Universitatea Tehnică din Cluj Napoca

Facultatea de Automatică și Calculatoare, specializarea Calculatoare

Măsurarea temperaturii cu senzorul de pe placa Nexys 4 DDR și transmiterea ei la un dispozitiv mobil

Stundenți: Bonțea Carmen , Martin Denisa

Grupa 302310

Îndrumător proiect: Daniela Fati

Data: 16.11.2021

Cuprins

Contents

1.Rezumat

Măsurarea temperaturii este un aspect important al activităților umane zilnice. Odată cu progresele tehnologice în dispozitivele inteligente, multe sarcini sunt acum automatizate pentru a maximiza performanța. De la telefoane inteligente la mașini autonome cu conducere autonomă, posibilitățile sunt nesfârșite când vine vorba de inovație. Acest senzor de temperatură dinamic își propune să evidențieze puterea tehnologiei și modul în care aceasta poate îmbunătăți calitatea vieții. Acestea fiind spuse acest proiect își propune să creeze un protocol de comunicare între placa Nexys 4 DDR și între un device mobil, trimțând temperatura.

2. Introducere

Acest proiect are ca scop proiectarea utlizand limbajul VHDL si implementarea pe placa Digilent Nexys 4 DDR a unui sistem care sa permita masurarea temperaturii si transmiterea acesteia unui dispozitiv mobil. Obiectivele proiectului sunt : utilizarea limbajului VHDL și folosirea placii Nexys 4 DDR, măsurarea temperaturii cu senzorul de temperatură ADT7420, utilizarea modulului Bluetooth Pmod BT2 pentru transmiterea temperaturii la un dispozitiv mobil, crearea unei aplicații pe dispozitivul mobil pentru preluarea și afișarea temperaturii.

Un prim pas în realizarea proiectului este utilizarea dispozitivului ADT7420. Acesta este un cip cu senzor de temperatură care este integrat pe placa FPGA Nexys 4-DDR. Folosind funcționalitatea acestui senzor cu cod VHDL ne poate ajuta în construirea multor aplicații precum observarea modificării temperaturii zonei înconjurătoare sau implementarea unui sistem de control pentru a menține o temperatură fixă într-o locație dorită, cum ar fi o grădină. Cipul ADT7420 transmite datele semnale prin protocolul de comunicare I2C. Citirea temperaturii va fi apoi convertită în valorile respective Celsius și va fi transmisă utilizatorului prin intermediul modulului Bluetooth Pmod BT2 . Implementarea acestui proiect intenționează să demonstreze caracterul practic al tehnologiei în automatizarea sarcinilor obișnuite de zi cu zi, cum ar fi detectarea temperaturii.

3. Fundamentare teoretică

3.1 Tehnologii

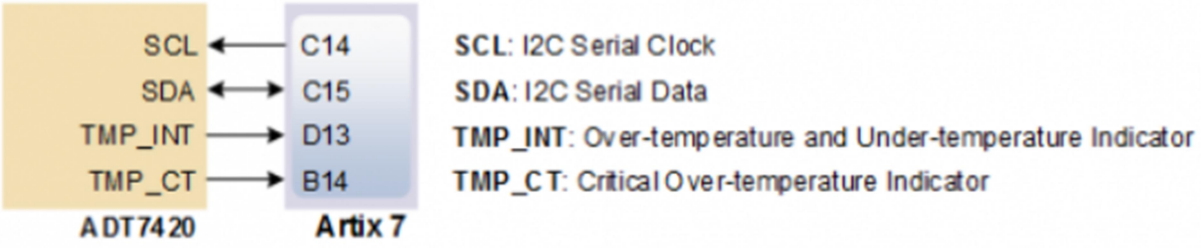
In cadrul acestui proiect s-a utilizat limbajul VHDL. VHDL (VHSIC HDL) este acronimul folosit pentru limbajul de descriere hardware pentru circuite integrate de foarte mare viteză.Este un limbaj de descriere hardware (HDD), destinat să descrie comportamentul și/sau arhitectura unui „modul” electronic logic, cu alte cuvinte o funcție logică combinatorie și/sau secvențială. Alături de Verilog, este cel mai utilizat limbaj pentru proiectarea sistemelor electronice digitale. Este unul dintre principalele instrumente de proiectare a circuitelor integrate moderne, aplicat cu succes în domeniul microprocesoarelor (DSP, acceleratoare grafice), al telecomunicațiilor (TV, telefoane mobile), al automobilelor (navigație, sisteme de control al stabilității) și altele.

3.2 Placa de dezvoltare FPGA Nexys 4 DDR

Placa Nexys 4 DDR este o platformă de dezvoltare de circuite digitale completă, bazată pe cel mai recent Artix-7™ Field Programmable Gate Array (FPGA) de la Xilinx. Cu FPGA de mare capacitate, memorii externe generoase și colecția de porturi USB, Ethernet și alte porturi, DDR-ul Nexys4 poate găzdui designuri variind de la circuite combinaționale introductive până la procesoare încorporate puternice. Mai multe periferice încorporate, inclusiv un accelerometru, senzor de temperatură, microfon digital MEM, un amplificator de difuzor și mai multe dispozitive I/O permit ca DDR-ul Nexys4 să fie utilizat pentru o gamă largă de modele fără a fi nevoie de alte componente.

3.3 Senzorul de temperatură ADT7420

ADT7420 este un senzor de temperatură digital de mare precizie oferind performanțe revoluționare peste o largă gamă industrială, găzduită într-un pachet LFCSP de 4 mm × 4 mm. Conține o referință de bandă internă, un senzor de temperatură și un ADC pe 16 biți pentru a monitoriza și digitiza temperatura la 0,0078°C rezoluţie. Rezoluția ADC, implicit, este setată la 13 biți (0,0625°C) și este un mod programabil de utilizator care poate fi schimbat prin interfața serială. ADT7420 este garantat să funcționeze peste tensiuni de alimentare de la 2,7 V până la 5,5 V. Funcționând la 3,3 V, curentul mediu de alimentare este de obicei de 210 µA. ADT7420 are un mod de oprire care alimentează dispozitivul și oferă un curent de oprire de obicei de 2,0 µA la 3,3 V. ADT7420 este evaluat pentru funcționare peste -40°C până la Interval de temperatură +150°C. Pinul A0 și Pinul A1 sunt disponibile pentru selectarea adresei. Pinul CT este o ieșire opendrain care devine activă atunci când temperatura depășește o limită de temperatură critică programabilă. Pinul INT este de asemenea o ieșire de scurgere deschisă care devine activă atunci când temperatura depășește o limită programabilă. Pinul INT și pinul CT poate funcționa în moduri de comparare și întrerupere a evenimentelor.

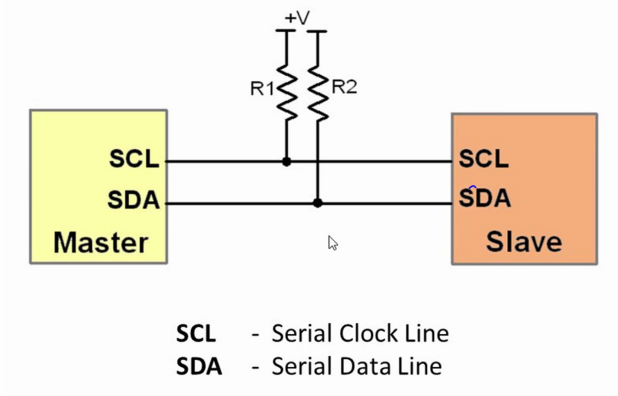


3.4 Comunicarea I2C

Comunicarea I2C este o parte integrantă a designului sistemului încorporat; mai ales pentru sistemele în care prioritatea nu este concentrată pe atingerea unor rate de ceas foarte mari. Deci, nu ar trebui să fie surprinzător faptul că I2C este utilizat pe scară largă în aplicații cu viteză redusă și cu costuri reduse. Tehnologia a fost concepută pentru prima dată în 1982 și, în ciuda faptului că este veche de trei decenii, popularitatea și aplicațiile I2C nu au scăzut la număr. Chiar și astăzi, protocolul I2C alimentează o mare majoritate a proiectelor de sisteme încorporate de nivel entry-level și de nivel mediu și este probabil să continue să facă acest lucru în viitorul apropiat.

Comunicarea sau protocolul I2C are un avantaj semnificativ , cum ar fi comunicația cu portul serial și SPI. I2C are urmatoarele avantaje: este flexibil, protocolul I2C acceptă comunicarea multi-master, multi-slave, ceea ce înseamnă că putem adăuga o mulțime de funcționalități designului. Mai mult de un circuit integrat master care controlează și comunică cu circuitele integrate slave poate accelera lucrurile și poate adăuga funcționalități sistemului încorporat. Un alt avantaj al protocolului I2C constă în capacitatea sa inerentă de a utiliza adresarea cu cip, astfel putem adăuga cu ușurință componente la magistrală, fără nicio complexitate și elimină necesitatea liniilor CS (chip select). Protocolul I2C nu complică designul. Este nevoie de doar două linii de semnal bidirecționale pentru a stabili comunicarea între mai multe dispozitive. În plus, numărul de pini este, de asemenea, scăzut. Pe de altă parte, are un mecanism mai bun de gestionare a erorilor. Pentru a îmbunătăți mecanismul de detectare și corectare a erorilor, protocolul I2C se bazează pe caracteristica ACK (Acknowledgement)/NACK (No Acknowledgement). În cele din urmă, protocolul I2C este adaptabil, în sensul că poate funcționa bine atât cu circuite integrate lente, cât și cu circuite integrate rapide.

Comunicarea I2C nu are prea multe dezavantaje. Faptul că protocolul este în uz de peste 30 de ani evidențiază acest fapt. Cu toate acestea, suferă de câteva limitări minore: din cauza adresei cu cip, există întotdeauna posibilitatea unui conflict de adrese, viteza este limitată, datorită designului cu drenaj deschis și necesită mai mult spațiu.

