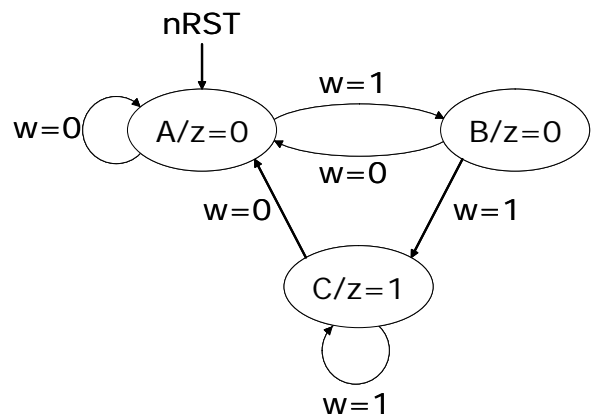
```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY detect IS
    PORT(      clk, nRST, w : IN STD_LOGIC ;
              z : OUT STD_LOGIC) ;
END detect ;

ARCHITECTURE arch OF detect IS
    TYPE fsm_states IS (A, B, C) ;
    SIGNAL state: fsm_states ;
BEGIN
    PROCESS ( nRST, clk )
    BEGIN
        IF nRST = '0' THEN
            state <= A;
        ELSIF rising_edge(clk) THEN
            CASE state IS
                WHEN A =>
                    IF w='0' THEN
                        state <= A;
                    ELSE
                        state <= B;
                    END IF;
                WHEN B =>
                    IF w='0' THEN
                        state <= A;
                    ELSE
                        state <= C;
                    END IF;
                WHEN C =>
                    IF w='0' THEN
                        state <= A;
                    ELSE
                        state <= C;
                    END IF;
            END CASE;
        END IF;
    END PROCESS;
    z <= '1' WHEN state = C ELSE '0';  --More-ov tip avtomata
END arch;

```



Zgornja VHDL koda je primer Moore-ovega avtomata, ki postavi izhod ( $z \leq '1'$ ), ko se na zaporednem vhodu ( $w$ ) pojavi zaporedje "11". S stavkom (`TYPE fsm_states IS (A, B, C);`) definiramo naštevni tip stanj avtomata, ki lahko zavzame vrednosti A, B ali C. S stavkom (`SIGNAL state: fsm_states`) definiramo signal, ki določa trenutno stanje, v katerem se nahajamo.

V procesnem stavku opišemo prehajanja stanj: Če avtomat ponastavimo ( $nRST = '0'$ ) preide trenutno stanje v stanje A ( $state \leq A$ ). Sicer opišemo prehajanje med stanji s sekvenčnim izbirnim stavkom (`CASE state IS`) v katerem navedemo vse možne prehode v druga stanja, kot opisuje diagram prehajanja stanj na desni strani. Prehajanje med stanji se odvija na prednji rob prehoda signala ure (`rising_edge(clk)`). Vsi izrazi določanja stanj in prehajanja med stanji so enaki tako

pri Moore-ovi kot Mealy-evi izvedbi. Izvedbi Moore-ovega in Mealy-evega avtomata se razlikujeta ravno v kombinacijskem izbirnem stavku, ki določa izhod avtomata:

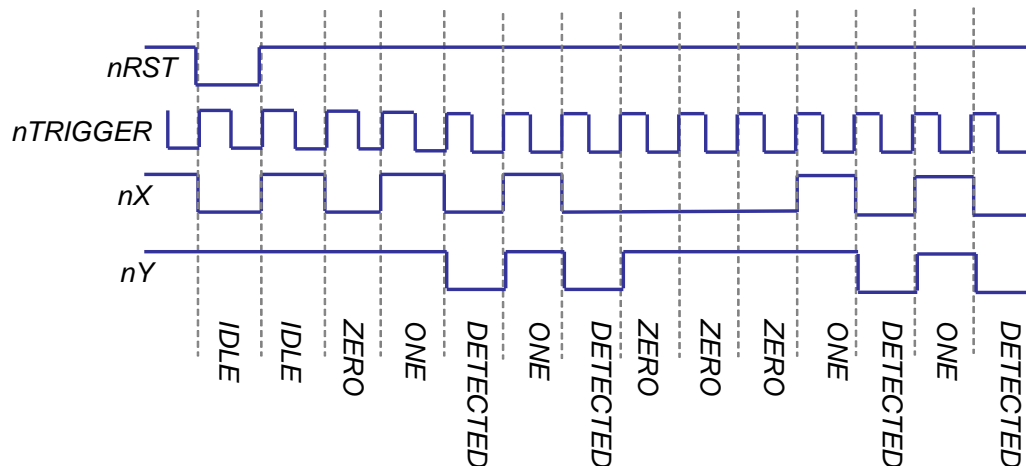
```
z <= '1' WHEN state = C ELSE '0';
```

V prikazanem primeru je izhod samo funkcija trenutnega stanja (*state*), pri Mealy-evi izvedbi bi bil izhod odvisen še od vhodov.

