TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nikita Budovey 206252IACB

Süstoolsel maatriksil põhineva närvivõrgu kiirendi realisatsioon FPGAs

Bakalaureusetöö

Juhendaja: Jaan Raik
PhD Täisprofessor tenuuris

Kaasjuhendaja: Mahdi Taheri
MSc Doktorant-nooremteadur

Tallinn 2023

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Nikita Budovey

11.04.2023

Annotatsioon

Antud töö raames kujundati ja töötati välja süstoolsel maatriksil põhineva närvivõrgu kiirendi arvutuslik osa. Süstoolse massiivi projekt koosneb 7 moodulist, millest 3 on mäluelemendid ja 4 ülejäänut arvutuslikud elemendid. Projektis saab hõlpsasti muuta selliseid parameetreid nagu maatriksi mõõtmed ja andmesalvestusregistrite suurus, mis muudab selle piisavalt paindlikuks, et seda teistes projektides rakendada. Samuti võib süstoolne maatriks töötada negatiivsete arvudega, mis laiendab selle funktsionaalsust.

Lisaks käsitletakse töö raames süstoolse maatriksi testimist Basys 3 plaadil, mille jaoks loodi veel üks moodul, mis ühendab Basys 3 plaadi perifeeria projektiga. Testimisel otsustati kasutada 7-segmendilist kuva, mis kuvab ühes kolmest maatriksist salvestatud andmed.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 5 peatükki, 26 joonist.

Abstract

Realization of 2D systolic array-based neural network accelerator in FPGA

In this thesis, the computational part of the systolic matrix neural network accelerator was designed and developed. The systolic array project consists of 7 modules. of which 3 are memory elements and the remaining 4 are computational elements. In the project, parameters such as the size of the matrix and the data storage registers can be easily changed. It makes the project is flexible enough to be applied to other projects. Also, the systolic matrix can work with negative numbers, which increases its functionality.

The work also deals with the testing of the systolic matrix on the Basys 3 board, for which another module was created that connects the Basys 3 board to the peripheral project. During testing, it was decided to use a 7-segment display, which displays data stored in one of the three matrices.

The thesis is written in Estonian and contains 29 pages of text, 5 chapters, 26 figures.

Lühendite ja mõistete sõnastik

ASIC	*Application-Specific Integrated Circuit*, integraallülitus, mis on spetsiaalselt projekteeritud ja toodetud konkreetse ülesande või rakenduse jaoks elektroonikas.
CNN	*Convolutional Neural Network*, närvivõrgu tüüp, mida kasutatakse piltide töötlemiseks ja klassifitseerimiseks.
CPU	*Central Processing Unit*, arvuti põhiseade, mis teostab aritmeetilisi, loogilisi ja juhtimistoiminguid.
DNN	*Deep Neural Networks*, kunstliku närvivõrgu tüüp, mis koosneb mitmest neuronikihist.
DPI	*Dots per inch*, punkti tolli kohta
DSP plokk	*Digital Signal Processing* plokk, programmeeritav riistvaraplokk, mida kasutatakse digitaalsete signaalide töötlemiseks.
FC kihid	*Fully Connected* kihid, kihid närvivõrgus, milles iga neuron on ühendatud iga neuroniga eelmises ja järgmises kihis.
FPGA	*Field-Programmable Gate Array*
GPU	*Graphics Processing Unit*, graafika töötlemisele spetsialiseerunud protsessori tüüp
IA	Arvutisüsteemide instituut

LED	*Light-Emitting Diode*, pooljuhtelement, mis toodab valgust, kui seda läbib elektrivool.
MAC	*Multiply-and-Accumulate*, korrutamis- ja akumuleerimisoperatsioon, mida teostab maatrikskiirend (või protsessor) süstoolse maatriksi arhitektuuris.
PE	*Processing Element*, süstoolse maatriksi põhielement ja seda kasutatakse andmetega toimingute tegemiseks.
ReLu	*Rectified Linear Unit*, aktiveerimisfunktsioon, mida kasutatakse närvivõrkudes

Sisukord

1 Sissejuhatus 9
1.1 Taust 9
1.2 Eesmärgid 9
2 Teema ülevaade 10
2.1 CNN 10
2.2 Täielikult ühendatud kihid 11
2.2.1 Kaal 12
2.3 Närvivõrgu kiirendi 13
2.4 Süstoolne maatriks 13
2.4.1 Näide maatrikskorrutamisest süstoolse maatriksi abil 14
3 Projekteerimine 16
3.1 Adder moodul 18
3.1.1 carry_look_ahead_4bit moodul 19
3.2 Mult8s 20
3.3 Accumulator 22
3.4 PE 23
3.5 Systolic_Array 24
3.6 wheights_register ja inputs_register 28
4 Testimine 29
5 Kokkuvõte 38
Kasutatud kirjandus 39
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 42
Lisa 2 – SystemVerilogis kirjutatud mooduli Static_Memory kood 43

