Universitatea Politehnica din Bucuresti
Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Proiect Verilog - Joc Snake pe FPGA

Voda Mihai Alexandru

Grupa 421D

Bucuresti 2017

Introducere

Scopul principal al jocului este de a controla sarpele pe ecran cu ajutorul tastelor (↑,←,↓,→) , evitarea coliziunii cu marginea neagra a ecranului, si acumularea de puncte prin culesul merelor de pe ecran.

- Merele se genereaza random pe ecran , iar in momentul coliziunii cu un mar dimensiunea sarpelui creste cu 1;

-In momentul coliziunii cu marginea neagra a ecranului , ecranul se face rosu, iar pentru a incepe un **joc nou** se apasa tasta **SPACE**;

-Sarpele nu moare cand se loveste de propriul corp (optional);

-**Scorul** (contorul merelor culese) se poate observa pe afisajul cu 7 segmente al placutei DE1, scorul maxim este 30;

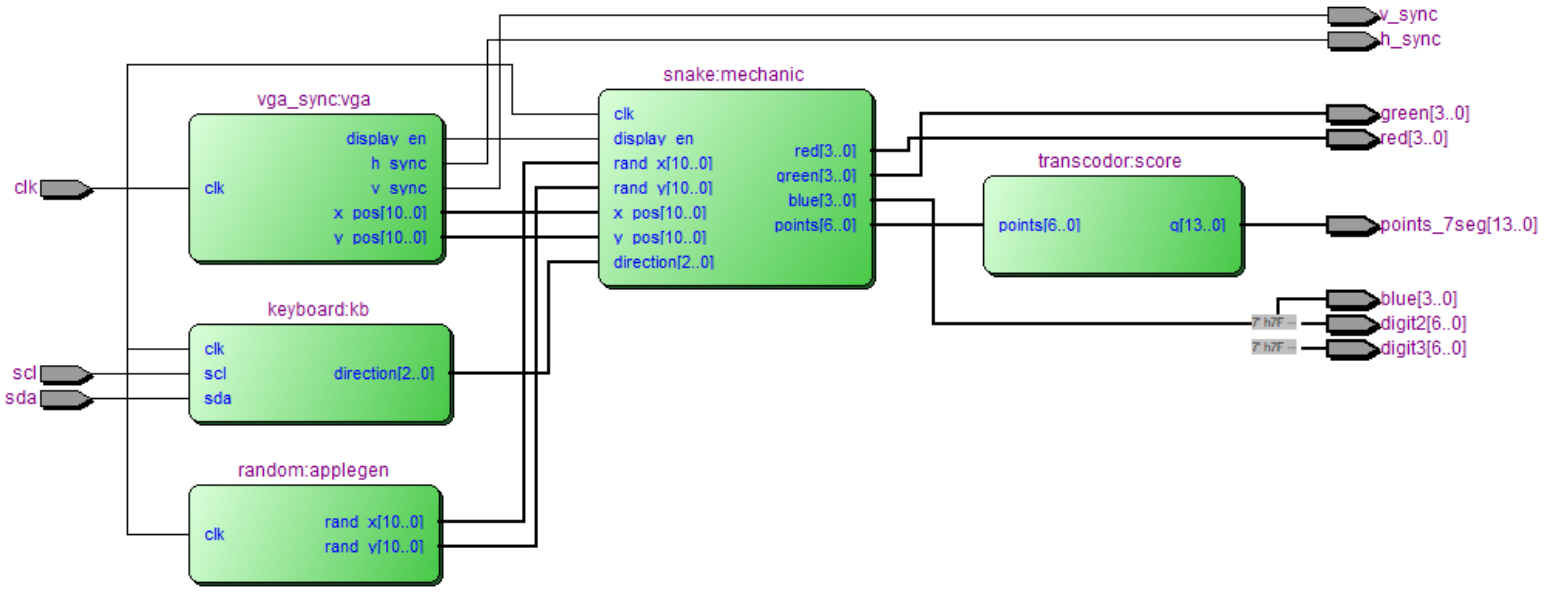
-La atingerea scorului maxim ecranul se face verde, pentru a reseta scorul si jocul se apasa tasta **SPACE**;

-Jocul incepe in momentul in care este apasata una dintre tastele (↑,←,↓,→);

-Jocul a fost realizat in asa fel incat nu este posibila schimbarea sensului doar pe orizontala sau doar pe verticala (ca jocul snake clasic).

