MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Catedra de Calculatoare

Inregistrarea unor date audio de la microfonul placii Nexys 4 DDR si transmiterea lor la un dispozitiv mobil

Pavel Madalina-Adriana
Popa Alexandra-Maria
Grupa 30234

Indrumator de proiect : Samuel Dolean
13 ianuarie 2020

Cuprins

1 Rezumat

Proiectul ales a avut ca scop proiectarea si implementarea unui sistem de inregistrare a unor date audio provenite de la microfonul placii Nexys 4 DDR si transmiterea acestor date catre un dispozitiv mobil. Tema propusa de acest proiect a adus in prim plan proiectarea utilizand limbajul VHDL, cat si implementarea pe placa Digilent Nexys 4 DDR, stocarea datelor audio povenite de la microfonul placii Nexys 4 DDR in memoria placii, utilizarea modulului Bluetooth Pmod BT2 pentru transmiterea datelor audio la un dispozitiv mobil. Problema enuntata anterior a necesitat cunostinte legate de dezvoltarea de sisteme utilizand limbajului VHDL, cat si cunostinte cu privire la placa programabila Digilent Nexys 4 DDR, a diferitelor caracteristici ale acesteia, printre care memoria, microfonul si modulul Bluetooth Pmod BT2. Din analiza problemei a rezultat faptul ca datele de intrare necesare sunt reprezentate de sunetele inregistrate prin intermediul placii, iar in ceea ce priveste setul de date de iesire, acesta este reprezentat de aceleasi sunete, transmise pe un dispozitivului mobil. Rezultatele obtinute in urma realizarii acestui proiect demostreaza faptul ca datele reprezentate de sunete au fost captate de microfonul placii de dezvoltare, inregistrate intr-o memorie RAM si transmise mai departe prin modulul bluetooth catre dispzitivul mobil. Prin rezultatele finale se demostreaza faptul ca aplicatia propusa la inceput a reusit sa fie implementata la nivel hardware.

2 Introducere

Transmiterea de informatii reprezinta o necesitate tot mai mare in contextul evolutiei tehnologiei, acest fapt reprezentand fondul nevoii de dezvoltare si eficientizare a comunicarii prin transmiterea de date, printre care o mare parte dintre aceste date fiind cele audio. Transmiterea de informatii audio se regaseste in majoritatea domeniilor, astfel cererea in ceea ce priveste calitatea transmiterii audio este tot mai mare in ultimii ani. Industria ingineriei face eforturi mari pentru imbunatatirea calitatii transmiterii sunetelor pentru consumatorii din domeniul electric pana la cel al automobilelor. Scopul acestui proiect il reprezinta cercetarea cat mai amanuntita in ceea ce priveste transmiterea semnalelor audio si simularea unui sistem ce permite propagarea datelor inregistrate prin intermediul microfonului placii de dezvoltare Digilent Nexys 4 DDR si stocate in memoria interna, catre un dispozitiv mobil. Acest proces se poate realiza utilizand caracteristicile placii de dezvoltare in care se stocheaza informatia captata pt ca mai apoi sa poata fi transmisa si receptionata de un alt dispozitiv.

Proiectul ales a avut ca scop proiectarea si implementarea unui sistem de inregistrare a unor date audio provenite de la microfonul placii Nexys 4 DDR si transmiterea acestor date catre un dispozitiv mobil. Tema propusa de acest proiect a adus in prim plan proiectarea utilizand limbajul VHDL, cat si implementarea pe placa Digilent Nexys 4 DDR, stocarea datelor audio provenite de la microfonul placii Nexys 4 DDR in memoria placii, utilizarea modulului Bluetooth Pmod BT2 pentru transmiterea datelor audio la un dispozitiv mobil. Problema enuntata anterior a necesitat cunostinte legate de dezvoltarea de sisteme utilizand limbajului VHDL, cat si cunostinte cu privire la placa programabila Digilent Nexys 4 DDR, a diferitelor caracteristici ale acesteia, printre care memoria, microfonul si modulul Bluetooth Pmod BT2. Din analiza problemei a rezultat faptul ca datele de intrare necesare sunt reprezentate de sunetele inregistrate prin intermediul placii, iar in ceea ce priveste setul de date de iesire, acesta este reprezentat de aceleasi sunete, transmise pe un dispozitivului mobil.

Domeniul de studiu cuprinde intelegerea modului de functionare si utilizarea componentelor placii de dezvoltare Nexys 4 DDR, implicate in realizarea acestui proiect, dar si a protocolului de comunicare dintre placa si dispozitivul mobil, prin intermediul modulului Bluetooth Pmod BT2. Realizarea proiectului impune aplicarea unor cunostinte dobandite la o parte din materiile studiate in domeniul ingineriei hardware, cunostinte ce cuprind scrierea de cod intr-un limbaj de descriere hardware (in cazul nostru, VHDL - Very High Speed Integrated Circuit Hardware Description Language), descrierea de sisteme, simularea, sintetizarea si implementarea lor cu ajutorul mediului de dezvoltare Vivado, cunostintele legate de componentele integrate ale placii si de modalitatile de transmitere a informatiilor intre calculator si placa. Acest proiect implica insa comunicarea dintre placa de dezvoltare si un dispozitiv mobil, prin intermediul modulului Bluetooth Pmod BT2, pentru care a fost necesara o documentare suplimentara in vederea intelegerii modului de functionare a modulului, a modalitatii de conversie a datelor audio in date a caror forma poate fi stocata in memoria placii si a protocolului de comunicare necesar transmisiei de informatii intre cele 2 dispozitive.

In ceea ce priveste tehnologia de baza, inregistrarea datelor audio se realizeaza prin

intermediul microfonului omnidirectional pus la dispozitie de placa, ADMP421, a carui latime de banda de frecventa este foarte mare, ceea ce permite inregistrarea unor sunete cat mai naturale. [1] Microfonul transmite sunetele audio la iesire in format PDM (Pulse Density Modulated format). [3] In acest format, sunetele sunt reprezentate prin semnale compuse dintr-o secventa de valori de 0 si 1 : 0 reprezinta valoarea minima a amplitudinii, iar 1 reprezinta valoarea maxima pozitiva a amplitudinii. Pentru a putea fi redate pe dispozitivul mobil, aceste semnale trebuie convertite intr-un tip recunoscut de dispozitivul mobil. PCM (Pulse-Code Modulation) este o modalitate de codificare a semnalelor audio necomprimate. Semnalele audio astfel convertite vor putea fi redate pe dispozitivul mobil prin intermediul unei aplicatii care permite redarea unor fisiere audio cu acest format.

Transmiterea datelor de la placa spre dispozitivul mobil se realizeaza prin intermediul modulului Pmod BT2, un modul periferic care utilizeaza un alt modul, RN42/RN42N, folosit pentru a oferi capacitatea de comunicare wireless. Acest modul pune la dispozitie 2 antene (RN42 – PCB trace antenna, RN42N – external antenna) si suport pentru Bluetooth, astfel incat ofera o viteza de transfer a datelor de pana la 3 Mbps pentru distante de pana la 10 m.[6] Modulul Pmod BT2 poate comunica cu placa de dezvoltare prin transmiterea si receptia datelor pe cale seriala (UART - Universal Asynchronous Receiver/Transmitter). [5]

Problema de rezolvat consta in divizarea proiectului in mai multe subprobleme si gasirea unor solutii pt fiecare astfel de subproblema, urmatorul pas fiind integrarea acestor solutii gasite intr-una singura, proiectul final, ce urmeaza sa fie implementat si testat pentru verificarea functionalitatii. In setul de subprobleme ce participa la solutionarea proiectului se numara stabilirea clara a momentelor de timp in care incepe si se termina inregistrarea, in vederea determinarii duratei, acest lucru fiind important din punctul de vedere al stocarii in memoria placii. O alta subproblema este reprezentata de realizarea conversiei din formatul in care microfonul inregistreaza sunetele (format PDM) intr-un format recunoscut de dispozitivul mobil, pentru a putea reda inregistrarea prin intermediul sau (format PCM). Odata datele convertite, este necesara gasirea unei solutii prin care sa transferam aceste date de pe placa de dezvoltare pe dispozitivul mobil, cu ajutorul modulului Bluetooth BT2 pus la dispozitie. Asadar, trebuie determinata modalitatea de comunicare dintre modul si placa, trebuie stabilita o conexiune intre cele 2 dispozitive (placa de dezvoltare si dispozitivul mobil), urmata de transferul efectiv si salvarea datelor pe dispozitivul mobil.