Jooniste loetelu

Joonis 1 CNN-i töö visualisatsioon [10] 11
Joonis 2 FC kihid [8] 12
Joonis 3 2x2 maatrikskorrutis 14
Joonis 4 Maatrikskorrutamise algoritm [6] 15
Joonis 5 Maatriksi korrutamine süstoolse maatriksi abil 16
Joonis 6 Andmevoog läbi moodulite 17
Joonis 7 Kood väärtuste lisamiseks moodulis “Adder” 19
Joonis 8 4-bitine kiire ülekandega summaatori arhitektuur [3] 20
Joonis 9 Ülekandebiti genereerimine 20
Joonis 10 8x8 osalise tootemaatriksi 4-kihiline Wallace'i redutseerimine [12] 21
Joonis 11 32-bitise registri jaoks ReLu funktsiooni eest vastutav plokkskeem [7] 22
Joonis 12 “Accumulator” mooduli diagramm 23
Joonis 13 “PE” ploki visuaalne diagramm 24
Joonis 14 Generate plokk “Systolic_Array” mooduli jaoks 26
Joonis 15 “Systolic_Array” mooduli diagramm 27
Joonis 16 Kood andmete mälupuhvrist mahalaadimiseks 28
Joonis 17 Sisendmaatriksite initsialiseerimine moodulis “Satic_Memory” 29
Joonis 18 Maatriksi korrutamise tulemus “Satic_Memory” moodulis 30
Joonis 19 Maatriksi andmetega *result_marix* ülekandmiseks moodulist “Systolic_Array” moodulisse “Static_Memory” 31
Joonis 20 Väärtuse *display_matix* muutumine sõltuvalt lülitist väärtusest 32
Joonis 21 Ühine anoodiahela sõlm [2] 33
Joonis 22 Basys 3 arendusplaat [2] 34
Joonis 23 Plaat pärast lähtestussignaali 35
Joonis 24 Maatriksi a ja maatriksi b väärtus 36
Joonis 25 Maatrikskorrutamise tulemus 37
Joonis 26 Static_Memory mooduli kood 46

1 Sissejuhatus

1.1 Taust

Juhataja professor Jaan Raik ja tema uurimisrühm “Usaldusväärsete arvutisüsteemide keskus” uurivad närvivõrgu riistvara robustsust. Keskkonnas on palju tegureid, mis võivad algoritmide tulemuste täpsust oluliselt mõjutada. Uurimisrühm soovib kasutada pakutavaid andmeid, et luua süsteem, mis on vastupidavam ümbritsevate häirete ja müra suhtes. Probleemi üksikasjalikuks uurimiseks vajavad nad täiesti avatud ja konfigureeritavat riistvaralahendust. Enamikus riistvaras, mis on suunatud kujutiste töötlemisele ja piltidelt teabe hankimisele, põhinevad need maatrikskorrutamisel, mida realiseeritakse süstoolsetel maatriksitel.

1.2 Eesmärgid

Lõputöö eesmärgiks on koostada pakendatud Xilinx Vivado projekt, mis on realiseeritud kasutades SystemVerilog riistvara kirjelduskeelt. Projekt on kirjutatud nullist ja kasutab 7 moodulit: Systolic_Array, weights_register, inputs_register, PE, Mult8s, Accumulator, Adder. Selle projekti peamised omadused on modulaarsus ja paindlikkus. Modulaarsus – tähendab, et seda projekti on lihtne teistes projektides rakendada. Paindlikkus – et mõningaid süsteemi parameetreid saab hõlpsasti muuta vastavalt oma eelistustele.

Käesoleva töö teiseks eesmärgiks on testida antud projekti FPGA peal kasutades Basys 3 arendusplaati. Eesmärgi saavutamiseks lisati eraldi “Static_Memory” moodul, mis on põhiprojekti sisse ehitatud ning võtab vastu väljastpoolt tulevaid signaale ning salvestab korrutamiseks ettevalmistatud maatriksi.

2 Teema ülevaade

Selles peatükis käsitletakse lühidalt närvivõrgu kiirendi teemasid ja selgitatakse, milleks see vajalik on ning kuidas toimib süstoolne maatriks ja miks seda kiirendites kasutatakse.

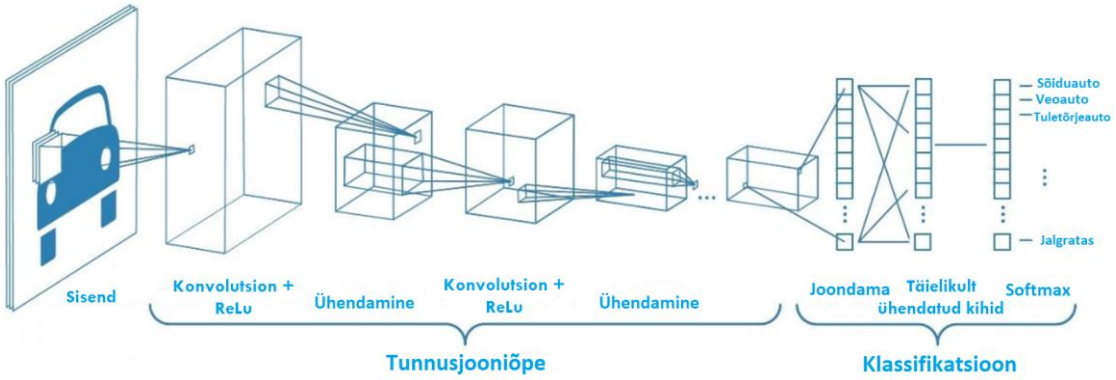
2.1 CNN

Konvolutsioonilisi närvivõrke (ingl. k. Convolutional Neural Network e. CNN) kasutatakse arvutinägemise ülesannetes, nagu mustrituvastus, kujutiste klassifitseerimine ja segmenteerimine. Need töötavad kahe põhioperatsiooni alusel: konvolutsioon ja ühendamine (ingl. k. pooling) [10], [18], [20].

