UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Măsurarea și afișarea accelerației pe trei direcții cu senzorul cuplat la placa Basys3 folosind grafice pentru cele trei axe

Structura sistemelor de calcul

Autori: Marcus Cristian si Marginean Teodor

Grupa: 30236

Indrumator: Lisman Florin

An universitar: 2022-2023

Facultatea de Automatica
si Calculatoare

15 Ianuarie 2023

Cuprins

1 Rezumat

Proiectul pe care l-am avut de realizat consta in generarea de grafice pentru valorile acceleratiei pe cele trei axe de coordonate. Prin comunicarea cu placuta si accelerometrul, cu ajutorul unui SPI, se configureaza si se pregateste accelerometrul pentru citirea datelor. Mai departe, datele primite se trasmit printr-un modul UART care, odata ajunse la portul COM, se trec pe un grafic folosind matplotlib din limbajul de programare python. Pentru modulele SPI si UART am folosit limbajul VHDL, in mediul de dezvoltare Xilinx Vivado.

In urma procesului de proiectare si dezvoltare, proiectul nostru a trecut prin niste etape lungi de depanare. Initial, valorile primite de la accelerometru le scriam direct pe un afisor cu sapte segmente. Dupa depanare, am trecut toate datele pe un grafic generat in python, ajungand astfel la forma actuala, finala a proiectului.

2 Introducere

2.1 Tema proiectului si tendinte in tehnologie legate de tema proiectului

Tema proiectului nostru consta in masurarea si afisarea acceleratiei pe trei directii folosind un accelerometru conectat la placa Basys3. Afisarea rezultatelor pe cele 3 axe se va face, in plus, pe un grafic generat dintr-o aplicatie externa, codata in limbajul de programare Python, folosind biblioteca de prelucrare a datelor matplotlib. [1]

Accelerometrul este un echipament care ofera posibilitatea masurarii si analizarii acceleratiei liniare si unghiulare. Aceasta functie este necesara in multe echipamente si sisteme de baza folosite in aproape orice domeniu – atat in aparatele de uz casnic, cotidian, precum si in aplicatiile industriale si de cercetare-dezvoltare de tip profesional. [5]

2.2 Domeniul de studiu si tehnologia de baza

Accelerometrul este implementat direct pe un obiect care vibreaza, ceea ce ii permite sa transforme energia vibratiilor in semnal electric, care este proportional cu acceleratia momentana a obiectului. Masurarea vibratiilor este utilizata concret, de obicei, pentru diagnosticarea functionarii masinilor, echipamentelor si a constructiilor supuse unor solicitari mari. Ele au importanta ridicata si in protejarea hard disk-urilor impotriva deteriorarii. Practic, accelerometrul la baza este nimic altceva decat un convertor al acceleratiei, care masoara miscarea proprie in spatiu. [5]

2.3 Definirea problemei si obiectivele principale

Practic, proiectul nostru consta in 2 module principale: un modul SPI (Serial Pheripheral Interface) si un modul UART (Universal Asynchronous Receiver Transmitter). Folosind limbajul de programare VHDL, am reusit sa implementam cele 2 module, fiecare cu rolurile lor.

Modulul SPI este cel care va comunica direct cu accelerometrul pentru a-i cere date si pentru a-l configura asa cum ne dorim. Pe cealalta parte, modulul UART va transmite mai departe datele primite de la accelerometru, prin comunicarea cu un port COM. Astfel, putem receptiona date citite de la portul serial si sa realizam astfel un grafic pe cele 3 axe xOyOz.

Concret, ca sarcini principale, se numara:

- Implementarea modulului SPI – in VHDL
- Implementarea modulului UART – in VHDL

- Implementarea unei aplicatii pentru generarea de grafice – in Python
- Incarcarea efectiva pe placa Basys3 si rularea aplicatiei de generare de grafice

2.4 Solutia propusa si comparatie cu solutii existente

Solutia propusa de noi consta in modelarea UART-ului si a SPI-ului, fiind practic niste adaptari la solutii existente propuse deja [3] [4], dar simplificate de noi – pastrand, totodata, precizia si acuratatea datelor furnizate pe porturi si buffer-e. Ambele module au in spate un FSM care modeleaza comportamentul lor specific si, in functie de specificatiile producatorului accelerometrului, noi ii trimitem semnale (octeti) pentru a-l configura, urmarind sa primim date in mod continuu. Asadar, diferenta dintre solutia propusa de noi si solutiile deja existente din diferite surse este ca, in principiu, am modelat modulele dupa nevoile noastre, deci am particularizat, de exemplu, la nivel de biti de control, si ne-am asigurat ca nu va fi nevoie de biti de paritate la modulul UART, si nici de mai mult de 1 bit de stop, iar pachetul de date trimis va fi de un byte (8 biti).

Limbajul de programare este exclusiv VHDL, iar ca mediu de programare am ales sa folosim Xilinx Vivado.

In urmatoarele capitole vom prezenta pe scurt pasii si detaliile pentru dezvoltarea proiectului nostru astfel:

- **Fundamentare teoretica**: in cadrul acestei sectiuni se va prezenta in amanunt principiul de functionare al unui accelerometru, comunicand printr-un modul SPI, alaturi de transmiterea efectiva de date pe un port COM folosind un modul UART.
- **Proiectare si implementare**: in cadrul acestei sectiuni se vor prezenta etapele de proiectare pentru realizarea obiectivelor, alaturi de diferite scheme bloc pentru intelegerea exacta a blackbox-ului, dar si a componentelor interne.
- **Rezultate experimentale**: in cadrul acestei sectiuni se va demonstra corectitudinea sistemului implementat de noi, efectuand astfel o simulare de grafice in diferite situatii si input-uri diferite.