Modulul TOP

Schema Interconectarii Blocurilor



Inputs:

- clk (ceas placa DE1 de 50MHz)
- scl (ceas tastatura ~17kHz)
- sda (date primite de la tastatura pe negedge de scl)

Outputs:

- v_sync (sincronizare verticala pentru monitor)
- h_sync (sincronizare orizontala pentru monitor)
- red (4 biti pentru culoarea rosie)
- green (4 biti pentru culoarea verde)
- blue (4 biti pentru culoarea albastra)
- points_7seg (bitii semnificativi pe HEX1, restul pe HEX0)
- digit2 (mereu HIGH stinge segmentul HEX2)
- digit3 (mereu HIGH stinge segmentul HEX3)

Module:

- vga_sync (face sincronizarea,transmite zona activa, pozitiile x si y)
- keyboard(citeste datele de la tastatura si le transmite sub forma de stari)
- random(genereaza aleator doua pozitii x si y)
- snake(contine mecanismul jocului)
- transcodor(afisarea scorului pe afisajul cu 7 segmente)

Modulul Snake

Acesta contine logica efectiva a jocului. Pentru memorarea sarpelui am generat 2 memorii ram cu 32 de adrese, snake_x si snake_y corespunzatoare pozitiilor x si y. Pe adresa 0 se va afla mereu capul sarpelui iar pe adresele 1-31 se va afla mereu corpul sarpelui.

Jocul porneste din starea de reset in care capul sarpelui (snake_x[0] si snake_y[0]) se genereaza pe mijlocul ecranului. **Acesta se actualizeaza la o frecventa redusa de cea a ceasului de 50Mhz al VGA-ului pentru a putea observa cu ochiul liber schimbarile pe ecran.** Cat timp resetul este activ (nu a fost apasata nicio sageata de la tastatura) se initalizeaza cu ajutorul unei structuri repetitive adresele 1-31 pentru corpul sarpelui in afara rezolutiei monitorului (este important sa reinitializez mereu toate adresele in afara rezolutiei ori de cate ori a fost apasat resetul ca sa nu ramana pe display). Odata apasata o sageata de la tastatura jocul iese din starea de reset. Fiecare sageata corespunde unei alte stari **↑-starea 'w' , ←-starea' a', ↓-starea 's', → -starea 'd'.**

```
case(direction)
   w: snake_y[0] <= (snake_y[0] - 11'd20);//w
   a:snake_x[0] <= (snake_x[0] - 11'd20);//a
   s:snake_y[0] <= (snake_y[0] + 11'd20);//s
   d: snake_x[0] <= (snake_x[0] + 11'd20);//d
```

In functie de stare capul sarpelui isi schimba pozitia cu 20 pixeli (pe x sau pe y). Cat timp resetul nu este activ cu ajutorul unei structuri repetitive snake_x[0] devine snake_x[1], snake_x[1] devine snake_x[2] si tot asa pana la size, care se incrementeaza la fiecare mar mancat. (la fel si pentru y). Aceasta procedura este importanta pentru a crea iluzia de miscare.

Obs! Am folosit o structura repetitiva (for) deoarece toate interschimbarile dintre adresele pozitiilor trebuie realizate in paralel in momentul in care s-a activat ceasul redus (update).

La fiecare posedge al ceasului de pe VGA este actualizata marginea de pe display, marul, capul de sarpe si fundalul. Pentru a actualiza corpul sarpelui am folosit o structura repetitiva mai speciala in functie de valoarea registrului size. Folosesc un registru check pentru a verifica ce segment al corpului trebuie colorat. Toate verificarile se fac in paralel pe acelasi posedge.

```
always @(posedge clk)begin
   check = 0;
   for(j = 1; j < size; j = j + 1)
      begin
         if(~check)
         begin
         snake_body = ((x_pos > snake_x[j]-2 && x_pos < snake_x[j]+20-1) && (y_pos > snake_y[j]-1 && y_pos < snake_y[j]+20));
         check = snake_body;
         end
      end
```

