SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

PROJEKTIRANJE DIGITALNIH SUSTAVA

IZVJEŠTAJ PROJEKTA

IMPLEMENTACIJA IGRE BLACKJACK U VERILOGU

Petra Šteko, Julija Kvesić, Tomislav Kuliš

Split, srpanj 2022.

SADRŽAJ

1. UVOD

Zadatak ovog projekta je implementacija kartaške igre *blackjack* na Xilinx Spartan-3E pločici u Verilogu.

U nastavku izvještaja dan je pregled pravila igre te implementacija igre na pločici. Objašnjeni su pojedini dijelovi Verilog koda, dijagram stanja, korišteni moduli, rezultati simulacija, definirani ulazi i izlazi te prikaz rezultata igre na LCD zaslonu. Cjeloviti kod (kao i testni moduli i *.ucf* datoteka) su priloženi na kraju izvještaja.

U zaključku je dan pregled projekta kao i prednosti i mane implementacije te moguća poboljšanja.

2. PRAVILA IGRE

Blackjack je kartaška igra na sreću koja se igra protiv kuće. Cilj igre je uzeti karte što bliže, ali ne i više od vrijednost 21 i pobijediti djelitelja. Međutim, ako ruka igrača (engl. *hand*) prijeđe 21, smatrati će se da je izgubio bez obzira na ruku djelitelja (engl. *dealer*).

Vrijednosti karata su:

- Kralj, kraljica i dečko vrijede 10 bodova;
- Karte s brojevima zadržavaju nominalnu vrijednost;
- Asevi se mogu računati kao 1 bod ili 11 bodova, ovisno o tome kako više odgovara igraču (npr. ako igrač dobije dva asa koja bi se računala kao 11, premašio bi 21 i izgubio igru pa se zbog toga jedan as računa kao 1 te je njegov rezultat onda 12).

U početku igre igrač i djelitelj dobiju po dvije karte. Nakon toga igrač ima pravo na četiri poteza: *hit, stay*, *double down* i *split*.

- *Hit* – Nakon što igrač dobije dvije karte, može zatražiti novu. To može raditi sve dok nije zadovoljan rezultatom (dok ne dođe što bliže 21, a da ne premaši 21).
- *Stay* – Nakon što je igrač dobio minimalno dvije karte (početne dvije) ili zatražio još karata (*hit*) s ovim potezom signalizira da je zadovoljan rezultatom te slijedi nastavak igre (djelitelj dobiva svoje karte te se nakon toga određuje rezultat igre).
- *Double down* – Igrač može odabrati *double down* u slučaju da je zbroj početne dvije karte 9, 10 ili 11 i s tim potezom pristaje na to da će u nastavku dobiti samo još jednu kartu. Prilikom *double downa* ulog se udvostručuje.
- *Split* – Ako igrač dobije dvije iste karte može ih razdvojiti te nakon toga igrati na svaku kao posebnu igru. Nakon toga mora birati *hit* minimalno jednom ili onoliko puta koliko želi (dok ne premaši 21 ili bude zadovoljan rezultatom).

Nakon što igrač odluči da je zadovoljan svojim rezultatom ili je premašio 21, djelitelj dobiva svoje karte. Djeliteljeva je igra strogo propisana i u njoj ne postoji element vještine jer djelitelj ne može donositi nikakve odluke o njoj. Nakon što dobije početne dvije karte i ako je njihov zbroj manji od 17, djelitelj mora uzeti još karata sve dok njegov rezultat ne bude 17 ili veći. Bez obzira na kakav se potez igrač odluči, potezi djelitelja su uvijek isti.

Nakon toga slijedi usporedba rezultata. Mogući ishodi igre (s perspektive igrača) su:

- Pobjeda (engl. *win*) – ako igrač ima veći rezultat u odnosu na djelitelja (a da nije premašio 21);
- Poraz (engl. *lose*) – ako je igračev rezultat manji ili ako igračev rezultat premaši 21 u odnosu na djelitelja koji nije premašio 21;
- Izjednačeno (engl. *tie*) – ako igrač i djelitelj imaju isti rezultat ili su oba premašila 21

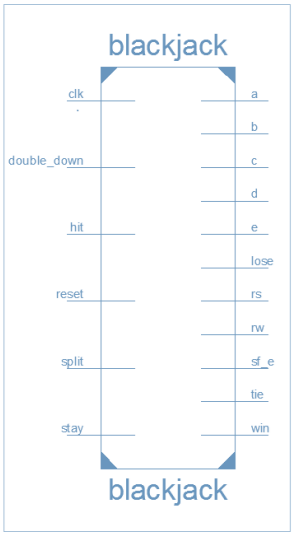
Prije igre igrač mora uložiti određeni iznos novca. U slučaju pobjede igrača, djelitelj igraču isplaćuje onoliko koliko je igrač uložio, tzv. isplata u omjeru 1:1. Npr. ako igrač koji ima 100 dolara i odluči uložiti 20 dolara i pobjedi u igri, dobit će 20 dolara od djelitelja (sad ukupno ima 120 dolara). Ukoliko izgubi, ostaje bez uloženog novca (ukupno ima 80 dolara). Iznimno u slučaju da igrač pobjedi s rezultatom 21 (engl. *blackjack*), ulog mu se isplaćuje u omjeru 3:2 (u prethodnom primjeru s 20 uloženih dolara igrač bi na kraju ovakve igre dobio 30 dolara, odnosno ukupno ima 130 dolara). U slučaju da igrač odabere *double down*, igračev ulog se dupla (u prethodnom primjeru bi dobio/izgubio 40 dolara ovisno o ishodu igre) te u slučaju da se igrač odluči za *split* isplata je u omjeru 1:1 na svaku ruku.

3. IMPLEMENTACIJA

U nastavku je objašnjeno koji se ulazi i izlazi koriste, svi potrebni moduli i komponente Spartan-3E pločice. Pojašnjeni su najbitniji dijelovi koda nekih modula. Cjeloviti kod se nalazi na kraju kao prilog.

3.1. Ulazi i izlazi

Na slici 3.1. prikazan je top modul sa svim njegovim ulazima i izlazima.



Slika 3.1. Top modul blackjack

Ulazi modula:

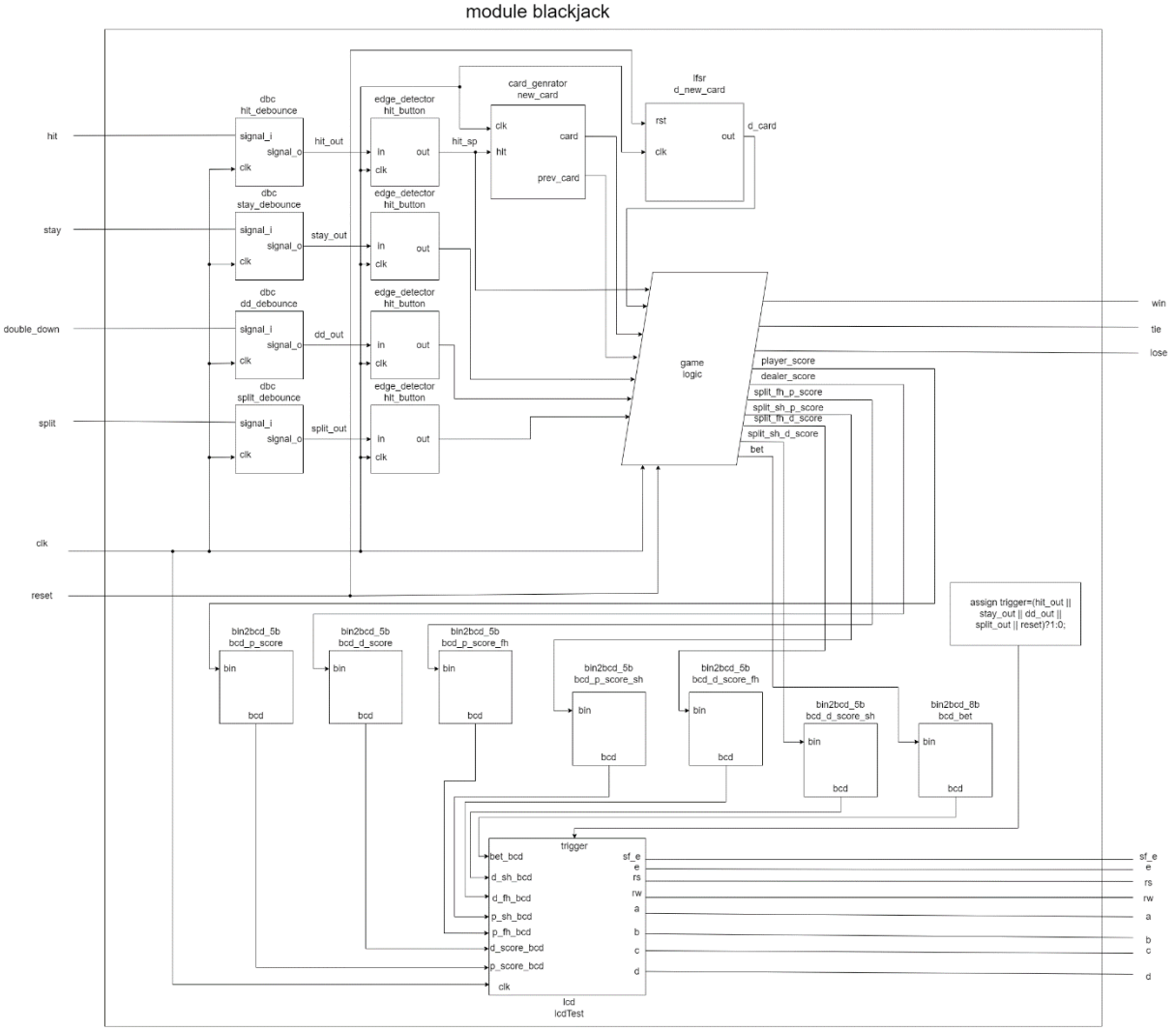
- *clk* – takt sklopa – korišten je interni takt Spartan-3E pločice (50 MHz oscilator)
- *hit*, *stay*, *double_down*, *split* – ulazi potrebni za igru (korištena su 4 tipkala s pločice)
- *reset* – resetira igru i vraća sklop u početno stajne (korištena je jedna sklopka s pločice)

Izlazi modula:

- *a*, *b*, *c*, *d* – podatkovni izlazi LCD ekrana
- *rs*, *rw*, *sf_e*, *e* – izlazi potrebni za upravljanje LCD ekranom (objašnjeni kasnije u poglavlju koje se odnosi na LCD prikaz)
- *win*, *lose*, *tie* – izlazi koji govore o ishodu igre

3.2. Korišteni moduli

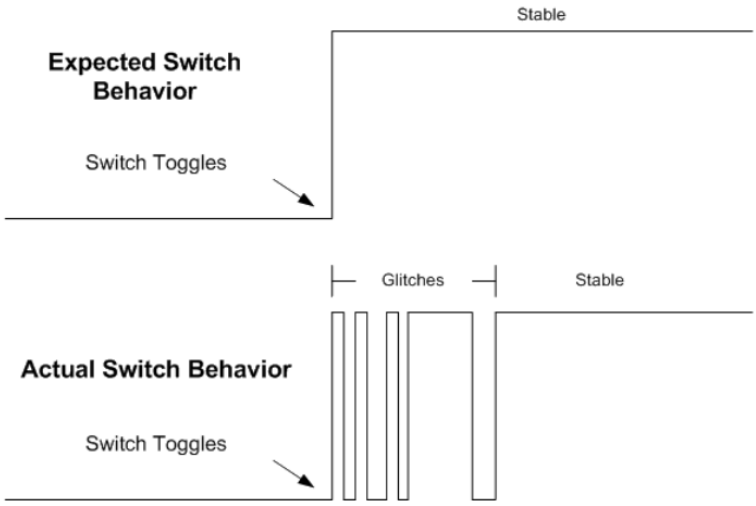
Na slici 3.2. prikazana je blok shema svih korištenih modula. Top modul blackjack sadrži module: *dbc*, *edge_detector*, *card_genrator*, *lfsr*, *bin2bcd_5b*, *bin2bcd_8b* i *lcd*.



Slika 3.2. Blok shema svih korištenih modula

3.2.1. Modul *dbc*

Modul *dbc* predstavlja *debounce* modul za tipkala s pločice. Instancirana su 4 *dbc* modula za svaki od ulaza potrebnih za igru (*hit*, *stay*, *double_down* i *split*). Prilikom korištenja tipkala ili prekidača s pločice dolazi do „poskakivanja“ signala uslijed brzog spajanja i odvajanja metalnih kontakata prije nego imaju vremena da se slegnu. Kao što je prikazano na slici 3.3., prilikom pritiska nekog od tipkala očekuje se čista tranzicija signala iz logičke nule u jedinicu (ili obrnuto). Međutim, stvarni signal ne izgleda tako zbog prethodno navedenoga razloga. Modul sadrži 1b ulaz *signal_i* i 1b izlaz *signal_o*.



