Univerza v Ljubljani

Fakulteta za elektrotehniko

Metod Langus

Poročilo

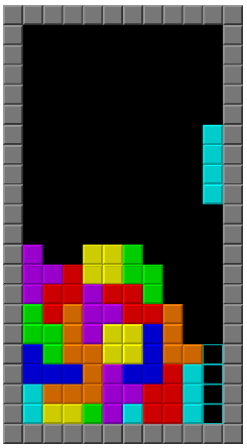
Digitalna integrirana vezja in sistemi

Mentor: izr. prof. dr. Andrej Trost

Ljubljana, januar 2024

Uvod

Za končni projekt sem si zadal cilj izdelati izvedbo dobro znane igre Tetris. Zgodovina igre sega v leto 1985, ko jo je izdelal Alexey Pajitnov, sovjetski programski inženir [1]. Uporabil sem razvojno ploščo MiniZed, ki z razširitveno ploščico Pmod VGA omogoča priključitev monitorja. Za izhodišče sem vzel projekt *vgaboj* [2], ki vsebuje komponente za prikazovanje slike na monitorju VGA. V programu za visokonivojsko sintezo (Vivado HLS 2019.1) sem izdelal funkciji (*VGApixel*, *tetris*) ter preveril delovanje z izdelano testno strukturo. Sintetizirane vhdl datoteke sem nato vključil v projekt (Vivado 2019.1), ter v programu Xilinx SDK 2019.1 [3] napisal še aplikacijo za krmiljenje komponent in s tem oživitev igre Tetris.

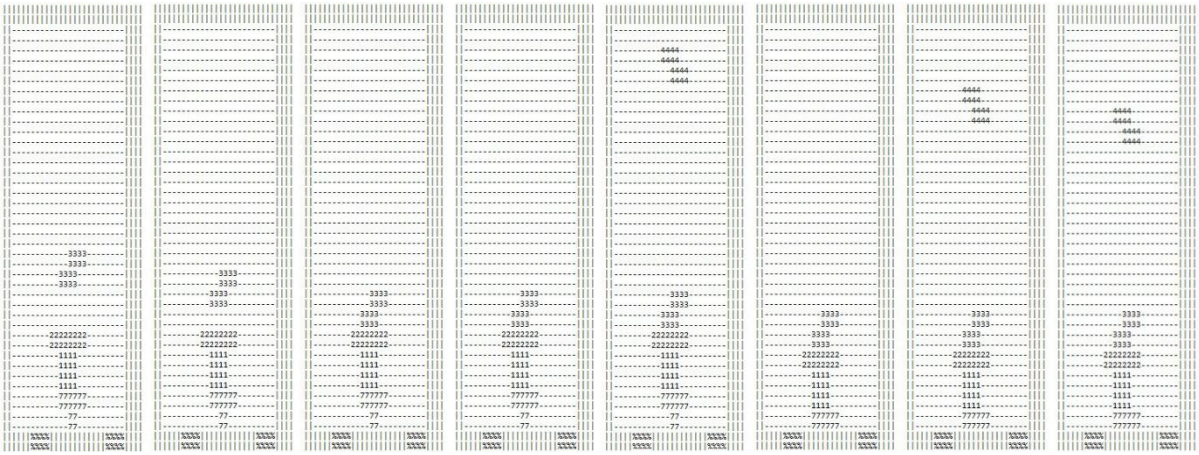


Slika 1. Igra Tetris [1].

Načrtovanje komponente

1. Testna struktura

Razvijanja komponente sem se lotil počasi po korakih. V programu Vivado HLS, sem si za sprotno testiranje kmalu izdelal testno strukturo (*VGApixel_test*), ki posamezne slikovne točke s simboli zapiše v tekstovno datoteko (slika 2) [4]. 'Posnetke' monitorja za lažjo preglednost loči nekaj praznih vrstic. Za lažjo preglednost je možen vklop prikazovanja le igralne površine z minimalno okolico. Možno je tudi nastavljanje skalirnega faktorja celotne igralne površine. Za samo simulacijo je to zelo priročno (na spodnji sliki je skalirni faktor 2), medtem ko je skalirni faktor končne igralne površine nastavljen na 16. Na spodnji sliki je zlepljenih osem zaporednih 'posnetkov' igralne površine, kar predstavlja del dobljene tekstovne datoteke. Na sliki testna struktura simulira prosto padanje in shranjevanje blokov, medtem ko je bil pred šesto sličico vsiljen test izbrisa čisto spodnje vrstice.



Slika 2: Horizontalno zlepljen izsek iz tekstovne datoteke - osem zaporednih 'posnetkov' igralne površine.

