***Măsurarea si afișarea umidității cu senzorul de umiditate Pmod HYGRO***

Student: Balint Cătălin-Vasile

Grupa: 30236/1

Indrumător de proiect: Lișman Dragoș

Data: 08.01.2024

***Cuprins***

[***1.*** ***Rezumat*** 3](#_Toc60610308)

[***2.*** ***Introducere*** 4](#_Toc60610309)

[***3.*** ***Fundamentare Teoretica*** 6](#_Toc60610310)

[***3.1.*** ***Limbajul de descriere hardware VHDL*** 6](#_Toc60610311)

[***3.2.*** ***Xilinx Vivado*** 7](#_Toc60610312)

[***3.3.*** ***Placa de dezvolgtare Nexys 4 DDR*** 8](#_Toc60610313)

[***3.4.*** ***Senzorul de umiditate*** 9](#_Toc60610314)

[***3.5.*** ***Interfața I2C*** 11](#_Toc60610315)

[***3.6.*** ***Solutia propusa*** 12](#_Toc60610316)

[***4.*** ***Proiectare si implementare*** 13](#_Toc60610317)

[***4.1.*** ***Arhitectura generala*** 13](#_Toc60610318)

[***4.2.*** ***Afisorul 7 segmente*** 14](#_Toc60610320)

[***4.3.*** ***Interfata I2C*** 15](#_Toc60610321)

[***4.3.1.*** ***Dispozitivul Master*** 18](#_Toc60610322)

[***4.3.2.*** ***Controller*** 19](#_Toc60610323)

[***4.4.*** ***Modulul principal*** 20](#_Toc60610324)

[***4.5.*** ***Binar în decimal*** 21](#_Toc60610324)

[***5.*** ***Rezultate Experimentale*** 22](#_Toc60610325)

[***6.*** ***Concluzii*** 23](#_Toc60610326)

[***Bibliografie*** 24](#_Toc60610327)

# ***Rezumat***

În cadrul proiectului realizat în mediu Vivado 2017.4, am abordat cu succes proiectarea și implementarea unui sistem destinat măsurării și afișării umidității relative, utilizând senzorul de umiditate Pmod HYGRO pe placa Nexys 4 DDR. Scopul principal al acestei inițiative a constat în utilizarea limbajului de programare VHDL și implementarea standardului I2C pentru a facilita o comunicare eficientă între placa Nexys 4 DDR și senzorul de umiditate. În cadrul procesului de dezvoltare, am integrat tehnici pentru manipularea datelor provenite de la senzor, asigurând astfel o prelucrare precisă și fiabilă a informațiilor de umiditate. Rezultatele obținute au fost sintetizate și afișate pe interfața plăcii Digilent Nexys 4 DDR, evidențiind cu succes funcționalitatea sistemului dezvoltat. Prin această abordare, am reușit să obținem o soluție robustă și eficientă pentru măsurarea și monitorizarea umidității relative, cu aplicabilitate practică. În concluzie, proiectul a reprezentat o integrare coezivă a componentelor hardware și software, furnizând o soluție comprehensivă pentru măsurarea umidității într-un mediu specific utilizând senzorul Pmod HYGRO.

# ***Introducere***

Proiectul prezent abordează dezvoltarea și implementarea unui sistem dedicat măsurării și afișării umidității relative, utilizând senzorul de umiditate Pmod HYGRO pe placa Nexys 4 DDR. În cadrul acestui proiect, am explorat potențialul limbajului de programare VHDL și am implementat standardul I2C pentru a asigura o comunicare eficientă între placa Nexys 4 DDR și senzorul de umiditate. Scopul principal al proiectului constă în furnizarea unei soluții robuste și eficiente pentru măsurarea și monitorizarea umidității relative, având o aplicabilitate practică.

În contextul evoluției tehnologice actuale, monitorizarea și măsurarea parametrilor de mediu, cum ar fi umiditatea, devin din ce în ce mai importante în domenii variate precum agricultura, industria alimentară și medicina. Avansul tehnologic oferă soluții inovatoare pentru dezvoltarea dispozitivelor de senzorizare și control, cu impact semnificativ asupra eficienței și calității proceselor din diferite domenii.

Domeniul de studiu al măsurătorii și monitorizării umidității relative, în cadrul proiectului nostru, se intersectează direct cu materia "Structura Sistemelor de Calcul". Această conexiune poate fi explicată prin implicarea directă a unui sistem încorporat (sistemul dezvoltat) în procesul de colectare, procesare și afișare a datelor de mediu, în acest caz, umiditatea.

În cadrul proiectului, senzorul de umiditate Pmod HYGRO furnizează datele de mediu necesare pentru măsurarea umidității relative. Aceste date sunt apoi procesate de către sistemul implementat în limbajul VHDL, în conformitate cu standardul I2C. Acest proces de procesare a datelor reprezintă un aspect esențial al structurii sistemelor de calcul.

Implementarea standardului I2C facilitează comunicarea eficientă între placa Nexys 4 DDR și senzorul de umiditate. Interacțiunea între diferitele componente ale sistemului, cum ar fi senzorul, controlerul I2C și placa de dezvoltare, evidențiază aspectele de interfațare și comunicare specifice structurii sistemelor de calcul.

Structura sistemelor de calcul implică definirea și controlul fluxului de date într-un sistem integrat. În proiectul nostru, controlul este realizat prin intermediul mașinii de stări finite (FSM) implementată în VHDL, care gestionează stările sistemului în funcție de evenimente precum comenzi de citire/scriere, confirmări și timpi de așteptare.