Slika 3.3. „Poskakivanje“ signala tipkala ili prekidača

Modul *dbc* je implementiran kao automat s stanjima: s_*initial*, *s_one*, *s_zero*, *s_zero_to_one*, *s_one_to_zero*. Ovisno o ulaznom signalu *signal_i* prelazi se u stanje *s_one* (ukoliko je ulaz u 1) ili stanje *s_zero* (ukoliko je ulaz u 0).

Razmotrimo na primjeru kada se događa prijelaz iz 0 u 1 (odnosno u slučaju da igrač pritisne neki od tipkala (*hit*, *stay*, *double_down* ili *split*). Automat reagira na pozitivan brid taktnog signala. Dok je ulaz *signal_i* u 0, automat se nalazi u stanju *s_zero* te na izlazu *signal_o* daje 0 sve dok ne dođe do promjene na ulazu.

```
s_zero: begin
                signal_o <= 1'b0;

                if (signal_i == 1'b1)
                    state <= s_zero_to_one;
            end
```

Kada se dogodi promjena na ulazu, prelazi se u stanje *s_zero_to_one* u kojemu se isto na izlazu daje 0 i aktivira brojač *timer* pomoću varijable *timer_en*.

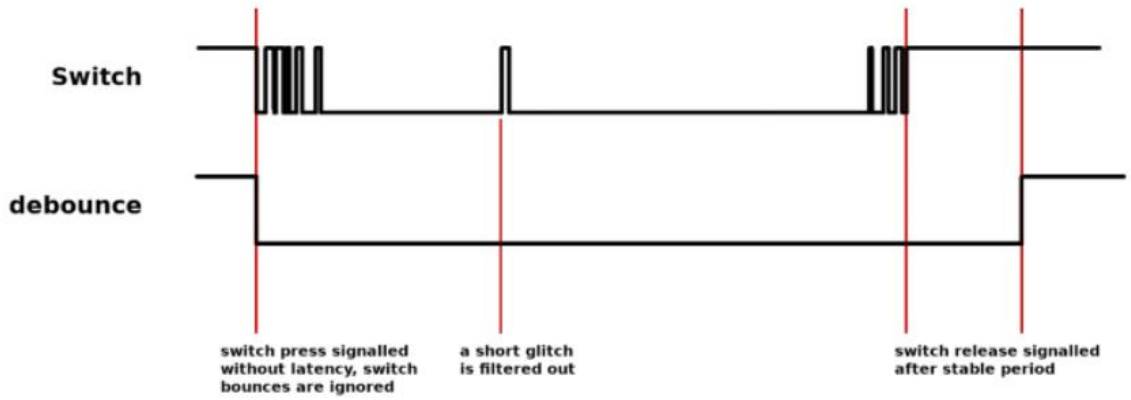
```
s_zero_to_one: begin
                signal_o <= 1'b0;
                timer_en <= 1'b1;

                if (timer_tick == 1'b1) begin
                    state <= s_one;
                    timer_en <= 1'b0;
                end
```

```
            if (signal_i == 1'b0) begin
                state <= s_zero;
                timer_en <= 1'b0;
            end
        end
```

Sve dok je *timer_en* postavljen, ulazi se u dio koda koji uvećava brojač *timer* koji kada dođe do svoje maksimalne vrijednosti (definirane varijablom *timerlim*) postavlja *timer_tick* u 1 koja definira prijelaz u iduće stanje. Dok timer ne izbroji do maksimalne vrijednosti, izlaz se drži u 0. Tako se osigura da će ulazni signal nakon vremena potrebnog *timeru* da dosegne maksimalnu vrijednost imati stabilnu vrijednost te nakon toga automat može prijeći u stanje *s_one* kako bi davao 1 na izlazu sve dok se detektira promjena iz 1 u 0 na ulazu nakon čega se odvija analogna stvar s *timerom* u stanju *s_one_to_zero*.

```
        if (timer_en == 1'b1) begin
            if (timer == (timerlim - 1)) begin
                timer_tick <= 1'b1;
                timer <= 17'b0;
            end
            else begin
                timer_tick <= 1'b0;
                timer <= timer + 17'b1;
            end
        end
        else begin
            timer <= 17'b0;
            timer_tick <= 1'b0;
        end
```

Na slici 3.3. prikazan je ulazni i izlazni signal modula *dbc*. U glavnom modulu *blackjack* signali bez poskakivanja nazvani su *hit_out*, *stay_out*, *dd_out* i *split_out*.

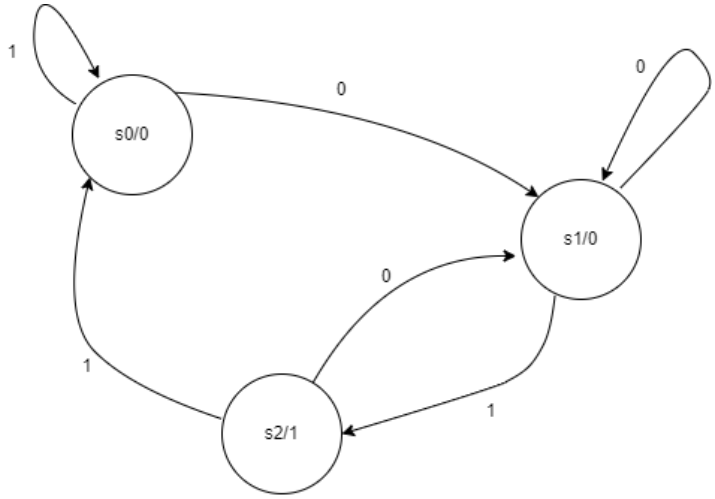


Slika 3.3. Ulazni i izlazni signal

```
dbc hit_debounce(.signal_i(hit),.clk(clk), .signal_o(hit_out));
dbc stay_debounce(.signal_i(stay),.clk(clk), .signal_o(stay_out));
dbc dd_debounce(.signal_i(double_down),.clk(clk), .signal_o(dd_out));
dbc split_debounce(.signal_i(split),.clk(clk), .signal_o(split_out));
```

3.2.2. Modul *edge_detector*

Modul *edge_detector* služi kako bi iz čistog signala (bez „poskakivanja“) na izlazu dao signal koji traje točno jedan period taktnog signala (20 ns). Ukoliko bi npr. ulazni signal *hit* (*hit_out*) trajao više od 20 ns, ovisno o trajanju signala igrač bi dobio više karata umjesto jedne (jer je jednom pritisnuo *hit*). Slična situacija vrijedi i za ulaze *stay*, *double_down* i *split*. Trajanje bilo kojeg od ovih ulaza više od jednog perioda takta na kojemu radi cijela logika igre uzorkovao bi neispravan rad cijelog sklopa. Modul je implementiran kao automat koji detektira promjenu iz 0 u 1 te na izlazu daje jedinicu u trajanju jednog perioda taktnog signala kada detektira takvu promjenu. Izlazni signal kasni jedan takt za ulaznim signalom. Na slici 3.4. prikazan je graf stanja automata. *Edge_detector* sadrži 1b ulaze *in* (ulazni signal) i *clk* (taktni signal) te 1b izlaz *out*.



Slika 3.4. Graf stanja automata edge_detector

Automat se ponaša kao detektor sekvence 01 i reagira na pozitivan brid taktnog signala. Ulaz automata su signali *hit_out*, *stay_out*, *dd_out* i *split_out*. Stanje *s1* govori da se na ulazu dogodila 0, te u trenutku kada detektira 1 na ulazu prelazi u konačno stanje *s2* u kojemu postavlja izlaz u 1. U idućem taktu je izlaz već u 0 budući da se događa promjena stanja (prijelaz u stanje *s0* ili *s1* koji na izlazu daju 0).

U glavnom modulu *blackjack* instancirana su 4 *edge_detector* modula za svaki ulaz. Izlazi *edge_detectora* su *hit_sp*, *stay_sp*, *double_down_sp* i *split_sp*. Ti izlazi su ulazi automata koji definira cijelu logiku igre.

```
// generates a single pulse if hit, stay, double_down or split button is
pressed

edge_detector hit_button(.clk(clk), .in(hit_out), .out(hit_sp));
edge_detector stay_button(.clk(clk), .in(stay_out), .out(stay_sp));
edge_detector double_down_button(.clk(clk), .in(dd_out),
.out(double_down_sp));
edge_detector split_button(.clk(clk), .in(split_out), .out(split_sp));
```

3.2.3. Modul *card_generator*

Modul *card_generator* služi za generiranje nove karte igraču kada je zatraži (odnosno kada pritisne *hit*). Sadrži 1b ulaze *hit*, *clk* (taktni signal) te 4b izlaze *card* i *prev_card*. Modul generira novu kartu na način da se brojač *counter* uvećava svaki takt za jedan (počinje od 2 jer ne postoji karta vrijednosti 1 – drugi as se broji kao 1 ali to je osigurano unutar logike igre). Kada dosegne 11 brojač se vraća na početak budući da ne postoji karta veća od 11. U trenutku kada igrač pritisne *hit* na izlazu *card* se daje njegova nova karta koja je vrijednost do koje je do tada izbrojao brojač *counter*. Pamti se i prethodna karta koja se daje na izlazu *prev_card* kako bi mogli ispravno implementirati *split* potez (znati da je igrač dobio dvije iste karte).

```
module card_generator(input hit, input clk, output reg [3:0] card, output reg
[3:0] prev_card);

reg [3:0] counter;

initial begin
counter=2;
end

always@(posedge clk) begin
counter<=counter+1'b1;
if(counter==11)
counter<=2;
end

always@(posedge hit) begin
card<=counter;
prev_card<= card;
end
```

```
endmodule
```

Ulaz *card_generatora* je signal *hit_sp*, a izlazi su *card* i *prev_card* glavnog modula.

```
// genereates a new card
card_generator new_card(.hit(hit_sp), .clk(clk), .card(card),
.prev_card(prev_card));
```

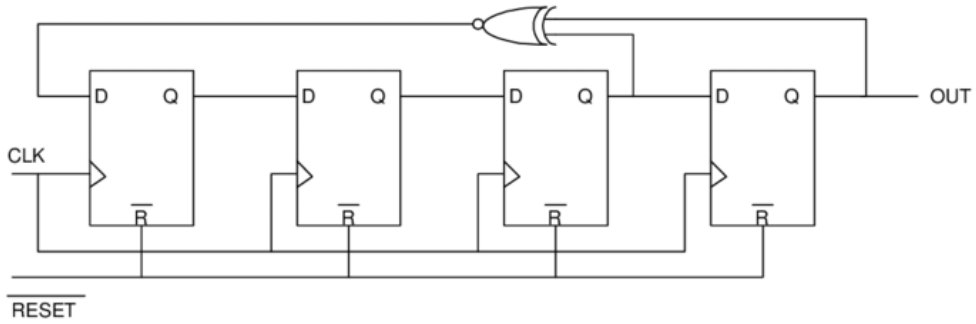
3.2.4. Modul *lfsr*

Modul *lfsr* predstavlja generator slučajnih brojeva koji generira karte djelitelja. Djelitelj je automatiziran te prelaskom na njegovu igru on dobiva svoje karte. Ako bi koristili modul *card_generator* za generiranje djeliteljevih karata on bi dobivao uzastopne karte (npr. ako je igračeva zadnja karta bila 8, djelitelj bi dobio 9 i 10). Kako bi dobivao „random“ karte kreiran je novi modul *lfsr* (generator pseudo-slučajnih brojeva, LFSR – Linear Feedback Shift Register). Modul sadrži 1b ulaz clk (taktni ulaz) i rst (za resetiranje generatora) te 4b izlaz out (na kojemu daje karte djelitelju). Budući da karta ne smije biti manja od 2 i veća od 11, te vrijednosti su zamijenjene kartama koje su moguće.

```
module lfsr (out, clk, rst);

  output reg [3:0] out;
  reg [3:0] number;
  input clk, rst;

  wire feedback;

  assign feedback = ~(number[3] ^ number[2]);


always @(posedge clk, posedge rst)
  begin
    if (rst)
      number = 4'b0;
    else
      number = {number[2:0],feedback};
  end

  always @(*) begin
  case(number)
  0: out=5;
  1: out=11;
```

```
  14: out=8;
  13: out=3;
  12: out=10;
  default: out=number;
  endcase
  end
endmodule
```

Shema sklopa prikazana je na slici 3.5. Sklop XNOR-a dva najznačajnija bita i vraća ih na ulaz (*feedback*). Na svaki pozitivan brid taktnog signala svi bitovi se *shiftaju* za jednu poziciju.