2. Funkcija *VGApixel*

Funkcija *VGApixel* je vezni člen med zunanjim svetom ter glavno funkcijo (*tetris*). Skrbi za celotno ozadje, ko pa se slikovna točka nahaja na področju igralne površine, pa za določitev barve slikovne točke kliče funkcijo *tetris*. Skrbi tudi za posredovanje zunanjih signalov funkciji *tetris* v ustreznem trenutku.

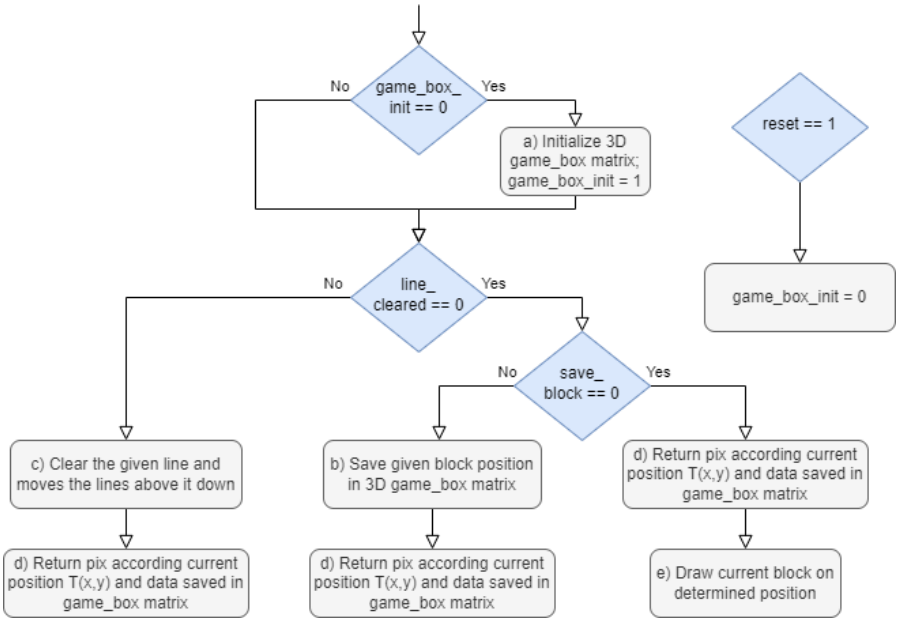
3. Funkcija *tetris*

Vhodi:

- ***block*** – oblika ter ustrezna rotacija bloka
- ***x, y*** – koordinati slikovne točke
- ***x1***, ***y1*** – položaj trenutnega bloka (določata pozicijo prve slikovne točke bloka)
- ***reset*** – izbris oziroma ponovna inicializacija 3D matrike
- **save_block** – ukaz za shranitev bloka v 3D matriko
- ***line_to_clear*** – ukaz za izbris polne vrstice iz 3D

Izhodi:

- ***pix*** – vrednost slikovne točke, ki naj bo prikazana
- ***block_saved*** – povratna informacija, da je bil blok shranjen



Slika 3. Blokovna shema zgradbe funkcije tetris.

V komponenti so shranjene sličice vseh sedmih blokov, vključno z vsemi njihovimi rotacijami [5]. Ustrezno sličico izberemo z vrednostjo signala ***block***. Primer zapisa prvih dveh blokov v pomnilniku je prikazan na sliki 4, zapisani so v minimalni velikosti.

```
// Declare and initialize ROM of tetris blocks
#define ROM_SIZE 76 // 19 blocks * 4 lines
static uint8_t block_rom[ROM_SIZE] = {
    0b0000,// %      %1O
    0b0110,// %  **  %
    0b0110,// %  **  %
    0b0000,// %      %

    0b0000,// %      %2I1
    0b1111,// % **** %
    0b0000,// %      %
    0b0000,// %      %
```

Slika 4: Zapis prvih dveh sličic blokov.

Sledi opis delov kode prikazanih na zgornji blokovni shemi (slika 3). Zaradi lažjega razumevanja je vrstni red drugačen kot na blokovni shemi, ki sledi dejanskemu vrstnemu redu v kodi.

a) Inicializacija 3D matrike

Ob prvem klicu funkcije se inicializira 3D matrika velikosti 20 x 13 osembitnih vrednosti:

```
static T_uint8_t game_box[GAME_BOX_HEIGHT][GAME_BOX_WIDTH + 1];
```

Vsak element matrike na začetku hrani barvo igralne površine, ta je velika 20 x 12 kvadratkov, kamor se nato shranjujejo barve blokov. Dodaten, zadnji stolpec matrike hrani barvo ozadja, saj zaradi same zakasnitve signalov v realnosti pride do prelivanja nekaj zadnjih slikovnih točk vrstic na začetek naslednjih vrstic. Kot že prej omenjeno, je vsa logika igralne površine skalirana s faktorjem 16, da se na monitorju prikaže zadovoljivo velika slika igre.