Konvolutsioon on toiming, mis käivitab filtri (konvolutsiooni tuumad) kogu pildil piki mõlemat telge, kogudes teavet pildifragmentide kohta [16]. Filter on kaalude maatriks, mis liigub läbi pildi, koondades teavet iga selle fragmendiga. Konvolutsiooni tulemuseks on uus maatriks, mida nimetatakse tunnuskaardiks.

Ühendamine on teabe tihendamise toiming, mida rakendatakse objektikaardile. Ühendamiseprotsessi ajal vähendatakse pildiala ühe väärtuseni, valides piirkonnast suurima väärtuse. Näiteks maksimaalse ühendamise meetodi kasutamisel valitakse pildi igast fragmendist suurim väärtus, mis salvestatakse uude maatriksisse.

Seega kasutab CNN kujutiste funktsioonide eraldamiseks konvolutsiooni- ja ühendamiseoperatsioone ning seejärel läbib klassifitseerimiseks mitu kihti. Seda protseduuri korratakse mitu korda, et saada närvivõrgu poolt arvutatav tulemus. Joonis 1 näitab seda protsessi visualiseerituna.



Joonis 1 CNN-i töö visualisatsioon [10]

2.2 Täielikult ühendatud kihid

Pärast andmete töötlemist konvolutsiooniliste kihtide abil edastatakse need täielikult ühendatud kihtidele [19].

Täielikult ühendatud kihid (ingl. k. Fully Connected e. FC kihid) on tänapäevaste närvivõrkude peamine lüli. Nende peamine eesmärk on sisendandmete töötlemine ja väljundväärtuste genereerimine [8].

FC kihid koosnevad neuronite kihtidest, kus iga kihis olev neuron saab teavet kõikidelt eelmise sisendkihi neuronitelt. Seejärel teostab see matemaatilise tehte, sealhulgas korrutab sobivate kaaludega ja lisab kalde (ingl. k. bias). Iga toimingu tulemus kantakse üle järgmisele kihile. Seda korratakse, kuni võrgu lõplik väljund on vastu võetud.

FC kihte saab kasutada mustrite tuvastamiseks, andmete klassifitseerimiseks, rühmitamiseks ja tulevikuväärtuste ennustamiseks. Erinevalt konvolutsioonikihtidest, mis töötavad struktureeritud funktsioonidega, on FC kihte lihtsam kasutada ja need nõuavad vähem arvutusressursse. Joonisel 2 on kujutatud täielikult ühendatud kihtidega FC võrk.

FC-kihtide puuduseks on asjaolu, et need võivad olla arvutuslikult kallid ja nõuavad palju mälu. Standardsetel FC-kihtidel võib olla probleeme ka suhtelise täpsusega, eriti kui tegemist on väikese andmemahuga. Neid puudusi saab aga kõrvaldada, optimeerides FC-kihtide arhitektuuri ja kasutades regulaarsust.

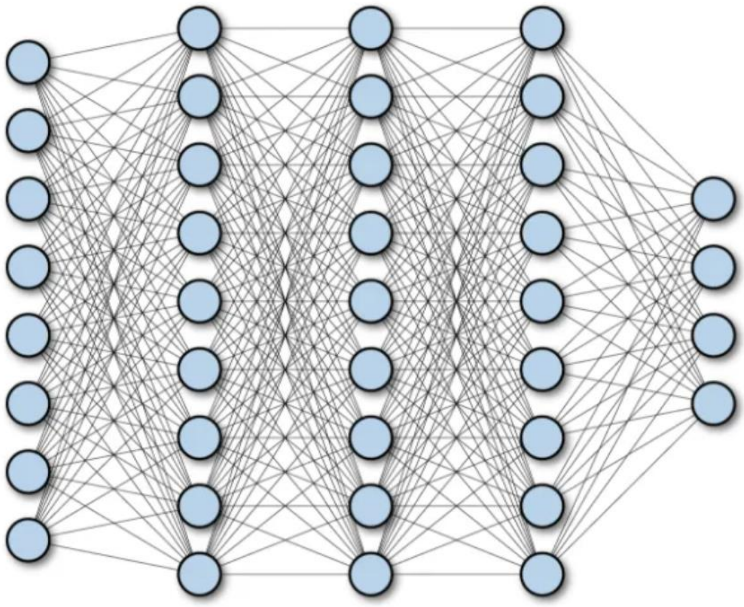
Softmax on aktiveerimisfunktsioon, mida kasutatakse tavaliselt närvivõrkude viimases kihis mitme klassi ülesannete klassifitseerimiseks. See võtab väärtuste vektori ja "tihendab" selle uueks tõenäosusvektoriks, kus iga väärtus näitab tõenäosust, et sisendväärtus kuulub igasse klassi [15].

2.2.1 Kaal

Närvivõrkudes on kaal (ingl. k. weight) parameeter, mis määrab iga sisendmuutuja mõju väljundtulemusele. Närvivõrgu treenimisel valitakse kaalud selliselt, et oleks saavutatud väljundandmetes vajalik täpsus [16].