3 Fundamentare teoretica

3.1 Placa de dezoltare Basys3

Basys3 este o placa de dezvoltare completa, gata de folosit, bazata pe FPGA-ul (Field Programmable Gate Arrays) familiei Artix-7, produsa de compania Xilinx. Cu capacitatea sa mare, cost redus si o colectie generoasa de porturi dispuse pe placa precum USB, VGA etc., Basys3 poate sa sustina design-uri de la cele mai simple, la cele mai extinse, complicate – precum procesoare embedded sau controller-e. Ca si caracteristici generale, acest FPGA are un clock intern de 450MHz+, un numar de 33280 celule logice dispuse in 200 de slice-uri, iar block-ul RAM are 1800Kbiti. Totodata, acest FPGA dispune si de o multitudine de LED-uri, switch-uri, si alte dispozitive de intrare-iesire. [6] Portul cheie relevant, de fapt, mai ales pentru proiectul realizat de noi este portul PMOD prin care am conectat accelerometrul la placa Basys3.

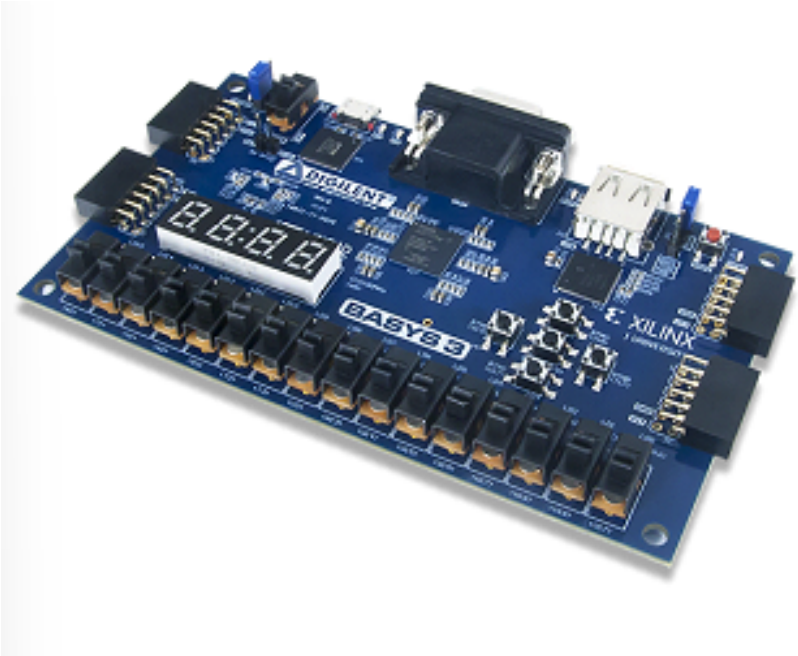

Figura 1: Placuta de dezvoltare FPGA Basys3

3.2 Accelerometrul PMOD ACL2 ADXL362

Modulul accelerometrului PMOD ACL2 este un accelerometru furnizat de Digilent, cu arhitectura pe 3 axe de coordonate. Este un MEMS (Micro Electro-Mechanical System) care foloseste ADXL362 sa sustina o rezolutie de pana la 12 biti pentru fiecare axa a acceleratiei. Pe de alta parte, acest modul poate oferi si un feedback in cazul unor surse explicite de intreruperi, dar dispune si de un mod de consum redus care poate fi activat pentru protectia acestuia in timp. Ca si caracteristici generale, acest modul poate fi programat la o rezolutie dorita de catre utilizator, la frecventa nominala de 100Hz va consuma doar 2 picoAmperi si dispune de niste comenzi specifice (octeti) pentru a comunica cu usurinta prin comenzi de scriere si citire in registrii accelerometrului. [7]

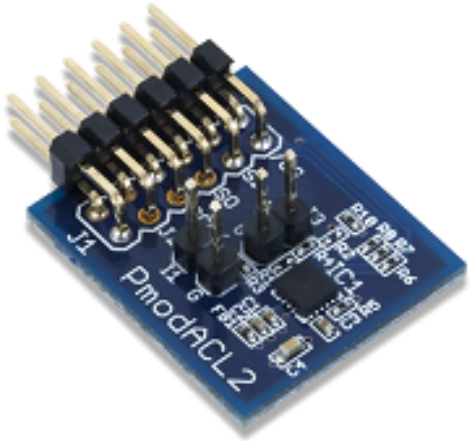

Figura 2: Accelerometrul PMOD ACL2 ADXL362

3.3 Interfata SPI

Interfata SPI (Serial Peripheral Interface) a fost dezvoltata de firma Motorola, o interfata asincrona seriala in care datele sunt transmise impreuna cu un semnal de ceas al carui front crescator/descrescator se utilizeaza de catre receptor pentru esantionarea corecta a liniei seriale. In consecinta, este necesara transmiterea vitezei de transmisie (baudrate) daca avem diferite circuite care pot avea viteze maxime diferite la care pot functiona. Practic, interfata SPI functioneaza in modul duplex, mod in care datele se transmit in ambele directii simultan. Arhitectura interfetei SPI este de tip master-slave, in care poate exista decat un singur master si, eventual, mai multi slave. Se utilizeaza patru semnale: SCLK, MOSI, MISO si SS. Interfata este deci folosita la scara larga in cazul unor sisteme periferice, precum: senzori, convertoare, a unor tipuri de memorii, ceasuri de timp real etc. [4]