Obiectivele principale ale proiectului sunt reprezentate de proiectarea unui automat de stare care sa realizeze tranzitia dintr-o etapa in alta pana la finalizarea intregului proces ce constituie tema proiectului, un automat care sa asigure mentinerea sistemului intr-o stare consistenta, indiferent de momentul de timp in care se afla, proiectarea componentelor care au ca scop solutionarea unei subprobleme enuntate mai sus, realizarea conexiunilor dintre acestea care asigura functionalitatea intregului sistem, implementarea acestor componente si realizarea de teste care sa demonstreze functionalitatea.

In vederea proiectarii, implementarii si solutionarii problemei impuse de proiect, a fost necesara o documentare mai amanuntita cu privire la modul de functionare al microfonului placii de dezvoltare, cu privire la modulul Bluetooth Pmod BT2 utilizat pentru realizarea conexiunii dintre cele 2 dispozitive, cu privire la modul de conectare si transmitere a informatiei si nu in ultimul rand, cu privire la modalitatea de stocare a unor date audio, in format digital. Urmatorul pas in dezvoltarea solutiei este reprezentat de realizarea schemelor bloc, care ajuta la identificarea componentelor necesare si a conexiunilor dintre acestea, oferind o privire de ansamblu asupra proiectului, componentele fiind percepute ca niste cutii negre, a caror implementare este ascunsa. Dupa aceasta etapa, este necesara implementarea propriu-zisa a componentelor prin descrieri in limbajul VHDL, in mediul de proiectare Vivado. Urmeaza etapa de simulare, in care este testat comportamentul sistemului si este urmarita evolutia semnalelor. Se pot testa astfel mai multe variante de proiectare, urmand

apoi sa se aleaga cea mai eficienta varianta pentru implementarea pe placa de dezvoltare si testarea de pe placa.

In capitolele urmatoare se va descrie pe larg fiecare etapa indeplinita pentru realizarea proiectului. Asadar, in capitolul Fundamentare teoretica se vor prezenta detalii despre mediul de dezvoltare in care se vor descrie componentele necesare, limbajul de descriere hardware utilizat, dar si informatii cu privire la modul de functionare al placii de dezvoltare si al modulelor necesare. In cadrul capitolului Proiectare si implementare se vor regasi schemele bloc care ofera o imagine de ansamblu asupra proiectului, algoritmii utilizati, dar si detalii cu privire la implementarea hardware si software. Capitolul Rezultate experimentale are ca scop prezentarea rezultatelor obtinute, prezentarea simularilor de functionare, a rapoartelor de implementare, impreuna cu imagini care sa sustina rezultatele. In capitolul Concluzii se va prezenta un rezumat al tuturor etapelor care au condus la realizarea proiectului, observatii cu privire la rezultate si posibilitatile de dezvoltare ulterioara. In final, se vor regasi inca 2 sectiuni : Bibliografia, in care se vor enumera toate resursele consultate pentru realizarea proiectului, si Anexa, in care va fi atasat codul sursa.

3 Fundamentare teoretica

Proiectarea sistemului de inregistrare si transmitere a datelor audio a necesitat utilizarea unei palete largi de resurse, dintre acestea facand parte mediul de dezvoltare a sistemului, limbajul de dezvoltare, resursele hardware puse la dispozitie de placa de dezvoltare Nexys 4 DDR, cat si dispozitivul mobil implicat in procesul de preluare a semnalului audio.

3.1 Mediul de dezvoltare Xilinx Vivado Design Suite

Figure 3.1: Logo Vivado

Prin urmare, mediul de dezvoltare ce pune la dispozitie resursele necesare pentru crearea sistemului ce se doreste a fi implementat, intr-un limbaj specific, il reprezinta Vivado Design Suite. Mediul de proiectare Xilinx Vivado Design Suite reuneste toate utilitarele necesare pentru proiectarea sistemelor digitale utilizand circuite FPGA Xilinx. Mediul Vivado Design Suite pune la dispozitia utilizatorilor o interfata grafica numita Vivado Integrated Design Environment (IDE). Aceasta interfata grafica Vivado IDE permite accesarea utilitarelor de proiectare aferente pentru punerea in executie a operatiilor necesare in diferite etape ale fluxului de proiectare cu circuite FPGA, precum: descrierea sistemului, sinteza, implementarea, analiza proiectului, configurarea circuitului, verificarea și depanarea. [2]. Printre caracteristicile mediului de proiectare se numara faptul ca Vivado permite dezvoltatorilor sa sintetizeze propriile implementari ale sistemului creat, sa efectueze analize de sincronizare, sa examineze diagrame RTL (RTL – Register Transfer Level) si sa analizeze comportamentul unui sistem la diferiti stimuli. De asemenea, printre facilitatile puse la dispozitie de mediul prezentat se regaseste si posibilitatea de analiza a proiectului prin simulare logica si testare in diferite etape ale fluxului de proiectare, a puterii consumate dar si a perioadei de timp in care are loc simularea. Invocarea utilitarelor de proiectare integrate in mediul Vivado Design Suite se face prin doua moduri:

1. invocarea utilitarelor din interfata grafica Vivado IDE - necesita crearea unui proiect utilizand Vivado IDE si includerea fisierelor sursa in proiect;
2. utilizarea unor comenzi TCL (Tool Command Language) - control total asupra fluxului de proiectare pentru programator.

Prin urmare, Vivado Design Suite include si pune la dispozitie utilitare de proiectare de nivel inalt precum: IP Integrator, folosit pentru instantierea si configurarea modulului procesului, selectarea dispozitivelor periferice, configurarea setarilor hardware si conectarea acestor componente pentru a crea un sistem inglobat complex; System Generator, ce permite utilizarea mediului de modelare si simulare pentru proiectarea unor module de procesare digitala a semnalelor implementate in circuitele FPGA; High-Level Synthesis, ce permite programelor C, C++ si SystemC sa fie directionate catre dispozitivele Xilinx fara a fi necesara crearea manuala a RTL.

Vivado Design Suite ofera posibilitatea de a utiliza diferite fisiere sursa ce includ descrierea partiala sau totala a sistemului digital ce se doreste a fi implementat. In ceea ce priveste dispozitivele de suport, Xilinx recomanda Vivado Design Suite pentru dezvoltarea utilizand Ultrascale, Ultrascale +, Spartan-7, Virtex-7, Kintex-7, Artix-7 și Zynq-7000. [9].

