

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**Utilizarea afisorului PmodOLED**

**Indrumator de proiect: Dr. Ing. Dragos Florin Lisman**

**Student: Igna Alexandra Andreea**

**Data: 10.01.2024**

**Grupa: 30235**

1. Rezumat

Proiectul prezent se concentrează pe dezvoltarea și implementarea unui sistem de afișare a literei 'H' pe un ecran OLED, folosind limbajul VHDL și placa de dezvoltare Nexys 4 DDR. Scopul principal al proiectului a fost de a crea o reprezentare grafică a literei 'H', evidențiind abilitățile de programare VHDL și înțelegerea interfețelor hardware. Metodologia a inclus utilizarea tehnicii SPI pentru comunicarea cu ecranul OLED și aplicarea principiilor de design digital. Rezultatele proiectului au demonstrat succesul în afișarea clară și precisă a literei 'H', atingând obiectivele stabilite.

1. Introducere

Proiectul abordeaza integrarea tehnologiei avansate de afisare OLED cu programarea VHDL, focalizandu-se pe afisarea literei "H". Contextul actual al tehnologiei afisajelor digitale evidentiaza tranzitia rapida catre solutii de afisare de inalta calitate, precum OLED, care ofera un contrast superior si un consum redus de energie. In acest cadru, VHDL se impune ca un limbaj esential pentru proiectarea si simularea circuitelor electronice, avand o aplicabilitate larga in inginerie.

Domeniul de studiu combina ingineria electronica cu programarea, punand accent pe importanta intelegerii interactiunii dintre software si hardware. Prin acest proiect, ne propunem sa demonstram aplicarea practica a VHDL in controlul unui ecran OLED, un domeniu esential in tehnologia moderna de afisaj.

Problema centrala a proiectului este dezvoltarea unui sistem VHDL pentru afisarea literei "H" pe un ecran OLED, utilizand placa Nexys 4 DDR. Aceasta provocare implica intelegerea detaliata a interfetei SPI si implementarea eficienta a comunicatiei intre software si hardware.

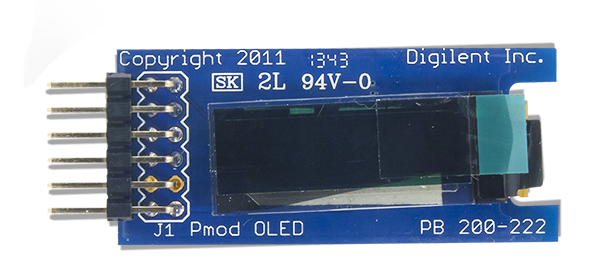
Solutia propusa se distinge prin aplicarea specifica a VHDL pentru controlul afisajului OLED, o abordare care combina in mod unic teoria circuitelor cu practica programarii. Aceasta ofera o perspectiva valoroasa asupra modului in care limbajele de descriere hardware pot fi utilizate pentru a crea solutii inovatoare in domeniul afisajelor electronice.

In raportul urmator, fiecare sectiune va explora diferite aspecte ale proiectului: de la fundamentarea teoretica, care va oferi detalii despre tehnologiile si conceptele utilizate, pana la proiectarea si implementarea sistemului, rezultatele experimentale si concluziile finale. Aceasta structura va oferi o intelegere cuprinzatoare a proiectului, demonstrand aplicarea practica a cunostintelor teoretice intr-un context real.

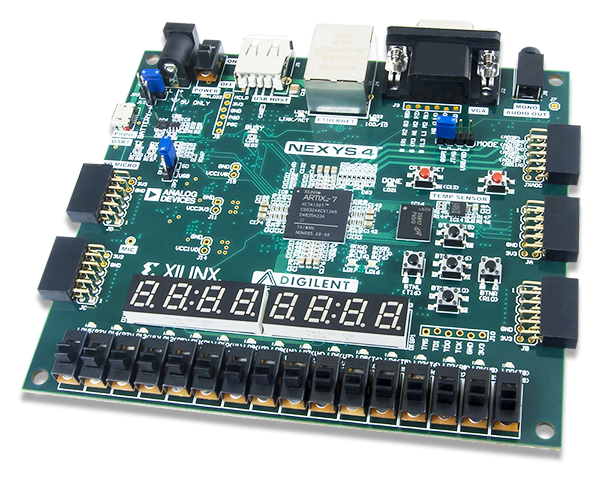
1. Fundamentare teoretica

VHDL, un acronim pentru VHSIC (Very High Speed Integrated Circuit) Hardware Description Language, este un limbaj de descriere hardware dezvoltat in anii 1980. Este utilizat in mod larg pentru proiectarea, simularea si testarea circuitelor electronice digitale. VHDL permite inginerilor sa descrie structura si functionarea sistemelor electronice la nivel inalt, facilitand modelarea comportamentala si structurala. Acest limbaj se distinge prin capacitatea sa de a modela simultan comportamentul si structura circuitului, oferind o flexibilitate sporita in designul de sisteme electronice complexe[1]. In acest proiect, VHDL este utilizat pentru a crea un model digital care controleaza un afisaj OLED, demonstrand aplicabilitatea sa in interfatarea cu tehnologiile moderne de afisaj.

