Logo

Description automatically generated

**Înregistrarea unor date audio de la microfonul plăcii Nexys 4 DDR și transmiterea lor la un dispozitiv mobil**

**Studenți: Pop Ruxandra Maria Indrumator de proiect**

**Zelenszky Bianca Cristi Mocan**

**Grupa 30236**

**Data 16.10.2021**

**Cuprins**

[1. Rezumat 4](#_Toc92625312)

[2. Introducere 5](#_Toc92625313)

[2.1 Tema proiectului 5](#_Toc92625314)

[2.2 Problema și obiectivele principale ale proiectului 5](#_Toc92625315)

[2.3 Terminologia de bază 6](#_Toc92625316)

[2.4 Conținutul lucrării 7](#_Toc92625317)

[3. Fundamentare teoretică 8](#_Toc92625318)

[3.1 Placa de dezvoltare Nexys4 DDR 8](#_Toc92625319)

[3.2 Microfonul plăcii Nexys 4 DDR 9](#_Toc92625320)

[3.3 PDM ( Pulse Density Modulation) 9](#_Toc92625321)

[3.4 Ieșirea Mono Audio 10](#_Toc92625322)

[3.5 PWM (Pulse Width Modulation) 11](#_Toc92625323)

[3.6 Bluetooth PMOD BT2 12](#_Toc92625324)

[3.7 Interfața UART 14](#_Toc92625325)

[3.8 Soluția propusă 15](#_Toc92625326)

[3.9 Scenarii de utilizare 15](#_Toc92625327)

[4. Proiectare și Implementare 17](#_Toc92625328)

[4.1.Arhitectură 17](#_Toc92625329)

[4.2 Debouncer 19](#_Toc92625330)

[4.3 Deserializer 20](#_Toc92625331)

[4.3 Serializer 21](#_Toc92625332)

[4.5 Memory Controller 22](#_Toc92625333)

[4.6 uart\_TX 25](#_Toc92625334)

[4.7 Entitatea principală 28](#_Toc92625335)

[4.8 Aplicația pe telefon 30](#_Toc92625336)

[5.Rezultate experimentale 31](#_Toc92625337)

[5.1 Deserializer 31](#_Toc92625338)

[5.2 Serializer 32](#_Toc92625339)

[5.3 Memory Controller 33](#_Toc92625340)

[5.4 Aplicația pe telefon 35](#_Toc92625341)

[6.Concluzii 36](#_Toc92625342)

[Bibliografie 36](#_Toc92625343)

[Anexa 37](#_Toc92625344)

[A.1.Debouncer 37](#_Toc92625345)

[A.2.Deserializer 38](#_Toc92625346)

[A.3 Serializer 42](#_Toc92625347)

[A.4 Memory Controller 46](#_Toc92625348)

[A.5 uart\_TX 57](#_Toc92625349)

[A.6 Entitatea principală 60](#_Toc92625350)

[A.8 Aplicație pe telefon 68](#_Toc92625351)

[A.9 Fisierul de conștrângeri 74](#_Toc92625352)

[Manual de utilizare 76](#_Toc92625353)

[8.1 Modul de funcționare al aplicației 76](#_Toc92625354)

[8.2 Generarea IP-urilor 77](#_Toc92625355)

[8.2.1 Generarea IP-ului DDR2 77](#_Toc92625356)

[8.2.2 Generarea IP-ului ClockGen 80](#_Toc92625357)

# Rezumat

În vederea realizării proiectului, am încercat să îmbinăm tehnicile software cu cele hardware. Și anume, partea software a fost necesară pentru a realiza aplicația care să recepționeze datele primite de la UART, iar cea de hardware a fost necesară pentru a realiza sistemul de calcul, care să fie capabil să interpreteze datele primite de la microfon și să le stocheze în memoria DDR2 a plăcii.

Astfel, am trecut prin mai multe etape pentru implementarea acestei aplicații. Ne-am folosit de un deserializator pentru citirea datelor de la microfon și generarea unui semnal de tip PDM (capitolul 3.3), după care le-am stocat în memoria plăcii. Pentru testarea faptului că înregistrarea s-a realizat corect, am folosit un serializator pentru convertirea semnalelor în semnale de tip PWM (capitolul 3.5), urmând să fie redate pe ieșirea mono audio a plăcii (capitolul 3.4). Următorul pas a fost folosirea modulului Bluetooth PMOD BT2 (capitolul 3.6) pentru transmiterea datelor, prin intermediul protocolului de transmisie UART (capitolul 3.7), către o aplicație a unui dispozitiv mobil. Aplicația a fost creată cu ajutorul IDE-ului Android Studio, iar datele vor fi transmise cu succes prin urmarea pașilor detaliați din secțiunea Manual de utilizare.

# 2. Introducere

## 2.1 Tema proiectului

Tema proiectului face referire la realizarea unui dispozitiv, capabil în înregistrarea unor date audio de la microfonul plăcii Nexys4 DDR, precum și stocarea lor în memorie. Totodată trebuie să utilizeze și modulul Bluetooth PMOD BT2 pentru a transmite aceste date audio la un dispozitiv mobil, care prin intermediul unei aplicații va reda datele înregistrate. Pentru a realiza o interacțiune între mediul de dezvoltare și utilizator, vom folosi butoanele prezente la nivelul plăcii Nexys4, și descrise în lucrarea de referință [1]. Și anume, în momentul în care utilizatorul apasă butonul BTNU, dispozitivul este pregătit să înregistreze datele primite de la microfonul plăcii (prezentat în [1]). În timpul înregistrării, LED-urile vor sta aprinse, urmând ca atunci când trec cele 5 secunde, să se stingă, simbolizând terminarea înregistrării și stocarea acesteia în memoria plăcii. Sunetul înregistrat va fi citit din memorie și transmis prin intermediul modulului Bluetooth PMOD BT2 la aplicația de pe telefon, care îl va reda.

## 2.2 Problema și obiectivele principale ale proiectului

Problema proiectului este modalitatea de a înregistra datele audio prin intermediul microfonului plăcii Nexys4 DDR, de a transmite acele date la un dispozitiv mobil și de a le putea reda în mod analog.

Obiectivele principale ale proiectului sunt:

* Folosirea microfonului plăcii Nexys4 DDR pentru a stoca datele audio în memoria plăcii.
* Transmiterea datelor audio la un dispozitiv mobil prin intermediul modulului Bluetooth PMOD BT2.
* Realizarea unei aplicații pe dispozitivul mobil pentru redarea datelor audio

## 2.3 Terminologia de bază

FPGA-urile (Field-programmable gate array) sunt circuite integrate care sunt făcute pentru a fi configurate de către proiectanți cu ajutorul HDL-urilor (hardware description language). Acestea conțin blocuri logice programabile care pot fi folosite ca porți logice de bază sau pot fi configurate să realizeze funcții combinaționale complexe, și o serie de interconexiuni reconfigurabile care leagă blocurile între ele. Este de remarcat rolul acestor circuite în dezvoltarea sistemelor embedded datorită ușurinței folosirii lor în simulare rapidă, dezvoltare software deodată cu hardware, precum și partiționări ale sistemului.

HDL (Hardware Description Language) este un limbaj de programare hardware, folosit pentru a descrie structura și comportamentul circuitelor. VHDL este un HDL, standardizat de IEEE (Institute of Electrical and Electronics Engineers) în 1987, și este limbajul în care este realizat acest proiect.

Placa Nexys4 DDR este un circuit digital bazat pe FPGA-urile din familia Artix-7TM de la Xilinx®️. Această placă este dotată cu un FPGA de capacitate mare, memorii externe, porturi de USB, Ethernet și altele, precum și periferice cum ar fi accelerometru, senzor de temperatură, microfon digital MEMS, amplificator de speaker și device-uri de I/E, care fac acest circuit foarte folositor în multe proiecte fără a fi nevoie de alte componente.

Am folosit mediul de proiectare Xilinx Vivado Design Suite 2016.4 pentru realizarea proiectului. Acesta permite utilizarea circuitelor din familia Artix-7TM pentru proiectarea în limbaj VHDL, și include de asemenea ca utilitar un simulator, prin care se poate verifica modul de funcționare a proiectului înainte de a implementa pe placa Nexys 4 DDR. Acest IDE este ușor de folosit, ceea ce crește productivitatea în realizarea obiectivelor propuse.

Pentru realizarea aplicației mobile s-a folosit IDE-ul Android Studio, conceput pentru dispozitivele care rulează pe sistemul de operare Android, și fiind construit pe software-ul IntelliJ IDEA de la JetBrains.

Microfonul circuitului Nexys 4 DDR este unul omnidirecțional MEMS (Micro-Electro Mechanical Systems). MEMS se referă la procesul tehnologic de a crea dispozitive integrate mici sau sisteme care combină componente mecanice sau electronice. Microfonul se află direct pe placă, și convertește semnalul analogic în semnal PDM (Pulse Density Modulation) care este explicat mai pe larg în capitolul 3.2.

Ieșirea Mono Audio a circuitului Nexys 4 DDR se bazează pe un Filtru Trece-Jos de Ordin 4 Sallen-Key Butterworth, și convertește semnalul digital (PDM) stocat în memoria plăcii în semnal analogic (PWM). Acest proces e explicat mai pe larg în capitolele 3.4 și 3.5.

Modulul periferic Bluetooth PMOD (Peripheral module interface) este folosit pentru transmiterea datelor audio la telefonul mobil. Acesta permite o comunicare fără fir securizată și cu putere scăzută, adaptând standardul Pmod, în timp ce folosește o interfață UART (Universal Asynchronous Receiver Transmitter). Mai multe despre UART este menționat în capitolul 3.5

## 2.4 Conținutul lucrării

În capitolul 3, ***Fundamentare teoretică***, va fi prezentată o sinteză a ceea ce am citit despre conceptele și tehnicile care le vom folosi în realizarea proiectului. Totodată aici am introdus și câteva referințe bibliografice, unde sunt descrise mai în detaliu metodele prezentate, sau sunt descrise și alte tehnici decât cele amintite. Vor fi detaliate conceptele legate de microfonul plăcii, de dispozitivul periferic Bluetooth PMOD BT2, și despre realizarea conexiunii dintre placă și dispozitivul mobil.

În capitolul 3, ***Proiectare și implementare***,reprezintă capitolul principal din conținutul lucrării, aici fiind descrise în detaliu fiecare etapă parcursă în realizarea proiectului. Se vor prezenta algoritmi implementați de fiecare modul, precum și schema bloc a proiectului.

În capitolul 5, ***Rezultate Experimentale***, se vor arăta rezultatele simulării, precum și faptul că sistemul a fost implementat cu succes și că rezultatele obținute în urma simulării sunt valide. Vor fi prezentate instrumentele de proiectare utilizate, și rapoartele de implementare sub formă tabelară pentru a fi vizualizate cu ușurință. Totodată, aici, vom atașa și câteva capturi de ecran care să demonstreze corectitudinea funcționării fiecărei componente în parte, precum și a programului complet.

În capitolul 6, ***Concluzii***, va fi prezentat un sumar al raportului proiectului, care va fi descris în câteva paragrafe. Acest sumar va conține observațiile proprii despre rezultatele obținute, ce am învățat din proiect, ce noțiuni noi am dobândit în urma realizării proiectului, ce dificultăți am întâmpinat. Totodată, vom prezenta principalele avantaje și dezavantaje ale soluției alese, vom oferi câteva propuneri pentru dezvoltarea ulterioară a acestuia.

În capitolul 8, ***Manual de utilizare***, vor fi prezentați atât pașii de rulare a programului în IDE-ul Vivado pentru a putea utiliza plăcuța și modulul Bluetooth, cât și pașii utilizați pentru pornirea aplicației dispozitivului mobil și utilizarea ei împreună cu placa Nexys4 DDR.

# 3. Fundamentare teoretică

## 3.1 Placa de dezvoltare Nexys4 DDR

Pentru realizarea proiectului, a fost necesar să folosim placa de dezvoltare Nexys 4 DDR. Placa Nexys4 DDR este o platformă completă, gata de utilizare pentru dezvoltarea de circuite digitale bazate pe cele mai recente FPGA-uri Artix-7 de la Xilinx®. Această placa beneficiază de un FPGA de mare capacitate, memorie externă de dimensiune mare, o colecție de porturi USB, și o varietate de porturi și periferice din care amintim: 16 led-uri, 2 led-uri de tip RGB, 2 display-uri 7-segment, 5 butoane, 16 switch-uri, senzor de temperatură, accelerometru, 4 porturi PMOD. Acestea permit utilizarea plăcii Nexys4 DDR pentru o gamă largă de sisteme fără a avea nevoie de alte resurse.

Mai multe detalii despre resursele și modurile de funcționare ale plăcii se găsesc în [1].

Sistemul pe care urmează să îl implementam, utilizând mediul de proiectare Vivado, va fi încărcat pe placa FPGA Nexys4 DRR, și va folosi următoarele resurse: microfonul plăcii, un buton, 16 led-uri, modul Bluetooth PMOD BT2. Aceste resurse se pot observa în figura 3.1.

A picture containing text, circuit, electronics

Description automatically generated



Led-uri

Butoane

Microfon



Figura 3.1. Placa de dezvoltare Nexys4 DDR

## 3.2 Microfonul plăcii Nexys 4 DDR

Placa Nexys4 DDR pune la dispoziție un microfon omnidirecțional MEMS (Micro Electro-Mechanical Systems). În [1], la secțiunea 15, se descriu următoarele caracteristici ale microfonului:

* Este format dintr-un chip Analog Device ADMP421,
* Primește ca input un semnal analog cu un raport semnal/zgomot de 61 dBA, și o sensibilitate de -26 dBFS.
* Are o frecvență de răspuns plat între 100Hz și 15 kHz.
* Are ca output un semnal digital în format PDM (Pulse Density Modulation)

Graphical user interface, application

Description automatically generated

Figura 3.2. Schema bloc a microfonului

## 3.3 PDM ( Pulse Density Modulation)

Pulse Density Modulation (PDM) este o modificare a semnalului din unul analog în unul digital format din biți de 0 (puls pozitiv) sau 1 (puls negativ). Frecvența de clock este folosită pentru a controla modulatorul [2], ea încadrându-se între 1MHz și 3MHz. Un exemplu de reprezentare a unei forme de undă sinusoidale în semnal PDM este descris în figura 3.3. 1

Icon

Description automatically generated

Figura 3.3.1.*Exemplu de reprezentare a unei forme de undă sinusoidale în semnal PDM*

PDM folosește modularea delta-sigma ca metodă de codificare a semnalelor analogice în semnal digital precum un CAD (Convertor Analog-Digital). Un circuit al acestei metode e prezentat în figura 3.3.2.