d) Izrisovanje 3D matrike

Del kode na sliki 5 skrbi za vračanje slikovne točke (***pix***), ki se na podlagi trenutne koordinate (***x***, ***y***) prebere iz 3D matrike, ta del kode se izvede ob vsakem klicu funkcije. Ob tem se upošteva položaj igralne površine na monitorju (BOX_POSITION_X, BOX_POSITION_Y), ter njeno skaliranje – makro BLOCK_PRESCALER.

```
// Returns pix according current position T(x,y) and data saved in game_box matrix (always)
T_uint8_t pix_in_game_box;
if ( ((y - BOX_POSITION_Y) >= 0) & ((y - BOX_POSITION_Y) < GAME_BOX_HEIGHT * BLOCK_PRESCALER) ){
    if ( ((x - BOX_POSITION_X) >= 0) & ((x - BOX_POSITION_X) < (GAME_BOX_WIDTH + 1) * BLOCK_PRESCALER) ){
        pix_in_game_box = game_box[(y - BOX_POSITION_Y) / BLOCK_PRESCALER][(x - BOX_POSITION_X) / BLOCK_PRESCALER];
        pix = pix_in_game_box;
    }
}
```

Slika 5. Koda za izrisovanje igralne površine oz. 3D matrike.

e) Izrisovanje trenutnega bloka

Barva slikovne točke za izrisovanje trenutnega bloka se določi glede na pozicijo slikovne točke (***x***, ***y***), položaja bloka (***x1***, ***y1***) ter vrsto bloka (***block***). Kot prikazuje slika 6, na podlagi vrste bloka in vertikalnih koordinat iz blokovnega pomnilnika izberemo ustrezno vrstico (*data*), iz katere nato iz horizontalnih koordinat preračunamo vrednost slikovne točke (***pix***). Za tem sledi le še izbira barve glede na vrednost signala ***block***.

```
// Draw block starting on T(x1,y1) position
if ( ((y - y1) >= 0) & ((y - y1) < 4 * BLOCK_PRESCALER) ){
    if ( ((x - x1) >= 0) & ((x - x1) < 4 * BLOCK_PRESCALER) ){
        // Access the block_rom data based on block current and position
        data = block_rom[((block << 2) + ((y - y1) / BLOCK_PRESCALER))];
        // Access pix based on current position
        pix = (data >> ((3 - ((x - x1) / BLOCK_PRESCALER)) & 0b11)) & 0b1;
    }
    if (pix == 1){
        if (block == 0) {
            pix = 0b11111100;   //O-shaped block (yellow)
        }
        if(block >= 1){
            pix = 0b00011111;   // I-shaped block (cyan or light blue)
        }
```

Slika 6. Izsek kode za izrisovanje trenutnega bloka.

b) Shranitev bloka v 3D matriko

Blok se shrani v enem preletu igralne površine, princip je zato podoben kot pri izrisovanju trenutnega bloka. Tu namreč, glede na pozicijo slikovne točke (***x***, ***y***), položaja bloka (***x1***, ***y1***) ter vrsto bloka (***block***), barvo slikovne točke (***pix***) shranimo na ustrezno mesto v 3D matriki:

```
game_box[(y - BOX_POSITION_Y) / BLOCK_PRESCALER][(x - BOX_POSITION_X) / BLOCK_PRESCALER] = pix;
```

c) Izbris vrstice 3D matrike

Ko pri igri zapolnimo vrstico, se ta izbriše, sej se vrstice nad njo pomaknejo navzdol. Vrstice so oštevilčene od 1 do 20, pri čemer je zgornja vrstica prva, spodnja pa zadnja. Koda na sliki 7 prikazuje premik elementov matrike, od ustrezne vrstice navzgor, navzdol.

```
// Clears the given line and moves the lines above it down
if(line_to_clear > 0){
    if(clearing_line == 0){
        if (line_to_clear <= GAME_BOX_HEIGHT) {
            for (height_cl = line_to_clear - 1; height_cl > 0; height_cl --) {
                for (width_cl = 0; width_cl < GAME_BOX_WIDTH + 1; width_cl ++) {
                    game_box[height_cl][width_cl] = game_box[height_cl - 1][width_cl];
                }
            }
            clearing_line = 1;
        }
    }
}

// Reset clearing_line flag
if (line_to_clear == 0) {
    clearing_line = 0;
}
```

Slika 7. Koda za izbris vrstice oziroma pomik vrstic nad njo navzdol.