Tehnologia ecranului OLED monocromatic Pmod pe care il folosesc in proiectul meu, cu o rezolutie de 128x32 pixeli, este un exemplu excelent al aplicarii tehnologiei OLED in afisaje cu dimensiuni reduse si consum redus de energie. Aceste ecrane OLED monocromatice sunt ideale pentru afisarea informatiilor simple, cum ar fi text sau simboluri, datorita claritatii si contrastului ridicat pe care il ofera. Desi nu dispun de culori, aceste ecrane sunt extrem de eficiente pentru aplicatii care necesita un afisaj compact si economic din punct de vedere energetic. Utilizarea unui astfel de ecran OLED permite o afisare clara si eficienta a literei "H", demonstrand capacitatea ecranului de a reda detalii fine intr-un format restrans.



Placa Nexys 4 DDR este un instrument esentia. Este o placa de dezvoltare bazata pe FPGA (Field-Programmable Gate Array), care permite utilizatorilor sa creeze circuite digitale complexe. Caracteristicile sale includ un procesor puternic, o varietate de porturi de intrare/iesire si o capacitate mare de memorie. In contextul proiectului meu, placa Nexys 4 DDR serveste ca platforma centrala pentru rularea si demonstrarea codului VHDL care controleaza ecranul OLED. Permite o interactiune flexibila si directa cu hardware-ul si ofera feedback vizual imediat, esential pentru procesul de dezvoltare si depistare a erorilor[2].

****

Interfata Serial Peripheral Interface (SPI) joaca un rol crucial in acest proiect, facilitand comunicarea intre placa Nexys 4 DDR si ecranul OLED. SPI este un protocol de comunicare seriala care permite schimbul rapid de date intre microcontrolere si diverse dispozitive periferice, cum ar fi ecranele OLED. In cazul nostru, SPI este folosit pentru a transmite date si comenzi catre ecranul OLED, permitand controlul precis al afisajului. Avantajele SPI includ viteza mare de transfer si simplitatea circuitului, ceea ce il face ideal pentru proiecte care necesita comunicare rapida si eficienta intre componente. Implementarea protocolului SPI in designul VHDL permite un control detaliat asupra modului in care litera "H" este reprezentata pe ecranul OLED.

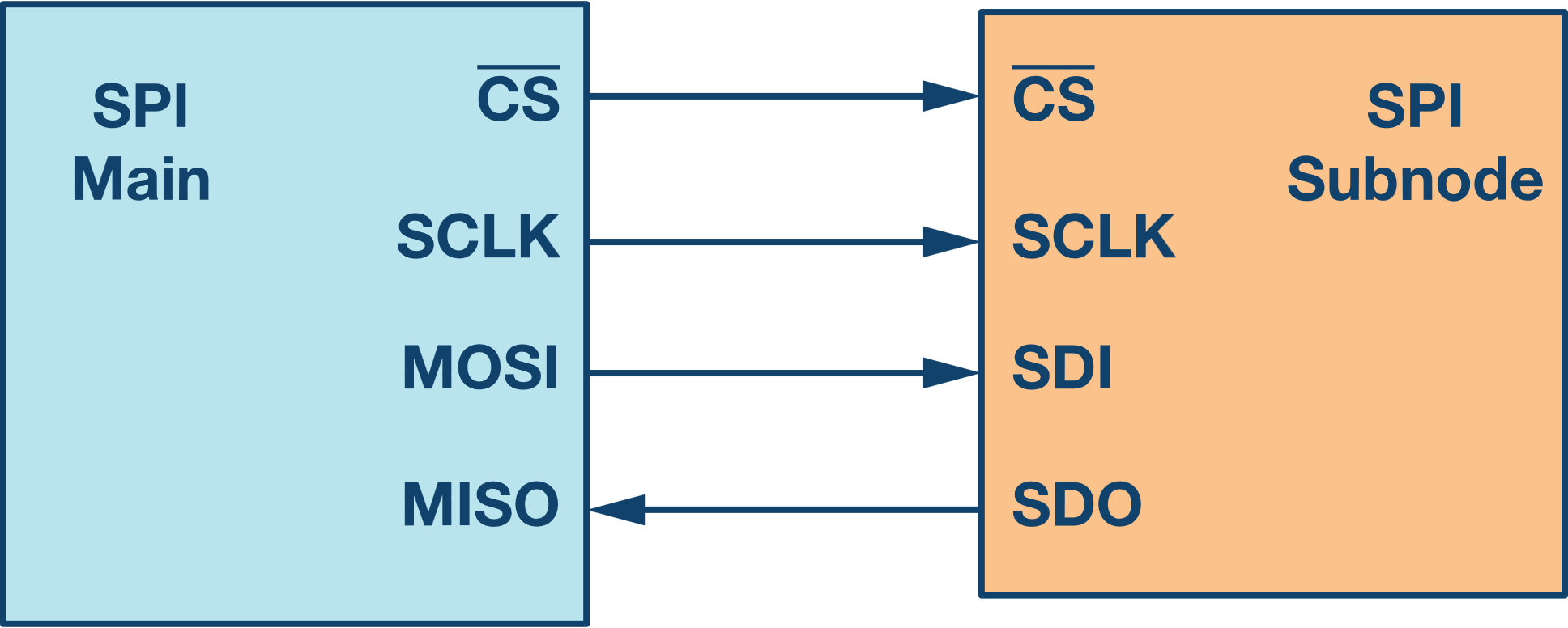
Interfata SPI este folosita pentru a comunica intre placa Nexys 4 DDR (master) si ecranul OLED (slave). Dintre cele patru semnale SPI standard, proiectul meu va utiliza trei:

SCLK (Serial Clock): acest semnal este generat de placa Nexys 4 DDR si sincronizeaza transferul de date catre ecranul OLED.