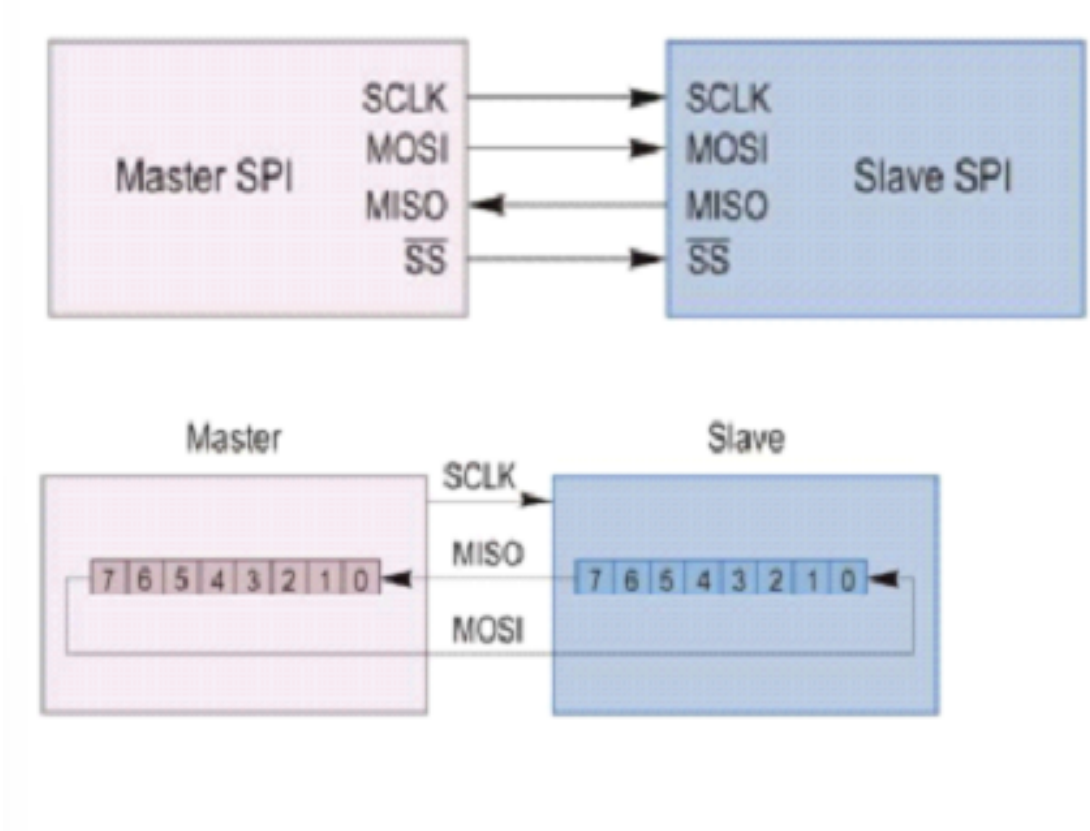

Figura 3: Interfata SPI

3.4 Transmitatorul serial UART

Modulul UART (Universal Asynchronous Receiver Trasmitter) este un modul de comunicatie asincron si serial, in care nu se transmite un semnal de ceas comun, ci receptorul utilizeaza un semnal de ceas generat local pentru esantionarea liniei seriale la momente de timp corespunzatoare vitezei de comunicatie setata in prealabil (baudrate). Practic, cu ajutorul unui bit de START, care preceda fiecare caracter transmis, se realizeaza la fiecare tact resincronizarea ceasului receptorului. Transmitatorul UART dispune asadar de urmatoarele porturi: CLK (pentru linia de comunicatie), TX (pentru transmiterea datelor seriale) si RX (pentru primirea datelor seriale). [4]

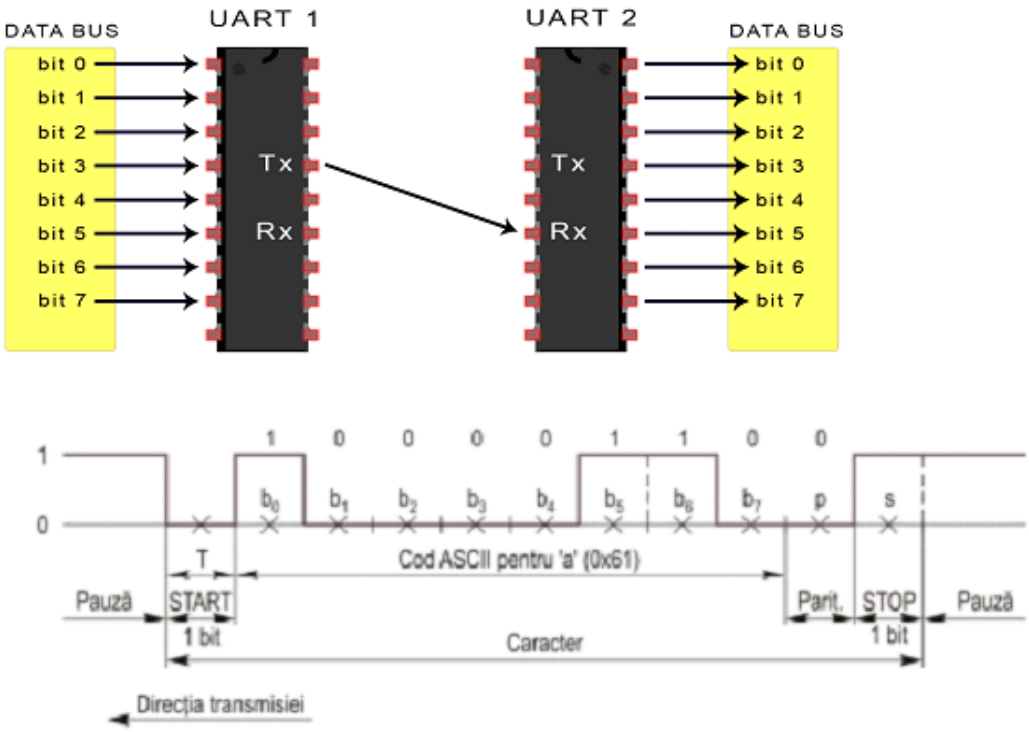

Figura 4: UART [4]