Problema de rezolvat constă în dezvoltarea unui sistem eficient care să permită măsurarea și afișarea umidității relative. Obiectivele principale ale proiectului includ implementarea standardului I2C pentru comunicare, integrarea senzorului Pmod HYGRO și furnizarea unei interfețe vizuale pentru monitorizarea datelor de umiditate.

Soluția propusă constă în utilizarea limbajului de programare VHDL și implementarea unui sistem de control I2C pentru a realiza comunicarea între placa Nexys 4 DDR și senzorul de umiditate. Acest aspect oferă o abordare robustă și flexibilă pentru măsurarea și afișarea precisă a umidității relative.

Proiectul evidențiază integrarea coezivă a componentelor hardware și software. Prin limbajul de programare VHDL, am realizat o implementare eficientă a funcționalităților sistemului, subliniind importanța structurii bine definite a sistemelor de calcul.

Raportul cuprinde detaliile procesului de dezvoltare, implementare și rezultatele obținute. Secțiunile ulterioare vor explora în detaliu fundamentele teoretice, proiectarea sistemului, implementarea tehnică, rezultatele experimentelor și concluziile finale ale proiectului.

Prin această inițiativă, ne propunem să contribuim la evoluția tehnologiei de măsurare a umidității relative și să oferim o soluție practică și eficientă pentru monitorizarea parametrilor de mediu într-un context specific.

# ***Fundamentare Teoretica***

Proiectarea sistemului de măsurare și afișare a umidității cu senzorul Pmod HYGRO pe placa Nexys 4 DDR se sprijină pe un set robust de fundamente teoretice. Utilizarea senzorului de umiditate Pmod HYGRO constituie o alegere esențială, având în vedere capacitatea sa de a oferi măsurători precise ale umidității relative a mediului înconjurător. Integrarea standardului I2C în proiect reprezintă un aspect crucial, facilitând comunicarea eficientă între placa Nexys 4 DDR și senzorul Pmod HYGRO. Protocolul I2C (Inter-Integrated Circuit) asigură o interfață de comunicație fiabilă și flexibilă.

Implementarea acestei soluții în limbajul VHDL adaugă un nivel de abstractizare și structurare, permitând definirea comportamentului hardware într-un mod coerent și sintetic. Implementarea proiectului pe placa Digilent Nexys 4 DDR oferă mediul de testare și afișare a rezultatelor, constituind un cadru fizic pentru validarea funcționalității sistemului. Împreună, aceste fundamente teoretice formează baza proiectului, asigurând un cadru solid pentru dezvoltarea și implementarea cu succes a sistemului de măsurare a umidității.

## ***Limbajul de descriere hardware VHDL***

VHDL (VHSIC Hardware Description Language) reprezintă un limbaj de descriere hardware, o formă de programare de nivel înalt, utilizată în special pentru modelarea circuitelor electronice digitale. Acest limbaj este un standard IEEE și se remarcă prin capacitatea sa de a descrie și modela diverse aspecte ale circuitelor digitale, inclusiv comportamentul, structura și aspecte mixte ale acestora.

Fiecare unitate de proiect este definită în fișiere sursă cu extensia \*.vhd, unde pot exista mai multe unități de proiect în același fișier, însă se recomandă o structură care să conțină o pereche entitate-arhitectură într-un fișier.

VHDL utilizează o structură de date orientată spre conceptul de bază de date, incluzând noțiuni precum directoare și biblioteci. Bibliotecile conțin rezultatele compilării fișierelor sursă și sunt utilizate de simulatorul VHDL pentru simularea entităților sau configurațiilor definite.

## ***Xilinx Vivado***

Xilinx ISE (Integrated Synthesis Environment) este o unealtă software dezvoltată de Xilinx pentru sinteza și analiza proiectelor de limbaj de descriere hardware (HDL), cu accent principal pe dezvoltarea firmware-ului încorporat pentru familia de circuite integrate FPGA și CPLD de la Xilinx. Acest software a fost întrerupt și a fost înlocuit de Xilinx Vivado.

Vivado Design Suite este o suită software pentru sinteza și analiza proiectelor de limbaj de descriere hardware (HDL), înlocuind Xilinx ISE și aducând funcționalități suplimentare pentru dezvoltarea de sisteme pe un cip și sinteză la nivel înalt. Vivado reprezintă o rescriere completă și o reconsiderare a întregului flux de proiectare (în comparație cu ISE).

A yellow and black logo

Description automatically generated A red and orange logo

Description automatically generated

*Figura 1 :LOGO Vivado Figura 2:LOGO Xilinx*

## ***Placa de dezvoltare Nexys 4 DDR***

Nexys4-DDR de la Digilent este o înlocuire directă pentru plăcile noastre Nexys bazate pe RAM celular, datorită încheierii ciclului de viață al acestora. Este o platformă de dezvoltare a circuitelor digitale gata de utilizare, proiectată pentru a aduce aplicații industriale suplimentare în mediul academic. Cu FPGA-ul său mare și de mare capacitate și o serie de porturi USB, Ethernet și altele, Nexys4-DDR poate găzdui proiecte începând de la circuite combinaționale introductive până la procesoare înglobate puternice.