5.1 Narišite diagram prehajanja stanj Moore-ovega avtomata za detektor zaporedja, ki postavi izhod ( $nY \leq '0'$ ), ko se na zaporednem vhodu (*nX*) pojavi zaporedje "010". Prekrivanje zaporedja je dovoljeno. Stanja avtomata kodirajte kot:

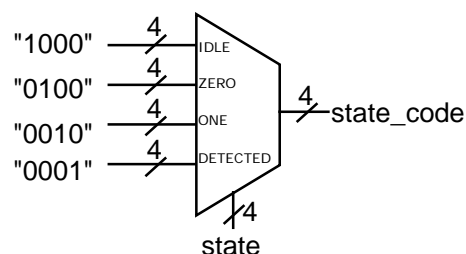
- Mirovno stanje avtomata: IDLE
- Na vhodu se je pojavila '0': ZERO
- Na vhodu se je pojavila '0', nato se je pojavila '1': ONE
- Pojavila se je '0', '1' in nato spet '0': DETECTED.

Avtomat ima vhod (*nRST*) za ponastavitev (ang. reset), ki v aktivnem stanju '0' postavlja stanje avtomata v mirovno stanje. Delovanje avtomata ilustrira spodnja časovna tabela, prikazana ob trenutnih sprememb na sprednji rob signala (*nTRIGGER*):



5.2 Ustvarite nov projekt (Vaja5\_2) in realizirajte vezje avtomata iz naloge 5.1 v datoteki (**fsm010.vhd**). Avtomat ima izhod (*nY*), vhod (*nRST*) za ponastavitev (ang. reset), zaporedni vhod (*nX*) in 4-bitni vektorski izhod (*state\_code*), na katerem izpisujemo kodo trenutnega stanja avtomata. Kode posameznih stanj so:

STANJE	state_code
IDLE	"1000"
ZERO	"0100"
ONE	"0010"
DETECTED	"0001"



Prehajanje med stanji avtomata se izvaja na prednji rob signala proženja (*nTRIGGER*) znotraj procesnega stavka. Za izvedbo večbitnega izbiralnika si oglejte 2. vajo (**bi n21 ed.vhd**)– podoben primer smo srečali pri prikazovanju segmentov na LED prikazovalniku.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fsm010 is
    Port (
        nY : out STD_LOGIC;
        state_code: out STD_LOGIC_VECTOR (3 downto 0);
        nX, nRST, nTRI GGER : in STD_LOGIC
    );
end fsm010;