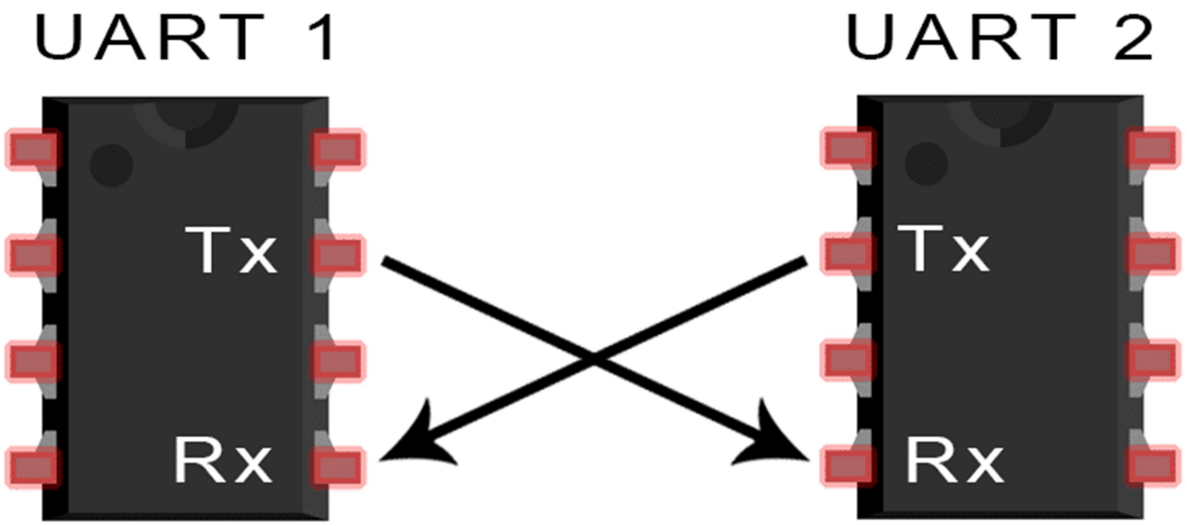


3.5 Modul Bluetooth HC-05

HC-05 este un modul care poate adăuga funcții Wireless în două direcții (full-duplex) pentru proiectele dumneavoastră. Puteți folosi acest modul pentru a comunica intre 2 microcontrolere ca de exemplu Arduino, sau cu orice dispozitiv ce folosește Bluetooth precum un telefon sau laptop. Există la momentul actual foarte multe aplicații de Android ce sunt deja disponibile, și vă pot ușura acest proces. Acest modul comunică prin intermediul interfeței USART la o rată baud de 9600. De asemenea se pot configura valorile predefinite, folosind comenzi specifice.

3.6 Comunicarea UART

Comunicare UART UART (Receptor/Transmițător Asincron Universal) este una dintre cele mai simple interfețe seriale asincrone, numită și ACIA (Adaptor de interfață de comunicare asincronă). Protocolul de comunicare UART este unul asincron deoarece nu este transmis niciun semnal de ceas prin linia de date seriale. Receptorul recunoaște valori binare individuale fără o linie comună de ceas. Interfața UART este formată din două părți: - un receptor (receptor - Rx) care convertește un flux de biți serial în date (cuvinte) paralele pentru microprocesor; - un transmițător (transmitter - Tx) care convertește datele paralele de la microprocesor într-un flux de biți serial pentru transmisie.

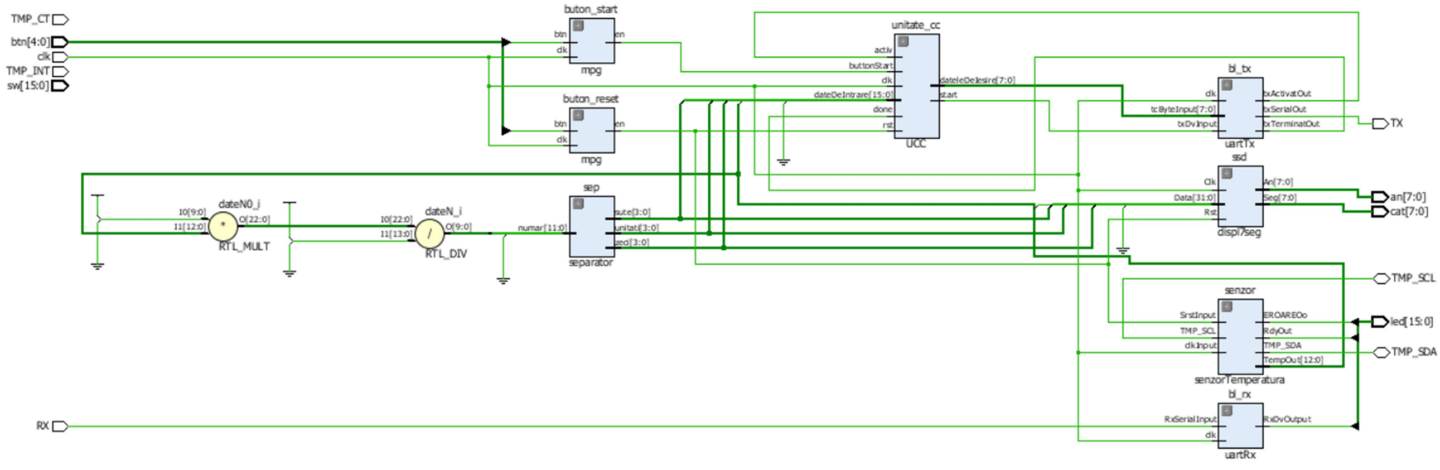


4. Proiectare şi implementare

Pentru implementarea proiectului am utilizat cele doua protocoale de comunicare seriale si anume I2C si UART. Comunicarea sincrona I2C am folosit-o pentru a transmite informatiile de la senzorul de temparatura la placa, in timp ce comunicarea asincrona UART a fost folosita pentru a trimite datele primite de placuta FPGA de la senzorul de temperatura, la modulul bluetooth, care le va redirectiona spre dispozitivul mobil (Samsung Galaxy A6+). Datele au fost afisate pe telefonul mobil prin intermediul terminalului BLE Terminal.

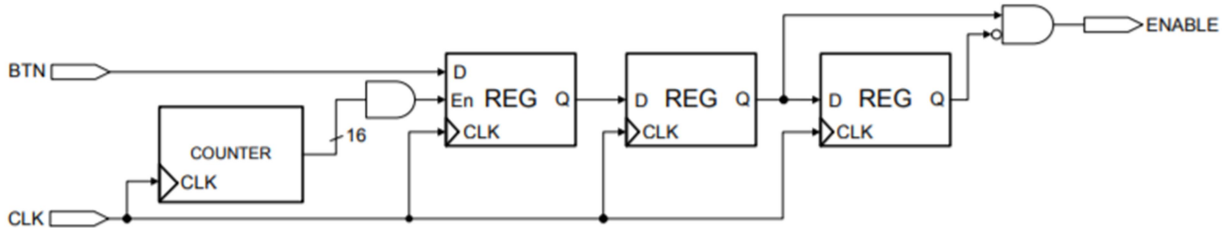
Proiectul contine urmatoarele entitati:

- Separator
- Mpg
- SenzorTemperatura
- UartRx
- UartTx
- UCC
- I2C
- SSD



Implementare MPG (MonoPulse Generator)

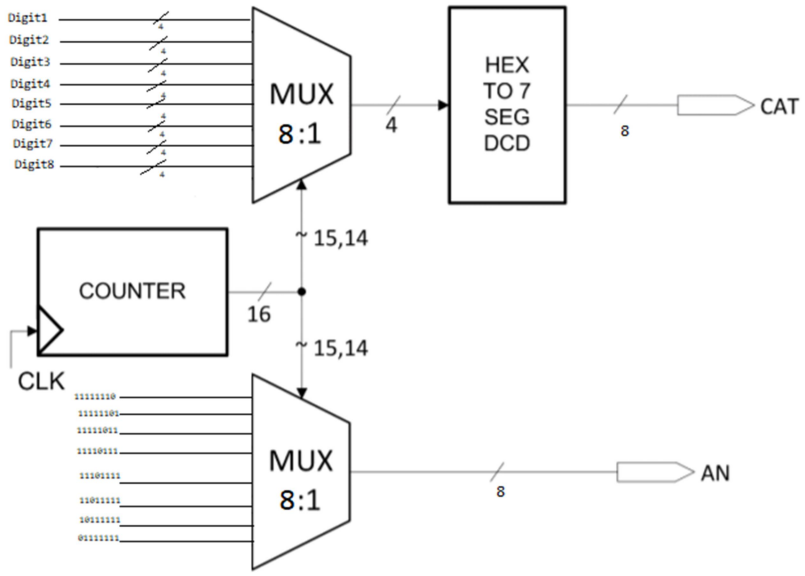
În acest proiect am avut nevoie să controlăm pas cu pas circuitele secvenţiale, pentru a urmări şi testa fluxul de date şi controlul circuitelor implementate. Din această cauză avem nevoie de un semnal, ENABLE, pentru a activa/valida frontul crescator al ceasului. Circuitul necesar, care activează un semnal ENABLE o singură dată la apăsarea unui buton, este prezentat în figura următoare.



Rolul primului registru, împreună cu contorul, este de a asigura robustețea la utilizarea butoanelor uzate, când pot apărea multiple activări ale semnalului ENABLE la apăsarea unui buton. În funcție de uzură, pot fi necesari mai multi biți de contor (17-20+) pe care să se aplice un ŞI logic, astfel încât să mărească domeniul de eșantionare al butonului. Pe plan intern, componentele diagramei MPG, registre/flip-flops, contor, porti SI, vor fi descrise comportamental prin declararea semnalelor necesare, respectiv procesele si atribuirile concurente in arhitectura MPG.

Implementare SSD

Pentru a afișa 8 cifre diferite pe SSD, este necesar să implementam un circuit care trimite cifrele pe semnalele catodice ale SSD-ului in conformitate cu diagrama de timp a SSD. Perioada maximă de reîmprospătare este calculată astfel încât ochiul uman să nu perceapă pornirea și oprirea succesivă a fiecărei cifre de pe SSD. Aceasta face o afișare ciclică a numerelor (la un moment dat este afișată doar o cifră, dar ochiul nu percepe acest aspect).



Implementare UartRx

Entitatea UartRx reprezinta un automat de stari finite. Arhitectura entitatii este definită de tipul t_SM_Main, care are cinci stări: idle (starea in care se afla automatul initial), startBitsRXs (detectarea bitului de start) , dataBitsRxs (starea in care se citesc pe rand cei 8 biti), rxStopB (citirea bitului de stop), cleanUp (starea de reinitializare).

