**Facultatea de Automatica și Calculatoare**

**Calculatoare și Tehnologia Informației**



**Structura sistemelor de calcul**

**Implementarea procesorului**

**AM2901 de 4 biți**

**Profesor îndrumator: Realizat de:**

**Cristi Mocan Câmpean Casiana Ștefana**

**Grupa: 30238**

**28.05.2019**

**Cuprins**

[**1. Rezumat** 3](#_Toc9809642)

[**2. Introducere** 4](#_Toc9809643)

[**3. Fundamentare teoretică** 5](#_Toc9809644)

[**3.1 Caracteristici** 5](#_Toc9809645)

[**3.2 Arhitectura** 6](#_Toc9809646)

[**3.3 Microinstrucțiuni** 9](#_Toc9809647)

[**4. Proiectare şi implementare** 10](#_Toc9809648)

[**4.1 Unitatea Aritmetico-Logică** 10](#_Toc9809649)

[**4.2 Memoria RAM** 11](#_Toc9809650)

[**4.3 Program MicroInstrucțiuni** 12](#_Toc9809651)

[**4.4 A și B Latch** 12](#_Toc9809652)

[**4.5 Registrul Q** 13](#_Toc9809653)

[**4.6 MPG MonoPulse Generator** 13](#_Toc9809654)

[**4.7 SSD** 14](#_Toc9809655)

[**4.8 AMD** 15](#_Toc9809656)

[**5. Rezultate experimentale** 18](#_Toc9809680)

[**5.1 Unitatea Aritmetico-Logică testare** 18](#_Toc9809682)

[**5.2 Memoria RAM testare** 19](#_Toc9809683)

[**5.3 AMD testare** 19](#_Toc9809684)

[**6. Manual de utilizare** 23](#_Toc9809686)

[**7. Concluzii** 28](#_Toc9809826)

[**Bibliografie** 29](#_Toc9809830)

[**Anexe** 29](#_Toc9809836)

[**Cod pentru ALU** 29](#_Toc9809837)

[**Cod pentru memoria RAM** 32](#_Toc9809839)

[**Cod pentru program MicroInstructiuni** 33](#_Toc9809877)

[**Cod pentru AMD** 35](#_Toc9809878)

1. **Rezumat**

Implementarea procesorului AM2901de 4 biti pe o placa FPGA a fost tema proiectului. Procesorul face parte din familia de circuite integrate AM2900 create de catre Advanced Micro Devices(AMD). AM2901 a fost o unitate aritmetico-logică, fiind “nucleul” seriei, putând realiza operații binare și de shiftare pe 4 biți.

Obiectivul principal al proiectului a fost realizarea procesorului Am2901 si implementarea acestuia pe placa FPGA. Astfel, pentru realizarea proiectului s-au citit cu atentie documentatiile existente, s-a proiectat fiecare componenta necesara in programul Vivado 2018.3, scrisa in limbajul VHDL, s-a simulat tot in acest program, dupa care s-a pus pe placuta Basys3 si Nexys4 DDR.

Prin urmare, in urma implementarii procesorului Am2901, s-a realizat procesorul cu un set de instructiuni, fiecare dintre acestea realizand una dintre operatiile posibile, iar pentru utilizarea acestui proiect a fost creata o sectiune cu un manual de utilizare.

**2. Introducere**

Tema proiectului reprezintă implementarea procesorului AM2901 de 4 biți pe o placă de dezvoltare FPGA, folosind o proiectare structurală utilizând limbajul FPGA.

Acest procesor face parte din familia de circuite integrate AM2900, create in 1975 de către Advanced Micro Devices(AMD). Acestea au fost contruite cu dispositive bipolare, pe 4 biți, proiectate sa fie folosite ca și componente modulare, fiecare reprezentând aspect diferite ale unei unități de control a unui calculator(CCU). AM2901 a fost o unitate aritmetico-logică, fiind “nucleul” seriei, putând realiza operații binare și de shiftare pe 4 biți. Această familie de procesoare a fost foarte importantă pentru realizarea mai multor mașinării, precum minicomputere, sisteme de grafică pe calculator, alte procesoare si calculatoare industriale.[2]

Obiectivul principal al acestui proiect este implementarea un procesor AM2901, folosind o plăcuță FPGA. Astfel, trebuie mai întâi parcursă documentația acestui procesor, tebuie studiate diagramele necesare implementării, mai apoi trebuie proiectate toate componentele necesare acestui procesor în VHDL, urmând cu exactitate schemele și diagramele existente.

Soluția propusă reprezintă proiectarea în VHDL a componentelor procesorului AM2901 urmând diagramele din documentația procesorului [1].

In următoarele secțiuni se vor prezenta mai amănunțit fundamentele teoretice legate de proiect, ce modele, metode si tehnologii vor fi utilizate, plus referințe la documentațiile existente. Mai apoi, partea de proiectare și implementare va conține descriere fiecare etapă parcursă pentru realizarea obiectivelor proiectului, incluzând metoda utilizată, soluția aleasă, schema bloc și arhitectura generală, detalii de implementare și manual de utilizare. Secțiunea cu rezultate experimentale va demonstra faptul ca sistemul proiectat a fost implementat cu succes și rezultatele obținute sunt valide. Acesta va conține detalii despre instrumentele de proiectare utilizate, procedura de testare utilizată, capturi de ecran si comparații între mai multe implementări. La final, urmând secțiunea cu concluziile, care va conține un sumar al raportului proiectului.

**3. Fundamentare teoretică**

Acest procesor face parte din familia de circuite integrate AM2900, create in 1975 de către Advanced Micro Devices(AMD). Acestea au fost contruite cu dispositive bipolare, pe 4 biți, proiectate sa fie folosite ca și componente modulare, fiecare reprezentând aspect diferite ale unei unități de control a unui calculator(CCU). AM2901 a fost o unitate aritmetico-logică, fiind “nucleul” seriei, putând realiza operații binare și de shiftare pe 4 biți. Este proiectat pentru a fi folosit in CPU, in controlere periferice, microprocesoare programabile si multe alte aplicații.[2]

Pe lângă această variantă de procesor, mai există încă 3 modele, AM2901A, AM2901B și AM2901C, AM2901A fiind mai rapid, având rezultate mai bune. AM2901B este cu 25% mai rapid decât AM2901A si cu 50% mai rapid decât AM2901, iar AM2901C este de 33% mai rapid decât AM2901B.[3][4]

## **3.1 Caracteristici**

Principalele caracteristici ale microprocesorului sunt: Unitatea aritmetico-logica are 8 funcții, mai exact adunare, scadere, funcții logice pe doi operanzi. Selecția de date este flexibilă, shiftările pe stanga sau dreapta sunt indepenente de ALU, deoarece acestea necesită doar un ciclu. Pe langă acestea, AM2901 are patru steaguri de stare, carry, overflow, zero si negative, se poate extinde, putând fi conectat împreună cu oricâte alte AM2901 pentru cuvinte mai lungi, iar în final, procesorul este microprogramabil, existând 3 grupuri de 3 biți pentru fiecare operand sursă, funție ALU și control destinație.[1]

Dispozitivul, ilustrat in figura 1, este conținut din 16 registrii de 4 biți, o unitate aritmetico-logică rapidă de 4 biți, și din alte componente de shiftare, decodificare și multiplexoare. Micro-instrucțiunea de 9 biți este organizată în 3 grupuri de 3 biți fiecare și selectează operanzii, funcțiile si registrii destinație din ALU. Procesorul se poate cascada, are ieșiri de 3 stari și pot rezulta mai multe steaguri de stare din ALU.[2]

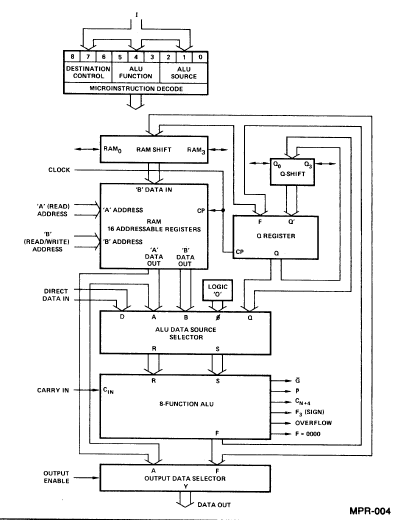


Figura 1 ilustrează diagrama bloc a procesorului AM2901[1]

## **3.2 Arhitectura**

Arhitectura documentatiei este ilustrată în detaliu în figura 2. AM2901 fiind pe 4 biți, toate date vor fi pe 4 biți, acesta se poate cascada. Cele mai importante componente din diagrama detaliată sunt unitatea aritmetico-logică și memoria RAM. Unitatea aritmetico-logică pe 4 biți poate citi date din memoria internă, din conexiuni externe sau din registrul Q și poate să realizeze adunare, scădere, or, and, xor, xnor. Memoria are 16 adrese, 2 divizări de 4 biți, A și B, 2 porturi de ieșire A și B, poate să citească din A sau B și adresa de scriere pentru B este folosita pentru ambele adrese de scriere A și B. Memoria este scrisă și citită pe fiecare ciclu de ceas, scrierea se face pe frontul crescător, iar citirea pe frontul descrescător. Altă component este registrul Q, care este pe 4 biți, și poate shifta la stanga sau la dreapta 1 bit pe un ciclu de ceas și poate primi intrarea de la ieșirea ALU sau de la el însuși.