MOSI (Master Out Slave In): prin aceasta linie, placa Nexys 4 DDR trimite date si comenzi catre ecranul OLED. Este canalul principal prin care instructiunile si datele sunt transmise.

SS (Slave Select): acest semnal este folosit pentru a activa ecranul OLED inainte de a incepe comunicatia.

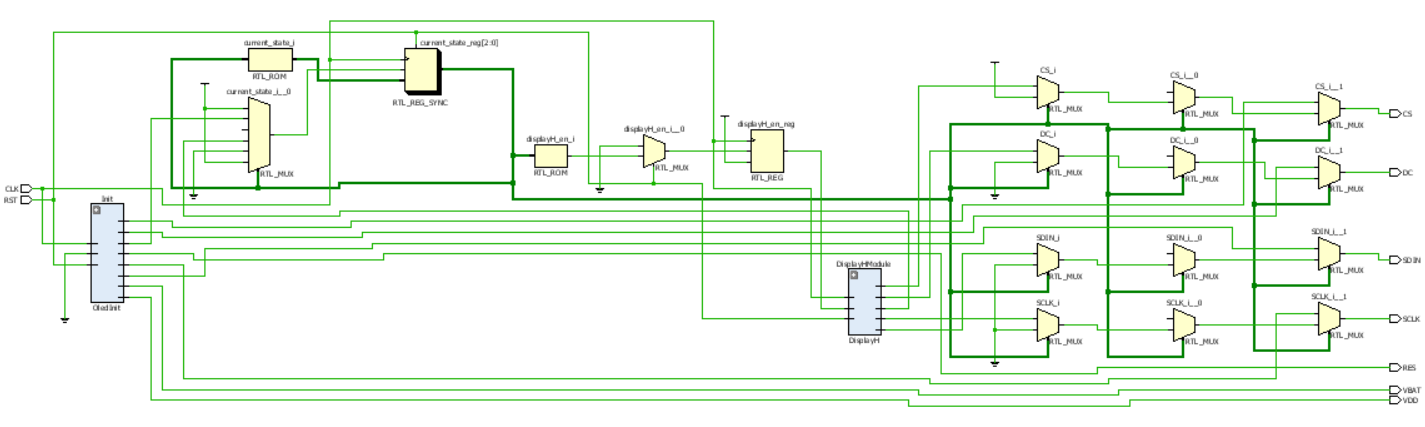
In acest caz, MISO (Master In Slave Out) nu este utilizat, deoarece comunicatia este unidirectionala - de la placa Nexys 4 DDR catre ecranul OLED, fara a necesita feedback. Acest mod de operare simplifica schema circuitului si este suficient pentru afisarea literei "H" pe ecranul OLED.

****

Prin implementarea unei comunicatii eficiente SPI intre placa Nexys 4 DDR si ecranul OLED, proiectul demonstreaza o intelegere profunda a importantei sincronizarii si a comunicatiei intre diferitele componente ale unui sistem electronic. Acest lucru este deosebit de relevant intr-o epoca in care interconectivitatea si integrarea dispozitivelor electronice sunt esentiale.

1. Proiectare si implementare

Schema bloc

****

Schema bloc prezentata ilustreaza arhitectura detaliata a sistemului VHDL utilizat pentru a controla afisajul OLED in cadrul proiectului. Aceasta evidentiaza interactiunea intre placa Nexys 4 DDR si modulul OLED, precum si fluxul semnalelor de control si de date prin intermediul interfetei SPI.

Placa Nexys 4 DDR (Master):

* CLK: Ceasul de sistem este generat de FPGA si ofera ritmul pentru toate operatiunile sincrone.
* RST: Semnalul de reset este utilizat pentru a initia starea initiala a sistemului si a modulelor VHDL la pornire sau in cazul unei resetari manuale.

Initializarea si Controlul OLED:

Semnalele de initializare sunt transmise de la FPGA catre ecranul OLED pentru a-l pregati pentru afisare. Aceasta include configurarea parametrilor de afisare si sincronizarea cu ceasul sistemului.

Modulul de Afisare 'DisplayH':

Acest modul genereaza datele necesare pentru reprezentarea literei "H" si le transmite ecranului OLED prin intermediul interfetei SPI.