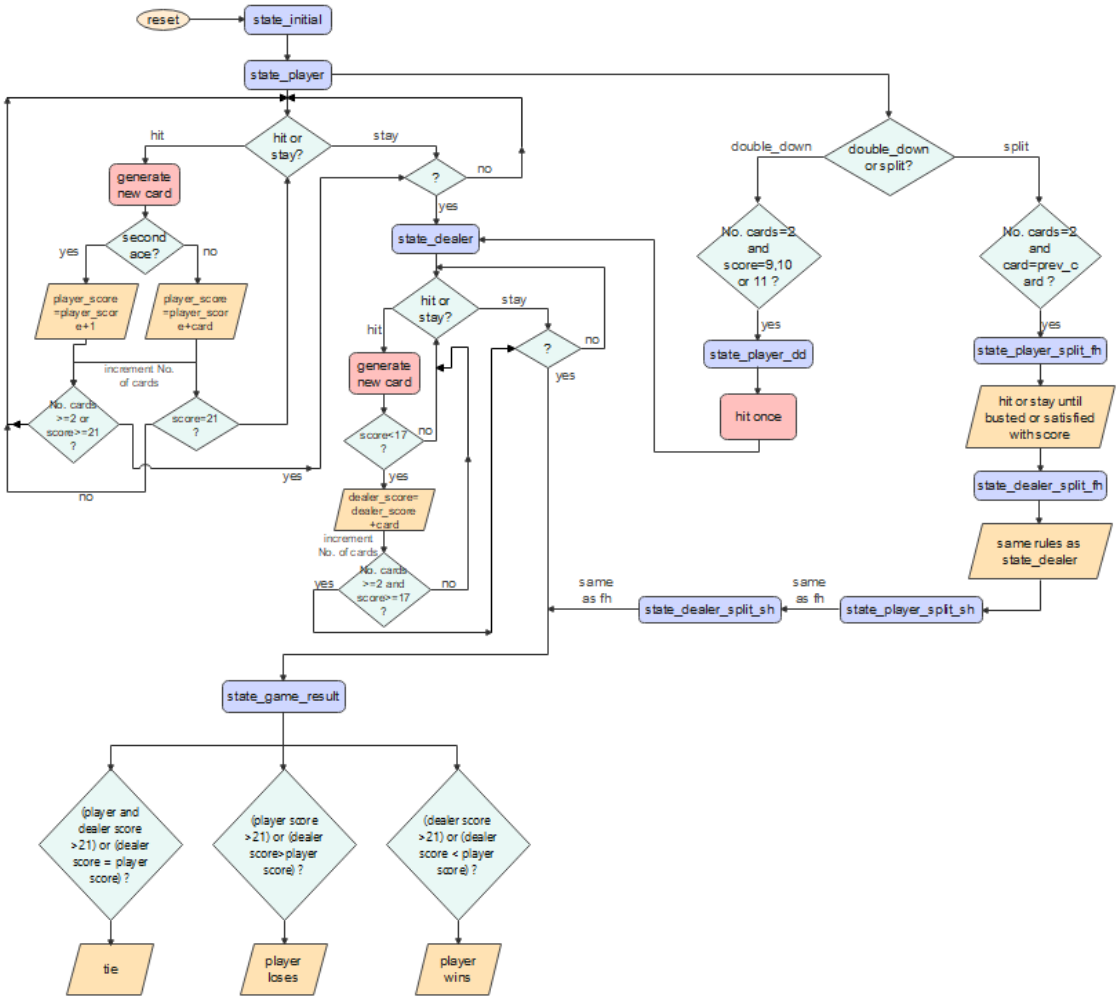
Slika 3.5. Shema 4b LFSR-a

Ovim sklopom je postignuto slučajnije generiranje djeliteljevih karti. Iako LFSR generira uvijek istu sekvencu brojeva, dosta ovisi o potezima igrača u kojem trenutku će se prijeći u stanje djelitelja te će tako karte djelitelja ovisiti o tome koji je zadnji broj iz niza LFSR generirao do tog trenutka. Djelitelju će karte biti dodijeljene u minimalno dva takta (ako je rezultat 17 ili viši). Izlaz *lfsra* je *d_card* glavnog modula.

```
lfsr d_new_card(.clk(clk), .rst(reset), .out(d_card));
```

3.2.5. Logika igre

Logika igre je implementirana u top modulu *blackjack*. Realizirana je kao automat prikazan na slici 3.6. Automat se sastoji od 9 stanja: *state_initial*, *state_player*, *state_dealer*, *state_player_dd*, *state_player_split_fh*, *state_player_split_sh*, *state_dealer_split_fh*, *state_dealer_split_sh* i *state_game_result*. Kako bi igrač započeo igru potrebno je pritisnuti *reset* za prijelaz u stanje *state_initial* te se odmah u idućem taktu prelazi u *state_player* u kojem započinje igra.



Slika 3.6. Dijagram stanja

Prilikom pritiska na *reset*, sve interne varijable (rezultati, raspoloživi novac, broj karata) se postavljaju na početne vrijednosti. Raspoloživi novac je fiksan i u početku postavljen na 100 (varijabla *bet* koja se prikazuje kasnije na LCD zaslonu). Ulog je također fiksan te za svaku partiju iznosi 10. Igrač igru smije započeti/nastaviti samo ako je raspoloživi novac minimalno 10 iz razloga što je 10 ulog te ako ima manje od toga ne može uložiti 10. Igrač može birati *hit* sve dok nije zadovoljan rezultatom ili premaši 21 (minimalno 2 puta mora pritisnuti *hit* kako bi dobio početne dvije karte i tek nakon toga smije pritisnuti *stay* čime se prelazi na djeliteljevu igru tj. u stanje *state_dealer*). Signali na koje reagira automat su *hit_sp*, *stay_sp*, *double_down_sp* i *split_sp* iz razloga koji je prethodno objašnjen u poglavlju koje se odnosu na modul *edge_detector*. Unutar stanja *state_player* osigurano je da se dobiveni as računa kao 1 ili 11 ovisno o tome što ide igraču u prilog. Igrač također može birati *double down* ili *split* ukoliko zadovoljava sve uvijete za to (za double down treba imati dvije karte, dosadašnji rezultat 9, 10 ili 11 te ulog minimalno jednak 20 budući da se ulog prilikom *double downa* dupla; za *split* je potrebno imati prve dvije karte iste).

```
        state_player:  // Player turn to draw cards
        if(bet >= 10)
        begin
                begin

                if(hit_sp == 1) begin
                    if(card == 11 && player_score >
10)                                 // If the player gets another ace, it
counts as 1 (in his favour)
                        begin
                        player_score = player_score + 1'b1;
                        number_of_cards_player = number_of_cards_player +
1'b1;
                        end
                else
                    begin
          // Add new card to score
                    player_score = player_score + card;
                    number_of_cards_player = number_of_cards_player + 1'b1;
                    end
                end


                if((stay_sp == 1 && number_of_cards_player >=2 ) ||
player_score >= 21) // If the player decides to stay with min. 2 cards or the
score is over 21
                    begin
          // it's the dealers turn
                        state <= state_dealer;
                    end
                // DOUBLE DOWN
                if(double_down_sp == 1 && number_of_cards_player==2 &&
(player_score==9 || player_score==10 || player_score==11 ) && bet >= 20)
                    begin
                        state <=
state_player_dd;                                        // If players score
equals 9, 10 or 11 after drawing 2 cards, he's allowed
                    end
          // to double down and recieve only one card in advance
                // SPLIT
                if(split_sp == 1 && number_of_cards_player == 2 &&
card==prev_card)     // If the first two cards are the same value, player's
alowed to split
                    begin
          // and treat the splitted cards as seperate hands
                        split_fh_p_score = card;
                        split_sh_p_score = card;
                        splitted = 0;
                        state <= state_player_split_fh;
```

```
                    end


                end
            end
```

Ukoliko se igrač odluči za *double down*, prelazi se u stanje *state_player_dd*. Unutar njega, igrač je ograničen na dobivanje još samo jedne karte pritiskom na hit. Ukoliko pritisne *hit* više puta, ništa se neće dogoditi budući da varijabla *dd_flag* osigurava da se samo prvi *hit* računa. Nakon toga je potrebno da igrač pritisne *stay* kako bi se nastavilo s igrom tj. prijelaz u stanje *state_dealer*.

```
        state_player_dd:   // DOUBLE DOWN
            begin
                if(hit_sp == 1 && dd_flag == 1)
begin
                    if(card == 11 && player_score >
10)                                 // If the player gets another ace, it
counts as 1 (in his favour)
                        begin
                        player_score = player_score + 1'b1;
                        number_of_cards_player = number_of_cards_player +
1'b1;
                        dd_flag =
0;                                                  // Set dd_flag to 0 to
disallow player to draw more than one card
                        end
                else
                    begin
          // Add new card to score
                    player_score = player_score + card;
                    number_of_cards_player = number_of_cards_player + 1'b1;
                    dd_flag =
0;                                                      // Set dd_flag to 0
to disallow player to draw more than one card
                    end
                end

                if(stay_sp == 1 && dd_flag == 0
)                                           // Player must stay after drawing
one card
                    begin

                        state <= state_dealer;
                    end
            end
```

U stanju *state_dealer*, djelitelj dobiva svoje karte. Budući da su njegovi potezi automatizirani, a karte dobiva iz LFSR-a, to se obavi u minimalno 2 takta. Varijabla *hit_d* služi kako bi pripazili da dobije minimalno dvije karte te po potrebi još karata u slučaju da je rezultat manji od 17. Nakon što djelitelj dobije sve svoje karte može se prijeći u konačno stanje – *state_game_result* u kojemu se određuje rezultat igre. Potezi djelitelja su uvijek isti te ne ovise o tome za kakav potez se igrač odluči tako da u stanjima *state_dealer_split_fh* i *state_dealer_split_sh* se događa ista stvar kao i u stanju *state_dealer* samo se rezultati upisuju u druge varijable budući da ih moramo posebno pamtiti.

```
        state_dealer:   // Dealer turn to draw cards
            begin

                if(hit_d == 1) begin
                dealer_score = dealer_score + d_card;
                number_of_cards_dealer = number_of_cards_dealer + 1'b1;
                    if(number_of_cards_dealer >=2 && dealer_score >=
17)                                             // If dealers score is
under 17, dealer must draw another card until his score
                        hit_d=0;
                end

                else     // If dealer stays with min. 2 cards and his score
>=17 move to game result state
                    begin
                        hit_d=1;
                        state <= state_game_result;
                    end
            end
```

Ukoliko se igrač odluči za *split*, prelazi se u stanje *state_player_split_fh*. Pravila su praktički ista kao i u uobičajenoj igri osim što se rezultat pamti u zasebnoj varijabli budući da je potrebno svaku kartu tretirati kao posebnu ruku. Varijabla *split_flag* osigurava da igrač nakon *splita* uzme minimalno još jednu kartu. Nakon što igrač odigra prvu ruku *splita*, prelazi se u stanje *state_dealer_split_fh* u kojem djelitelj dobiva svoje karte, nakon toga u stanje *state_player_split_sh* u kojem igrač igra na svoju drugu ruku te nakon toga u stanje *state_dealer_split_sh* u kojem djelitelj dobiva karte za svoju drugu ruku. Nakon što svi dobiju svoje karte za obje ruke, prelazi se u stanje *state_game_result* gdje se određuje ishod igre.

```
        state_player_split_fh:   // Split - players first hand
            begin
            if(hit_sp == 1) begin
                    if(card == 11 && split_fh_p_score >
10)
                        begin
                        split_fh_p_score = split_fh_p_score + 1'b1;
                        //number_of_cards_player = number_of_cards_player +
1'b1;
                        split_flag = 0;
                        end
                else
                    begin

                    split_fh_p_score = split_fh_p_score + card;
                    //number_of_cards_player = number_of_cards_player + 1'b1;
                    split_flag = 0;
                    end
                end

                if((stay_sp == 1 && split_flag==0) || split_fh_p_score >=
21)
                    begin

                        state <= state_dealer_split_fh;
                        split_flag=1;
                    end
            end
```

