SSC Project

Utlizarea modulului Pmom JSTK

Miklós Balázs

30234

19.10.2019

UTCN

Facultatea de automatica si calculatoare

1.Conținut

2. Introducere

In lumea tehnologiei informatiei placile electronice sunt folosite de multe ori pentru a proiecta anumite idei si sarcini. Digilent este o companie lider de produse de inginerie electrică care deservesc studenți, universități în toată lumea cu instrumente de proiectare educațională bazate pe tehnologie. Digilent proiectează, produce și distribuie produsele sale la nivel mondial. De la anul 2000, produsele Digilent pot fi găsite acum în peste 2000 de universități in peste 70 de țări din întreaga lume. Fiind o companie multinațională cu birouri în SUA, China și România, Digilent este capabil să ofere soluții si calitate pentru o varietate de nevoi ale clienților. Placile de dezvoltare folosite aici sunt de tipul FPGA. (Field-programable gate array). Cu capacitățile lor expansive adecvate în mod unic pentru o gamă largă de aplicații, FPGA-urile sunt ideale pentru a rezolva multe dintre problemele cu care se confruntă sectorul tehnologic în evoluție rapidă.

Materia proiectului este structura sistemelor de calcul. In domeniul aceasta in practica studentii lucreaza in limbajul de descriere hardware VHDL folosind software-ul Vilvaldo. Scopul materiei este ca ei sa aiba cunostinta despre cum anume pot fi proiectate anumite logici sau sau actiuni comportamentale pe placile digitale integrate. Prin folosirea acestor dispozitive electronice, studentii pot sa invete bazele eletronicii, proiectarii si scrierii programelor in VHDL. Prioectele de tipul aceasta ajuta studentilor la intelegerea dezvoltarii sarcinilor pe placile digitale, ca mai tarziu ei sa aiba cunostinte la proiectarea sarcinilor mai mari.

Scopul proiectului ca si cum apare in titlul principal este utilizarea modulului Pmod JTSK. Obiectivele principale ale proiectului sunt urmatoarele:

- Proiectare utilizând limbajul VHDL
- Citirea poziției și a stării butonului prin interfața SPI (Serial Peripheral Interface)
- Aplicație pe calculator pentru afișarea poziției și a stării butonului
- Implementare pe o placă de dezvoltare FPGA

Capabilitatatile placilor de control FPGA pot fi extinse cu ajutorul modulelor utilizate cum ar fi si modululele Pmod. In cazul nostru modulul folosit este cea Pmod JSTK. De fapt este vorba despre conectarea modulului cu o placa de dezvoltare FPGA (in cazul nostru Nexys4 DDR) si folosirea lui. Modulul aceasta este un modul joystick ca si cum arata si numele. Aceasta inseamna

ca are un brat cu care putem misca sau controla un punct digital vazand pozitia acestui punct pe calculator. El mai are un buton de apasare central cu doua butoane suplimentare. De asemenea, Pmod JSTK are două LED-uri programabile pe placă care pot oferi informații aditioale utilizatorului. Deci de fapt, in general solutia consta in conectarea circuitului Nexys4 DDR la un calculator, dupa care conectarea modulului la circuitul si folosirea lui. Ceea ce inseamna ca prin miscarea bratului sau prin apasarea butoanelor o sa apare anumite rezultate pe placa Nexys 4 sau pe calculator, cum ar fi coordonatele Jostick-ului, starea butoanelor etc.

2. Fundamentare teoretica

Conform https://reference.digilentinc.com/reference/programmable-logic/nexys-4/reference-manual manualului de referinta urmatoarele sunt caracteristicile placii Nexys4.

Cum a fost amintit și mai anterior placa de dezvoltare care folosim aici pe lânga modulul joystick este Nexys4 DDR. El are pare din familia Artix 7. Ca și proprietațile plăcii putem aminti ca oferă mai mulți tipuri de conexiuni cum ar fi USB, Ethernet, 4 porturi Pmod, poartă VGA pe 12 biți. Este evidentă ca modulului Pmod JSTK poate fi conectat la placă prin una dintre porturile Pmod. Placa are un clock de 450 MHz, 16MB celular Ram, 4860 Kbits bock Ram, două display-uri de 7 segmente, câte 4 cifre pe display, 16 întrerupătoare, 4 butoane, 16 leduri. Pe lângă accesorii enumerate, placa Nexys 4 mai are o ieșire de audio PWM, un microfon PDM, un sensor de temperatură dar accesoriile acestea n-au atât de mare importanță în proiectul acesta. Deci placa de dezvoltare are multe proprietăți uzuale ceea ce arată că poate fi utilizată și în sine, fără alte componente conectate în plus.

Conform https://reference.digilentinc.com/reference/pmod/pmodjstk/reference-manual urmatoarele sunt caracteristicile modulului Pmod JSTK.

Modulul Pmod JSTK poate fi utilizat într-o mare varietate de proiecte. Cum a fost amintit și mai anterior, conține un joystick rezistiv la axa dublă (x și y), butoane și leduri adiționale. Este ideal folosind împreună cu un circuit FPGA ca și Nexys 4 DDR.

Caracteristici:

- Joystick rezistiv cu 2 axe cu buton central
- Două butoane suplimentare pentru utilizator
- Două LED-uri de utilizator
- Conector Pmod cu 6 pini cu interfață SPI

Pmod JSTK utilizează un microcontroler Atmel ATtiny24. Joystick-ul utilizează două potențiometre pentru a măsura poziția curentă în direcțiile de coordonate x și y și stochează informațiile în două valori de 10 biți cuprinse între 0 și 1023.

Pmod JSTK comunică cu placa gazdă prin intermediul protocolului de comunicare SPI în bucăți de 5 octeți la o viteză maximă de ceas de 1 MHz. Primii patru octeți corespund celor două valori de 10 biți care reprezintă direcțiile de coordonate X și Y, iar ultimul octet pentru a determina starea celor trei butoane apăsate.

Pmod JSTK va trimite totalul său de 23 de biți de informații către placa de sistem prin 40 de cicluri de ceas. Primii doi octeți primiți vor consta în poziția pe 10 biți a potențiometrului în direcția x. Cele 8 biți mai mici din această valoare de 10 biți vor ajunge MSB în primul octet, iar

cei doi MSB rămași din valoarea de 10 biți vor ajunge ca ultimii doi biți din al doilea octet. În mod similar, poziția pe 10 biți a potențiometrului în direcția y, cele 8 biți mai mici din această valoare

de 10 biți vor ajunge MSB în al treilea octet, iar cei doi MSB rămași de 10 biți vor ajunge ca ultimii doi biți în al patrulea octet. Cei trei biți reprezentând cele trei butoane ajung ca ultimii trei biți în al cincilea octet, unde un '1' indică faptul că este apăsat butonul și un '0' indică că butonul nu este apăsat.

Modul în care este organizat cele 5 octeți este prezetat în tabelul următor:

1. Octet	x	x	x	x	x	x	x	x
2. Octet							x	x
3. Octet	y	y	y	y	y	y	y	y
4. Octet							y	y
5. Octet						B1	B2	B3

Axa x primii 8 biți
Axa x ultimii 2 biți cei mai semnificativi
Axa y primii 8 biți
Axa y ultimii 2 biți cei mai semnificativi
Butoanele ultimii 3biți

Conform protocolului SPI, placa de sistem trebuie, de asemenea, să trimită cinci octeți de informații către Pmod. Primul octet va conține informații care să indice dacă cele două LED-uri de bord ar trebui să fie pornite sau dezactivate, cu restul de patru octeți ignorați de Pmod JSTK. Ultimii doi biți ai primului octet trimis indică starea LED2 și respectiv LED1.

Timpul minim recomandat între sfârșitul unui octet care este deplasat și începutul următorului este de 10 µs. Totuși întârzierea recomandată este 15 µs.