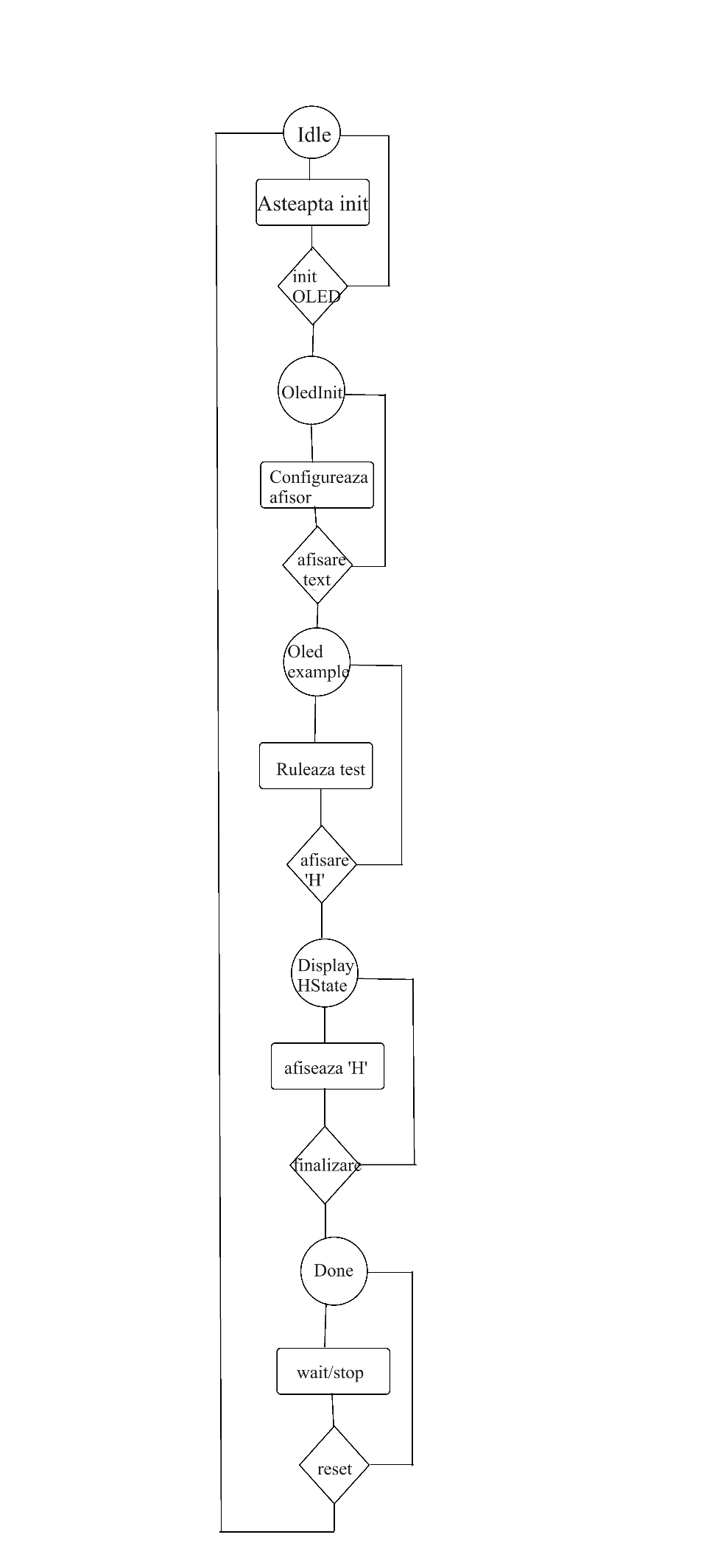
Interfata SPI:

* SCLK: Semnalul de ceas pentru SPI coordoneaza transferul de date.
* SDIN: Linia de date prin care sunt transmise secventele de biti catre ecranul OLED.
* CS: Semnalul de selectare a dispozitivului este activat pentru a incepe comunicatia cu ecranul OLED.

Ecranul OLED:

Primeste si afiseaza datele primite prin linia SDIN, sincronizand fiecare bit cu semnalul de ceas SCLK. Selectarea ecranului este controlata de semnalul CS.

Organigrama pentru PmodOLEDCtrl



Metoda Experimentala Utilizata:

Proiectul a combinat hardware-ul specific cu programarea VHDL pentru a atinge obiectivele propuse. Placa Nexys 4 DDR, echipata cu un FPGA, a fost utilizata ca platforma de dezvoltare hardware, in timp ce ecranul OLED monocromatic a servit ca dispozitiv de afisare. Programarea VHDL a fost folosita pentru a dezvolta si implementa logica necesara controlului afisajului OLED, inclusiv initializarea acestuia si afisarea literei "H".

Solutia Aleasa si Motivatia:

Din diversele optiuni posibile, combinatia de placa Nexys 4 DDR si ecranul OLED a fost aleasa datorita capabilitatilor sale de a demonstra eficient utilizarea VHDL in controlul afisajelor. Nexys 4 DDR ofera flexibilitate in programare si capacitate suficienta pentru a gestiona logica complexa, iar ecranul OLED asigura o calitate superioara a afisarii.

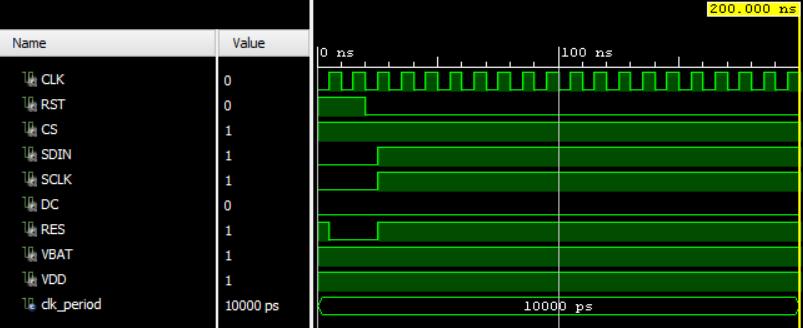
Algoritmii Implementati:

In modulul de initializare OLED, sunt implementate secvente pentru configurarea ecranului pentru operare. Modulul de afisare gestioneaza datele care vor fi afisate, inclusiv generarea si transmiterea bitmap-ului pentru litera "H". Algoritmul SPI controleaza comunicatia intre FPGA si ecranul OLED, asigurand transmiterea datelor si comenzi in mod eficient si sincronizat.

Detalii de Implementare:

Arhitectura software include module VHDL interconectate care comunica intre ele si cu hardware-ul. Interfata cu utilizatorul este minimala, concentrandu-se pe afisarea literei "H".

1. Rezultate experimentale

****

Aici current\_state se afla in stare initiala, iar CS este 1 ceea ce inseamna ca dispozitivul slave (ecranul oled) nu este selectat in acest moment al simularii. Nu am avut la dispozitie o placuta Nexys 4 sau un PmodOLED pentru a verifica cele scrise pana acum.