4 Proiectare si implementare

4.1 Solutia aleasa

Pentru a proiecta sistemul descris in capitolele anterioare, singura varianta valabila si usor de implementat pentru receptionarea datelor de la un accelerometru PMOD ACL2 ADXL362 este folosirea unei interfete SPI specifice acestuia. Dupa ce am implementat concret modulul cu FSM-ul atasat si semnalele de control, datele trebuie transmise mai departe, pentru a putea genera graficele mentionate anterior. Astfel, ca solutie eficienta am ales sa implementam si un modul aditional UART care va asigura transferul in timp real a celor 3 axe de coordonate (3 axe a cate 8 biti), pentru a putea fi citite de pe portul COM si trasate mai departe datele pe graficul dorit. Pentru generarea de grafice, am ales sa folosim biblioteca matplotlib care ruleaza pe script-uri de Python, servere de aplicatii web sau diferite kit-uri. Este o biblioteca versatila care lucreaza cu date ce urmeaza sa fie transpuse vizual prin diferite grafice interactive sau statice. [9]

4.2 Modulul SPI

4.2.1 Schema bloc si porturi de intrare

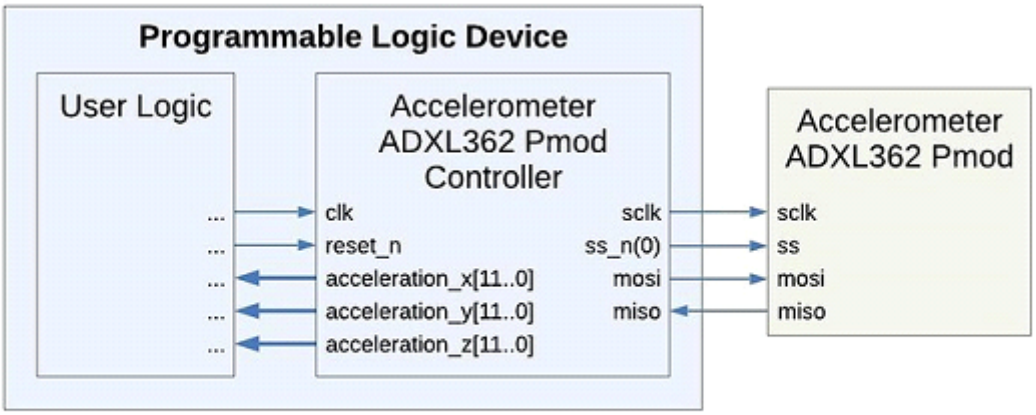

Figura 5: Schema bloc si porturi de intrare

Schema bloc a modulului SPI este una relativ simpla, si practic dispune de 2 mari componente concrete: partea de controller a accelerometrului si partea de logica. Ca setup (partea de generic), avem CLK-ul setat la 100Mhz, data rate de tip 3 (b011) – pentru functionarea la 100MHz si latimea de banda 50Hz, data range "11" – configurat pentru plus/minus 8g sensibilitate. Toate acestea sunt setate conform tabelului din cartea tehnica a accelerometrului [10].

Pe partea de intrari-iesiri, porturile sunt urmatoarele:

- CLK
- MISO (master in - slave out)
- MOSI (master out - slave in)
- SS (slave select)
- SCLK (spi clock)
- Reset

4.2.2 Mod de functionare, arhitectura interna

Acest SPI functioneaza pe un FSM compus din 5 stari: start, pause, configure, read_data si output_result. La pornirea componentei, se intra automat in starea de start, in care isi initializeaza parametrii necesari urmatoarelor stari. In continuare, trece in starea de pauza, in care asteapta 200ns intre tranzactii (prin divizarea ceasului la 5 si folosind un counter). Practic, la fiecare 200ns poate sa se afle in urmatorele stari, succesiv, in functie de parametrul pasat de catre SPI_master.

In prima situatie (param = 0), se incrementeaza parametrul, se furnizeaza bancul de date pentru configurarea registrilor interni (b101100) si se concateneaza la parameter_data range-ul ales, rata de transmisie si niste biti suplimentari.

In a doua situatie (param = 1), se incrementeaza parametrul si se transmite urmatorul banc de date la accelerometru, prin care se initializeaza modul de citire (b101101) precum si un octet specific (b00000010).

In a treia situatie (param = 2), se trece la executia efectiva, si se masoara acceleratia pe cele 3 axe de coordonate.

In starea de configurare, se vor trimite datele de mai sus cand se detecteaza o schimbare a semnalului spi_busy (trecere din 1 in 0). Se vor contoriza numarul de tranzitii, si in fuctie de acesta, se vor realiza urmatoarele operatiuni:

- La inceput, se va seta accelerometrul pe modul continuu de lucru si se va propaga comanda de scriere (b00001010) sau se va trimite adresa parametrului, in functie de spi_ busy.
- Dupa prima tranzitie, se trimit datele parametrului.
- Dupa a doua tranzitie, se opreste modul continuu de functionare, iar SPI se va opri, deci se trece in starea de pauza.

In starea de citire, se vor retine din nou tranzitiile pentru a secventia operatiile:

- La inceput, se seteaza modul continuu de functionare, si se activeaza comanda de citire (b00001011), trimitandu-se si adresa registrului.
- In continuare, la fiecare stare se vor citi, pe rand, cele 2 axe de coordonate (x si y) dpdv. al acceleratiei furnizate de ADXL362.
- La final, se dezactiveaza modul continuu de utilizare, si se va citi si ultima axa (axa z). Se pastreaza rezultatul si se trece in starea de pauza, iar procesul se reia. De mentionat este faptul ca acceleratia este receptionata pe cate 8 biti o data, deci pentru fiecare axa este nevoie de 2 stari.