Mașina de stare pornește în modul inactiv fără biți de date primiți. Starea in care se trezeste automatul este idle in care se initializeaza semnalele rxDvR, clkCountR, indexBitRXs . In aceasta stare verificam daca primul bit primit este '0', adica bitul de start. In caz afirmativ trecem automat in starea urmatoare, iar in caz contrar ramanem in acesta stare pana la receptionareabitului de start. Urmatoarea stare startBitsRXs reprezinta un mecanism de sincronizare cu slave-ul, cu alte cuvinte ne precizeaza cat trebuie sa asteptam ca sa primim date de la slave(putem sa-l consideram un divizor de frecventa) . In starea dataBitsRxs se așteaptă ca un octet de date de 8 biți să fie trimis de la dispozitivul gazdă (bufferul de transmisie). Dacă nu mai sunt octeți în așteptare în buffer-ul de transmisie, acesta va trece in starea de oprire unde va astepta primirea bitului de stop. Cand este primit bitul de stop se activeaza semnalul rxDvR (ceea ce ne spune ca s-a realizat cu succes conexiunea dintre telefon si placa Nexys) si atomatul va trece in starea de cleanUp, unde va astepta o perioada ceas dupa care va trece in starea idle.

Implementare UartTx

La fel ca si componenta RX, UartTX este tot un automat de stari finite, cu ajutorul careia trimit datele de la placuta spre telefon. Principiul de functionare este asemanator cu cel de RX cu mentionarea ca are nevoie de semnalul de divice de la acesta pentru a-si incepe activitatea. Daca acesta este 1 logic, masina de stare trece in txStartBit unde asteapta bitul de start care va seta semnalul tx_active cu 1. In starea txStartBit, vom astepta mai multe perioade clock, tot din cauza sincronizarii (la fel cum am precizat si la RX), dupa care vom trece in txDataBit, unde vom astepta pana cand cei 8 biti de date vor fi trasnmisi spre telefon. Dupa transmiterea datelor, vom trece automat in starea de stop, iar apoi in cleanup, stari asemanatoare cu cele de la RX.

Implementare Unitatea de Comanda

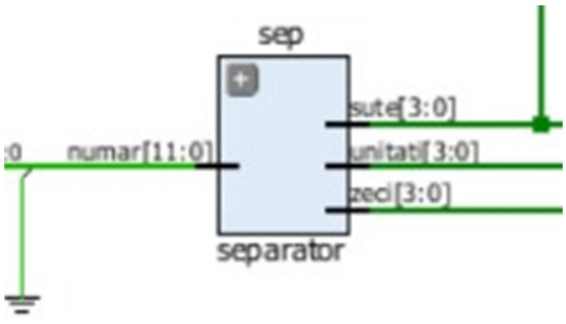
Unitatea de comanda este un atomat de stare format din trei procese: unul pentru resetarea acestuia, care il aduce in starea initiala, un proces pentru stabilirea starii in care ne aflam si ultimul process pentru atribuirea semnalelor in functie de starea in care suntem. In al doilea proces, starea in care se afla automatul este cea de inceput, in care se asteapta primirea bitului de start pentru a trece in starea urmatoare. Din starea de incData, trecem direct in starea oct1, atribuind inainte semnalelor oc1 si oc2 datele de intrae. In starea oct1 atribuim semnalului de iesire valoarea primului octet, dupa care trecem in starea de transmisie. Aici verificam daca automatul si-a terminat activitatea (done=1 si active=0) . In caz afirmativ trecem in starea oct2 care are acelsi comportament ca oct1. Aceasta este entitatea care face legatura dintre entitatiel RX si TX, si care se asigura de receptionarea si trimiterea corecta a datelor.

Implementare senzor de temperatura

Când ADT7420 este pornit, acesta se află într-un mod care poate fi utilizat ca un simplu senzor de temperatură fără nicio configurație inițială. În mod implicit, registrul adresei dispozitivului indică registrul MSB de temperatură, astfel încât o citire de doi octeți fără a specifica un registru va citi valoarea registrului de temperatură din dispozitiv. Primul octet citit înapoi va fi octetul cel mai semnificativ (MSB) al datelor de temperatură, iar al doilea va fi octetul cel mai puțin semnificativ (LSB) al datelor. Acești doi octeți formează un număr întreg pe 16 biți in complement fata de doi . Dacă rezultatul este deplasat la dreapta cu 3 trei biți și înmulțit cu 0,0625, valoarea rezultată în virgulă mobilă va fi o citire a temperaturii în grade Celsius.

Implementare separator

Pentru o afisare mai frumoasa a temperaturii, am folosit un separator (convertor BCD) care ne separa numarul nostru in unitati, zeci, sute si mii. Pentru acest separator am declarat o variabila bcd : UNSIGNED (15 downto 0) si am implementat urmatorul algoritm : intr-un for de la zero la unsprezece , adica numarul de biti de pe intrare, daca variabila noastra bcd(3 downto 0) este mai mare decat patru, atunci o sa adunam un trei la aceasta valoare. Daca bcd(7 downto 4) este mai mare decat 4, atunci o sa adaugam un trei si la aceasta valoare, la fel si pentru bcd(11 downto 8). Dupa acestea, bcd va primi bcd(14 downto 0) concatenat cu numarul nostru, dupa care vom shifta numarul la stanga cu o pozitie. La final unitatile o sa primeasca STD_LOGIC_VECTOR(bcd(3 downto 0)), zecile o sa devina STD_LOGIC_VECTOR(bcd(7 downto 4)), sutele STD_LOGIC_VECTOR(bcd(11 downto 8)), iar miile o sa primmeasca valoarea STD_LOGIC_VECTOR(bcd(15 downto 12)).



4.1 Manual de utilizare

Pentru a folosi acest proiect, utilizatorul trebuie sa indeplineasca urmatoarele cerinte:sa dispuna de o placuta Nexys 4 DDR, un calculator de preferabil cu un sistem de operare Windows, in care sa fie instalat programul Vivado in versiunea 2016.4 (sau orice versiune mai noua), un dispozitiv mobil cu un sistem de operare Android, care sa detina functia Bluetooth si sa aiba instalat terminalul BLE, dar si de un modul Bluetooth HC-05.

Utilizatorul va incepe prin conectarea modulului Bluetooth HC-05 la placuta FPGA Nexys 4 DDR si conectarea placutei la calculator. Urmatorul pas este incarcarea codului din Vivado pe placuta. Din acest moment temperatura citita de catre senzorul ADT7420 poate fi vizualizata pe afisorul placutei. Pentru a vedea temperatura si pe dispozitivul mobil trebuie sa activam functia Bluetooth al acestuia, dupa care ar trebui sa detectam cu usurinta modulul. Dupa ce s-a realizat conexiunea dintre placuta si telefon prin intermediul modulului Bluetooth, trebuie sa pornim pe dispozitivul mobil terminalul BLE. Din acest moment la apasarea butonului de start de pe placuta, ar trebui sa putem vizualiza temperatura din terminalul telefonului.

5. Rezultate experimentale

5.1 Instrumentele de proiectare utilizate

Proiectul a fost implementat in limbajul VHDL, in mediul Vivado. Acesta este un limbaj de descriere hardware, destinat să descrie comportamentul și/sau arhitectura unui modul electronic logic, fiind unul dintre principalele instrumente de proiectare a circuitelor integrate moderne, aplicat cu succes în domeniul microprocesoarelor (DSP, acceleratoare grafice), al telecomunicațiilor (TV, telefoane mobile), al automobilelor (navigație, sisteme de control al stabilității) și altele.

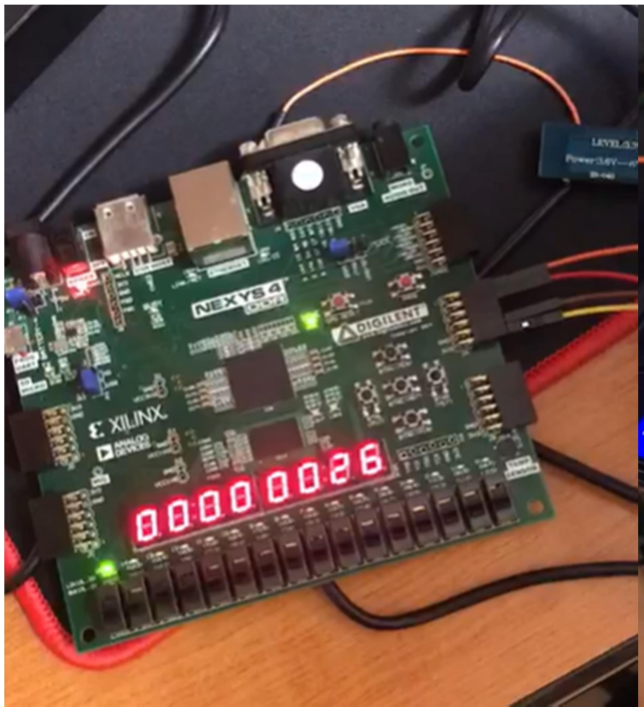
Vivado Design Suite este o suită de software produsă de Xilinx pentru sinteza și analiza modelelor de limbaj de descriere hardware (HDL), înlocuind Xilinx ISE cu caracteristici suplimentare pentru dezvoltarea sistemului pe un cip și sinteza la nivel înalt. Vivado reprezintă o rescrire și o re – gândire a întregului flux de proiectare (comparativ cu ISE). La fel ca și versiunile ulterioare ale ISE, Vivado include simulatorul logic încorporat. Vivado introduce, de asemenea, sinteza la nivel înalt, cu un lanț de instrumente care convertește codul C în logică programabilă. Vivado a fost introdus în aprilie 2012 și este un mediu de proiectare integrat (IDE) cu instrumente la nivel de sistem IC construite pe un model de date scalabil partajat și un mediu comun de depanare.