**Caracteristici ale plăcii**

* 16 comutatoare de utilizator
* Pod USB-UART
* 16 LED-uri de utilizator
* Senzor de temperatură
* Flash serial
* Port USB-JTAG Digilent pentru programare și comunicare FPGA
* Două afișaje cu 7 segmente cu 4 cifre
* Conector card Micro SD
* Patru porturi Pmod
* Gazdă USB HID pentru mouse-uri, tastaturi și stick-uri de memorie

A close-up of a circuit board

Description automatically generatedA green electronic board with a black cord

Description automatically generated

*FIGURA 3&4,Placa de dezvoltate NEXYS4 DRR*

## ***Senzorul de umiditate***

Umiditatea se referă la cantitatea de vapori de apă din aer sau într-un alt gaz. Aceasta poate fi exprimată sub formă absolută sau relativă.Umiditatea relativa este raportul dintre cantitatea reală de vapori de apă prezenți în aer și cantitatea maximă pe care aerul o poate conține la o anumită temperatură, fără a se produce condensarea

Pmod HYGRO este proiectat pentru a raporta digital umiditatea relativă și temperatura ambiantă la cerere de către placa gazdă. Până la 14 biți de rezoluție pot fi colectați pentru fiecare senzor, permițând timpuri mai lungi de conversie. Un element de încălzire rezistiv poate fi activat pentru a îndepărta condensul care se acumulează pe senzor după expunerea la condiții de umiditate ridicată.

Registrul temperaturii (adresa 0x00) și registrul umidității (adresa 0x01) sunt ambele registre de citire cu 16 biți, având cele 14 biți MSB corespondenți rezoluției maxime de 14 biți pentru ambii senzori. Cei doi LSB-uri (biții D1 și D0) sunt întotdeauna 0 pentru ambele registre. Pe măsură ce rezoluția măsurătorii crește, timpul corespunzător de conversie pentru fiecare senzor crește, de asemenea.

Pentru a efectua o măsurare, utilizatorii trebuie să configureze (sau să accepte) setările de măsurare în Registrul de Configurare și apoi să declanșeze măsurarea sau măsurările, trimitând o tranzacție de scriere I²C împreună cu pointerul de adresă setat la registrul corespunzător. După așteptarea timpilor de conversie corespunzători, utilizatorii pot efectua o tranzacție de citire, beneficiind de pointerul cu auto-incrementare pentru citirea atât a registrului de temperatură, cât și a celui de umiditate. După ce datele au fost citite, utilizatorii trebuie să aștepte cel puțin o secundă între tranzacții pentru a evita încălzirea internă a senzorului și distorsionarea nivelurilor măsurate de temperatură și umiditate.



*Figura 5: formula de conversie pentru “readability of” datelor*

A close-up of a blue circuit board

Description automatically generated

*Figura 6 : Senzorul Pmod HYGRO*

A diagram of a computer

Description automatically generated

*Figura 7 : schematica senzorului Pmod HYGRO*

## ***Interfața I2C***

Protocolul I2C (sau IIC - Inter-Integrated Circuit) este un protocol de comunicație serială sincron, multi-master - multi-slave, dezvoltat de către Phillips în anul 1982. O magistrală I2C este formată din următoarele semnale:

**SDA - linia de date**

**SCL - semnalul de clock**

Semnalul de ceas este generat de către master iar linia de date este controlată atât de master cât și de către slave. La un moment dat un singur dispozitiv de pe magistrală poate controla linia de date. Din acestă cauză protocolul I2C este half-duplex.

Pmod HYGRO comunică cu placa gazdă prin protocolul I²C. Prin trimiterea inițială a adresei I²C de 7 biți 1000000 (0x40), urmată de un bit de citire/scriere (nivel logic ridicat/scăzut, respectiv), apoi de adresa registrului de interes la o frecvență maximă a ceasului de 400 kHz, utilizatorii pot configura și citi de la Pmod HYGRO. Conectorul J2 de pe Pmod HYGRO transmite toate semnalele prezente pe conectorul principal J1 pentru a permite interconectarea în cască a mai multor module compatibile I²C.

A diagram of a sla

Description automatically generated

*Figura 8: Protocol I2C Master-Slave*

A diagram of a sensor

Description automatically generated

*Figura 9 :Comunicare cu senzorul(Sensor #5)*

## ***Soluția propusă***

Modul de functionare al poriectului consta in masaurarea umiditatii folosind senzorul de umiditate Pmod HYGRO si afisarea acestuia pe afisorul 7 segmenete al placii.

Initial la incarcarea proiectului pe placa de dezvoltare Nexys 4 DDR, pe display va aparea valoarea 0, dupa care la apasarea unui buton se va face reset la comunicarea I2C dintre placa si senzorul integrat de umiditate, dupa care pe afisor va aparea valoarea masurata de catre senzor.

# ***Proiectare și implementare***

## ***Arhitectura generală***

A diagram of a diagram

Description automatically generated

*Figura 10: Diagrama de stare a aplicatiei*

A diagram of a computer

Description automatically generated

*Figura 11 : RLT Schematic*

## ***Afisorul 7 segmente***

A diagram of a computer

Description automatically generated

*Figura 12 : Schema de functionare a afisorului cu 7 segmente*

Acest modul are rolul de a afisa informatia primita de la senzorul de umiditate pentru a instiinta utilizatorul care este valoarea temperaturii.

Deși Nexys4 DDR dispune de 8 anozi , afisarea umidității măsurate va fi pe doar pe 4.

## ***Interfata I2C***

Acest modul are rolul de a permite comunicarea intre placa de dezoltare Nexys 4 DDR si senzorul integrat de umiditate Pmod HYGRO

Comunicarea I2C este realizata intre un dispozitiv master si unul sau mai multe dispositive de tip slave.In acest caz particular senzorul de umiditate va juca rolul slave-ului , iar placa Nexys 4 DDR va fi master-ul.

Această mașină de stări finite (FSM) este proiectată pentru a controla un proces în funcție de semnalul de ceas (clk) și alte semnale de intrare, cum ar fi resetarea (rst), activarea (en), activarea negată (en\_n), citire/scriere (rw), și informațiile despre adresă (address și old\_address). FSM-ul are mai multe stari și comportamentul său este determinat de starea curentă (st).