Igal neuronil on oma kaaluvektor. See kaaluvektor määrab, kuidas neuron sisendile reageerib. Mida suurem on kaal, seda olulisem on vastav sisendmuutuja.

Suur kaalude edastamine võib olla seotud närvivõrgu ülepaigutamise probleemidega, seega on see üks peamisi parameetreid, mida tõhusa närvivõrgu kavandamisel arvesse võtta.



Joonis 2 FC kihid [8]

2.3 Närvivõrgu kiirendi

Digitehnoloogia viimaste edusammude ja suurandmete kättesaadavuse tõttu on tekkinud tehisintellekti valdkond, sügav õpe, mis on näidanud oma võimet ja tõhusust keeruliste õpiprobleemide lahendamisel, mis varem polnud võimalikud. Eelkõige on CNN näidanud oma tõhusust kujutisetuvastuse rakendustes. Kuid need nõuavad intensiivseid protsessori toiminguid ja mälu ribalaiust, mistõttu ei suuda tavalised mikropürotsessorid e CPU-d soovitud jõudlust saavutada. Seetõttu on CNN-ide läbilaskevõime suurendamiseks kasutatud riistvarakiirendeid, mis kasutavad ASIC, FPGA-sid ja graafikaprotsessoreid (GPU-sid). Täpsemalt on hiljuti kasutusele võetud FPGA-d, et kiirendada süvaõppevõrkude rakendamist, kuna need suudavad maksimeerida paralleelsust ja nende energiatõhusust [1], [14].

Üks peamisi põhjuseid FPGA-de kasutamiseks närvivõrgu kiirendites on selle paindlikkus [17]. FPGA võimaldab teil kiiresti riistvara ümber konfigureerida, et täita närvivõrkude spetsiifilisi funktsioone. See võimaldab arendajatel kiiresti luua ja optimeerida seadmeid, mis sobivad konkreetsete ülesannete jaoks.

Käesolevas artiklis käsitletava kiirendi põhieesmärk on lihtsustada konvolutsiooni maatrikskorrutamise protsessi, mis omakorda lihtsustab oluliselt kiirendi arhitektuuri.

2.4 Süstoolne maatriks

Süstoolne maatriks on riistvaraarhitektuur, mida rakendatakse nii närvivõrkude töö käigus kui ka treenimise kiirendamiseks. See kasutab sisend- ja väljundkihtide vaheliste tulemuste arvutamiseks maatrikskorrutamist [1], [6].

Närvivõrgu kiirendi kasutab paljusid paralleelseid arvutusseadmeid, mis töötavad maatriksi korrutamise teostamiseks samaaegselt. Need plokid on paigutatud ruudustikku e maatriksi massiivi. Neid plokke nimetatakse PE-plokkideks, mis teostavad arvutustoimingut – MAC (ingl. k. Multiply-and-Accumulate e. korrutada-akumuleerida), mis võimaldab kiirendada andmetöötlust, kasutades paralleelsust ja arvutustulemuste lokaalset salvestamist. Iga massiivi element korrutab sisendi oma teguriga ja edastab korrutise endast paremal asuvale elemendile, mis omakorda liidab selle enda väljundsummale. Selle tulemusel kogutakse väljundandmeid korduvalt massiivi järgmistesse elementidesse, mis tagab maatriksoperatsioonide suure kiiruse. Näiteks kui

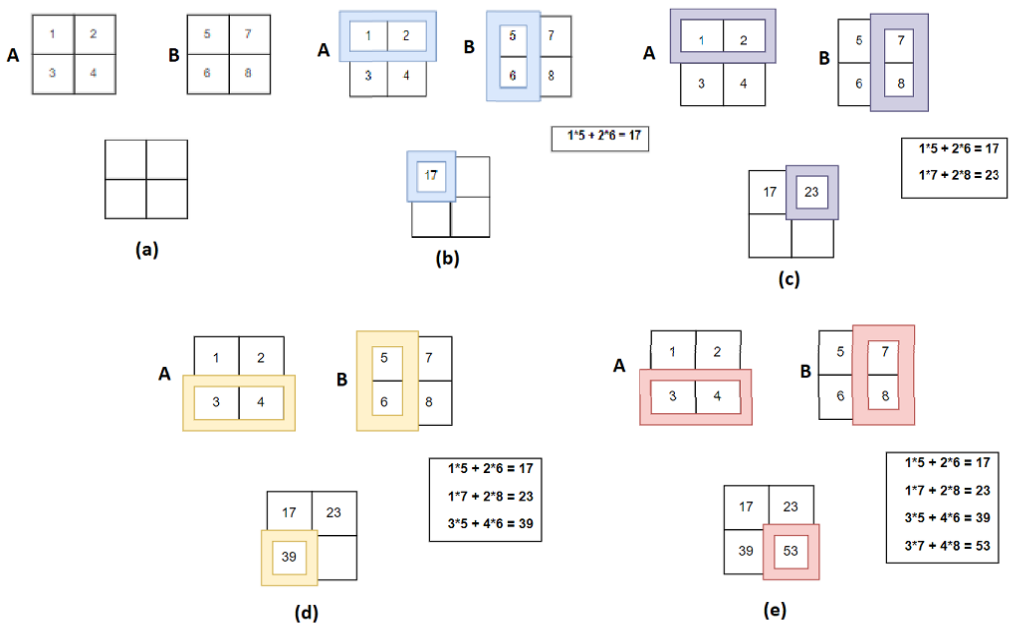
sisendandmed on 10x10 maatriksis, siis on ploki kohta 1 sisendandmete element. Kõik ruudustiku plokid töötlevad sisendit paralleelselt ja toodavad iga sisendelemendi jaoksväljundväärtusi.