U stanju *state_game_result* određuje se rezultat igre te igrač gubi/dobiva novac ovisno o ishodu igre. Varijabla *result_flag* služi tome da se rezultat odredi samo u jednom taktu kako ne bi došlo do toga da se u svakom idućem taktu dok je automat u tom stanju dodaje/oduzima novac dok se igrač ne odluči pritisnuti tipkalo za novu partiju. Varijabla *splitted* govori o tome da li je igrač pritisnuo *split* te ukoliko je potrebno je posebno promatrati varijable rezultata za takav slučaj. Ukoliko igrač pobjedi u uobičajenoj partiji s rezultatom 21, dobiva novac u omjeru 3:2 (pošto je ulog 10, dobiva 15). Ako pobjedi s rezultatom koji nije 21 dobiva novac u omjeru 1:1 (dobiva 10). U slučaju da igrač izgubi, gubi 10. U slučaju da je se odlučio za *double down* i pobijedio, dobiva dupli iznos (20), u suprotnome gubi dupli iznos. Za *split* je isplata u omjeru 1:1 na svaku ruku (čak i kada pobijedi s rezultatom 21).

```
state_game_result:    // Determines the result of the game (for the player)

        begin
```

```
        if(result_flag) begin

        result_flag = 0;
        if(splitted) begin
                if((player_score > 21) && (dealer_score >
21))                        // If both are over 21, it's a tie
                    tie = 1;
                else if(player_score >
21)  begin                                          // If the player is over
21, he loses
                    lose = 1;
                    bet = bet - 10;
                    end
                else if(dealer_score >
21)  begin                                          // If the dealer is over
21, player wins
                    win = 1;
                        if(dd_flag == 0) begin
                        bet = bet + 20;
                        end
                        else begin
                            if(player_score==21)
                            bet = bet + 15;
                            else bet = bet + 10;
                        end
                    end
                else if (dealer_score > player_score)
begin                                   // If the dealers score is over the
players score, player loses
                    lose = 1;
                        if(dd_flag == 0)
                        bet = bet - 20;
                        else
                        bet = bet - 10;
                    end
                else if (dealer_score < player_score)
begin                                   // If player score is over the dealers
score, he wins
                    win = 1;
                    if(dd_flag == 0) begin
                        bet = bet + 20;
                    end
                    else begin
                        if(player_score==21)
                        bet = bet + 15;
                        else bet = bet + 10;
                        end
                    end
```

```
                else if (dealer_score ==
player_score)                                   // If scores are equal, it's a
tie
                    tie = 1;

            end
```

Nakon određivanja rezultata, igrač može nastaviti igru ukoliko ima dovoljno novca. Nastavak igre se odvija pritiskom na *split* čime se automat vraća u stanje *state_initial* te započinje nova partija.

3.2.6. Modul *bin2bcd_5b* i *bin2bcd_8b*

Moduli *bin2bcd_5b* i *bin2bcd_8b* služe za pretvaranje rezultata iz binarnog koda u BCD kod kako bi se mogao rezultat ispravno prikazati na LCD zaslonu. LCD zaslon koristi ASCII kod. Tablica simbola je prikazana na slici 3.7. Npr. ako igračev rezultat iznos 19 i želimo ga prikazati na LCD zaslonu, potrebno je prvo prikazati znamenku 1 zatim znamenku 9. Iz binarnog zapisa broja 19 (0001 0011) ne možemo zaključiti da se dekadski ekvivalent sastoji od znamenki 1 i 9. Zbog toga je potrebno izvršiti pretvorbu takvog zapisa u BCD zapis. BCD ili binarno kodirani decimalni broj (engl. *binary-coded decimal*) ime je za vrstu kodiranja odnosno predstavljanja decimalnih brojeva u binarnom obliku, u kojem je svaki decimalni broj predstavljen s fiksnim brojem binarnih brojeva. Tako bi BCD ekvivalent broja 19 bio 0001 1001 (4 najznačajnija bita kada se pretvore iz binarnog koda u dekadski predstavljaju broj 1, a 4 najmanje značajna broj 9).

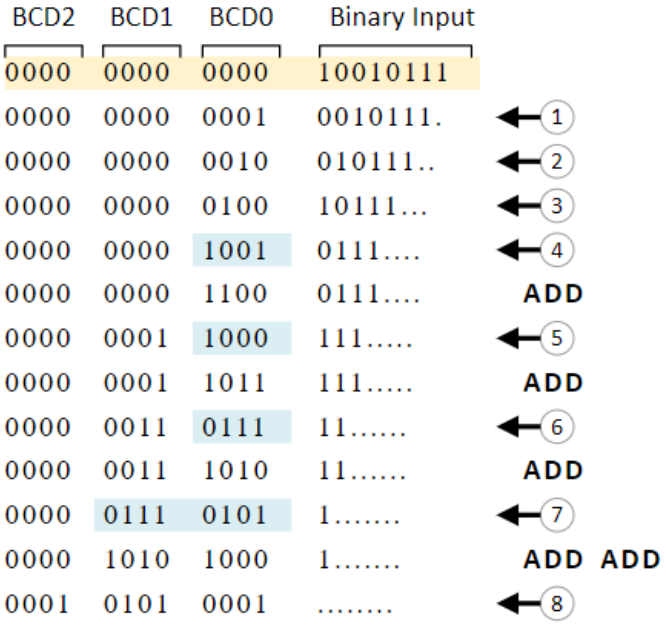
Slika 3.7. Tablica LCD simbola

U glavnom modulu *blackjack* instancirano je *5 bin2bcd_5b* modula (ulazi su 5b) i 1 *bin2bcd_8b* (ulaz je 8b). Moduli *bin2bcd_5b* pretvaraju varijable (ulaze) *player_score*, *dealer_score*, *split_fh_p_score*, *split_sh_p_score*, *split_fh_d_score* i *split_sh_d_score* u BCD ekvivalente. Modul *bin2bcd_8b* pretvara varijablu *bet* u BCD ekvivalent. Na izlazu daju BCD ekvivalent ulazne vrijednosti na varijable *p_score_bcd*, *d_score_bcd*, *p_fh_bcd*, *p_sh_bcd*, *d_fh_bcd*, *d_sh_bcd* i *bet_bcd* koji su ulazi modula *lcd* za LCD prikaz. Izlazi su 16b.

```
bin2bcd_5b bcd_p_score(.bin(player_score), .bcd(p_score_bcd));
bin2bcd_5b bcd_d_score(.bin(dealer_score), .bcd(d_score_bcd));
bin2bcd_5b bcd_p_score_fh(.bin(split_fh_p_score), .bcd(p_fh_bcd));
bin2bcd_5b bcd_p_score_sh(.bin(split_sh_p_score), .bcd(p_sh_bcd));
bin2bcd_5b bcd_d_score_fh(.bin(split_fh_d_score), .bcd(d_fh_bcd));
bin2bcd_5b bcd_d_score_sh(.bin(split_sh_d_score), .bcd(d_sh_bcd));
bin2bcd_8b bcd_bet(.bin(bet), .bcd(bet_bcd));
```

Pretvaranje prirodnog binarnog koda u BCD kod se obavlja tzv. *double dabble* algoritmom.

Npr. potrebno je naći BCD ekvivalent dekadskog broja 151 (binarno 1001 0111). Algoritam je prikazan na slici 3.8. U početku se BCD znamenke postave na vrijednost nula. Prvi korak je posmak binarnog zapisa za jedno mjesto u lijevo. U nastavku se binarna vrijednost posmiče u lijevo dok je jedna od BCD znamenki jednaka ili veća od 5 (označeno plavom bojom u koraku 4, BCD0 je 1001 odnosno 9 koji je veći od 5). Tada se toj znamenki dodaje 3 (9+3=12, binarno 1100). Nakon toga se ponovno vrši posmak sve dok opet neka od BCD znamenki ne iznosi 5 ili više nakon čega se toj znamenki dodaje 3 i ponovno vrši posmak. Postupak se ponavlja sve dok se ne obrade svi bitovi ulazne binarne riječi. Tako na kraju možemo zaključiti da je BCD2 0001 odnosno dekadski znamenka 1, BCD1 0101 odnosno dekadski znamenka 5 i BCD0 0001 odnosno dekadski znamenka 1.



Slika 3.8. Prikaz double dabble algoritma

U implementaciji algoritma prvo se izlaz *bcd* na koji dajemo BCD vrijednost ulaznog binarnog broja postavi u 0. For petlja se koristi kako bi izvršili posmak u desno onoliko puta koliko ima ulaznih bitova. Unutar nje se vrši provjera da li je neka od BCD znamenaka veća ili jednaka 5 i ako je dodaje joj se 3. Nakon toga se vrši posmak za jedno mjesto ulijevo.

```
module bin2bcd_5b(
   input [4:0] bin,
   output reg [15:0] bcd
   );

integer i;

always @(bin) begin
```

```
    bcd=0;
    for (i=0;i<5;i=i+1) begin                    //Iterate once for each bit in
input number
        if (bcd[3:0] >= 5) bcd[3:0] = bcd[3:0] + 3;     //If any BCD digit is
>= 5, add three
    if (bcd[7:4] >= 5) bcd[7:4] = bcd[7:4] + 3;
    if (bcd[11:8] >= 5) bcd[11:8] = bcd[11:8] + 3;
    if (bcd[15:12] >= 5) bcd[15:12] = bcd[15:12] + 3;
    bcd = {bcd[14:0],bin[4-i]};             //Shift one bit, and shift in
proper bit from input
    end
end
endmodule
```

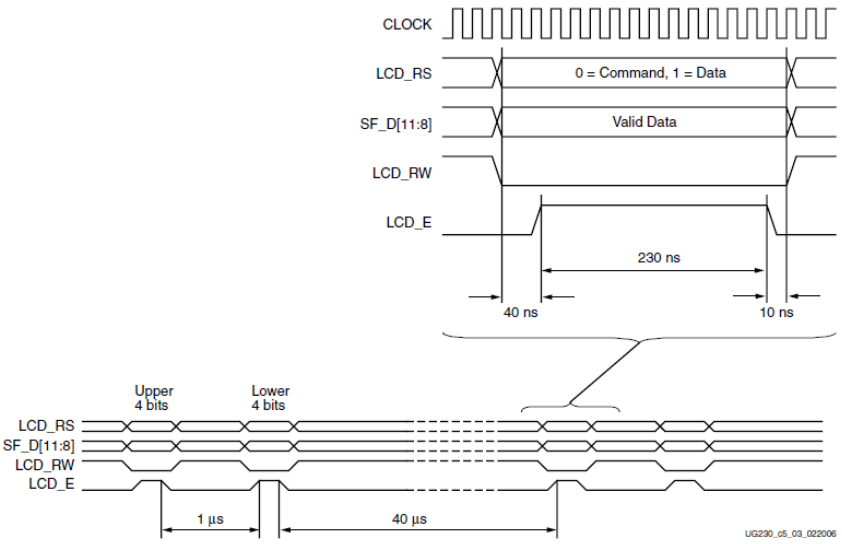
3.2.7. Modul *lcd*

Modul lcd je odgovoran za ostvarivanje komunikacije s LCD zaslonom te prikaz rezultata na zaslonu. Sadrži ulaze clk (taktni ulaz), 7 ulaza putem kojih se modulu prosljeđuju BCD vrijednosti *p_score_bcd*, *d_score_bcd*, *p_fh_bcd*, *p_sh_bcd*, *d_fh_bcd*, *d_sh_bcd* i *bet_bcd*, 4 podatkovna 1b ulaza *a*, *b*, *c* i *d*, kontrolni ulazi za upravljanje zaslonom *sf_e*, *e*, *rs*, *rw* te ulaz *trigger*.

```
// LCD
lcd lcdTest( .clk(clk), .p_score_bcd(p_score_bcd), .d_score_bcd(d_score_bcd),
.p_fh_bcd(p_fh_bcd), .p_sh_bcd(p_sh_bcd), .d_fh_bcd(d_fh_bcd),
.d_sh_bcd(d_sh_bcd),
 .bet_bcd(bet_bcd), .sf_e(sf_e), .e(e), .rs(rs), .rw(rw), .d(d), .c(c), .b(b),
.a(a), .trigger(trigger));
```

Unutar jednog perioda slanja ili čitanja s zaslona, moguće je slati/čitati po 4 bita (to su 4 podatkovna bita). U implementaciji igre *blackjack* LCD zaslon je korišten isključivo za pisanje. Da bi komunikaciju mogli ostvariti (tj. da bi mogli nešto pisati po zaslonu) potrebno je ispoštovati komunikaciju koja je prikazana na slici 3.9. Kako bi mogli pisati po zaslonu potrebno je da su prilikom slanja podataka izlazi LCD_RW i LCD_RS postavljenu u 0 i 1 respektivno. Zbog toga se prilikom slanja podatka za ispis uvijek šalju ovakva dva bita preko svojih linija (izlazi *rw* i *rs*). Isto tako za ispis je nužno postavljanje linije LCD_E (izlaz *e* modula *lcd*) u 1 minimalno 40 ns nakon što su linije *rw* i *rs* stabilno postavljene. Linija *e* mora biti postavljena minimalno 230 ns. Kada je LCD_E postavljen u 1, omogućeno je čitanje/pisanje po zaslonu. Inače, zaslon je „isključen“. LCD_RS mora biti u 1 kako bismo mogli zaslon koristiti za čitanje/pisanje, u suprotnome se koristi za nešto drugo. LCD_RW definira da li se

čita ili piše po zaslonu. Za pisanje potrebno je da se on nalazi u 0. Podatci se unutar jednog ciklusa šalju putem 4 1b podatkovne linije. Tako se unutar jednog ciklusa može poslati samo 4 bita. Npr. za ispisivanje nekog simbola na zaslon, potrebno je u jednom ciklusu poslati gornja 4 bita (engl. *upper nibble*) te nakon toga donja 4 bita (engl. *lower nibble*) simbola iz tablice simbola (slika 3.7.).