3.2 Limbajul de descriere hardware VHDL

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity signed_adder is
6   port
7   (
8     aclr : in   std_logic;
9     clk  : in   std_logic;
10    a    : in   std_logic_vector;
11    b    : in   std_logic_vector;
12    q    : out  std_logic_vector
13  );
14 end signed_adder;
15
16 architecture signed_adder_arch of signed_adder is
17   signal q_s : signed(a'high+1 downto 0); -- extra bit wide
18
19 begin -- architecture
20   assert(a'length >= b'length)
21     report "Port A must be the longer vector if different sizes!"
22     severity FAILURE;
23   q <= std_logic_vector(q_s);
24
25   adding_proc:
26   process (aclr, clk)
27     begin
28       if (aclr = '1') then
29         q_s <= (others => '0');
30       elsif rising_edge(clk) then
31         q_s <= ('0'&signed(a)) + ('0'&signed(b));
32       end if; -- clk'd
33     end process;
34
35 end signed_adder_arch;
```

Figure 3.2: Exemplu de cod scris in VHDL

Editorul de texte al mediului de proiectare Vivado IDE pune la dispozitie constructii ale limbajului VHDL, utilizat pentru simulare si sinteza. Astfel, VHDL (Very High Speed Integrated Circuit Hardware Description Language) este un limbaj de descriere hardware utilizat in descrierea comportamentului si arhitecturii sistemelor digitale, fiind totodata impreuna cu Verilog, cel mai utilizat limbaj de acest tip. VHDL a fost dezvoltat initial pentru a documenta comportamentul ASIC (application-specific integrated circuit). Ulterior, au fost dezvoltate simulatoare logice prin care se puteau citi fisiere VHDL, pentru ca mai apoi sa se dezvolte instrumente de sinteza logica prin care sa se poata citi VHDL si prin care sa se obtina o definitie a implementarii fizice a circuitului. Avantajul principal al utilizarii VHDL il

reprezinta faptul ca permite descrierea comportamentului sistemului si verificarea acestuia inainte ca instrumentele de sinteza sa realizeze o implementare propriu-zisa a acestuia cu porti si fire. De asemenea, VHDL ofera posibilitatea descrierii unui sistem concurent, astfel incat instructiunile sunt executate in mod simultan. Totodata, un proiect VHDL este multifunctional, portabil dar si un sistem de tip full-type, deoarece cuprinde cod structurat.[8]

3.3 Placa de dezvoltare Nexys 4 DDR

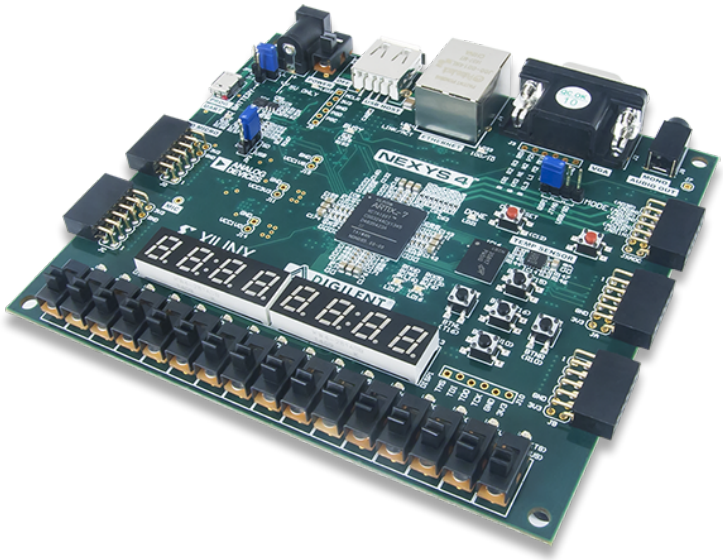

Figure 3.3: Nexys 4 DDR

Pentru implementarea proiectului, am folosit, de asemenea, placa de dezvoltare Digilent Nexys 4 DDR, care pune la dispozitie module necesare pentru inregistrarea si transmiterea semnalelor audio, dar si memoria placii in care se stocheaza datele audio preluate. Astfel, Nexys 4 DDR (Nexys A7) este o placa de dezvoltare digitala completa, programabila, ce se bazeaza pe cea mai recenta placa de tip Artix-7 Field Programmable Gate Array (FPGA), produsa de Xilinx. Nexys 4 DDR beneficiaza de un FPGA de mare capacitate, memorie externa de dimensiune mare, dar si o colectie de porturi USB, internet si alte porturi. Aceasta placa de dezvoltare poate sustine o varietate de tipuri de implementari, incepand cu cele de baza, circuite combinationale, pana la sisteme integrate. Totodata placa pune la dispozitie o gama larga de periferice integrate printre care, cele necesare implementarii proiectului propus. Totodata se regasesc diferite componente ce confera o posibilitate de dezvoltare cat mai mare prin suportul oferit de placa, fara a fi necesara utilizarea altor componente aditionale, astfel: microfon digital MEMs, amplificator de sunete, accelerometru, sensor de temperratura si mai multe dispozitive de intrare/iesire. Printre principalele caracteristici ale placii Nexys4 DDR se numara:

- 15850 slice-uri logice programabile
- 4860 Kbits de blocuri de RAM
- Semnal de ceas cu viteza de 450 MHz
- Memorie ce se imparte in : 128MiB DDR2, Serial Flash si microSD card slot
- Alimentarea se face de la USB sau orice sursa de putere externa intre 4.5-5.5 V
- I/O: 16 switch-uri, 16 LED-uri, 2 LED-uri de tip RGB si doua display-uri 7 Segmente

- Audio si Video: iesire VGA de 12 biti, iesire PWM audio si microfon PDM
- 4 porturi Pmod [4]

3.4 Microfonul placii de dezvoltare

Una dintre resursele puse la dispozitie de placa de dezvoltare Nexys4 DDR si folosite pentru implementarea proiectului propus o reprezinta microfonul. Nexys 4 DDR include un microfon MEMS omnidirectional ce utilizeaza un chip Analog Device ADMP421. Totodata, microfonul are un raspuns de frecventa plata (flat frequency) care variaza intre 100Hz si 15 kHz. Semnalul audio digitalizat este emis in format PDM (pulse density modulated). Arhitectura acestei componente este prezentata in figura urmatoare: [4]

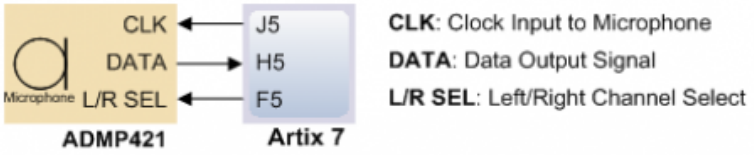

Figure 3.4: Schema bloc a microfonului

3.5 PDM (Pulse Density Modulation)

PDM sunt conexiuni de date ce isi castiga din ce in ce mai mult popularitatea in aplicatii audio portabile, printre care telefoane mobile si tablete, totodata fiind elemente esentiale in realizarea acestui proiect. Utilizand PDM, doua canale pot fi transmise doar cu doua fire, frecventa unui semnal find intre 1MHz si 3MHz. Intr-un bitstream PDM, '1' corespunde unui impuls pozitiv si '0' corespunde unui impuls negativ, astfel o rulare ce are doar valori de '1' corespunde valorii maxime pozitive ale amplitudinii, iar o serie formata doar din '0' corespunde amplitudinii minime. [4]

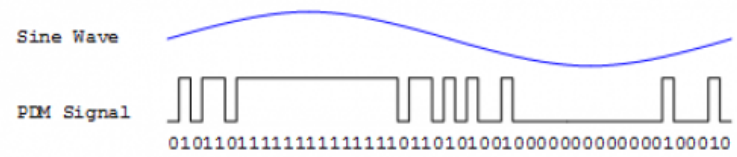

Figure 3.5: Reprezentarea PDM a unui semnal sinusoidal

Un semnal PDM este generat dintr-un semnal analogic printr-un proces numit modulare delta-sigma.

In continuare este prezentat modul de functionare. Se presupune ca intrarea analogica si iesirea digitala se incadreaza in acelasi interval de tensiune (0-VDD). Intrarea bistabilului actioneaza ca un comparator, si anume orice semnal peste Vdd/2 este considerat '1' si oricare semnal sub aceasta valoare este considerat '0'. Intrarea circuitului integral este diferenta dintre semnalul analog de intrare si semnalul PDM al ciclului de ceas anterior. Circuitul integral integreaza ambele intrari, iar iesirea acestuia este preluata de un D-Flip-flop. Media iesirii bistabilului flip-flop este egala cu valoare semnalului analogic de intrare, deci pentru a obtine valoarea intrarii analogice, este nevoie de un numarator ce numara valorile de '1' intr-o perioada de timp. [4]

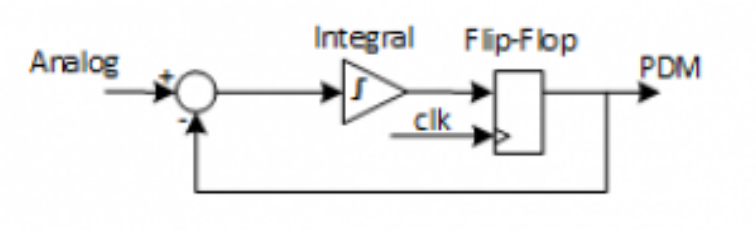

Figure 3.6: Circuit modulator delta-sigma

3.6 Timing-ul interfetei digitale pentru microfon

Intrarea de clock a microfonului poate varia intre 1MHz si 3.3MHz pe baza vitezei de esantionare si a cerintelor de precizie. Semnalul L/R trebuie setat la un nivel valid, astfel un nivel scazut pe L/PSEL face ca datele sa fie disponibile pe frontul crescator al semnalului de clock, in timp ce un nivel ridicat corespunde frontului descrescator al semnalului de clock: [4]

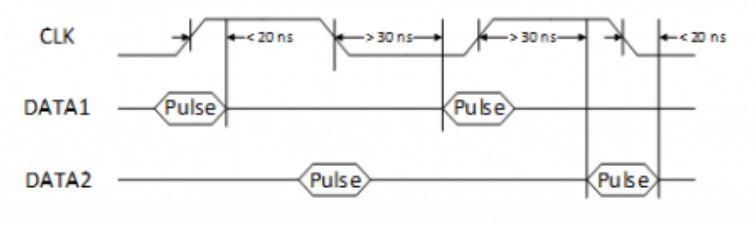

Figure 3.7: Diagrama de timp PDM

4 Proiectare si implementare

4.1 Componenta de transfer de date UART

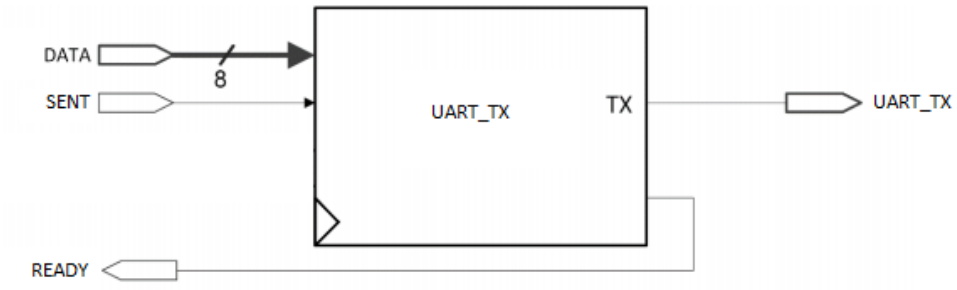

Figure 4.1: Schema bloc componenta UART

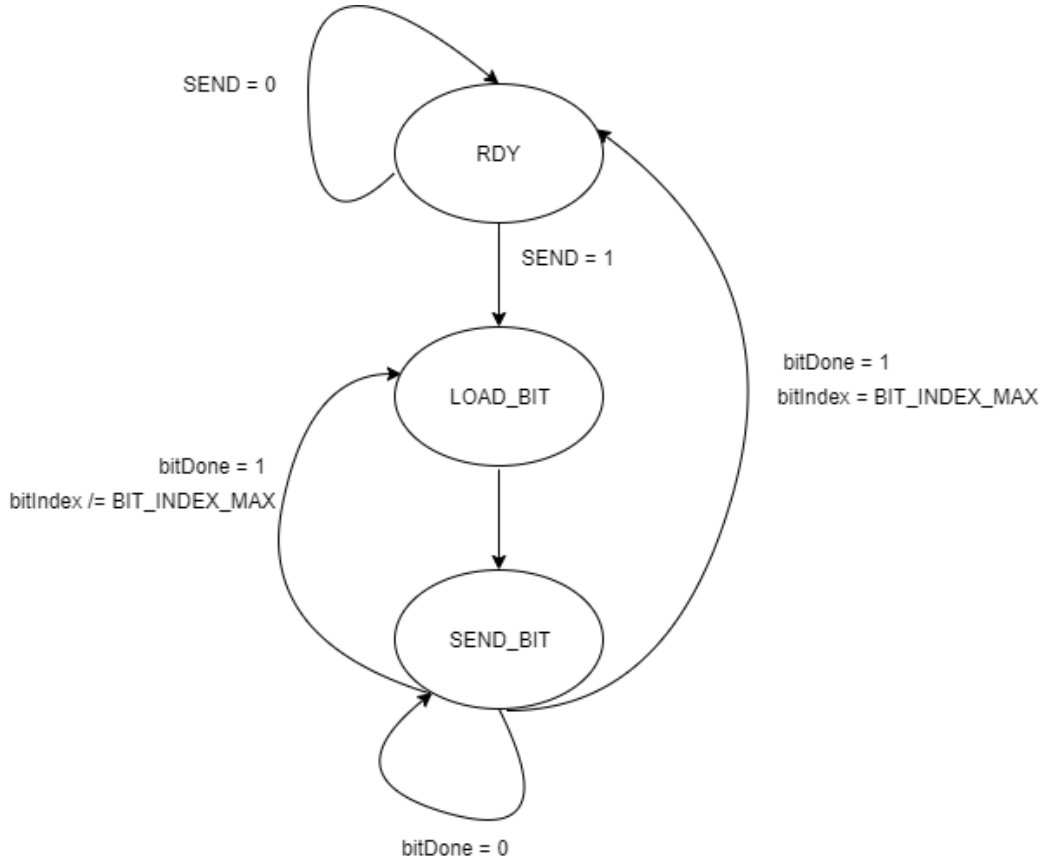

Figure 4.2: Diagrama de stare

Aceasta componenta poate fi utilizata pentru a transfera date pe un dispozitiv UART.Se va serializa un octet de date si se va transmite pe o linie TX.Datele serializate au urmatoarele caracteristici: 9600 Boud Rate, 8 biti de date, mai intai LSB, 1 bit de oprire, fara paritate.