1. Concluzii

Problema rezolvata:

Proiectul a avut ca scop afisarea literei "H" pe un ecran OLED folosind un program VHDL pe o placa Nexys 4 DDR. Am reusit sa indeplinesc acest obiectiv, invatand mult despre programarea VHDL si despre cum functioneaza ecranele OLED.

Contributii proprii:

Contributia mea a fost sa scriu codul care controleaza ecranul si sa creez un design care sa reprezinte clar litera "H". Unul dintre avantajele majore ale proiectului a fost ca mi-a permis sa aplic cunostintele teoretice intr-o situatie reala. Printre provocari s-a numarat intelegerea profunda a modului in care componentele hardware comunica intre ele.

Dezvoltari viitoare:

Acest proiect ar putea fi folosit in sisteme unde informatiile trebuie afisate clar, cum ar fi panouri de afisaj electronice sau dispozitive inteligente. In viitor, ar fi interesant sa extindem proiectul pentru a include mai multe caractere si simboluri, sau sa-l facem interactiv, astfel incat sa poata raspunde la comenzi sau la alte semnale.

1. Bibliografie

[1] <https://ro.wikipedia.org/wiki/VHDL>

[2] <https://digilent.com/reference/programmable-logic/nexys-4/start>

[3] <https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html>

**Anexa**

**PmodOLEDCtrl:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.ALL;

use ieee.std\_logic\_arith.all;

entity PmodOLEDCtrl is

Port (

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

CS : out STD\_LOGIC;

SDIN : out STD\_LOGIC;

SCLK : out STD\_LOGIC;

DC : out STD\_LOGIC;

RES : out STD\_LOGIC;

VBAT : out STD\_LOGIC;

VDD : out STD\_LOGIC);

end PmodOLEDCtrl;

architecture Behavioral of PmodOLEDCtrl is

component OledInit is

Port ( CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

EN : in STD\_LOGIC;

CS : out STD\_LOGIC;

SDO : out STD\_LOGIC;

SCLK : out STD\_LOGIC;

DC : out STD\_LOGIC;

RES : out STD\_LOGIC;

VBAT : out STD\_LOGIC;

VDD : out STD\_LOGIC;

FIN : out STD\_LOGIC);

end component;

component DisplayH is

Port ( CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

EN : in STD\_LOGIC;

CS : out STD\_LOGIC;

SDO : out STD\_LOGIC;

SCLK : out STD\_LOGIC;

DC : out STD\_LOGIC;

FIN : out STD\_LOGIC);

end component;

type states is (Idle, OledInitialize, OledExample, DisplayHState, Done);

signal current\_state : states := Idle;

signal init\_en : STD\_LOGIC := '0';

signal init\_done : STD\_LOGIC;

signal init\_cs : STD\_LOGIC;

signal init\_sdo : STD\_LOGIC;

signal init\_sclk : STD\_LOGIC;

signal init\_dc : STD\_LOGIC;

signal example\_en : STD\_LOGIC := '0';

signal example\_done : STD\_LOGIC;

signal example\_cs : STD\_LOGIC;

signal example\_sdo : STD\_LOGIC;

signal example\_sclk : STD\_LOGIC;

signal example\_dc : STD\_LOGIC;

signal displayH\_en : STD\_LOGIC := '0';

signal displayH\_done : STD\_LOGIC;

signal displayH\_cs : STD\_LOGIC;

signal displayH\_sdo : STD\_LOGIC;

signal displayH\_sclk : STD\_LOGIC;

signal displayH\_dc : STD\_LOGIC;

begin

Init: OledInit port map(

CLK => CLK, RST => RST, EN => init\_en,

CS => init\_cs, SDO => init\_sdo, SCLK => init\_sclk,

DC => init\_dc, RES => RES, VBAT => VBAT, VDD => VDD,

FIN => init\_done

);

DisplayHModule: DisplayH port map(

CLK => CLK, RST => RST, EN => displayH\_en,

CS => displayH\_cs, SDO => displayH\_sdo, SCLK => displayH\_sclk,

DC => displayH\_dc, FIN => displayH\_done

);

-- MUXes to control which block is active and routing their outputs

CS <= init\_cs when (current\_state = OledInitialize) else

example\_cs when (current\_state = OledExample) else

displayH\_cs when (current\_state = DisplayHState) else

'1';

SDIN <= init\_sdo when (current\_state = OledInitialize) else

example\_sdo when (current\_state = OledExample) else

displayH\_sdo when (current\_state = DisplayHState) else

'0';

SCLK <= init\_sclk when (current\_state = OledInitialize) else

example\_sclk when (current\_state = OledExample) else

displayH\_sclk when (current\_state = DisplayHState) else

'0';

DC <= init\_dc when (current\_state = OledInitialize) else

example\_dc when (current\_state = OledExample) else

displayH\_dc when (current\_state = DisplayHState) else

'0';

-- State machine process

process(CLK)

begin

if rising\_edge(CLK) then

if RST = '1' then

current\_state <= Idle;

else

case current\_state is

when Idle =>

current\_state <= OledInitialize;

when OledInitialize =>

if init\_done = '1' then

current\_state <= OledExample;

end if;

when OledExample =>

if example\_done = '1' then

current\_state <= DisplayHState;

end if;

when DisplayHState =>

displayH\_en <= '1';

if displayH\_done = '1' then

current\_state <= Done;

end if;

when Done =>

-- Remain in this state or add additional logic as needed

when others =>

current\_state <= Idle;

end case;

end if;

end if;

end process;

end Behavioral;