Si pentru restul de conditii pentru a colora pe display:

```
        apple <= (x_pos > apple_x-2 && x_pos < (apple_x+20-1)) && (y_pos > apple_y-1 && y_pos < (apple_y+20));
        border <= (((x_pos >= 0) && (x_pos <=21) || (x_pos >= 780) && (x_pos <= 799)) ||
((y_pos >= 0) && (y_pos <= 21) || (y_pos >= 580) && (y_pos <= 599)));
        snake_head <= (x_pos > snake_x[0]-2 && x_pos < (snake_x[0]+20-1)) && (y_pos > snake_y[0]-1 && y_pos < (snake_y[0]+20));
```

In momentul coliziunii cu un mar , adica daca se indeplinesc simultan conditiile pentru snake_head si apple, size precum si scorul care urmeaza sa fie trimis la transcodor se incrementate cu 1. Tot pe acelasi principiu in momentul coliziunii capului cu marginea ecranului flagul bad_collision se activeaza , iar in momentul acumularii a 30 de puncte flagul de win se activeaza. Toate aceste conditii au ca rezultat actualizarea culorilor transmise prin VGA catre monitor.

```
assign apple_c = ((apple || bad_collision) && ~win);
assign snake_c = ((snake_head || snake_body) && ~bad_collision && ~win);
assign border_c = (border && ~bad_collision && ~win);
always@(posedge clk)
   begin
      if(display_en) begin
      if(snake_c||apple_c||border_c||win_c) begin
         if(snake_c)begin
            red=4'h6;
            green=4'hC;
            blue=4'h0;
         end
         if(apple_c)begin
            red=4'hC;
            green=4'h0;
            blue=4'h0;
         end
         if(border_c)begin
            red=4'b0000;
            green=4'b0000;
            blue=4'b0000;
         end
         if(win_c)begin
            red=4'h6;
            green=4'hC;
            blue=4'h0;
         end
```

De exemplu cat timp este activ snake_head sau snake_body se va colora in verde.

Modulul Random

Se genereaza pe frontul pozitivc al ceasului de 50Mhz cu ajutorul a doua countere doua pozitii x si y care vor fi folosite in modulul snake pentru generarea marului in mod aleator.

Modulul Transcodor

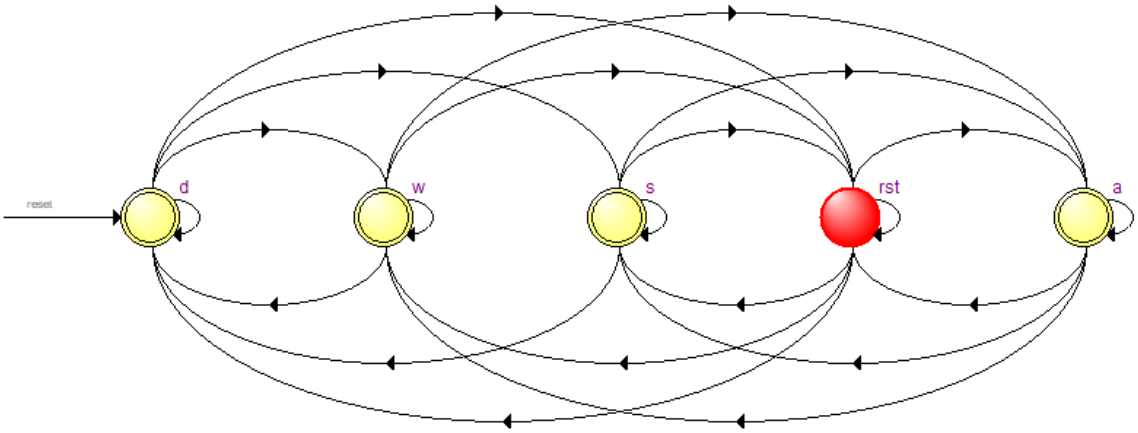
Se afiseaza pe afisajul cu 7 segmente scorul adica numarul de mere culese transmis din modulul snake . Afisajele 3 si 2 sunt mereu HIGH ca sa ramana stinse iar cei mai semnificativi 7 biti sunt folositi pentru afisajul 1 si restul pentru afisajul 0 pentru a genera cu usurinta numerele .

Modulul Keyboard

Acest modul este responsabil cu transmiterea datelor primite de la tastatura catre modulul snake sub forma de stari (w,a,s,d). Datele se valideaza doar atunci cand un pachet

complet de 11 biti a fost primit. Pentru stocarea datelor trimise am folosit un registru de shiftare la dreapta.