Descrierea portului:

SEND - Folosit pentru a declansa o operatiune de trimitere. Logica nivelului superior ar trebui sa seteze acest semnal la un singur ciclu de ceas pentru a declansa un transfer.Cand acest semnal este setat, semnalul DATA trebuie sa fie valid. Nu poate fi primit pana cand READY nu este pe '1'.

DATA - datele paralele care trebuie trimise.

CLK - este asteptat un semnal de ceas de 100 MHz.

READY - acest semnal trece in starea low odata ce o operatiune de transmitere a inceput si ramane in aceasta stare pana cand s-a finalizat si modulul este gata sa trimita un alt octet.

UART_TX - acest semnal trebuie dirijat catre pinul TX corespunzator al dispozitivului UART extern.

4.2 Modulul de deserializare PDM

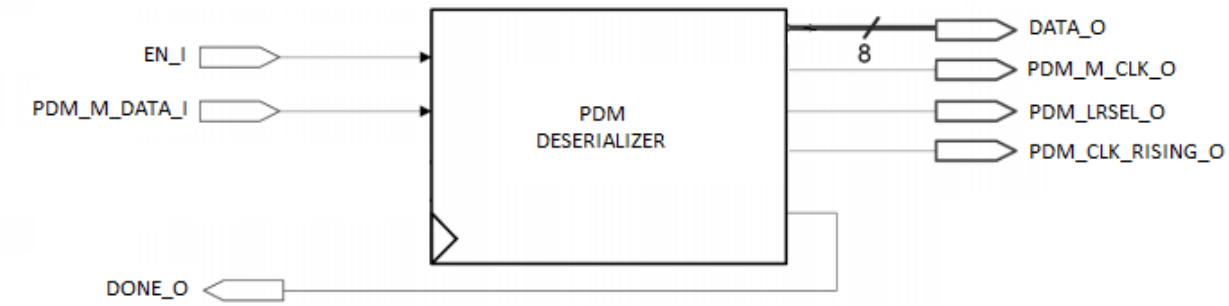

Figure 4.3: Schema bloc componenta PDM deserializator

Acest modul reprezinta deserializatorul datelor care provin de la microfon. Modulul generează semnalul pdm_m_clk_o catre microfonul ADMP421 (M_CLK) si datele sunt citite pe frontul pozitiv al acestui semnal. Apoi, modulul deserializeaza semnalul pe 8 biti atunci cand en_i='1'(ceea ce inseamna ca recodarea continua).

EN_I - semnal ce permite deserializarea.

DONE_O - semnalizeaza ca 8 biti sunt deserializati.

DATA_O - informatia de la iesire deserializata.

PDM_M_CLK_O - iesirea semnalului M_CLK catre microfon.

PDM_M_DATA_I - intrarea datelor PDM de la microfon.

PDM_LRSEL_O - setat pe '0', deci datele sunt citite pe frontul crescator.

PDM_CLK_RISING_O - semnalizeaza frontul crescator al M_CLK, folosit de componenta MicDisplay in controlerul VGA (neutilizat).

4.3 Debouncer

Acest modul reprezinta un debouncer si este utilizat pentru sincronizarea cu clock-ul sistemului si eliminarea activarilor multiple care pot aparea la apasarea unui buton.

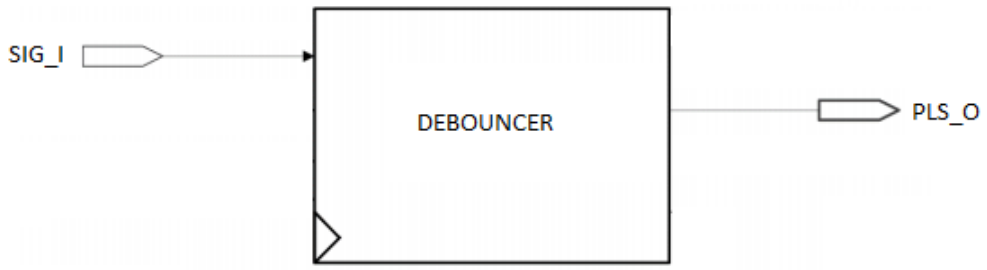

Figure 4.4: Schema bloc debouncer

4.4 Memoria RAM

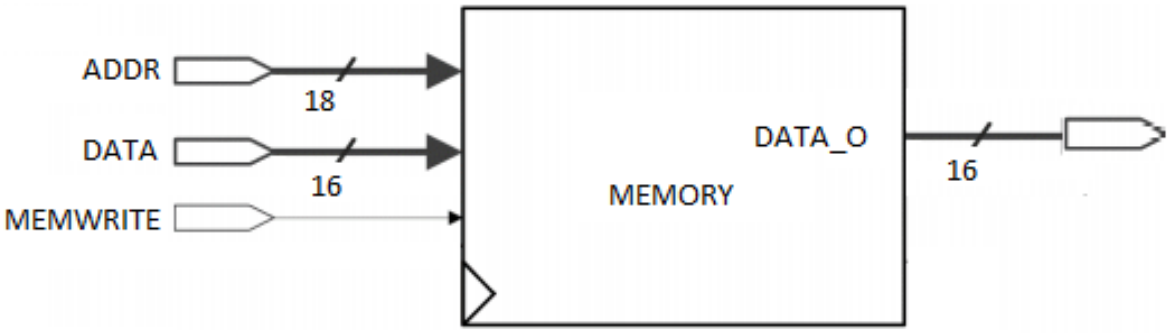

Figure 4.5: Schema bloc memorie RAM

Memoria este utilizata pentru stocarea datelor inregistrate de la microfon, datele fiind in format PDM, in cuvinte de 16 biti.

ADDR - adresa de memorie este reprezentata pe 18 biti, acest numar provenind din valoarea SECONDS_TO_RECORD*PDM_FREQ_HZ)/NR_OF_BITS, ce reprezinta numarul de cuvinte ce trebuie memorate.

DATA - datele ce urmeaza sa fie scrise in memorie

MEMWRITE - semnale de validare a scrierii

DATA_O - datele citite din memorie

4.5 Dispozitivul mobil

MANUAL DE UTILIZARE :

Am utilizat aplicatia Serial Bluetooth Terminal [7], o aplicatie terminal prin intermediul careia am realizat conexiunea dintre telefon si modulul bluetooth si am receptionat datele transmise de acesta din urma. Pasii pe care i-am urmat sunt :

- dupa instalarea aplicatiei pe telefon, se activeaza bluetooth-ul si se cauta dispozitivul bluetooth pentru a se face pair intre cele 2 dispozitive;

- in cadrul aplicatiei, se apasa primul buton din coltul dreapta sus pentru a se realiza conexiunea;

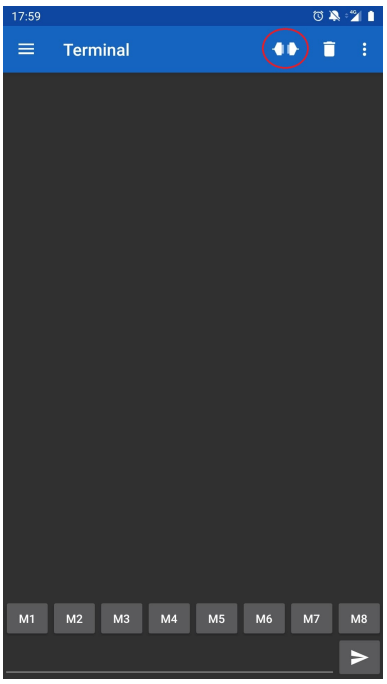

Figure 4.6: Interfata aplicatiei si butonul pentru realizarea conexiunii

- dupa realizarea cu succes a conexiunii, aspect ce este ilustrat de aplicatie, se va apasa butonul "up" de pe placa de dezvoltare si se vor rosti sunete timp de 4 secunde;

- daca pasii de mai sus au fost respectati, in cateva secunde in aplicatie vor aparea datele in format hexazecimal.

4.6 Alternative de proiectare considerate

Initial ne-am dorit sa convertim datele audio inregistrate de microfon din format PDM in format PCM pentru a putea asculta inregistrarea pe dispozitivul mobil. Nu am reusit acest lucru deoarece nu am gasit un convertor care sa poata fi integrat in proiect fara sa aduca modificari considerabile. In plus, aplicatia de terminal prin care se primesc datele nu ne ofera posibilitatea de a salva datele in format .pcm, doar .txt.

4.7 Schema bloc

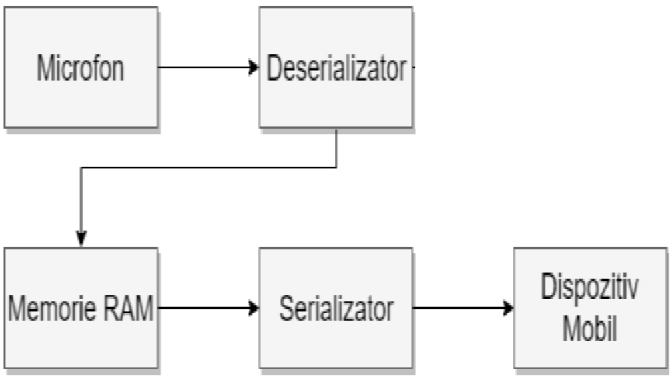

Figure 4.7: Schema bloc a proiectului

5 Rezultate experimentale

Instrumentele de proiectare utilizate sunt: limbajul VHDL, mediul de dezvoltare Vivado versiunea 2018.3, iar in ceea ce priveste dispozitivul mobil am utilizat aplicatia Serial Bluetooth Terminal.

5.1 Informatii din rapoartele de utilizare

Resource	Utilization	Available	Utilization %
LUT	346	63400	0.55
FF	369	126800	0.29
BRAM	128	135	94.81
IO	9	210	4.29

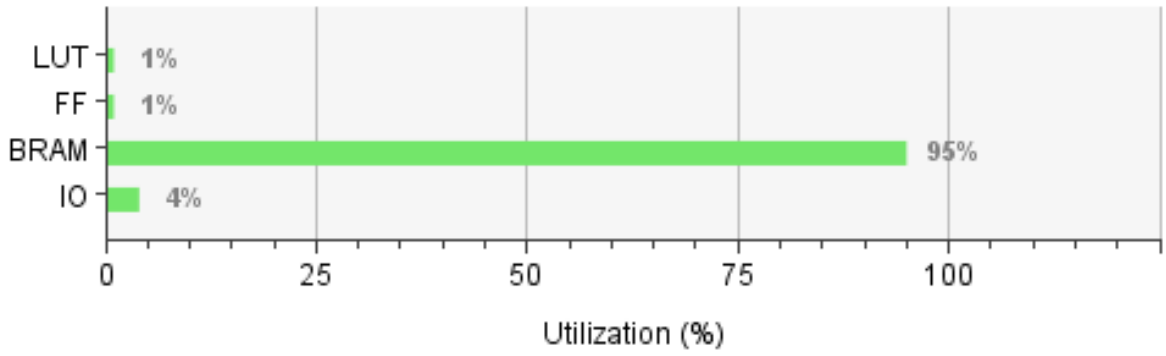

Figure 5.1: Raport de utilizare

Name	Slice LUTs (63400)	Block RAM Tile (135)	Bonded IOB (210)	BUFGCTRL (32)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)
project_module	346	128	9	1	369	8	256	346
debouncer_start_record (Dbncr)	11	0	0	0		0	10	11
pdm_des (PdmDes)	13	0	0	0		0	28	13
ram_memory (memory)	154	128	0	0		8	124	154
uart (UART_TX_CTRL)	26	0	0	0		0	30	26

Figure 5.2: Raport de utilizare

5.2 Rezultate

Am inregistrat cuvantul ”da” (plus zgomotul din sala), iar in aplicatia utilizata am observat urmatorul rezultat :

Figure 5.3: Datele audio receptionate in format hexazecimal

6 Concluzii

Proiectul a constat in proiectarea si implementarea unei solutii in vederea inregistrarii a 4 secunde de date audio prin intermediul microfonului placii de dezvoltare. Datele incep sa fie inregistrate in momentul apasarii butonului "up" de pe placa, iar timp de 4 secunde acestea sunt captate de microfon, deserializate si salvate in memorie in format PDM (16 biti). Urmatorul pas este reprezentat de citirea pe rand a datelor din memorie, impartirea acestora in cuvinte de lungime 8 biti, serializarea si transmiterea lor spre dispozitivul mobil prin intermediul modulului bluetooth pus la dispozitie.

Contributia noastra originala este reprezentata de proiectarea si implementarea memoriei de date si a modulului principal care leaga toate componentele intre ele si le gestioneaza.

Avantajul principal al proiectului este faptul ca este usor de inteles si utilizat, iar in ceea ce priveste dezavantajele, se poate mentiona faptul ca se pot inregistra doar 4 secunde.

In ceea ce priveste dezvoltarile ulterioare, se poate proiecta un filtru care sa realizeze conversia din format PDM in format PCM, se poate dezvolta o aplicatie care sa preia datele transmise prin bluetooth si sa le salveze intr-un fisier .pcm pentru a putea fi redate pe telefon.

Bibliografie

[1] *ADMP421 (Rev. D) - Analog Devices*. URL: https://www.analog.com/media/en/technical-documentation/obsolete-data-sheets/ADMP421.pdf.

[2] *Fluxul de proiectare cu circuite FPGA*. URL: http://users.utcluj.ro/~baruch/ssc/labor/Flux-Proiectare.pdf.

[3] *Nexys 4 DDR Reference*. URL: https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-4-ddr/nexys4ddr_rm.pdf.

[4] *Nexys 4 DDR Reference Manual*. URL: https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual.

[5] *Pmod BT2: Bluetooth Interface - Digilent*. URL: https://store.digilentinc.com/pmod-bt2-bluetooth-interface/.

[6] *RN42/RN42N Data Sheet*. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/50002328A.pdf?_ga=2.137490098.386362720.1571316779-1390112243.1526243289.

[7] *Serial Bluetooth Terminal App*. URL: https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=ro.

[8] *VHDL*. URL: https://en.wikipedia.org/wiki/VHDL.

[9] *Xilinx Vivado*. URL: https://en.wikipedia.org/wiki/Xilinx_Vivado.

A Codul sursa

A.1 Modulul principal

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 11/18/2019 08:39:34 AM