Tabelul pinilor de ieşire:

Pin	Semnal	Descriere
1	~CS	Chip select (Active low)
2	MOSI	Master-Out-Slave-In
3	MISO	Master-In-Slave-Out
4	SCK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)

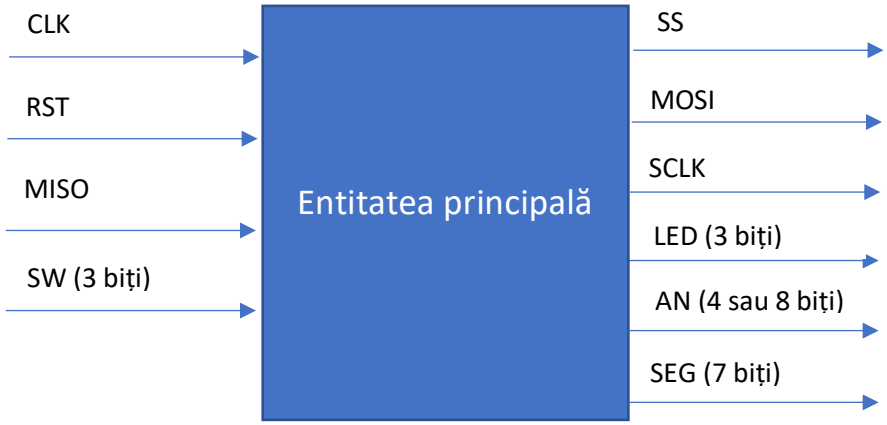
Refeințe:

- https://reference.digilentinc.com/reference/pmod/pmodjstk/reference-manual manualul de referință a modulului Pmod JSTK
- https://reference.digilentinc.com/reference/programmable-logic/nexys-4/reference-manual manualul de referință a plăcii Nexys4 DDR
- https://reference.digilentinc.com/_media/reference/pmod/pmodjstk/pmodjstk_sch.pdf schema

3. Proiectare logică

Componentele principale ale proictului sunt prezentate mai jos:

Schema entității principal:



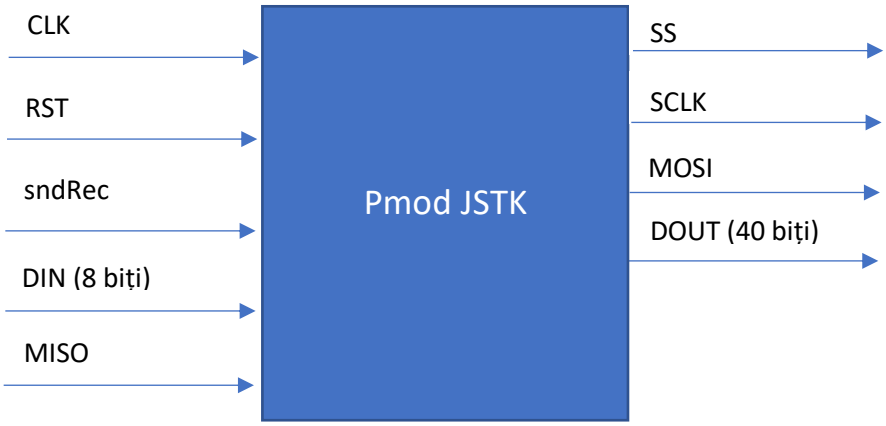
Intrările componentului sunt Clock-ul Reset-ul, Miso (Master In Slave Out) și SW (Switch-urile). Ieșirile sunt SS (Slave Select), MOSI (Master Out Slave In), SCLK (Serial Clock), LED-urile, Anozii și Catozii pentru afșorul SSD.

Entitatea principală are următoarele componente:

A. Modulului Joystick
B. Controlul SSD (Seven segment dispay)
C. Un divizor de fecvență de 5 Hz

Schema subcomponentelor apare mai jos:

A. <u>Modulul Joystick</u>



Intrările componentului sunt Clock-ul, Reset-ul, sndRec (Send Recieve), DIN (Data in), MISO. Ieșirile sunt SS, SCLK, MOSI și DOUT (Data out).

Aici putem observa ieșirea de 5 octeți (40 de biți) a modului care este trimisă către sistemul plăcii Nexys4 DDR.

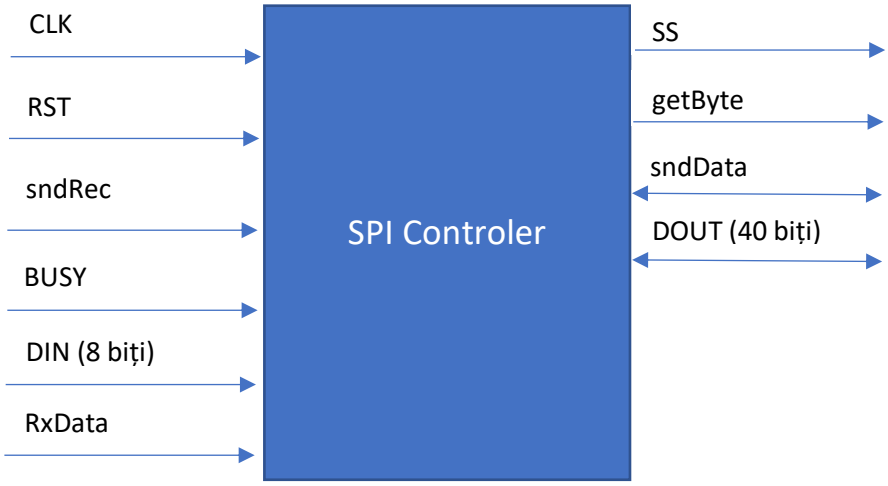
Această componentă este formată din trei subcomponente:

a) *un ceas serial de 66,67kHz*
b) *un controler SPI (Serial Peripheral Interface)*
c) *o interfață SPI.*

a. Divizorul de frecvență de 66.67 kHz

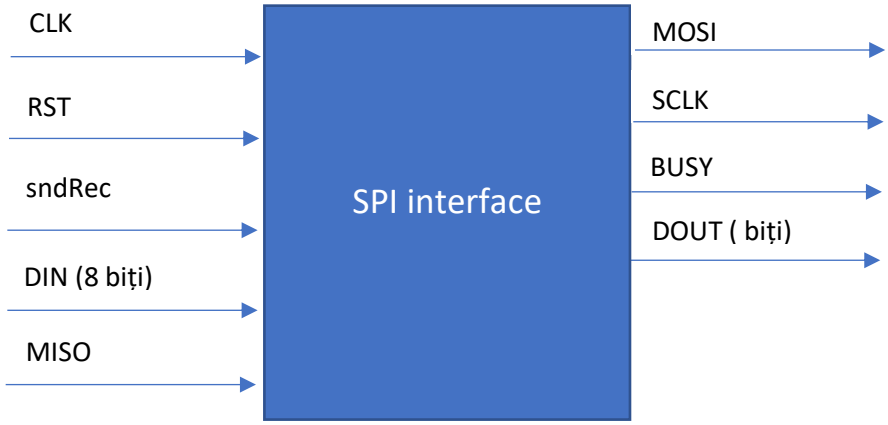
Intrările circuitului sunt un semnal de Clock de 100 Mhz și un semnal de Reset. Componenta are rolul de a diviza frecvența ceasului de intrare la o frecvență de 66.67 kHz.

b. Controlerul SPI



Componeneta are intrările Clock, Reset, sndRec, BUSY, DIN și RxData (Recive). Ieșirile sunt SS și getByte. Porturile sndData (Send Data) și DOUT sunt porturi de tip INOUT. Controlerul SPI gestionează toate cererile de transfer de date și gestionează octeții de date care sunt trimiși către PmodJSTK.

c. Interfața SPI



Intrările sunt Clock, Reset, sndRec, DIN, MISO. Ieșirile sunt MOSI, SCLK, BUSY și DOUT.

Acest modul oferă interfața pentru trimiterea și primirea datelor către și dinspre PmodJSTK, modul SPI este utilizat pentru comunicare. Masterul Nexys4 citește datele de pe intrarea MISO pe front crescător al ceasului, slave-ul (PmodJSTK) citește datele de pe ieșirea MOSI pe front crescător al ceasului. Datele de ieșire la Slave sunt schimbate pe front descrescător, iar datele de intrare de la Slave se schimbă tot pe front descrescător al ceasusui.

Pentru inițializarea unui transfer de date între Master și Slave pur și simplu setăm intrarea sndRec. În timp ce transferul de date este în desfășurare, ieșirea BUSY este setat la 1 logic pentru a indica celorlalți componenți că un transfer de date este în desfășurare. Datele care trebuie trimise Slave-ului sunt introduse pe intrarea DIN, iar datele citite de la Slave sunt transmise la ieșirea DOUT.

Odată ce un semnal sndRec a fost primit, cinci octeți de date vor fi trimise către PmodJSTK și cinci octeți vor fi cititi de la PmodJSTK. Datele trimise provin de la intrarea DIN.

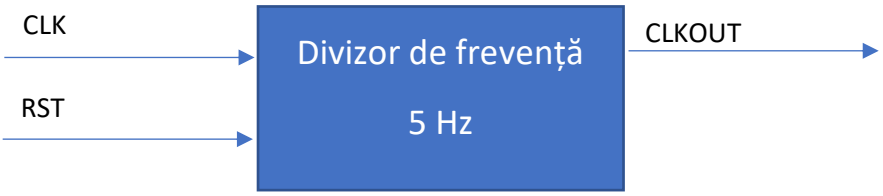
B. Controlul SSD (Seven segment dispay)

Intrările sunt Clock-ul, Reset-ul și intrarea de date (DIN). Ieșirile sunt AN și SEG adică anozii și catozii.

Acest modul interfațează afișajul SSD (șapte segmente) de pe Nexys3 și formatează datele care urmează să fie afișate. Intrarea DIN este un număr binar care devine convertit zecimal și este afișat ca un număr de 4 cifre pe SSD. Busul de ieșire AN parcurgă anozii lui SSD, și controlează care cifră să fie iluminată. Ieșirea SEG pargurgă printre cele 7 catozi și afișează cifra care trebuie afișat.