Intrarea de Clock controlează memoria RAM, registrul Q, și A și B. Datele intră in registrul Q pe frontrul descrescător, iar cand intrarea de Clock e pe HIGH, datele intră prin A și B în memoria RAM. Când intrarea de Clock e LOW, A si B se închid și păstrează ultimele date.

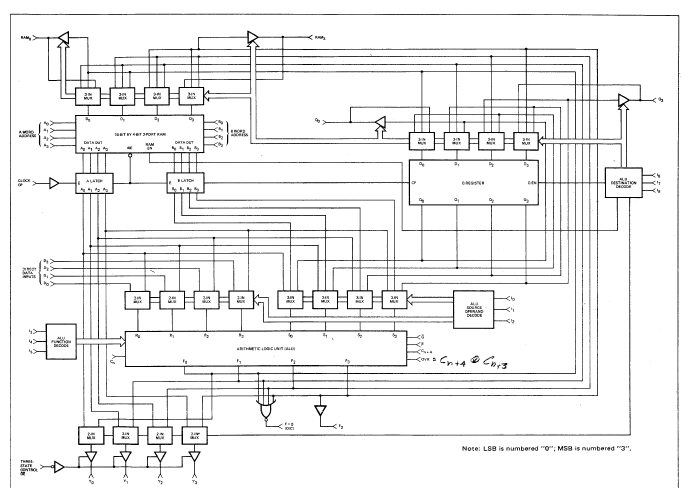
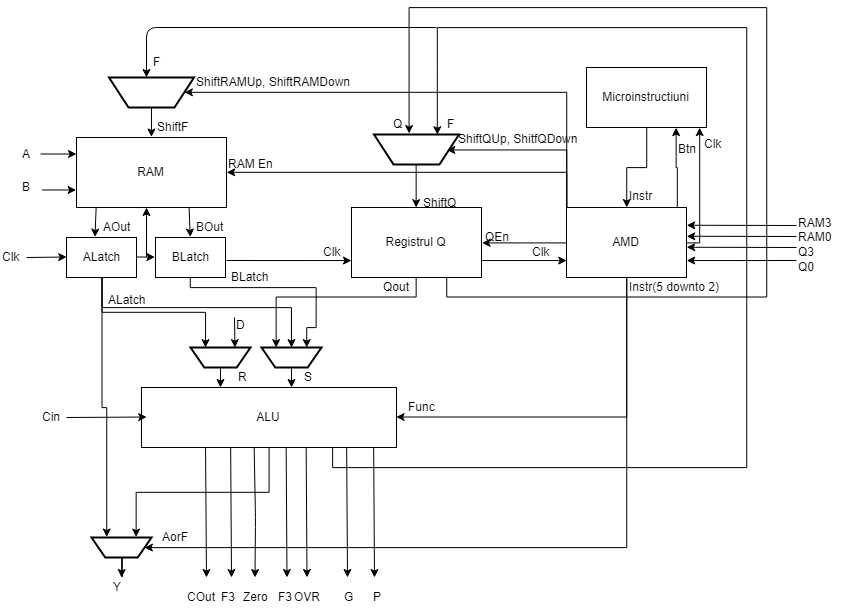


Figura 2 ilustrează digrama bloc detaliată a procesorului AM2901 [1]



Arhitectura finala a proiectului este reprezentata de figura precedenta. Aici am reprezentat in plus componenta principala AMD, care este cea care genereaza toate semnalele principale.

## **3.3 Microinstrucțiuni**

Intrările microinstrucțiunilor folosite pentru a selecta opreranzii sursă din ALU sunt I0, I1, I2, combinațiile acestora fiind ilustrate in primul tabel din figura 3. În al2-lea tabel sunt definite combinațiile pentru microinstrucțiunile I3, I4, I5 care sunt folosite pentru a selecta funcția ALU. În ultimul tabel sunt definite combinașiile microinstrucțiunilor I6, I7, I8 care sunt folosite pentru a selecta destinațiile din ALU, care pot fi iesirile dispozitivului, memoria RAM sau registrul Q.

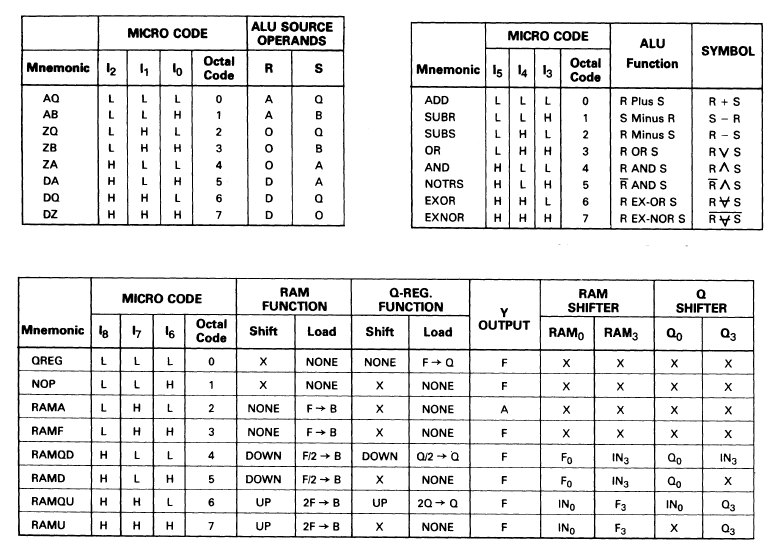


Figura 3 ilustrează tabelele pentru controlul operanzilor sursă din ALU, funcțiile de control din ALU și controlul destinație din ALU, fiind detaliate codurile acestora.[1]

**4. Proiectare şi implementare**

Pentru proiectarea și implementarea procesorului AM2901, s-au parcurs cu atenție infomațiile și diagramele existente. Din aceste date, putem stabili componentele necesare proiectarii dispozitivului, acestea fiind:

* Unitatea Artimetico-Logică
* Memoria RAM
* Memoria Programului de MicroInstructiuni
* A și B Latch
* Registrul Q
* MUX 3:1
* MUX 2:1
* AMD

Fiecare componentă va fii descrisă in detaliu în subcapitolele următoare:

## **4.1 Unitatea Aritmetico-Logică**



Figura 4 ilustreză diagrama bloc a unității aritmetico-logice

Această componentă, conform documentației, poate realiza 3 operații binare aritmetice și 5 operații logice pe intrările de 4 biți R și S. Intrarea R vine de la un multiplexor 2:1, fiind posibilă sa fie egală fie cu portul A de la RAM sau cu date directe, iar intrarea S vine de la un multiplexor 3:1, putând fi egală fie cu portul A sau B din RAM sau cu iesirea de la registrul Q. Ca și intrare mai este Cin, Carry In, iar ca și ieșiri avem F, rezultatul produs de ALU după aplicarea unei operații asupra lui R și S, și incă 6 semnale cu rol de flag, G, carry generate, P, carry propagate, active pe 0, Cn+4, Carry out activ pe 1, F3 care detectează un rezultat negativ sau positiv, ținând cont de bitul de semn, Zero care e activ atunci cand rezultatul e zero si OVR, care detectează overflow.

Toate semnalele de flag si operațiile din ALU au fost realizate si calculate confrom

tabelului urmator:

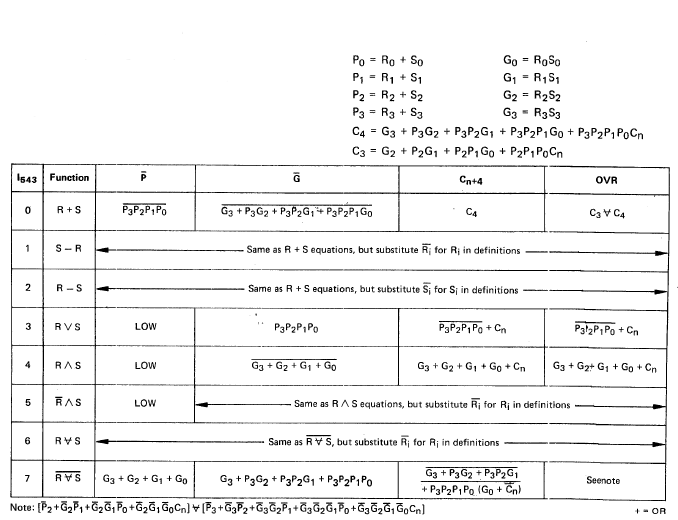


Figura 5 reprezintă modul în care se calculează semnalele de flag de la ALU, în funcție de de biții 5, 4, 3 ai microinstrucțiunilor

## **4.2 Memoria RAM**

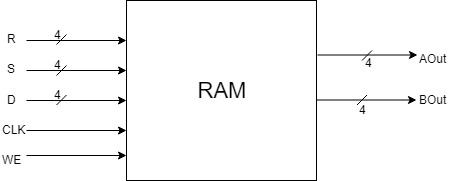


Figura 6 reprezintă diagram bloc a memoriei RAM

Memoria RAM reprezintă spațiul de stocare central al procesorului AM2901, fiind asemănător cu un bloc de registrii. Majoritatea operațiilor implică folosirea sau modificarea datelor stocate în memorie. Această componentă conține:

5 intrări: A, B, D, clk și WE. A reprezintă adresa de unde putem citi, iar B reprezintă adresa de unde putem citi, dar si scrie. CLK este semnalul de ceas, iar WE peste Write Enable, care ne permite sa scriem la adresa B datele de la intrarea D.