Juhtimissüsteem koordineerib kõigi ruudustiku plokkide tööd, jaotades nende vahel ülesanded. Seega väheneb oluliselt ühe sisendandmete komplekti töötlemise aeg, mis võimaldab omakorda kiirendada närvivõrgu kiirendi tööd.

2.4.1 Näide maatrikskorrutamisest süstoolse maatriksi abil

Esiteks vaatame, kuidas me tavaliselt maatrikseid korrutame.

Joonisel 3 (a) näitab kahte 2x2 maatriksit A ja B. Joonis 3 (b) näitab maatriksi A esimese rea korrutamist maatriksi B esimese veeruga. Järgmisena näidatakse maatriksi A esimese rea korrutamist, kuid alles nüüd maatriksi B teise veeruga Joonisel 3 (c). Sama juhtub maatriksi A teise reaga Joonisel 3 (d) ja Joonisel (e).



Joonis 3 2x2 maatrikskorrutis

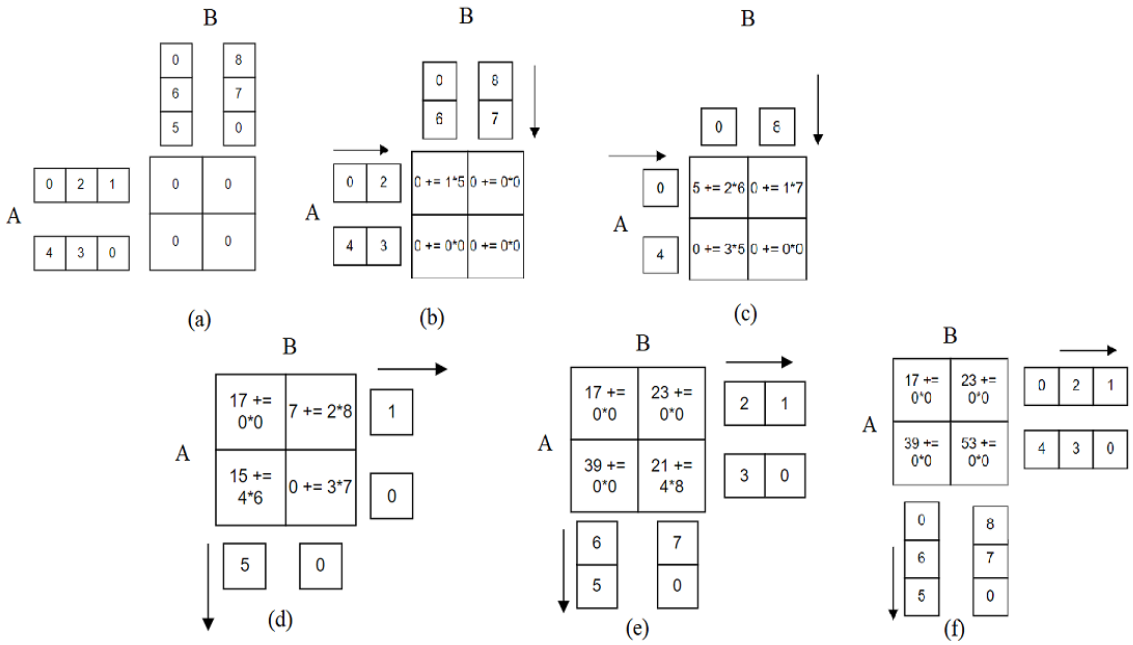
Joonis 4 näitab maatrikskorrutamise algoritmi. Kus A ja B on algmaatriksid, C on nende korrutamise tulemus, I on ridade arv, J on veergude arv, K on osuti antud veerus või reas olevale elemendile.
Selle valemi põhjal võime järeldada, et selle korrutamise jaoks kulub N^3 iteratsiooni. Kui meil on 2x2 maatriks (st N=3), siis on tulemuse saamiseks vaja 8 iteratsiooni.

```
For I = 1 to N
   For J = 1 to N
      For K = 1 to N
         C[I,J] = C[I,J] + A[J,K] * B[K,J];
      End
   End
End
```

Joonis 4 Maatrikskorrutamise algoritm [6]

Vaatleme, kuidas sama protsessi rakendataks kahemõõtmelise süstoolse maatriksi abil. Joonis 5 näitab samm-sammult kahe maatriksi korrutamist süstoolses maatriksis. Joonis 5 (a) Näidatud on kaks algmaatriksit A ja B. Maatriks A on kujutatud 2 vektorina, kus viimane vektor on nihutatud vasakule. Maatriksit B esitatakse samamoodi nagu maatriksit A, kuid jagatakse vektoriteks pigem veergude kui ridade kaupa ning viimane vektor nihutatakse vasaku suuna asemel ülespoole. 2x2 süstolüütiline massiiv ise on esitatud keskel. Joonisel 5 (b), (c), (d), (e) ja (f) maatriks A liigub vasakult paremale ja maatriks B liigub ülalt alla. Mõlemad maatriksid liiguvad süstoolse maatriksi suunas. Vastuvõetud andmed korrutatakse omavahel ja akumuleeritakse sisemällu. Nagu on näha Jooniselt 5 (d), (e), (f), jätkavad andmed pärast massiivi läbimist oma voogu samas suunas.