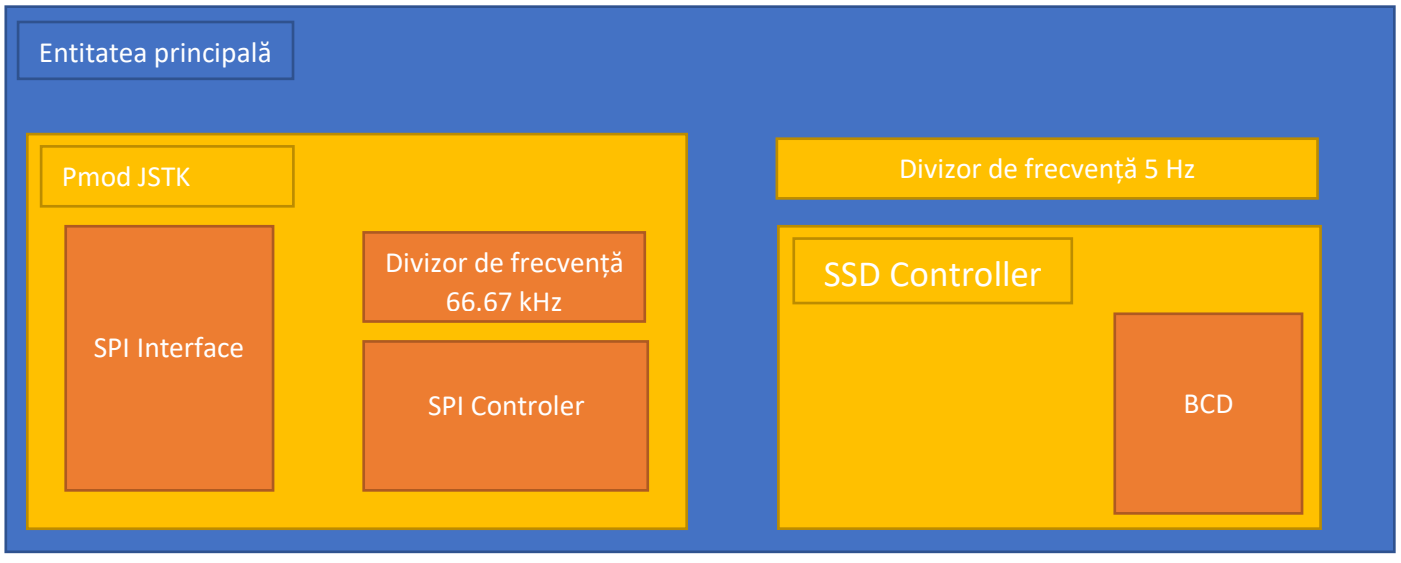
Modulul are subcomponenta BCD cu intrările CLK, RST, Start, BIN și portul BCDOUT care este de tipul INOUT. El convertește numărul pe 10 biți la zecimal pe 4 cifre, având o ieșire pe 16 biți. Biții grupate din 4 în 4 reprezit o cifră din cele 4 cifre ale afișorului.

C. Divizorul de frecvență de 5 Hz

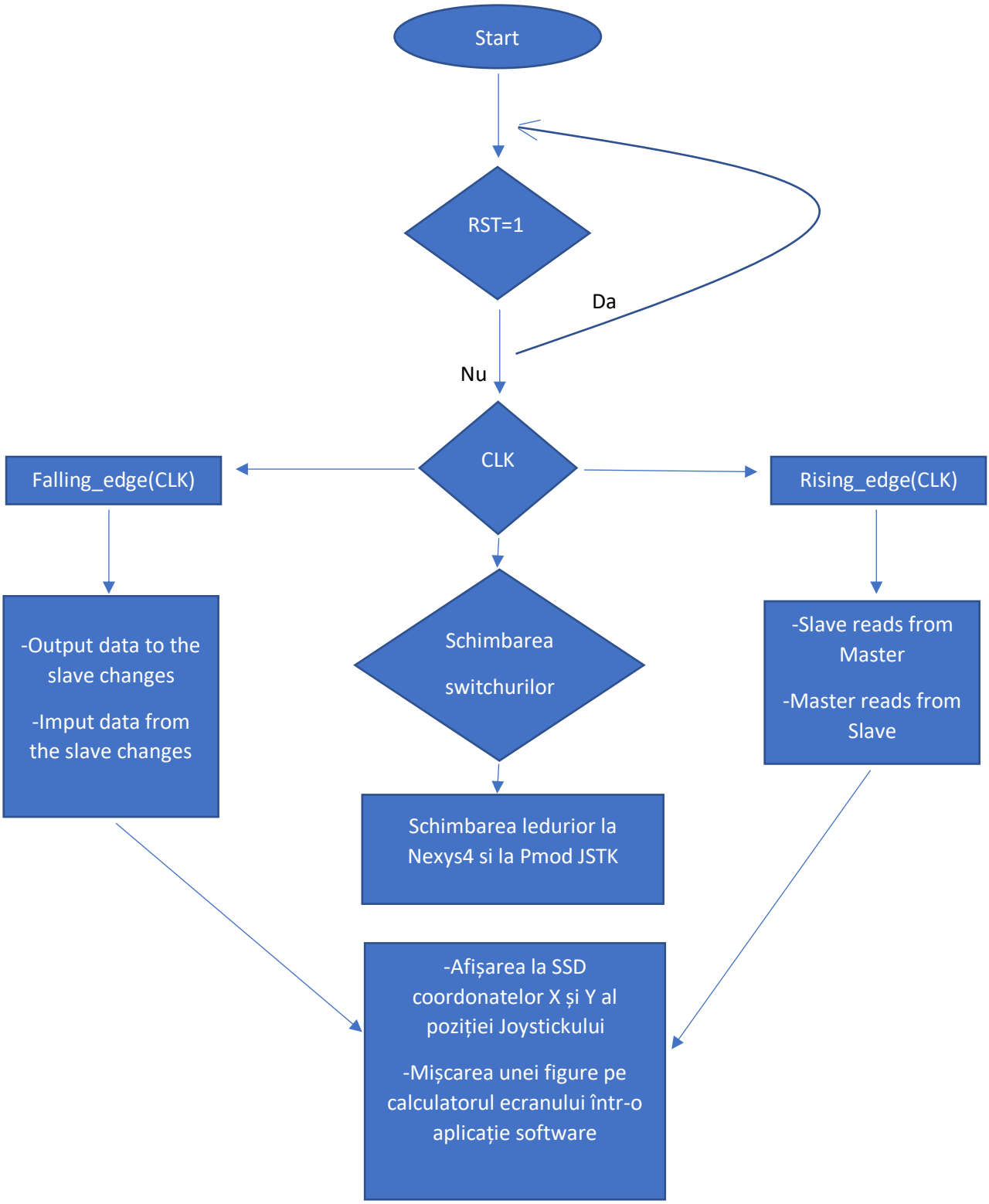


Intrările circuitului sunt un semnal de Clock de 100 Mhz și un semnal de Reset. Componenta are rolul de a diviza frecvența ceasului de intrare la o frecvență de 5 Hz care iese la CLKOUT.

Mai joi apare schema generală a proiectului:



Organigrama operațiilor:



4. Rezultate experimentale

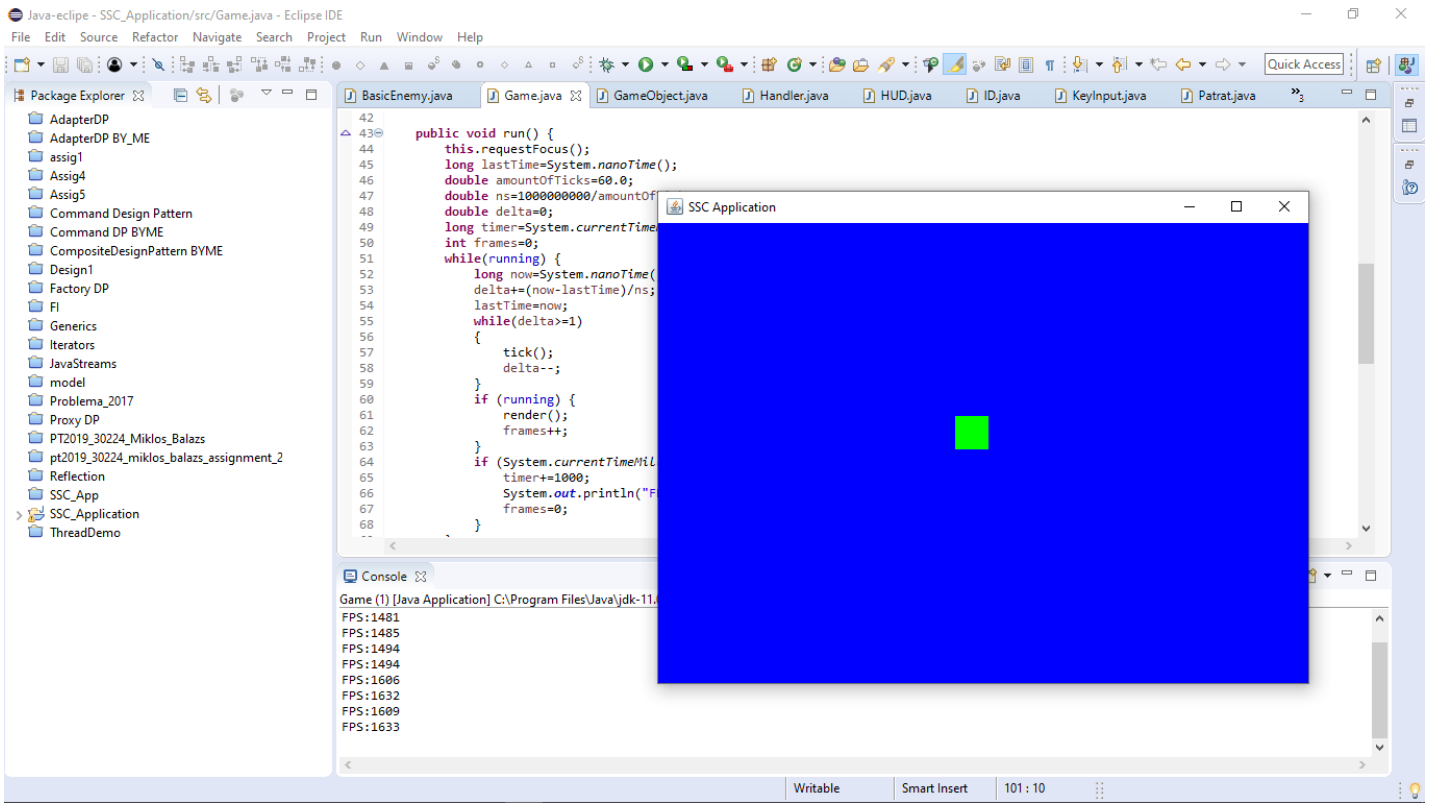
Proiectul este scrisă în libajul VHLD în mediul Vivado 2018.3. Aplicația pentru proiect este scrisă în limbajul Java. Sistemul de operare în calculator este Windows 10.