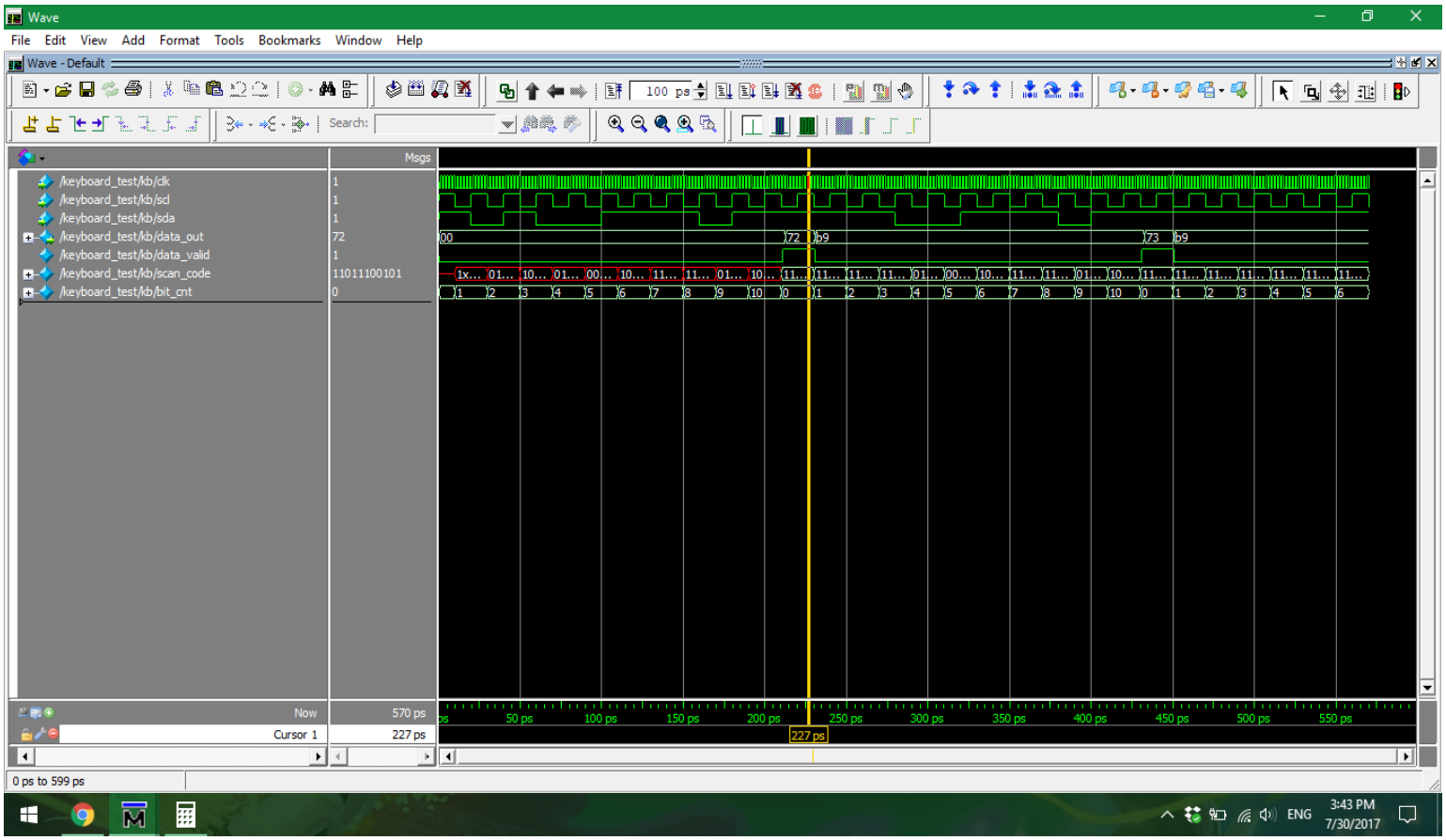
Automat pentru starile trimise catre modulul snake.



De exemplu daca automatul este in starea ‘d’ nu va putea niciodata trece in starea ‘a’ deci nu se va putea schimba sensul sarpelui doar pe orizontala sau doar pe verticala.