Maatriksi korrutamine süstoolse massiivi abil võtab 4 iteratsiooni. Nagu näeme, on iteratsioonide arv kaks korda väiksem kui tarkvaraliselstandardmeetodil.



Joonis 5 Maatriksi korrutamine süstoolse maatriksi abil

3 Projekteerimine

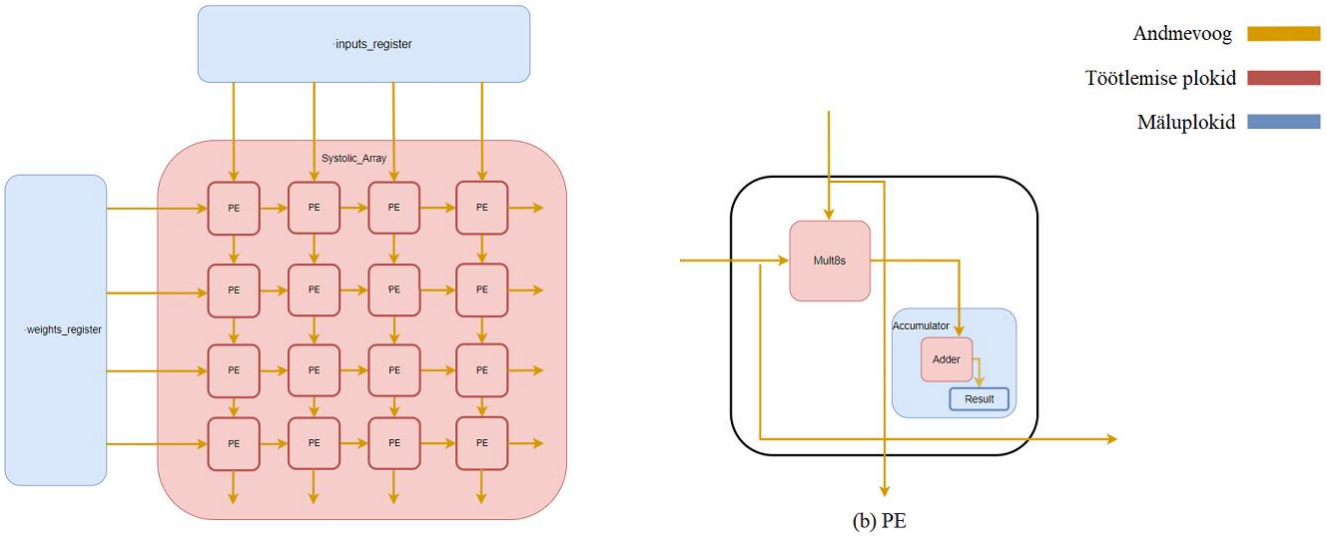
See peatükk käsitleb projekti loomise teemat riistvarakirjelduskeeles SystemVerilog. Projekt koosneb 7 moodulist:

- Systolic_Array
- weights_register
- inputs_register
- PE
- Mult8s
- Accumulator
- Adder

Kogu projektil on 2 parameetrit. Esimene parameeter MATRIX_SIZE vastutab maatriksi suuruse eest. Teine parameeter DATA_WIDTH vastutab sisendandmete registri pikkuse e andmete bitilaiuse.

Joonisel 6 näitab, kuidas sisend meie süsteemi iga mooduliga suhtleb. Joonisel 6 (a) näitab, kuidas andmed mälupuhvritest (sisendiregister ja kaaluregister) teisaldatakse arvutusüksusesse endasse. Andmeid teisaldatakse välismälust mälupuhvritesse, kuni need on täis. Mõlemad mälupuhvrid annavad märku andmete saatmise valmisolekust. Pärast seda, kui "Systolic_Array" moodul võtab vastu mõlemad signaalid, et mälupuhvrid on valmis, hakkab see paralleelselt saatma andmeid välistele "PE"-moodulitele. Joonisel 6 (b) näete, kuidas andmed "PE"-moodulis liiguvad. Andmed tulevad "Systolic_Array" moodulist ja lähevad kohe "Mult8s" moodulisse. "Mult8s" moodulist lähevad andmed "Akumulaator" moodulisse, milles lisatakse andmed mooduli "Adder" abil Tulemuste registrisse.

Korrutamis- ja liitmisoperaatori jaoks otsustati teha eraldi moodulid, mitte kasutada DSP-plokke, mis on väga ressursimahukad ja võivad maatrikskorrutamise protsessi oluliselt aeglustada.



(a) Süstoolne massiiv

(b) PE

Joonis 6 Andmevoog läbi moodulite

3.1 Adder moodul

See moodul vastutab kahe sisendväärtuse ühendamise eest. Liitja põhines kiire ülekandega summatoril (ingl. K. Carry Look-Ahead Adder) [3], kuna see on teist tüüpi liitjatega võrreldes kiire. See liitja vajab liitmistoimingu lõpuleviimiseks ainult ühte tsüklit, olenemata sisendregistrite pikkusest.