**OledInit:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity OledInit is

Port ( CLK : in STD\_LOGIC; --System Clock

RST : in STD\_LOGIC; --Global Synchronous Reset

EN : in STD\_LOGIC; --Block enable pin

CS : out STD\_LOGIC; --SPI Chip Select

SDO : out STD\_LOGIC; --SPI data out

SCLK : out STD\_LOGIC; --SPI Clock

DC : out STD\_LOGIC; --Data/Command Pin

RES : out STD\_LOGIC; --PmodOLED RES

VBAT : out STD\_LOGIC; --VBAT enable

VDD : out STD\_LOGIC; --VDD enable

FIN : out STD\_LOGIC); --OledInit Finish Flag

end OledInit;

architecture Behavioral of OledInit is

COMPONENT SpiCtrl

PORT(

CLK : IN std\_logic;

RST : IN std\_logic;

SPI\_EN : IN std\_logic;

SPI\_DATA : IN std\_logic\_vector(7 downto 0);

CS : OUT std\_logic;

SDO : OUT std\_logic;

SCLK : OUT std\_logic;

SPI\_FIN : OUT std\_logic

);

END COMPONENT;

COMPONENT Delay

PORT(

CLK : IN std\_logic;

RST : IN std\_logic;

DELAY\_MS : IN std\_logic\_vector(11 downto 0);

DELAY\_EN : IN std\_logic;

DELAY\_FIN : OUT std\_logic

);

END COMPONENT;

type states is (Transition1,

Transition2,

Transition3,

Transition4,

Transition5,

Idle,

VddOn,

Wait1,

DispOff,

ResetOn,

Wait2,

ResetOff,

ChargePump1,

ChargePump2,

PreCharge1,

PreCharge2,

VbatOn,

Wait3,

DispContrast1,

DispContrast2,

InvertDisp1,

InvertDisp2,

ComConfig1,

ComConfig2,

DispOn,

FullDisp,

Done

);

signal current\_state : states := Idle;

signal after\_state : states := Idle;

signal temp\_dc : STD\_LOGIC := '0';

signal temp\_res : STD\_LOGIC := '1';

signal temp\_vbat : STD\_LOGIC := '1';

signal temp\_vdd : STD\_LOGIC := '1';

signal temp\_fin : STD\_LOGIC := '0';

signal temp\_delay\_ms : STD\_LOGIC\_VECTOR (11 downto 0) := (others => '0');

signal temp\_delay\_en : STD\_LOGIC := '0';

signal temp\_delay\_fin : STD\_LOGIC;

signal temp\_spi\_en : STD\_LOGIC := '0';

signal temp\_spi\_data : STD\_LOGIC\_VECTOR (7 downto 0) := (others => '0');

signal temp\_spi\_fin : STD\_LOGIC;

begin

SPI\_COMP: SpiCtrl PORT MAP (

CLK => CLK,

RST => RST,

SPI\_EN => temp\_spi\_en,

SPI\_DATA => temp\_spi\_data,

CS => CS,

SDO => SDO,

SCLK => SCLK,

SPI\_FIN => temp\_spi\_fin

);

DELAY\_COMP: Delay PORT MAP (

CLK => CLK,

RST => RST,

DELAY\_MS => temp\_delay\_ms,

DELAY\_EN => temp\_delay\_en,

DELAY\_FIN => temp\_delay\_fin

);

DC <= temp\_dc;

RES <= temp\_res;

VBAT <= temp\_vbat;

VDD <= temp\_vdd;

FIN <= temp\_fin;

--Delay 100 ms after VbatOn

temp\_delay\_ms <= "000001100100" when (after\_state = DispContrast1) else --100 ms

"000000000001"; --1ms

STATE\_MACHINE : process (CLK)

begin

if(rising\_edge(CLK)) then

if(RST = '1') then

current\_state <= Idle;

temp\_res <= '0';

else

temp\_res <= '1';

case (current\_state) is

when Idle =>

if(EN = '1') then

temp\_dc <= '0';

current\_state <= VddOn;

end if;

--Initialization Sequence

--This should be done everytime the PmodOLED is started

when VddOn =>

temp\_vdd <= '0';

current\_state <= Wait1;

when Wait1 =>

after\_state <= DispOff;

current\_state <= Transition3;

when DispOff =>

temp\_spi\_data <= "10101110"; --0xAE

after\_state <= ResetOn;

current\_state <= Transition1;

when ResetOn =>

temp\_res <= '0';

current\_state <= Wait2;

when Wait2 =>

after\_state <= ResetOff;

current\_state <= Transition3;

when ResetOff =>

temp\_res <= '1';

after\_state <= ChargePump1;

current\_state <= Transition3;

when ChargePump1 =>

temp\_spi\_data <= "10001101"; --0x8D

after\_state <= ChargePump2;

current\_state <= Transition1;

when ChargePump2 =>

temp\_spi\_data <= "00010100"; --0x14

after\_state <= PreCharge1;

current\_state <= Transition1;

when PreCharge1 =>

temp\_spi\_data <= "11011001"; --0xD9

after\_state <= PreCharge2;

current\_state <= Transition1;

when PreCharge2 =>

temp\_spi\_data <= "11110001"; --0xF1

after\_state <= VbatOn;

current\_state <= Transition1;

when VbatOn =>

temp\_vbat <= '0';