Diagram

Description automatically generated

Figura 3.3.2 *Circuit de modulare delta-sigma*

În acest circuit se face diferența dintre intrarea semnalului analog și semnalul PDM a ciclului anterior de ceas, rezultatul fiind intrarea integratorului. Ieșirea acestui integrator intră într-un comparator, care dă o valoare „high” dacă valoarea tensiunii este între (TensiuneMaxima – TensiuneMinimă + 1)/2 și TensiuneMaximă, sau „low” dacă este între TensiuneMinimă sau (TensiuneMaximă – TensiuneMinimă + 1)/2 (circuitul este descris în mai multe detalii în [1], secțiunea 15.1).

## 3.4 Ieșirea Mono Audio

Audio Jack-ul de pe plăcuță (J8) are un filtru trece-jos Sallen-Key Butterworth prin care datele pot fi transmise la ieșirea mono audio. Circuitul unui asemenea filtru se poate vedea în figura 3.2.

Modul de funcționare a filtrului este următorul: intrarea de un bit al filtrului (AUD\_PWM), care e conectat la FPGA prin pinul A11, este de tipul unui semnal PWM (Pulse Width Modulated) sau PDM (Pulse Density Modulated). Semnalul trebuie tras în jos pentru 0 logic și în stânga pentru Mare Impedanță (1 logic), care se realizează cu ajutorul unui rezistor pull-up.

Filtrul trece-jos va converti semnalul digital PWM într-o tensiune analogică pe ieșirea mono audio.

Diagram, schematic

Description automatically generated

Figura 3.4. Circuitul filtrului trece-jos

## 3.5 PWM (Pulse Width Modulation)

O imagine care conține text, antenă

Descriere generată automatUn semnal PWM (pulse-width modulation) e un șir de pulsuri la frecvențe fixe, de lățime diferită. Acest semnal poate să treacă printr-un filtru trece-jos ca să producă o tensiune analogică proporțională cu media lățimii pulsului pe un anumit interval. De exemplu, dacă pulsurile au valori high în medie de 10% a numărului total de pulsuri, atunci un integrator va produce o valoare analogică care e 10% din tensiunea Vdd. Figura 3.3 afișează o formă de undă reprezentată ca un semnal PWM.

Figura 3.5. O formă de undă reprezentată ca și PWM

Filtrul trece-jos ar trebui să fie cu un nivel de magnitudine mai mică decât frecvența PWM, pentru ca semnalul frecvenței PWM să fie filtrat cu un sunet cât mai bun.

## 3.6 Bluetooth PMOD BT2

Proiectul propus pentru implementare propune utilizarea modului Bluetooth PMOD BT2, pentru transmiterea datelor audio la aplicația de pe telefonul Android.

Conform [3], este un modul periferic puternic, care oferă o modalitate ideală de a extinde capacitățile logicii programabile. Comunică cu plăcile Nexys4, utilizând conectori cu 6,8 sau 12 pinii care pot transporta mai multe semnale digitale de control, inclusiv SPI și alte protocoale seriale. Bluetooth PMOD BT2 asigură comunicații securizate, criptate; suportă legături cu dispozitive mobile; prezinta profilul HID pentru realizarea accesoriilor.

Acest cip prezintă o stivă Bluetooth integrată care face posibilă conectarea și comunicarea cu orice dispozitiv prin comenzi UART simple.

A close-up of a computer chip

Description automatically generated with low confidence

Figura 3.6.1 Modulul Bluetooth PMOD BT2

Graphical user interface

Description automatically generated with low confidence

Bluetooth PMOD BT2 are mai multe module disponibile utilizatorului prin setările jumper-ului după cum se poate observa în figura 3.4.3.

Table

Description automatically generated

Figura 3.6.2 Pinii Jumper al

modului Bluetooth PMOD BT2

Figura 3.6.3 Tabel setări ai pinilor Jumper

Pentru a se activa PMOD BT2, acesta trebuie să primească în linia de comandă ‘$$$’, la care va răspunde cu ‘CMD’. Când se află în această stare, modulul este capabil să răspundă la un număr mare de comenzi, astfel proiectantul să poate să-l modifice în funcție de necesitățile programului pe care îl implementează. Pentru a dezactiva modulul, se aplică comanda ‘—<cr>’, la care dispozitivul va răspunde cu END.

Configurarea modulului cu un dispozitiv mobil prezintă mai multe restricții; de exemplu, PMOD BT2 poate fi configurat la distanță doar pentru 60 sec, în afara acestui timp nu va răspunde la nici o comandă aplicată.

După cum se menționează în [3], PMOD BT2 prezintă 6 moduri de operare, care sunt accesate prin comandă “SM,<5,4,3,2,1,0>”.

Modurile în ordine sunt:

* Slave (SM command, 0) - modul implicit în care modulul așteaptă în continuare cererile de conexiune pe care le poate accepta sau respinge
* Master (SM command, 1) - modulul poate iniția conexiuni pe cont propriu, dar nu poate fi localizat de alte dispozitive
* Trigger Master (SM command, 2) - ori de care ori un caracter este primit pe UART, acesta se conectează automat la un slave care fusese împerecheat anterior.
* Auto-connect (Master Mode) (SM command, 3) - la pornire, modulul se conectează automat la un slave care a fost asocial anterior. Conexiunea rămâne activat tot timpul cât modulul este alimentat.
* Auto-connect (DTR Mode) (SM command, 4) - similar cu modul anterior, cu precizarea că putem porni și închide conexiunea cu pinul PIO6.

Auto-connect ANY Mode (SM command, 5): similar modului anterior, cu excepția faptului că de fiecare dată când PIO6 schimbă starea este pornită o nouă procedură de asociere a slave.

## 3.7 Interfața UART

UART - Universal Asynchronous Receiver Transmitter este unul dintre cele mai utilizate protocoale de comunicații de la dispozitiv la dispozitiv, implicând transmiterea și primirea de date seriale, adică datele sunt transmise bit cu bit .

Nicio linie de ceas nu reglementează viteza de transmitere a datelor (fiind vorba de o interfață asincronă), astfel trebuie să setăm plăcuța și dispozitivul mobil să comunice la aceeași viteză. Această viteză este cunoscută sub numele de baud rate, exprimată în biți(stări) pe secundă sau BPS.

Utilizează o rată de transmisie de 115.2 kbps, 8 biți de date, fără paritate și un singur bit de oprire. Pinul de resetare (RST) de pe J1 este active LOW, iar dacă este comutat, dispozitivul va suferi o comutare hard, fiind asemănătoare cu o repornire a dispozitivului. Pinul STATUS de pe J1 reflectă direct starea conexiunii dispozitivului.

În figura 3.5.1 se poate observa conexiunea dintre transmițător și receptor în cazul interfeței UART. Comunicarea funcționează folosind doar 3 fire: ground, Transmiter(Tx) și Receive(Rx).

A picture containing line chart

Description automatically generated

Figura 3.7.1 Conexiunea transmițător-receptor

Informația de sincronizare este încorporata în fluxul de date, iar sincronizarea de la fiecare capăt este realizată printr-un protocol care include biți de pornire și oprire.

Transferul serial asincron conține un număr de mecanisme care asigură transferul robust și fără erori. Acest mecanism conține:

* Rata de transfer: cât de rapid sunt transmise datele pe linia serială.
* Pachetul de date: fiecare bloc de date care urmează să fie transmis este adăugat în acest pachet. Pachetul se formează adăugând biții de sincronizare sau paritate datelor ce urmează să fie transmise.
* Data chunk: dimensiunea datelor pot să varieze între 5 și 9 biți.
* Biții de sincronizare: sunt biți speciali, transferați cu fiecare caracter de date. Sunt biți de start și biți de stop.
* Bitul de paritate: asigură un tip rudimentar de control al erorii. Paritatea poate fi impară sau pară. Paritatea se calculează prin efectuarea unor operații XOR pe toți biții de date.

## 3.8 Soluția propusă

În continuare este descrisă soluția pe care am propus-o pentru realizarea acestui proiect.

Pentru înregistrarea datelor audio, folosim un automat, și în funcție de starea în care se află acel automat, se va folosi un deserializator pentru a converti semnalul PMD al microfonului în semnal digital. Apoi îl memorăm în RAM, iar pe urmă va fi transmis spre serializator.

După care am decis să ne folosim de interfața UART pentru a realiza transferul datelor de pe placa Nexys4 DDR spre dispozitivul mobil. Pentru a avea o parte interactivă, între sistem și utilizator, led-urile plăcii vor fi setate pe 1 pe parcursul înregistrării.

## 3.9 Scenarii de utilizare

Un scenariu de utilizare reprezintă comportamentul sistemului, în timp ce răspunde la o solicitare.

Pentru a descoperi cazurile de utilizare, trebuie mai întâi analizate cerințele proiectului, astfel încât să se determine ceea ce trebuie să facă proiectul respectiv.

În cazul de față principalele cerințe pe care trebuie sa le îndeplinească sistemul pe baza problemei puse sunt:

* înregistrarea unor date audio prin intermediul microfonului plăcii Nexys4 DDR
* folosirea modului Bluetooth PMOD BT2 pentru transmiterea datelor audio la un dispozitiv mobil
* folosirea unei aplicații mobile pentru redarea sunetului înregistrat

Sistemul de față prezintă un singur caz de utilizare, după cum se poate observa, și anume atunci când se apasă butonul BTNU și are loc începerea înregistrării.

Diagram

Description automatically generated

Figura 3.9. Diagrama de use case

După cum se poate observa în figura 3.9 scenariu de utilizare este următorul:

***Use Case:***  **INREGISTRARE DATE AUDIO**

**Actor:** utilizator

**Scenariu de succes:**

1. Utilizatorul apasă butonul BTNU, în acest momentul led-urile de pe placă sunt activate, simbolizând începerea înregistrării.

2. După ce au trecut cele 5 secunde, înregistrarea s-a terminat, iar sunetul va fi stocat în memorie. În acest moment, led-urile vor fi stinse.

3. După stocarea în memorie, datele audio vor fi redate pe un dispozitiv mobil prin intermediul modulului PMOD BT2.

# 4. Proiectare și Implementare

## 4.1.Arhitectură

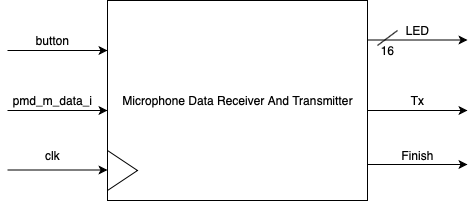
În prima fază, am construit cutia neagră “black box” a sistemului, pentru a avea o privire de ansamblu asupra proiectului. Pe această schemă sunt reprezentate doar semnalele de intrare și de ieșire.

Figura 4.1. Cutia neagră a sistemului

Intrările principale ale sistemului sunt:

* button: este butonul BTNU care simbolizează începerea înregistrării.
* md\_m\_data\_i: reprezintă datele care au fost înregistrate de la microfon.
* Clk: semnalul de ceas.

Ieșirile principale ale sistemului sunt:

* TX: reprezintă bitul care urmează să fie transmis prin Bluetooth.
* Finish: semnal care indică faptul că transmisia datelor s-a terminat.
* LED: reprezintă starea led-urilor, dacă sunt aprinse sau nu, în funcție de caz.

Pe urmă, pornind de la această cutie, am construit schema bloc, identificând astfel componentele de care aveam nevoie pentru a realiza proiectul.

Apoi, următoarea etapă a fost scrierea codului, în Vivado, a fiecărei componente în parte.

Această descompunere ușurează înțelegerea sistemul de către noii proiectanți, precum crește gradul de utilizare, în cazul în care se dorește modificarea anumitor funcționalități.

Graphical user interface, application, Teams

Description automatically generated

Figura 4.1. Schema bloc a circuitului

Diagram

Description automatically generated

Figura 4.2. Schema bloc (generată) a circuitului

## 4.2 Debouncer

**ROL**: Această componentă conține o logică simplă pentru filtrarea oscilaților unui buton. Adică are ca scop sincronizarea cu clock-ul principal al sistemului și eliminarea activărilor multiple care pot apărea la apăsarea unui buton.

A picture containing text

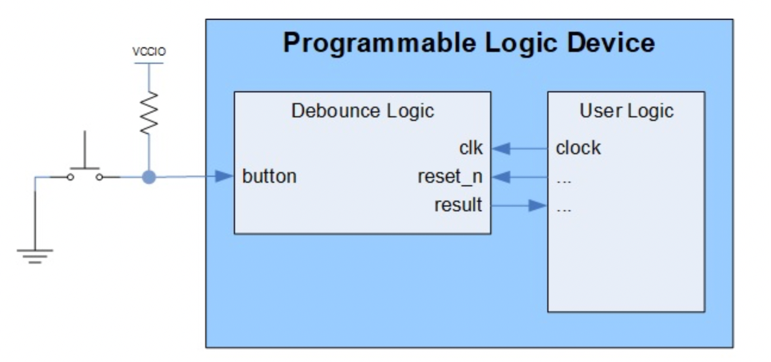
Description automatically generated

Figura 4.2.1.Exemplu implementare a Figura 4.2.2. Schema bloc

Unui Debouncer Debouncer

În cazul sistemul nostru se utilizează un Debouncer, pentru butonul BTNU.   
După cum se poate observa în figura 4.2.2, Debouncer prezintă 2 intrări: butonul și semnalul de ceas; și o ieșire: result .