4.3 Modulul UART

4.3.1 Schema bloc si porturi de intrare

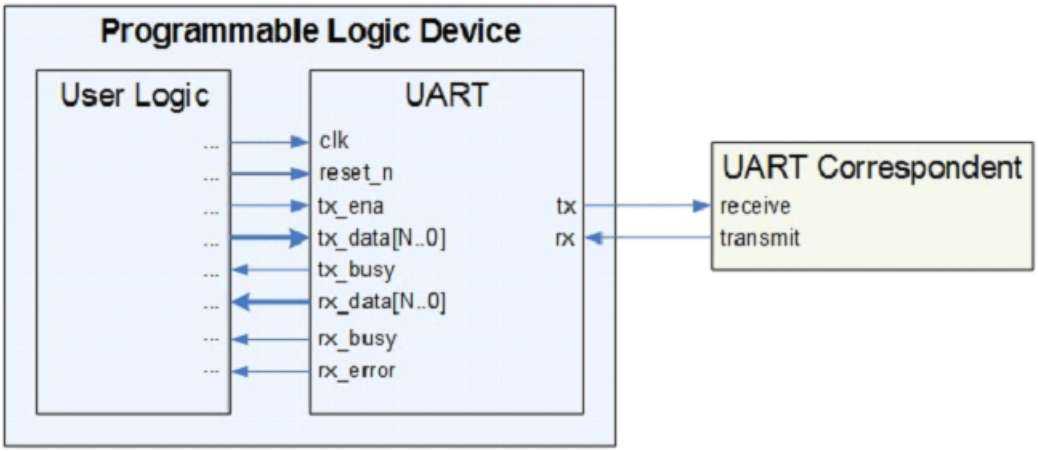

Figura 6: Schema bloc si porturi de intrare

Modulul UART proiectat de noi are urmatoarea configuratie (generice): CLK_FREQ (frecventa clock-ului nominala a sistemului) setata la 50_000_000 Hz, rata de transfer (baud_rate) la 19200 biti/sec (norminala) si marimea pachetului de date receptionat si transmis (d_width) pe 24 de biti – a cate 8 biti pentru fiecare axa de coordonate.

Pe partea de porturi, se dispun urmatoarele:

1. **CLK**
2. **RESET**
3. **tx-ena**
4. **tx si rx**
5. **vectori de 24 de biti cu tx-data si rx-data**

4.3.2 Mod de functionare, arhitectura interna

Acest UART functioneaza pe 2 stari pentru ficare sens: **idle si transmit** pentru TX si separat, pentru RX. In urmatoarea parte, se va explica modul de functionare al UART-ului.

La inceput, se genereaza concurent un semnal de puls de timp baud (baud_pulse). La fiecare perioada de baud, care se calculeaza impartind frecventa clock-ului la baud_rate, se genereaza astfel semnalul de baud. Acest semnal este util pentru a transmite exact datele pe TX.

In paralel, un proces se ocupa de modelarea FSM-ului pentru portul RX. Pentru starea de idle, se numara fiecare bit receptionat intr-un counter si se adauga la buffer-ul RX bitul de start. Se trece in starea de receptie a datelor, pana cand pachetul de date receptionate este complet (24 de biti, in acest caz). Se concateneaza in buffer bitii receptionati, iar la atingerea valorii din counter, se trece in starea idle.

Asemanator cu procesul de mai sus, un alt proces se ocupa de modelarea FSM-ului pentru portul TX. Diferenta este in momentul in care, aflat in starea de idle, UART-ul trimite si niste biti de start si stop pe langa datele concrete. In starea de "transmit" a aceluiasi proces, se va tine cont si de baud_pulse de care am precizat anterior, si joaca un rol important intrucat se va shifta in buffer un bit de '1' la fiecare baud_pulse. La final, se trimit bitii de control, printre care si bitul de stop.

5 Rezultate experimentale

5.1 Instrumente de poiectare utilizate

Limbajul de programare utilizat pentru programarea partii de hardware a fost VHDL, in mediul de dezvoltare Vivado (versiunea 2018.3). Pentru partea de software am folosit Python, cu mediul de dezvoltare PyCharm 2022.2.3.

5.2 Procedura de testare utilizata

5.2.1 Componenta SPI

Penru a testa componenta SPI a prograului nostru am folosit un afisor pe sapte segmente, adaptat pentru placuta FPGA Basys3. Codul pentru afisor se poate gasi **aici**. Aceasta componenta a trecut de testarea noastra cu succes - am testat, pe rand, valori pentru fiecare axa.

5.2.2 Componenta UART

Pentru testarea UART am folosit **codul de python**. In acest cod am importat pachetul "serial" care ne-a permis citirea de date de pe conexiunea seriala stabilita intre placuta si computer. Am folosit un obiect de tip Serial care avea ca atribute: portul de conectare, baudrate-ul, paritatea, bitii de stop, bytesize-ul si un atribut timeout prin care se putea pune o pauza prestabilita intre citirile consecutive de pe conexiunea seriala.

```
ser = serial.Serial(
    port='COM6',
    baudrate=19200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=0)
```

Figura 7: Obiectul de tip Serial

5.2.3 Programul python

Pentru a testa generarea de grafuri am folosit metoda incercarii si optimizarii codului - astfel, ne-a aparut si constanta de TRESHOLD pentru a evita pe cat se poate valorile influentate de catre zgomot. Urmatorul grafic este generat cand placa sta pe loc:

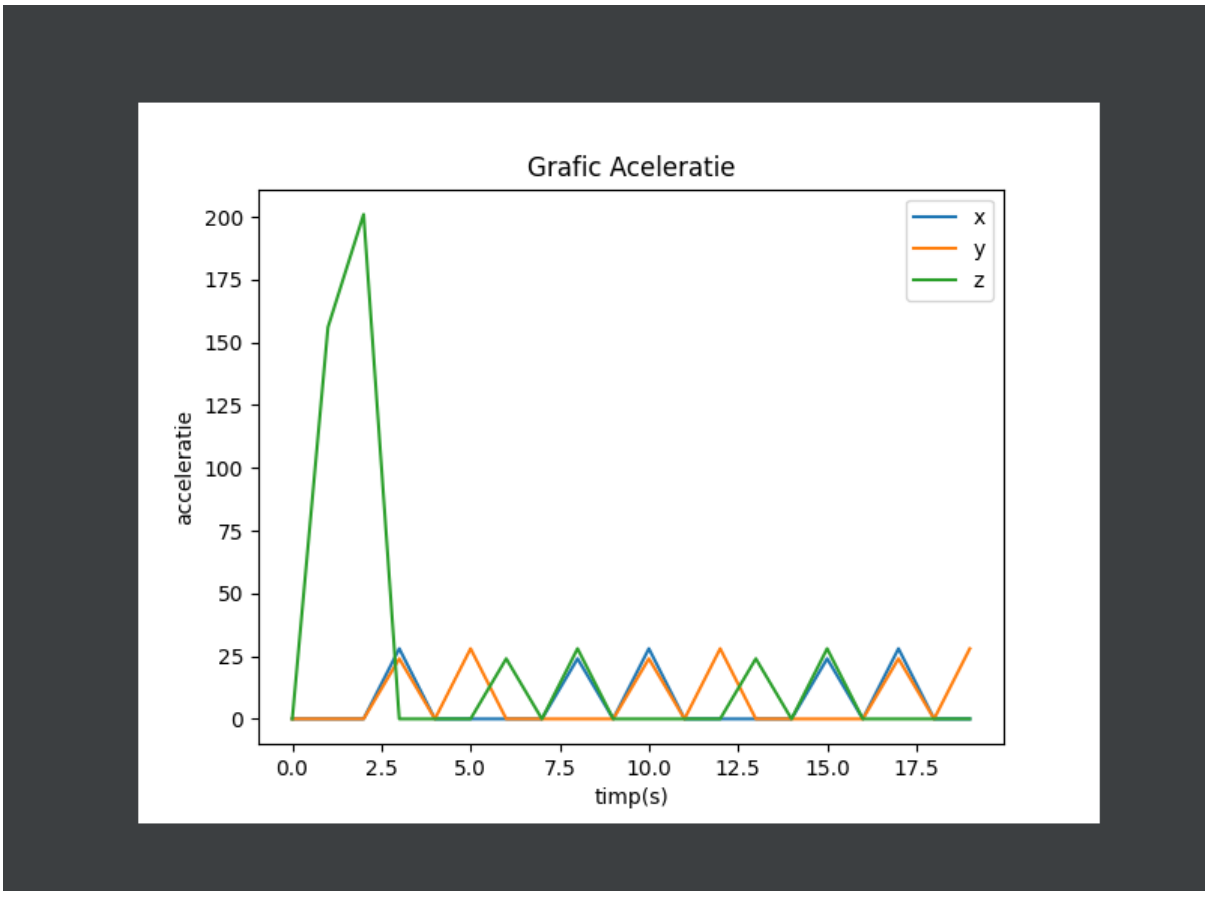

Figura 8: Grafic generat pentru placa stationata

Urmatoarele doua grafice sunt in urma unei miscari usoare a placii si o miscare mai puternica a acesteia.