4. Sinteza

Za sintezo sem uporabil nepredznačene podatkovne tipe s fiksno vejico:

```
// Ap_ufixed data types for synthesis
typedef ap_ufixed<12,12> T_uint12_t;
typedef ap_ufixed<10,10> T_uint10_t;
typedef ap_ufixed<8,8> T_uint8_t;
typedef ap_ufixed<5,5> T_uint5_t;
typedef ap_ufixed<4,4> T_uint4_t;
typedef ap_ufixed<2,2> T_uint2_t;
typedef ap_ufixed<1,1> T_uint1_t;
```

Slika 8. Uporabljeni nepredznačeni tipi s fiksno vejico.

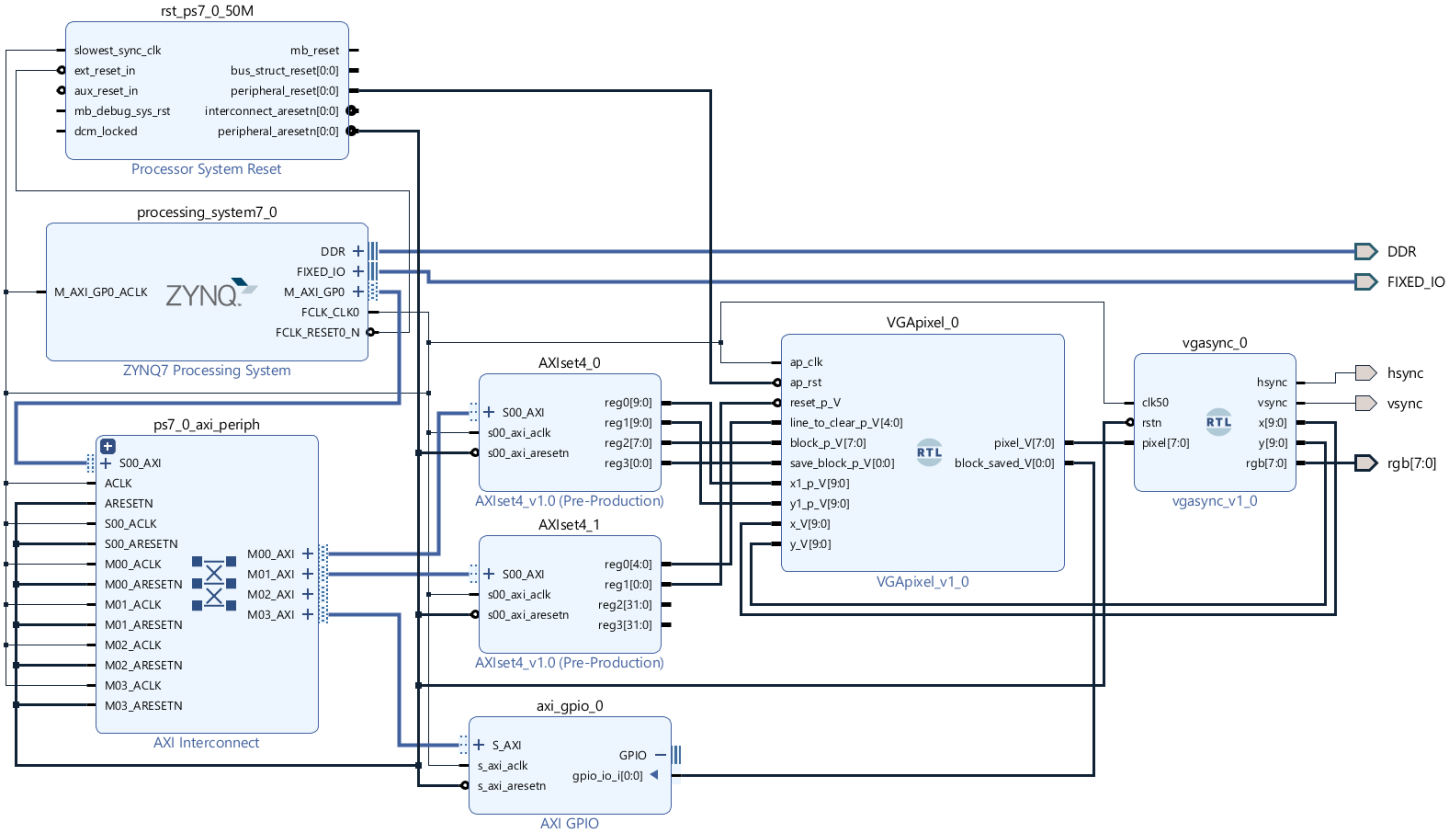
Za razvijanje zank pri inicializaciji 3D matrike sem uporabil direktivo UNROLL, za optimizacijo dostopa do podatkovnih zbirk ARRAY_PARTITION, za odstranitev nepotrebnih krmilnih signalov komponente pa sem uporabil direktivo INTERFACE in izbral način ap_ctrl_none. Kot rezultat sem dobil šest vhdl datotek, ki skupaj predstavljajo mojo komponento *VGApixel*. Rezultat porabe virov komponente (v programu Vivado HLS) prikazuje spodnja tabela 1.