Pe simularea facuta in ModelSim se poate observa ca atunci cand data_valid este 1 se transmite data_out=’h 72 respectiv ‘h73 in hexazecimal, ceea ce arata ca modulul de ps2 functioneaza bine fara erori. Nu am luat in calcul faptul ca dupa fiecare tasta apasata urmeaza un break code ‘hF0, modulul de

DUT fiind unul foarte simplu , am vrut sa verific doar functionalitatea in momentul transmiterii de date.

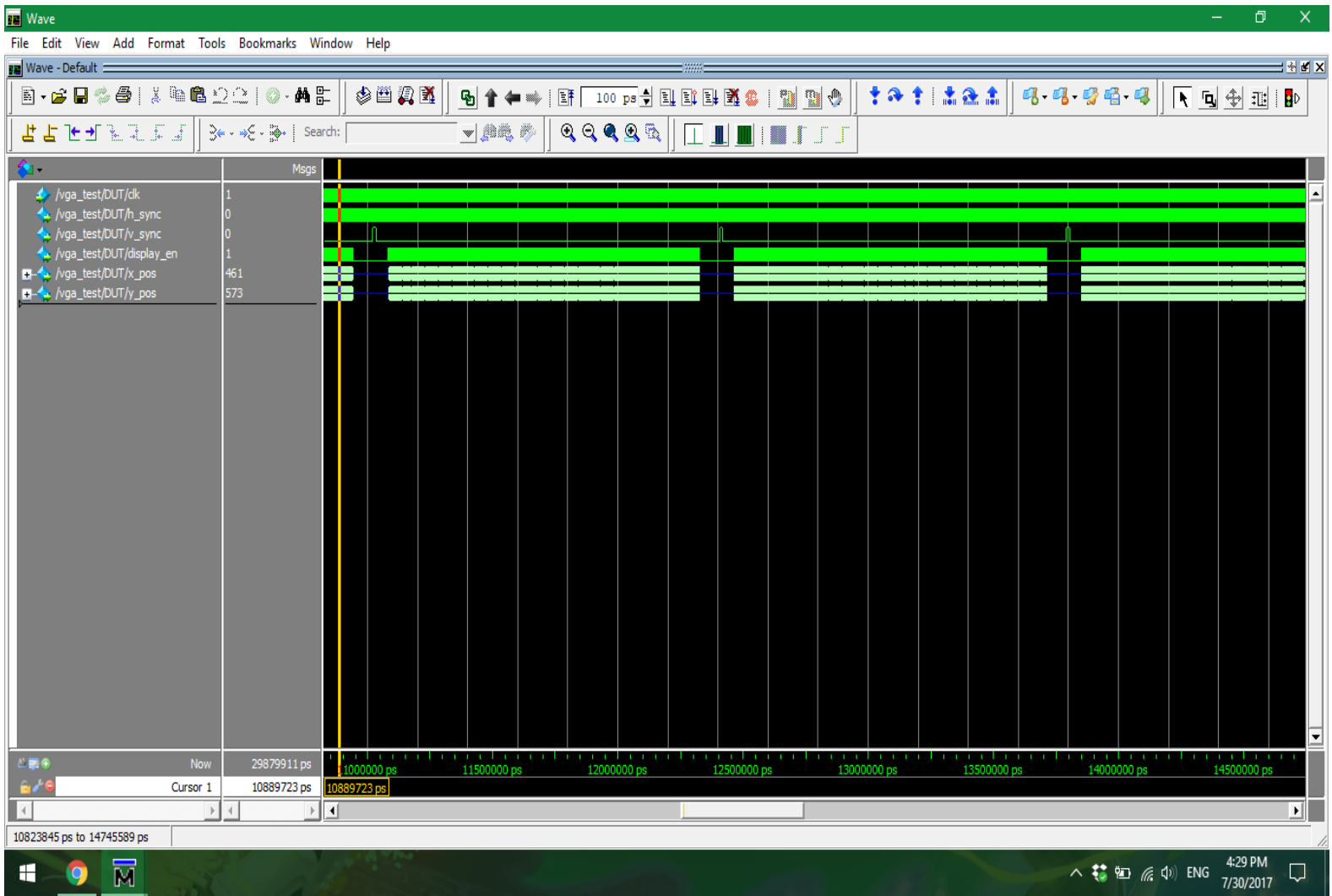


Modulul Keyboard folosit in proiect este unul asemanator, diferenta fiind faptul ca este special doar pentru tastele cu care lucrez (sageti si space) , si pe langa asta am renuntat la partea in care verific paritatea cu data_valid deoarece am observat ca

apare un mic delay in momentul in care schimb directia, care devine deranjant in timpul jocului. Modulul functioneaza foarte bine si fara data_valid, daca apare o eroare automatul intra pe modul default unde se transmite in continuare data de dinainte.