Placa de FPGA folosită e Nexys 4 DDR. Pe placa acesta este utilizat un port Pmod, 2 switchuri și 3 leduri. Coordonatele joystick-ului sunt apdatate la frecvența de 5 Hz. Mai există și o frecvență serială de 66.67 Hz.

Statultul butoanelor la modulul pmod JSTK este afișat pe 3 leduri pe placa Nexys. Ledul corespunzator unui buton aprinde cand butonul este apăsat. Primele două switch-uri o sa pornească ledurile corespunzătoare pe modulul pmod JSTK. Prin joystick-ului se schimb și coordonatele x și y care sunt afișate pe SSD în același timp. La aplicația tot se mișcă simultan pătratul după coordonatele x și y al joystick- ului.

Rezultatul simulării pe placa este ceva de genu ca și apare pe poza de mai sus. Diferența este doar că la cazul nostru placa folosită este Nexys 4 și apar simultan 8 afișoare de 7 segmente. Pe primele 4 afișoare apare coordonata x iar pe afișoare 5-8 apare coordonata y.

Aplicația arată în felul următor:



Pătratul este un gameObject pe care este afișat într-o fereastră Jwindow. Prin mișcarea joystick-ului se mișcă automat și pătratul verde.

5. Concluzii

Ca și concluzie pot sa afirm ca am învățat mult prin proiectul aceasta. Prima data am înțeles cum funcționează modului joystick cu o placă de dezvoltare FPGA. Pentru aceasta trebuia să înțeleg codul scris în VHDL, cum sunt transmise datele de la Joystck către placa ssmd.

A doua oară am scris și o aplicație în JAVA unde tot am învățat cum trebuie scrise aplicațiile de genu aceasta care are interfață grafică și unde pot să mișc obiecte diferite. (În aplicația am si obiecte de tip Player, obiecte de tip Enemy doar în proiectul aceasta folosesc doar un singur obiect de tip Player.)

Avantajele proiectului este că e ușor de modificat codul ca să funcționează cu orice placă de dezvoltare FPGA. Doar trebuie schimbat fișierul de constrângere și mai niște linii de cod la

afișorul și la switchuri, depinde cum este afișorul plăcii respective. Evident că placa trebuie să aibă port PMOD pentru legarea cu modul.

Ca și dezavantaj, nu pot să enumer nimic în afara de faptul că nu am reușit să leg proiectul cu aplicația 😊.

Proiectul poate fi dezvoltat mai ulterior, fiindcă există nenumerate opțiuni când este vorba despre la ce să folosim un joystick. Putem mișca cu el orice obiecte, roboțele ssmd, sau dacă este cazul aam putea să legăm și alte joystick-uri și să mișcă mai mulți obiecte simultan. Ca și idee sună fain, ca să realizăm e mai greu.

Ca și concluzie, am învățat și am distrat mult cu proiectul.

6. Bibliografie

- https://reference.digilentinc.com/reference/pmod/pmodjstk/reference-manual
- https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start
- https://github.com/MichalLytek/Java-Serial-Terminal

7. Anexa pentru codul sursă

Aici apare codul sursă pentru proiect și pentru aplicație.

Codul VHDL

Pmod JSTK Demo

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PmodJSTK_Demo is
   Port ( CLK : in  STD_LOGIC;
          RST : in  STD_LOGIC;
          MISO : in  STD_LOGIC;
          SW : in  STD_LOGIC_VECTOR (1 downto 0);
          SS : out  STD_LOGIC;
          MOSI : out  STD_LOGIC;
          SCLK : out  STD_LOGIC;
          LED : out  STD_LOGIC_VECTOR (2 downto 0);
          AN : out  STD_LOGIC_VECTOR (7 downto 0);
```

```
        SEG : out  STD_LOGIC_VECTOR (6 downto 0));
end PmodJSTK_Demo;

architecture Behavioral of PmodJSTK_Demo is

                component PmodJSTK

                        Port ( CLK : in  STD_LOGIC;
                                        RST : in  STD_LOGIC;
                                        sndRec : in  STD_LOGIC;
                                        DIN : in  STD_LOGIC_VECTOR (7 downto 0);
                                        MISO : in  STD_LOGIC;
                                        SS : out  STD_LOGIC;
                                        SCLK : out  STD_LOGIC;
                                        MOSI : out  STD_LOGIC;
                                        DOUT : inout  STD_LOGIC_VECTOR (39 downto 0)
                        );

                end component;

                -- ***********************************************
                --                              Display Controller
                -- ***********************************************
                component ssdCtrl

                        Port ( CLK : in  STD_LOGIC;
                                        RST : in  STD_LOGIC;
                                        DIN : in  STD_LOGIC_VECTOR(19 downto 0);
                                        AN  : out STD_LOGIC_VECTOR(7 downto 0);
                                        SEG : out STD_LOGIC_VECTOR(6 downto 0)
                        );

                end component;

                -- ***********************************************
                --                              5Hz Clock Divider
                -- ***********************************************
                component ClkDiv_5Hz

                        Port ( CLK : in  STD_LOGIC;
                                        RST : in  STD_LOGIC;
```

```
                CLKOUT : inout STD_LOGIC
            );

    end component;


        -- Holds data to be sent to PmodJSTK
        signal sndData : STD_LOGIC_VECTOR(7 downto 0) := X"00";

        -- Signal to send/receive data to/from PmodJSTK
        signal sndRec : STD_LOGIC;

        -- Signal indicating that SPI interface is busy
        signal BUSY : STD_LOGIC := '0';

        -- Data read from PmodJSTK
        signal jstkData : STD_LOGIC_VECTOR(39 downto 0) := (others => '0');

        -- Signal carrying output data that user selected
        signal posData : STD_LOGIC_VECTOR(19 downto 0);

begin

        ------------------------------------------------
        --                          PmodJSTK Interface
        ------------------------------------------------
        PmodJSTK_Int : PmodJSTK port map(
                    CLK=>CLK,
                    RST=>RST,
                    sndRec=>sndRec,
                    DIN=>sndData,
                    MISO=>MISO,
                    SS=>SS,
                    SCLK=>SCLK,
                    MOSI=>MOSI,
                    DOUT=>jstkData
        );



        ------------------------------------------------
```

```
		--		Seven Segment Display Controller
		------------------------------------------------
		DispCtrl : ssdCtrl port map(
				CLK=>CLK,
				RST=>RST,
				DIN=>posData,
				AN=>AN,
				SEG=>SEG
		);



		------------------------------------------------
		--		Send Receive Signal Generator
		------------------------------------------------
		genSndRec : ClkDiv_5Hz port map(
				CLK=>CLK,
				RST=>RST,
				CLKOUT=>sndRec
		);



		posData <= (jstkData(9 downto 8) & jstkData(23 downto 16)) &(jstkData(25 downto 24) & jstkData(39 downto 32));

		-- Data to be sent to PmodJSTK, lower two bits will turn on leds on PmodJSTK
		sndData <= "100000" & SW(0) & SW(1);