Figura 4.2.3 descrie modul de funcționare a unui Debouncer. După cum se poate observa în structura unui debouncer se afla 3 bistabile D și un numărător. Se fixează continuu nivelul logic al butonului în FF1 și ulterior în FF2. Aceste două bistabile stochează ultimele două niveluri logice ale butonului. Când aceste două valori rămân identice pentru un timp specificat (aici intră în acțiune counter-ul), atunci FF3 este activat, iar valoarea stabilă este transmisă până la ieșirea rezultatului. Poarta XOR și numărătorul realizează sincronizarea. Dacă valorile FF1 și FF2 diferă pentru un ciclu de ceas, numărătorul este resetat prin intermediul porții XOR, dacă nivelul butonului este neschimbat, atunci poarta XOR eliberează resetarea numărătorului, și acesta începe să numere. Counter-ul continuă să numere până atinge timpul specificat sau până când este resetat.

Diagram, schematic

Description automatically generated

Figura 4.2.3. Structura internă a unui Debouncer

## 4.3 Deserializer

**ROL**: Deserializează datele venite de la microfon, când semnalul de enable este 1, adică când mai există date care trebuie citite de la microfon, și generează semnalul pdm\_m\_clk\_o către microfonul plăcii, care este de tip ADMP421, datele fiind citite pe frontul pozitiv al semnalul m\_clk.

Text

Description automatically generated



Figura 4.3. Schema bloc Deserializer

**Intrări:**

* En - semnalul care permite deserializarea
* done - semnalul care simbolizează că toți cei 16 biți au fost deserializați
* pdm\_m\_data\_i - data PDM primită de la microfon
* clk - semnalul de ceas

**Ieșiri:**

* data\_output - informația deserializată
* pdm\_m\_clk\_o - ieșirea semnalului m\_clk către microfon
* pdm\_lsrel\_o - va fi mereu setat pe ‘0’, datele vor fi citite pe front crescător
* done - semnalul care simbolizează că toți cei 16 biți au fost deserializați

## 4.3 Serializer

Serializatorul de date pornește de la un semnal de date de tip PDM și îl convertește în semnal PWM.

**ROL:** Această componentă va fi folosită pentru a verifica dacă semnalele audio de la microfon au fost înregistrate cu succes, prin redarea acestora prin ieșirea mono audio a plăcii. De asemenea, acest modul generează un semnal de tact având aceeași frecvență cu semnalul de tact trimis microfonului pe deserializator odată ce redarea e pornită (en= 1).

Text

Description automatically generated with low confidence

Figura 4.4 Schema bloc Serializer

**Intrări:**

* PDM\_audio: un semnal pe 16 biți care reprezintă biții semnalului PDM de la microfonul plăcii
* En: semnal de enable
* Clk: semnal de tact

**Ieșiri:**

* PWM\_audio: semnalul care va fi transmis pinului A11 al ieșirii mono audio. Acesta e 0 dacă cel mai semnificativ bit al semnalului PDM\_audio e 0, respectiv Z (înaltă impedanță) dacă e 1.
* Done: semnal care indică faptul că transferul celor 16 biți s-a terminat

## 4.5 Memory Controller

**ROL**: Controllerul memoriei este o componentă care gestionează scrierea și citirea din memoria SRAM a plăcii, realizând operațiile necesare în funcție de limitările și posibilitățile oferite de memoria internă.

Graphical user interface, application, Teams

Description automatically generated

Figura 4.5.1 Schema bloc Memory Controller

**Intrări:**

* Rst: intrare de reset
* RW\_en: semnal care indică dacă va avea loc o citire (1) sau o scriere (0)
* Addr: semnal pe 32 de biți care reprezintă adresa datelor care vor fi citite atunci
* Data\_in: semnal pe 32 biți care reprezintă datele care vor fi scrise din memorie
* CS: semnal de enable
* Byte: indică pe câți biți sunt datele. În cazul nostru va fi setat în entitatea principala ca fiind “0011”, adică 16 biți
* Mem\_DQ\_I: semnal pe 16 biți care reprezintă datele citite din memorie și scrise pe ieșirea Data\_output
* MEM\_WAIT: semnal intern al memoriei PSRAM, nu îl folosim în proiect
* Clk: semnal de ceas

**Ieșiri:**

* Data\_output: semnal de 32 biți pe care se vor scrie datele citite din Mem\_DQ \_I
* rd\_ack: semnal de enable pentru citire
* wr\_ack: semnal de enable pentru scriere
* MEM\_A: semnal de 27 biți care reprezintă adresa din memorie unde se va face scrierea/citirea.
* MEM\_DQ\_O: semnal de 16 biți care reprezintă datele din memorie care se vor suprascrie .
* MEM\_DQ\_T: folosit pentru magistralele bidirecționale, va fi 1 când are loc o citire și 1 când o scriere.
* MEM\_CEN: semnal de enable pentru memorie (activ pe 0 logic)
* MEM\_OEN: semnal de enable pentru output-ul memoriei (activ pe 0 logic)
* MEM\_WEN: semnal de enable pentru scrierea în memorie (activ pe 0 logic)
* MEM\_UB: semnal pentru selecția bitului cel mai semnificativ (activ pe 0 logic)
* MEM\_LB: semnal pentru selecția bitului cel mai nesemnificativ (activ pe 0 logic)
* MEM\_ADV,MEM\_CLK,MEM\_CRE: semnale interne ale memoriei PSRAM, pe care nu le vom folosi în implementarea proiectului.

Acest controller funcționează după următoarea diagrama de stare, care prezinta următoarele 8 stări: **Idle, AssertCen, AssertOen, Waitt, Deassert, SendData, Ack, Done**.

* Idle: este starea inițială a automatului. Dacă CS este 1 se trece în următoarea stare, dacă este 0 se rămâne în starea curentă
* AssertCen: este starea de început. Aici se verifică dacă este nevoie de 2 cicluri de ceas pentru date.
* AssertOen: starea pentru asignarea semnalelor interne ale memoriei de efectuare a operației de citire/scriere, asignarea semnalelor fiind influențate doar de semnalul RW\_en.
* Waitt: este starea unde se așteaptă să treacă 8 ns. Acest lucru se realizează pentru a lăsa memoriei destul timp ca să efectuează citirea sau scrierea.
* Deassert: în acest moment se reactualizează semnalele interne ale memoriei în funcție de lungimea datei și adresei utilizate. Totodată, aici, se realizează fie înregistrarea datei citite într-un registru auxiliar, fie scrierea în memorie a datei.
* SendData: este starea unde se atribuie valoarea semnalului de Data\_output. Dacă datele depășesc 16 biți, este nevoie de 2 cicluri de ceas pentru a realiza operațiile, și astfel se revine în starea AssertCen.
* Ack: reprezintă starea unde se atribuie valorii semnalelor de înștiințare a tipului de operație. Aceste semnale sunt rd\_ack și wr\_ack.
* Done este starea finală a automatului, după care se revine iar în starea Idle.

Icon

Description automatically generated with medium confidence

Figura 4.5.2 Diagrama de stare

## 4.6 uart\_TX

**ROL**: UART\_tx este o componentă care asigură transmisia datelor, bit cu bit, (un singur bit poate fi transmis la un moment dat), spre dispozitivul mobil.

A picture containing text

Description automatically generated

Figura 4.6.1 Schema bloc uart\_TX

**Intrări:**

* Semnalul TxData: semnal pe 8 biți care reprezintă octetul pe 8 biți care trebuie transmis spre dispozitiv
* Rst : semnalul de resetare asincronă
* Start: semnalul care reprezintă începerea transmisiei
* Clk: semnalul de ceas

**Ieșiri:**

* Tx: trebuie dirijat către pinul TX corespunzător al dispozitivului UART extern
* TxRdy: acest semnal trece în 0 logic odată ce operațiunea de transmitere a început și rămâne în această stare până când s-a finalizat transmisia unui octet.
* Done: acest semnal devine 1 logic când toți cei 10 biți au fost eșantionați.

Această componentă funcționează după următoarea diagramă de stare:

Graphical user interface, application

Description automatically generated

Figura 4.6.2 Diagrama stare UART

* Ready: este starea inițială a automatului. Se trece în următoarea stare dacă semnalul Start este 1.
* Load: în această stare se încarcă registrul de deplasare cu octetul care trebuie transmis, completat cu biți de START și STOP.
* Send: aici se incrementează contorul CntBit care reprezintă câți biți din octet au fost transmiși pe linia serială.
* Waitbit: aici se așteaptă trecerea intervalului de timp egal cu durata unui bit, incrementând contorul CntRate. Dacă acest contor ajunge la valoarea T\_BIT-3 care reprezintă numărul ciclurilor de ceas corespunzători duratei unui bit, se trece în următoarea stare. S-a pus T\_BIT-3 pentru a elimina cele 3 cicluri de ceas suplimentare.
* Shift: aici se deplasează la dreapta registrul de deplasare. Dacă nu s-au transmis toți cei 10 biți automatul revine în starea Send. În sens contrar, acesta revine în starea Ready în care se activează semnalul TxRdy, indicând terminarea transmiterii unui octet.

**Principiul transmiteri unui octet este următorul:**

Inițial se transmite bitul de Start, urmat de biții de date, iar mai apoi bitul de STOP. Protocolul se repetată pentru fiecare octet din secvența ce trebuie transmisă. Bitul de start este unul singur, iar biții de stop sunt fie unul, fie doi. Bitul de start este tot timpul indicat de o linie de date inactivă care trece de la 1 la 0, pe când biții/bitul de stop va merge înapoi la starea inactivă ținând linia la nivelul 1 logic.

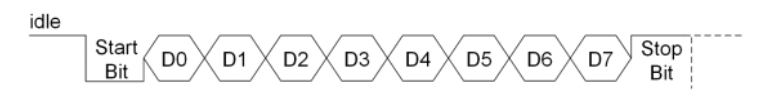


Figura 4.6.2 Diagrama de timp pentru transmiterea serială asincronă

**Octetul care va fi eșantionat are următoarele caracteristici :**

* Baud Rate de 9600 BPS
* 8 biți de date
* Mai întâi se transmite cel mai nesemnificativ (LSB)
* 1 bit de oprire
* fără paritate

## 4.7 Entitatea principală

**ROL:** Entitatea principală este modulul care inițializează toate componentele din aplicație și care le controlează, cu ajutorul unui automat de stare.

Graphical user interface, text, application

Description automatically generated

Figura 4.7.1 Schema bloc Entitate principală

Componentele utilizate sunt: Debouncer, Deserializer, Memory\_Controller, Ram2Ddr și uart\_TX. Mai multe detalii despre ele se găsesc în capitolul 4, în afară de Ram2Ddr, care e un convertor de la o memorie de tip PSRAM la memoria de tip DDR2 a plăcii, având ca intrări ieșirile din Memory\_Controller, și ca ieșiri semnalele importante pentru DDR2. Detalii legate de această componentă se găsesc în [6].

**Intrări:**

* Clk: semnalul de ceas al entității de 100 MHz.
* Rst: semnalul care resetează circuitul.
* Btn: semnalul corespunzător butonului BTNU de pe placă.
* Pdm\_m\_data: semnalul corespunzător datelor PDM primite de la microfon.

**Ieșiri:**

* Leds: cele 16 led-uri ale plăcii.
* Pdm\_m\_clk: semnal de clk corespunzător microfonului plăcii.
* Pdm\_lrsel: semnal de selecție a frontului de la care se citesc datele de la microfon. Acest semnal este 0 (front crescător).
* UartData: semnalul de la pinul RX corespunzător modulului Pmod BT2, care se conectează de pinul TX al dispozitivului Uart extern.
* Pmodbt\_rst: semnalul care resetează modulul Bluetooth. Implicit e 0, iar reset-ul e activ pe 0, de aceea îl setăm manual la valoarea 1 logic.
* Pmodbt\_cts: semnalul Clear To Send setat pe 0 al modulului Bluetooth.
* Ddr2 output: reprezintă 14 semnale de diferite lungimi care fac legătura cu componenta DDR2 din placă.

A picture containing text, electronics

Description automatically generated

Gestionarea componentelor se realizează după următoarea diagramă de stare:

Diagrama are 4 stări, și anume: **Idle**, **Rec**, **Inter** și **Play**.

* Idle este starea inițială unde se regăsește automatul atunci când butonul BTNU nu este apăsat, sau este starea din care se revine după ce s-a efectuat înregistrarea și redarea la ieșirea mono audio.
* Rec este starea de Recording, având loc înregistrarea datelor de la microfon pentru un timp de 5 secunde. Pentru ca automatul să rămână în această stare atâta timp, se folosește semnalul rSamplesCnt, care se incrementează de fiecare dată când se termină procesarea datelor de către deserializator, și se compară cu 5\_SEC\_SAMPLES, care reprezintă numărul de eșantionări ale semnalului pdm\_m\_clk în 5 secunde.

În această stare se setează semnalul de enable al deserializatorului, se actualizează adresa memoriei și se configurează controllerul memoriei pentru a permite scrierea în memoria DDR2. De asemenea, se aprind și ledurile.

* Inter, sau Intermediate, este o stare intermediară între Rec și Play. În această stare, toate componentele

sunt resetate și se așteaptă următorul front crescător Figura 4.7.2 Diagrama de de ceas pentru a trece în următoarea stare. stare

* Play este starea de PlayBack, având loc transmiterea datelor prin intermediul modulului Bluetooth la o aplicație a telefonului mobil. Transmiterea datelor se face pe jumătăți deoarece UART poate transmite doar câte 8 biți deodată. În această stare se setează semnalul de enable al uart\_TX-ului, se actualizează adresa memoriei și se configurează controllerul memoriei pentru a permite citirea din memoria DDR2. După trimiterea datelor, se revine la starea Idle.