În starea inițială, ready, FSM-ul trece la starea start atunci când semnalul en este activat. În starea start, sunt inițializate unele variabile, iar FSM-ul trece la starea command atunci când semnalul en\_n este activat. În starea command, un contor (count) este decrementat și FSM-ul trece la starea slave\_ack când contorul ajunge la 1. Starea slave\_ack resetează contorul și decide dacă să treacă la starea write sau read în funcție de semnalul de citire/scriere (rw). Restul tranzițiilor implică manipularea contorului și decizii bazate pe semnalele de intrare pentru a ghida FSM-ul prin diferitele stări. Starea stop permite revenirea la starea ready atunci când semnalul en\_n este activat sau să rămână în starea stop și să seteze un indicator (stop\_clk) atunci când semnalul en\_p este activat. Este important să rețineți că comportamentul real și scopul acestei FSM depind de aplicația specifică și de codul înconjurător care utilizează ieșirile aceste

A diagram of a computer

Description automatically generated

*Figura 13: FSM pentru I2C*

A diagram of a process flow

Description automatically generated

*Figura 14 : FSM pentru I2C\_MODULE(Controller-ul I2C)*

* + 1. ***Dispozitivul Master***

Acest modul I2\_C Master este un automat cu stari finite(FSM) care gestioneaza fluxul si transmiterea de date intre placa de dezvoltare Nexys 4 DDR si senzorul de umiditate Pmod HYGRO

A diagram of a square with arrows pointing to the center

Description automatically generated

*Figura 15: Schema bloc a dispozitivului Master*

**Semnale:**

* clk : semnalul de ceas
* en : semnalul de enable
* rst : semnalul de restart
* rw: semnalul de scriere ( rw=’0’)/ citire din registru ( rw=’1’)
* address : adresa registrului pentru sciere / citire
* SDA : Serial Data Line (semnal bidirectional)
* SCL : Serial Clock Line (semnal bidirectional)
* Term : semnaleaza terminarea actiunii
* data\_out : datele de iesire
* data\_in : datele de intrare

### ***Controller***

Modulul I2C\_module este un automat cu stari finite(FSM) care reprezinta modulul de control al interfetei si protocolului de comunicare IIC.Se evidentiaza legatura dintre dispozitivul Master si dispozitivul Slave.Master-ul este mapat in interiorul acestui modul comunicand direct cu Slave-ul ,adica senzorul de umiditate Pmod HYGRO.

A diagram of a computer

Description automatically generated

*Figura 16: Schema bloc a Controller-ului interfetei I2C*

**Semnale:**

* clk : semnalul de ceas
* rst : semnalul de resetare al transmiterii
* SDA : Serial Data Line
* SCL : Serial Clock Line
* humidity : valorea obtinuta in urma comunicarii intre cele doua dispositive

## ***Modulul principal Modulul principal***

In acest modul se realizeaza maparile , si “legarea” propriu-zisa a componentelor proiectelui . Se evidentiaza transmiterea datelor primate de la senzorul Pmod HYGRO prin intermediul protocolului de comunicare IIC catre afisorul cu 7 segmente al placii de dezvoltare Nexys4 DDR

A black and white rectangular object with a black text

Description automatically generated

*Figura 17: Schema bloc a modulului principal*

**Semnale:**

* clk : semnalul de ceas
* rst : butonul care reseteaza procesul de comunicare I2C
* sda: Serial Data Line
* scl : Serial Clock Line
* seg : catodul afisorul cu 7 segmnete
* an: anodul afisorului cu 7 segmente
* dp: al-8-lea segment al afisorului cu 7 segmente

## ***Binar în decimal***

Modulul BinaryToBCD este conceput pentru a transforma un număr binar de 8 biți într-un format BCD (Binary-Coded Decimal) de 12 biți. Această conversie este utilă atunci când vrei să afișezi numere pe un afișaj cu 7 segmente. Formatul BCD descompune fiecare cifră a unui număr în patru biți, facilitând astfel afișarea pe afișaje cu 7 segmente, care sunt configurate pentru a afișa cifre BCD.

În mod specific, modulul efectuează adăugarea de 3 la fiecare grup de 4 biți ai numărului binar, acționând în mod corespunzător pentru a gestiona depășirea cifrei 9. Rezultatul final este un număr BCD de 12 biți care poate fi folosit pentru afișare.

# ***Rezultate Experimentale***

Proiectul a fost test pe placa de dezvoltare Nexys 4 DDR utilizand un sensor nou de masurare a umidității Pmod HYGRO in cadrul orelor de laborator . Dupa configurarea constrangerilor , si rezolvarea unor erori in generarea bitstream-ului , 4 din cele 8 anozi au reprezentat valoarea aproximativa de 59 .Aceasta valoarea a umiditatii relative masurata de sensor si comunicata afisoarelor de 7 segmente ale placutei Nexys 4 DDR a fluctuat natural intre 58-61%RH(RH = relative humidity) în încăperea unde se desfasoare laboratorul de SSC(Structura Sistemelor de Calcul).

Pentru a verifica capibilitatea sensorului si eficienta codului ,am introdus senzorul intr-o sticla de apa ( cu deschidere mare ) altfel crescandu-i valoarea pana la valoarea aproximativa de 70%.Un alt experiment utilizat , este suflarea direct pe sensor care a crescut umiditatea relativa la 62-65%.

# ***Concluzii***

Concluziile obținute din acest proiect sunt esențiale pentru înțelegerea rezultatelor și a impactului în domeniul sistemelor de calcul și măsurătorilor de mediu. Problema principală abordată în proiect a fost dezvoltarea unui sistem de măsurare și afișare a umidității relative folosind senzorul Pmod HYGRO și placa Nexys 4 DDR, utilizând limbajul VHDL și mediul Vivado. Obiectivele proiectului au fost îndeplinite prin implementarea standardului I2C pentru comunicarea eficientă cu senzorul și utilizarea unui model VHDL bine structurat.