Tabela 1. Poraba resursov razvite komponente VGApixel.

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	100	-
FIFO	-	-	-	-	-
Instance	20	-	327	3717	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	54	-
Register	-	-	84	-	-
Total	20	0	411	3871	0
Available	100	66	28800	14400	0
Utilization (%)	20	0	1	26	0

Blokovna shema celotnega sistema

Komponento *VGApixel* sem vključil v projekt, ter jo povezal kot prikazuje spodnja blokovna shema. Nastavljanje vhodov komponente in branje izhoda poteka preko AXI vodila. Za pravilno delovanje mora biti uporabljena ura s frekvenco 50 MHz, ter vključen UART 1.



Slika 9. Blokovna shema celotnega sistema.

Tabela 2 prikazuje še porabo resursov celotnega projekta (v programu Vivado).

Tabela 2: Poraba resursov celotnega projekta.

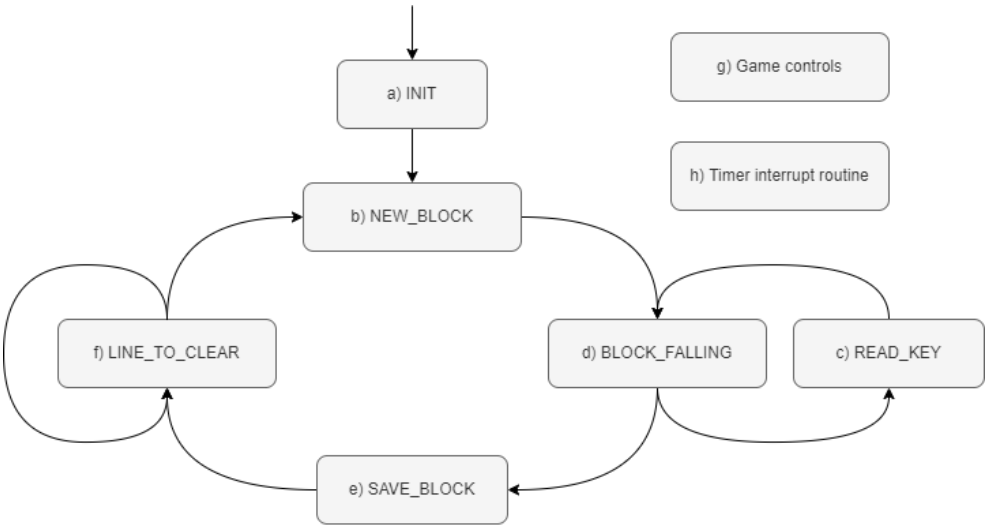
Resource	Utilization	Available	Utilization %
LUT	1148	14400	7.97
LUTRAM	62	6000	1.03
FF	1350	28800	4.69
BRAM	10	50	20.00
IO	10	54	18.52
BUFG	1	32	3.13

Delitev nalog med logiko in programsko opremo

Kot je razvidno že iz opisa načrtovanja komponente sem se odločil, da so oblike blokov ter barve blokov, ozadja in igralne površine shranjene na strojni opremi oziroma v FPGA-ju. To omogoča hitro določitev barve slikovne točke ter s tem seveda hitro delovanje same igre. Potrebna je še programska oprema, ki bo skrbela za pošiljanje kontrolnih signalov preko AXI vodila in s tem oživitev igre Tetris.

Programski del

Čeprav preprosta igra, se izkaže, da je potrebne kaj precej krmilne logike za njeno delovanje, zato sledeč opis ne bo šel v podrobnosti. Tudi tu potrebujemo 3D matriko igralne površine, ki pa je nekoliko večja. Poleg osrednjega dela potrebujemo še stene na levi, desni in spodnji strani, kar nam omogoča zaznavanje robov oziroma tal. Vanjo tekom igre shranjujemo pozicije blokov, katerih barva ni pomembna. Tudi tu potrebujemo sličice vseh blokov. Pred samim začetkom je potreba inicializacija komunikacije UART ter časovnika, ki bo skrbel za prekinitve [6] oziroma postavitev zastavice za avtomatsko pomikanje bloka navzdol. Na spodnji sliki 10 je prikazana struktura programa, sledi pa kratka razlaga stanj.