current\_state <= Wait3;

when Wait3 =>

after\_state <= DispContrast1;

current\_state <= Transition3;

when DispContrast1=>

temp\_spi\_data <= "10000001"; --0x81

after\_state <= DispContrast2;

current\_state <= Transition1;

when DispContrast2=>

temp\_spi\_data <= "00001111"; --0x0F

after\_state <= InvertDisp1;

current\_state <= Transition1;

when InvertDisp1 =>

temp\_spi\_data <= "10100001"; --0xA1

after\_state <= InvertDisp2;

current\_state <= Transition1;

when InvertDisp2 =>

temp\_spi\_data <= "11001000"; --0xC8

after\_state <= ComConfig1;

current\_state <= Transition1;

when ComConfig1 =>

temp\_spi\_data <= "11011010"; --0xDA

after\_state <= ComConfig2;

current\_state <= Transition1;

when ComConfig2 =>

temp\_spi\_data <= "00100000"; --0x20

after\_state <= DispOn;

current\_state <= Transition1;

when DispOn =>

temp\_spi\_data <= "10101111"; --0xAF

after\_state <= Done;

current\_state <= Transition1;

--END Initialization sequence

--Used for debugging, This command turns the entire screen on regardless of memory

when FullDisp =>

temp\_spi\_data <= "10100101"; --0xA5

after\_state <= Done;

current\_state <= Transition1;

--Done state

when Done =>

if(EN = '0') then

temp\_fin <= '0';

current\_state <= Idle;

else

temp\_fin <= '1';

end if;

--SPI transitions

--1. Set SPI\_EN to 1

--2. Waits for SpiCtrl to finish

--3. Goes to clear state (Transition5)

when Transition1 =>

temp\_spi\_en <= '1';

current\_state <= Transition2;

when Transition2 =>

if(temp\_spi\_fin = '1') then

current\_state <= Transition5;

end if;

--Delay Transitions

--1. Set DELAY\_EN to 1

--2. Waits for Delay to finish

--3. Goes to Clear state (Transition5)

when Transition3 =>

temp\_delay\_en <= '1';

current\_state <= Transition4;

when Transition4 =>

if(temp\_delay\_fin = '1') then

current\_state <= Transition5;

end if;

--Clear transition

--1. Sets both DELAY\_EN and SPI\_EN to 0

--2. Go to after state

when Transition5 =>

temp\_spi\_en <= '0';

temp\_delay\_en <= '0';

current\_state <= after\_state;

--END SPI transitions

--END Delay Transitions

--END Clear transition

when others =>

current\_state <= Idle;

end case;

end if;

end if;

end process;

end Behavioral;

**SpiCtrl:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity SpiCtrl is

Port ( CLK : in STD\_LOGIC; --System CLK (100MHz)

RST : in STD\_LOGIC; --Global RST (Synchronous)

SPI\_EN : in STD\_LOGIC; --SPI block enable pin

SPI\_DATA : in STD\_LOGIC\_VECTOR (7 downto 0); --Byte to be sent

CS : out STD\_LOGIC; --Chip Select

SDO : out STD\_LOGIC; --SPI data out

SCLK : out STD\_LOGIC; --SPI clock

SPI\_FIN : out STD\_LOGIC);--SPI finish flag

end SpiCtrl;

architecture Behavioral of SpiCtrl is

type states is (Idle,

Send,

Hold1,

Hold2,

Hold3,

Hold4,

Done);

signal current\_state : states := Idle; --Signal for state machine

signal shift\_register : STD\_LOGIC\_VECTOR(7 downto 0); --Shift register to shift out SPI\_DATA saved when SPI\_EN was set

signal shift\_counter : STD\_LOGIC\_VECTOR(3 downto 0); --Keeps track how many bits were sent

signal clk\_divided : STD\_LOGIC := '1'; --Used as SCLK

signal counter : STD\_LOGIC\_VECTOR(4 downto 0) := (others => '0'); --Count clocks to be used to divide CLK

signal temp\_sdo : STD\_LOGIC := '1'; --Tied to SDO

signal falling : STD\_LOGIC := '0'; --signal indicating that the clk has just fell

begin

clk\_divided <= not counter(4); --SCLK = CLK / 32

SCLK <= clk\_divided;

SDO <= temp\_sdo;

CS <= '1' when (current\_state = Idle and SPI\_EN = '0') else

'0';

SPI\_FIN <= '1' when (current\_state = Done) else

'0';

STATE\_MACHINE : process (CLK)

begin

if(rising\_edge(CLK)) then

if(RST = '1') then --Synchronous RST

current\_state <= Idle;

else

case (current\_state) is

when Idle => --Wait for SPI\_EN to go high

if(SPI\_EN = '1') then

current\_state <= Send;

end if;

when Send => --Start sending bits, transition out when all bits are sent and SCLK is high

if(shift\_counter = "1000" and falling = '0') then

current\_state <= Hold1;

end if;

when Hold1 => --Hold CS low for a bit

current\_state <= Hold2;

when Hold2 => --Hold CS low for a bit

current\_state <= Hold3;

when Hold3 => --Hold CS low for a bit

current\_state <= Hold4;

when Hold4 => --Hold CS low for a bit

current\_state <= Done;

when Done => --Finish SPI transimission wait for SPI\_EN to go low

if(SPI\_EN = '0') then

current\_state <= Idle;

end if;

when others =>

current\_state <= Idle;

end case;

end if;

end if;

end process;