Contribuțiile noastre originale includ proiectarea detaliată a sistemului, integrarea corectă a senzorului Pmod HYGRO și a plăcii Nexys 4 DDR, precum și implementarea eficientă a standardului I2C în limbajul VHDL. Acest proiect a evidențiat avantajele unei abordări coezive între hardware și software în proiectarea sistemelor încorporate.

Avantajele proiectului includ precizia măsurătorilor de umiditate, eficiența comunicării I2C, și o implementare robustă și fiabilă în limbajul VHDL. Dezavantajele pot include posibilele limitări ale hardware-ului disponibil și necesitatea unor ajustări fine pentru diverse medii de utilizare.

Sugestii pentru aplicații ulterioare ale acestui proiect pot include extinderea sistemului pentru măsurători multiple sau integrarea cu alte dispozitive. Dezvoltările viitoare ar putea viza optimizarea consumului de energie și extinderea funcționalităților sistemului.

Prin acest proiect, am dobândit o înțelegere mai profundă a structurii sistemelor de calcul, limbajului VHDL și mediului Vivado. Experiența acumulată reprezintă o bază solidă pentru proiecte viitoare și ne-a oferit oportunitatea de a aplica cunoștințele teoretice într-un context practic.

# ***Bibliografie***

1. Xilinx Vivado. URL : <https://en.wikipedia.org/wiki/Xilinx_Vivado>
2. VHDL. URL : <https://en.wikipedia.org/wiki/VHDL>
3. Nexys 4 DDR FPGA Board Reference Manual. URL : <http://users.utcluj.ro/~baruch/resources/Digilent/nexys4ddr_rm.pdf>
4. Interfata I2C Specificatii URL : <https://www.mouser.de/datasheet/2/1026/i2c_master-1660139.pdf>
5. Digital Humidity Sensor : [HDC1080 Low Power, High Accuracy Digital Humidity Sensor with Temperature Sensor datasheet (Rev. A) (ti.com)](https://www.ti.com/lit/ds/symlink/hdc1080.pdf)
6. Pmod HYGRO : [pmod-hygro-rm.pdf (tme.eu)](https://www.tme.eu/Document/effcce74709a8bc1f59ac1a5639b3178/pmod-hygro-rm.pdf)
7. Temperature sensor I2C : [I2C on FPGA Temperature Sensor Nexys A7 or Basys 3 w/ Pmod TMP2 Verilog - YouTube](https://www.youtube.com/watch?v=4W9MQa3jBm8&t=6s)
8. I2C MASTER : [I2C Master (VHDL) - Semiconductor / Logic - Electronic Component and Engineering Solution Forum - TechForum │ Digi-Key (digikey.com)](https://forum.digikey.com/t/i2c-master-vhdl/12797)

# ***Anexa***

## ***Modulul principal***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

Library UNISIM;

use UNISIM.vcomponents.all;

entity Principal is

Port ( Clk : in STD\_LOGIC;

Rst : in STD\_LOGIC;

Sda : inout STD\_LOGIC;

Scl : inout STD\_LOGIC;

dp : out STD\_LOGIC;

an : out STD\_LOGIC\_VECTOR (3 downto 0);

cat : out STD\_LOGIC\_VECTOR (6 downto 0));

end Principal;

architecture Behavioral of Principal is

signal humidity : STD\_LOGIC\_VECTOR(15 downto 0);

signal bin : STD\_LOGIC\_VECTOR(15 downto 0);

signal dataShow : STD\_LOGIC\_VECTOR(15 downto 0);

signal binfrac : STD\_LOGIC\_VECTOR(15 downto 0);

signal bcd : STD\_LOGIC\_VECTOR(23 downto 0);

begin

binfrac <= std\_logic\_vector(unsigned(bin(7 downto 0)) \* 100);

DUT0: entity WORK.I2C\_module port map (

Clk => Clk,

rst => rst,

scl => scl,

humidity => humidity,

sda => sda);

DUT1: entity WORK.displ7seg port map (

Clk => Clk,

rst => rst,

data => dataShow,

Dp => Dp,

an => an,

seg => cat);

DUT2: entity WORK.Convert port map (

Clk => Clk,

EnClk => '1',

SensorVal => humidity,

Humidity => bin);

DUT3: entity WORK.BinaryToBcd port map (

bin => bin(15 downto 8),

bcd => bcd(23 downto 12));

DUT4: entity WORK.BinaryToBcd port map (

bin => binfrac(15 downto 8),

bcd => bcd(11 downto 0));

dataShow <= bcd(19 downto 12) & bcd(7 downto 0);

end Behavioral

## ***BinarytoBCD***

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

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.NUMERIC\_STD.ALL;

entity BinaryToBCD is

Port ( bin : in STD\_LOGIC\_VECTOR (7 downto 0);

bcd : out STD\_LOGIC\_VECTOR (11 downto 0));

end BinaryToBCD;

architecture Behavioral of BinaryToBCD is

procedure add3 (signal bin: in std\_logic\_vector (3 downto 0);

signal bcd: out std\_logic\_vector (3 downto 0)) is

variable is\_gt\_4: std\_logic;

begin

is\_gt\_4 := bin(3) or (bin(2) and (bin(1) or bin(0)));

if is\_gt\_4 = '1' then

bcd <= std\_logic\_vector(unsigned(bin) + "0011");

else

bcd <= bin;

end if;

end procedure;

signal U0bin,U1bin,U2bin,U3bin,U4bin,U5bin,U6bin : std\_logic\_vector (3 downto 0);

signal U0bcd,U1bcd,U2bcd,U3bcd,U4bcd,U5bcd,U6bcd : std\_logic\_vector (3 downto 0);

begin

U0bin <= '0' & bin (7 downto 5);