Slika 10. Avtomat stanj igre Tetris.

a) Inicializacija

V tem stanju se izbriše oziroma ponovno inicializira lokalna 3D matrika, ter 3D matrika na strojni opremi (signal ***reset***). Inicializirajo se še potrebne spremenljivke, naloži vrednost za prekinitev časovnika, ki ustreza hitrosti igre v prvi stopnji, ter v konzolo izpiše pozdravno sporočilo (slika 11).

```
-----------------
Let's play TETRIS
-----------------


-----------------------------------------
Use arrow keys for control
For restart game press an intuitive key (R)
If you need a break, press pause/resume (P)
-----------------------------------------
```

Slika 11. Pozdravno sporočilo v terminalu Putty.

b) Nov blok

S funkcijo rand() se izbere nov blok (signal ***block***), ki se nastavi na začetni položaj (signala ***x1***, ***y1***). V tem stanju se prištejejo tudi točke v primeru uspešne zapolnitve vrstice oziroma vrstic [7].

c) Branje tipke

Če je bila pritisnjena katera od smernih tipk (gor, dol, levo, desno), P ali R, se v tem stanju postavi ustrezno zastavico. Na pritisk tipke ne čakamo, temveč le v vsakem ciklu pogledamo, če je katera pritisnjena. Procesor izvajanja kodo tako hitro, da je bilo branje tipk prepogosto, zato sem tu dodal še čakanje za 5 ms.

d) Padanje bloka

V tem stanju se najprej preveri možnosti premikanja položaja trenutnega bloka (dol, levo, desno in rotacija). V primeru da je bila pri branju tipk postavljena zastavica za premikanje bloka desno in je hkrati to tudi izvedljivo, povečamo horizontalni signal za položaj bloka (signal ***x1***), medtem ko signala ***y1*** in ***block*** ostaneta nespremenjena. V primeru rotacije se tako spremeni le signal ***block***. Če premik navzdol ni več mogoč, je nekaj trenutkov še mogoče premikanje bloka v ostale smeri, nakar gre program v naslednje stanje (Shranjevanje bloka).

e) Shranjevanje bloka

Blok se shrani v lokalno 3D matriko, ter 3D matriko na strojni opremi. Slednje se naredi tako, da se signal ***save_block*** postavi na 1, počaka na odziv (signal ***block_saved***), nato pa postavi signal ***save_block*** nazaj na 0.

f) Izbris vrstice

Najprej se v lokalni 3D matriki preveri, če je kakšna vrstica polna. Če ni, gre program naprej v naslednje stanje, če je, pa se izvede sledeče. V lokalni 3D matriki se vse elemente nad polno vrstico premakne za eno mesto navzdol. Za izbris na strojni opremi pa se pošlje številko vrstice (signal ***line_to_clear***), počaka 30 ms, nastavi signal ***line_to_clear*** nazaj na 0, ter zopet počaka 30 ms. Če je še kakšna vrstica polna se cikel ponovi, hkrati se v tem stanju tudi šteje število izbrisanih vrstic za točkovaje.

g) Upravljanje igre

Predstavlja sklop kode, ki ni vključen v avtomat stanj, skrbi pa za odziv na različne zastavice:

- če je pritisnjena tipka P, se igra ustavi oziroma nadaljuje,
- če je pritisnjena tipka R, se igra resetira,
- če igralcu ni uspelo premagati Tetrisa [8] in je igre konec, se izpišejo zbrane točke in nekaj statistike (slika 12),
- ko je zapolnjenih oziroma izbrisanih nastavljeno število vrstic, se poveča stopnja igre – torej faktor za množenje točk ter hitrost padanja blokov. Igra ima 15 stopenj, pri katerih se hitrost povečuje nekako eksponentno [9], kar je doseženo z zmanjševanjem vrednosti, pri kateri se proži prekinitev časovnika.

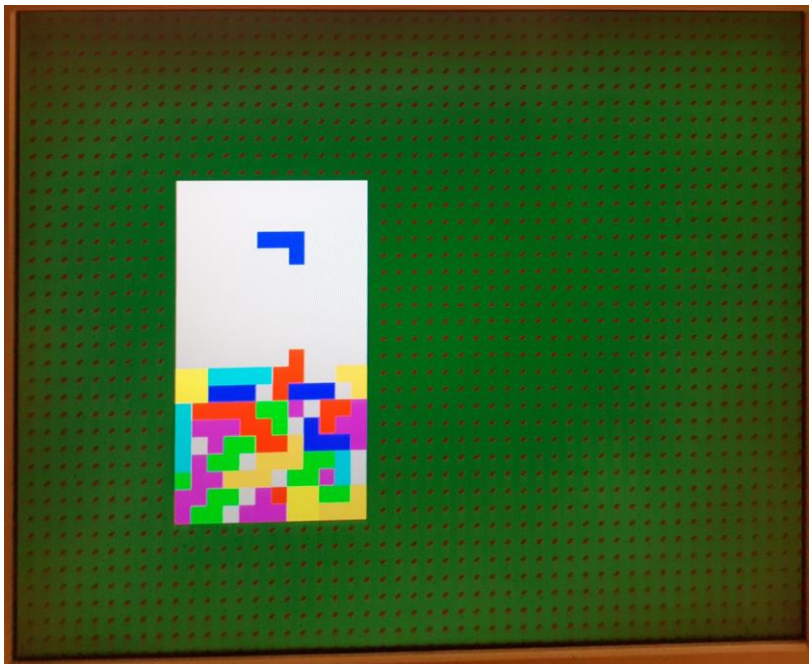
Slika 12. Izgled ob koncu igre v terminalu Putty.