		-- Assign PmodJSTK button status to LED[2:0]
		process(sndRec, RST) begin
				if(RST = '1') then
						LED <= "000";
				elsif rising_edge(sndRec) then
						LED <= jstkData(1) & jstkData(2) & jstkData(0);
				end if;
		end process;

end Behavioral;
```

Pmod JSTK

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

--
================================================================================
======
--                                                                  Define Module
--
================================================================================
======
entity PmodJSTK is
   Port ( CLK : in  STD_LOGIC;
          RST : in  STD_LOGIC;
          sndRec : in  STD_LOGIC;
          DIN : in  STD_LOGIC_VECTOR (7 downto 0);
          MISO : in  STD_LOGIC;
          SS : out  STD_LOGIC;
          SCLK : out  STD_LOGIC;
          MOSI : out  STD_LOGIC;
          DOUT : inout  STD_LOGIC_VECTOR (39 downto 0));
end PmodJSTK;

architecture Behavioral of PmodJSTK is


                -- ***********************************************
                --                                  SPI Controller
                -- ***********************************************
                component spiCtrl

                        Port ( CLK : in  STD_LOGIC;
                                        RST : in  STD_LOGIC;
                                        sndRec : in STD_LOGIC;
                                        BUSY : in STD_LOGIC;
                                        DIN : in  STD_LOGIC_VECTOR(7 downto 0);
                                        RxData : in  STD_LOGIC_VECTOR(7 downto 0);
                                        SS : out STD_LOGIC;
                                        getByte : out STD_LOGIC;
                                        sndData : inout STD_LOGIC_VECTOR(7 downto 0);
                                        DOUT : inout STD_LOGIC_VECTOR(39 downto 0)
```

```
        );

    end component;

    -- ********************************************
    --                                  SPI Interface
    -- ********************************************
    component spiMode0

        Port ( CLK : in  STD_LOGIC;
                    RST : in  STD_LOGIC;
                    sndRec : in STD_LOGIC;
                    DIN : in  STD_LOGIC_VECTOR(7 downto 0);
                    MISO : in  STD_LOGIC;
                    MOSI : out STD_LOGIC;
                    SCLK : out STD_LOGIC;
                    BUSY : out STD_LOGIC;
                    DOUT : out STD_LOGIC_VECTOR (7 downto 0)
        );

    end component;

    -- ********************************************
    --                              66.67kHz Clock Divider
    -- ********************************************
    component ClkDiv_66_67kHz

        Port ( CLK : in  STD_LOGIC;
                    RST : in  STD_LOGIC;
                    CLKOUT : inout STD_LOGIC
        );

    end component;


    signal getByte : STD_LOGIC;                                                    --
Initiates a data byte transfer in SPI_Int
    signal sndData : STD_LOGIC_VECTOR(7 downto 0);  -- Data to be sent to Slave
    signal RxData : STD_LOGIC_VECTOR(7 downto 0);    -- Output data from SPI_Int
    signal BUSY : STD_LOGIC;
-- Handshake from SPI_Int to SPI_Ctrl
```

```
        -- 66.67kHz Clock Divider, period 15us
        signal iSCLK : STD_LOGIC;
    -- Internal serial clock,

                                -- not directly output to slave,

                                -- controls state machine, etc.

begin

                    ------------------------------------------------
                    --                                      SPI Controller
                    ------------------------------------------------
                    SPI_Ctrl : spiCtrl port map(
                                CLK=>iSCLK,
                                RST=>RST,
                                sndRec=>sndRec,
                                BUSY=>BUSY,
                                DIN=>DIN,
                                RxData=>RxData,
                                SS=>SS,
                                getByte=>getByte,
                                sndData=>sndData,
                                DOUT=>DOUT
                    );

                    ------------------------------------------------
                    --                                      SPI Mode 0
                    ------------------------------------------------
                    SPI_Int : spiMode0 port map(
                                CLK=>iSCLK,
                                RST=>RST,
                                sndRec=>getByte,
                                DIN=>sndData,
                                MISO=>MISO,
                                MOSI=>MOSI,
                                SCLK=>SCLK,
                                BUSY=>BUSY,
                                DOUT=>RxData
```

```
                        );

                        ----------------------------------------------
                        --                                          SPI Controller
                        ----------------------------------------------
                        SerialClock : ClkDiv_66_67kHz port map(
                                        CLK=>CLK,
                                        RST=>RST,
                                        CLKOUT=>iSCLK
                        );

end Behavioral;
```

SPI_Ctrl

```
entity spiCtrl is
   Port ( CLK : in  STD_LOGIC;
          RST : in  STD_LOGIC;
          sndRec : in  STD_LOGIC;
          BUSY : in  STD_LOGIC;
          DIN : in  STD_LOGIC_VECTOR (7 downto 0);
          RxData : in  STD_LOGIC_VECTOR (7 downto 0);
          SS : out  STD_LOGIC;
          getByte : out  STD_LOGIC;
          sndData : inout  STD_LOGIC_VECTOR (7 downto 0);
          DOUT : inout  STD_LOGIC_VECTOR (39 downto 0));
end spiCtrl;

architecture Behavioral of spiCtrl is

                        -- FSM States
                        type state_type is (stIdle, stInit, stWait, stCheck, stDone);

                        -- Present state, Next State
                        signal STATE, NSTATE : state_type;

                        signal byteCnt : STD_LOGIC_VECTOR(2 downto 0) := "000";
                -- Number bits read/written
                        constant byteEndVal : STD_LOGIC_VECTOR(2 downto 0) := "101";
        -- Number of bytes to send/receive
                        signal tmpSR : STD_LOGIC_VECTOR(39 downto 0) := X"0000000000";
                -- Temporary shift register to
```

```
        -- accumulate all five data bytes

begin

                STATE_REGISTER: process(CLK, RST) begin
                        if (RST = '1') then
                                STATE <= stIdle;
                        elsif falling_edge(CLK) then
                                STATE <= NSTATE;
                        end if;
                end process;



                --------------------------------
                --              Output Logic/Assignment
                --------------------------------
                OUTPUT_LOGIC: process (CLK, RST)
                begin
                        if(RST = '1') then
                                -- Reset/clear values
                                SS <= '1';
                                getByte <= '0';
                                sndData <= X"00";
                                tmpSR <= X"0000000000";
                                DOUT <= X"0000000000";
                                byteCnt <= "000";

                        elsif falling_edge(CLK) then

                                case (STATE) is

                                        when stIdle =>

                                                SS <= '1';
                                                -- Disable slave
                                                getByte <= '0';
                                        -- Do not request data
```

```
                    sndData <= X"00";
                -- Clear data to be sent
                    tmpSR <=
X"0000000000";      -- Clear temporary
data

                    DOUT <= DOUT;
                    -- Retain output data
                    byteCnt <= "000";
                -- Clear byte count

            when stInit =>

                    SS <= '0';
                    -- Enable slave
                    getByte <= '1';
            -- Initialize data transfer
                    sndData <= DIN;
                -- Store input data to be sent
                    tmpSR <= tmpSR;
                -- Retain temporary data
                    DOUT <= DOUT;
                    -- Retain output data

                    if(BUSY = '1') then

        byteCnt <= byteCnt + '1';       -- Count

                    end if;

            when stWait =>

                    SS <= '0';
                    -- Enable slave
                    getByte <= '0';

                    sndData <= sndData;

                    tmpSR <= tmpSR;

                    DOUT <= DOUT;
```

```
                                        byteCnt <= byteCnt;
                                -- Count

                                when stCheck =>

                                        SS <= '0';
                                        -- Enable slave
                                        getByte <= '0';

                                        sndData <= sndData;

                                        tmpSR  <=  tmpSR(31
downto 0) & RxData;             -- Store byte just read
                                        DOUT <= DOUT;
                                        -- Retain output data
                                        byteCnt <= byteCnt;
                                -- Do not count

                                when stDone =>

                                        SS <= '1';
                                        -- Disable slave
                                        getByte <= '0';
                                -- Do not request data
                                        sndData <= X"00";
                                -- Clear input
                                        tmpSR <= tmpSR;
                                -- Retain temporary data
                                        DOUT <= tmpSR;
                                        -- Update output data
                                        byteCnt <= byteCnt;
                                -- Do not count

                                end case;

                        end if;
                end process;

                --------------------------------
                --              Next State Logic
                --------------------------------
```

```
NEXT_STATE_LOGIC: process (sndRec, STATE, BUSY, byteCnt)
begin
        -- Define default state to avoid latches
        NSTATE <= stIdle;

        case (STATE) is

                -- Idle
                when stIdle =>

                        -- When send receive signal received
begin data transmission
                        if sndRec = '1' then
                                NSTATE <= stInit;
                        else
                                NSTATE <= stIdle;
                        end if;

                -- Init
                when stInit =>

                        if(BUSY = '1') then
                                NSTATE <= stWait;
                        else
                                NSTATE <= stInit;
                        end if;

                -- Wait
                when stWait =>

                        -- Finished reading byte so grab data
                        if(BUSY = '0') then
                                NSTATE <= stCheck;
                        -- Data transmission is not finished
                        else
                                NSTATE <= stWait;
                        end if;

                -- Check
                when stCheck =>
```

```
                                        -- Finished reading bytes so done
                                        if(byteCnt = "101") then
                                                NSTATE <= stDone;
                                        -- Have not sent/received enough
bytes
                                        else
                                                NSTATE <= stInit;
                                        end if;

                                -- Done
                                when stDone =>

                                        -- Wait for external sndRec signal to
be de-asserted
                                        if(sndRec = '0') then
                                                NSTATE <= stIdle;
                                        else
                                                NSTATE <= stDone;
                                        end if;

                                -- Default
--                              when others =>
--                                      NSTATE <= stIdle;

                        end case;
                end process;

end Behavioral;
```