U1bin <= U0bcd(2 downto 0) & bin(4);

U2bin <= U1bcd(2 downto 0) & bin(3);

U3bin <= U2bcd(2 downto 0) & bin(2);

U4bin <= U3bcd(2 downto 0) & bin(1);

U5bin <= '0' & U0bcd(3) & U1bcd(3) & U2bcd(3);

U6bin <= U5bcd(2 downto 0) & U3bcd(3);

U0: add3(U0bin,U0bcd);

U1: add3(U1bin,U1bcd);

U2: add3(U2bin,U2bcd);

U3: add3(U3bin,U3bcd);

U4: add3(U4bin,U4bcd);

U5: add3(U5bin,U5bcd);

U6: add3(U6bin,U6bcd);

OUTP : bcd <= '0' & '0' & U5bcd(3) & U6bcd & U4bcd & bin(0);

end Behavioral;

## ***Afisor 7 Segmente***

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 (15 downto 0);

An : out STD\_LOGIC\_VECTOR (3 downto 0);

Dp : out STD\_LOGIC;

Seg : out STD\_LOGIC\_VECTOR (6 downto 0));

end displ7seg;

architecture Behavioral of displ7seg is

constant CNT\_100HZ : integer := 2\*\*20;

signal Num : integer range 0 to CNT\_100HZ - 1 := 0;

signal NumV : STD\_LOGIC\_VECTOR (19 downto 0) := (others => '0');

signal LedSel : STD\_LOGIC\_VECTOR (1 downto 0) := (others => '0');

signal Hex : STD\_LOGIC\_VECTOR (3 downto 0) := (others => '0');

begin

divclk: process (Clk)

begin

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

if (Rst = '1') then

Num <= 0;

elsif (Num = CNT\_100HZ - 1) then

Num <= 0;

else

Num <= Num + 1;

end if;

end if;

end process;

NumV <= CONV\_STD\_LOGIC\_VECTOR (Num, 20);

LedSel <= NumV (19 downto 18);

An <= "1110" when LedSel = "00" else

"1101" when LedSel = "01" else

"1011" when LedSel = "10" else

"0111" when LedSel = "11" else

"1111";

Hex <= Data (3 downto 0) when LedSel = "00" else

Data (7 downto 4) when LedSel = "01" else

Data (11 downto 8) when LedSel = "10" else

Data (15 downto 12) when LedSel = "11" else

X"0";

Dp <= '0' when LedSel = "10" else '1';

Seg <= "1111001" when Hex = "0001" else -- 1

"0100100" when Hex = "0010" else -- 2

"0110000" when Hex = "0011" else -- 3

"0011001" when Hex = "0100" else -- 4

"0010010" when Hex = "0101" else -- 5

"0000010" when Hex = "0110" else -- 6

"1111000" when Hex = "0111" else -- 7

"0000000" when Hex = "1000" else -- 8

"0010000" when Hex = "1001" else -- 9

"0001000" when Hex = "1010" else -- A

"0000011" when Hex = "1011" else -- b

"1000110" when Hex = "1100" else -- C

"0100001" when Hex = "1101" else -- d

"0000110" when Hex = "1110" else -- E

"0001110" when Hex = "1111" else -- F

"1000000"; -- 0

end Behavioral;

## ***Master\_I2C***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

entity I2C\_master is

Generic(clk\_freq : INTEGER := 100\_000\_000;

i2c\_freq : INTEGER := 100\_000);

Port ( clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

rw : in STD\_LOGIC;

en : in STD\_LOGIC;

address : in STD\_LOGIC\_VECTOR(6 downto 0);

data\_in : in STD\_LOGIC\_VECTOR (7 downto 0);

data\_out : out STD\_LOGIC\_VECTOR (7 downto 0);

scl : inout STD\_LOGIC;

term : out STD\_LOGIC;

sda : inout STD\_LOGIC);

end I2C\_master;

architecture Behavioral of I2C\_master is

signal old\_address : STD\_LOGIC\_VECTOR(6 downto 0);

signal clk\_i2c : STD\_LOGIC := '1';

signal sda\_inter : STD\_LOGIC := '1';

signal scl\_inter : STD\_LOGIC := '1';

type TIP\_STARE is (ready, start, command, slave\_ack, write, read, written, red, stop);

signal st : TIP\_STARE := ready;

signal count : INTEGER := 0;

signal en\_p : STD\_LOGIC := '0';

signal en\_n : STD\_LOGIC := '0';

signal second\_quarter : STD\_LOGIC := '0';

signal fourth\_quarter : STD\_LOGIC :='0';

signal stop\_clk : STD\_LOGIC := '1';

signal sda\_inter\_reg : STD\_LOGIC := '1';

signal in\_reg : STD\_LOGIC\_VECTOR(7 downto 0);

signal out\_reg : STD\_LOGIC\_VECTOR(7 downto 0) := "00000000";

signal red\_value : STD\_LOGIC\_VECTOR(7 downto 0) := "00000000";

constant div : INTEGER := clk\_freq / i2c\_freq;

signal ld: STD\_LOGIC := '0';

signal sh : STD\_LOGIC := '0';

signal addr\_ld : STD\_LOGIC := '0';

signal first\_red : STD\_LOGIC := '0';