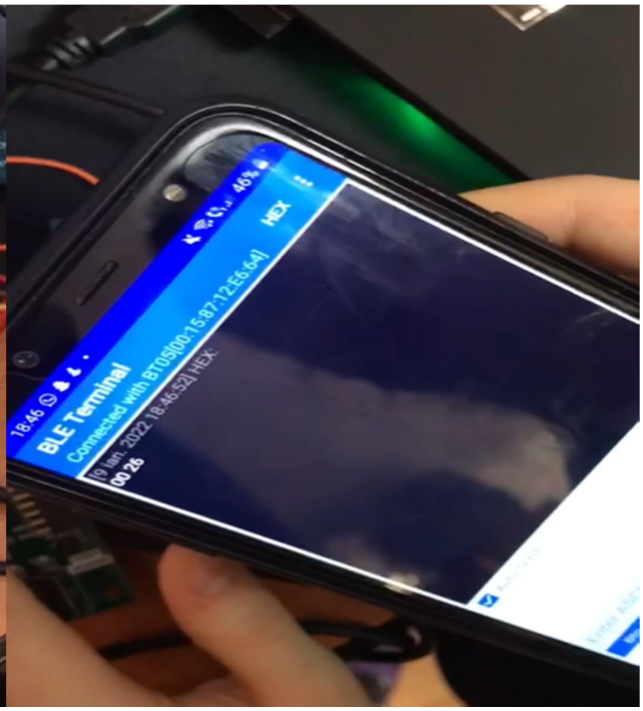
5.2 Informatii din rapoartele de implementare sub formă tabelară

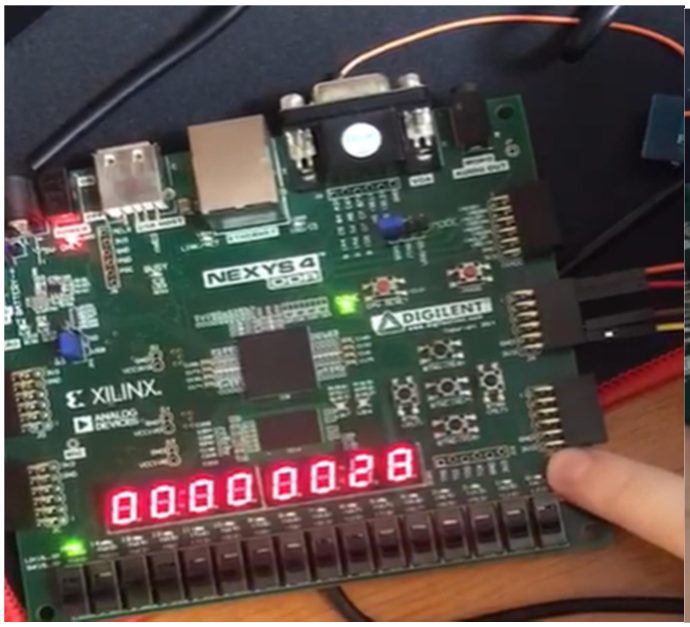
Componenta folosita	Numar
butoane	2
blocuri logice	6

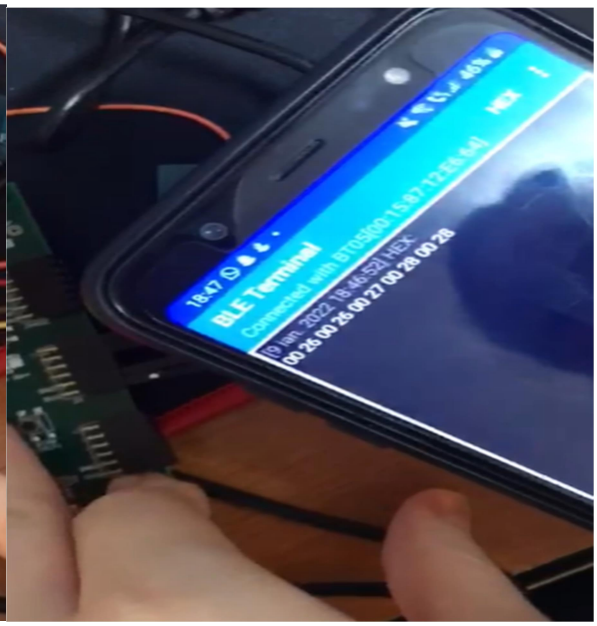
5.3 Procedura de testare

Proiectul a fost testat fizic, chiar pe placuta.









Dupa cum se poate observa si in imagini, temperatura este afisata cu succes atat pe ecranul telefonului mobil cat si pe afisorul placutei. Pentru a testa schimbarea temperaturii, placuta a fost plasata in apropierea unei surse de caldura, unde s-a constatat ca temperatura afisata a crescut constant. Analog, la apropierea placutei de un obiect cu o temperatura scazuta, s-a putut observa micsorarea treptata a temperaturii pe placuta, dar si pe dispozitivul mobil.

5.4 Dificultăţile întâlnite

La realizarea proiectului am intampinat si dificultati precum: datele trimise spre dispozitivul mobil nu corespundeau cu cele de pe afisorul placutei, aceasta problema a fost rezolvata prin revizuirea codului si corectarea greselilor de implementare, o alta problema intampinata a fost trimiterea datelor spre telefon fara oprire, temperatura fiind greu de distins din terminal, aceasta problema a fost rezolvata prin trimiterea temperaturii, doar dupa apasarea unui buton.

6. Concluzii

In concluzie, acest proiect utilizeaza limbajul VHDL si implementeaza pe placa Digilent Nexys 4 DDR unui sistem care sa permita masurarea temperaturii si transmiterea acesteia unui dispozitiv mobil, folosind senzorul de temperatură ADT7420 si modulul Bluetooth HC-05. Ca dezavantaj al proiectului ar fi costul ridicat al componentelor necesare asamblarii (in special placuta Nexys 4 DDR), deoarece exista variante mai ieftine de proiectare pentru aceste functionalitati.

O dezvoltare ulterioara ar fi realizarea unui dispozitiv care afiseaza ora si data curenta, pe langa temperatura , dar si inglobarea proiectului intr-un disozitiv care monitorizeaza temperatura dintr-o incapere, care cu ajutorul unei aplicatii pe telefonul mobil sa poata sa regleze temperatura camerei dupa bunul plac.

7. Bibliografie

Florin Oniga, Mihai Negru , ARHITECTURA CALCULATOARELOR - Îndrumător de laborator, Editura UTPRESS Cluj-Napoca, 2019

https://www.wikipedia.org/

https://www.analog.com/media/en/technical-documentation/data-sheets/adt7420.pdf

https://digilent.com/reference/programmable-logic/nexys-4-ddr/reference-manual

Anexa A

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


entity placuta is
Port
 (
 signal clk:in std_logic;
 signal btn:in std_logic_vector(4 downto 0);
 signal sw:in std_logic_vector(15 downto 0);
 signal TMP_INT:in std_logic;
 signal TMP_CT:in std_logic;
 signal cat:out std_logic_vector(7 downto 0);
 signal an:out std_logic_vector(7 downto 0);
 signal led:out std_logic_vector(15 downto 0);
 signal TMP_SCL:inout std_logic;
 signal TMP_SDA:inout std_logic;
 signal RX:in std_logic;
 signal TX:out std_logic
  );
end placuta;

architecture Behavioral of placuta is

signal TSR:std_logic_vector(23 downto 0):=(others=>'0');
signal buttonReset:std_logic;
signal semnalRead:std_logic;
signal buttonStart:std_logic;
signal counter:INTEGER:=0;
signal ena:std_logic:='0';
signal unitati : STD_LOGIC_VECTOR (3 downto 0);
signal zeci : STD_LOGIC_VECTOR (3 downto 0);
signal sute: STD_LOGIC_VECTOR (3 downto 0);
signal mii : STD_LOGIC_VECTOR (3 downto 0);
signal date:std_logic_vector(12 downto 0);
signal afisor:std_logic_vector(31 downto 0);
signal intVal : integer;
signal dateN : integer;
signal final : std_logic_vector(12 downto 0);
signal final_1 : std_logic_vector(15 downto 0);
signal start:std_logic;
signal activ:std_logic;
signal mesaj:std_logic_vector(7 downto 0);
signal rx8:std_logic_vector(7 downto 0);
signal numarator:integer:=0;
```

```
signal done:std_logic;
begin

sep: entity WORK.separator port map (
          numar => final(11 downto 0),
         unitati => unitati,
         zeci  => zeci,
         sute => sute,
         mii  => mii
);



buton_start:entity WORK.mpg port map
(
btn=>btn(0),
clk=>clk,
en=>buttonStart
);

buton_reset:entity WORK.mpg port map
(
btn=>btn(1),
clk=>clk,
en=>buttonReset
);

senzor:entity WORK.senzorTemperatura port map
(
TMP_SCL=>TMP_SCL,
                        TMP_SDA=>TMP_SDA,
                        TempOut =>date,
                        RdyOut =>led(15),
                        EROAREOo =>led(1),
                        clkInput=>clk,
                        SrstInput=>buttonReset
);

dateN <=  625 * to_integer(unsigned (date)) /10000;
final<= date(12) & std_logic_vector(to_unsigned(dateN, 12));

afisor<="0000000000000000000"& date(12) &sute & zeci &unitati;
final_1 <= "000" & date(12) &sute & zeci &unitati;

ssd:entity WORK.displ7seg port map
   (
   Clk=>Clk,
         Rst=>buttonReset,
         Data=>afisor,
```

```
        An=>an,
        Seg=>cat
    );

--data de la tel la placuta
bl_rx:entity WORK.uartRx
  generic map
    (
    clkPerBit => 10416
    )
    port map
        (
       Clk=>clk,
       RxSerialInput=>RX,
       RxDvOutput=>led(0),
       RxByteOutput=>rx8
        );