CLK\_DIV : process (CLK)

begin

if(rising\_edge(CLK)) then

if (current\_state = Send) then --start clock counter when in send state

counter <= counter + 1;

else --reset clock counter when not in send state

counter <= (others => '0');

end if;

end if;

end process;

SPI\_SEND\_BYTE : process (CLK) --sends SPI data formatted SCLK active low with SDO changing on the falling edge

begin

if(CLK'event and CLK = '1') then

if(current\_state = Idle) then

shift\_counter <= (others => '0');

shift\_register <= SPI\_DATA; --keeps placing SPI\_DATA into shift\_register so that when state goes to send it has the latest SPI\_DATA

temp\_sdo <= '1';

elsif(current\_state = Send) then

if( clk\_divided = '0' and falling = '0') then --if on the falling edge of Clk\_divided

falling <= '1'; --Indicate that it is passed the falling edge

temp\_sdo <= shift\_register(7); --send out the MSB

shift\_register <= shift\_register(6 downto 0) & '0'; --Shift through SPI\_DATA

shift\_counter <= shift\_counter + 1; --Keep track of what bit it is on

elsif(clk\_divided = '1') then --on SCLK high reset the falling flag

falling <= '0';

end if;

end if;

end if;

end process;

end Behavioral;

**Delay:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Delay is

Port ( CLK : in STD\_LOGIC; --System CLK

RST : in STD\_LOGIC; --Global RST (Synchronous)

DELAY\_MS : in STD\_LOGIC\_VECTOR (11 downto 0); --Amount of ms to delay

DELAY\_EN : in STD\_LOGIC; --Delay block enable

DELAY\_FIN : out STD\_LOGIC); --Delay finish flag

end Delay;

architecture Behavioral of Delay is

type states is (Idle,

Hold,

Done);

signal current\_state : states := Idle; --Signal for state machine

signal clk\_counter : STD\_LOGIC\_VECTOR(16 downto 0) := (others => '0'); --Counts up on every rising edge of CLK

signal ms\_counter : STD\_LOGIC\_VECTOR (11 downto 0) := (others => '0'); --Counts up when clk\_counter = 100,000

begin

--DELAY\_FIN goes HIGH when delay is done

DELAY\_FIN <= '1' when (current\_state = Done and DELAY\_EN = '1') else

'0';

--State machine for Delay block

STATE\_MACHINE : process (CLK)

begin

if(rising\_edge(CLK)) then

if(RST = '1') then --When RST is asserted switch to idle (synchronous)

current\_state <= Idle;

else

case (current\_state) is

when Idle =>

if(DELAY\_EN = '1') then --Start delay on DELAY\_EN

current\_state <= Hold;

end if;

when Hold =>

if( ms\_counter = DELAY\_MS) then --stay until DELAY\_MS has occured

current\_state <= Done;

end if;

when Done =>

if(DELAY\_EN = '0') then --Wait til DELAY\_EN is deasserted to go to IDLE

current\_state <= Idle;

end if;

when others =>

current\_state <= Idle;

end case;

end if;

end if;

end process;

--Creates ms\_counter that counts at 1KHz

CLK\_DIV : process (CLK)

begin

if(CLK'event and CLK = '1') then

if (current\_state = Hold) then

if(clk\_counter = "11000011010100000") then --100,000

clk\_counter <= (others => '0');

ms\_counter <= ms\_counter + 1; --increments at 1KHz

else

clk\_counter <= clk\_counter + 1;

end if;

else --If not in the hold state reset counters

clk\_counter <= (others => '0');

ms\_counter <= (others => '0');

end if;

end if;

end process;

end Behavioral;

**DisplayH:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

entity DisplayH is

Port ( CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

EN : in STD\_LOGIC;

CS : out STD\_LOGIC;

SDO : out STD\_LOGIC;

SCLK : out STD\_LOGIC;

DC : out STD\_LOGIC;

FIN : out STD\_LOGIC);

end DisplayH;

architecture Behavioral of DisplayH is

signal temp\_fin : STD\_LOGIC := '0';

signal temp\_cs : STD\_LOGIC := '1';

signal temp\_sdo : STD\_LOGIC := '0';

signal temp\_sclk : STD\_LOGIC := '0';

signal temp\_dc : STD\_LOGIC := '1'; -- Data command set to data

-- Bitmap pentru litera "H"

type bitmap\_array is array (0 to 4) of std\_logic\_vector(7 downto 0);

constant H\_bitmap: bitmap\_array := (

"11000011", -- Linia 1

"11000011", -- Linia 2

"11111111", -- Linia 3

"11000011", -- Linia 4

"11000011" -- Linia 5

);

signal bitmap\_index : integer range 0 to 4 := 0;

begin

process(CLK)

begin

if rising\_edge(CLK) then

if RST = '1' then

temp\_fin <= '0';

bitmap\_index <= 0;

elsif EN = '1' then

temp\_dc <= '1'; -- Set to data mode

temp\_cs <= '0'; -- Start transmission

temp\_sdo <= H\_bitmap(bitmap\_index)(7); -- Send first bit

bitmap\_index <= bitmap\_index + 1;

if bitmap\_index > 4 then

temp\_fin <= '1'; -- Finish transmission

temp\_cs <= '1'; -- End transmission

bitmap\_index <= 0;

end if;

else

temp\_fin <= '0';

bitmap\_index <= 0;

end if;

end if;

end process;

CS <= temp\_cs;

SDO <= temp\_sdo;

SCLK <= temp\_sclk;

DC <= temp\_dc;

FIN <= temp\_fin;

end Behavioral;