Moodulis on veel üks alammoodul “carry_look_ahead_4bit”. See täidab 4-bitise liitja funktsiooni.

Moodul aktsepteerib registri *a* ja *b* väärtusi suurusega DATA_WIDTH ja *cin* signaali. Moodul tagastab sum registri suuruse DATA_WIDTH ja *cout* signaaliga.

Moodulis “Adder” kasutame *generate* plokki, mis aheldab olenevalt DATA_WIDTH/4 suurusest kokku “carry_look_aheade_4bit” ja ühendab need registri *c* abil, kus *c[i]* täidab *cin* rolli ja *c[i +1]* mängib *cout* rolli nagu näidatud Joonisel 7.

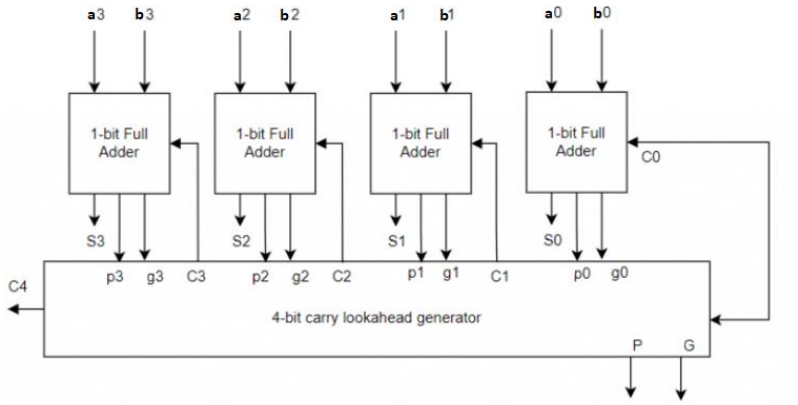
Näiteks DATA_WIDTH suurus on 8 bitti, siis on meil kaks “carry_look_ahead_4bit” moodulit, mis ühendatakse registriga. *c[0]* ja *c[1]* on esimese mooduli jaoks *cin* ja *cout* ning *c[1]* ja *c[2]* on teise mooduli jaoks *cin* ja *cout*.

```
wire     c [NUMBER_SIZE/4-1:0];

assign    c[0] = cin;

genvar i;

generate

    for (i = 0; i < NUMBER_SIZE/4; i++) begin

        carry_look_ahead_4bit cla (

        .a(a[4*(i+1)-1:4*i]),

        .b(b[4*(i+1)-1:4*i]),

        .cin(c[i]),

        .sum(sum[4*(i+1)-1:4*i]),

        .cout(c[i+1]));

    end

endgenerate
```

Joonis 7 Kood väärtuste lisamiseks moodulis “Adder”

3.1.1 carry_look_ahead_4bit moodul

Kiire ülekandega summaator toimib kahe biti genereerimisega, mida nimetatakse ulekande levitamine (ingl. k. Carry Propagate) ja ülekande genereerimine (ingl. k. Carry Generate) ning mida tähistavad vastavalt *p* ja *g*. Levitatakse *p*-bitt järgmisse etappi ja *g*-bitti kasutatakse väljundi ülekandebiti genereerimiseks ja see ei sõltu sisendi ülekandebitist. Joonis 8 näitab kiire ülekandega summaatori arhitektuuri.



Joonis 8 4-bitine kiire ülekandega summaatori arhitektuur [3]

Kasutatakse ainult 4-bitist liitjat, kuna iga suuremat järku ülekandebiti genereerimise algoritm muutub mahukamaks ja keerulisemaks, nagu Joonis 9 näitab. Seetõttu on seda lihtsam jagada väikesteks osadeks ja kasutada eraldi.

```
assign c[0]=cin;

assign c[1]= g[0]|(p[0]&c[0]);

assign c[2]= g[1] | (p[1]&g[0]) | p[1]&p[0]&c[0];

assign c[3]= g[2] | (p[2]&g[1]) | p[2]&p[1]&g[0]
| p[2]&p[1]&p[0]&c[0];

assign cout= g[3] | (p[3]&g[2]) | p[3]&p[2]&g[1]
| p[3]&p[2]&p[1]&g[0] | p[3]&p[2]&p[1]&p[0]&c[0];

assign sum=p^c;
```