2 ieșiri: Aout si Bout, care sunt datele care se află la adresele registrilor de la intrări.

## **4.3 Program MicroInstrucțiuni**



Figura 7 reprezintă diagram bloc a memoriei programului

Această componentă este programul principal al procesorului, unde vor fi stocate intr-o memorie ROM toate microinstrucțiunile.

Ca și intrări vor fi un clk, reprezentând semnalul de ceas și un buton, mereu cand vom apăsa vom trece la instrucțiunea următoare.

Ca ieșire este instrucțiunea citită din memorie pe 9 biți.

## **4.4 A și B Latch**



Figura 8 reprezintă diagram bloc a componentei Latch

Aceste componente au ca și scop să păstreze datele din RAM cât timp semnalul de clk este pe LOW. Astfel, eliminăm posibile condiții de cursă care ar putea apărea cât timp noi date se scriu in memorie.

Avem 2 intrări, CLK și D, și o ieșire Q. D vor fi datele din portul A sau B.

Intrararea D vine de la memoria RAM, datele din Aout sau Bout ajung in A Latch sau B Latch. Iesirile din latch-uri vor merge la multimplexoare, mai apoi catre unitatea aritmetico-logică.

## **4.5 Registrul Q**



Figura 9 reprezintă diagram bloc a registrului Q

Această componentă reprezinta un registru pe 4 biți, fiind una dintre destinațiile din Unitatea Aritmetico-Logică. În registru se vor reține datele din ALU sau se vor reține datele shiftate din acest acest registru.

Avem ca și intrari CLK de la procesor, un enble Qen, care doar cand va fii 1 vom putea scrie date in registru, D, pe 4 biți, care vine de la un multiplexor 3:1, iar ca ieșire avem Q, pe 4 biți care reprezintă datele din registru.

## **4.6 MPG MonoPulse Generator**

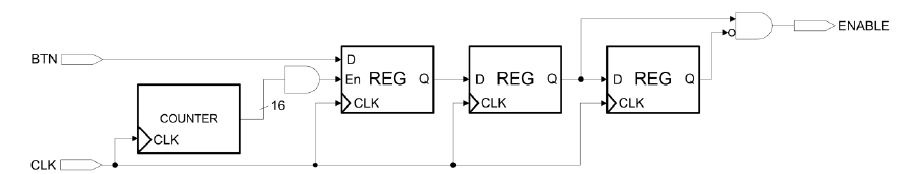


Figura 10 reprezinta digrama bloc a unui Generator de Impuls sisncron[5]

Rolul primului registru, împreună cu numărătorul, este de a asigura robustețe la utilizarea butoanelor uzate fizic, când pot să apară activări multiple ale semnalului ENABLE la o apăsare de buton. În funcție de uzură, este posibil să fie nevoie de mai mulți biți ai numărătorului (17-20+) pe care să se aplice un ŞI logic, astfel încât să se mărească intervalul de eșantionare al butonului.[5]

## **4.7 SSD**

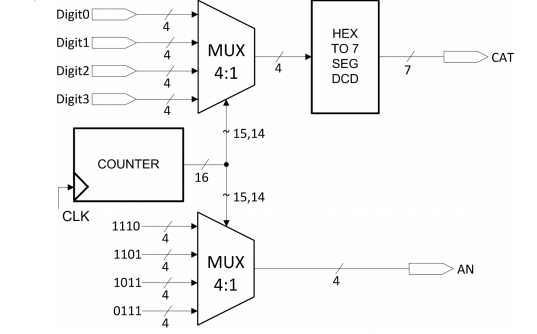


Figura 11 reprezinta digrama bloc a afisorului pe 7 segmente. [5]

Această componenta folosește șapte leduri pentru fiecare cifră; fiecare cifră este activată de un semnal de anod. Toate semnalele interfeței SSD (7 semnale comune de catod și 4 semnale distincte de anod) sunt active pe 0. Semnalele de catod controlează ledurile care se aprind de pe acele cifre care au semnalul de anod activ (de exemplu dacă se activează toate 4 anodurile, atunci se va afișa aceeași cifră pe cele 4 poziții). Perioada maximă de reîmprospătare (refresh) este astfel calculată încât ochiul uman să nu perceapă aprinderea și stingerea succesivă a fiecărei cifre de pe SSD (16 ms <=> 60 Hz). Se realizează astfel o afișare ciclică acifrelor (la un moment dat doar o cifră este afișată, dar ochiul nu percepe acestaspect) [5]

## **4.8 AMD**

Această componentă reprezintă partea principală a procesorului AMD. În această parte sunt inițializate toate componentele și sunt realizate operațiile necesare funcționarii procesorului.

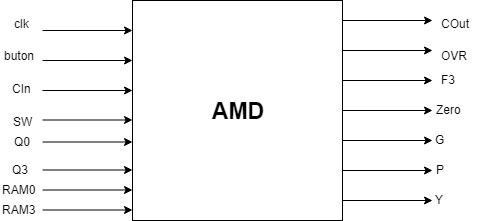


Figura 12 reprezintă diagram bloc a componentei AMD

Intrările procesorului sunt Clk, clock-ul principal al procesorului, buton, butonul care va fi utilizat pentru parcurgerea secvențială a microinstrucțiunulor din memorie, CIn, fiind carry in utilziat in ALU. Intrarea SW care este pe 16 biti, va fii impartita in 3 parti, in A, B si D, fiecare valoarea fiind data de la un switch. A este 4 biți si este folosită in memoria RAM, pentru a determina de unde citim datele, B, pe 4 biti care este folosită tot in memoria RAM, pentru adetermina unde scriem si de unde citim datele. D, tot pe 4 biti utilizat tot in memoria RAM, reprezinta ce date vom da in procesul de scriere in memorie.

Ca si iesiri, avem iesirile de la Unitatea Aritmetico-Logica, mai exact semnalele Cout, OVR, F3, Zero, G și P, rolul acestora fiind explicat in sectiunea 4.1.

Componenta MicroInstructions, declarată în interiorul entității AMD, are ca ieșire Instr, declarat ca si semnal in AMD, care reprezinta microinstructiunea procesorului. Aceasta este impartita in 3 partiȘ Op( I2I1I0), Func(I5I4I3) și Dest(I8I7I6), semnale pe 3 biți. Pe baza semnalului Op este declarat un proces care stabileste care sunt operanzii R și S din Unitatea Aritmetico-Logică. Semnalul Func va fi o intrare pentru ALU, care reprezintă care dintre operații se va executa în ALU, iar cu ajutorul semnalului Dest se creează alt proces, care va stabili valoarea tuturor semnalelor necesare pentru stabilirea desinatiei din ALU. Aceste semnale sunt urmatoarele:

* ShiftRAMDown și ShiftRAMUp, care se folosesc pentru aflarea valorii intrării în memoria RAM, pe baza lor, aceasta putând fi shiftata in sus, in jos sau neshiftată.
* ShiftQDown și ShiftQUp, care se utilizează pentru stabilirea valorii intrarii registrului Q, intrarea putând di shiftata in sus sau in jos sau neshiftată.
* QEn, semnal care este utilizat la registrul Q, daca este activ atunci se poate scrie in registru.
* EnRAM, semnal care este utilizat la memoria RAM, daca este activ atunci se poate scrie in memorie.
* AorF, semnal care stabileste ieșirea procesorului, dacă semnalul este 1 atunci Y este sgal cu valoarea din portul A, altfel este egal c F, iesirea din ALU
* RAM0 și RAM3 sunt folosite în procesul de shiftare al intrării din memoria RAM
* Q0 și Q3 sunt folosite în procesul de shiftare al intrării din registrul Q

În continuare, în programul principal sunt declarate doua procese care au ca rol stabilirea intrarii memoriei RAM si a registrului Q. Ambele funționează ca un mltimplexor 3:1.