-- Design Name:
-- Module Name: project_module - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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

entity project_module is
  Port (btn_u : in  STD_LOGIC;
        clk_i : in  STD_LOGIC;
        rst_i : in STD_LOGIC;
        uart_txd : out  STD_LOGIC;
        pmodbt_rst : out  STD_LOGIC;
        pmodbt_cts : out  STD_LOGIC;
        pdm_m_clk_o : out std_logic;
        pdm_m_data_i : in  std_logic;
        pdm_lrsel_o : out std_logic);
end project_module;

architecture Behavioral of project_module is
```

```
signal en_des : STD_LOGIC := '0';
signal done_des : STD_LOGIC := '0';
signal data_des : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
signal btnu_int : std_logic;
constant SECONDS_TO_RECORD    : integer := 4;
constant PDM_FREQ_HZ          : integer := 1024000;
constant SYS_clk_FREQ_MHZ     : integer := 100;
constant NR_OF_BITS           : integer := 16;
constant NR_SAMPLES_TO_REC    : integer :=(((SECONDS_TO_RECORD*PDM_FREQ_HZ)/NR_OF_BITS)
- 1);
type state_type is (stIdle, stRecord, stInter, stSend);
signal state : state_type;
signal next_state : state_type;
signal addr : STD_LOGIC_VECTOR(17 downto 0) := (others => '0');
signal write_en_mem : STD_LOGIC := '0';
signal data_o_mem : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
signal cntRecSamples : INTEGER := 0;
signal done_des_dly : STD_LOGIC := '0';
signal send_uart : STD_LOGIC := '0';
signal ready : STD_LOGIC := '0';
signal done_ser : STD_LOGIC := '0';
signal done_ser_dly : STD_LOGIC := '0';
signal cntSentSamples : INTEGER := 0;
signal addr_record : STD_LOGIC_VECTOR(17 downto 0) := (others => '0');
signal addr_send : STD_LOGIC_VECTOR(17 downto 0) := (others => '0');
signal half : STD_LOGIC := '0';
signal data_half_to_send : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
begin
-------------------------------------------------------------------------
-- Debouncer
-------------------------------------------------------------------------
    debouncer_start_record : entity WORK.Dbncr
           generic map(
              NR_OF_CLKS          => 4095)
           port map(
              clk_i               => clk_i,
              sig_i               => btn_u,
              pls_o               => btnu_int);
-------------------------------------------------------------------------
-- Deserializer
-------------------------------------------------------------------------
    pdm_des : entity WORK.PdmDes
           generic map(
              C_NR_OF_BITS        => NR_OF_BITS,
              C_SYS_CLK_FREQ_MHZ  => SYS_CLK_FREQ_MHZ,
              C_PDM_FREQ_HZ       => PDM_FREQ_HZ)
           port map(
              clk_i               => clk_i,
              en_i                => en_des,
              done_o              => done_des,
```

```
            data_o              => data_des,
            pdm_m_clk_o         => pdm_m_clk_o,
            pdm_m_data_i        => pdm_m_data_i,
            pdm_lrsel_o         => pdm_lrsel_o);

    process(clk_i)
    begin
       if rising_edge(clk_i) then
          if state = stRecord then
             if done_des = '1' then
                cntRecSamples <= cntRecSamples + 1;
             end if;
             if done_des_dly = '1' then
                addr_record <= addr_record + 1;
             end if;
          else
             cntRecSamples <= 0;
             addr_record <= (others => '0');
          end if;
          done_des_dly <= done_des;
       end if;
    end process;
-------------------------------------------------------------------------
-- Memory
-------------------------------------------------------------------------
    ram_memory : entity WORK.memory
            port map (
                clk      => clk_i,
                addr     => addr,
                data     => data_des,
                memWrite => write_en_mem,
                data_o   => data_o_mem);
-------------------------------------------------------------------------
--  FSM
-------------------------------------------------------------------------
    SYNC_PROC: process(clk_i)
    begin
       if rising_edge(clk_i) then
           if rst_i = '1' then
               state <= stIdle;
           else
               state <= next_state;
           end if;
       end if;
    end process;

    NEXT_STATE_DECODE: process(state, btnu_int, cntRecSamples, cntSentSamples)
    begin
       next_state <= state;
       case (state) is
```

```
        when stIdle =>
           if btnu_int = '1' then
              next_state <= stRecord;
           end if;
        when stRecord =>
           if cntRecSamples = NR_SAMPLES_TO_REC then
              next_state <= stInter;
           end if;
        when stInter =>
           next_state <= stSend;
        when stSend =>
           if btnu_int = '1' then
              next_state <= stIdle;
           elsif cntSentSamples = NR_SAMPLES_TO_REC then
              next_state <= stIdle;
           end if;
        when others =>
           next_state <= stIdle;
     end case;
  end process;

--Decode Outputs from the State Machine
OUTPUT_DECODE: process(clk_i)
begin
   if rising_edge(clk_i) then
      case (state) is
         when stIdle =>
            send_uart    <= '0';
            en_des       <= '0';
            addr         <= (others => '0');
            write_en_mem <= '0';
         when stRecord =>
            addr         <= addr_record;
            en_des       <= '1';
            write_en_mem <= '1';
         when stInter =>
            en_des       <= '0';
            write_en_mem <= '0';
         when stSend =>
            addr         <= addr_send;
            if half = '0' then
                data_half_to_send <= data_o_mem(15 downto 8);
            else
                data_half_to_send <= data_o_mem(7 downto 0);
            end if;
            if cntSentSamples = NR_SAMPLES_TO_REC then
                send_uart <= '0';
            else
                send_uart <= ready;
            end if;
```

```
            when others =>
               send_uart    <= '0';
               en_des       <= '0';
               addr         <= (others => '0');
               write_en_mem <= '0';
         end case;
      end if;
   end process;
--------------------------------------------------------------------------------
--  UART_TX_CTRL
--------------------------------------------------------------------------------
   uart : entity WORK.UART_TX_CTRL
        port map (
            SEND    => send_uart,
            DATA    => data_half_to_send,
            CLK     => clk_i,
            READY   => ready,
            UART_TX => uart_txd,
            DONE    => done_ser);

   -- count the sent samples
   process(clk_i)
   begin
      if rising_edge(clk_i) then
         if state = stSend then
            if done_ser = '1' then
               half <= not half;
            end if;
            if done_ser_dly = '1' then
               if half='0' then
                   addr_send <= addr_send + 1;
                   cntSentSamples <= cntSentSamples + 1;
               end if;
            end if;
         else
            cntSentSamples <= 0;
            addr_send <= (others => '0');
         end if;
         done_ser_dly <= done_ser;
      end if;
   end process;

   pmodbt_rst <= '1';
   pmodbt_cts <= '0';
end Behavioral;
```