Pentru a gestiona componenta Ram2Ddr, am avut nevoie de un semnal de clock de 200 MHz. Cum semnalul maxim de tact al plăcuței este de 100 MHz, a fost necesară folosirea unui generator de frecvență. Acesta reprezintă modulul ClkGen care se folosește de intrarea de 100 MHz a clock-ului pentru a genera o ieșire de 200 MHz.

## 4.8 Aplicația pe telefon

Pentru recepția datelor de la modulul Pmod BT2 ne-am folosit o aplicație de telefon creată în Android Studio. Prin această aplicație, utilizatorul poate să se conecteze prin Bluetooth la dispozitiv și să recepționeze datele prin butonul Connect, să le afișeze în format hexazecimal prin butonul Play, și să se deconecteze de la dispozitiv prin butonul Disconnect.

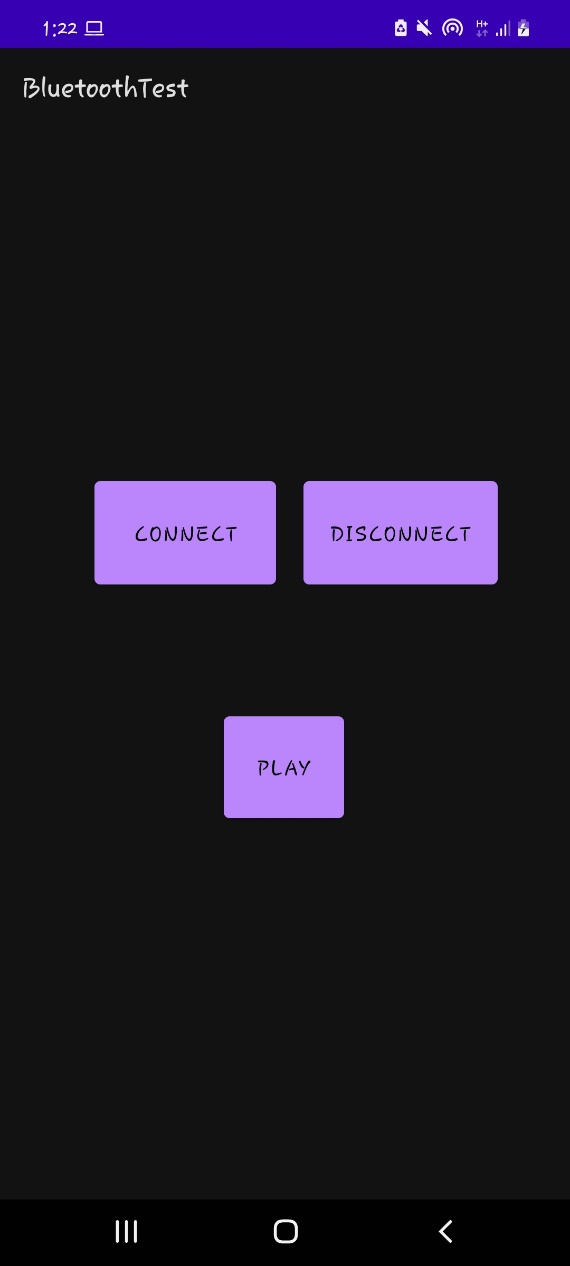


Figura 4.8 Interfața cu utilizatorul

# 5.Rezultate experimentale

## 5.1 Deserializer

Pentru simularea acestei componente am creat un for care merge de la 0 la 15 , și am atribuit la semnalul de intrare, pdm\_m\_data\_i , valori de ‘1’ și ‘0’ ,în funcție de i , și anume:

* Dacă i este număr par atunci intrarea deserializatorului va avea valoarea ‘1’ logic
* Dacă i este număr impar atunci intrarea deserializatorului va avea valoarea ‘0’ logic

Am ales un for de 16 iterații deoarece ieșirea deserializatorului trebuie să fie un vector de 16 biți. Am pus un timp de așteptare de 300 ns între iterații, pentru ca datele să se sincronizeze corect, dacă aș fi pus un timp mai mic sau mai mare de 300 ns, s-ar fi putut ca o valoare să fie citită de 2 ori de către deserializator sau poate niciodată.

În urma logici aplicate, la sfârșitul simulării pe data\_output ar trebui să apăra valoarea hexazecimala “aaaa” care în binar înseamnă ”1010 1010 1010 1010”.

Se poate observa în figura de mai jos că într-adevăr data\_output are valoarea ”aaaa”.

Graphical user interface

Description automatically generated

Figura 5.1. Simulare Deserializer

## 5.2 Serializer

Pentru a testa acest modul, am folosit simulatorul din mediul de proiectare Xilinx Vivado:

A picture containing graphical user interface

Description automatically generated

Figura 5.2. Simulare Serializer

Pentru testare am folosit o intrare PDM pe 16 biți, 0101011011110111, care a generat ieșirea pwm\_audio\_out după cum se observă în figură: biții de 0 de pe pdm\_audio\_in corespund valorii 0 din pwm\_audio\_out, iar biții de 1 din pdm\_audio\_in corespund valori Z de pe pwm\_audio\_out. De asemenea, valoarea semnalului En este 1.

## 5.3 Memory Controller

Pentru a testa funcționalitatea controller-ului de memorie, am simulat trecerea prin cele 8 stări ale automatului finit, atât în modul de scriere Rn\_en=0 (figura 5.3.1), cât și în modul de citire Rn\_en (figura 5.3.2).

După cum se poate observa în figurile de mai jos, sunt parcurse toate cele 8 stări în ordinea corespunzătoare, semnalele de control fiind și ele activate în funcție de starea în care ne aflăm.

Atunci când se realizează scrierea, se poate observa în figura 5.3.1 activarea semnalului MEM\_OEN permanentă, care are loc numai dacă nu se citește din memorie. Semnalele MEM\_CEN și MEM\_WEN sunt 0 numai în cazul când se scrie în memorie (stările Wait și Deassert), iar după ce s-a încheiat operația se observă setarea semnalului wr\_ack ce indică o scriere realizată cu succes.

A picture containing graphical user interface

Description automatically generated

Figura 5.3.1 Simulare Memory Controller pentru scriere

Atunci când se realizează citirea, se poate observa în figura 5.3.2 activarea și dezactivarea semnalelor asemănătoare cu cea pentru scriere. Diferențele sunt date de faptul că MEM\_WEN este în permanent pe 1 logic pentru a nu se realiza scrierea, și rd\_ack devine 1 după ce citirea a fost realizată cu succes. De asemenea, MEM\_DQ\_T primește valoarea ”ffff” la citire.

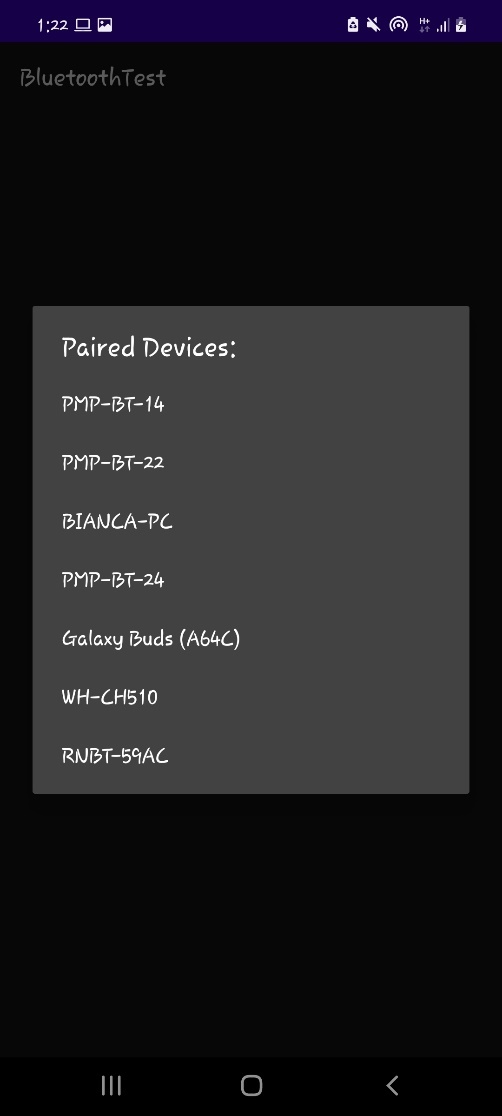
A picture containing diagram

Description automatically generated

Figura 5.3.2 Simulare Memory Controller pentru citire

## 5.4 Aplicația pe telefon

Am testat aplicația apăsând pe butonul Connect și conectându-mă mai întâi la dispozitivul corespunzător modulului Pmod Bt2 (RNBT-59AC) cum se vede în figura 5.4.1. De observat este faptul că s-a făcut pairing cu telefonul dinainte de a începe aplicația.



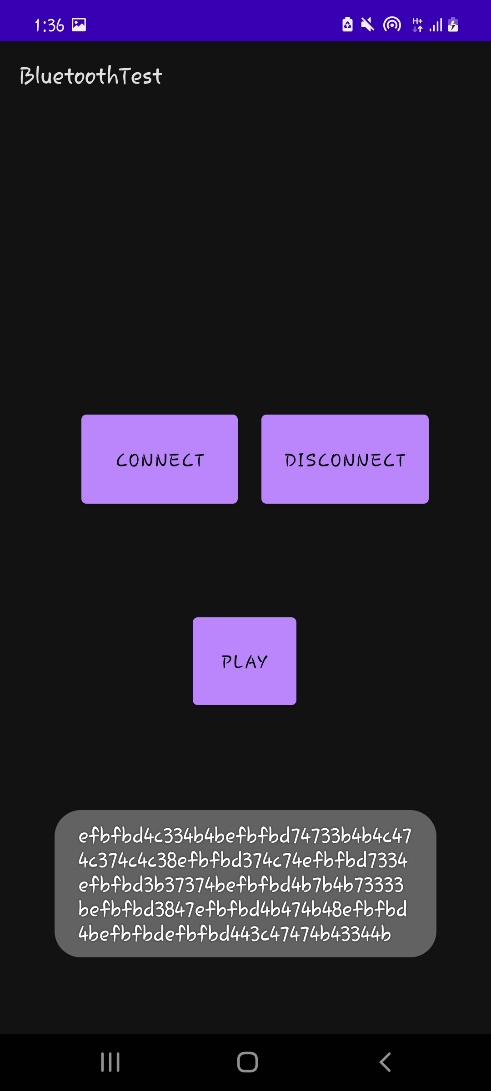


Figura 5.4.1. Alegerea unui dispozitiv Figura 5.4.2 Afișarea valorii întregistrate în valori hexa

După ce conectarea a fost realizată cu succes (LED-ul de pe modulul Bluetooth nu rămâne roșu constant), se poate începe transmiterea datelor prin apăsarea butonului de pe placa Nexys4 DDR pentru a înregistra de la microfon. După ce s-a realizat acest lucru și LED-urile de pe placă s-au stins, apăsând pe butonul Play, se poate vizualiza valorile înregistrate în format hexazecimal, precum în figura 5.4.2.

# 6.Concluzii

Proiectul a constat în proiectarea și implementarea unei soluții în vederea înregistrării a 5 secunde de date audio prin intermediul microfonului plăci Nexys4 DDR. Datele încep să fie înregistrate în momentul apăsări butonului, fiind captate de microfon, deserealizate și salvate în memorie sub forma PDM.

Avantajul proiectului este dat de faptul că e ușor de înțeles și utilizat, iar dezavantajul este dat de faptul că poate înregistra doar timp de 5 secunde.

Proiectul este deschis dezvoltărilor ulterioare, astfel încât aplicația de mobil să poată reda sub forma audio datele, nu sub forma hexazecimala. De asemenea, s-ar putea adăuga anumite opțiuni care să permită modificarea sunetului înregistrat, să poată fi redat cu voce mai joasă sau mai ridicată.