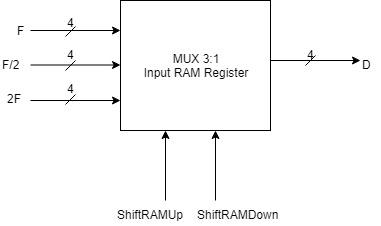


Figura 13 reprezintă MUX-ul 3:1pentru stabilirea intării B a memoriei RAM

Primul caz reprezintă cel în care se determină intrarea la memoria RAM. Aceasta poate fi ori rezultatul Unității Aritmetico-Logice(intrarea pe 4 biți F), ori acest rezultat shiftat in jos(F/2) ori in sus(2F). Semnalele de selecție ale multiplexorului sunt ShiftRAMUp și ShiftRAMDown, în funție de aceste se va selecta una dintre cele 3 intrari.

ShiftRAMUp este 0, ShiftRAMDown este 0, ieșirea D va lua valoarea F;

ShiftRAMUp este 0, ShiftRAMDown este 1, ieșirea D va lua valoarea F/2;

ShiftRAMUp este 1, ShiftRAMDown este 0, ieșirea D va lua valoarea 2F;

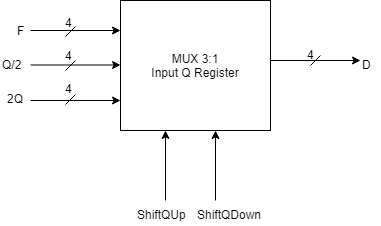


Figura 14 reprezintă MUX-ul 3:1 pentru stabilirea intării B a registrului Q

Al doilea caz reprezintă cel în care se determină intrarea la registrul Q. Aceasta poate fi ori rezultatul Unității Aritmetico-Logice(intrarea pe 4 biți F), ori conținutul registrului shiftat in jos(Q/2) sau in sus(2Q). Semnalele de selecție ale multiplexorului sunt ShiftQUp și ShiftQDown, în funție de aceste se va selecta una dintre cele 3 intrari.

ShiftQUp este 0, ShiftQDown este 0, ieșirea D va lua valoarea F;

ShiftQUp este 0, ShiftQDown este 1, ieșirea D va lua valoarea Q/2;

ShiftQUp este 1, ShiftQDown este 0, ieșirea D va lua valoarea 2Q;

Rezultatul principal al procesorului, ieșirea pe 4 biți Y este determinată de semnalul AorF, valoarea sa fiind stabilită de biții I8I7I6 ai microinstrucțiunii. Aflarea iesirii functionează ca un multiplexor 2:1, în care intrările sunt A si F, iar semnalul de selecție este AorF. Dacă AorF este ‘1’ atunci Y ia valoarea lui portului A din memoria RAM, altfel ia valoarea rezultatului F din Unitatea Aritmetioc-Logică.

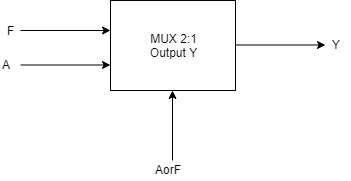


Figura 15 reprezintă MUX-ul 2:1pentru stabilirea iesirii Y a procesorului AMD

**5. Rezultate experimentale**

Instrumentele de proiectare utilizatate pentru realizarea testelor au fost simulatorul din programul Vivado v2018.3.

Procedura de testare este bancul de test. Pentru fiecare componenta s-a scris unul sau mai multe cazuri pentru verificarea rezultatelor.

## **5.1 Unitatea Aritmetico-Logică testare**

Aceasta componenta a fost testate cu ajutorul unui banc de test. A fost testat pentru fiecare dintre cele 8 operatii posibile, mai exact adunare, 2 scaderi, or, and, notsr, exnor si exor. De asemenea s-au verificat si corectitudinea rezultatelor de tip flag.

In acesta captura de ecran, R si S reprezinta operanzii, Operation reprezinta intrarea care defineste ce operatie se face in ALU, CIn este Carry in, F este iesirea componentei. COut, OVR, F3, Zero, G, P sunt semnalele care ies din ALU.

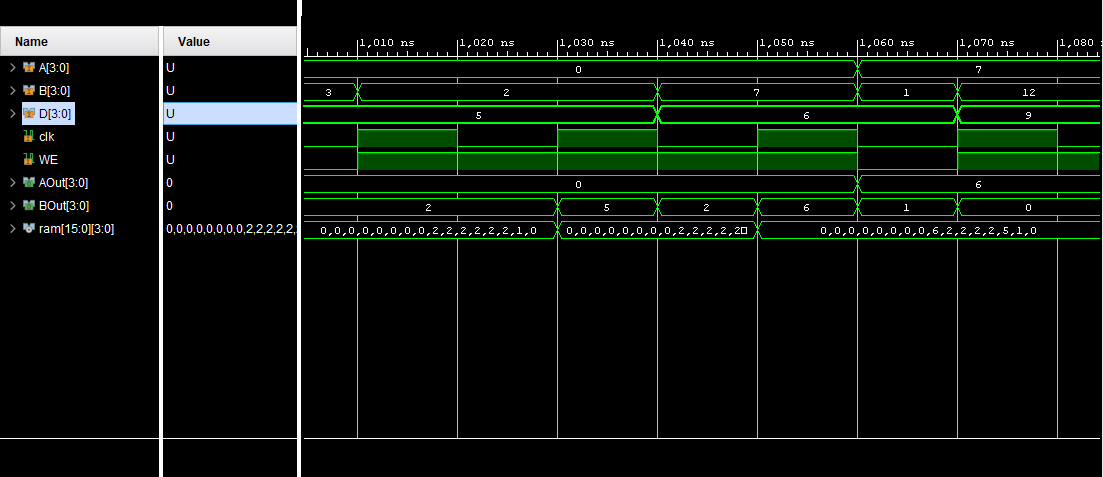
Pentru a testa aceasta component am intampinat multe dificultati, deoarece au trebuit facute multe schimbari la codul scris pentru a putea afisa toate rezultatele dorite.



## 

## **5.2 Memoria RAM testare**

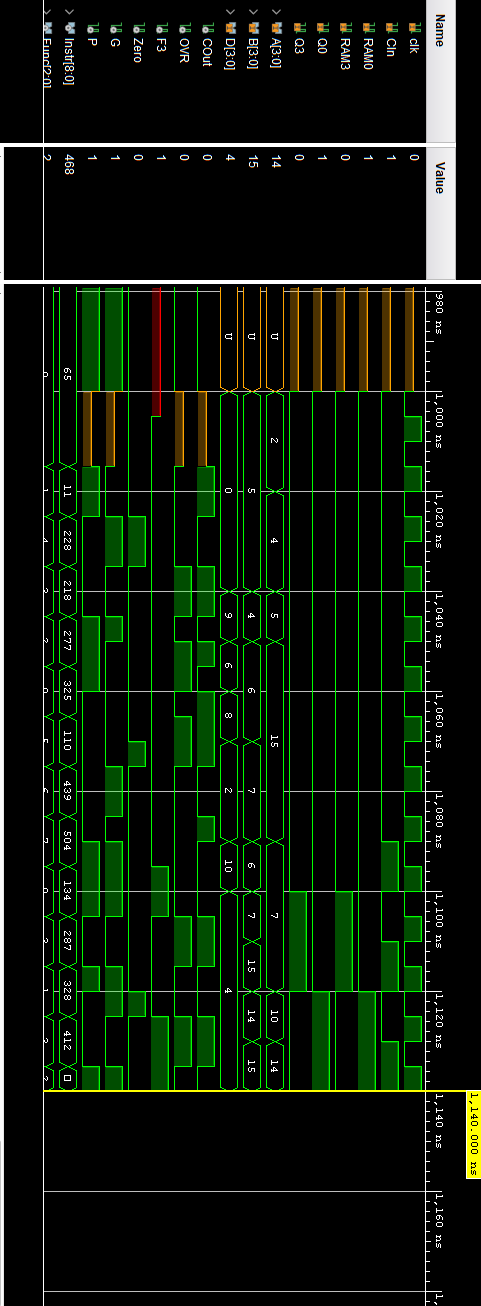
Pentru testarea acestei componente s-a folosit un banc de test. In memoria noastra RAM putem sa scriem sau sa citim date, astfel am simulat mai multe cazuri pentru aceste functionalitati. Prima data am citit din memorie datele, dupa care am realizat doua operatii de scriere la adrese diferite, mai apoi cititnd de la acea adresa.