A.2 Debouncer

```
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
-- Author:  Mihaita Nagy
--          Copyright 2014 Digilent, Inc.
----------------------------------------------------------------------------------
--
-- Create Date:    17:11:29 03/06/2013
-- Design Name:
-- Module Name:    dbncr - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
-- This module represents a debouncer and is used to synchronize with the system clock
-- and remove glitches from the incoming button signals
--
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Dbncr is
   generic(
      NR_OF_CLKS : integer := 4095 -- Number of System Clock periods while the
                                   -- incoming signal
   );                              -- has to be stable until a one-shot output signal
                                   -- is generated
   port(
      clk_i : in std_logic;
      sig_i : in std_logic;
      pls_o : out std_logic
   );
end Dbncr;

architecture Behavioral of Dbncr is

signal cnt : integer range 0 to NR_OF_CLKS-1;
signal sigTmp : std_logic;
signal stble, stbleTmp : std_logic;

begin

   DEB: process(clk_i)
   begin
```

```
      if rising_edge(clk_i) then
         if sig_i = sigTmp then -- Count the number of clock periods if the signal
                                -- is stable
            if cnt = NR_OF_CLKS-1 then
               stble <= sig_i;
            else
               cnt <= cnt + 1;
            end if;
         else -- Reset counter and sample the new signal value
            cnt <= 0;
            sigTmp <= sig_i;
         end if;
      end if;
   end process DEB;

   PLS: process(clk_i)
   begin
      if rising_edge(clk_i) then
         stbleTmp <= stble;
      end if;
   end process PLS;

   -- generate the one-shot output signal
   pls_o <= '1' when stbleTmp = '0' and stble = '1' else '0';

end Behavioral;
```