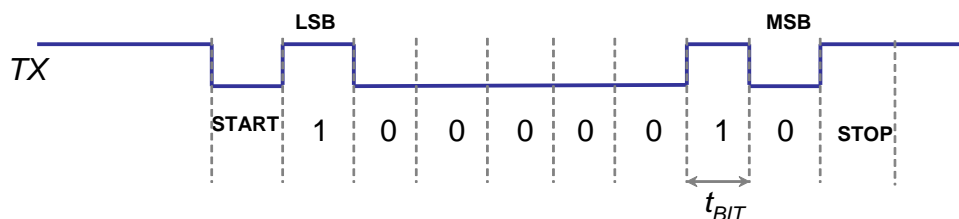
architecture arch of fsm010 is
    type fsm_states is (IDLE, ZERO, ONE, DETECTED);
    signal state: fsm_states;
begin
end arch;

```

Naredite jo lahko doma in na podlagi časovne tabele iz naloge 5.1 naredite simulacijo (fsm010\_tb.vhd), s katero preverite pravilnost delovanja avtomata. Vezje ima tudi UCF datoteko v kateri je vhod za ponastavitev (nRST) vezan na stikalo SW1, vhod za proženje (nTRI GGER) je vezan na tipko BTN0 in zaporedni vhod (nX) na stikalo SW0. Stanje avtomata (state\_code) je vezano na 4 LED diode, tako da LED4 predstavlja MSB spremenljivke. Izhod avtomata (nY) je vezan na LED3 diodo na Basys 2 ploščici.

5.3 Ustvarite nov projekt (Vaja5\_3) in realizirajte vezje univerzalnega asinhronnega serijskega oddajnika (ang. universal asynchronous transmitter) v novi datoteki (uart\_tx.vhd).

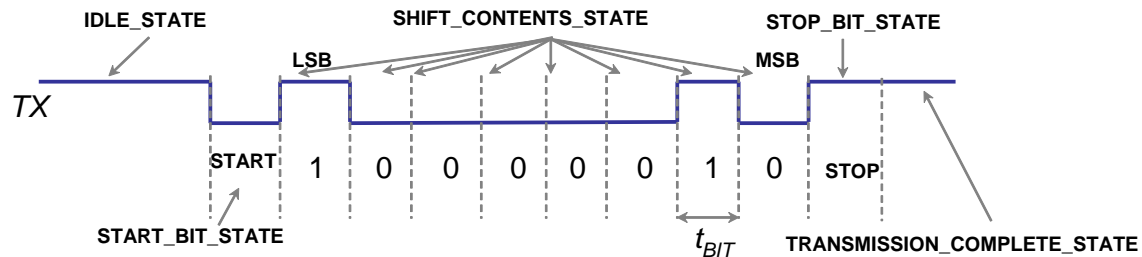
Asinhrona serijska komunikacija za svoje delovanje ne pošilja sprejemniku svojega signala ure, ampak informacijo pošilja glede na nastavitev hitrosti delovanja oddajne linije. Njeno hitrost merimo v BAUD oz. BPS (bit/sekundo). Tipična hitrost te komunikacije znaša 9600 BPS. Od tod sledi čas trajanja prenosa enega bita  $t_{BIT}=1/9600 \approx 104 \mu s$ . Vse asinhrone vrste komunikacij za pravilno delovanje potrebujejo nek okvir (ang. frame), ki omogoča sinhronizacijo ure na sprejemnikovi in oddajnikovi strani. Pri asinhroni serijski komunikaciji je okvir sestavljen iz enega start in enega (ali več) stop bitov. Na spodnji sliki je prikazan primer oddajanja ASCII znaka črke A, katerega koda je  $41_{16}$ .



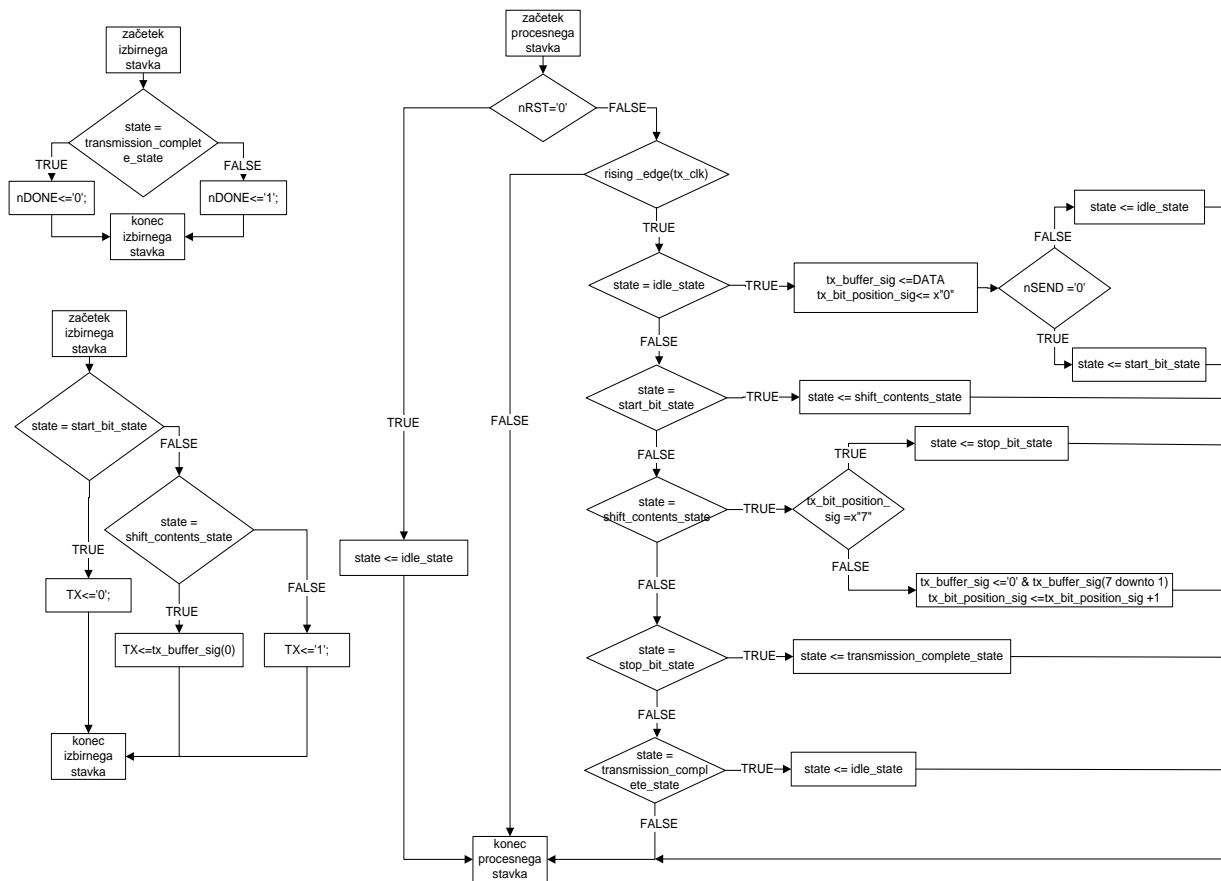
Oddajna linija (TX) se na začetku nahaja v mirovnem stanju, ki je po dogovoru vedno visoko stanje ( $TX \leq '1'$ ). Ob začetku prenosa se pošlje start bit, ki je po dogovoru vedno ( $TX \leq '0'$ ). Podatek, ki je v našem primeru črka  $A_{ASCII}=41_{16}=01000001_2$ , prenašamo tako, da najprej pošljemo LSB mesto. Pošiljanje podatka je ponavadi realizirano kot 8-bitni pomikalni register v desno. Prenos podatka končamo z enim ali več stop bit, ki so po dogovoru vedno logična '1'. Asinhrono serijsko komunikacijo lahko realiziramo kot avtomat končnih stanj, ki ima 11 stanj, s tem da bomo pomikanje 8 bitnega podatka obravnavali kot eno stanje v katerem vztrajamo osemkrat – dokler pomikamo podatek. Avtomat prehaja med stanji s hitrostjo ure oddajnika (TX\_CLK), katere perioda je  $104 \mu s$ . Stanja, med katerimi prehaja avtomat označimo kot:

IDLE\_STATE – mirovno stanje, START\_BIT\_STATE – stanje oddajanja START bita,

SHIFT\_CONTENTS\_STATE – stanje pomikanja podatka, STOP\_BIT\_STATE – stanje oddajanja STOP bita in TRANSMISSION\_COMPLETE\_STATE – stanje zaključenega prenosa.



Avtomat oddajnika ima še kontrolni vhod za pošiljanje (`nSEND`), ki drži avtomat v mirovnem stanju, dokler ne postane (`nSEND <= '0'`). Ko avtomat prejme ukaz za delovanje (`nSEND <= '0'`), preide v stanje oddajanja START bita. V stanju oddajanja bi postavili oddajno linijo (TX) na '0' znotraj avtomata, a bomo to raje storili pozneje s posebnim izbirnim stavkom zaradi enostavnejše realizacije. Ker avtomat teče s taktom (TX\_CLK), bo ob naslednjem prednjem robu (TX\_CLK) prešel v stanje pomikanja podatka, pri katerem povečuje števec pomikov (`tx_bit_position`) in izvaja pomikanje tako, da se LSB mesto pomnilnika za podatek (`tx_buffer_sig`) pomakne na oddajno linijo (`TX <= tx_buffer_sig(0)`), preostali biti se pomaknejo eno mesto desno. Tudi v tem stanju bomo postavljanje oddajne linije (TX) raje realizirali z izbirnim stavkom na koncu zaradi enostavnosti. Avtomat ostaja v stanju pomikanja podatka, dokler števec pomikov ni enak 7 (`tx_bit_position = x"7"`), nato preide v stanje oddajanja STOP bita. Ko je bil STOP bit oddan, mora avtomat nadrejenemu modulu javiti, da je končal prenos, kar stori tako, da preide v stanje zaključenega prenosa, kjer postavi izhodni signal (`nDONE <= '0'`) in se vrne nazaj v mirovno stanje. Prehajanje med stanji v VHDL najlažje realiziramo s sekvenčnim izbirnim stavkom (**CASE state IS**), kot je prikazano v uvodu te vaje. Izhod oddajne linije (TX) realiziramo izven procesnega stavka s posebnim izbirnim stavkom, ki ima gnezden (**WHEN ELSE WHEN ELSE**) pogoj: Oddajna linija bo (`TX <= '0'`), če je avtomat v stanju oddaje start bita, sicer bo postala LSB mesto pomikalnega registra (`TX <= tx_buffer_sig(0)`), če je v stanju pomikanja podatka, sicer bo oddajna linija (`TX <= '1'`). Signal (`nDONE`) postane '0', ko je avtomat v stanju zaključenega prenosa, sicer je '1'.



Vaja nima UCF datoteke. Naredite jo lahko doma in s priloženo datoteko testnih vrednosti (UART\_TX\_test.vhd) preverite pravilnost delovanja.