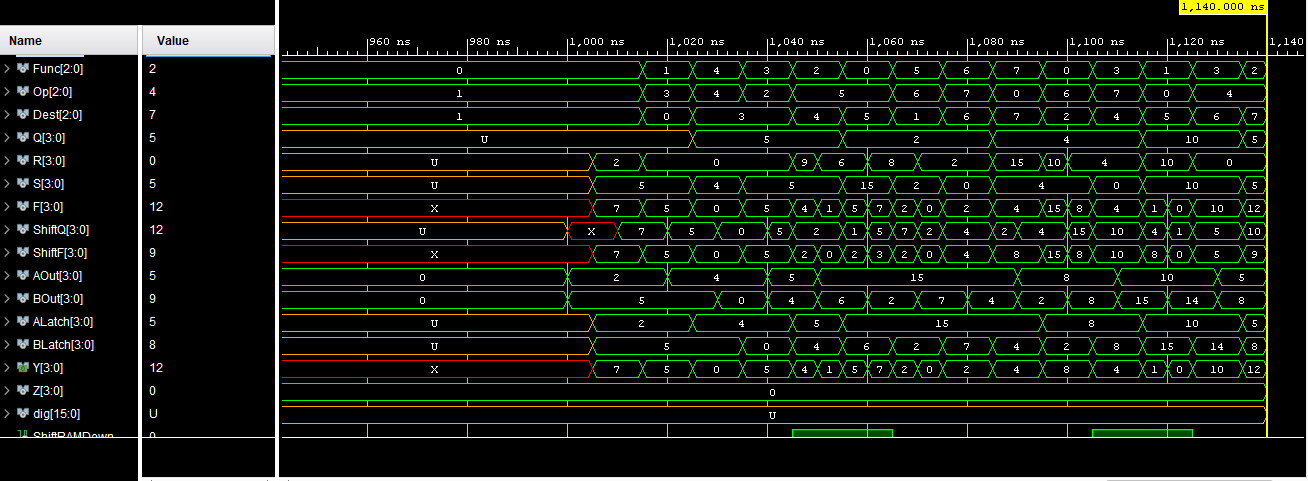


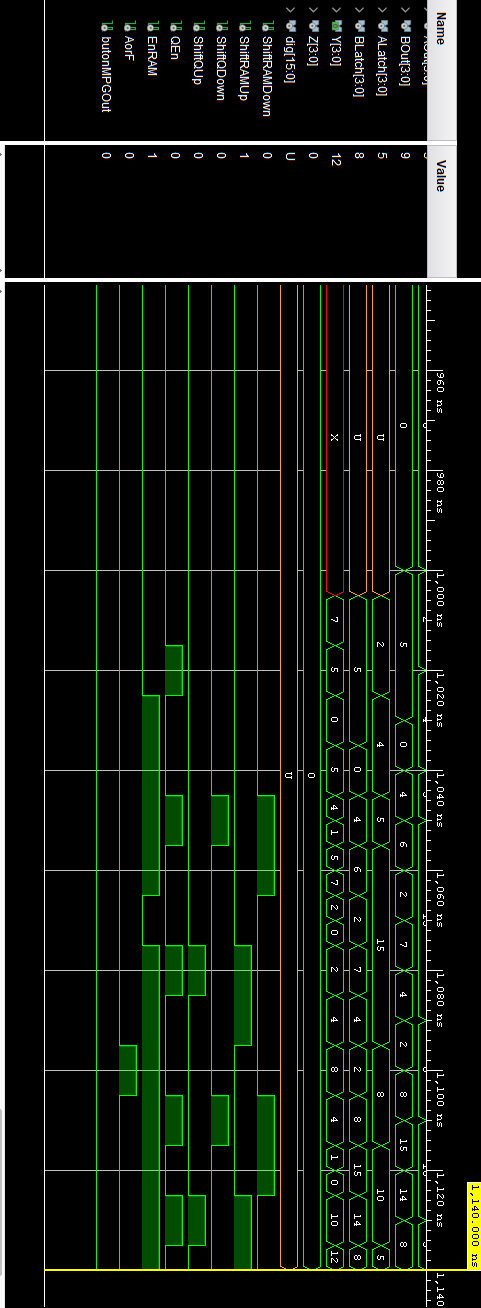
## **5.3 AMD testare**

Pentru testarea acestei componente s-a folosit de asemenea un banc de test. S-au dat diferite semnale pe intrarile A, B, D, in functie d e acestea putand fi observate diferitele operatii efectuate.

Semnalele principale din simulare sunt uramtoarele: CIn(Carry in din ALU), RAM3, RAM0, Q3, Q0, semnalele pentru cazurile in care se realizeaza shiftarea, A, B, D, intrarile procesului, A si B fiind adrese de memorie de unde se va citi valorile din memoria RAM, D ffind intrare pe 4 biti cu care se vor realiza diverse operatii. COut, OVR, Zero, F3, G si P sunt semnalele calculate in ALU. Instr reprezinta instructiunea la care suntem in memoria de instructiuni, Dest, Func si Op si fiind Instr(8 downto 6), Instr(5 downto 3), Instr(2 downto 0). Q reprezinta valoarea din registrul Q. R si S sunt intrarile din ALU, F este reultatul calculat din ALU. AOut si Bout sunt valorile care ies din ALU, iar ALatch si BLatch sunt valorile care ies din latch-urile A si B. ShiftRAMUp, ShiftRAMDown, ShiftQUp si ShiftDown sunt semnalele necesare pentru realizarea shiftarii. AorF este semnalul care decide daca pe Y va fii valoarea de la adresa data de A, sau rezultatul din ALU. QEn si RAMEn sunt semnalele de intrare pentru memoria RAM si pentru registrul Q pentru scrierea datelor. Y este rezultatul final care se va afisa pe placuta.



****



**6. Manual de utilizare**

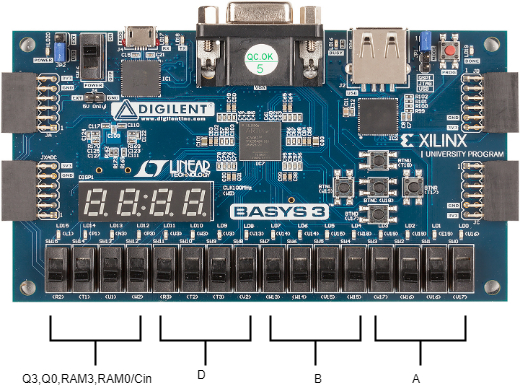
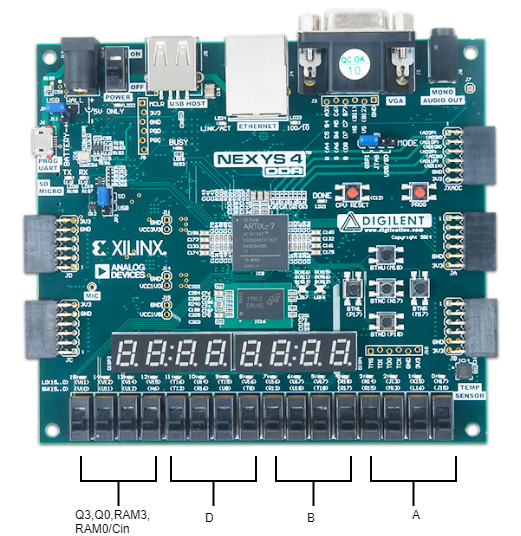


Figura reprezinta placuta Basys3. Pentru a utiliza placuta, trebuie folosite cele 16 switch-uri ale placutei. In figura sunt declarate ce reprezinta switch-urile. Pentru a folosi placuta si pentru a parcurge secvential instructiunile procesorului, trebuie apasat butonul din mijloc al placutei, iar in cazul in care dorim sa reincepem parcugerea programului de instructiuni, puntem apasa butonul de sus de de pe placuta. Astfel vom ajunge din nou la instructiunea cu numarul 0. De asemenea, placuta are si leduri care se afla deasupra switch-urilor, fiecare LED fiind un semnal de iesire din unitatea aritmetico-logica. Ledul 0 reprezinta Carry Out, ledul 1 Overflow, ledul 2 F3, bitul 3 rezultatului din unitatea aritmetico-logica, ledul 3 semnalul de Zero, ledul 4 G si ledul 5 P.

Exact aceleasi intrari si iesiri sunt reprezentate si pe placuta Nexys4 DDR, aceasta putand fi folosita in exact acelasi fel.



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nr instr | **A** | **B** | **D** | **Q** | **Y** | **CIn** | **Ram0, Ram3** | **Q0,Q3** |
| 1. | **2** | **5** | **0** | **0** | **7** | **0** | **0, 0** | **0, 0** |
| 2. | **0** | **5** | **0** | **5** | **5** | **0** | **0, 0** | **0, 0** |
| 3. | **4** | **5** | **0** | **5** | **0** | **0** | **0, 0** | **0, 0** |
| 4. | **4** | **5->4** | **0** | **5** | **5** | **0** | **0, 0** | **0, 0** |
| 5. | **5** | **4->6** | **9** | **2** | **4** | **0** | **0, 0** | **0, 0** |
| 6. | **15** | **6** | **6** | **2** | **5** | **0** | **0, 0** | **0, 0** |
| 7. | **15** | **6->7** | **8** | **2** | **2** | **0** | **0, 0** | **0, 0** |
| 8. | **15** | **7** | **2** | **4** | **2** | **0** | **0, 0** | **0, 0** |
| 9. | **15** | **7->6** | **2** | **4** | **4** | **0** | **0, 0** | **0, 0** |
| 10. | **7** | **6->7** | **10** | **4** | **8** | **1** | **0, 0** | **0, 0** |
| 11. | **7** | **7->15** | **4** | **10** | **4** | **0** | **0, 1** | **0, 1** |
| 12. | **7** | **15->14** | **4** | **10** | **1** | **1** | **0, 1** | **0, 1** |
| 13. | **10** | **14->15** | **4** | **5** | **10** | **1** | **1, 0** | **1, 0** |
| 14. | **14** | **15** | **4** | **5** | **12** | **0** | **1, 0** | **1, 0** |

Pentru a utiliza procesorul AMD, trebuie sa porniti placa, sa puneti pe switch-urile A, B, D, CIn, RAM0, RAM3, Q0, Q3 pe valorile potrivite conform tabelului, si sa apasati butonul pentru a parcurge secvential urmatoarele instructiuni.