A.3 Modulul de deserializare

```
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
-- Author:  Mihaita Nagy
--          Copyright 2014 Digilent, Inc.
----------------------------------------------------------------------------------
--
-- Create Date:    14:24:36 04/02/2013
-- Design Name:
-- Module Name:    PdmDes - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--       This module represents the deserializer of the microphone data. The module
--    generates the pdm_m_clk_o signal to the ADMP421 Microphone (M_CLK) and data
-- is read on the positive edge of this signal.
--
--       Then the module deserializes the signal on 8 bits when en_i = '1'
-- (it means that recoding is going on)
--
--       The module also generates the pdm_clk_rising_o signal, that is active when
```

```
-- the positive edge of the pdm_m_clk_o signal occures. This signal is used in the VGA
-- controller, the MicDisplay component to display audio data on the screen. The signal
-- is two system clock period length, in order to make it easier the synchronizing with
-- the VGA clock domain (108MHz)
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

----------------------------------------------------------------------------
-- Module Declaration
----------------------------------------------------------------------------
entity PdmDes is
   generic(
      C_NR_OF_BITS : integer := 16;
      C_SYS_CLK_FREQ_MHZ : integer := 100;
      C_PDM_FREQ_HZ : integer := 1024000
   );
   port(
      clk_i : in std_logic;
      en_i : in std_logic; -- Enable deserializing (during record)

      done_o : out std_logic; -- Signaling that 16 bits are deserialized
      data_o : out std_logic_vector(C_NR_OF_BITS - 1 downto 0); -- output deserialized
                                                                -- data

      -- PDM
      pdm_m_clk_o : out std_logic; -- Output M_CLK signal to the microphone
      pdm_m_data_i : in std_logic; -- Input PDM data from the microphone
      pdm_lrsel_o : out std_logic--; -- Set to '0', therefore data is read on the
                                    -- positive edge
   );
end PdmDes;

architecture Behavioral of PdmDes is

----------------------------------------------------------------------------
-- Signal Declarations
----------------------------------------------------------------------------
-- Divider to create pdm_m_clk_0
signal cnt_clk : integer range 0 to 127 := 0;
-- Internal pdm_m_clk_o signal
signal clk_int : std_logic := '0';
```

```
-- Piped clk_int signal to create pdm_clk_rising
signal pdm_clk_rising : std_logic;

-- Shift register to deserialize incoming microphone data
signal pdm_tmp : std_logic_vector((C_NR_OF_BITS - 1) downto 0);
-- Count the number of bits
signal cnt_bits : integer range 0 to 31 := 0;

-- To create a pdm_clk_rising impulse of two clock period length
-- This signal will be registered in the MicDisplay module on the 108MHz pxlclk
--signal pdm_clk_rising_reg : std_logic_vector (2 downto 0);

signal en_int : std_logic;
signal done_int : std_logic;

------------------------------------------------------------------------
-- Module Implementation
------------------------------------------------------------------------
begin

   -- with L/R Sel tied to GND => output = DATA1 (rising edge)
   pdm_lrsel_o <= '0';

   -- Synchronize the enable input
   SYNC: process(clk_i)
   begin
      if rising_edge(clk_i) then
         en_int <= en_i;
      end if;
   end process SYNC;

------------------------------------------------------------------------
-- Deserializer
------------------------------------------------------------------------
-- Sample input serial data process
   SHFT_IN: process(clk_i)
   begin
      if rising_edge(clk_i) then
         if pdm_clk_rising = '1' then
            pdm_tmp <= pdm_tmp(C_NR_OF_BITS-2 downto 0) & pdm_m_data_i;
         end if;
      end if;
   end process SHFT_IN;


-- Count the number of sampled bits
   CNT: process(clk_i) begin
      if rising_edge(clk_i) then
         if en_int = '0' then
            cnt_bits <= 0;
```

```
         else
            if pdm_clk_rising = '1' then
               if cnt_bits = (C_NR_OF_BITS-1) then
                  cnt_bits <= 0;
               else
                  cnt_bits <= cnt_bits + 1;
               end if;
            end if;
         end if;
      end if;
   end process CNT;

-- Generate the done signal
   process(clk_i)
   begin
      if rising_edge(clk_i) then
         if pdm_clk_rising = '1' then
            if cnt_bits = 0 then
               if en_int = '1' then
                  done_int <= '1';
                  data_o <= pdm_tmp;
               end if;
            end if;
         else
            done_int <= '0';
         end if;
      end if;
   end process;

   done_o <= done_int;

-- Generate PDM Clock, that runs independent from the enable signal, therefore
-- the onboard microphone will always send data, that is displayed on the VGA screen
-- using the MicDisplay component
   CLK_CNT: process(clk_i)
   begin
      if rising_edge(clk_i) then
            if cnt_clk = (((C_SYS_CLK_FREQ_MHZ*1000000)/(C_PDM_FREQ_HZ*2))-1) then
               cnt_clk <= 0;
               clk_int <= not clk_int;
               if clk_int = '0' then
                  pdm_clk_rising <= '1';
               end if;
            else
               cnt_clk <= cnt_clk + 1;
               pdm_clk_rising <= '0';
            end if;
      end if;
   end process CLK_CNT;
```

```
   pdm_m_clk_o <= clk_int;

end Behavioral;
```