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

begin

divider: process(clk)

variable counter : INTEGER := 0;

begin

if rising\_edge(clk) then

if clk\_en = '1' then

if counter = div then

counter := 0;

else

counter := counter;

end if;

if counter = 0 then

clk\_i2c <= '1';

en\_p <= '1';

en\_n <= '0';

second\_quarter <= '0';

fourth\_quarter <= '0';

elsif counter = div / 4 then

clk\_i2c <= clk\_i2c;

en\_p <= '0';

en\_n <= '0';

second\_quarter <= '1';

fourth\_quarter <= '0';

elsif counter = div / 2 then

clk\_i2c <= '0';

en\_n <= '1';

en\_p <= '0';

second\_quarter <= '0';

fourth\_quarter <= '0';

elsif counter = 3 \* div / 4 then

clk\_i2c <= clk\_i2c;

en\_n <= '0';

en\_p <= '0';

second\_quarter <= '0';

fourth\_quarter <= '1';

else

en\_p <= '0';

en\_n <= '0';

clk\_i2c <= clk\_i2c;

second\_quarter <= '0';

fourth\_quarter <= '0';

end if;

counter := counter + 1;

else

counter := 0;

end if;

end if;

end process;

addr\_reg : process(clk) is

begin

if rising\_edge(clk) then

if addr\_ld = '1' then

old\_address <= address;

else

old\_address <= old\_address;

end if;

end if;

end process;

input\_reg : process(clk)

begin

if rising\_edge(clk) then

if ld = '1' then

in\_reg <= data\_in;

elsif sh = '1' then

in\_reg <= in\_reg(6 downto 0) & sda;

end if;

end if;

end process;

sda\_flip\_flop: process(clk)

begin

if rising\_edge(clk) then

if (st = start or st = ready) and second\_quarter = '1' then

sda\_inter\_reg <= sda\_inter;

elsif st = stop and second\_quarter = '1' then

sda\_inter\_reg <= '1';

elsif fourth\_quarter = '1' then

sda\_inter\_reg <= sda\_inter;

else

sda\_inter\_reg <= sda\_inter\_reg;

end if;

end if;

end process;

value\_read: process(clk)

begin

if rising\_edge(clk) then

if st = read and en\_p = '1' then

red\_value <= red\_value(6 downto 0) & sda;

else

red\_value <= red\_value;

end if;

end if;

end process;

output: process(clk)

begin

if rising\_edge(clk) then

if st = red then

out\_reg <= red\_value;

else

out\_reg <= out\_reg;

end if;

end if;

end process;

fsm: process(clk)

begin

if rising\_edge(clk) then

if rst = '1' then

st <= ready;

else

case st is

when ready =>

if en = '1' then

st <= start;

else

st <= ready;

end if;

when start =>

first\_red <= '0';

stop\_clk <= '0';

count <= 8;

if en\_n = '1' then

st <= command;

else

st <= start;

end if;

when command =>

if en\_n = '1' then

if count = 1 then

st <= slave\_ack;

else

count <= count - 1;

st <= command;

end if;

else

st <= command;

end if;

when slave\_ack =>

count <= 8;

if en\_n = '1' then

if rw = '0' then

st <= write;

else

st <= read;

end if;

else

st <= slave\_ack;

end if;

when write =>

if en\_n = '1' then

if count = 1 then

st <= written;

else

count <= count - 1;

st <= write;

end if;

else

st <= write;

end if;

when read =>

if en\_n = '1' then

if count = 1 then

st <= red;

else

count <= count - 1;

st <= read;

end if;

else

st <= read;

end if;

when written =>

count <= 8;

if en\_n = '1' then

if en = '0' then

st <= stop;

else

if rw = '0' and (address = old\_address) then

st <= write;

else

st <= start;

end if;

end if;

else

st <= written;

end if;

when red =>

count <= 8;

if en\_n = '1' then

if en = '0' then

st <= stop;

else

if rw = '1' and (address = old\_address) then

st <= read;

first\_red <= '1';

else

st <= start;

first\_red <= '1';

end if;

end if;

else

st <= red;

end if;

when stop =>

if en\_n = '1' then

st <= ready;

elsif en\_p = '1' then

stop\_clk <= '1';

else

st <= stop;

end if;

when others =>

st <= ready;

end case;

end if;

end if;

end process;

term <= '1' when st = written or st = red or st = slave\_ack else '0';

addr\_ld <= '1' when st = start else '0';

ld <= '1' when st = start or st = slave\_ack or st = written or st = red else '0';

sh <= '1' when (st = command and en\_n = '1') or (st = write and en\_n = '1') or (st = read and en\_p = '1') else '0';

scl\_inter <= '1' when st = ready else clk\_i2c;

sda\_inter <= '1' when st = ready or (st = red and first\_red = '1') else

'0' when st = start or st = stop or ( st = red and first\_red = '0')else

in\_reg(7) when st = command or st = write else

'1';

clk\_en <= '0' when st = ready else '1';

scl <= '0' when (scl\_inter = '0' and stop\_clk = '0') else 'Z';

sda <= '0' when sda\_inter\_reg = '0' else 'Z';

data\_out <= out\_reg;

end Behavioral;

## ***Controller\_I2C***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity I2C\_module is

Port ( clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

sda : inout STD\_LOGIC;

scl : inout STD\_LOGIC;

humidity : out STD\_LOGIC\_VECTOR (15 downto 0));

end I2C\_module;

architecture Behavioral of I2C\_module is

constant SENSOR\_ADDR : STD\_LOGIC\_VECTOR(6 downto 0) := "1000000";

signal rw : STD\_LOGIC := '0';

signal en : STD\_LOGIC := '0';

signal term : STD\_LOGIC := '0';

signal data\_in : STD\_LOGIC\_VECTOR (7 downto 0) := x"00";

signal data\_out : STD\_LOGIC\_VECTOR (7 downto 0) := x"00";

signal address : STD\_LOGIC\_VECTOR (6 downto 0) := "1000000";

type TIP\_STARE is (start, wait1, wait2, wait3, wait4, stat\_reg\_wr0, stat\_reg\_wr1, stat\_reg\_wr2, stat\_reg\_wr3,

addr\_point\_reg\_wr0, addr\_point\_reg\_wr1, recieve\_data0, recieve\_data1, recieve\_data2);

signal st : TIP\_STARE := start;

signal counter : INTEGER := 0;

signal to\_count : INTEGER := 0;