SPI_Int

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity spiMode0 is
   Port ( CLK : in  STD_LOGIC;                                          --
100Mhz clock
          RST : in  STD_LOGIC;                                          --
Reset
```

```
        sndRec : in  STD_LOGIC;                                                 -- Send
receive, initializes data read/write
        DIN : in  STD_LOGIC_VECTOR (7 downto 0);          -- Data that is to be sent to the slave
        MISO : in  STD_LOGIC;                                                   --
Master input slave output
        MOSI : out  STD_LOGIC;                                                  --
Master out slave in
        SCLK : out  STD_LOGIC;                                                  -- Serial clock
        BUSY : out  STD_LOGIC;                                                  -- Busy
if sending/receiving data
        DOUT : out  STD_LOGIC_VECTOR (7 downto 0));       -- Data read from the slave
end spiMode0;

architecture Behavioral of spiMode0 is

                -- FSM States
                type state_type is (Idle, Init, RxTx, Done);

                -- Present state, Next State
                signal STATE, NSTATE : state_type;

                signal bitCount : STD_LOGIC_VECTOR(3 downto 0) := X"0";
        -- Number bits read/written
                signal rSR : STD_LOGIC_VECTOR(7 downto 0) := X"00";
        -- Read shift register
                signal wSR : STD_LOGIC_VECTOR(7 downto 0) := X"00";
        -- Write shift register

                signal CE : STD_LOGIC := '0';
                                        -- Clock enable, controls serial

                                                                                -- clock  signal
sent to slave

--
================================================================================
=====
--
        Implementation
```

```
--
================================================================================
=====
begin

                        -- Serial clock output, allow if clock enable asserted
                        SCLK <= CLK when (CE = '1') else '0';
                        -- Master out slave in, value always stored in MSB of write shift register
                        MOSI <= wSR(7);
                        -- Connect data output bus to read shift register
                        DOUT <= rSR;

                        ---------------------------------------
                        --                      Write Shift Register
                        --      slave reads on rising edges,
                        -- change output data on falling edges
                        ---------------------------------------
                        process(CLK, RST) begin
                                if(RST = '1') then
                                        wSR <= X"00";
                                elsif falling_edge(CLK) then
                                        -- Enable shift during RxTx state only
                                        case(STATE) is
                                                when Idle =>
                                                        wSR <= DIN;

                                                when Init =>
                                                        wSR <= wSR;

                                                when RxTx =>
                                                        if(CE   =   '1')
then

        wSR <= wSR(6 downto 0) & '0';
                                                        end if;

                                                when Done =>
                                                        wSR <= wSR;
                                        end case;
                                end if;
                        end process;
```

```
                ---------------------------------------
                --                      Read Shift Register
                --          master reads on rising edges,
                -- slave changes data on falling edges
                ---------------------------------------
                process(CLK, RST) begin
                        if(RST = '1') then
                                rSR <= X"00";
                        elsif rising_edge(CLK) then
                                -- Enable shift during RxTx state only
                                case(STATE) is
                                        when Idle =>
                                                rSR <= rSR;

                                        when Init =>
                                                rSR <= rSR;

                                        when RxTx =>
                                                if(CE     =     '1')
then

        rSR <= rSR(6 downto 0) & MISO;
                                                end if;

                                        when Done =>
                                                rSR <= rSR;
                                end case;
                        end if;
                end process;


        -------------------------------
        --                  State Register
        -------------------------------
        STATE_REGISTER: process(CLK, RST) begin
                        if (RST = '1') then
                                STATE <= Idle;
```

```
                elsif falling_edge(CLK) then
                        STATE <= NSTATE;
                end if;
        end process;



        --------------------------------
        --              Output Logic/Assignment
        --------------------------------
        OUTPUT_LOGIC: process (CLK, RST)
        begin
                if(RST = '1') then
                        -- Reset/clear values
                        CE <= '0';
                -- Disable serial clock
                        BUSY <= '0';
            -- Not busy in Idle state
                        bitCount <= X"0";
            -- Clear #bits read/written

                elsif falling_edge(CLK) then

                        case (STATE) is

                                when Idle =>

                                        CE <= '0';
                -- Disable serial clock
                                        BUSY <= '0';
            -- Not busy in Idle state
                                        bitCount <= X"0";
-- Clear #bits read/written

                                when Init =>

                                        BUSY <= '1';
            -- Output a busy signal
                                        bitCount <= X"0";
-- Have not read/written anything yet
```

```
                                                    CE <= '0';
                        -- Disable serial clock
                                                when RxTx =>
                                                    BUSY <= '1';
                    -- Output busy signal
                                                    bitCount  <=  bitCount
+ 1;      -- Begin counting bits received/written
                                                    -- Have written all bits
to slave so prevent another falling edge
                                                    if(bitCount   >=   X"8")
then
                                                        CE    <=
'0';
                                                    -- Have not written all
data, normal operation
                                                    else
                                                        CE    <=
'1';
                                                    end if;
                                                when Done =>
                                                    CE <= '0';
                        -- Disable serial clock
                                                    BUSY <= '1';
                    -- Still busy
                                                    bitCount <= X"0";
                    -- Clear #bits read/written
                                            end case;
                        end if;
            end process;

            --------------------------------
            --                Next State Logic
            --------------------------------
```

```
NEXT_STATE_LOGIC: process (sndRec, bitCount, STATE)
begin
        -- Define default state to avoid latches
        NSTATE <= Idle;

        case (STATE) is

                when Idle =>
                        if sndRec = '1' then
                                NSTATE <= Init;
                        else
                                NSTATE <= Idle;
                        end if;

                when Init =>
                        NSTATE <= RxTx;

                when RxTx =>
                        -- Read last bit so data transmission is
finished
                        if(bitCount = X"8") then
                                NSTATE <= Done;

                        -- Data transmission is not finished
                        else
                                NSTATE <= RxTx;
                        end if;

                when Done =>
                        NSTATE <= Idle;
                when others =>
                        NSTATE <= Idle;
        end case;
end process;

end Behavioral;
```

Serial_CLK

```
entity ClkDiv_66_67kHz is
   Port ( CLK : in  STD_LOGIC;                    -- 100MHz onboard clock
          RST : in  STD_LOGIC;                    -- Reset
          CLKOUT : inout  STD_LOGIC);             -- New clock output
```

```
end ClkDiv_66_67kHz;

architecture Behavioral of ClkDiv_66_67kHz is

--
============================================================================
======
--                                                    Signals and Constants
--
============================================================================
======

        -- Value to toggle output clock at
        constant cntEndVal : STD_LOGIC_VECTOR(9 downto 0) := "1011101110";  --   End   count
value
        -- Current count
        signal clkCount : STD_LOGIC_VECTOR(9 downto 0) := (others => '0');          -- Stores count
value

--
============================================================================
=====
--
        Implementation
--
============================================================================
=====
begin

                        ------------------------------------------------
                        --        5Hz Clock Divider Generates Send/Receive signal
                        ------------------------------------------------
                        process(CLK, RST) begin

                                -- Reset clock
                                if(RST = '1')  then
                                        CLKOUT <= '0';
                                        clkCount <= "0000000000";
                                elsif rising_edge(CLK) then
                                        if(clkCount = cntEndVal) then
                                                CLKOUT <= NOT CLKOUT;
```

```
                                                clkCount <= "0000000000";
                                            else
                                                clkCount <= clkCount + '1';
                                            end if;
                                    end if;

                            end process;