Momentan in memora RAM, la fiecare dintre cele 16 adrese avem 16 numere pe 4 biti, pentru usurinta la adresa i se afla numarul i. Deci la adresa 0 se afla adresa 0, tot asa pana adresa 15, unde se afla numarul 15.

1. **0=>B"001\_000\_001" NOP\_ADD\_AB**

Prima data facem o adunare intre A si B. Pe intrarea A si B putem pune orice valori ca sa observam ca se poate face o adunare. Dupa care apasam butonul pentru urmatoarea instructiune.

1. **1=>B"000\_001\_011", --QREG\_SUBR\_ZB B=5 , F=5, Y=5 Q=5**

La aceasta instruciune il punem pe B sa fie 5. Avem ca si operatie scadere intre B si Z, adica intre 5 si 0. Astfel se va afisa rezultatul 5. Tot aici incarcam valoarea rezultatului F in registrul Q. Deci in registrul Q va fii valoarea 5.

1. **2=>B"011\_100\_100", --RAMF\_AND\_ZA A=4 F=0 B=5 RAM(5)=0 Y=0**

Aceasta instructiune face un load in memoria RAM. Se va incarca valoarea rezultatului din ALU, adica 0 la adresa data de B, adica 5. Deci in memorie la adresa 5, va fii valoarea 0. Ca si operatie vom face un AND intre Z si A, adica intre 0 si 4, deci se va afisa 0.

1. **3=>B"011\_011\_010", --RAMF\_OR\_ZQ Q=5 F=5 B=5 RAM(5)=5**

Aici vom face tot o incarcare in memorie, la adresa data de B, adica de 5, se va incarca valoarea rezultatului obtinut in ALU, (F=5) adica la adresa 10 va fi valoarea 5. Ca si operatie, vom realiza un OR intre Z si Q, deci intre 0 si 5, astfel se va afisa 5. Inainte de a apasa butonul pentru a trece la urmatoarea instructiune, vom seta B pe 4, D pe 9 si A pe 5.

1. **4=>B"100\_010\_101", --RAMQD\_SUBS\_DA D=9 A=5 D-A=F=4 Y=4 B=4 RAM(4)=4/2=2**

In acest pas se activeaza semnalul de shiftare la dreapta in Q, deci in Q vom avea valoarea 5/2, adica 2. Tot aici se realizeaza o shiftare la dreapta si in RAM, deci la adresa data de B, adica la adresa 6 va fii rezultatul din ALU imparit la 2, adica 4/2=2. Ca si operatie realizam o scadere intre D si A, intre 9 si 5, astfel se va afisa 4. Inainte de a trece la urmatoarea instructiune il punem pe B pe 6.

1. **5=>B"101\_101\_110", --RAMD\_ADD\_DA D=6 A=15 F=5 Y=5 B=6 RAM(6)=2**

Aici vom seta D=9, A=5. Aici se realizeaza doar shiftarea la dreapta a rezultatului actual din F(2) care va merge in RAM la adresa data de B, deci la adresa 6 vom avea valoarea 5/2=2. Ca si operatie facem o adunare intre A si D, adica intre 6 si 15. Se va afisa valoarea 5.

1. **6=>B”001\_101\_110” NOP\_NOTRS\_DQ D=8 Q=2 Y=2**

La aceasta instructiune se realizeaza un NOTRS intre D si Q(not(D) AND Q), D il setam pe 8, Qeste 2, astfel se va afisa valoarea 2. Tot aici il setam pe B ca fiind 7.

1. **7=>B"110\_110\_111", --RAMQU\_EXOR\_DZ D=2 F=2 B=7 RAM(7)=2\*2=4 Y=2 Q=2**

La se realizeaza shiftarea la stanga in registrul Q, deci Q va fii valoarea 2\*2=4. In RAM, la adresa data de B=7, va fii rezultatul din ALU(F=2) shiftat la stanga, deci vom avea valoarea 4. Ca si operatie vom realiza un XOR intre D si Z, adica intre 2 si 0, deci se va afisa 2.

1. **8=>B"111\_111\_000", --RAMU\_EXNOR\_AQ A=15 Q=4 F=4 Y=4 B=7 RAM(7)=4\*2=8 Q=2\*2=4**

In RAM, la adresa data de B=14, vom avea rezultatul din ALU, adica 4, shiftat la stanga, deci vom avea valoarea 8. Ca si operatie facem EXNOR intre A=15 si Q=4, deci se va afisa 4. Inainte sa apasam butonul sa trecem la instructiunea urmatoare, il setam pe B sa fie 6.

1. **9=>B"010\_000\_110",--RAMA\_ADD\_DQ Q=4 D=10 CARRY=1 A=7 Y=8 B=6 RAM(6)=15**

La aceasta instructiune realizam adunarea cu activarea carry-ului. Avand in registrul Q valoarea 4, si la D valoarea 10, cu 1 pe Carry atunci ca si rezultatul avem F=15. Pentru ca avem RAMA, atunci ca si iesire se va afisa valoarea de la adresa data de A(7), adica se va afisa RAM(14)=8. Inainte sa apasam butonul pentru a trece la instructiunea urmatoare, il setam pe B pe 7.

1. **10=>B"100\_011\_111",--RAMQD\_OR\_DZ D=4 Y=4 B=7 RAM(7)=10 Q=10 RAM3=Q3=1**

Aici se realizeaza shiftarea in jos a registrului Q si a rezultatului din ALU, astfel in Q vom avea Q=10 si la adresa data de B din memorie vom avea: RAM(7)=10, deoarece pe Q3 si RAM3 punem valoarea 1. Ca si operatie realizam un OR intre D si Z, adica il punem pe D pe 4 si se va afisa rezultatul 4. Inainte de a apasa butonul pentru a trece la instructiunea urmatoare, il vom seta pe B sa fie 15.

1. **11=>B"100\_001\_000",--RAMD\_SUBR\_AQ A=7(10) Q=4 F=10-10+1 Y=1 CARRY=1 RAM3=1 RAM(B=15)=8**

Aici realizam shiftare la stanga doar pe RAM, il punem pe RAM3 pe 1, deci la adresa data de B, adica 15, vom avea rezultatul din ALU(1) shiftat la dreapta, adica 11. Ca si operatie facem o scadere intre intre A (10) si Q(10), deci se va afisa 1. Inainte de a apasa butonul pentru a trece la instructiunea urmatoare, il vom seta pe B sa fie 14.

1. **12=>B"110\_011\_100",--RAMQU\_OR\_ZA A=10 Y=10 Q=5 B=14 RAM(14)=5**

Aici il punem pe A sa fie 10, astfel se va afisa 10. Vom realiza o shiftare pe registrul Q si pe memoria RAM, dar de data aceasta va fii shiftare la stanga. Punem RAM0 si Q0 pe 1, in Q vom avea 5, iar in RAM, la adresa data de B, adica 14, vom avea rezultatul 5. Inainte de a apasa butonul pentru a trece la instructiunea urmatoare, il vom seta pe B sa fie 15.

1. **13=>B"111\_010\_100",--RAMU\_SUBS\_ZA A=14 A=14(5) Y=12 CARRY=1**

Aici realizam shiftare la dreapta doar pe RAM, il punem pe RAM0 pe 1, deci la adresa data de B, adica 14, vom avea 12. Ca si operatie facem o scadere intre 0 si A(5), deci rezultatul va fii 12.

**7. Concluzii**

Prin urmare, in urma implementarii procesuluri AM2901 pe 4 biti am invatat cum este sa proiectezi un procesor de la 0, fara ajutor, bazat doar pe documentatia de la bibliografia [1]. Pentru realizarea procesorului, in urma studierii in detaliu a documentatiilor, au fost implementate componentele principale ale procesorului, conform arhitecturii pricipale. In urma analizei modului de functionare, am decis sa implementez o componenta principal unde am stabilit toate semnalele necesare si am realizat shiftarile. Conform arhitetcurii, existau mai multe componente separate, precum mai multe multiplexoare, unitati separate de control si shiftari, acestea fiind toate incluse in proiectul principal. Astfel, obiectivul proiectului de a realiza procesorul Am2901 si de arespecta toate functionalitatile a fost indeplinit. Ca si extindere a procesorului, am introdus un semnal de reset pentru resetarea programului, pentru a ajunge din nou la prima instructiunea, si posibilatatea sa verifici mai multe semnale din procesor(Q, A, B) pe baza switch-urilor.