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

signal wait\_term : STD\_LOGIC := '0';

constant CONS\_30\_MS : INTEGER := 3\_000\_000;

constant CONS\_20\_MS : INTEGER := 2\_000\_000;

constant CONS\_10\_MS : INTEGER := 1\_000\_000;

signal sign : STD\_LOGIC := '0';

signal ld\_hum\_reg : STD\_LOGIC := '0';

signal humidity\_reg : STD\_LOGIC\_VECTOR(15 downto 0) := x"0000";

signal term\_delayed : STD\_LOGIC := '0';

begin

flip\_flop: process(clk)

begin

if rising\_edge(clk) then

term\_delayed <= term;

end if;

end process;

wait\_counter: process(clk)

begin

if rising\_edge(Clk) then

if wait\_en = '1' then

if counter = to\_count - 1 then

wait\_term <= '1';

counter <= 0;

else

counter <= counter + 1;

wait\_term <= '0';

end if;

else

counter <= 0;

wait\_term <= '0';

end if;

end if;

end process;

hum\_reg: process(clk)

begin

if rising\_edge(clk) then

if ld\_hum\_reg = '1' then

if sign = '0' then

humidity\_reg(15 downto 8) <= data\_out;

else

humidity\_reg(7 downto 0) <= data\_out;

end if;

else

humidity\_reg <= humidity\_reg;

end if;

end if;

end process;

DUT0: entity WORK.I2C\_master port map (

clk => clk,

rst => rst,

rw => rw,

en => en,

address => address,

data\_in => data\_in,

data\_out => data\_out,

scl => scl,

term => term,

sda => sda);

fsm: process(clk)

begin

if rising\_edge(clk) then

if rst = '1' then

st <= start;

else

case st is

when start =>

st <= wait1;

when wait1 =>

if wait\_term = '1' then

st <= stat\_reg\_wr0;

else

st <= wait1;

end if;

when stat\_reg\_wr0 =>

if term = '1' and term\_delayed = '0' then

st <= stat\_reg\_wr1;

else

st <= stat\_reg\_wr0;

end if;

when stat\_reg\_wr1 =>

if term = '1' and term\_delayed = '0' then

st <= stat\_reg\_wr2;

else

st <= stat\_reg\_wr1;

end if;

when stat\_reg\_wr2 =>

if term = '1' and term\_delayed = '0' then

st <= stat\_reg\_wr3;

else

st <= stat\_reg\_wr2;

end if;

when stat\_reg\_wr3 =>

if term = '1' and term\_delayed = '0' then

st <= wait2;

else

st <= stat\_reg\_wr3;

end if;

when wait2 =>

if wait\_term = '1' then

st <= addr\_point\_reg\_wr0;

else

st <= wait2;

end if;

when addr\_point\_reg\_wr0 =>

if term = '1' and term\_delayed = '0' then

st <= addr\_point\_reg\_wr1;

else

st <= addr\_point\_reg\_wr0;

end if;

when addr\_point\_reg\_wr1 =>

if term = '1' and term\_delayed = '0' then

st <= wait3;

else

st <= addr\_point\_reg\_wr1;

end if;

when wait3 =>

if wait\_term = '1' then

st <= recieve\_data0;

else

st <= wait3;

end if;

when recieve\_data0 =>

if term = '1' and term\_delayed = '0' then

st <= recieve\_data1;

else

st <= recieve\_data0;

end if;

when recieve\_data1 =>

if term = '1' and term\_delayed = '0' then

st <= recieve\_data2;

else

st <= recieve\_data1;

end if;

when recieve\_data2 =>

if term = '1' and term\_delayed = '0' then

st <= wait4;

else

st <= recieve\_data2;

end if;

when wait4 =>

if wait\_term = '1' then

st <= addr\_point\_reg\_wr0;

else

st <= wait4;

end if;

when others =>

st <= start;

end case;

end if;

end if;

end process;

address <= SENSOR\_ADDR;

wait\_en <= '1' when st = wait1 or st = wait2 or st = wait3 or st = wait4 else '0';

to\_count <= CONS\_20\_MS when st = wait1 else

CONS\_10\_MS when st = wait2 else

CONS\_10\_MS when st = wait3 else

CONS\_30\_MS when st = wait4 else 0;

rw <= '1' when st = recieve\_data0 or st = recieve\_data1 or st = recieve\_data2 else '0';

en <= '0' when st = start or st = wait1 or st = wait2 or st = wait3 or st = wait4 else '1';

data\_in <= x"02" when st = stat\_reg\_wr1 else

x"00" when st = stat\_reg\_wr2 else

x"00" when st = stat\_reg\_wr3 else

x"01" when st = addr\_point\_reg\_wr1 else

"1000000" & rw when st = stat\_reg\_wr0 or st = addr\_point\_reg\_wr0 or st = recieve\_data0 else

x"00";

sign <= '1' when st = recieve\_data1 else '0';

ld\_hum\_reg <= '1' when st = recieve\_data1 or st = recieve\_data2 else '0';

humidity <= humidity\_reg;

end Behavioral;

A diagram of a computer component

Description automatically generatedA white paper with writing on it

Description automatically generatedA diagram of a diagram

Description automatically generated with medium confidenceA diagram of a product

Description automatically generated with medium confidence

A diagram on a paper

Description automatically generated