--data spre telefon

bl_tx:entity WORK.uartTx
    generic map
    (
    clkPerBit => 10416
    )
    port map
        (
       Clk=>Clk,
       txDvInput=>start,
       tcByteInput=>mesaj,--x"41",
       txActivatOut=>activ,
       txSerialOut=>TX,
       txTerminatOut=>done
        );

    unitate_cc:entity WORK.UCC
    port map
    (
    clk=>clk,
    rst=>buttonReset,
    buttonStart=>buttonStart,
    dateDeIntrare=> final_1,
    activ=>activ,
    done=>done,
    dateleDeIesire=>mesaj,
    start=>start
    );

end Behavioral;
```

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

entity separator is
    Port ( numar : in  STD_LOGIC_VECTOR (11 downto 0);
           unitati : out  STD_LOGIC_VECTOR (3 downto 0);
           zeci : out  STD_LOGIC_VECTOR (3 downto 0);
           sute : out  STD_LOGIC_VECTOR (3 downto 0);
           mii : out  STD_LOGIC_VECTOR (3 downto 0)
          );
end separator;
architecture SEPARARE of separator is
begin
bcd1: process(numar)
  variable temp : STD_LOGIC_VECTOR (11 downto 0);
  variable bcd : UNSIGNED (15 downto 0) := (others => '0');
  begin
    bcd := (others => '0');
    temp(11 downto 0) := numar;
    for i in 0 to 11 loop
      if bcd(3 downto 0) > 4 then
        bcd(3 downto 0) := bcd(3 downto 0) + 3;
      end if;
      if bcd(7 downto 4) > 4 then
        bcd(7 downto 4) := bcd(7 downto 4) + 3;
      end if;
```

```
      if bcd(11 downto 8) > 4 then
        bcd(11 downto 8) := bcd(11 downto 8) + 3;
      end if;
      bcd := bcd(14 downto 0) & temp(11);
      temp := temp(10 downto 0) & '0';
    end loop;
    unitati <= STD_LOGIC_VECTOR(bcd(3 downto 0));
    zeci <= STD_LOGIC_VECTOR(bcd(7 downto 4));
    sute <= STD_LOGIC_VECTOR(bcd(11 downto 8));
    mii <= STD_LOGIC_VECTOR(bcd(15 downto 12));
  end process bcd1;
end SEPARARE;

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

entity mpg is
 Port ( signal btn:in std_logic;
        signal clk:in std_logic;
        signal en:out std_logic
 );
end mpg;

architecture Behavioral of mpg is
signal count_int1: std_logic_vector(31 downto 0) :=x"00000000";
signal Q1 : std_logic;
signal Q2 : std_logic;
```

```
signal Q3 : std_logic;
begin
en <= Q2 AND (not Q3);
process (clk)
begin
if clk='1' and clk'event then
count_int1 <= count_int1 + 1;
end if;
end process;
process (clk)
begin
if clk'event and clk='1' then
if count_int1(15 downto 0) = "1111111111111111" then
Q1 <= btn;
end if;
end if;
end process;
process (clk)
begin
if clk'event and clk='1' then
Q2 <= Q1;
Q3 <= Q2;
end if;
end process;

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.math_real.all;
use work.TWIUtils.ALL;



entity senzorTemperatura is
Generic (fregvClock : natural := 100);
	Port (
		TMP_SCL : inout STD_LOGIC;
		TMP_SDA : inout STD_LOGIC;
		TempOut : out STD_LOGIC_VECTOR(12 downto 0);
		RdyOut : out STD_LOGIC;
		EROAREOo : out STD_LOGIC;
		clkInput : in STD_LOGIC;
		SrstInput : in STD_LOGIC
	);
end senzorTemperatura;

architecture Behavioral of senzorTemperatura is



	component TWICtl
  generic
```

```
  (
                fregvClock : natural := 50;
                aSlaveUnblock : boolean := false
        );
        port (
                imsg : in STD_LOGIC;
                stbIi : in STD_LOGIC;
                A_I : in  STD_LOGIC_VECTOR (7 downto 0);
                D_I : in  STD_LOGIC_VECTOR (7 downto 0);
                D_O : out  STD_LOGIC_VECTOR (7 downto 0);
                doneOutput : out  STD_LOGIC;
    EROAREOo : out  STD_LOGIC;
                errTp : out error_type;
                CLK : in std_logic;
                SRST : in std_logic;
                SDA : inout std_logic;
                SCL : inout std_logic
        );
end component;
        constant IWR : std_logic := '0';
        constant IRD : std_logic := '1';

  constant ADT7420ID : std_logic_vector(7 downto 0)      := x"CB";
        constant ADT7420RID : std_logic_vector(7 downto 0)     := x"0B";
        constant ADT7420RRESET : std_logic_vector(7 downto 0)   := x"2F";
  constant ADT7420ADDRess : std_logic_vector(7 downto 1)    := "1001011";
        constant ADT7420RdTEMP : std_logic_vector(7 downto 0)    := x"00";
  constant RETRY_COUNT : NATURAL := 10;
        constant DELAY : NATURAL := 1;
```

```
        constant CYCLESDelay : NATURAL := natural(ceil(real(DELAY*1000*fregvClock)));


  type state_type is (
                idle,
                IniRegSt,
                stInitData,
                stRetry,
                readssTempR,
                readssTempD1,
                readssTempD2,
                stError
                );
  signal state, myState : state_type;



  constant NO_OF_INIT_VECTORS : natural := 3;
        constant DATA_WIDTH : integer := 1 + 8 + 8;
        constant ADDR_WIDTH : natural := natural(ceil(log(real(NO_OF_INIT_VECTORS), 2.0)));


        type TempSensInitMap_type is array (0 to NO_OF_INIT_VECTORS-1) of std_logic_vector(DATA_WIDTH-1
downto 0);
        signal TempSensInitMap: TempSensInitMap_type := (
                IRD & x"0B" & x"CB",
                IWR & x"2F" & x"00",
                IRD & x"0B" & x"CB"
                );
        signal initEn : std_logic;
        signal wordini: std_logic_vector (DATA_WIDTH-1 downto 0);
        signal twiMsg, twiStb, DoneTWI, errTwi : std_logic;
```

```
        signal iniA : natural range 0 to NO_OF_INIT_VECTORS := 0;
        signal twiDi, dotwi, twiAddr : std_logic_vector(7 downto 0);
        signal waitCnt : natural range 0 to CYCLESDelay := CYCLESDelay;
        signal waitCntEn : std_logic;
        signal retryCnt : natural range 0 to RETRY_COUNT := RETRY_COUNT;
        signal tempReg : std_logic_vector(15 downto 0) := (others => '0');
        signal retryCntEn : std_logic;
  signal fReady : boolean := false;
begin


TempOut <= tempReg(15 downto 3);

EROAREOo <= '1' when state = stError else
                        '0';
RdyOut <= '1' when fReady else
                        '0';

ctlCtl : TWICtl
        generic map (
                aSlaveUnblock => true,
                fregvClock => 100
        )
        port map (
                imsg => twiMsg,
                stbIi => twiStb,
                A_I => twiAddr,
                D_I => twiDi,
                D_O => dotwi,
```

```
                doneOutput => DoneTWI,
EROAREOo => errTwi,
                errTp => open,
                CLK => clkInput,
                SRST => SrstInput,
                SDA => TMP_SDA,
                SCL => TMP_SCL
        );

        wordini <= TempSensInitMap(iniA);
        iniA_CNT: process (clkInput)
        begin
                if Rising_Edge(clkInput) then
                        if (state = idle or iniA = NO_OF_INIT_VECTORS) then
                                iniA <= 0;
                        elsif (initEn = '1') then
                                iniA <= iniA + 1;
                        end if;
                end if;
        end process;


        Wait_CNT: process (clkInput)
        begin
                if Rising_Edge(clkInput) then
                        if (waitCntEn = '0') then
                                waitCnt <= CYCLESDelay;
                        else
                                waitCnt <= waitCnt - 1;
```

```
                end if;
        end if;
end process;

Retry_CNT: process (clkInput)
begin
        if Rising_Edge(clkInput) then
                if (state = idle) then
                        retryCnt <= RETRY_COUNT;
                elsif (retryCntEn = '1') then
                        retryCnt <= retryCnt - 1;
                end if;
        end if;
end process;

TemperatureReg: process (clkInput)
variable temp : std_logic_vector(7 downto 0);
begin
        if Rising_Edge(clkInput) then
                if (state = readssTempD1 and DoneTWI = '1' and errTwi = '0') then
                        temp := dotwi;
                end if;
                if (state = readssTempD2 and DoneTWI = '1' and errTwi = '0') then
                        tempReg <= temp & dotwi;
                end if;
        end if;
end process;
```

```
        ReadyFlag: process (clkInput)
        begin
                if Rising_Edge(clkInput) then
                        if (state = idle or state = stError) then
                                fReady <= false;
                        elsif (state = readssTempD2 and DoneTWI = '1' and errTwi = '0') then
                                fReady <= true;
                        end if;
                end if;
        end process;


synicppp: process (clkInput)
 begin
   if (clkInput'event and clkInput = '1') then
     if (SrstInput = '1') then
       state <= idle;
     else
       state <= myState;
     end if;
   end if;
 end process;

 OUTPUT_DECODE: process (state, wordini, DoneTWI, errTwi, dotwi, retryCnt, waitCnt, iniA)
 begin
                twiStb <= '0';
                waitCntEn <= '0';
                twiMsg <= '0';
                twiDi <= "--------";
```

```
                retryCntEn <= '0';
                twiAddr <= ADT7420ADDRess & '0';
                initEn <= '0';



                case (state) is
        when idle =>


        when IniRegSt =>
            twiStb <= '1';
                                twiMsg <= '1';
                                twiAddr(0) <= IWR;
                                twiDi <= wordini(15 downto 8);
                        when stInitData =>
          twiStb <= '1';
                                twiAddr(0) <= wordini(wordini'high);
                                twiDi <= wordini(7 downto 0);
                                if (DoneTWI = '1' and
                                        (errTwi = '0' or (wordini(16) = IWR and wordini(15 downto 8) =
ADT7420RRESET)) and
                                        (wordini(wordini'high) = IWR or dotwi = wordini(7 downto 0))) then
                                        initEn <= '1';
                                end if;
                        when stRetry=>
                                if (retryCnt /= 0) then
                                        waitCntEn <= '1';
                                        if (waitCnt = 0) then
                                                retryCntEn <= '1';
                                        end if;
```

```
                    end if;

                when readssTempR =>
                  twiMsg <= '1';
                  twiDi <= ADT7420RdTEMP;
                    twiStb <= '1';
                    twiAddr(0) <= IWR;
                when readssTempD1 =>
                    twiStb <= '1';
                    twiAddr(0) <= IRD;
                when readssTempD2 =>
      twiStb <= '1';
                    twiAddr(0) <= IRD;

                when stError =>
                    null;
            end case;

end process;

NEXT_STATE_DECODE: process (state, DoneTWI, errTwi, wordini, dotwi, retryCnt, waitCnt)
begin
  myState <= state;

  case (state) is
    when idle =>
      myState <= IniRegSt;

    when IniRegSt =>
```

```
        if (DoneTWI = '1') then
                                        if (errTwi = '1') then
                                                myState <= stRetry;
                                        else
                                                myState <= stInitData;
                                        end if;
                                end if;

                        when stInitData =>
        if (DoneTWI = '1') then
                                        if (errTwi = '1') then
                                                myState <= stRetry;
                                        else
                                                if (wordini(wordini'high) = IRD and dotwi /= wordini(7 downto 0))
then
                                                        myState <= stRetry;
                                                elsif (iniA = NO_OF_INIT_VECTORS-1) then
                                                        myState <= readssTempR;
                                                else
                                                        myState <= IniRegSt;
                                                end if;
                                        end if;
                                end if;
                        when stRetry =>
                                if (retryCnt = 0) then
                                        myState <= stError;
                                elsif (waitCnt = 0) then
                                        myState <= IniRegSt;
                                end if;
```

```
                        when readssTempR =>
  if (DoneTWI = '1') then
                                    if (errTwi = '1') then
                                            myState <= stError;
                                    else
                                            myState <= readssTempD1;
                                    end if;
                            end if;
                        when readssTempD1 =>
  if (DoneTWI = '1') then
                                    if (errTwi = '1') then
                                            myState <= stError;
                                    else
                                            myState <= readssTempD2;
                                    end if;
                            end if;
                        when readssTempD2 =>
  if (DoneTWI = '1') then
                                    if (errTwi = '1') then
                                            myState <= stError;
                                    else
                                            myState <= readssTempR;
                                    end if;
                            end if;

when stError =>
                            null;
```

```
        when others =>
          myState <= idle;
      end case;
    end process;

end Behavioral;



package TWIUtils is
  type busState_type is (busUnknown, busBusy, busFree);
  type error_type is (errArb, errNAck);
end TWIUtils;

package body TWIUtils is
end TWIUtils;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.math_real.all;
use work.TWIUtils.ALL;



entity TWICtl is

            generic (
                        fregvClock : natural := 50;
                        aSlaveUnblock : boolean := false
```

```
        );
        port (
                imsg : in STD_LOGIC;
                stbIi : in STD_LOGIC;
                A_I : in  STD_LOGIC_VECTOR (7 downto 0);
                D_I : in  STD_LOGIC_VECTOR (7 downto 0);
                D_O : out  STD_LOGIC_VECTOR (7 downto 0);
                doneOutput : out  STD_LOGIC;
    EROAREOo : out  STD_LOGIC;
                errTp : out error_type;
                CLK : in std_logic;
                SRST : in std_logic;
                SDA : inout std_logic;