Ca si avantaje, proiectul realizat are capacitatea sa realizeze diverse operatii, precum adunare, scarede, or, and, xor, shiftari, etc., sa retina valori in memoria sa, sa se poata introduca date directe pe placuta si sa poata scoate semnalele din ALU prin switch-uri.

Ca si dezavantaje, procesorul are o memorie mica, calculele nu sunt realizate cu numere prea mari, iar numarul de instructiuni este redus. Astfel, pentru extinderea proiectuli se poate realiza un procesor pe 8, sau chiar 16 biti, se poate extinde memoria RAM si se pot include si alte instructiuni.

**Bibliografie**

[1] Micro Dispozitive avansate “Advanced Micro Devices. The Am2900 Family Data Book WithRelated Support Circuits”*,* 1979, <http://www.decadecounter.com/vta/pdf/1979_AMD_2900family.pdf> pp 9-14.

[2] AMD\_Am2900, Wikipedia, <https://en.wikipedia.org/wiki/AMD_Am2900>

[3] Documentatie am2901a, <http://www.jrok.com/datasheet/am2901a.pdf> pp 1-6.

[4] Documentatie am2901bc, <http://www.jrok.com/datasheet/am2901bc.pdf> pp 4-8.

[5] Indrumator laborator Arhitectura Calculatoarelor

<http://users.utcluj.ro/~onigaf/files/teaching/AC/AC_indrumator_laborator.pdf> pp 11, 14.

**Anexe**

## **Cod pentru ALU**

Mai jos este descrisa o parte din impementarea unitatii aritmetico-logice, primele doua cazuri cu adunare si scadere.

entity ALU is

Port(R, S: in STD\_LOGIC\_VECTOR(3 downto 0);

Operation:in STD\_LOGIC\_VECTOR(2 downto 0);

CIn: in STD\_LOGIC;

COut: out STD\_LOGIC;

OVR: out STD\_LOGIC;

F3: out STD\_LOGIC;

Zero: out STD\_LOGIC;

G: out STD\_LOGIC;

P: out STD\_LOGIC;

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

end ALU;

architecture Behavioral of ALU is

signal R\_aux,S\_aux,F\_aux: STD\_LOGIC\_VECTOR(4 downto 0);

signal P0, P1, P2, P3: STD\_LOGIC;

signal G0, G1, G2, G3: STD\_LOGIC;

signal C4, C3: STD\_LOGIC;

begin

R\_aux <= '0' & R;

S\_aux <= '0' & S;

process(Operation, R\_aux, S\_aux)

begin

case Operation is

when "000" => --add

P0 <= R(0) or S(0);

P1 <= R(1) or S(1);

P2 <= R(2) or S(2);

P3 <= R(3) or S(3);

G0 <= R(0) and S(0);

G1 <= R(1) and S(1);

G2 <= R(2) and S(2);

G3 <= R(3) and S(3);

C4 <= G3 or (P3 and G2) or (P2 and (P1 and G0)) or

((P3 and P2) and (P1 and G0)) or ((P3 and P2) and (P1 and CIn));

C3 <= G2 or (P2 and G1) or (P2 and (P1 and G0))

or ((P2 and P1) and (P0 and CIn));

if CIn = '1' then

F\_aux <= R\_aux + S\_aux + 1;

else

F\_aux <= R\_aux + S\_aux;

end if;

P <= not((P3 and P2) and (P1 and P0));

G <= not(G3 or (P3 and G2) or (P3 and (P2 and G1)) or

((P3 and P2) and (P1 and G0)));

COut <= G3 or (P3 and G2) or (P2 and (P1 and G0)) or

((P3 and P2) and (P1 and G0)) or ((P3 and P2) and (P1 and CIn));

OVR <= (G2 or (P2 and G1) or (P2 and (P1 and G0))

or ((P2 and P1) and (P0 and CIn))) xor (G3 or (P3 and G2) or (P2 and (P1 and G0)) or

((P3 and P2) and (P1 and G0)) or ((P3 and P2) and (P1 and CIn)));

when "001" => --subr

P0 <= not(R(0)) or S(0);

P1 <= not(R(1)) or S(1);

P2 <= not(R(2)) or S(2);

P3 <= not(R(3)) or S(3);

G0 <= not(R(0)) and S(0);

G1 <= not(R(1)) and S(1);

G2 <= not(R(2)) and S(2);

G3 <= not(R(3)) and S(3);

C4 <= G3 or (P3 and G2) or (P2 and (P1 and G0)) or

((P3 and P2) and (P1 and G0)) or ((P3 and P2) and (P1 and CIn));

C3 <= G2 or (P2 and G1) or (P2 and (P1 and G0)) or ((P2 and P1) and (P0 and CIn));

if CIn = '0' then

F\_aux <= S\_aux + not(R\_aux) + 1;

else

F\_aux <= S\_aux + not(R\_aux) + 2;

end if;

P <= not((P3 and P2) and (P1 and P0));

G <= not(G3 or (P3 and G2) or (P3 and (P2 and G1)) or

((P3 and P2) and (P1 and G0)));

COut <= G3 or (P3 and G2) or (P2 and (P1 and G0)) or

((P3 and P2) and (P1 and G0)) or ((P3 and P2) and (P1 and CIn));

OVR <= (G2 or (P2 and G1) or (P2 and (P1 and G0))

or ((P2 and P1) and (P0 and CIn))) xor (G3 or (P3 and G2) or (P2 and (P1 and G0)) or

((P3 and P2) and (P1 and G0)) or ((P3 and P2) and (P1 and CIn)));

[…]

## **Cod pentru memoria RAM**

entity RAM is

Port(A, B, D: in STD\_LOGIC\_VECTOR(3 downto 0);

clk, WE: in STD\_LOGIC;

AOut, BOut:out STD\_LOGIC\_VECTOR(3 downto 0));

end RAM;

architecture Behavioral of RAM is

type type\_mem is array(15 downto 0) of STD\_LOGIC\_VECTOR(3 downto 0);

signal ram: type\_mem := (

0 => "0000",

1 => "0001",

2 => "0010",

3 => "0011",

4 => "0100",

5 => "0101",

6 => "0110",

7 => "0111",

8 => "1000",

9 => "1001",

10 => "1010",

11 => "1011",

12 => "1100",

13 => "1101",

14 => "1110",

15 => "1111",

others => "0000");

begin

process(clk)

begin

if rising\_edge(clk) then

if WE = '1' then

ram(conv\_integer(B)) <= D;

end if;

end if;

end process;

AOut <= ram(conv\_integer(A));

BOut <= ram(conv\_integer(B));

end Behavioral;

## **Cod pentru program MicroInstructiuni**

entity MicroInstructions is

Port( clk, buton, rst:in STD\_LOGIC;

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

Instr: out STD\_LOGIC\_VECTOR(8 downto 0));

end MicroInstructions;

architecture Behavioral of MicroInstructions is

type program\_type is array(0 to 255) of STD\_LOGIC\_VECTOR(8 downto 0);

signal program: program\_type:=(

0=>B"001\_000\_001",--NOP\_ADD\_AB A=2, B=8, Y=10

1=>B"000\_001\_011", --QREG\_SUBR\_ZB B=5, F=5, Q=5, Y=5

2=>B"011\_100\_100", --RAMF\_AND\_ZA A=4 F=0 B=3 RAM(3)=0 Y=0

3=>B"011\_011\_010", --RAMF\_OR\_ZQ Q=5 F=5 B=10 RAM(10)=5 A=3 Y=5

4=>B"100\_010\_101", --RAMQD\_SUBS\_DA D=9 A=5 F=4 Y=4 B=4, RAM(4)=5/2=2 Q=Q/2=2

5=>B"101\_000\_101", --RAMD\_ADD\_DA D=6 A=15 F=5 Y=5 B=6 RAM(6)=5/2=2

6=>B"001\_101\_110",--NOP\_NOTRS\_DQ d=8 Q=2 Y=2

7=>B"110\_110\_111", --RAMQU\_EXOR\_DZ D=2 F=2 Y=2 B=15 RAM(15)=2\*2=4 Q=2\*2=4

8=>B"111\_111\_000", --RAMU\_EXNOR\_AQ A=15 Q=4 F=4 Y=4 B=7 RAM(7)=4\*2=8

9=>B"010\_000\_110",--RAMA\_ADD\_DQ Q=4 D=10 Q=4 CARRY=1 Y=15 B=6 RAM(6)=15

10=>B"100\_011\_111",--RAMQD\_OR\_DZ D=4 Y=4 RAM3=Q3=1 B=7 RAM(7)=10 Q=10

11=>B"101\_001\_000",--RAMD\_SUBR\_AQ A=7(10) Q=10 CARRY=1 Y=1 B=15 RAM(15)=8

12=>B"110\_011\_100",--RAMQU\_\_OR\_ZA A=10 Y=10 B=14 RAM(14)=5 Q=5

13=>B"111\_010\_100",--RAMU\_SUBS\_ZA A=14(5) Y=12 B=15 RAM(15)=9

others=>"000000000");