A.4 Memoria RAM

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 12/12/2019 04:58:49 PM
-- Design Name:
-- Module Name: memory - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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

entity memory is
    Port ( clk : in STD_LOGIC;
           addr : in STD_LOGIC_VECTOR(17 downto 0);
           data : in STD_LOGIC_VECTOR(15 downto 0);
           memWrite : in STD_LOGIC;
           data_o : out STD_LOGIC_VECTOR(15 downto 0));
end memory;

architecture Behavioral of memory is
type memorie is array(0 to 255999) of STD_LOGIC_VECTOR(15 downto 0);
signal RAM : memorie := (others=> "0000000000000000");
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if addr < CONV_STD_LOGIC_VECTOR(256000,18) then
                data_o<=RAM(conv_integer(addr));
            else
```

```
                data_o<= (others =>'0');
            end if;
            if(memWrite='1') then
                if addr < CONV_STD_LOGIC_VECTOR(256000,18) then
                    RAM(conv_integer(addr))<=data;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

A.5 UART

```
----------------------------------------------------------------------------
--UART_TX_CTRL.vhd -- UART Data Transfer Component
----------------------------------------------------------------------------
-- Author:  Sam Bobrowicz
--          Copyright 2011 Digilent, Inc.
----------------------------------------------------------------------------
--
----------------------------------------------------------------------------
--This component may be used to transfer data over a UART device. It will
-- serialize a byte of data and transmit it over a TXD line. The serialized
-- data has the following characteristics:
--         *9600 Baud Rate
--         *8 data bits, LSB first
--         *1 stop bit
--         *no parity
--
-- Port Descriptions:
--
--    SEND - Used to trigger a send operation. The upper layer logic should
--           set this signal high for a single clock cycle to trigger a
--           send. When this signal is set high DATA must be valid . Should
--           not be asserted unless READY is high.
--    DATA - The parallel data to be sent. Must be valid the clock cycle
--           that SEND has gone high.
--    CLK  - A 100 MHz clock is expected
--   READY - This signal goes low once a send operation has begun and
--           remains low until it has completed and the module is ready to
--           send another byte.
-- UART_TX - This signal should be routed to the appropriate TX pin of the
--           external UART device.
--
----------------------------------------------------------------------------
--
----------------------------------------------------------------------------
-- Revision History:
--  08/08/2011(SamB): Created using Xilinx Tools 13.2
----------------------------------------------------------------------------
```

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

entity UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC;
           DONE : out STD_LOGIC);
end UART_TX_CTRL;

architecture Behavioral of UART_TX_CTRL is

type TX_STATE_TYPE is (RDY, LOAD_BIT, SEND_BIT);

constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "10100010110000";
                                    --10416 = (round(100MHz / 9600)) - 1
constant BIT_INDEX_MAX : natural := 10;

--Counter that keeps track of the number of clock cycles the current bit
--has been held stable over the UART TX line. It is used to signal when the ne
signal bitTmr : std_logic_vector(13 downto 0) := (others => '0');

--combinatorial logic that goes high when bitTmr has counted to the proper
--value to ensure a 9600 baud rate
signal bitDone : std_logic;

--Contains the index of the next bit in txData that needs to be transferred
signal bitIndex : natural;

--a register that holds the current data being sent over the UART TX line
signal txBit : std_logic := '1';

--A register that contains the whole data packet to be sent, including start
--and stop bits.
signal txData : std_logic_vector(9 downto 0);

signal txState : TX_STATE_TYPE := RDY;

begin

    --Next state logic
    next_txState_process : process (CLK)
    begin
        if (rising_edge(CLK)) then
            case txState is
                when RDY =>
                    if (SEND = '1') then
```

```
                    txState <= LOAD_BIT;
                end if;
            when LOAD_BIT =>
                txState <= SEND_BIT;
            when SEND_BIT =>
                if (bitDone = '1') then
                    if (bitIndex = BIT_INDEX_MAX) then
                        txState <= RDY;
                    else
                        txState <= LOAD_BIT;
                    end if;
                end if;
            when others=> --should never be reached
                txState <= RDY;
        end case;
    end if;
end process;

bit_timing_process : process (CLK)
begin
    if (rising_edge(CLK)) then
        if (txState = RDY) then
            bitTmr <= (others => '0');
        else
            if (bitDone = '1') then
                bitTmr <= (others => '0');
            else
                bitTmr <= bitTmr + 1;
            end if;
        end if;
    end if;
end process;

bitDone <= '1' when (bitTmr = BIT_TMR_MAX) else '0';

bit_counting_process : process (CLK)
begin
    if (rising_edge(CLK)) then
        if (txState = RDY) then
            bitIndex <= 0;
        elsif (txState = LOAD_BIT) then
            bitIndex <= bitIndex + 1;
        end if;
    end if;
end process;

tx_data_latch_process : process (CLK)
begin
    if (rising_edge(CLK)) then
        if (SEND = '1') then
```

```
                txData <= '1' & DATA & '0';
            end if;
        end if;
    end process;

    tx_bit_process : process (CLK)
    begin
        if (rising_edge(CLK)) then
            if (txState = RDY) then
                txBit <= '1';
            elsif (txState = LOAD_BIT) then
                txBit <= txData(bitIndex);
            end if;
        end if;
    end process;

    UART_TX <= txBit;
    READY <= '1' when (txState = RDY) else '0';
    DONE <= '1' when (bitIndex = BIT_INDEX_MAX) else '0';
end Behavioral;
```