                SCL : inout std_logic
        );
end TWICtl;

architecture Behavioral of TWICtl is
        attribute fsm_encoding: string;
        constant FSCL : natural := 400_000;
        constant TIMEOUT : natural := 10;
        constant cicTS : natural :=
                natural(ceil(real(fregvClock*1_000_000/FSCL)));
        constant TIMEOUT_CYCLES : natural :=
                natural(ceil(real(fregvClock*TIMEOUT*1_000)));
  type state_type is (idle, startSs, readss, writeSs, stError, stop,
                stSAck, MackSttt, stMNAckStop, MNAckStartSs, stopError);
  signal state, myState : state_type;
        attribute fsm_encoding of state: signal is "gray";
```

```
        signal dSda, ddSda, dScl : std_logic;
        signal busState : busState_type := busUnknown;
        signal startFi, stopFi : std_logic;
        signal sdaSync, sclSync : std_logic_vector(2 downto 0);
  signal busFreeCnt, sclCnt : natural range cicTS downto 0 := cicTS;
        signal errTypeR, errType : error_type;
        signal timeOutCnt : natural range TIMEOUT_CYCLES downto 0 := TIMEOUT_CYCLES;
        signal waitSlaveW, lostArb : std_logic;
        signal dataByte, loadByte, adresaCurenta : std_logic_vector(7 downto 0);
        signal stareSub : std_logic_vector(1 downto 0) := "00";
        signal latchData, latchAddr, iDone, iErr, iSda, iScl, shiftBit, dataBitOut, rwBit, adrDataNN : std_logic;
        signal resetIni : std_logic := '0';
        signal sdaR, rScl : std_logic := '1';
        signal bitCount : natural range 0 to 7 := 7;
begin



mySync: process(CLK)
  begin
    if Rising_Edge(CLK) then
                        sdaSync(0) <= SDA;
                        sdaSync(1) <= sdaSync(0);
                        sdaSync(2) <= sdaSync(1);
                        sclSync(0) <= SCL;
                        sclSync(1) <= sclSync(0);
                        sclSync(2) <= sclSync(1);
    end if;
  end process;
```

```
            dSda <= sdaSync(1);
    dScl <= sclSync(1);
            ddSda <= sdaSync(2);
            stopFi <= dScl and dSda and not ddSda;
            startFi <= dScl and not dSda and ddSda;



TWISTATE: process(CLK)
  begin
    if Rising_Edge(CLK) then
                                if (resetIni = '1') then
                                        busState <= busUnknown;
      elsif (startFi = '1') then
        busState <= busBusy;
                                elsif (busFreeCnt = 0) then
        busState <= busFree;
      end if;
    end if;
  end process;


TBUF_CNT: process(CLK)
  begin
    if Rising_Edge(CLK) then
      if (dSCL = '0' or dSDA = '0' or resetIni = '1') then
        busFreeCnt <= cicTS;
      elsif (dSCL = '1' and dSDA = '1') then
        busFreeCnt <= busFreeCnt - 1;
      end if;
    end if;
```

```
  end process;

  lostArb <=     '1' when (dSCL = '1' and dSDA = '0' and sdaR = '1') else
              '0';
  waitSlaveW <=   '1' when (dSCL = '0' and rScl = '1') else
              '0';

  RST_PROC: process (CLK)
  begin
    if Rising_Edge(CLK) then
      if (state = idle and SRST = '0') then
        resetIni <= '0';
      elsif (SRST = '1') then
        resetIni <= '1';
      end if;
    end if;
  end process;

sclcount: process (CLK)
            begin
                  if Rising_Edge(CLK) then
                        if (sclCnt = 0 or state = idle) then
                              sclCnt <= cicTS/4;
                        elsif (waitSlaveW = '0') then
                              sclCnt <= sclCnt - 1;
```

```
                end if;
            end if;
        end process;


UnblockTimeout: if aSlaveUnblock generate
timeo_c: process (CLK)
        begin
            if Rising_Edge(CLK) then
                if (state /= idle or busState = busFree or ((ddSda xor dSda) = '1')) then
                    timeOutCnt <= TIMEOUT_CYCLES;
                else
                    timeOutCnt <= timeOutCnt - 1;
                end if;
            end if;
        end process;
end generate;


shRegDataByte: process (CLK)
        begin
            if Rising_Edge(CLK) then
                if ((latchData = '1' or latchAddr = '1') and sclCnt = 0) then
                    dataByte <= loadByte;
                    bitCount <= 7;
                    if (latchData = '1') then
                        adrDataNN <= '0';
                    else
                        adrDataNN <= '1';
```

```
                end if;
            elsif (shiftBit = '1' and sclCnt = 0) then
                dataByte <= dataByte(dataByte'high-1 downto 0) & dSDA;
                bitCount <= bitCount - 1;
            end if;
        end if;
    end process;

    loadByte <= A_I when latchAddr = '1' else
                D_I;
    dataBitOut <= dataByte(dataByte'high);

    D_O <= dataByte;

adresaCurenta_REG: process (CLK)
    begin
        if Rising_Edge(CLK) then
            if (latchAddr = '1') then
                adresaCurenta <= A_I;
            end if;
        end if;
    end process;

    rwBit <= adresaCurenta(0);

stareSub_CNT: process (CLK)
  begin
    if Rising_Edge(CLK) then
            if (state = idle) then
```

```
                    stareSub <= "00";
                elsif (sclCnt = 0) then
                    stareSub <= stareSub + 1;
                end if;
            end if;
        end process;

SYNC_PROC: process (CLK)
  begin
    if Rising_Edge(CLK) then
      state <= myState;

                sdaR <= iSda;
      rScl <= iScl;
                if (resetIni = '1') then
                    doneOutput <= '0';
                    EROAREOo <= '0';
                    errTypeR <= errType;
                else
                    doneOutput <= iDone;
                    EROAREOo <= iErr;
                    errTypeR <= errType;
                end if;
    end if;
  end process;

OUTPUT_DECODE: process (myState, stareSub, state, errTypeR, dataByte(0),
        sclCnt, bitCount, sdaR, rScl, dataBitOut, lostArb, dSda, adrDataNN)
  begin
```

```
iSda <= sdaR;
iDone <= '0';
iScl <= rScl;
iErr <= '0';
errType <= errTypeR;
latchAddr <= '0';
shiftBit <= '0';
latchData <= '0';

if (state = startSs) then
	case (stareSub) is
		when "00" =>
			iSda <= '1';
		when "01" =>
			iSda <= '1';
			iScl <= '1';
		when "10" =>
			iSda <= '0';
			iScl <= '1';
		when "11" =>
			iSda <= '0';
			iScl <= '0';
		when others =>
	end case;
end if;

if (state = stop or state = stopError) then
	case (stareSub) is
		when "00" =>
```

```
                    iSda <= '0';
                when "01" =>
                    iSda <= '0';
                    iScl <= '1';
                when "10" =>
                    iSda <= '1';
                    iScl <= '1';
                when "11" =>
                    iScl <= '0';
                when others =>
        end case;
end if;

if (state = readss or state = stSAck) then
        case (stareSub) is
                when "00" =>
                    iSda <= '1';
                when "01" =>
                    iScl <= '1';
                when "10" =>
                    iScl <= '1';
                when "11" =>
                    iScl <= '0';
                when others =>
        end case;
end if;

if (state = MackSttt) then
        case (stareSub) is
```

```
                when "00" =>
                        iSda <= '0';
                when "01" =>
                        iScl <= '1';
                when "10" =>
                        iScl <= '1';
                when "11" =>
                        iScl <= '0';
                when others =>
        end case;
end if;

                if (state = writeSs) then
        case (stareSub) is
                when "00" =>
                        iSda <= dataBitOut;
                when "01" =>
                        iScl <= '1';
                when "10" =>
                        iScl <= '1';
                when "11" =>
                        iScl <= '0';
                when others =>
        end case;
end if;


if (state = stMNAckStop or state = MNAckStartSs) then
        case (stareSub) is
```

```
                when "00" =>
                        iSda <= '1';
                when "01" =>
                        iScl <= '1';
                when "10" =>
                        iScl <= '1';
                when "11" =>
                        iScl <= '0';
                when others =>
        end case;
end if;

if (state = stSAck and sclCnt = 0 and stareSub = "01") then
        if (dSda = '1') then
                iErr <= '1';
                iDone <= '1';
                errType <= errNAck;
        elsif (adrDataNN = '0') then
                iDone <= '1';
        end if;
end if;

if (state = readss and stareSub = "01" and sclCnt = 0 and bitCount = 0) then
        iDone <= '1';
end if;

if (state = writeSs and lostArb = '1') then
        errType <= errArb;
        iDone <= '1';
```

```
                iErr <= '1';
            end if;

            if ((state = writeSs and sclCnt = 0 and stareSub = "11") or
                ((state = stSAck or state = readss) and stareSub = "01")) then
                shiftBit <= '1';
            end if;

            if (state = startSs) then
                latchAddr <= '1';
            end if;

            if (state = stSAck and stareSub = "11") then
                latchData <= '1';
            end if;

        end process;

NEXT_STATE_DECODE: process (state, busState, waitSlaveW, lostArb, stbIi, imsg,
SRST, stareSub, bitCount, resetIni, dataByte, A_I, adresaCurenta, rwBit, sclCnt, adrDataNN)
  begin

    myState <= state;

    case (state) is
      when idle =>
        if (stbIi = '1' and busState = busFree and SRST = '0') then
          myState <= startSs;
                                elsif (aSlaveUnblock and timeOutCnt = 0) then
```

```
                                        myState <= stop;
    end if;

when startSs =>
    if (sclCnt = 0) then
                                        if (resetIni = '1') then
                                                myState <= stop;
                                        elsif (stareSub = "11") then
                                                myState <= writeSs;
                                        end if;
                                end if;

                        when writeSs =>
                                if (lostArb = '1') then
                                        myState <= idle;
                                elsif (sclCnt = 0) then
                                        if (resetIni = '1') then
                                                myState <= stop;
                                        elsif (stareSub = "11" and bitCount = 0) then
                                                myState <= stSAck;
                                        end if;
                                end if;

                        when stSAck =>
                                if (sclCnt = 0) then
                                        if (resetIni = '1' or (stareSub = "11" and dataByte(0) = '1')) then
                                                myState <= stop;
                                        elsif (stareSub = "11") then
                                                if (adrDataNN = '1') then
```

```
                                        if (rwBit = '1') then
                                                myState <= readss;
                                        else
                                                myState <= writeSs;
                                        end if;
                                elsif (stbIi = '1') then
                                        if (imsg = '1' or adresaCurenta /= A_I) then
                                                myState <= startSs;
                                        else
                                                if (rwBit = '1') then
                                                        myState <= readss;
                                                else
                                                        myState <= writeSs;
                                                end if;
                                        end if;
                                else
                                        myState <= stop;
                                end if;
                        end if;
                end if;

when stop =>
                if (stareSub = "10" and sclCnt = 0 and lostArb = '0') then
                        myState <= idle;
                end if;


        when readss =>
                if (sclCnt = 0) then
```

```
                if (resetIni = '1') then
                        myState <= stop;
                elsif (stareSub = "11" and bitCount = 7) then
                        if (stbIi = '1') then
                                if (imsg = '1' or adresaCurenta /= A_I) then
                                        myState <= MNAckStartSs;
                                else
                                        myState <= MackSttt;
                                end if;
                        else
                                myState <= stMNAckStop;
                        end if;
                end if;
        end if;

when MackSttt =>
        if (sclCnt = 0) then
                if (resetIni = '1') then
                        myState <= stop;
                elsif (stareSub = "11") then
                        myState <= readss;
                end if;
        end if;

when MNAckStartSs =>
        if (lostArb = '1') then
                myState <= idle;
        elsif (sclCnt = 0) then
                if (resetIni = '1') then
```

```
                                        myState <= stop;
                                    elsif (stareSub = "11") then
                                        myState <= startSs;
                                    end if;
                                end if;

                            when stMNAckStop =>
                                if (lostArb = '1') then
                                    myState <= idle;
                                elsif (sclCnt = 0) then
                                    if (resetIni = '1') then
                                        myState <= stop;
                                    elsif (stareSub = "11") then
                                        myState <= stop;
                                    end if;
                                end if;

                        when others =>
                          myState <= idle;
                    end case;
                end process;

                SDA <= 'Z' when sdaR = '1' else
                      '0';
                SCL <= 'Z' when rSCL = '1' else
                      '0';

            end Behavioral;
```