end Behavioral;
```

Display_SSD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


--
================================================================================
=====
--                                                  Define Module, Inputs and Outputs
--
================================================================================
=====
entity ssdCtrl is
   Port ( CLK : in  STD_LOGIC;
          RST : in  STD_LOGIC;
          DIN : in  STD_LOGIC_VECTOR (19 downto 0);
          AN : out  STD_LOGIC_VECTOR (7 downto 0);
          SEG : out  STD_LOGIC_VECTOR (6 downto 0));
end ssdCtrl;

architecture Behavioral of ssdCtrl is

--
================================================================================
=====
--
          Components
```

```
--
================================================================================
=====

		-- ******************************************
		--                                  Binary to BCD Converter
		-- ******************************************
		component Binary_To_BCD

			Port ( CLK : in  STD_LOGIC;
					RST : in  STD_LOGIC;
					START : in STD_LOGIC;
					BIN : in  STD_LOGIC_VECTOR(9 downto 0);
					BCDOUT : inout STD_LOGIC_VECTOR(15 downto 0)
			);

		end component;

--
================================================================================
=====
--                                                                  Signals and Constants
--
================================================================================
=====

			-- 1 kHz Clock Divider
			constant cntEndVal : STD_LOGIC_VECTOR(15 downto 0) := X"C350";
			signal clkCount : STD_LOGIC_VECTOR(15 downto 0) := X"0000";
			signal DCLK : STD_LOGIC := '0';

			-- 2 Bit Counter
			signal CNT : STD_LOGIC_VECTOR(2 downto 0) := "000";

			-- Binary to BCD
			signal bcdDataX : STD_LOGIC_VECTOR(15 downto 0) := X"0000";
			signal bcdDataY : STD_LOGIC_VECTOR(15 downto 0) := X"0000";

			-- Output Data Mux
			signal muxData : STD_LOGIC_VECTOR(3 downto 0);
```

```
--
================================================================================
=====
--
        Implementation
--
================================================================================
=====
begin

                        ------------------------------------------------
                        --                      Convert Binary to BCD
                        ------------------------------------------------
                        BtoBCD_X : Binary_To_BCD port map(
                                        CLK=>CLK,
                                        RST=>RST,
                                        START=>DCLK,
                                        BIN=>DIN(19 downto 10),
                                        BCDOUT=>bcdDataX
                        );
                        BtoBCD_Y : Binary_To_BCD port map(
                                        CLK=>CLK,
                                        RST=>RST,
                                        START=>DCLK,
                                        BIN=>DIN(9 downto 0),
                                        BCDOUT=>bcdDataY
                        );


                        ------------------------------------------------
                        --                                      Output Data Mux
                        --              Select data to display on SSD
                        ------------------------------------------------
                        process(CNT(1), CNT(0), bcdDataX, bcdDataY, RST) begin
                                        if(RST = '1') then
                                                        muxData <= "0000";
                                        else
                                                        case (CNT) is
                                                                        when "000" => muxData <=
bcdDataX(3 downto 0);
```

```
                                                when "001" => muxData <=
bcdDataX(7 downto 4);
                                                when "010" => muxData <=
bcdDataX(11 downto 8);
                                                when "011" => muxData <=
bcdDataX(15 downto 12);
                                                when "100" => muxData <=
bcdDataY(3 downto 0);
                                                when "101" => muxData <=
bcdDataY(7 downto 4);
                                                when "110" => muxData <=
bcdDataY(11 downto 8);
                                                when "111" => muxData <=
bcdDataY(15 downto 12);
                                                when others => muxData <=
"0000";
                                        end case;
                                end if;
                        end process;



                        --------------------------------
                        --                  Segment Decoder
                        -- Determines cathode pattern
                        --   to display digit on SSD
                        --------------------------------
                        process(DCLK, RST) begin
                                if(RST = '1') then
                                        SEG <= "1000000";
                                elsif rising_edge(DCLK) then
                                        case (muxData) is

                                                when   X"0"   =>   SEG   <=
"1000000";  -- 0
                                                when   X"1"   =>   SEG   <=
"1111001";  -- 1
                                                when   X"2"   =>   SEG   <=
"0100100";  -- 2
                                                when   X"3"   =>   SEG   <=
"0110000";  -- 3
```

```
                                                when   X"4"   =>   SEG   <=
"0011001";  -- 4
                                                when   X"5"   =>   SEG   <=
"0010010";  -- 5
                                                when   X"6"   =>   SEG   <=
"0000010";  -- 6
                                                when   X"7"   =>   SEG   <=
"1111000";  -- 7
                                                when   X"8"   =>   SEG   <=
"0000000";  -- 8
                                                when   X"9"   =>   SEG   <=
"0010000";  -- 9
                                                when   others  =>  SEG   <=
"1000000";

                                        end case;
                        end if;
        end process;



        ----------------------------------
        --                      Anode Decoder
        --   Determines digit digit to
        --  illuminate for clock period
        ----------------------------------
        process(DCLK, RST) begin
                        if(RST = '1') then
                                        AN <= "00000000";
                        elsif rising_edge(DCLK) then
                                        case (CNT) is

                                                when   "000"   =>   AN   <=
"11111110";  -- 0
                                                when   "001"   =>   AN   <=
"11111101";  -- 1
                                                when   "010"   =>   AN   <=
"11111011";  -- 2
                                                when   "011"   =>   AN   <=
"11110111";  -- 3
```

```
                                                when   "100"   =>   AN   <=
"11101111";  -- 4
                                                when   "101"   =>   AN   <=
"11011111";  -- 5
                                                when   "110"   =>   AN   <=
"10111111";  -- 6
                                                when   "111"   =>   AN   <=
"01110111";  -- 7
                                                when   others   =>   AN   <=
"11111111";

                                        end case;
                        end if;
        end process;


        --------------------------------
        --                      3 Bit Counter
        --          Used to select which diigt
        --           is being illuminated, and
        --         selects data to be displayed
        --------------------------------
        process(DCLK) begin

                        if rising_edge(DCLK) then
                                        CNT <= CNT + 1;
                        end if;

        end process;

        --------------------------------
        --                      1khz Clock Divider
        --  Timing for refreshing the
        --                       SSD, etc.
        --------------------------------
        process(CLK) begin

                        if rising_edge(CLK) then
                                        if(clkCount = cntEndVal) then
                                                DCLK <= '1';
                                                clkCount <= X"0000";
```

```
                                        else
                                            DCLK <= '0';
                                            clkCount <= clkCount + 1;
                                        end if;
                                end if;

                        end process;

end Behavioral;
```

Binary_to_BCD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- ============================================================================
--                                                 Define Module, Inputs and Outputs
-- ============================================================================
entity Binary_To_BCD is
   Port ( CLK : in  STD_LOGIC;
          -- 100Mhz CLK
          RST : in  STD_LOGIC;                                                    --
             Reset
          START : in  STD_LOGIC;                                                  --
             Signal to initialize conversion
          BIN : in  STD_LOGIC_VECTOR (9 downto 0);          --      Binary value to be converted
          BCDOUT : inout  STD_LOGIC_VECTOR (15 downto 0));  --      4 digit binary coded decimal output
end Binary_To_BCD;

architecture Behavioral of Binary_To_BCD is



                -- Stores number of shifts executed
                signal shiftCount : STD_LOGIC_VECTOR(4 downto 0) := "00000";
                -- Temporary shift regsiter
                signal tmpSR : STD_LOGIC_VECTOR(27 downto 0) := (others=>'0');

                -- FSM States
                type state_type is (Idle, Init, Shift, Check, Done);

                -- Present state, Next State
                signal STATE, NSTATE : state_type;

begin
```

```
---------------------------------
--                    State Register
---------------------------------
STATE_REGISTER: process(CLK, RST) begin
                    if (RST = '1') then
                                        STATE <= Idle;
                    elsif rising_edge(CLK) then
                                        STATE <= NSTATE;
                    end if;
end process;


---------------------------------
--                    Output Logic/Assignment
---------------------------------
OUTPUT_LOGIC: process (CLK, RST)
begin
                    if(RST = '1') then
                                        -- Reset/clear values
                                        BCDOUT <= X"0000";
                                        tmpSR <= X"0000000";

                    elsif rising_edge(CLK) then

                                        case (STATE) is

                                                            when Idle =>
                                                                                BCDOUT <= BCDOUT;
                                                            -- Output does not change
                                                                                tmpSR <= X"0000000";
                                                    -- Temp shift reg empty
                                                            when Init =>
                                                                                BCDOUT <= BCDOUT;
                                                            -- Output does not change
                                                                                tmpSR                       <=
"000000000000000000" & BIN;                 -- Copy input to lower 10 bits
                                                            when Shift =>
                                                                                BCDOUT <= BCDOUT;
                                                            -- Output does not change
                                                                                tmpSR     <=     tmpSR(26
downto 0) & '0';                                    -- Shift left 1 bit

                                                                                shiftCount <= shiftCount +
'1';                                        -- Count the shift
                                                            when Check =>
                                                                                BCDOUT <= BCDOUT;

                                                                                if(shiftCount /= X"C") then
```

```
        if(tmpSR(27 downto 24) >= X"5") then
        tmpSR(27 downto 24) <= tmpSR(27 downto 24) + X"3";
                                                                    end if;

        if(tmpSR(23 downto 20) >= X"5") then
        tmpSR(23 downto 20) <= tmpSR(23 downto 20) + X"3";
                                                                    end if;

        if(tmpSR(19 downto 16) >= X"5") then
        tmpSR(19 downto 16) <= tmpSR(19 downto 16) + X"3";
                                                                    end if;

        if(tmpSR(15 downto 12) >= X"5") then
        tmpSR(15 downto 12) <= tmpSR(15 downto 12) + X"3";
                                                                    end if;
                                                                end if;
                                                        when Done =>
                                                                BCDOUT  <=  tmpSR(27
downto 12);                             -- Assign output the new BCD values
                                                                tmpSR <= X"0000000";
                                                                shiftCount <= "00000";
                                                -- Clear shift count
                                                end case;
                        end if;
        end process;
        --------------------------------
        --              Next State Logic
        --------------------------------
        NEXT_STATE_LOGIC: process (START, shiftCount, STATE)
        begin
                        -- Define default state to avoid latches
                        NSTATE <= Idle;
```