Slika 3.9. LCD komunikacija

Modul *lcd* je realiziran kao automat s 2 stanja. U stanju 0 se vrši prikaz rezultata tako da se ispsiuje znak po znak na zaslon. Nakon što se obavi ispis prelazi se u stanje 1 u kojemu zaslon miruje (prikazuje ono što je u prethodnome stanju ispisao) sve dok se ne dogodi okidni događaj koji mu signalizira da se dogodila promjena neke od varijabli rezultata koje prikazuje. Okidni događaj se šalje putem ulaza *trigger*. Okidni događaji su kada igrač pritisne neko od tipkala koja se koriste za igru (*hit*, *stay*, *double down*, *split* ili *reset*). Kada se dogodi okidni događaj, na zaslon se ponovno ispisuje novi rezultat igre budući da je u trenutcima kada igrač pritisne neko tipkalo došlo do promjene rezultata.

```
wire trigger;
assign trigger=(hit_out || stay_out || dd_out || split_out || reset)?1:0;
```

U nastavku je dan opis kako se LCD postavlja za ispis (sve ono što se događa u stanju 0). Na slici 3.10. dan je pregled naredbi za postavljanje zaslona. Neke od njih su objašnjene u nastavku.

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Cursor Home	0	0	0	0	0	0	0	0	1	-
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display On/Off	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	-	-

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Function Set	0	0	0	0	1	0	1	0	-	-
Set CG RAM Address	0	0	0	1	A5	A4	A3	A2	A1	A0
Set DD RAM Address	0	0	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Address	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG RAM or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Read Data from CG RAM or DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0

Slika 3.10. Pregled naredbi za postavljanje rada zaslona

Kako bi zaslon ispravno radio i kako bi vodili računa o tome koliko je je vremena prošlo od postavljanja pojedinih linija potreban nam je taktni signal *clk*. Varijabla *count* broji taktove te se razmatraju gornjih 7 bitova 27b *counta* kako bi osigurali da je prošlo minimalno vrijeme između slanja naredbi za postavljanje zaslona. *Function Set* naredba postavlja veličinu podataka koji se šalju, broj linija zaslona te font. Pločica podržava samo jedan *Function Set* te se može samo poslati naredba 0x28 (u ciklusima od po 4 bita). Nakon toga se šalje naredba *Entry Mode* naredba koja definira u kojem smjeru se pomiče kursor za ispis (I/D bit, 1 za automatsko inkrementiranje adrese za ispis, pomicanje kursora u desno) te želi li se koristiti pomicanje zaslona (S bit, 0 za onemogućeno pomicanje). *Display On/Off* naredba se šalje kako bi omogućili/onemogućili prikaz kursora (C bit, 0 za nevidljiv kursor) , blinkanje kursora (B bit, 0 za onemogućeno blinkanje), prikaz znakova (D bit, 1 za prikaz). *Clear Display* naredba „čisti“ prikaz i postavlja kursor na gornju lijevu poziciju (fiksna naredba, 0x01).

```
count <= count +1;

        // mislim da mogu smanjiti i na 26:18 onda dolje 17 u refresh
        case ( count[ 26 : 20 ] )   // as top 6 bits change
// power-on init can be carried out before this loop to avoid the flickers
            0: code <= 6'h03;           // power-on init sequence
            1: code <= 6'h03;           // this is needed at least once
            2: code <= 6'h03;           // when LCD's powered on
            3: code <= 6'h02;           // it flickers existing char display
```

```
// Table 5-3, Function Set
// send 00 and upper nibble 0010, then 00 and lower nibble 10xx
            4: code <= 6'h02;           // Function Set, upper nibble 0010
            5: code <= 6'h08;           // lower nibble 1000 (10xx)

// Table 5-3, Entry Mode
// send 00 and upper nibble 0000, then 00 and lower nibble 0 1 I/D S
// last 2 bits of lower nibble: I/D bit (Incr 1, Decr 0), S bit (Shift 1, 0
no)
            6: code <= 6'h00;       // see table, upper nibble 0000, then
lower nibble:
            7: code <= 6'h06;           //  0110: Incr, Shift disabled

// Table 5-3, Display On/Off
// send 00 and upper nibble 0000, then 00 and lower nibble 1DCB:
// D: 1, show char represented by code in DDR, 0 don't, but code remains
// C: 1, show cursor, 0 don't
// B: 1, cursor blinks (if shown), 0 don't blink (if shown)
            8: code <= 6'h00;           // Display On/Off, upper nibble 0000
            9: code <= 6'h0C;           // lower nibble 1100 (1 D C B)

// Table 5-3 Clear Display, 00 and upper nibble 0000, 00 and lower nibble 0001
            10: code <= 6'h00;      // Clear Display, 00 and upper nibble 0000
            11: code <= 6'h01;      // then 00 and lower nibble 0001
```

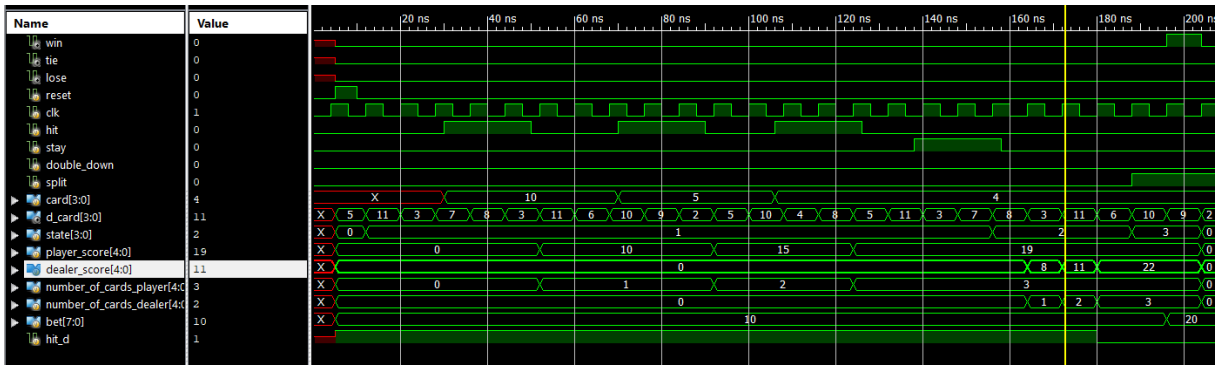
Nakon postavljanja rada zaslona mogu se slati podaci. Podatke šaljemo naredbom *Write Data to CG RAM or DD RAM*. Prva dva bita naredbe su *rw* i *rs* linije (0 i 1 respektivno). Dalje se šalje ASCII kod podatka koji želimo ispisati. Npr. za ispis rezultata igrača:

```
            12: code <= 6'h25;      // 'P' high nibble
            13: code <= 6'h20;      // 'P' low nibble
            14: code <= 6'h23;      // :
            15: code <= 6'h2A;
            16: code <= 6'h23;      // first dig
            17: code <= {2'h2,p_score_bcd[7:4]};
            18: code <= 6'h23;      // second dig
            19: code <= {2'h2,p_score_bcd[3:0]};
```

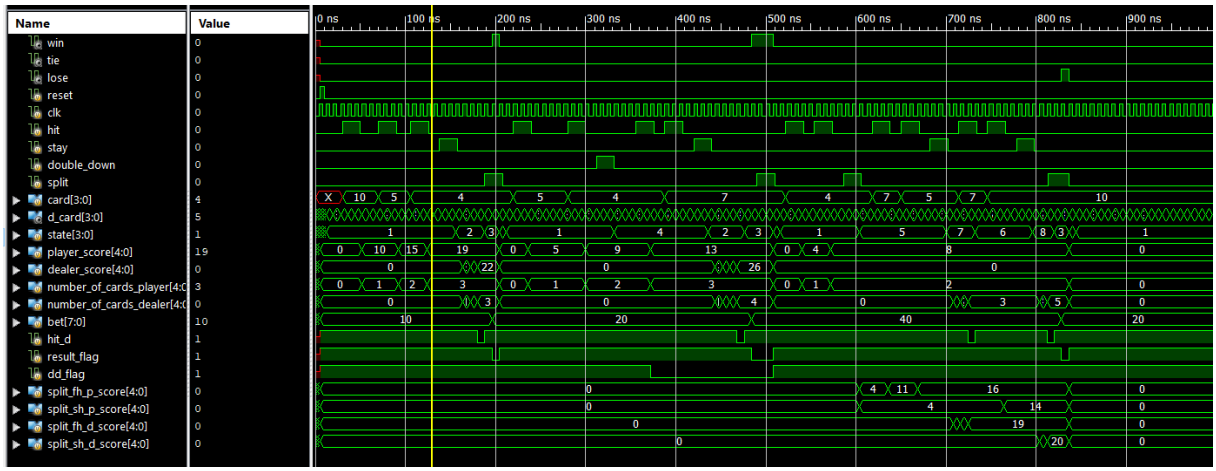
Nakon što se ispišu svi znakovi koje želimo, šalje se naredba *Read Busy Flag and Address*. Postavljanjem BF bita u 1, onemogućuje se čitanje iduće naredbe sve dok je taj bit postavljen. Izlaz *sf_e* uvijek mora biti postavljen u 1 kako bi uopće koristili LCD zaslon.

3.3. Rezultati simulacije

Prije testiranja na pločici, potrebno je simulirati rad sklopa kao i pojedinih modula kako bi se uvjerili da rade ono što trebaju. Pri tome su simulirani radovi modula: *blackjack* (top modul), *edge_detector*, *card_generator*, *lfsr*, *bin2bcd*. Kod za modul *lcd* se ne može simulirati te je potrebno provjeriti njegovu funkcionalnost na samoj pločici. Rezultati simulacija glavnog modula *blackjack* su prikazani na slikama 3.11. i 3.12. Za ostale module testni moduli su dani kao prilog na kraju. Kako bi ispravno simulirali rad sklopa te prijelaze iz stanja u stanje uklonjen je modul *card_generator* te dodan ulaz *card* kako bi se lakše simulirali svi potrebni uvjeti za *double down* i *split* (namještanje karata u testnom modulu).