h) Prekinitvena rutina časovnika

Ko števec prišteje do nastavljene vrednosti, se izvajanje glavne zanke prekine in program pride v prekinitveno rutino. Tu se postavi zastavica za avtomatski pomik bloka navzdol, ter resetira časovnik.

Rezultat

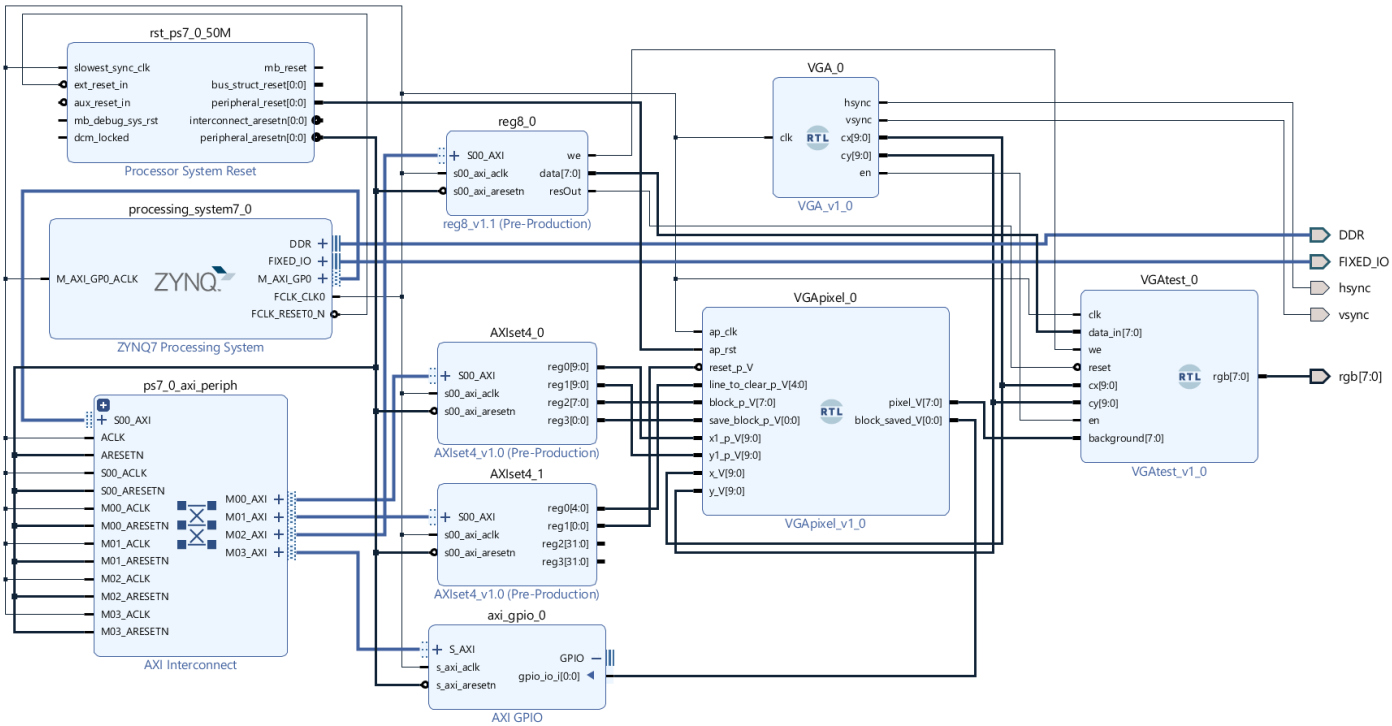
Moj približek igre Tetris prikazuje spodnja slika, pri čemer se vse izpisuje v terminal (slika 12). Padajoči blok premikamo z uporabo smernih tipk, pri čemer s pritiskom tipke 'gor' blok obračamo. Igro lahko tudi ustavimo oziroma nadaljujemo s pritiskom tipke P, če pa pritisnemo tipko R se igra začne znova.



Slika 13. Rezultat razvite komponente na monitorju.

Nadgradnja

V želji po kakovostnejšem izgledu igre, sem v projekt vključil še komponente za znakovni terminal [10]. Izhodni signal moje igre Tetris (***pixel***) sem tako povezal, da skrbi za ozadje znakovnega terminala (vhod ***background***). Blokovno shemo nadgrajenega projekta prikazuje spodnja slika.