# Bibliografie

[1] “Nexys 4 DDR Reference Manual”, [Online], [https://digilent.com/reference/ programmable-logic/nexys-4-ddr/reference-manual](https://digilent.com/reference/%20programmable-logic/nexys-4-ddr/reference-manual)

[2] “Interfacing PDM digital microphones using STM32 MCUs and MPUs”, [Online], [https://www.st.com/resource/en/application\_note/dm00380469- interfacing-pdm-digital-microphones-using-stm32-mcus-and-mpus- stmicroelectronics.pdf](https://www.st.com/resource/en/application_note/dm00380469-%20interfacing-pdm-digital-microphones-using-stm32-mcus-and-mpus-%20stmicroelectronics.pdf)

[3]”PMOD BT2 Reference Manual” ,[Online],[https://digilent.com/reference/ \_media/reference/pmod/pmodbt2/pmodbt2\_rm.pdf](https://digilent.com/reference/%20_media/reference/pmod/pmodbt2/pmodbt2_rm.pdf)

[4]” Bluetooth Data Module Command Reference & Advanced Information User’s Guide”,[Online],[http://ww1.microchip.com/downloads/en/DeviceDoc/ bluetooth\_cr\_UG- v1.0r.pdf\_ga=2.120838028.523726394.1634331435-1353814548.1634331434](http://ww1.microchip.com/downloads/en/DeviceDoc/%20bluetooth_cr_UG-%20v1.0r.pdf_ga=2.120838028.523726394.1634331435-1353814548.1634331434)

[5] “Debounce Logic Circuit (VHDL)”,[Online Oct 2021] , <https://forum.digikey.com/t/debounce-logic-circuit-vhdl/12573>

[6], “SRAM to DDR Component”, [Online], <https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-4-ddr-sram-to-ddr-component/start>

# Anexa

## A.1.Debouncer

**-----------Modul Principal---------**

library ieee;

use ieee.std\_logic\_1164.all;

entity Dbncr is

generic(

NR\_OF\_CLKS : integer := 4095 );

port(

clk\_ : in std\_logic; -- input clock

button : in std\_logic; -- input signal to be debounced

result : out std\_logic); -- debounced signal

end Dbncr;

architecture logic of Dbncr is

signal flipflops : std\_logic\_vector(1 DOWNTO 0); -- input flip flops

signal counter\_set : std\_logic; -- sync reset to zero

signal count : integer range 0 to NR\_OF\_CLKS-1;

begin

counter\_set <= flipflops(0) xor flipflops(1); --determine when to start/reset counter

process(clk)

begin

if rising\_edge (clk) then

flipflops(0) <= button; --store button value in 1st flipflop

flipflops(1) <= flipflops(0); --store 1st flipflop value in 2nd flipflop

if (counter\_set = '1') then --reset counter because input is changing

count <= 0; --clear the counter

elsif(count = NR\_OF\_CLKS-1) then --stable input time is met

Result<= flipflops(1); --output the stable value

else --stable input time is not yet

count <= count + 1; --increment counter

end if;

end if;

end process;

end logic;

## A.2.Deserializer

**-----------Modul Principal---------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity Deserializer is

generic(

NbBits : integer := 16;

FREQ\_IO\_CLK\_MHZ : integer := 100\_000\_000;

FREQ\_PDM\_CLK\_HZ : integer := 2\_500\_000 );

port(

clk : in std\_logic;

en : in std\_logic; -- semnalul care permite deserializarea

done : out std\_logic; -- semnalizeaza ca 16 biti au fost deserializati

pdm\_m\_data\_i : in std\_logic; -- data primita de la microfon in format PDM

data\_output : out std\_logic\_vector(NbBits - 1 downto 0); --data pe 16 biti care a fost deserializata

pdm\_m\_clk\_o : out std\_logic; -- iesirea semnalului M\_CLK catre microfon

pdm\_lrsel\_o : out std\_logic -- datele sunt citite pe front crescator deci mereu va fi setat pe 0

);

end Deserializer;

architecture Behavioral of Deserializer is

signal cnt\_clk : integer := 0;

signal clk\_int : std\_logic := '0';

signal m\_clk : std\_logic;

signal pdm\_data : std\_logic\_vector((NbBits - 1) downto 0);

signal count\_bits : integer := 0; -- nr de biti citi /maxim 16

signal en\_int : std\_logic;

signal pdm\_clk\_rising\_reg : std\_logic\_vector (2 downto 0);

begin

-- dupa cum am zis mai sus trebuie setat pe '0' pentru a simboliza o citire pe front crescator

pdm\_lrsel\_o <= '0';

-- generarea semnalului de tact cu aceeasi frecventa cu semnalul m\_clk trimis microfonului

PDM\_GEN\_CLK: process(clk)

begin

if rising\_edge(clk) then

if cnt\_clk = ((FREQ\_IO\_CLK\_MHZ/(FREQ\_PDM\_CLK\_HZ\*2))-1) then

cnt\_clk <= 0;

clk\_int <= not clk\_int;

if clk\_int = '0' then

m\_clk <= '1';

end if;

else

cnt\_clk <= cnt\_clk + 1;

m\_clk <= '0';

end if;

end if;

end process PDM\_GEN\_CLK;

-- numaram nr de biti care au fost esantionati

CNT\_BITS: process(clk) begin

if rising\_edge(clk) then

if en = '0' then

count\_bits <= 0;

else

if m\_clk = '1' then

if count\_bits = (NbBits-1) then

count\_bits <= 0;

else

count\_bits <= count\_bits + 1;

end if;

pdm\_data <= pdm\_data(NbBIts-2 downto 0) & pdm\_m\_data\_i;

end if;

end if;

end if;

end process CNT\_BITS;

-- verificam daca au fost esantionati toti cei 16 biti

done <= '1' when m\_clk = '1' and en = '1' and count\_bits = 0 else '0';

data\_output <= pdm\_data when m\_clk = '1' and en = '1' and count\_bits = 0;

pdm\_m\_clk\_o <= clk\_int;

end Behavioral;

**-----------Test Bench---------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Deserializer\_tb is

--  Port ( );

end Deserializer\_tb;

architecture Behavioral of Deserializer\_tb is

--------------------------------------------------------

-- Input signals

--------------------------------------------------------

signal clk : std\_logic := '0';

signal en : std\_logic := '0';

signal done : std\_logic := '0';

signal pdm\_m\_data\_i : std\_logic := '0';

--------------------------------------------------------

-- Output signals

--------------------------------------------------------

signal data\_output : std\_logic\_vector(15 downto 0) := (others => '0');

signal pdm\_m\_clk\_o : std\_logic := '0';

signal pdm\_lrsel\_o : std\_logic := '0';

--------------------------------------------------------

-- CLK value constant

--------------------------------------------------------

constant clk\_period : TIME := 10 ns;

begin

--------------------------------------------------------

-- Port mapping

--------------------------------------------------------

DUT:  entity WORK.Deserializer generic map(

    NbBits => 16,

   FREQ\_IO\_CLK\_HZ => 75000000,

    FREQ\_PDM\_CLK\_HZ => 2500000

)

port map(

clk => clk,

   en => en,

   done => done,

   pdm\_m\_data\_i => pdm\_m\_data\_i,

   data\_output => data\_output,

   pdm\_m\_clk\_o => pdm\_m\_clk\_o,

 pdm\_lrsel\_o => pdm\_lrsel\_o

);

gen\_clock: process

begin

    clk <= '0';

    wait for (CLK\_PERIOD/2);

    clk <= '1';

    wait for (CLK\_PERIOD/2);

end process gen\_clock;

test: process

begin

    for i in 0 to 15 loop

        if i mod 2 = 0 then

            pdm\_m\_data\_i <= '1';

        else

            pdm\_m\_data\_i <= '0';

        end if;

    en <= '1';

    wait for 300 ns;

  end loop;

end process;

end Behavioral;

## A.3 Serializer

**-----------Modul Principal---------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.std\_logic\_arith.ALL;

entity Serializer is

generic(

NbBits : integer := 16;

FREQ\_IO\_CLK\_MHZ : integer := 100\_000\_000;

FREQ\_PDM\_CLK\_HZ : integer := 2\_500\_000 );

port(

pdm\_audio\_in : in std\_logic\_vector(NbBits - 1 downto 0);

clk : in std\_logic;

en : in std\_logic;

pwm\_audio\_out : out std\_logic;

done : out std\_logic);

end Serializer;

architecture Behavioral of Serializer is

signal m\_clk : std\_logic := '0';

signal clk\_count : integer := 0;

signal tmp : std\_logic\_vector(NbBits-1 downto 0) := (others => '0');

signal bit\_count : integer := 0;

signal prev\_bit\_count : integer := 0;

begin

-- generarea semnalului de tact cu aceeasi frecventa cu semnalul m\_clk trimis microfonului

PDM\_CLK\_GEN: process(clk)

begin

if rising\_edge(clk) then

if en = '0' then

clk\_count <= 0;

m\_clk <= '0';

else

if clk\_count = ((FREQ\_IO\_CLK\_MHZ / FREQ\_PDM\_CLK\_HZ) - 1) then

m\_clk <= '1';

clk\_count <= 0;

else

clk\_count <= clk\_count + 1;

m\_clk <= '0';

end if;

end if;

end if;

end process PDM\_CLK\_GEN;

SHIFT: process(clk)

begin

if rising\_edge(clk) then

if en = '0' then

bit\_count <= 0;

else

if m\_clk = '1' then

if bit\_count = (NbBits-1) then

bit\_count <= 0;

else

bit\_count <= bit\_count + 1;

end if;

if bit\_count = 0 then

tmp <= pdm\_audio\_in;

else

tmp <= tmp(NbBits-2 downto 0) & '0';

end if;

end if;

end if;

end if;

end process SHIFT;

-- output

pwm\_audio\_out <= '0' when tmp(NbBits-1) = '0' else 'Z';

done <= '1' when m\_clk = '1' and bit\_count = (NbBits-1) else '0';

end Behavioral;

**-----------Test Bench---------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Serializer\_tb is

--  Port ( );

end Serializer\_tb;

architecture Behavioral of Serializer\_tb is

--------------------------------------------------------

-- Input signals

--------------------------------------------------------

signal clk : std\_logic := '0';

signal pdm\_audio\_in : std\_logic\_vector(15 downto 0) := (others => '0');

signal en : std\_logic := '0';

--------------------------------------------------------

-- Output signals

--------------------------------------------------------

signal done : std\_logic := '0';

signal pwm\_audio\_out : std\_logic := '0';

--------------------------------------------------------

-- CLK value constant

--------------------------------------------------------

constant clk\_period : TIME := 10 ns;

begin

--------------------------------------------------------

-- Port mapping

--------------------------------------------------------

DUT: entity WORK.Serializer

    generic map(FREQ\_IO\_CLK\_HZ => 75,

           FREQ\_PDM\_CLK\_HZ => 2 )

  port map( pdm\_audio\_in => pdm\_audio\_in,

            clk => clk,

            en => en,

            pwm\_audio\_out => pwm\_audio\_out,

            done => done);

gen\_clock: process

begin

    clk <= '0';

    wait for (clk\_period/2);

    clk <= '1';

    wait for (clk\_period/2);

end process gen\_clock;

test: process

begin

    pdm\_audio\_in <= "0101011011110111";

    en <= '1';

    wait for 6500 ns;

    pdm\_audio\_in <= "0001000010010101";

    wait for 6500 ns;

end process test;

end Behavioral;

## A.4 Memory Controller

**-----------Modul Principal---------**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity RamCntrl is

generic(

-- read/write cycle (ns)

RW\_CYCLE\_NS : integer := 100

);

port(

-- Control interface

Clk : in std\_logic; -- 100 MHz system clock

Rst : in std\_logic; -- active high system reset

Byte : in std\_logic\_vector(3 downto 0); -- byte enable

Addr : in std\_logic\_vector(31 downto 0); -- address input

Data : in std\_logic\_vector(31 downto 0); -- data input

Data\_output : out std\_logic\_vector(31 downto 0); -- data output

CS : in std\_logic; -- active high chip select

rd\_ack : out std\_logic; -- read acknowledge flag

wr\_ack : out std\_logic; -- write acknowledge flag

RW\_en : in std\_logic; -- read/write

-- RAM Memory signals

MEM\_DQ\_I : in std\_logic\_vector(15 downto 0); -- Data In

MEM\_WAIT : in std\_logic;

MEM\_A : out std\_logic\_vector(26 downto 0); -- Address

MEM\_DQ\_O : out std\_logic\_vector(15 downto 0); -- Data Out

MEM\_DQ\_T : out std\_logic\_vector(15 downto 0); -- Data Tristate Enable, used for a bidirectional data bus only

MEM\_CEN : out std\_logic; -- Chip Enable

MEM\_OEN : out std\_logic; -- Output Enable

MEM\_WEN : out std\_logic; -- Write Enable

MEM\_UB : out std\_logic; -- Upper Byte

MEM\_LB : out std\_logic; -- Lower Byte

MEM\_ADV : out std\_logic;

MEM\_CLK : out std\_logic;

MEM\_CRE : out std\_logic);

end RamCntrl;

architecture Behavioral of RamCntrl is

-- State machine state names

type type\_state is(Idle, AssertCen, AssertOenWen, Waitt, Deassert, SendData,

Ack, Done);

-- State machine signals

signal state : type\_state := Idle;

-- For a 32-bit access, two cycles are needed

signal need2Cycle : std\_logic := '0';

-- Memory LSB

signal AddrLsb : std\_logic;

-- Internal registerred IP2\_Bus bus

signal DataRdInt : std\_logic\_vector(31 downto 0);

-- Integer for the counter of the rd/wr cycle time

signal CntCycleTime : integer range 0 to 255;

begin

--unused signals

MEM\_ADV <= '0';

MEM\_CRE <= '0';

MEM\_CLK <= '0';

------------------------------------------------------------------------

-- State Machine

------------------------------------------------------------------------

FSM: process(Clk)

begin

if rising\_edge(Clk) then

if(Rst='1') then

state<=idle;

need2Cycle <='0';

else

case state is

when Idle =>

if CS = '1' then

state <= AssertCen;

else

state <= Idle;

end if;

when AssertCen => state <= AssertOenWen;

when AssertOenWen => state <= Waitt;

when Waitt =>

if CntCycleTime = ((RW\_CYCLE\_NS/10) - 2) then

state <= Deassert;

else

state <= Waitt;

end if;

when Deassert => state <= SendData;

when SendData =>

if need2Cycle = '1' then

state <= AssertCen;

else

state <= Ack;

end if;

when Ack => state <= Done;

when Done => state <= Idle;

when others => state <= Idle;

end case;

end if;

end if;

end process FSM;

------------------------------------------------------------------------

-- Counter for the write/read cycle time

------------------------------------------------------------------------

CYCLE\_COUNTER: process(Clk)

begin

if rising\_edge(Clk) then

if state = Waitt then

CntCycleTime <= CntCycleTime + 1;

else

CntCycleTime <= 0;

end if;

end if;

end process CYCLE\_COUNTER;

------------------------------------------------------------------------

-- Assert signals

------------------------------------------------------------------------

MEM\_CEN<='0' when state =AssertOenWen or state =Waitt or state =Deassert else '1';

MEM\_OEN<='0' when ((state =Waitt or state =Deassert )and RW\_en='1') else '1';

MEM\_WEN<='0' when ((state =Waitt or state =Deassert )and RW\_en='0') else '1';

MEM\_A<=(Addr(26 downto 1) & AddrLsb) when (( state = AssertOenWen or state = Waitt or state = Deassert) and Rst='0') else (others=>'0');

------------------------------------------------------------------------

-- Send data to AXI bus

--acknowlegement signals

------------------------------------------------------------------------

Data\_output<=DataRdInt when State=Ack;

rd\_ack<= '1' when State = Ack and need2Cycle = '0' and RW\_en='1' else '0'; --read

wr\_ack<= '1' when State = Ack and need2Cycle = '0' and RW\_en='0' else '0'; --write