```
                case (STATE) is
                        when Idle =>
                                if(START = '1') then
                                        NSTATE <= Init;
                                else
                                        NSTATE <= Idle;
                                end if;
                        when Init =>
                                NSTATE <= Shift;
                        when Shift =>
                                NSTATE <= Check;
                        when Check =>
                                if(shiftCount /= X"C") then
                                        NSTATE <= Shift;
                                else
                                        NSTATE <= Done;
                                end if;
                        when Done =>
                                NSTATE <= Idle;
                        when others =>
                                NSTATE <= Idle;
                end case;
        end process;

end Behavioral;
```

Clk_Div 5 Hz

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ClkDiv_5Hz is
   Port ( CLK : in  STD_LOGIC;                    -- 100MHz onboard clock
          RST : in  STD_LOGIC;                    -- Reset
          CLKOUT : inout  STD_LOGIC);             -- New clock output
end ClkDiv_5Hz;

architecture Behavioral of ClkDiv_5Hz is

                -- Current count value
                signal clkCount : STD_LOGIC_VECTOR(23 downto 0) := (others => '0');
                -- Value to toggle output clock at
                constant cntEndVal : STD_LOGIC_VECTOR(23 downto 0) := X"989680";
```

```
begin
                    ------------------------------------------------
                    --        5Hz Clock Divider Generates Send/Receive signal
                    ------------------------------------------------
                    process(CLK, RST) begin
                                -- Reset clock
                                if(RST = '1')  then
                                            CLKOUT <= '0';
                                            clkCount <= X"000000";
                                elsif rising_edge(CLK) then
                                            if(clkCount = cntEndVal) then
                                                        CLKOUT <= NOT CLKOUT;
                                                        clkCount <= X"000000";
                                            else
                                                        clkCount <= clkCount + '1';
                                            end if;
                                end if;
                    end process;
end Behavioral;
```

Cod pentru aplicația JAVA

Game

```
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferStrategy;
import java.util.Random;

public class Game extends Canvas implements Runnable{
        public static final int WIDTH=640, HEIGHT=WIDTH*9/12;

        private static final long serialVersionUID = -1442798787354930462L;
        private Thread thread;
        private boolean running=false;
        private Handler handler;
        private HUD hud;
        Random r=new Random();
        public Game() {
```

```
		handler=new Handler();
		this.addKeyListener(new KeyInput(handler));
		new Window(WIDTH, HEIGHT, "SSC Application", this);
		hud=new HUD();
			handler.addObject(new Player(250,250,ID.Player,handler));
			//handler.addObject(new
BasicEnemy(r.nextInt(WIDTH),r.nextInt(HEIGHT),ID.BasicEnemy,handler));

	}

	public synchronized void start() {
		thread=new Thread(this);
		thread.start();
		running=true;

	}
	public synchronized void stop() {
		try {
			thread.join();
			running=false;
		}catch(Exception e) {
			e.printStackTrace();
		}

	}


	public void run() {
		this.requestFocus();
		long lastTime=System.nanoTime();
		double amountOfTicks=60.0;
		double ns=1000000000/amountOfTicks;
		double delta=0;
		long timer=System.currentTimeMillis();
		int frames=0;
		while(running) {
			long now=System.nanoTime();
			delta+=(now-lastTime)/ns;
			lastTime=now;
			while(delta>=1)
			{
```

```
                tick();
                delta--;
            }
            if (running) {
                render();
                frames++;
            }
            if (System.currentTimeMillis()-timer>1000) {
                timer+=1000;
                System.out.println("FPS:"+frames);
                frames=0;
            }
        }
        stop();


}
private void tick() {
        handler.tick();
        hud.tick();
}
private void render() {
        BufferStrategy bs=this.getBufferStrategy();
        if (bs==null){
            this.createBufferStrategy(3);
            return;
        }
        Graphics g=bs.getDrawGraphics();
        g.setColor(Color.BLUE);
        g.fillRect(0, 0, WIDTH, HEIGHT);

        handler.render(g);
//      hud.render(g);
        g.dispose();
        bs.show();
}
public static int clap(int var,int min, int max){
        if (var>=max)
        {
            return var=max;
        }
```

```
		else if (var<=min)
		{
			return var=min;
		}
		else
			return var;
	}
	public static void main(String[] args) {
		new Game();

	}

}
```

GameObject

```
import java.awt.Graphics;
import java.awt.Rectangle;

public abstract class  GameObject {
	protected int x,y;
	protected ID id;
	protected int velX,velY;
	public GameObject(int x, int y, ID id) {
		this.x = x;
		this.y = y;
		this.id = id;
	}
	public abstract void tick();
	public abstract void render(Graphics g);
	public abstract Rectangle getBounds();
	public int getX() {
		return x;
	}
	public void setX(int x) {
		this.x = x;
	}
	public int getY() {
		return y;
	}
	public void setY(int y) {
		this.y = y;
```

```
	}
	public ID getId() {
		return id;
	}
	public void setId(ID id) {
		this.id = id;
	}
	public int getVelX() {
		return velX;
	}
	public void setVelX(int velX) {
		this.velX = velX;
	}
	public int getVelY() {
		return velY;
	}
	public void setVelY(int velY) {
		this.velY = velY;
	}

}
```

Handler

```
import java.awt.Graphics;
import java.util.LinkedList;

public class Handler {
	LinkedList<GameObject> object=new LinkedList<GameObject>();
	public void tick() {
		int i;
		for (i=0;i<object.size();i++) {
			GameObject tempObject=object.get(i);
			tempObject.tick();
		}
	}
	public void render(Graphics g) {
		int i;
		for (i=0;i<object.size();i++) {
			GameObject tempObject=object.get(i);
			tempObject.render(g);
		}
	}
```

```
	public void addObject(GameObject object){
		this.object.add(object);
	}
	public void removeObject(GameObject object){
		this.object.remove(object);
	}
}
```

Patrat

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;

public class Patrat{
	private int x;
	private int y;
	private int velX, velY;
	private Handler handler;
	public Patrat(int x, int y, Handler handler) {
		this.x=x;
		this.y=y;
		this.handler=handler;

	}

	public void tick() {
		x+=velX;
		y+=velY;
		x=Game.clap(x, 0, Game.WIDTH-32);
		y=Game.clap(y, 0, Game.HEIGHT-80);

	}

	public void render(Graphics g) {
		g.setColor(Color.GREEN);
		g.fillRect(x, y, 32, 32);
		Graphics2D g2d=(Graphics2D)g;

	}
```

```
	public Rectangle getBounds() {
		return new Rectangle(x,y,16,16);
	}



}
```

Player

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;

public class Player extends GameObject{
	Handler handler;
	public Player(int x, int y, ID id, Handler handler) {
		super(x, y, id);
		this.handler=handler;

	}

	@Override
	public void tick() {
		x+=velX;
		y+=velY;
		x=Game.clap(x, 0, Game.WIDTH-32);
		y=Game.clap(y, 0, Game.HEIGHT-80);
		collision();
	}

	@Override
	public void render(Graphics g) {
		g.setColor(Color.GREEN);
		g.fillRect(x, y, 32, 32);
		Graphics2D g2d=(Graphics2D)g;

	}
```

```
	@Override
	public Rectangle getBounds() {
		return new Rectangle(x,y,16,16);
	}
	private void collision() {
		for (int i=0;i<handler.object.size();i++){
			GameObject tempObject=handler.object.get(i);
			if (tempObject.getId()==ID.BasicEnemy) {
				if (getBounds().intersects(tempObject.getBounds())) {
					HUD.HEALTH-=2;
				}
			}
		}
	}


}
```

Window

```
import java.awt.Canvas;

import javax.swing.JFrame;

public class Window extends Canvas{
	/**
	 *
	 */
	private static final long serialVersionUID = 9034494958129720942L;
	/**
	 *
	 */

	public Window(int Width, int Height, String title, Game game) {
		JFrame f=new JFrame(title);
		f.setSize(Width, Height);
		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		f.setLocationRelativeTo(null);
		f.add(game);
		f.setVisible(true);
		game.start();
```

```
	}
}
```

KeyImput

```
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

public class KeyInput extends KeyAdapter{
	private Handler handler;
	public KeyInput(Handler handler) {
		this.handler=handler;
	}
	public void keyPressed(KeyEvent e) {
		int key=e.getKeyCode();
		//System.out.println(key);
		for (int i=0;i<handler.object.size();i++)
		{
			GameObject tempObject=handler.object.get(i);
			if (tempObject.getId()==ID.Player)
			{

				if (key==KeyEvent.VK_W) tempObject.setVelY(-5);
				if (key==KeyEvent.VK_A) tempObject.setVelX(-5);
				if (key==KeyEvent.VK_S) tempObject.setVelY(5);
				if (key==KeyEvent.VK_D) tempObject.setVelX(5);
				if (key==KeyEvent.VK_ESCAPE) System.exit(1);
			}



		}

	}
	public void keyReleased(KeyEvent e) {
		int key=e.getKeyCode();
		for (int i=0;i<handler.object.size();i++)
		{
			GameObject tempObject=handler.object.get(i);
			if (tempObject.getId()==ID.Player)
			{
```

```
                        if (key==KeyEvent.VK_W) tempObject.setVelY(0);
                        if (key==KeyEvent.VK_A) tempObject.setVelX(0);
                        if (key==KeyEvent.VK_S) tempObject.setVelY(0);
                        if (key==KeyEvent.VK_D) tempObject.setVelX(0);
                    }

                }
            }
        }
```