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

entity displ7seg is
   Port ( Clk  : in  STD_LOGIC;
          Rst  : in  STD_LOGIC;
          Data : in  STD_LOGIC_VECTOR (31 downto 0);
          An   : out STD_LOGIC_VECTOR (7 downto 0);
          Seg  : out STD_LOGIC_VECTOR (7 downto 0));
end displ7seg;

architecture Behavioral of displ7seg is

constant CNT_100HZ : integer := 2**20;
signal number       : integer range 0 to CNT_100HZ - 1 := 0;
signal numberVect       : STD_LOGIC_VECTOR (19 downto 0) := (others => '0');
signal Hex        : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
signal selectareLed      : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');

begin

divclk: process (Clk)
   begin
   if (Clk'event and Clk = '1') then
      if (Rst = '1') then
          number<= 0;
```

```
    elsif (number= CNT_100HZ - 1) then
        number<= 0;
    else
        number<= number+ 1;
    end if;
end if;
end process;

numberVect <= CONV_STD_LOGIC_VECTOR (Number, 20);
selectareLed <= numberVect (19 downto 17);

An <= "11111110" when selectareLed = "000" else
      "11111101" when selectareLed = "001" else
      "11111011" when selectareLed = "010" else
      "11110111" when selectareLed = "011" else
      "11101111" when selectareLed = "100" else
      "11011111" when selectareLed = "101" else
      "10111111" when selectareLed = "110" else
      "01111111" when selectareLed = "111" else
      "11111111";

Hex <= Data (3  downto  0) when selectareLed = "000" else
       Data (7  downto  4) when selectareLed = "001" else
       Data (11 downto  8) when selectareLed = "010" else
       Data (15 downto 12) when selectareLed = "011" else
       Data (19 downto 16) when selectareLed = "100" else
       Data (23 downto 20) when selectareLed = "101" else
       Data (27 downto 24) when selectareLed = "110" else
```

```
        Data (31 downto 28) when selectareLed = "111" else
        X"0";



   Seg <= "11111001" when Hex = "0001" else          -- 1
        "10100100" when Hex = "0010" else          -- 2
        "10110000" when Hex = "0011" else          -- 3
        "10011001" when Hex = "0100" else          -- 4
        "10010010" when Hex = "0101" else          -- 5
        "10000010" when Hex = "0110" else          -- 6
        "11111000" when Hex = "0111" else          -- 7
        "10000000" when Hex = "1000" else          -- 8
        "10010000" when Hex = "1001" else          -- 9
        "10001000" when Hex = "1010" else          -- A
        "10000011" when Hex = "1011" else          -- b
        "11000110" when Hex = "1100" else          -- C
        "10100001" when Hex = "1101" else          -- d
        "10000110" when Hex = "1110" else          -- E
        "10001110" when Hex = "1111" else          -- F
        "11000000";                                -- 0

end Behavioral;



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

```
entity uartRx is
generic (
    clkPerBit : integer := 868
    );
  port (
  clk      : in  std_logic;
    RxSerialInput : in  std_logic;
    RxDvOutput     : out std_logic;
    RxByteOutput   : out std_logic_vector(7 downto 0)
    );
end uartRx;

architecture Behavioral of uartRx is
 type t_SM_Main is (idle, startBitRXs, dataBitsRxS,
                    rxStopB, cleanUp);
  signal smMainR : t_SM_Main := idle;

  signal rxDataR_R : std_logic := '0';
  signal rxDataR   : std_logic := '0';

  signal clkCountR : integer range 0 to clkPerBit-1 := 0;
  signal indexBitR : integer range 0 to 7 := 0;
  signal byteRxRR   : std_logic_vector(7 downto 0) := (others => '0');
  signal rxDvR     : std_logic := '0';

begin
```

```
myProcS : process (clk)
begin
  if rising_edge(clk) then
    rxDataR_R <= RxSerialInput;
    rxDataR   <= rxDataR_R;
  end if;
end process myProcS;



uartRxProcess : process (Clk)
begin
  if rising_edge(Clk) then

    case smMainR is

      when idle =>
        clkCountR <= 0;
        rxDvR     <= '0';
        indexBitR <= 0;

        if rxDataR = '0' then
          smMainR <= startBitRXs;
        else
          smMainR <= idle;
        end if;



      when startBitRXs =>
```

```
 if clkCountR = (clkPerBit-1)/2 then
  if rxDataR = '0' then
   clkCountR <= 0;
   smMainR   <= dataBitsRxS;
  else
   smMainR   <= idle;
  end if;
 else
  clkCountR <= clkCountR + 1;
  smMainR   <= startBitRXs;
 end if;



when dataBitsRxS =>
 if clkCountR < clkPerBit-1 then
  clkCountR <= clkCountR + 1;
  smMainR   <= dataBitsRxS;
 else
  clkCountR          <= 0;
  byteRxRR(indexBitR) <= rxDataR;

  if indexBitR < 7 then
   indexBitR <= indexBitR + 1;
   smMainR   <= dataBitsRxS;
  else
   indexBitR <= 0;
   smMainR   <= rxStopB;
  end if;
 end if;
```

```
        when rxStopB =>
          if clkCountR < clkPerBit-1 then
            clkCountR <= clkCountR + 1;
            smMainR   <= rxStopB;
          else
            rxDvR     <= '1';
            clkCountR <= 0;
            smMainR   <= cleanUp;
          end if;


        when cleanUp =>
          smMainR <= idle;
          rxDvR   <= '0';


        when others =>
          smMainR <= idle;


      end case;
    end if;
  end process uartRxProcess;
  RxByteOutput <= byteRxRR;
  RxDvOutput   <= rxDvR;

end Behavioral;
```

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


entity uartTx is
 generic (
    clkPerBit : integer := 868
    );
  port (
    clk      : in  std_logic;
    txDvInput     : in  std_logic;
    tcByteInput   : in  std_logic_vector(7 downto 0);
    txActivatOut : out std_logic;
    txSerialOut : out std_logic;
    txTerminatOut   : out std_logic
    );
end uartTx;

architecture Behavioral of uartTx is

  signal indexBitR : integer range 0 to 7 := 0;
  type t_SM_Main is (idle, txStartBit, txDataBit,
                     stopBitTx, cleanUp);
  signal smMainR : t_SM_Main := idle;
  signal clkCountR : integer range 0 to clkPerBit-1 := 0;
```