------------------------------------------------------------------------

-- When a 32-bit access mode has to be performed, assert the need2Cycle

-- signal

need2Cycle <=not need2Cycle when state =AssertCen and Byte="1111" and Rst='0' else '0';

------------------------------------------------------------------------

-- Data is 32 bits

------------------------------------------------------------------------

data32Bits: process(Clk)

begin

if rising\_edge(Clk) then

if Rst='1' then

AddrLsb<='0';

elsif (state =AssertCen) then

-- '0' lower addres ,'1' higher addres

if Byte="1111" then AddrLsb<=not need2Cycle ;

elsif Byte="1100" or Byte="1000" or Byte="0100" then AddrLsb<='1';

elsif Byte="0011" or Byte="0010" or Byte="0001" then AddrLsb<='0';

end if;

end if;

end if;

end process data32Bits;

------------------------------------------------------------------------

-- Assign Mem\_DQ\_O and Mem\_DQ\_T

------------------------------------------------------------------------

WriteData:process(Clk)

begin

if rising\_edge(Clk) then

if (Rst='1') then

MEM\_DQ\_O<=(others=>'0');

elsif RW\_en='0' and (state =AssertOenWen or state=Waitt or state=Deassert) then

if Byte="1111" then

if need2Cycle ='1' then

MEM\_DQ\_O<=Data(15 downto 0);

else

MEM\_DQ\_O<=Data(31 downto 16); --higher

end if;

elsif Byte="1100" or Byte="1000" or Byte="0100" then MEM\_DQ\_O<=Data(31 downto 16);

elsif Byte="0011" or Byte="0010" or Byte="0001" then MEM\_DQ\_O<=Data(15 downto 0);

else

MEM\_DQ\_O<=(others=>'0');

end if;

end if;

end if;

end process WriteData;

MEM\_DQ\_T <= (others => '1') when RW\_en = '1' else (others => '0');

------------------------------------------------------------------------

-- Read data from the memory

------------------------------------------------------------------------

ReadData:process(Clk)

begin

if rising\_edge(Clk) then

if Rst='1' then

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

elsif state=Deassert then

if Byte="1111" then

if need2Cycle ='1' then

dataRdInt(15 downto 0)<=MEM\_DQ\_I;

else

dataRdInt(31 downto 16)<=MEM\_DQ\_I;

end if;

elsif Byte = "0011" or Byte ="1100" then

dataRdInt(15 downto 0)<=MEM\_DQ\_I;

dataRdInt(31 downto 16)<=MEM\_DQ\_I;

elsif Byte = "0100" or Byte ="0001" then

dataRdInt(7 downto 0)<=MEM\_DQ\_I(7 downto 0);

dataRdInt(15 downto 8)<=MEM\_DQ\_I(7 downto 0);

dataRdInt(23 downto 16)<=MEM\_DQ\_I(7 downto 0);

dataRdInt(31 downto 24)<=MEM\_DQ\_I(7 downto 0);

elsif Byte = "1000" or Byte ="0010" then

dataRdInt(7 downto 0)<=MEM\_DQ\_I(15 downto 8);

dataRdInt(15 downto 8)<=MEM\_DQ\_I(15 downto 8);

dataRdInt(23 downto 16)<=MEM\_DQ\_I(15 downto 8);

dataRdInt(31 downto 24)<=MEM\_DQ\_I(15 downto 8);

end if;

end if;

end if;

end process ReadData;

------------------------------------------------------------------------

-- Assign UB, LB (used in 8-bit write mode)

------------------------------------------------------------------------

asignare\_LB\_UB:process(Clk)

begin

if rising\_edge(Clk) then

if RW\_en='0' and (State=AssertOenWen or State=Waitt or State=Deassert) then

if Byte="1000" or Byte="0010" then

MEM\_UB<='0';

MEM\_LB<='1';

elsif Byte="0100" or Byte="0001" then

MEM\_UB<='1';

MEM\_LB<='0';

else

MEM\_UB<='0';

MEM\_LB<='0';

end if;

else

--cand se citeste

MEM\_UB<='0';

MEM\_LB<='0';

end if;

end if;

end process asignare\_LB\_UB;

end Behavioral;

**-----------Modul Simulare---------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity MemController\_TB is

-- Port ( );

end MemController\_TB;

architecture Behavioral of MemController\_TB is

signal Clk : std\_logic := '0';

signal Rst : std\_logic := '0'; -- active high system reset

signal Byte : std\_logic\_vector(3 downto 0) := (others => '0'); -- byte enable

signal Addr : std\_logic\_vector(31 downto 0) := (others => '0'); -- address input

signal Data : std\_logic\_vector(31 downto 0) := (others => '0'); -- data input

signal Data\_output : std\_logic\_vector(31 downto 0) := (others => '0'); -- data output

signal CS : std\_logic := '0'; -- active high chip select

signal rd\_ack : std\_logic := '0'; -- read acknowledge flag

signal wr\_ack : std\_logic := '0'; -- write acknowledge flag

signal RW\_en : std\_logic := '0'; -- read/write

signal MEM\_DQ\_I : std\_logic\_vector(15 downto 0) := (others => '0'); -- Data In

signal MEM\_WAIT : std\_logic := '0';

signal MEM\_A : std\_logic\_vector(26 downto 0) := (others => '0'); -- Address

signal MEM\_DQ\_O : std\_logic\_vector(15 downto 0) := (others => '0'); -- Data Out

signal MEM\_DQ\_T : std\_logic\_vector(15 downto 0) := (others => '0'); -- Data Tristate Enable, used for a bidirectional data bus only

signal MEM\_CEN : std\_logic := '0'; -- Chip Enable

signal MEM\_OEN : std\_logic := '0'; -- Output Enable

signal MEM\_WEN : std\_logic := '0'; -- Write Enable

signal MEM\_UB : std\_logic := '0'; -- Upper Byte

signal MEM\_LB : std\_logic := '0'; -- Lower Byte

signal MEM\_ADV : std\_logic := '0';

signal MEM\_CLK : std\_logic := '0';

signal MEM\_CRE : std\_logic := '0';

signal need2Cycle : std\_logic := '0';

constant CLK\_PERIOD : TIME := 10 ns;

begin

gen\_clock: process

begin

Clk <= '0';

wait for (CLK\_PERIOD/2);

Clk <= '1';

wait for (CLK\_PERIOD/2);

end process gen\_clock;

DUT : entity WORK.RamCntrl

generic map

(

RW\_CYCLE\_NS => 100

)

port map(

Clk => Clk,

Rst => Rst,

Byte => Byte,

Addr => Addr,

Data => Data,

Data\_output => Data\_output,

CS => CS,

rd\_ack => rd\_ack,

wr\_ack => wr\_ack,

RW\_en => RW\_en,

MEM\_DQ\_I => MEM\_DQ\_I,

MEM\_WAIT => MEM\_WAIT,

MEM\_A => MEM\_A,

MEM\_DQ\_O => MEM\_DQ\_O,

MEM\_DQ\_T => MEM\_DQ\_T,

MEM\_CEN => MEM\_CEN,

MEM\_OEN => MEM\_OEN,

MEM\_WEN => MEM\_WEN,

MEM\_UB => MEM\_UB,

MEM\_LB => MEM\_LB,

MEM\_ADV => MEM\_ADV,

MEM\_CLK => MEM\_CLK,

MEM\_CRE => MEM\_CRE);

test: process

variable cycle\_time\_count : integer := 0;

begin

-- writing

rw\_en <= '0';

-- idle -> idle

Rst <= '1';

wait for CLK\_PERIOD;

-- idle -> idle

Rst <= '0';

CS <= '0';

wait for CLK\_PERIOD;

-- idle -> AssertCen

CS <= '1';

wait for CLK\_PERIOD;

-- AssertCen -> AssertOen

wait for CLK\_PERIOD;

-- AssertOen -> Wait

wait for CLK\_PERIOD;

for i in 0 to 7 loop

cycle\_time\_count := cycle\_time\_count + 1;

wait for CLK\_PERIOD;

end loop;

-- Wait -> Deassert

wait for CLK\_PERIOD;

-- Deassert -> SendData

wait for CLK\_PERIOD;

-- SendData -> AssertCen

need2Cycle <= '1';

wait for CLK\_PERIOD;

-- AssertCen -> AssertOen

wait for CLK\_PERIOD;

-- AssertOen -> Wait

wait for CLK\_PERIOD;

for i in 0 to 7 loop

cycle\_time\_count := cycle\_time\_count + 1;

wait for CLK\_PERIOD;

end loop;

-- Wait -> Deassert

wait for CLK\_PERIOD;

-- Deassert -> SendData

wait for CLK\_PERIOD;

-- SendData -> Ack

need2Cycle <= '0';

wait for CLK\_PERIOD;

-- Ack -> Done

wait for CLK\_PERIOD;

-- Done -> Idle

wait for CLK\_PERIOD;

-- reading

rw\_en <= '1';

-- idle -> idle

Rst <= '1';

wait for CLK\_PERIOD;

-- idle -> idle

Rst <= '0';

CS <= '0';

wait for CLK\_PERIOD;

-- idle -> AssertCen

CS <= '1';

wait for CLK\_PERIOD;

-- AssertCen -> AssertOen

wait for CLK\_PERIOD;

-- AssertOen -> Wait

wait for CLK\_PERIOD;

for i in 0 to 7 loop

cycle\_time\_count := cycle\_time\_count + 1;

wait for CLK\_PERIOD;

end loop;

-- Wait -> Deassert

wait for CLK\_PERIOD;

-- Deassert -> SendData

wait for CLK\_PERIOD;

-- SendData -> AssertCen

need2Cycle <= '1';

wait for CLK\_PERIOD;

-- AssertCen -> AssertOen

wait for CLK\_PERIOD;

-- AssertOen -> Wait

wait for CLK\_PERIOD;

for i in 0 to 7 loop

cycle\_time\_count := cycle\_time\_count + 1;

wait for CLK\_PERIOD;

end loop;

-- Wait -> Deassert

wait for CLK\_PERIOD;

-- Deassert -> SendData

wait for CLK\_PERIOD;

-- SendData -> Ack

need2Cycle <= '0';

wait for CLK\_PERIOD;

-- Ack -> Done

wait for CLK\_PERIOD;

-- Done -> Idle

wait for CLK\_PERIOD;

end process test;

end Behavioral;

## A.5 uart\_TX

**-----------Modul Principal---------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

entity uart\_tx is

generic(

BitRate: integer := 9600

);

Port (

TxData : in std\_logic\_vector(7 downto 0);

Clk : in std\_logic;

Rst : in std\_logic;

Start : in std\_logic;

Tx : out std\_logic;

TxRdy : out std\_logic;

Done : out std\_logic

);

end uart\_tx;

architecture Behavioral of uart\_tx is

constant frecventa\_ceas : integer := 100\_000\_000;

constant T\_BIT : integer := frecventa\_ceas / BitRate;

type stare is (ready, load, send, waitbit, shift);

signal St : stare := ready;

signal CntBit, CntRate : integer := 0;

signal LdData, ShData, TxEn : std\_logic := '0';

signal TSR : std\_logic\_vector(9 downto 0) := (others=>'0');

--atributul keep, folosit la modulul ILA

--pt a evita modificarea numelelor unor semnale dupa sinteza proiectului

attribute keep : STRING;

attribute keep of St : signal is "TRUE";

attribute keep of CntRate : signal is "TRUE";

attribute keep of CntBit : signal is "TRUE";

attribute keep of TSR : signal is "TRUE";

begin

-- Automat de stare pentru unitatea de control a transmitatorului serial

proc\_control: process (Clk)

begin

if RISING\_EDGE (Clk) then

if (Rst = '1') then

St <= ready;

else

case St is

when ready =>

CntRate <= 0;

CntBit <= 0;

if (Start = '1') then

St <= load;

end if;

when load =>

St <= send;

when send =>

CntBit <= CntBit + 1;

St <= waitbit;

when waitbit =>

CntRate <= CntRate + 1;

-- facem T\_BIT-3 in loc de T\_BIT pentru a elimina ele 3 cicluri suplimenatre,

if (CntRate = T\_BIT-3 ) then

CntRate <= 0;

St <= shift;

end if;

when shift =>

if (CntBit = 10) then

St <= ready;

else

St <= send;

end if;

when others =>

St <= ready;

end case;

end if;

end if;

end process proc\_control;

-- Setarea semnalelor de comanda

LdData <= '1' when St = load else '0';

ShData <= '1' when St = shift else '0';

TxEn <= '0' when St = ready or St = load else '1';

-- Setarea semnalelor de iesire

Tx <= TSR(0) when TxEn = '1' else '1';

TxRdy <= '1' when St = ready else '0';

-- Registru deplasare TSR

reg\_TSR: process(Clk)

begin

if RISING\_EDGE(Clk) then

if Rst='1' then

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

-- reg se incarca cu datele de intrare TxData

-- completate cu un bit de STOP(1) pe poz bitului 9 si un bit de START(0) pe poz bitului 0

elsif LdData='1' then

TSR <= '1' & TxData & '0';

-- reg se deplaseaza la dreapta cu o pozitie

elsif ShData='1' then

TSR <= '0' & TSR(9 downto 1);

end if;

end if;

end process;

Done <= '1' when CntBit = 10 else '0';

end Behavioral;

## A.6 Entitatea principală

**-----------Modul Principal---------**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity EntitateaPrincipala is

port (

clk : in std\_logic;

rst : in std\_logic;

-- Periferice

btn : in std\_logic;

leds : out std\_logic\_vector(15 downto 0);

-- UART

uartData : out std\_logic;

pmodbt\_rst : out std\_logic;

pmodbt\_cts : out std\_logic;

-- Semnale microfon PDM

pdm\_m\_clk : out std\_logic; -- Output M\_CLK signal to the microphone

pdm\_m\_data : in std\_logic; -- Input PDM data from the microphone

pdm\_lrsel : out std\_logic; -- Set to '0', therefore data is read on the positive edge