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

begin

process(buton, clk)

begin

if rising\_edge(clk) then

if buton='1' then

count<=count+1;

end if;

if rst = '1' then

count<= x"0000";

end if;

end if;

end process;

Instr<=program(conv\_integer(count));

cnt<=count;

end Behavioral;

## **Cod pentru AMD**

entity AMD is

Port(Clk, Rst: in STD\_LOGIC;

Btn, BtnUp: in STD\_LOGIC;

--A, B, D:in STD\_LOGIC\_VECTOR(3 downto 0);

sw : in STD\_LOGIC\_VECTOR (15 downto 0);

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

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

an, cat : out STD\_LOGIC\_VECTOR (7 downto 0)

--CIn, RAM0, RAM3,Q0, Q3 : in STD\_LOGIC;

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

);

end AMD;

architecture Behavioral of AMD is

signal Instr: STD\_LOGIC\_VECTOR(8 downto 0);

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

signal Func,Op, Dest: STD\_LOGIC\_VECTOR(2 downto 0);

signal A, B, D: STD\_LOGIC\_VECTOR(3 downto 0):="0000";

signal Q, R, S, F, ShiftQ, ShiftF, AOut, BOut, ALatch, BLatch, Y: STD\_LOGIC\_VECTOR(3 downto 0):="0000";

signal Z: STD\_LOGIC\_VECTOR(3 downto 0):="0000";

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

signal dig: STD\_LOGIC\_VECTOR(31 downto 0);

signal ShiftRAMDown,ShiftRAMUp, ShiftQDown,ShiftQUp, QEn, EnRAM, AorF, butonMPGOut: STD\_LOGIC:='0';

signal COut, OVR, F3, Zero, G, P, rstInstr: STD\_LOGIC:='0';

signal RAM0, RAM3, Q0, Q3, CIn:STD\_LOGIC:='0';

begin

A <= sw(3 downto 0);

B <= sw(7 downto 4);

D <= sw(11 downto 8);

CIn <=sw(12);

RAM0<=sw(12);

RAM3<=sw(13);

Q0<=sw(14);

Q3<=sw(15);

Op <= Instr(2 downto 0);

Func <= Instr(5 downto 3);

Dest <= Instr(8 downto 6);

Y <= ALatch when AorF = '1' else F;

source\_operands:process(Op, ALatch, BLatch, Q, D, Instr,Z)

begin

case Op is

when "000" => R <= ALatch; S <= Q;

when "001" => R <= ALatch; S <= BLatch;

when "010" => R <= Z; S <= Q;

when "011" => R <= Z; S <= BLatch;

when "100" => R <= Z; S <= ALatch;

when "101" => R <= D; S <= ALatch;

when "110" => R <= D; S <= Q;

when "111" => R <= D; S <= Z;

when others => R<="0000"; S<= "0000";

end case;

end process;

alu\_destination:process(Dest, Instr)

begin

case Dest is

--QREG

when "000" => ShiftRAMDown<= '0'; ShiftRAMUp <= '0';ShiftQDown <= '0';ShiftQUp <= '0';

QEn <= '1'; AorF <= '0'; EnRAM <='0'; --RAM0 <= '0'; RAM3 <= '0'; Q0 <= '0'; Q3 <= '0';

--NOP

when "001" => ShiftRAMDown<= '0'; ShiftRAMUp <= '0';ShiftQDown <= '0';ShiftQUp <= '0';

QEn <= '0'; AorF <= '0'; EnRAM <='0'; --RAM0 <= '0'; RAM3 <= '0'; Q0 <= '0'; Q3 <= '0';

--RAMA

when "010" => ShiftRAMDown<= '0'; ShiftRAMUp <= '0';ShiftQDown <= '0';ShiftQUp <= '0';

QEn <= '0'; AorF <= '1'; EnRAM <='1';-- RAM0 <= '0'; RAM3 <= '0'; Q0 <= '0'; Q3 <= '0';

--RAMB

when "011" => ShiftRAMDown<= '0'; ShiftRAMUp <= '0';ShiftQDown <= '0';ShiftQUp <= '0';

QEn <= '0'; AorF <= '0'; EnRAM <='1'; --RAM0 <= '0'; RAM3 <= '0'; Q0 <= '0'; Q3 <= '0';

--RAMQD

when "100" => ShiftRAMDown<= '1'; ShiftRAMUp <= '0';ShiftQDown <= '1';ShiftQUp <= '0';

QEn <= '1'; AorF <= '0'; EnRAM <='1'; --RAM0 <= F(0); RAM3 <= '0'; Q0 <= Q(0); Q3 <= '0';

--RAMD

when "101" => ShiftRAMDown<= '1'; ShiftRAMUp <= '0';ShiftQDown <= '0';ShiftQUp <= '0';

QEn <= '0'; AorF <= '0'; EnRAM <='1'; --RAM0 <= F(0); RAM3 <= '0'; Q0 <= Q(0); Q3 <= '0';

--RAMQU

when "110" => ShiftRAMDown<= '0'; ShiftRAMUp <= '1';ShiftQDown <= '0';ShiftQUp <= '1';

QEn <= '1'; AorF <= '0'; EnRAM <='1'; --RAM0 <= '0'; RAM3 <= F(3); Q0 <= '0'; Q3 <= Q(3);

--RAMU

when "111" => ShiftRAMDown<= '0'; ShiftRAMUp <= '1';ShiftQDown <= '0';ShiftQUp <= '0';

QEn <= '0'; AorF <= '0'; EnRAM <='1'; --RAM0 <= '0'; RAM3 <= F(3); Q0 <= '0'; Q3 <= Q(3);

when others =>

end case;

end process;

Shift\_Q: process(ShiftQUp, ShiftQDown, QEn, Instr, clk, Q3, Q0, Q, F)

begin

if ShiftQDown = '1' and ShiftQUp = '0' then

ShiftQ <= Q3 & Q(3 downto 1);

end if;

if ShiftQUp = '1' and ShiftQDown = '0' then

ShiftQ <= Q(2 downto 0) & Q0;

end if;

if ShiftQUp = '0' and ShiftQDown = '0' then

ShiftQ <= F;

end if;

end process;

Shift\_F:process(ShiftRAMDown,ShiftRAMUp, EnRAM, Instr, clk)

begin

if ShiftRAMDown = '1' and ShiftRAMUp = '0' then

ShiftF <= RAM3 & F(3 downto 1);

end if;

if ShiftRAMDown = '0' and ShiftRAMUp = '1' then

ShiftF <= F(2 downto 0) & RAM0;

end if;

if ShiftRAMDown = '0' and ShiftRAMUp = '0' then

ShiftF <= F;

end if;

end process;

led(0)<=COut;

led(1)<=OVR;

led(2)<=F3;

led(3)<=Zero;

led(4)<=G;

led(5)<=P;

ALU\_component: entity WORK.ALU port map (

R => R,

S => S,

Operation => Func,

CIn => CIn,

COut => COut,

OVR => OVR,

F3 => F3,

Zero => Zero,

G => G,

P => P,

F => F

);

RAM\_component: entity WORK.RAM port map (

A => A,

B => B,

D => ShiftF,

clk => Clk,

WE => EnRAM,

AOut => AOut,

BOut => BOut

);

QRegister\_component: entity WORK.QRegister port map (

D => ShiftQ,

clk => Clk,

QEn => QEn,

Q => Q

);

LatchA\_component: entity WORK.Latch port map (

clk => Clk,

d => AOut,

q => ALatch

);

LatchB\_component: entity WORK.Latch port map (

clk => Clk,

d => BOut,

q => BLatch

);

MicroInstructions\_component: entity WORK.MicroInstructions port map (

clk => Clk,

buton => butonMPGOut,

--buton => btn(0),

rst => rstInstr,

--rst => btn(1),

cnt => cnt,

Instr => Instr

);

MPG\_component: entity WORK.MPG port map (

en => butonMPGOut,

input => Btn,

clk => Clk

);

MPG\_component2: entity WORK.MPG port map (

en => rstInstr,

input => BtnUp,

clk => Clk

);

dig<="0000000000000000000000000000"&Y;

--SSD\_component: entity WORK.SSD port map (

-- clk => clk,

-- digit => dig,

-- an => an,

-- cat => cat

--);

display\_component: entity WORK.displ7seg port map(

Clk => Clk,

Rst => Rst,

Data => dig,--: in STD\_LOGIC\_VECTOR (31 downto 0); -- datele pentru 8 cifre (cifra 1 din stanga: biti 31..28)

An => an, --: out STD\_LOGIC\_VECTOR (7 downto 0); -- selectia anodului activ

Seg => cat --: out STD\_LOGIC\_VECTOR (7 downto 0));

);

end Behavioral;