```
  signal txDoneR   : std_logic := '0';
  signal txDataR   : std_logic_vector(7 downto 0) := (others => '0');



begin


uartTxprocess : process (Clk)
  begin
    if rising_edge(Clk) then


      case smMainR is


        when idle =>
          txActivatOut <= '0';
          txSerialOut <= '1';
          txDoneR   <= '0';
          clkCountR <= 0;
          indexBitR <= 0;


          if txDvInput = '1' then
            txDataR <= tcByteInput;
            smMainR <= txStartBit;
          else
            smMainR <= idle;
          end if;




        when txStartBit =>
```

```
 txActivatOut <= '1';
 txSerialOut <= '0';



 if clkCountR < clkPerBit-1 then
  clkCountR <= clkCountR + 1;
  smMainR   <= txStartBit;
 else
  clkCountR <= 0;
  smMainR   <= txDataBit;
 end if;



when txDataBit =>
 txSerialOut <= txDataR(indexBitR);

 if clkCountR < clkPerBit-1 then
  clkCountR <= clkCountR + 1;
  smMainR   <= txDataBit;
 else
  clkCountR <= 0;

  if indexBitR < 7 then
   indexBitR <= indexBitR + 1;
   smMainR   <= txDataBit;
  else
   indexBitR <= 0;
   smMainR   <= stopBitTx;
  end if;
```

```
        end if;



      when stopBitTx =>
        txSerialOut <= '1';

        if clkCountR < clkPerBit-1 then
          clkCountR <= clkCountR + 1;
          smMainR   <= stopBitTx;
        else
          txDoneR   <= '1';
          clkCountR <= 0;
          smMainR   <= cleanUp;
        end if;
      when cleanUp =>
        txActivatOut <= '0';
        txDoneR   <= '1';
        smMainR   <= idle;
      when others =>
        smMainR <= idle;


    end case;
  end if;
end process uartTxprocess;

txTerminatOut <= txDoneR;

end Behavioral;
```

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


entity UCC is
Port
(
signal clk:in std_logic;
signal rst:in std_logic;
signal buttonStart:in std_logic;
signal dateDeIntrare:in std_logic_vector(15 downto 0);
signal activ:in std_logic;
signal done:in std_logic;
signal dateleDeIesire:out std_logic_vector(7 downto 0);
signal start:out std_logic
 );
end UCC;

architecture Behavioral of UCC is

type stari is (inceput,incData,oct1,trans1,oct2,trans2);
signal curentS:stari:=incData;
signal oc2:std_logic_vector(7 downto 0);
signal urmatoareaS:stari:=incData;
signal oc1:std_logic_vector(7 downto 0);
```

```
begin


process(clk,rst)

begin


if rst='1' then

curentS<=inceput;

elsif clk'event and clk='1' then

curentS<=urmatoareaS;

end if;


end process;


process(curentS,activ,done,dateDeIntrare,oc1,oc2)

begin


case curentS is

when inceput=>  if buttonStart='1' then

                urmatoareaS<=incData;

             else

                urmatoareaS<=inceput;

             end if;

when incData=>urmatoareaS<=oct1;

                oc1<=dateDeIntrare(15 downto 8);

                oc2<=dateDeIntrare(7 downto 0);

when oct1=>urmatoareaS<=trans1;

          dateleDeIesire<=oc1;

when trans1=>if activ='0' and done='1' then
```

```
                urmatoareaS<=oct2;
            else
                urmatoareaS<=trans1;
            end if;
when oct2=>urmatoareaS<=trans2;
        dateleDeIesire<=oc2;
when trans2=>if activ='0' and done='1' then
                urmatoareaS<=inceput;
            else
                urmatoareaS<=trans2;
            end if;
end case;

end process;

process(curentS)
begin
case curentS is
when inceput=>start<='0';
when incData=>start<='0';
when trans1=>start<='0';
when trans2=>start<='0';
when oct1=>start<='1';
when oct2=>start<='1';
end case;
end process;
end Behavioral;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;



ENTITY I2C IS
 GENERIC(
   clkInput : INTEGER := 50_000_000;
   clkBus   : INTEGER := 400_000);
 PORT(
   clk       : IN     STD_LOGIC;
   reset     : IN     STD_LOGIC;
   ena       : IN     STD_LOGIC;
   addr      : IN     STD_LOGIC_VECTOR(6 DOWNTO 0);
   rw        : IN     STD_LOGIC;
   dataWrite  : IN     STD_LOGIC_VECTOR(7 DOWNTO 0);
   busy      : OUT    STD_LOGIC;
   dataRead   : OUT    STD_LOGIC_VECTOR(7 DOWNTO 0);
   error : BUFFER STD_LOGIC;
   semnalRead: OUT STD_LOGIC;
   sda       : INOUT  STD_LOGIC;
   scl       : INOUT  STD_LOGIC);
END I2C;

architecture Behavioral of I2C is
 CONSTANT divider  :  INTEGER := (clkInput/clkBus)/4;
  TYPE machine IS(ready, start, command, ack1, wr, rd, ack2, mstr_ack, stop);
```

```
  SIGNAL dataClk      : STD_LOGIC;
  SIGNAL dataClkPrev : STD_LOGIC;
  SIGNAL state        : machine;
  SIGNAL clkScl       : STD_LOGIC;
  SIGNAL enScl       : STD_LOGIC := '0';
  SIGNAL enSclN     : STD_LOGIC;
   SIGNAL intSda       : STD_LOGIC := '1';
  SIGNAL addrRW       : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL RxData       : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL bitCount       : INTEGER RANGE 0 TO 7 := 7;
  SIGNAL TxData       : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL stretch       : STD_LOGIC := '0';
BEGIN


  PROCESS(clk, reset)
    VARIABLE count  :  INTEGER RANGE 0 TO divider*4;
  BEGIN
    IF(reset = '1') THEN
      stretch <= '0';
      count := 0;
    ELSIF(clk'EVENT AND clk = '1') THEN
      dataClkPrev <= dataClk;
      IF(count = divider*4-1) THEN
        count := 0;
      ELSIF(stretch = '0') THEN
        count := count + 1;
      END IF;
      CASE count IS
```

```
      WHEN 0 TO divider-1 =>
        clkScl <= '0';
        dataClk <= '0';
      WHEN divider TO divider*2-1 =>
        clkScl <= '0';
        dataClk <= '1';
      WHEN divider*2 TO divider*3-1 =>
        clkScl <= '1';
        IF(scl = '0') THEN
          stretch <= '1';
        ELSE
          stretch <= '0';
        END IF;
        dataClk <= '1';
      WHEN OTHERS =>
        clkScl <= '1';
        dataClk <= '0';
    END CASE;
  END IF;
END PROCESS;

PROCESS(clk)
begin
if clk'event and clk='1' then
      if state=mstr_ack then
         semnalRead<='1';
      else
         semnalRead<='0';
      end if;
```

```
end if;


end process;


PROCESS(clk, reset)
BEGIN
  IF(reset = '1') THEN
    state <= ready;
    enScl <= '0';
    intSda <= '1';
    busy <= '1';
    error <= '0';
    dataRead <= "00000000";
      bitCount <= 7;
  ELSIF(clk'EVENT AND clk = '1') THEN
    IF(dataClk = '1' AND dataClkPrev = '0') THEN
      CASE state IS
        WHEN ready =>
          IF(ena = '1') THEN
            busy <= '1';
            TxData <= dataWrite;
            addrRW <= addr & rw;
            state <= start;
          ELSE
            busy <= '0';
            state <= ready;
          END IF;
        WHEN start =>
          busy <= '1';
```

```
  intSda <= addrRW(bitCount);
  state <= command;
 WHEN command =>
  IF(bitCount = 0) THEN
    bitCount <= 7;
    intSda <= '1';
    state <= ack1;
  ELSE
    bitCount <= bitCount - 1;
    intSda <= addrRW(bitCount-1);
    state <= command;
  END IF;
 WHEN ack1 =>
  IF(addrRW(0) = '0') THEN
    intSda <= TxData(bitCount);
    state <= wr;
  ELSE
    intSda <= '1';
    state <= rd;
  END IF;
 WHEN wr =>
  busy <= '1';
  IF(bitCount = 0) THEN
    bitCount <= 7;
    intSda <= '1';
    state <= ack2;
  ELSE
    bitCount <= bitCount - 1;
    intSda <= TxData(bitCount-1);
```

```
    state <= wr;
   END IF;
  WHEN rd =>
   busy <= '1';
   IF(bitCount = 0) THEN
    IF(ena = '1' AND addrRW = addr & rw) THEN
     intSda <= '0';
    ELSE
     intSda <= '1';
    END IF;
    dataRead <= RxData;
    bitCount <= 7;
    state <= mstr_ack;
   ELSE
    bitCount <= bitCount - 1;
    state <= rd;
   END IF;
  WHEN ack2 =>
   IF(ena = '1') THEN
    addrRW <= addr & rw;
    busy <= '0';
    TxData <= dataWrite;
    IF(addrRW = addr & rw) THEN
     intSda <= dataWrite(bitCount);
     state <= wr;
    ELSE
     state <= start;
    END IF;
   ELSE
```

```
        state <= stop;
      END IF;
    WHEN mstr_ack =>
      IF(ena = '1') THEN
        busy <= '0';
        addrRW <= addr & rw;
        TxData <= dataWrite;
        IF(addrRW = addr & rw) THEN
          intSda <= '1';
          state <= rd;
        ELSE
          state <= start;
        END IF;
      ELSE
        state <= stop;
      END IF;
    WHEN stop =>
      busy <= '0';
      state <= ready;
  END CASE;
ELSIF(dataClk = '0' AND dataClkPrev = '1') THEN
  CASE state IS
    WHEN start =>
      IF(enScl = '0') THEN
        error <= '0';
        enScl <= '1';

      END IF;
    WHEN ack1 =>
```

```
          IF(sda /= '0' OR error = '1') THEN
            error <= '1';
          END IF;
        WHEN rd =>
          RxData(bitCount) <= sda;
        WHEN ack2 =>
          IF(sda /= '0' OR error = '1') THEN
            error <= '1';
          END IF;
        WHEN stop =>
          enScl <= '0';
        WHEN OTHERS =>
          NULL;
      END CASE;
    END IF;
  END IF;
END PROCESS;

WITH state SELECT
  enSclN <= dataClkPrev WHEN start,
            NOT dataClkPrev WHEN stop,
            intSda WHEN OTHERS;

sda <= '0' WHEN enSclN = '0' ELSE 'Z';
scl <= '0' WHEN (enScl = '1' AND clkScl = '0') ELSE 'Z';


end Behavioral;
```