-- Semnale de audio output (folosite la serializator

--pwm\_audio : inout std\_logic; -- Output Audio data to the lowpass filters

--pwm\_en : out std\_logic; -- Output Audio enable

-- Interfata DDR2

ddr2\_addr : out std\_logic\_vector(12 downto 0);

ddr2\_ba : out std\_logic\_vector(2 downto 0);

ddr2\_ras\_n : out std\_logic;

ddr2\_cas\_n : out std\_logic;

ddr2\_we\_n : out std\_logic;

ddr2\_ck\_p : out std\_logic\_vector(0 downto 0);

ddr2\_ck\_n : out std\_logic\_vector(0 downto 0);

ddr2\_cke : out std\_logic\_vector(0 downto 0);

ddr2\_cs\_n : out std\_logic\_vector(0 downto 0);

ddr2\_dm : out std\_logic\_vector(1 downto 0);

ddr2\_odt : out std\_logic\_vector(0 downto 0);

ddr2\_dq : inout std\_logic\_vector(15 downto 0);

ddr2\_dqs\_p : inout std\_logic\_vector(1 downto 0);

ddr2\_dqs\_n : inout std\_logic\_vector(1 downto 0)

);

End EntitateaPrincipala;

architecture Behavioral of EntitateaPrincipala is

-- Amplificator de Clk

component clk\_wiz\_0

port

(

clk\_in1 : in std\_logic; -- Intrare Clk de 100 MHz

clk\_200 : out std\_logic; -- Iesire Clk de 200 MHz

-- Semnale de control

reset : in std\_logic;

locked : out std\_logic

);

end component;

-- Semnalele componentei clk\_wiz\_0

signal reset : std\_logic;

signal locked : std\_logic;

signal clk\_100MHz\_buf : std\_logic;

signal clk\_200MHz\_buf : std\_logic;

-- Constante

constant SECONDS\_TO\_RECORD : integer := 5;

constant PDM\_FREQ\_HZ : integer := 2\_500\_000;

constant SYS\_CLK\_FREQ\_MHZ : integer := 100\_000\_000;

constant NR\_OF\_BITS : integer := 16;

constant SAMPLES\_5\_SEC : integer := (((SECONDS\_TO\_RECORD \* PDM\_FREQ\_HZ) / NR\_OF\_BITS) - 1);

constant RW\_CYCLE\_NS : integer := 1200;

-- Stari

type state\_type is (Idle, Rec, Inter, Play);

type state\_type2 is (RST\_REG, LD\_INIT\_STR, SEND\_CHAR, RDY\_LOW, WAIT\_RDY, WAIT\_BTN, LD\_BTN\_STR);

-- Semnale

signal state : state\_type;

signal uartState : state\_type2;

signal btnu\_int : std\_logic;

signal btn2\_int : std\_logic;

signal btnReg : std\_logic;

signal btnDetect : std\_logic;

signal rnw\_int : std\_logic;

signal addr\_int : std\_logic\_vector(31 downto 0);

signal done\_int : std\_logic;

signal pwm\_audio\_o\_int : std\_logic;

signal clk\_i : std\_logic;

-- Recording

signal pdm\_clk\_rising : std\_logic;

signal en\_des : std\_logic;

signal done\_des : std\_logic;

signal done\_async\_des : std\_logic;

signal data\_des : std\_logic\_vector(15 downto 0) := (others => '0');

signal data\_dess : std\_logic\_vector(31 downto 0) := (others => '0');

signal addr\_rec : std\_logic\_vector(31 downto 0) := (others => '0');

signal rSamplesCnt : integer := 0;

signal done\_des\_dly : std\_logic;

-- Play

signal en\_ser : std\_logic;

signal done\_ser : std\_logic;

signal rd\_ack\_int : std\_logic;

signal data\_ser : std\_logic\_vector(31 downto 0);

signal data\_serr : std\_logic\_vector(15 downto 0);

signal done\_async\_ser : std\_logic;

signal addr\_play : std\_logic\_vector(31 downto 0) := (others => '0');

signal pSamplesCnt : integer := 0;

signal done\_ser\_dly : std\_logic;

-- Memory

signal mem\_a : std\_logic\_vector(26 downto 0);

signal mem\_a\_int : std\_logic\_vector(26 downto 0);

signal mem\_dq\_i : std\_logic\_vector(15 downto 0);

signal mem\_dq\_o : std\_logic\_vector(15 downto 0);

signal mem\_cen : std\_logic;

signal mem\_oen : std\_logic;

signal mem\_wen : std\_logic;

signal mem\_ub : std\_logic;

signal mem\_lb : std\_logic;

signal mem\_adv : std\_logic := '0';

signal mem\_clk : std\_logic := '0';

signal mem\_cre : std\_logic := '0';

-- UART

signal half\_data : std\_logic\_vector(7 downto 0) := (others =>'0');

signal half : std\_logic := '0';

signal start\_uart : std\_logic := '0';

signal TxRdy : std\_logic := '0';

signal data\_To\_Send : std\_logic\_vector(7 downto 0);

begin

-- Configurare Bluetooth

pmodbt\_rst <= '1';

pmodbt\_cts <= '0';

-- Semnalul de reset

reset <= rst or (not locked);

-- Amplificator de Clk

Inst\_ClkGen: clk\_wiz\_0

port map(

clk\_in1 => clk,

clk\_200 => clk\_200MHz\_buf,

reset => rst,

locked => locked);

-- Debouncer

Btnu: entity work.Dbncr

generic map(

NR\_OF\_CLKS => 4095)

port map(

clk => clk\_i,

button => btn,

result => btnu\_int);

-- Deserializator

Deserializer: entity WORK.Deserializer

generic map(

NbBits => NR\_OF\_BITS,

FREQ\_IO\_CLK\_MHZ => SYS\_CLK\_FREQ\_MHZ,

FREQ\_PDM\_CLK\_HZ => PDM\_FREQ\_HZ)

port map(

clk => clk\_i,

en => en\_des,

done => done\_async\_des,

data\_output => data\_des,

pdm\_m\_clk\_o => pdm\_m\_clk,

pdm\_m\_data\_i => pdm\_m\_data,

pdm\_lrsel\_o => pdm\_lrsel);

-- Controler de Memorie

RAM: entity work.RamCntrl

generic map(

RW\_CYCLE\_NS => RW\_CYCLE\_NS)

port map(

Clk => clk\_i,

Rst => rst,

RW\_en => rnw\_int,

Addr => addr\_int,

Data => data\_dess,

Data\_output => data\_ser,

CS => done\_int,

rd\_ack => rd\_ack\_int,

wr\_ack => open,

Byte => "0011",

-- Semnale de memorie RAM

MEM\_DQ\_I => mem\_dq\_o,

MEM\_WAIT => '0',

MEM\_A => mem\_a,

MEM\_DQ\_O => mem\_dq\_i,

MEM\_DQ\_T => open,

MEM\_CEN => mem\_cen,

MEM\_OEN => mem\_oen,

MEM\_WEN => mem\_wen,

MEM\_UB => mem\_ub,

MEM\_LB => mem\_lb,

MEM\_ADV => mem\_adv,

MEM\_CLK => mem\_clk,

MEM\_CRE => mem\_cre);

-- Componenta Ram2DDR

DDR: entity work.Ram2Ddr

port map(

clk\_200MHz\_i => clk\_200MHz\_buf,

rst\_i => rst,

ui\_clk\_o => clk\_i,

-- Interfata RAM

ram\_a => mem\_a,

ram\_dq\_i => mem\_dq\_i,

ram\_dq\_o => mem\_dq\_o,

ram\_cen => mem\_cen,

ram\_oen => mem\_oen,

ram\_wen => mem\_wen,

ram\_ub => mem\_ub,

ram\_lb => mem\_lb,

-- Interfata DDR2

ddr2\_addr => ddr2\_addr,

ddr2\_ba => ddr2\_ba,

ddr2\_ras\_n => ddr2\_ras\_n,

ddr2\_cas\_n => ddr2\_cas\_n,

ddr2\_we\_n => ddr2\_we\_n,

ddr2\_ck\_p => ddr2\_ck\_p,

ddr2\_ck\_n => ddr2\_ck\_n,

ddr2\_cke => ddr2\_cke,

ddr2\_cs\_n => ddr2\_cs\_n,

ddr2\_dm => ddr2\_dm,

ddr2\_odt => ddr2\_odt,

ddr2\_dq => ddr2\_dq,

ddr2\_dqs\_p => ddr2\_dqs\_p,

ddr2\_dqs\_n => ddr2\_dqs\_n);

done\_int <= done\_des when state = Rec else

done\_ser when state = Play else '0';

-- Genereaza semnalele de Done pentru deserializator si serializator

process(clk\_i)

begin

if rising\_edge(clk\_i) then

if rd\_ack\_int = '1' then

data\_serr <= data\_ser(15 downto 0);

end if;

done\_des <= done\_async\_des;

data\_dess <= x"0000" & data\_des;

done\_ser <= done\_async\_ser;

end if;

end process;

-- Serializator

-- Serializer: entity WORK.Serializer

-- generic map(

-- NbBits => NR\_OF\_BITS,

-- FREQ\_IO\_CLK\_MHZ => SYS\_CLK\_FREQ\_MHZ,

-- FREQ\_PDM\_CLK\_HZ => PDM\_FREQ\_HZ)

-- port map(

-- pdm\_audio\_in => data\_serr,

-- clk => clk\_i,

-- en => en\_ser,

-- pwm\_audio\_out => pwm\_audio,

-- done => done\_async\_ser);

-- Numara esantioanele si le stocheaza in memorie

process(clk\_i)

begin

if rising\_edge(clk\_i) then

if state = Rec then

if done\_des = '1' then

rSamplesCnt <= rSamplesCnt + 1;

end if;

if done\_des\_dly = '1' then

addr\_rec <= addr\_rec + "10";

end if;

elsif state = Play then

if done\_ser = '1' then

half <= not half;

end if;

if done\_ser\_dly = '1' then

if half='0' then

pSamplesCnt <= pSamplesCnt + 1;

addr\_play <= addr\_play + "10";

end if;

end if;

else

pSamplesCnt <= 0;

addr\_play <= (others => '0');

rSamplesCnt <= 0;

addr\_rec <= (others => '0');

end if;

done\_des\_dly <= done\_des;

done\_ser\_dly <= done\_ser;

end if;

end process;

rnw\_int <= '1' when state = Play else '0';

en\_ser <= '1' when state = Play else '0';

en\_des <= '1' when state = Rec else '0';

addr\_int <= addr\_rec when state = Rec else

addr\_play when state = Play else

(others => '0');

data\_to\_send <= data\_ser(15 downto 8) when (state=Play and half = '0') else

data\_ser(7 downto 0) when (state=Play and half = '1') else

(others => '0');

start\_uart <= '0' when state = Idle or (state = Play and pSamplesCnt = SAMPLES\_5\_SEC ) else

TxRdy when (state = Play and pSamplesCnt /= SAMPLES\_5\_SEC);

--pwm\_en <= '1';

leds <= (others => '1') when state = Rec else

"1000000000000001" when state = Play else

(others => '0');

-- FSM

NEXT\_STATE\_DECODE: process(clk\_i)

begin

if rising\_edge(clk\_i) then

if rst = '1' then

state <= Idle;

else

case (state) is

when Idle =>

if btnu\_int = '1' then

state <= Rec;

end if;

when Rec =>

if rSamplesCnt = SAMPLES\_5\_SEC then

state <= Inter;

end if;

when Inter =>

state <= Play;

when Play =>

if btnu\_int = '1' then

state <= Idle;

elsif pSamplesCnt = SAMPLES\_5\_SEC then

state <= Idle;

end if;

when others =>

state <= Idle;

end case;

end if;

end if;

end process;

-- UART

uart\_TX: entity WORK.UART\_tx port map (TxData => data\_to\_send, Clk => clk\_i ,Start => start\_uart , Rst => rst, Tx=>uartData, TxRdy => TxRdy, Done => done\_async\_ser);

end Behavioral;

## A.8 Aplicație pe telefon

package com.example.bluetoothtest;

import androidx.appcompat.app.AlertDialog;

import androidx.appcompat.app.AppCompatActivity;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.bluetooth.BluetoothSocket;

import android.content.DialogInterface;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.Toast;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.io.UnsupportedEncodingException;

import java.math.BigInteger;

import java.nio.charset.StandardCharsets;

import java.util.ArrayList;

import java.util.Set;

import java.util.UUID;

public class MainActivity extends AppCompatActivity {

public static final String EXTRA\_MESSAGE = "com.example.bluetoothtest.MESSAGE";

BluetoothAdapter bluetoothAdapter;

private BluetoothSocket mBluetoothSocket;

private BluetoothDevice mBluetoothDevice;

private OutputStream outputStream;

private InputStream inputStream;

private UUID newUUID = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb"); // uuid for serial port pmod bt2

private int readBufferPosition = 0;

private boolean stop = false;

private String dd;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

}

public void connectToBluetooth(View view){

if (!bluetoothAdapter.isEnabled()){

Toast.makeText(this, "Turning on Bluetooth", Toast.LENGTH\_SHORT).show();

Intent intent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(intent, 0); // request to connect has code 0

while(!bluetoothAdapter.isEnabled()) {

// wait till it becomes enabled

}

Set<BluetoothDevice> devices = bluetoothAdapter.getBondedDevices();

ArrayList<String> msgs = new ArrayList<>();

ArrayList<String> addresses = new ArrayList<>();

for (BluetoothDevice d : devices) {

msgs.add(d.getName());

addresses.add(d.getAddress());

}

CharSequence[] csAddresses = addresses.toArray(new CharSequence[msgs.size()]);

CharSequence[] csNames = msgs.toArray(new CharSequence[msgs.size()]);

AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);

builder.setTitle("Paired Devices:")

.setItems(csNames, new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialogInterface, int i) {

connect(csAddresses[i].toString(), csNames[i].toString());

}

});

AlertDialog alert = builder.create();

alert.show();

}

else {

Set<BluetoothDevice> devices = bluetoothAdapter.getBondedDevices();

ArrayList<String> msgs = new ArrayList<>();

ArrayList<String> addresses = new ArrayList<>();

for (BluetoothDevice d : devices) {

msgs.add(d.getName());

addresses.add(d.getAddress());

}

CharSequence[] csAddresses = addresses.toArray(new CharSequence[msgs.size()]);

CharSequence[] csNames = msgs.toArray(new CharSequence[msgs.size()]);

AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);

builder.setTitle("Paired Devices:")

.setItems(csNames, new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialogInterface, int i) {

connect(csAddresses[i].toString(), csNames[i].toString());

}

});

AlertDialog alert = builder.create();

alert.show();

}

}

public void disconnectFromBluetooth(View view){

if (bluetoothAdapter.isEnabled()){

Toast.makeText(this, "Turning off Bluetooth", Toast.LENGTH\_SHORT).show();

closeBT();

bluetoothAdapter.disable();

}

else {

Toast.makeText(this, "Bluetooth is already disconnected!", Toast.LENGTH\_SHORT).show();

}

}

public void connect(String address, String name){

if(!bluetoothAdapter.isEnabled()){

Toast.makeText(this, "Bluetooth is disconnected! Connect first", Toast.LENGTH\_SHORT).show();

}

else {

Toast.makeText(this, "Connecting device " + name + " with address " + address + "...", Toast.LENGTH\_SHORT).show();

mBluetoothDevice = bluetoothAdapter.getRemoteDevice(address);

try {

mBluetoothSocket = mBluetoothDevice.createRfcommSocketToServiceRecord(newUUID);

mBluetoothSocket.connect();

Toast.makeText(MainActivity.this, "Device connected!", Toast.LENGTH\_SHORT).show();

inputStream = mBluetoothSocket.getInputStream();

outputStream = mBluetoothSocket.getOutputStream();

receiveData(inputStream, outputStream);

} catch (IOException ioE) {

Toast.makeText(MainActivity.this, "Can't connect", Toast.LENGTH\_SHORT).show();

try {

if (mBluetoothSocket != null)

mBluetoothSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

public void receiveData(InputStream inputStream, OutputStream outputStream){

Toast.makeText(MainActivity.this, "Trying to receive data...", Toast.LENGTH\_SHORT).show();

byte[] readBuffer = new byte[1024]; // is this related to memory?

StringBuilder data = new StringBuilder();

Thread t = new Thread(new Runnable() {

@Override

public void run() {

while (!Thread.currentThread().isInterrupted() && !stop) {

try {

int bytes = inputStream.available();

if (bytes > 0) {

byte[] bytesArray = new byte[bytes];

inputStream.read(bytesArray);

int finalBytes1 = bytes;

for (int i = 0; i < bytes; i++) {

byte b = bytesArray[i];

if (i + 1 == bytes){

byte[] encodedB = new byte[readBufferPosition];

System.arraycopy(readBuffer, 0, encodedB, 0, encodedB.length);

String d = new String(encodedB, "US-ASCII");

data.append(d);

readBufferPosition = 0;

}

else {

readBuffer[readBufferPosition++] = b;

}

}

dd=data.toString();

}

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

Toast.makeText(MainActivity.this, "UnsupportedEncodingException", Toast.LENGTH\_LONG).show();

closeBT();

} catch (IOException e) {

e.printStackTrace();

Toast.makeText(MainActivity.this, "IOException", Toast.LENGTH\_LONG).show();

closeBT();

}

}

}

});

t.start();

}

public String toHex(String arg) {

String s = "";

try {

s = String.format("%x", new BigInteger(1, arg.getBytes("UTF-8")));

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

return s;

}

void closeBT(){

stop = true;

try {

if(outputStream != null)

outputStream.close();

if(inputStream != null)

inputStream.close();

if(mBluetoothSocket != null)

mBluetoothSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

public void play(View view){

File path = getApplicationContext().getFilesDir();

try {

FileOutputStream writer = new FileOutputStream(new File(path, "coni.txt"));

writer.write(dd.getBytes(StandardCharsets.UTF\_8));

writer.close();

Toast.makeText(MainActivity.this, toHex(dd), Toast.LENGTH\_LONG).show();

} catch (Exception e){

e.printStackTrace();

}

}

}

## A.9 Fisierul de conștrângeri

set\_property BITSTREAM.GENERAL.COMPRESS TRUE [current\_design]

set\_property BITSTREAM.CONFIG.SPI\_BUSWIDTH 4 [current\_design]

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { leds[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { leds[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { leds[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { leds[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { leds[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { leds[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { leds[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { leds[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { leds[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { leds[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { leds[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { leds[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { leds[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { leds[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { leds[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { leds[15] }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

##Buttons

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { btn }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { btn2 }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { rst }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd

##Pmod Header JA

set\_property -dict { PACKAGE\_PIN D18 IOSTANDARD LVCMOS33 } [get\_ports { uartData }]; #IO\_L21N\_T3\_DQS\_A18\_15 Sch=ja[2]

set\_property -dict { PACKAGE\_PIN G17 IOSTANDARD LVCMOS33 } [get\_ports { pmodbt\_cts }]; #IO\_L18N\_T2\_A23\_15 Sch=ja[4]

set\_property -dict { PACKAGE\_PIN E17 IOSTANDARD LVCMOS33 } [get\_ports { pmodbt\_rst }]; #IO\_L16P\_T2\_A28\_15 Sch=ja[8]

##Omnidirectional Microphone

set\_property -dict { PACKAGE\_PIN J5 IOSTANDARD LVCMOS33 } [get\_ports { pdm\_m\_clk }]; #IO\_25\_35 Sch=m\_clk

set\_property -dict { PACKAGE\_PIN H5 IOSTANDARD LVCMOS33 } [get\_ports { pdm\_m\_data }]; #IO\_L24N\_T3\_35 Sch=m\_data

set\_property -dict { PACKAGE\_PIN F5 IOSTANDARD LVCMOS33 } [get\_ports { pdm\_lrsel }]; #IO\_0\_35 Sch=m\_lrsel

##PWM Audio Amplifier

set\_property -dict { PACKAGE\_PIN A11 IOSTANDARD LVCMOS33 } [get\_ports { pwm\_audio }]; #IO\_L4N\_T0\_15 Sch=aud\_pwm

set\_property -dict { PACKAGE\_PIN D12 IOSTANDARD LVCMOS33 } [get\_ports { pwm\_en }]; #IO\_L6P\_T0\_15 Sch=aud\_sd

#constraint used to replace a ddr.xdc constrant which is not working for some unknown reason

#"set\_false\_path -through [get\_nets -hier -filter {NAME =~ \*/u\_iodelay\_ctrl/sys\_rst\_i}]"

set\_false\_path -from [get\_pins Inst\_Audio/DDR/rstn\_reg/C] -to [get\_pins Inst\_Audio/DDR/Inst\_DDR/u\_ddr\_mig/u\_ddr2\_infrastructure/rstdiv0\_sync\_r\*/PRE]

#needed for sync-async modules

set\_false\_path -through [get\_pins -filter {NAME =~ \*SyncAsync\*/oSyncStages\_reg[\*]/D} -hier]

# Manual de utilizare

## 8.1 Modul de funcționare al aplicației

Pentru realizarea cu succes a aplicației avem nevoie de următoarele componente:

* Placa Nexys4 DDR
* Modulul Bluetooth PMOD BT2
* Cablu USB
* IDE-ul Vivado
* Un telefon cu sistemul de operare Android și cu capabilități Bluetooth

Următorii pași trebuie executați:

1. Se deschide IDE-ul Vivado Design Suite (preferabil versiunea 2018.2) și creează un nou proiect; în cazul unei versiuni mai vechi, s-ar putea ca IP-urile utilizate în proiect să nu funcționeze. În acest caz e recomandat consultarea secțiunilor 8.2 și 8.3.
2. La crearea unui nou proiect se alege placa Nexys4 DDR sau xc7a100tcsg324-1 ca și configurație.
3. Se atașează modulul Bluetooth PMOD BT2 de Jumper A și se conectează plăcuța prin USB la calculator. Se pornește plăcuța.
4. În panoul Flow Navigator din stânga, se apasă pe Open Hardware Manager -> Open Target -> Auto Connect. După conectare se apasă pe Program Device și se alege Bitstream-ul proiectului pentru a programa placa.
5. Pe un telefon mobil se instalează aplicația Proiect44.apk, și se pornește.
6. Se apasă pe butonul Connect pentru a realiza conectarea modulului Bluetooth cu telefonul.
7. O dată ce s-a conectat (PMOD BT2 are LED-ul roșu aprins constant și pe aplicație scrie acest lucru), se apasă pe butonul BTNU pentru a începe înregistrarea timp de 5 secunde (starea de Recording este ilustrată prin toate LED-urile de pe placă aprinse).
8. După ce s-au stins LED-urile de pe placă, se apasă pe butonul Play de pe aplicație. Pentru câteva secunde se vor afișa în format hexazecimal valorile înregistrate.
9. După finalizarea afișării, se poate realiza închiderea conexiunii apăsând pe butonul Disconnect.

## 8.2 Generarea IP-urilor

Pentru utilizarea componentei DDR2 a plăcii și a generatorului de frecvență, trebuie generat IP-urile specifice acestora. În acest subcapitol vom detalia pașii pentru generarea acestora.

### 8.2.1 Generarea IP-ului DDR2

Pentru a genera cu succes IP-ul necesar pentru utilizarea componentei DDR2 necesare acestui proiect se vor urma următorii pași:

1. O imagine care conține text

   Descriere generată automatDe la meniul *Flow Navigator* *-> Project Manager*, se selectează opțiunea **IP Catalog**.
2. După ce s-a deschis meniul **IP Catalog**, la submeniul *Cores* se selectează folderele *Memories & Storage Elements -> Memory Interface Generators* și se dă dublu-click pe **Memory Interface Generator (MIG 7 Series)***.*
3. Se va deschide o fereastră numită *Xilinx Memory Interface Generator.* Apăsați pe **Next**.
4. În continuare apăsați pe **Next** până la pagina **Memory Selection** (Obs. Pentru o altă placă

în afară de Nexys 4 DDR, Figura 8.2.1.1. Controller Options

configurați paginile anterioare

în funcție de placă pe care o aveți. De asemenea, s-ar putea ca opțiunile să difere).

1. La pagina **Memory Selection**, alegeți opțiunea DDR2 SDRAM și dați **Next**.
2. La pagina **Options for Controller 0 – DDR2 SDRAM** selectați opțiunile de la figura 8.2.1.1. și apoi apăsați butonul **Next.**

Obs: Datele pe care le-am luat ca să configurăm opțiunile de la următoarele pagini se regăsesc în [6].

1. La pagina **Memory Options for Controller 0 – DDR2 SDRAM** selectați opțiunile de la figura 8.2.1.2. și apoi apăsați butonul **Next**.

O imagine care conține text

Descriere generată automat

Figura 8.2.1.2. Memory Options

1. La următoarea pagină selectați opțiunile din figura 8.2.1.3. și apoi apăsați butonul **Next**.

O imagine care conține masă

Descriere generată automat

Figura 8.2.1.3. Extended FPGA Options

1. În continuare apăsați butonul **Next** până la **Bank Selection**. Asigurați-vă că sunt bifate cu verde *Address/Control* și *Data*. Dați **Next** mai departe fără să modificați nimic.

Graphical user interface

Description automatically generated with medium confidence

Figura 8.2.1.4. Bank Selection

1. Dați **Next** până la **Simulation Options**. Selectați **Accept** și dați **Next** mai departe până apare butonul **Generate**. Apăsați butonul și așteptați până apare meniul **Generate Output Products**. Apăsați **Generate**.
2. Componenta DDR2 este generată sub numele de *mig\_7series\_0.xci*. Ea poate fi vizualizată dând click pe săgeată și selectând fișierul *vhd*. De asemenea, se poate edita IP-ul dând dublu click pe ea. Componenta apare în proiect și e gata să fie utilizată de către convertorul Ram2Ddr.

### 8.2.2 Generarea IP-ului ClockGen

Pentru a genera cu succes IP-ul necesar pentru utilizarea crearea unui generator de clock necesar acestui proiect se vor executa următorii pași:

1. De la meniul *Flow Navigator* *-> Project Manager*, se selectează opțiunea **IP Catalog**.
2. După ce s-a deschis meniul **IP Catalog**, la submeniul *Cores* se selectează folderele *FPGA Features and Design -> Clocking* și se dă dublu-click pe **Clocking Wizard**.
3. La deschiderea ferestrei **Customize IP**, se vor selecta următoarele opțiuni:

* Pentru meniul **Clocking Options** se vor lăsa opțiunile *default*.
* La meniul **Output Clocks** se va alege un clock de ouput de 200 MHz cu factor de umplere de 50% ca în figura 8.2.1.1.

Table

Description automatically generated

Figura 8.2.2.1: Opțiuni pentru clk\_out1

* La meniul **Port Remaining** se vor lăsa opțiunile default.
* La meniul **MMCM Settings** se va da enable pe *Allow Override Mode* și se vor schimba valorile ca în figura 8.2.2.2.

Table

Description automatically generated

Figura 8.2.2.2. Opțiuni pentru MMCM

1. Dați pe **Ok** și la **Generate Output Products** dați pe **Generate**.
2. Componenta Generatorului de Clock este creată sub numele de *clk\_wiz\_0.xci*. Ea poate fi vizualizată dând click pe săgeată și selectând fișierul *vhd*. De asemenea, se poate edita IP-ul dând dublu click pe ea. Componenta apare în proiect și e gata să fie utilizată.