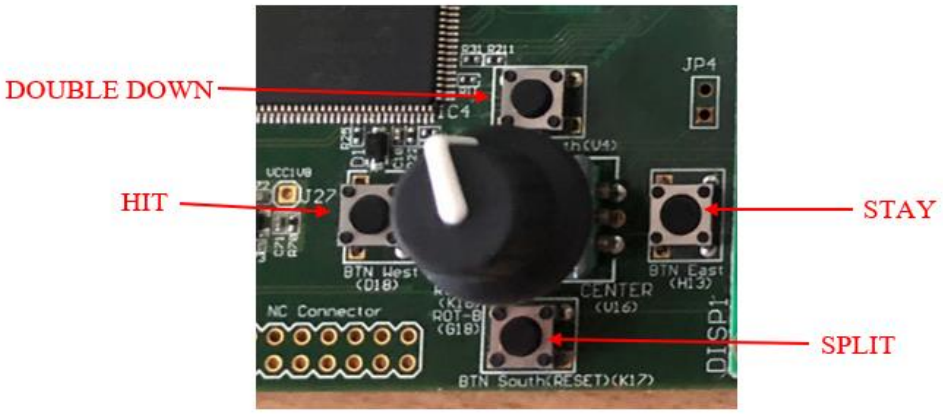
Slika 3.11. Simulacija top modula blackjack (uobičajena igra)



Slika 3.12. Simulacija top modula blackjack (obična igra, double down i split)

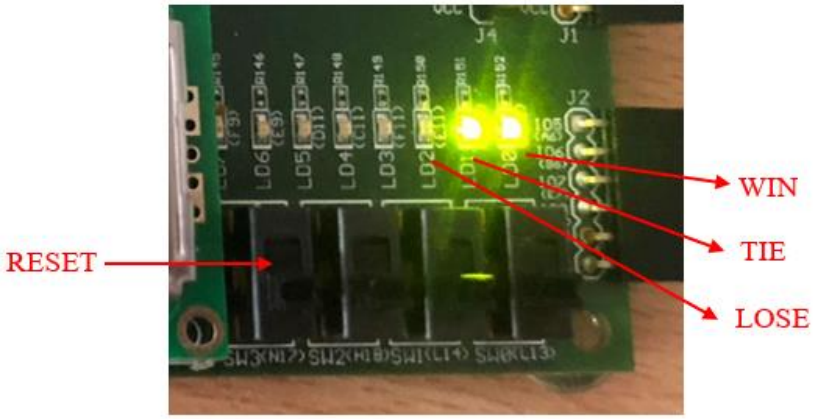
3.4. Korišteni ulazi i izlazi na Spartan-3E i LCD prikaz

Kao što je i prethodno spomenuto, za ulaze *hit*, *stay*, *double_down* i *split* korištena su 4 tipkala koja se nalaze na pločici. Kako su korišteni prikazano je na slici 3.13.



Slika 3.13. Korištena tipkala

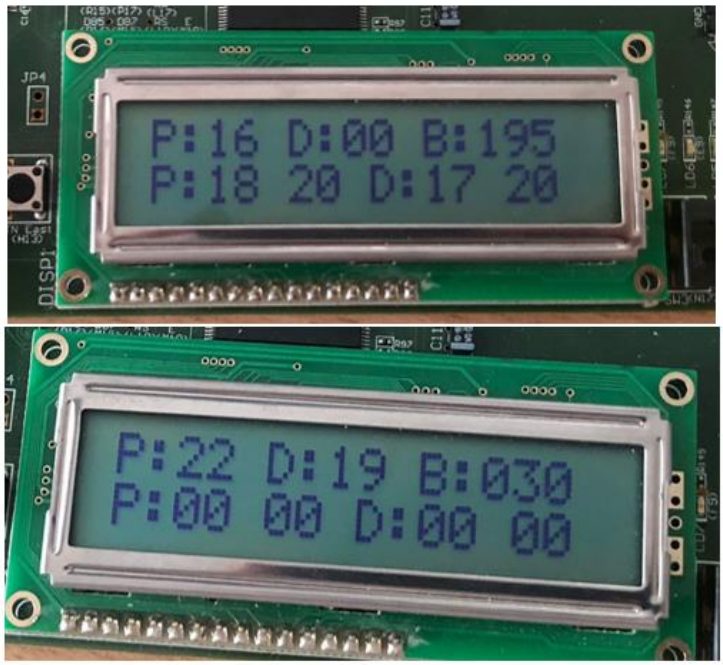
Za ulaz *reset* korištena jedna sklopka, a za izlaze *win*, *lose* i *tie* korištene su LED diode. (prikazano na slici 3.14.).



Slika 3.14. Ulaz reset i izlazi win, lose, tie

Rezultati su prikazani na LCD zaslonu kao na slici 3.15. Na gornjem dijelu slike prikazan je rezultat igre kada se igrač odluči za *split*. U prvom redu znakova prikazuju se varijable *player_score*, *dealer_score* te trenutno stanje raspoloživog novca *bet*. Igrač je dobio dvije 8 kao prve dvije karte te nakon toga odabrao *split*. U drugom redu znakova prikazani su redom *split_fh_p_score*, *split_sh_p_score*, *split_fh_d_score* te *split_sh_d_score*. Igrač je u tom slučaju pobijedio na prvu ruku, a izjednačeno je na drugu ruku.

U drugom dijelu slike, prikazan je prikaz rezultata „uobičajene“ igre.



Slika 3.15. Prikaz rezultata igre na LCD zaslonu

4. ZAKLJUČAK

Igra *blackjack* uspješno je realizirana i testirana na Spartan-3E pločici. Mana opisane implementacije je prikaz na LCD zaslonu koji je spor. Ispis znakova traje oko 2 sekunde prilikom čega igrač ne smije pritiskat neke od ulaznih tipkala jer inače neće vidjeti svoj trenutni rezultat (igrač treba pričekati dok se završi sa ispisom na ekranu kako bi nastavio normalno igrati). Kao moguće poboljšanje igre moguće je koristiti VGA izlaz na pločici te prikazivati rezultate na ekranu ili kreirati ljepši prikaz. Isto tako moguće je implementirati unos uloga preko tipkovnice tako da igrač može sam odlučiti koliko želi uložiti. Ulog u danoj implementaciji je fiksan.

5. LITERATURA

- https://www.admiral.hr/app/info/kako-igrati-blackjack
- https://bicyclecards.com/how-to-play/blackjack/
- https://nandland.com/debounce-a-switch/
- https://www.compuphase.com/electronics/debouncing.htm
- https://github.com/SafaKucukkomurler/verilog-button-debouncer/blob/master/sources_1/new/debouncer.v
- https://www.researchgate.net/figure/A-4-bit-linear-feedback-shift-register-circuit_fig8_238687766
- https://www.realdigital.org/doc/6dae6583570fd816d1d675b93578203d
- Spartan-3E Starter Kit Board User Guide, 2006.
- https://github.com/ColtPtrHun/LCD_Verilog_Spartan3E/blob/master/TestLCD.v