Modulul VGA_sync

Este modulul in care se sincronizeaza datele trimise prin vga cu monitorul, se genereaza pozitiile pixelilor de pe display x_pos si y_pos care sunt folosite mai departe in modulul snake pentru a genera culorile.

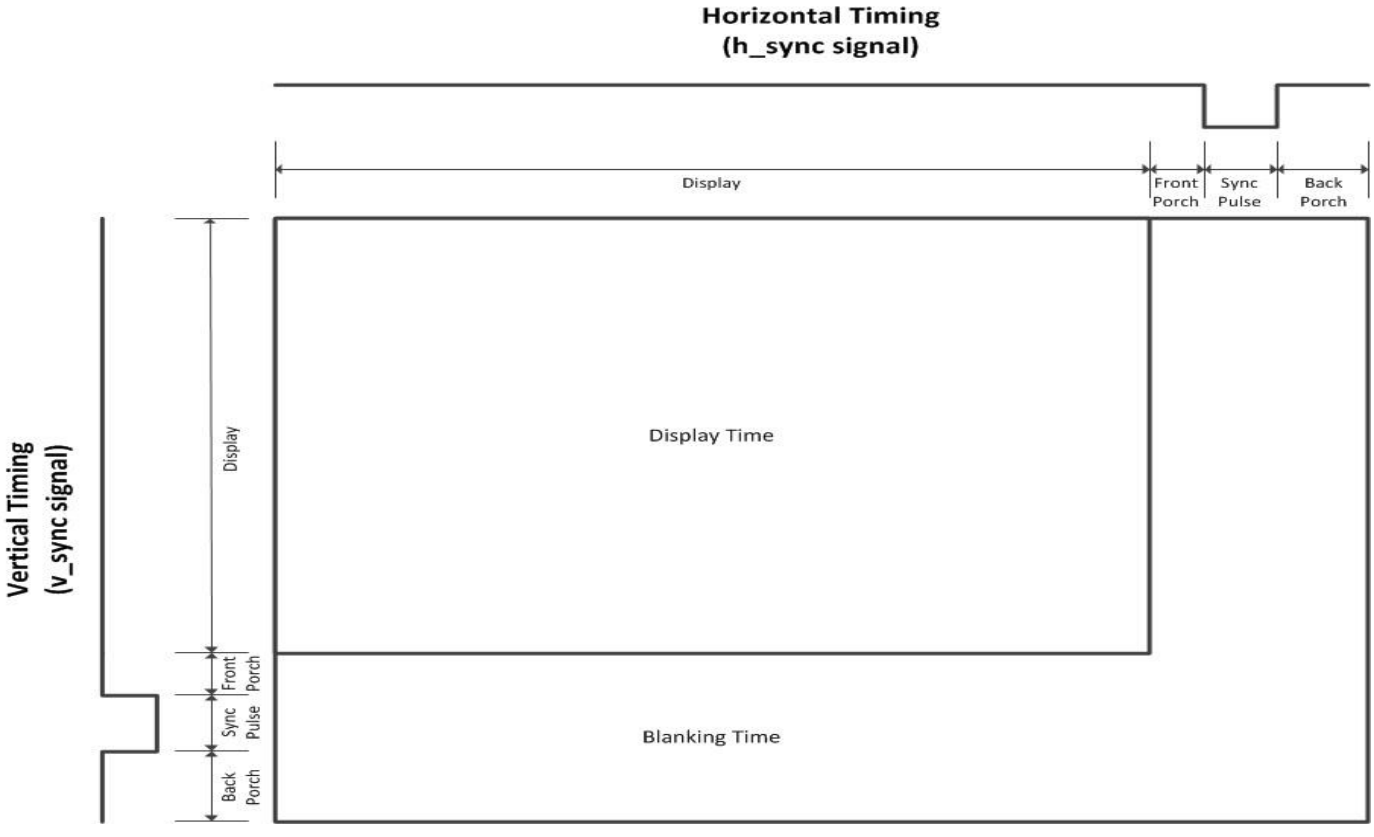


Pe simulare se poate observa ca vertical_sync devine 1 doar dupa ce a trecut de zona activa si mai exact cand este intre front porch si back porch, h_sync nu se poate observa pentru ca se intampla la fiecare incrementare de y_pos (foarte des).

x_pos si y_pos devin HiZ cand counterele pentru h_sync si v_sync ajung in zona inactiva si sunt transmise mai departe modulului principal snake.