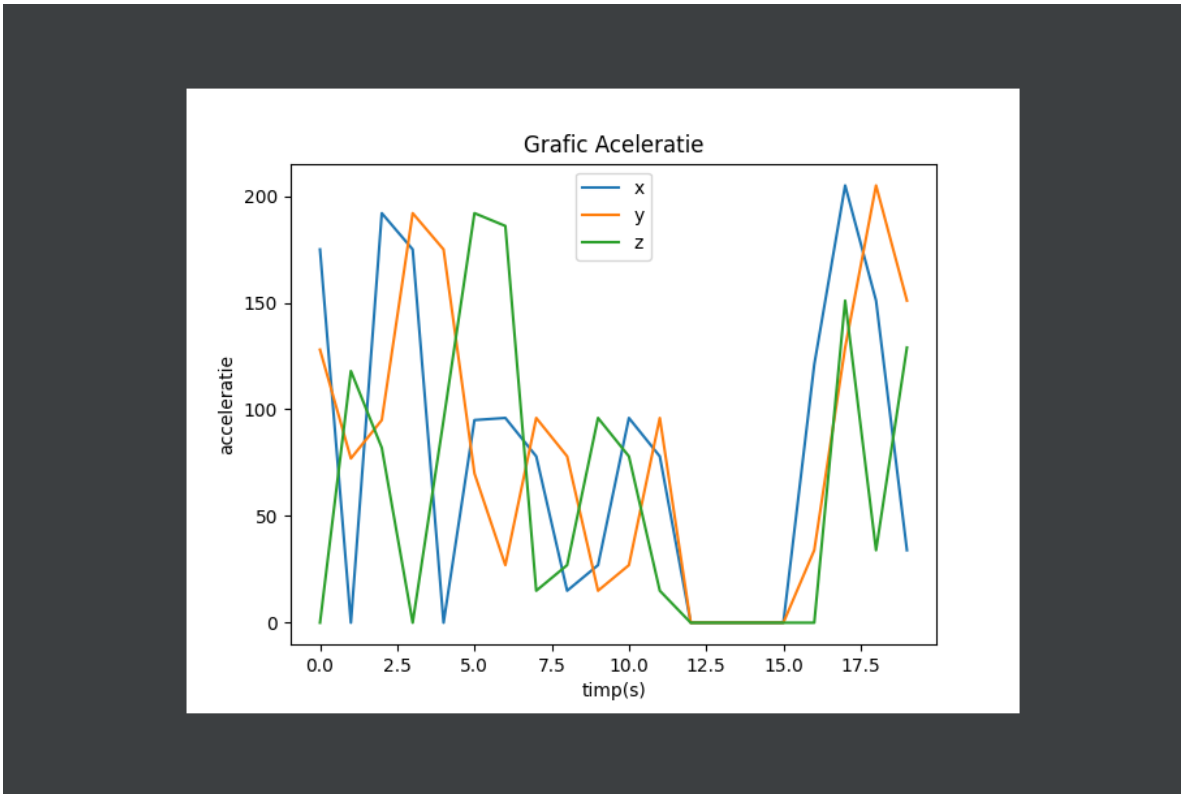

Figura 9: Grafic generat pentru miscare usoara

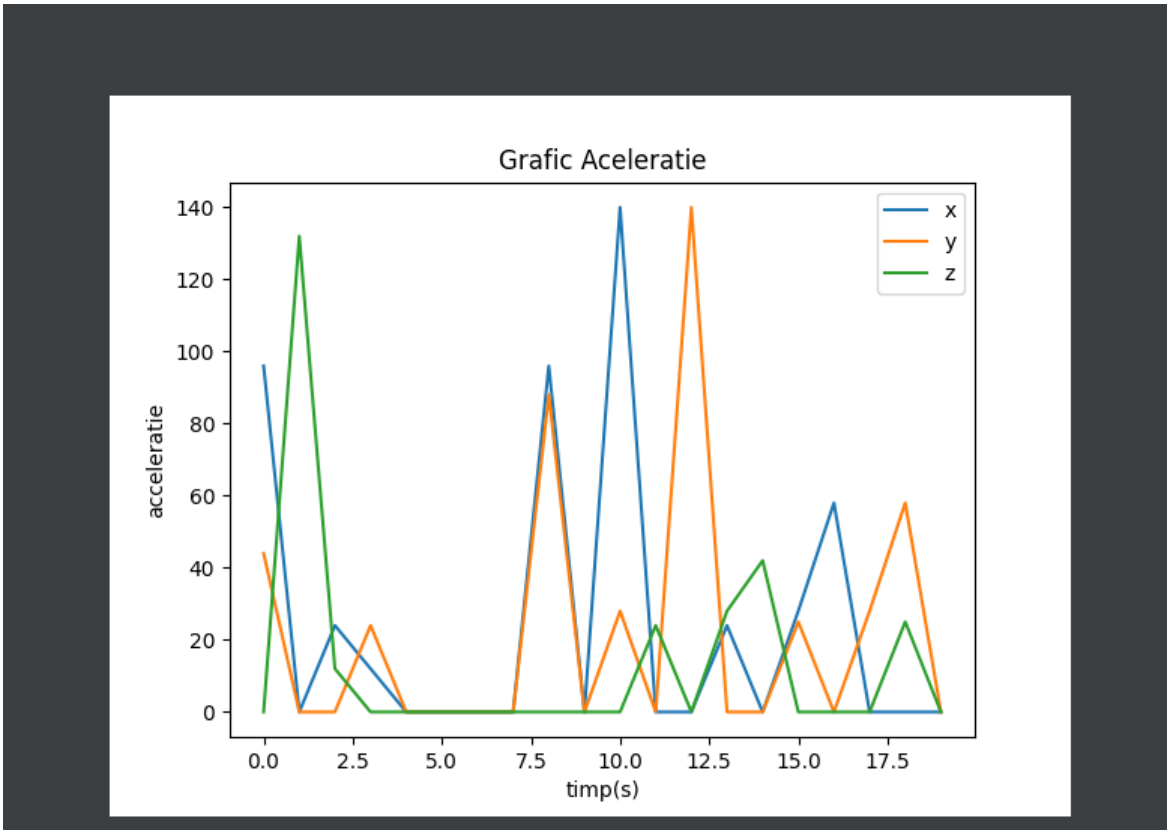

Figura 10: Grafic generat pentru miscare puternica

6 Concluzie

Scopul principal al proiectului a fost generarea de grafice pentru axele accelerometrului. Acesta reuseste deci cu succes sa genereze aceste grafice. Pentru a realiza acest lucru, am combinat doua limbaje de programare: VHDL si Python. Pentru a face legatura dintre placa si senzor, am folosit un SPI, iar pentru a realiza legatura dintre placa si computer am folosit un UART.

Proiectul are un mare avantaj fata de afisajul simplu pe afisorul cu sapte segmente (de care ne-am folosit la testare) - acesta ofera o varianta mult mai lizibila a datelor, cat si progresia vizuala a acestora in functie de timp.

Principala aplicatie care se aplica acestui proiect este acea de testare a senzorului ADXL362. Totodata, acest proiect poate fi folosit si ca material didactic pentru intelegerea accelerometrului si a legaturilor de tip UART si SPI.

In ceea ce priveste dezvoltarea ulterioara a proiectului, senzorul folosit poate fi unul mai precis, sau poate fi folosit un senzor integrat, prezent pe placile Digilent Nexys4. Mai departe, se poate face afisarea si pe afisorul cu sapte segmente sau se pot genera grafice cu evolutie in timp real.