PRILOG A – BLACKJACK

Verilog modul:

```
`timescale 1ns / 1ps

// triba maknit input card, odkomentirat card gen i var + LCD instanca pa
odkomentirat bin2bcd

module blackjack(

    input reset,
    input clk,
    input hit,
    input stay,
    input double_down,
    input split,
    //input [3:0] card,

    output reg win,
    output reg tie,
    output reg lose,
    output sf_e, e, rs, rw, d, c, b, a
    );


wire[3:0] card;
wire [3:0] prev_card;
wire hit_sp, stay_sp, double_down_sp, split_sp;
wire hit_out, stay_out, dd_out, split_out;
wire [3:0] d_card;

dbc hit_debounce(.signal_i(hit),.clk(clk), .signal_o(hit_out));
dbc stay_debounce(.signal_i(stay),.clk(clk), .signal_o(stay_out));
dbc dd_debounce(.signal_i(double_down),.clk(clk), .signal_o(dd_out));
dbc split_debounce(.signal_i(split),.clk(clk), .signal_o(split_out));

// generetes a new card
card_generator new_card(.hit(hit_sp), .clk(clk), .card(card),
.prev_card(prev_card));
lfsr d_new_card(.clk(clk), .rst(reset), .out(d_card));

// generates a single pulse if hit, stay, double_down or split button is
pressed

edge_detector hit_button(.clk(clk), .in(hit_out), .out(hit_sp));
edge_detector stay_button(.clk(clk), .in(stay_out), .out(stay_sp));
edge_detector double_down_button(.clk(clk), .in(dd_out),
.out(double_down_sp));
```

```
edge_detector split_button(.clk(clk), .in(split_out), .out(split_sp));


// FSM state register
reg [3:0] state;

// Score and card number FF
reg [4:0] player_score;
reg [4:0] dealer_score;
reg [4:0] number_of_cards_player;
reg [4:0] number_of_cards_dealer;
reg [4:0] split_fh_p_score;
reg [4:0] split_sh_p_score;
reg [4:0] split_fh_d_score;
reg [4:0] split_sh_d_score;

reg [7:0] bet; // Players bet

reg dd_flag;        // Limits player to draw only one card in double down
state
reg split_flag;     // To make sure player hits at least once before staying
reg result_flag;
reg splitted;
reg hit_d;

// States
localparam state_initial = 0;
localparam state_player = 1;
localparam state_dealer = 2;
localparam state_game_result = 3;
localparam state_player_dd = 4;
localparam state_player_split_fh = 5;
localparam state_player_split_sh = 6;
localparam state_dealer_split_fh = 7;
localparam state_dealer_split_sh = 8;

// For BCD decoder
wire [15:0] p_score_bcd, d_score_bcd, p_fh_bcd, p_sh_bcd, d_fh_bcd, d_sh_bcd;
wire [15:0] bet_bcd;
bin2bcd_5b bcd_p_score(.bin(player_score), .bcd(p_score_bcd));
bin2bcd_5b bcd_d_score(.bin(dealer_score), .bcd(d_score_bcd));
bin2bcd_5b bcd_p_score_fh(.bin(split_fh_p_score), .bcd(p_fh_bcd));
bin2bcd_5b bcd_p_score_sh(.bin(split_sh_p_score), .bcd(p_sh_bcd));
bin2bcd_5b bcd_d_score_fh(.bin(split_fh_d_score), .bcd(d_fh_bcd));
bin2bcd_5b bcd_d_score_sh(.bin(split_sh_d_score), .bcd(d_sh_bcd));
bin2bcd_8b bcd_bet(.bin(bet), .bcd(bet_bcd));

wire trigger;
assign trigger=(hit_out || stay_out || dd_out || split_out || reset)?1:0;
```

```
// LCD
lcd lcdTest( .clk(clk), .p_score_bcd(p_score_bcd), .d_score_bcd(d_score_bcd),
.p_fh_bcd(p_fh_bcd), .p_sh_bcd(p_sh_bcd), .d_fh_bcd(d_fh_bcd),
.d_sh_bcd(d_sh_bcd),
 .bet_bcd(bet_bcd), .sf_e(sf_e), .e(e), .rs(rs), .rw(rw), .d(d), .c(c), .b(b),
.a(a), .trigger(trigger));

always @(posedge clk or posedge reset)
begin
    // Reset to start
    if(reset) begin
    win = 0;
    tie = 0;
    lose = 0;
    number_of_cards_player = 0;
    number_of_cards_dealer = 0;
    player_score = 0;
    dealer_score = 0;
    dd_flag = 1;
    split_flag = 1;
    result_flag = 1;
    splitted = 1;
    hit_d=1;
    split_fh_p_score = 0;
    split_sh_p_score = 0;
    split_fh_d_score = 0;
    split_sh_d_score = 0;
    bet = 100;
    state <= state_initial;
    end
        // Game logic
    else begin


    case(state)

        state_initial:
            begin
                state <= state_player;
            end

        state_player:  // Player turn to draw cards
        if(bet >= 10)
        begin
                begin

                if(hit_sp == 1) begin
```

```
                    if(card == 11 && player_score >
10)                                 // If the player gets another ace, it
counts as 1 (in his favour)
                        begin
                        player_score = player_score + 1'b1;
                        number_of_cards_player = number_of_cards_player +
1'b1;
                        end
                else
                    begin
          // Add new card to score
                    player_score = player_score + card;
                    number_of_cards_player = number_of_cards_player + 1'b1;
                    end
                end


                if((stay_sp == 1 && number_of_cards_player >=2 ) ||
player_score >= 21) // If the player decides to stay with min. 2 cards or the
score is over 21
                    begin
          // it's the dealers turn
                        state <= state_dealer;
                    end
                // DOUBLE DOWN
                if(double_down_sp == 1 && number_of_cards_player==2 &&
(player_score==9 || player_score==10 || player_score==11 ) && bet >= 20)
                    begin
                        state <=
state_player_dd;                                        // If players score
equals 9, 10 or 11 after drawing 2 cards, he's allowed
                    end
          // to double down and recieve only one card in advance
                // SPLIT
                if(split_sp == 1 && number_of_cards_player == 2 &&
card==prev_card)     // If the first two cards are the same value, player's
alowed to split
                    begin
          // and treat the splitted cards as seperate hands
                        split_fh_p_score = card;
                        split_sh_p_score = card;
                        splitted = 0;
                        state <= state_player_split_fh;
                    end



                end
            end
```

```
        state_player_dd:   // DOUBLE DOWN
            begin
                if(hit_sp == 1 && dd_flag == 1)
begin
                    if(card == 11 && player_score >
10)                                 // If the player gets another ace, it
counts as 1 (in his favour)
                        begin
                        player_score = player_score + 1'b1;
                        number_of_cards_player = number_of_cards_player +
1'b1;
                        dd_flag =
0;                                                  // Set dd_flag to 0 to
disallow player to draw more than one card
                        end
                else
                    begin
          // Add new card to score
                    player_score = player_score + card;
                    number_of_cards_player = number_of_cards_player + 1'b1;
                    dd_flag =
0;                                                      // Set dd_flag to 0
to disallow player to draw more than one card
                    end
                end

                if(stay_sp == 1 && dd_flag == 0
)                                       // Player must stay after drawing
one card
                    begin

                        state <= state_dealer;
                    end
            end

        state_player_split_fh:   // Split - players first hand
            begin
            if(hit_sp == 1) begin
                    if(card == 11 && split_fh_p_score >
10)
                        begin
                        split_fh_p_score = split_fh_p_score + 1'b1;
                        //number_of_cards_player = number_of_cards_player +
1'b1;
                        split_flag = 0;
                        end
                else
                    begin
```

```
                    split_fh_p_score = split_fh_p_score + card;
                    //number_of_cards_player = number_of_cards_player + 1'b1;
                    split_flag = 0;
                    end
                end

                if((stay_sp == 1 && split_flag==0) || split_fh_p_score >=
21)
                    begin

                        state <= state_dealer_split_fh;
                        split_flag=1;
                    end
            end

        state_dealer_split_fh:   // Split - dealers first hand
        begin

                if(hit_d == 1) begin
                split_fh_d_score = split_fh_d_score + d_card;
                number_of_cards_dealer = number_of_cards_dealer + 1'b1;
                    if(number_of_cards_dealer >=2 && split_fh_d_score >=
17)                                             // If dealers score is
under 17, dealer must draw another card until his score
                        hit_d=0;
                end

                else     // If dealer stays with min. 2 cards and his score
>=17 move to game result state
                    begin
                        hit_d=1;
                        state <= state_player_split_sh;
                    end
            end

        state_player_split_sh:   // Split - players second hand
            begin
            if(hit_sp == 1) begin
                    if(card == 11 && split_sh_p_score >
10)
                        begin
                        split_sh_p_score = split_sh_p_score + 1'b1;
                        //number_of_cards_player = number_of_cards_player +
1'b1;
                        split_flag = 0;
                        end
                else
                    begin
```

```
                    split_sh_p_score = split_sh_p_score + card;
                    //number_of_cards_player = number_of_cards_player + 1'b1;
                    split_flag = 0;
                    end
                end

                if((stay_sp == 1 && split_flag==0) || split_sh_p_score >=
21)
                    begin

                        state <= state_dealer_split_sh;
                        split_flag = 1;
                    end
            end

            state_dealer_split_sh:   // Split - dealers second hand
            begin

                if(hit_d == 1) begin
                split_sh_d_score = split_sh_d_score + d_card;
                number_of_cards_dealer = number_of_cards_dealer + 1'b1;
                    if(number_of_cards_dealer >=2 && split_sh_d_score >=
17)                                             // If dealers score is
under 17, dealer must draw another card until his score
                        hit_d=0;
                end

                else     // If dealer stays with min. 2 cards and his score
>=17 move to game result state
                    begin
                        hit_d=1;
                        state <= state_game_result;
                    end
            end

        state_dealer:   // Dealer turn to draw cards
            begin

                if(hit_d == 1) begin
                dealer_score = dealer_score + d_card;
                number_of_cards_dealer = number_of_cards_dealer + 1'b1;
                    if(number_of_cards_dealer >=2 && dealer_score >=
17)                                             // If dealers score is
under 17, dealer must draw another card until his score
                        hit_d=0;
                end

                else     // If dealer stays with min. 2 cards and his score
>=17 move to game result state
```

```
                    begin
                        hit_d=1;
                        state <= state_game_result;
                    end
            end

        state_game_result:    // Determines the result of the game (for the
player)

        begin

        if(result_flag) begin

        result_flag = 0;
        if(splitted) begin
                if((player_score > 21) && (dealer_score >
21))                        // If both are over 21, it's a tie
                    tie = 1;
                else if(player_score >
21)  begin                                          // If the player is over
21, he loses
                    lose = 1;
                    bet = bet - 10;
                    end
                else if(dealer_score >
21)  begin                                          // If the dealer is over
21, player wins
                    win = 1;
                        if(dd_flag == 0) begin
                        bet = bet + 20;
                        end
                        else begin
                            if(player_score==21)
                            bet = bet + 15;
                            else bet = bet + 10;
                        end
                    end
                else if (dealer_score > player_score)
begin                                   // If the dealers score is over the
players score, player loses
                    lose = 1;
                        if(dd_flag == 0)
                        bet = bet - 20;
                        else
                        bet = bet - 10;
                    end
                else if (dealer_score < player_score)
begin                                   // If player score is over the dealers
score, he wins
```

```
                    win = 1;
                    if(dd_flag == 0) begin
                        bet = bet + 20;
                    end
                    else begin
                        if(player_score==21)
                        bet = bet + 15;
                        else bet = bet + 10;
                        end
                    end
                else if (dealer_score ==
player_score)                                   // If scores are equal, it's a
tie
                    tie = 1;

            end

            else begin
                // SPLIT - First hand; BET 1:1
                if((split_fh_p_score > 21) && (split_fh_d_score >
21))
                    tie = 1;
                else if(split_fh_p_score > 21)
begin
                    lose = 1;
                    bet = bet - 10;
                    end
                else if(split_fh_d_score > 21)
begin
                    win = 1;
                    bet = bet + 10;
                    end
                else if (split_fh_d_score > split_fh_p_score)
begin
                    lose = 1;
                    bet = bet - 10;
                    end
                else if (split_fh_d_score < split_fh_p_score)
begin
                    win = 1;
                    bet = bet + 10;
                    end
                else if (split_fh_d_score == split_fh_p_score &&
split_fh_p_score != 0 && split_fh_d_score !=
0)
                    tie = 1;

                // SPLIT - Second hand; BET 1:1
```

```
                if((split_sh_p_score > 21) && (split_sh_d_score >
21))
                    tie = 1;
                else if(split_sh_p_score >
21)  begin
                    lose = 1;
                    bet = bet - 10;
                    end
                else if(split_sh_d_score > 21)
begin
                    win = 1;
                    bet = bet + 10;
                    end
                else if (split_sh_d_score > split_sh_p_score)
begin
                    lose = 1;
                    bet = bet - 10;
                    end
                else if (split_sh_d_score < split_sh_p_score)
begin
                    win = 1;
                    bet = bet + 10;
                    end
                else if (split_sh_d_score == split_sh_p_score &&
split_sh_p_score != 0 && split_sh_d_score !=
0)
                    tie = 1;

            end
            end

                // if current bet isn't enough, reset for new game
                // if bet is enough, split to continue
                if (split_sp == 1) begin
                    state <= state_initial;
                    win = 0;
                    tie = 0;
                    lose = 0;
                    number_of_cards_player = 0;
                    number_of_cards_dealer = 0;
                    player_score = 0;
                    dealer_score = 0;
                    dd_flag = 1;
                    split_flag = 1;
                    result_flag = 1;
                    splitted = 1;
                    split_fh_p_score = 0;
                    split_sh_p_score = 0;
                    split_fh_d_score = 0;
```

```
                    split_sh_d_score = 0;
                    end


            end
    endcase
    end
end


endmodule
```

Testni modul:

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
//
// Company:
// Engineer:
//
// Create Date:   14:29:50 05/25/2022
// Design Name:   blackjack
// Module Name:   /home/ise/edge_detector/blackjack_test.v
// Project Name:  edge_detector
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: blackjack
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
//

module blackjack_test;

    // Inputs
    reg reset;
    reg clk;
    reg hit;
    reg stay;
    reg double_down;
```

```
    reg split;
    reg [3:0] card;


    // Outputs
    wire win;
    wire tie;
    wire lose;



    // Instantiate the Unit Under Test (UUT)
    blackjack uut (
        .reset(reset),
        .clk(clk),
        .hit(hit),
        .stay(stay),
        .double_down(double_down),
        .split(split),
        .win(win),
        .tie(tie),
        .lose(lose),
        .card(card)

    );
always begin
#4 clk=~clk;
end

initial begin
clk=0;
reset=0;
hit=0;
stay=0;
double_down=0;
split=0;

#5 reset=1;
#5 reset=0;


// prva partija

#20 hit=1;
card=10;
#20 hit=0;

#20 hit=1;
card=5;
```

```
#20 hit=0;


#16 hit=1;
card=4;
#20 hit=0;

#12 stay=1;
#20 stay=0;


#30 split=1;
#20 split=0;

// druga partija dd
#12 hit=1;
card=5;
#20 hit=0;

#40 hit=1;
card=4;
#20 hit=0;

#12 double_down=1;
#20 double_down=0;

#24 hit=1;
card=4;
#20 hit=0;

#12 hit=1;
card=7;
#20 hit=0;

#12 stay=1;
#20 stay=0;

#50 split=1;
#20 split=0;

// treca partija


#12 hit=1;
card=4;
#20 hit=0;

#12 hit=1;
card=4;
```

```
#20 hit=0;

#12 split=1;
#20 split=0;
// p fh
#12 hit=1;
card=7;
#20 hit=0;

#12 hit=1;
card=5;
#20 hit=0;

#12 stay=1;

#20 stay=0;

// p sh
#12 hit=1;
card=7;
#20 hit=0;

#12 hit=1;
card=10;
#20 hit=0;

#12 stay=1;

#20 stay=0;


#15 split=1;
#23 split=0;



#1500 $stop;

end

endmodule
```

PRILOG B – CARD_GENERATOR

Verilog modul:

```
`timescale 1ns / 1ps

module card_generator(input hit, input clk, output reg [3:0] card, output reg
[3:0] prev_card);

reg [3:0] counter;

initial begin
counter=2;
end

always@(posedge clk) begin
counter<=counter+1'b1;
if(counter==11)
counter<=2;
end

always@(posedge hit) begin
card<=counter;
prev_card<= card;
end


endmodule
```

Testni modul:

```
module card_test;

wire [3:0] card, prev_card;
reg hit, clk;

card_generator uut(.hit(hit), .clk(clk), .card(card), .prev_card(prev_card));

initial begin
clk=0;
hit=0;

#10 hit=1;
#2 hit=0;

#14 hit=1;
#2 hit=0;
```

```
#6 hit=1;
#2 hit=0;

#7 hit=1;
#2 hit=0;

#100 $stop;

end


always begin
#4 clk=~clk;
end



endmodule
```

PRILOG C – LFSR

Verilog modul:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer:
//
// Create Date:    19:21:51 06/10/2022
// Design Name:
// Module Name:    lfsr
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////
module lfsr (out, clk, rst);

  output reg [3:0] out;
  reg [3:0] number;
  input clk, rst;

  wire feedback;

  assign feedback = ~(number[3] ^ number[2]);


always @(posedge clk, posedge rst)
  begin
    if (rst)
      number = 4'b0;
    else
      number = {number[2:0],feedback};
  end

  always @(*) begin
  case(number)
  0: out=5;
```

```
  1: out=11;
  14: out=8;
  13: out=3;
  12: out=10;
  default: out=number;
  endcase
  end
endmodule
```

Testni modul:

```
module lfsr_tb();
reg clk_tb;
reg rst_tb;
wire [3:0] out_tb;

initial
begin
    clk_tb = 0;
    rst_tb = 1;
    #15;

    rst_tb = 0;
    #200;
end

always
begin
    #5;
    clk_tb = ~ clk_tb;
end

lfsr DUT(out_tb,clk_tb,rst_tb);
endmodule
```

PRILOG D – EDGE_DETECTOR

Verilog modul:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer:
//
// Create Date:    14:06:12 05/25/2022
// Design Name:
// Module Name:    edge_detector
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////
module edge_detector(
input clk, input in,
output reg out
    );

    reg [1:0] state;

    always @ (posedge clk) begin
    case (state)
        0: begin
        out=0;
        if(in)
            state<=0;
        else
            state<=1;
        end

        1: begin
        out=0;
        if(in)
            state<=2;
        else
            state<=1;
```

```
        end

        2: begin
        out=1;
        if(in)
            state<=0;
        else
            state<=1;
        end

        default: begin
        state<=0;
        out=0;
        end

    endcase
    end


endmodule
```

Testni modul:

```
`timescale 1ns / 1ps


module edge_detector_test;

    // Inputs
    reg clk;
    reg in;

    // Outputs
    wire out;

    // Instantiate the Unit Under Test (UUT)
    edge_detector uut (
        .clk(clk),
        .in(in),
        .out(out)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        in = 0;

        #4 in=0;
```

```
        #10 in=1;
        #8 in=0;
        #10 in=1;
        #8 in=1;
        #20 in=0;
        #8 in=1;
        #8 in=0;


        // Wait 100 ns for global reset to finish
        #100 $stop;