Se poate observa si buna functionare a modulului de VGA prin simplul fapt ca imaginea generata pe monitor este centrata corect si nu apar probleme de sincronizare, in caz contrar monitorul ar fi intrat in modul sleep.

Pentru buna functionare am respectat intocmai diagrama pentru h_sync pulse si v_sync pulse.



Observatii si Dificultati

Deoarece toate schimbarile ce tin de conditiile pentru colorat trebuie sa se intample pe acelasi posedge de ceas al VGA-ului, este necesar sa folosesc structuri repetitive pentru a executa toate schimbarile posibile in paralel. O alternativa este sa transform structurile repetitive in structuri cu multe if-uri, dar nu este o schimbare drastica intrucat structura ‘for’ este tot mai multe structuri ‘if’ executate in paralel pentru fiecare caz in parte.

De exemplu pentru cazul size == 5 structura repetitiva transformatata in if-uri arata asa:

```
if(size==5)begin
    if(~check) begin
      snake_body = ((x_pos > snake_x[size-4] && x_pos < snake_x[size-4]+20) && (y_pos > snake_y[size-4] && y_pos < snake_y[size-4]+20));
      check=snake_body;
    end
    if(~check) begin
      snake_body = ((x_pos > snake_x[size-3] && x_pos < snake_x[size-3]+20) && (y_pos > snake_y[size-3] && y_pos < snake_y[size-3]+20));
      check=snake_body;
    end
    if(~check) begin
      snake_body = ((x_pos > snake_x[size-2] && x_pos < snake_x[size-2]+20) && (y_pos > snake_y[size-2] && y_pos < snake_y[size-2]+20));
      check=snake_body;
    end
    if(~check) begin
      snake_body = ((x_pos > snake_x[size-1] && x_pos < snake_x[size-1]+20) && (y_pos > snake_y[size-1] && y_pos < snake_y[size-1]+20));
      check=snake_body;
    end
```

Pentru fiecare caz se mai adauga inca o linie de cod, deci pentru toate cazurile pana la 31 ar fi peste 300 de linii de cod. Am incercat sa folosesc un counter, dar nu merge deoarece acesta va fi incrementat la fiecare posedge, iar prin urmare culorile vor fi aproape inobservabile.

Am folosit o masina cu stari finite doar la modulul de ps2 care trasmite starile mai departe modulului snake. Practic partea in care coordonatele capului isi schimba pozitia este o masina cu stari finite .

Initial am lucrat cu un ceas suplimentar care trimitea un semnal de update, iar la fiecare posedge de update primeam date de la tastatura si schimbam coordonatele capului, am avut dificultati cu aceasta metoda si am renuntat la acel ceas , in schimb am folosit tot pe ceasul de 50 MHz un counter ‘update’ care se incrementeaza pana la o anumita valoare (cat sa poate fi vizibil ochiului) moment in care fac toate schimbarile ce tin de directie. In varianta finala folosesc doar ceasul de 50 Mhz si pentru modulul de VGA deoarece lucrez cu o rezolutie de 800x600 cu refresh rate de 72Hz.

O alta problema ar fi faptul ca un counter pe care il folosesc ca sa contorizez merele mancate ‘apple_count’ se incrementeaza din 3 in 3, si nu am reusit sa imi dau seama de ce se intampla asta, dar ca sa nu imi afecteze logica jocului am adaptat codul astfel incat la multiplii de 3 ai ‘apple_count’ , counterul ‘size’ sa se incrementeze doar odata. Dupa aceasta schimbare jocul functioneaza cum ar trebui in caz contrar la fiecare mar mancat apar 3 segmente de corp in loc de 1, nu ceea ce as fi vrut.