7 Bibliografie

[1] Geeks for Geeks - generare grafuri

[2] Digi-Key - adxl362 control + SPI

[3] Digi-Key - UART

[4] Site Dr. Baruch Zoltan Francisc - Structura sistemelor de calcul - Laborator

[5] Cum functioneaza si la ce servesc accelerometrele

[6] Digilent Reference - Basys3

[7] Digilent Reference - Accelerometrul PMOC ACL2 ADXL362

[8] BASICS OF UART COMMUNICATION

[9] matPlotLib

[10] Analog.com - adxl362

8 Anexa

8.1 Cod python

```
1
2  import time
3  import matplotlib.pyplot as plt
4  import serial
5
6  THRESHOLD = 40
7
8  ser = serial.Serial(
9      port='COM6',
10     baudrate=19200,
11     parity=serial.PARITY_NONE,
```

```
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=0)

print("connected to: " + ser.portstr)
x=[]
x2=[]
x3=[]
x4=[]
count=0
for i in range(20):
    x.append(i)
while count<60:
    for line in ser.read():
        if count % 3 == 1:
            count += 1
            x2 .append( max(int(line)-THRESHOLD,0))
        elif count % 3 == 2:
            count += 1
            x3 .append( max(int(line)-THRESHOLD,0))
        else:
            count += 1
            x4.append(max(int(line)-THRESHOLD,0))
        time.sleep(0.1)

print("S-a terminat citirea datelor")
plt.plot(x,x2, label ="x")
plt.plot(x,x3, label ="y")
plt.plot(x,x4, label="z")
plt.title('Grafic Aceleratie')
plt.xlabel('timp(s)')
plt.ylabel('acceleratie')
print(x2)
print(x3)
print(x4)
plt.legend()
plt.show()
ser.close()
```

8.2 Display sapte segmente Basys3

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


entity displ7seg is
    Port ( Clk : in STD_LOGIC;
```

```
9             Rst : in STD_LOGIC;
10            Data : in STD_LOGIC_VECTOR (15 downto 0);
11            -- date de afisat (cifra 1 din stanga: biti 63..56)
12            An : out STD_LOGIC_VECTOR (3 downto 0);
13            -- semnale pentru anozi (active in 0 logic)
14            Sseg : out STD_LOGIC_VECTOR (7 downto 0));
15            -- semnale pentru segmentele (catozii) cifrei active
16 end displ7seg;
17
18 architecture Behavioral of displ7seg is
19
20 constant CLK_RATE : INTEGER := 100_000_000;
21 -- frecventa semnalului Clk
22 constant CNT_100HZ : INTEGER := 2**20;
23 -- divizor pentru rata de -- reimprospatare de ~100 Hz
24 constant CNT_500MS : INTEGER := CLK_RATE / 2;
25 -- divizor pentru 500 ms
26 signal Count : INTEGER range 0 to CNT_100HZ - 1 := 0;
27 signal CountVect : STD_LOGIC_VECTOR (19 downto 0) := (others => '0');
28 signal LedSel : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
29 --signal Digit1 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
30 --signal Digit2 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
31 --signal Digit3 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
32 --signal Digit4 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
33 signal Digit5 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
34 signal Digit6 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
35 signal Digit7 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
36 signal Digit8 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
37 signal Seg : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
38
39 begin
40  -- Proces pentru divizarea frecventei ceasului
41 div_clk: process (Clk)
42             begin
43                if RISING_EDGE (Clk) then
44                    if (Rst = '1') then
45                        Count <= 0;
46                    elsif (Count = CNT_100HZ - 1) then
47                        Count <= 0;
48                    else
49                        Count <= Count + 1;
50                    end if;
51                end if;
52            end process div_clk;
53
54 CountVect <= CONV_STD_LOGIC_VECTOR (Count, 20);
55 LedSel <= CountVect (19 downto 17);
56
```

```
57   -- Date pentru segmentele fiecarei cifre
58  Digit8 <= "0000" & Data (3 downto 0);
59  Digit7 <= "0000" & Data (7 downto 4);
60  Digit6 <= "0000" & Data (11 downto 8);
61  Digit5 <= "0000" & Data (15 downto 12);
62  --Digit4 <= "0000" & Data (19 downto 16);
63  --Digit3 <= "0000" & Data (23 downto 20);
64  --Digit2 <= "0000" & Data (27 downto 24);
65  --Digit1 <= "0000" & Data (31 downto 28);
66
67   -- Semnal pentru selectarea cifrei active (anozi)
68  An <= "1110" when LedSel = "000" else
69        "1101" when LedSel = "001" else
70        "1011" when LedSel = "010" else
71        "0111" when LedSel = "011" else
72  --      "11101111" when LedSel = "100" else
73  --      "11011111" when LedSel = "101" else
74  --      "10111111" when LedSel = "110" else
75  --      "01111111" when LedSel = "111" else
76        "1111";
77
78   -- Semnal pentru segmentele cifrei active (catozi)
79  Seg <= Digit8 when LedSel = "000" else
80         Digit7 when LedSel = "001" else
81         Digit6 when LedSel = "010" else
82         Digit5 when LedSel = "011" else
83  --       Digit4 when LedSel = "100" else
84  --       Digit3 when LedSel = "101" else
85  --       Digit2 when LedSel = "110" else
86  --       Digit1 when LedSel = "111" else
87         x"FF";
88         process
89         begin
90              case Seg is
91              when "00000000" => Sseg <= "11000000"; -- 0
92              when "00000001" => Sseg <= "11111001"; -- 1
93              when "00000010" => Sseg <= "10100100"; -- 2
94              when "00000011" => Sseg <= "10110000"; -- 3
95              when "00000100" => Sseg <= "10011001"; -- 4
96              when "00000101" => Sseg <= "10010010"; -- 5
97              when "00000110" => Sseg <= "10000010"; -- 6
98              when "00000111" => Sseg <= "11111000"; -- 7
99              when "00001000" => Sseg <= "10000000"; -- 8
100             when "00001001" => Sseg <= "10010000"; -- 9
101             when "00001010" => Sseg <= "10001000"; -- A
102             when "00001011" => Sseg <= "10000011"; -- b
103             when "00001100" => Sseg <= "11000110"; -- C
104             when "00001101" => Sseg <= "10100001"; -- d
```

```
105             when "00001110" => Sseg <= "10000110"; -- E
106             when "00001111" => Sseg <= "10001110"; -- F
107             when others => Sseg <= "11111111";
108         end case;
109         end process;
110 
111 end Behavioral;
```