        // Add stimulus here

    end

    always
    #4 clk=~clk;

endmodule
```

PRILOG E – BIN2BCD_5B I BIN2BCD_8B

Verilog modul:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer:
//
// Create Date:    13:37:03 06/08/2022
// Design Name:
// Module Name:    bcd
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////
module bin2bcd_5b(
   input [4:0] bin,
   output reg [15:0] bcd
   );

integer i;

always @(bin) begin
    bcd=0;
    for (i=0;i<5;i=i+1) begin                    //Iterate once for each bit in
input number
        if (bcd[3:0] >= 5) bcd[3:0] = bcd[3:0] + 3;     //If any BCD digit is
>= 5, add three
    if (bcd[7:4] >= 5) bcd[7:4] = bcd[7:4] + 3;
    if (bcd[11:8] >= 5) bcd[11:8] = bcd[11:8] + 3;
    if (bcd[15:12] >= 5) bcd[15:12] = bcd[15:12] + 3;
    bcd = {bcd[14:0],bin[4-i]};             //Shift one bit, and shift in
proper bit from input
    end
end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer:
//
// Create Date:    13:37:03 06/08/2022
// Design Name:
// Module Name:    bcd
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////
module bin2bcd_8b(
   input [7:0] bin,
   output reg [15:0] bcd
   );

integer i;

always @(bin) begin
    bcd=0;
    for (i=0;i<8;i=i+1) begin                  //Iterate once for each bit in
input number
        if (bcd[3:0] >= 5) bcd[3:0] = bcd[3:0] + 3;     //If any BCD digit is
>= 5, add three
    if (bcd[7:4] >= 5) bcd[7:4] = bcd[7:4] + 3;
    if (bcd[11:8] >= 5) bcd[11:8] = bcd[11:8] + 3;
    if (bcd[15:12] >= 5) bcd[15:12] = bcd[15:12] + 3;
    bcd = {bcd[14:0],bin[7-i]};             //Shift one bit, and shift in
proper bit from input
    end
end
endmodule
```

PRILOG F – DBC

Verilog modul:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer:
//
// Create Date:    12:26:19 06/27/2022
// Design Name:
// Module Name:    dbc
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////
module dbc(
input clk,
input signal_i,
output reg signal_o
    );

    parameter clock_freq = 100000000,
              debounce_time = 1000,
              initial_value = 1'b0;

    localparam timerlim = clock_freq / debounce_time;

    `ifdef SIMULATION
        localparam s_initial = "s_initial",
                   s_zero = "s_zero",
                   s_zero_to_one = "s_zero_to_one",
                   s_one = "s_one",
                   s_one_to_zero = "s_one_to_zero";
        reg [13*8-1 : 0] state = s_initial;

    `else
        localparam s_initial = 3'b000,
```

```
                s_zero = 3'b001,
                s_zero_to_one = 3'b010,
                s_one = 3'b011,
                s_one_to_zero = 3'b100;
    reg [2:0] state = s_initial;
`endif

reg [16:0] timer = 17'b0;
reg timer_en = 1'b0, timer_tick = 1'b0;

always@ (posedge clk) begin

    case (state)

        s_initial: begin
            if (initial_value == 1'b0)
                state <= s_zero;
            else
                state <= s_one;
        end

        s_zero: begin
            signal_o <= 1'b0;

            if (signal_i == 1'b1)
                state <= s_zero_to_one;
        end

        s_zero_to_one: begin
            signal_o <= 1'b0;
            timer_en <= 1'b1;

            if (timer_tick == 1'b1) begin
                state <= s_one;
                timer_en <= 1'b0;
            end

            if (signal_i == 1'b0) begin
                state <= s_zero;
                timer_en <= 1'b0;
            end
        end

        s_one: begin
            signal_o <= 1'b1;

            if (signal_i == 1'b0)
                state <= s_one_to_zero;
        end
```

```
            s_one_to_zero: begin
                signal_o <= 1'b1;
                timer_en <= 1'b1;

                if (timer_tick == 1'b1) begin
                    state <= s_zero;
                    timer_en <= 1'b0;
                end

                if (signal_i == 1'b1) begin
                    state <= s_one;
                    timer_en <= 1'b0;
                end
            end

        endcase

        if (timer_en == 1'b1) begin
            if (timer == (timerlim - 1)) begin
                timer_tick <= 1'b1;
                timer <= 17'b0;
            end
            else begin
                timer_tick <= 1'b0;
                timer <= timer + 17'b1;
            end
        end
        else begin
            timer <= 17'b0;
            timer_tick <= 1'b0;
        end

    end

endmodule
```

PRILOG G – LCD

Verilog modul:

```
`timescale 1ns / 1ps

// Company: Alchemax
// Engineer: Tobey
// Create Date:    21:57:49 03/08/2099
// TestLCD.v test LCD of Spartan e+ board, XCS500E model, 320-pin package
// Additional Comments: https://www.youtube.com/watch?v=lQ6YKQt6Rz4


// dodat jos inpute za brojeve i promjenit ih u ascii dolje u caseu
module lcd( clk, p_score_bcd, d_score_bcd, p_fh_bcd, p_sh_bcd, d_fh_bcd,
d_sh_bcd,
 bet_bcd, sf_e, e, rs, rw, d, c, b, a, trigger );

     input trigger;
     input [15:0] p_score_bcd, d_score_bcd, p_fh_bcd, p_sh_bcd, d_fh_bcd,
d_sh_bcd, bet_bcd;
     input clk; // pin C9 is the 50-MHz on-board clock
     output reg sf_e; // 1 LCD access (0 StrataFlash access)
     output reg e; // enable (1)
     output reg rs; // Register Select (1 data bits for R/W)
     output reg rw; // Read/Write, 1/0
     output reg d; // 4th data bits (to from a nibble)
     output reg c; // 3rd data bits (to from a nibble)
     output reg b; // 2nd data bits (to from a nibble)
     output reg a; // 1st data bits (to from a nibble)


    reg [ 26 : 0 ] count = 0;   // 27-bit count, 0-(128M-1), over 2 secs
    reg [ 5 : 0 ] code;         // 6-bit different signals to give out
    reg refresh;                    // refresh LCD rate @ about 25Hz
    reg st = 0;

    always @ (posedge clk) begin
    case(st)
    0: begin
        count <= count +1;

        // mislim da mogu smanjiti i na 26:18 onda dolje 17 u refresh
        case ( count[ 26 : 20 ] )   // as top 6 bits change
// power-on init can be carried out before this loop to avoid the flickers
            0: code <= 6'h03;           // power-on init sequence
            1: code <= 6'h03;           // this is needed at least once
            2: code <= 6'h03;           // when LCD's powered on
            3: code <= 6'h02;           // it flickers existing char display
```

```
// Table 5-3, Function Set
// send 00 and upper nibble 0010, then 00 and lower nibble 10xx
            4: code <= 6'h02;           // Function Set, upper nibble 0010
            5: code <= 6'h08;           // lower nibble 1000 (10xx)

// Table 5-3, Entry Mode
// send 00 and upper nibble 0000, then 00 and lower nibble 0 1 I/D S
// last 2 bits of lower nibble: I/D bit (Incr 1, Decr 0), S bit (Shift 1, 0
no)
            6: code <= 6'h00;       // see table, upper nibble 0000, then
lower nibble:
            7: code <= 6'h06;           //  0110: Incr, Shift disabled

// Table 5-3, Display On/Off
// send 00 and upper nibble 0000, then 00 and lower nibble 1DCB:
// D: 1, show char represented by code in DDR, 0 don't, but code remains
// C: 1, show cursor, 0 don't
// B: 1, cursor blinks (if shown), 0 don't blink (if shown)
            8: code <= 6'h00;           // Display On/Off, upper nibble 0000
            9: code <= 6'h0C;           // lower nibble 1100 (1 D C B)

// Table 5-3 Clear Display, 00 and upper nibble 0000, 00 and lower nibble 0001
            10: code <= 6'h00;      // Clear Display, 00 and upper nibble 0000
            11: code <= 6'h01;      // then 00 and lower nibble 0001

// Characters are then given out, the cursor will advance to the right
// Table 5-3, Write Data to DD RAM (or CG RAM)
// Fig 5-4, 'H,' send 10 and upper nibble 0100, then 10 and lower nibble 1000
            12: code <= 6'h25;      // 'P' high nibble
            13: code <= 6'h20;      // 'P' low nibble
            14: code <= 6'h23;      // :
            15: code <= 6'h2A;
            16: code <= 6'h23;      // first dig
            17: code <= {2'h2,p_score_bcd[7:4]};
            18: code <= 6'h23;      // second dig
            19: code <= {2'h2,p_score_bcd[3:0]};
            20: code <= 6'h22;      // razmak
            21: code <= 6'h20;
            22: code <= 6'h24;      // D
            23: code <= 6'h24;
            24: code <= 6'h23;      // :
            25: code <= 6'h2A;
            26: code <= 6'h23;      // first dig
            27: code <= {2'h2,d_score_bcd[7:4]};
            28: code <= 6'h23;      // second dig
            29: code <= {2'h2,d_score_bcd[3:0]};
            30: code <= 6'h22;      // razmak
            31: code <= 6'h20;
```

```
            32: code <= 6'h24;      // B
            33: code <= 6'h22;
            34: code <= 6'h23;      // :
            35: code <= 6'h2A;
            36: code <= 6'h23;      // first dig
            37: code <= {2'h2,bet_bcd[11:8]};
            38: code <= 6'h23;      // second dig
            39: code <= {2'h2,bet_bcd[7:4]};
            40: code <= 6'h23;      // third dig
            41: code <= {2'h2,bet_bcd[3:0]};

// Table 5-3, Set DD RAM (DDR) Address
// position the cursor onto the start of the 2nd line
// send 00 and upper nibble 1???, ??? is the highest 3 bits of the DDR
// address to move the cursor to, then 00 and lower 4 bits of the addr
// so ??? is 100 and then 0000 for h40
            42: code <= 6'b001100;  // pos cursor to 2nd line upper nibble h40
(...)
            43: code <= 6'b000000;  // lower nibble: h0

// Characters are then given out, the cursor will advance to the right
            44: code <= 6'h25;      // 'P' high nibble
            45: code <= 6'h20;      // 'P' low nibble
            46: code <= 6'h23;      // :
            47: code <= 6'h2A;
            48: code <= 6'h23;      // first dig - prvi broj
            49: code <= {2'h2,p_fh_bcd[7:4]};
            50: code <= 6'h23;      // second dig
            51: code <= {2'h2,p_fh_bcd[3:0]};
            52: code <= 6'h22;      // razmak
            53: code <= 6'h20;
            54: code <= 6'h23;      // first dig - drugi broj
            55: code <= {2'h2,p_sh_bcd[7:4]};
            56: code <= 6'h23;      // second dig
            57: code <= {2'h2,p_sh_bcd[3:0]};
            58: code <= 6'h22;      // razmak
            59: code <= 6'h20;
            60: code <= 6'h24;      // D
            61: code <= 6'h24;
            62: code <= 6'h23;      // :
            63: code <= 6'h2A;
            64: code <= 6'h23;      // first dig - prvi broj
            65: code <= {2'h2,d_fh_bcd[7:4]};
            66: code <= 6'h23;      // second dig
            67: code <= {2'h2,d_fh_bcd[3:0]};
            68: code <= 6'h22;      // razmak
            69: code <= 6'h20;
            70: code <= 6'h23;      // first dig - drugi broj
            71: code <= {2'h2,d_sh_bcd[7:4]};
```

```
            72: code <= 6'h23;      // second dig
            73: code <= {2'h2,d_sh_bcd[3:0]};
// Table 5-3, Read Busy Flag and Address
// send 01 BF (Busy Flag) x x x, then 01xxxx
// idling
            default: code <= 6'h10;

        endcase

// refresh (enable) the LCD when
// (it flips when counted upto 2M, and flips again after another 2M)
            refresh <= count[ 19 ]; // flip rate almost 25 (50Mhz / 2^21-2M)
            sf_e <= 1;
            { e, rs, rw, d, c, b, a } <= { refresh, code };
            if(count[26:20]==127)
            st<=1;
            end

            1: begin
            if(trigger)
            st<=0;
            end
        endcase
    end // always

endmodule
```

PRILOG H – UCF DATOTEKA

```
NET "e" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "rs" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "rw" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "sf_e" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
# The LCD four-bit data interface is shared with the StrataFlash.
NET "a" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "b" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "c" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "d" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;

NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;

NET "stay" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "double_down" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "split" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "hit" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;

NET "reset" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;

NET "win" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "tie" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "lose" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```