Slika 14. Blokovna shema projekta z dodanim znakovnim terminalom.

Tabela 3 prikazuje porabo resursov še nadgrajenega projekta z vključenim znakovnim terminalom.

Tabela 3. Poraba resursov nadgrajenega projekta.

Resource	Utilization	Available	Utilization %
LUT	2593	14400	18.01
LUTRAM	62	6000	1.03
FF	1884	28800	6.54
BRAM	18	50	36.00
DSP	4	66	6.06
IO	10	54	18.52
BUFG	1	32	3.13

Upravljanje znakovnega terminala

Programskemu delu sem tako dodal še kodo za upravljanje z znakovnim terminalom. Upravljanje z njim poteka preko vmesnika (blok *reg8*). Resetiramo ga s pulzom preko signala ***reset***. Preko signala ***data_in*** pa na monitor izrisujemo želene znake, kot tudi nastavljamo barvo ter lokacijo izrisa.

Rezultat – med igro

Z uporabo znakovnega terminala sem dodal barvni naslov igre. Ker obstoječi znakovni terminal ne omogoča skaliranja znakov, sem mu v bazi znakov (komponenta *crom.vhd*) dodal sličico polnega kvadratka, in sicer pod ASCII številko 4. Dodal sem tudi prikaz pozdravnega sporočila, kot je na sliki 11, ter prikazovanje stopnje igre in doseženega števila točk. Poskrbel sem še za majhen dodatek v spodnjem desnem kotu.

Slika 15. Končen izgled moje izvedbe igre Tetris med igranjem.

Rezultat – ko je igre konec

Tudi za konec igre sem se malo poigral z znakovnim terminalom. Pod ASCII številko 5 sem v komponento *crom.vhd* dodal še 'mrežast' znak, s katerim sem nato poskrbel za zatemnitev skoraj celotnega monitorja. Nato se v oknu, s sivim ozadjem, zopet z uporabo kvadratkov, izpiše 'GAME OVER' ter še malo statistike, kot na sliki 12.

Slika 16. Končen izgled moje izvedbe igre Tetris, ko je igre konec.

Zaključek

Z rezultatom sem zelo zadovoljen, tako nad samo funkcionalnostjo, kot je na primer zelo hitra odzivnost na pritisk tipke ali pa izbris vrstic, kot tudi nad vizualnim izgledom. Še posebej nadgradnja z znakovnim terminalom da igri čisto drug, bolj poln oziroma popoln izgled.

Viri in Literatura

[1] ‘*Tetris*’, *Wikipedia*. Jan. 19, 2024. Accessed: Jan. 20, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Tetris&oldid=1197229528

[2] ‘Vivado 2020’. Accessed: Jan. 20, 2024. [Online]. Available: https://lniv.fe.uni-lj.si/xilinx/vivado-vgaobj.htm

[3] ‘Vivado’. Accessed: Jan. 23, 2024. [Online]. Available: https://lniv.fe.uni-lj.si/xilinx/sw.htm

[4] ‘C Write Text File’, Learn C Programming from Scratch. Accessed: Dec. 25, 2023. [Online]. Available: https://www.learnc.net/c-tutorial/c-write-text-file/

[5] S. N. Tural Polat, ‘VHDL İLE TAM GÖMÜLÜ BİR TETRİS OYUNU GERÇEKLEŞTİRMESİ’, *Ömer Halisdemir Üniversitesi Mühendis. Bilim. Derg.*, Jan. 2020, doi: 10.28948/ngumuh.522790.

[6] *IIITD AELD Lab6_P1: Zynq SoC Timers/Counter and Interrupts: Private Timer #zynq #vivado #zedboard*, (Mar. 04, 2022). Accessed: Jan. 22, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=Q5CtmgDgjkc

[7] ‘Scoring - TetrisWiki’. Accessed: Jan. 23, 2024. [Online]. Available: https://tetris.wiki/Scoring

[8] ‘Ameriški najstnik postal prvi človek, ki je premagal Tetris | 24ur.com’. Accessed: Jan. 22, 2024. [Online]. Available: https://www.24ur.com/novice/znanost-in-tehnologija/ameriski-najstnik-postal-prvi-clovek-ki-je-premagal-tetris.html

[9] Dan, ‘Understanding Tetris speed curve’, Game Development Stack Exchange. Accessed: Jan. 23, 2024. [Online]. Available: https://gamedev.stackexchange.com/q/159835

[10] M. Langus, ‘Znakovni terminal za mikroprocesorje v vezju FPGA’, thesis, Univerza v Ljubljani, Fakulteta za elektrotehniko, 2022.