Joonis 9 Ülekandebiti genereerimine

3.2 Mult8s

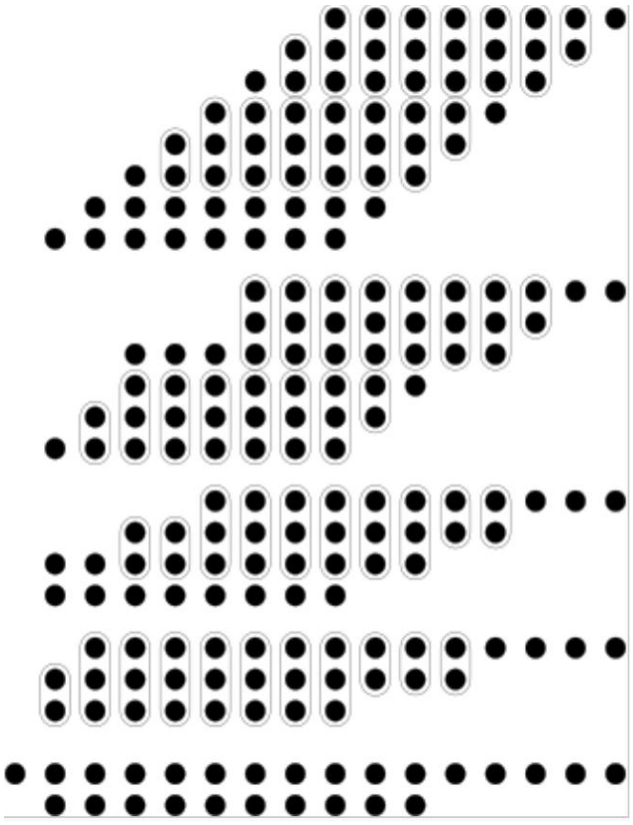
See moodul on saadud töö kaasjuherndajalt Mahdi Taheri-lt ja on Wallace'i puu korruti teostus. See korrutab ühe tsükli jooksul 2 sissetulevat väärtust, mis võimaldab meil väärtusi viivituseta korrutada. Sellele algoritmile lisati korrutatud registrite suuruse muutmise võimalus.

Wallace'i puu on andmestruktuur, mida kasutatakse mitme numbri liitmise toimingute kiirendamiseks. See struktuur koosneb liitmiste jadast ja nihutamisest arvutamise edenedes [12]. Iga kiht lisab 2 numbrit ja seejärel alandatakse kõrgemad numbrid, et saada madalamad numbrid. See meetod vähendab oluliselt liitmistoimingute arvu, mis on vajalik mitme numbriga toimingute tegemiseks. Joonisel 10 on näidatud 8x8 osalise produktmaatriksi 4-kihiline Wallace'i redutseerimine, kasutades 14 poolliitjat (kaks punkti) ja 38 täisliitjat (kolm punkti). Igas veerus olevad punktid on võrdse kaaluga bitid.

Esiteks suunatakse numbrid puu sisendisse, millel on kaks taset - esimesel tasemel on kaks sõlme, teisel - neli. Iga puu tase võtab kaks kõrvutiasetsevat numbrit, liidab need kokku ja edastab tulemuse järgmisele tasemele.

Lisaks jätkab puu laienemist, suurendades sõlmede arvu, kuni soovitud arv on saavutatud. Iga puu tase võtab kokku eelmise taseme tulemused ja annab need edasi.

Wallace'i puu algoritm säästab arvutusaega ja kasutab minimaalset arvu liitmistoiminguid. See muudab selle eriti kasulikuks paralleelarvutuses, kui peate kiiresti liitma suure hulga numbreid.



Joonis 10 8x8 osalise tootemaatriksi 4-kihiline Wallace'i redutseerimine [12]

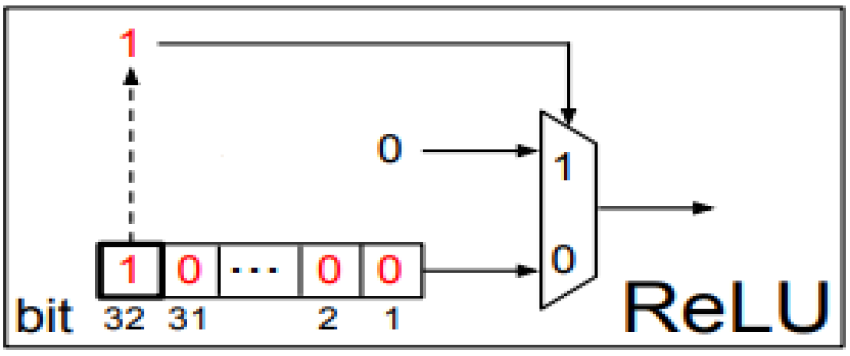
3.3 Accumulator

See moodul vastutab sisendandmete kogumise eest. See on “PE” mooduli alammoodul. Ja sellel on alammoodul “Adder”. Sisendis võtab ta korrutamise tulemuse *input_data* ja omistab selle väärtuse kohe siseregistrile *mult_result*. Samuti on olemas sisemine register *sum_result* ja *sum*, need on pärast taaskäivitamist võrdsed nulliga. Registrid *sum_result* ja *mult_result* lähevad mooduli “Adder” sisendväärtustena ja register *sum* väljundina. *cin* on seatud nullile, kuna väärtused võivad olla negatiivsed ja ülekandebitt pole sel juhul sobiv. Pärast seda määrame plokis *always_ff* väärtuse *sum* väärtusele *sum_result*.

Moodul saab ka signaali *start_accumulating*, mis annab märku vajalike andmete saabumisest. Register *number_iterations* sõltub parameetrist MATRIX_SIZE ja on vajalik sisendväärtuste kogumise peatamiseks pärast seda, kui sisemine loendur võrdub väärtusega *number_iterations+1*. Pärast seda kasutame tulemuse testimiseks ReLu-d.

ReLu funktsioon realiseeritakse väga lihtsalt, kui kõik olulised andmed on vastu võetud, siis kontrollime viimast numbrit märgi jaoks, kui arv on negatiivne, siis on registri *sum_result* väärtus võrdne nulliga, kui positiivne, siis salvestame selle register *sum_result*. Joonisel 11 on näidatud selle funktsiooni algoritm.

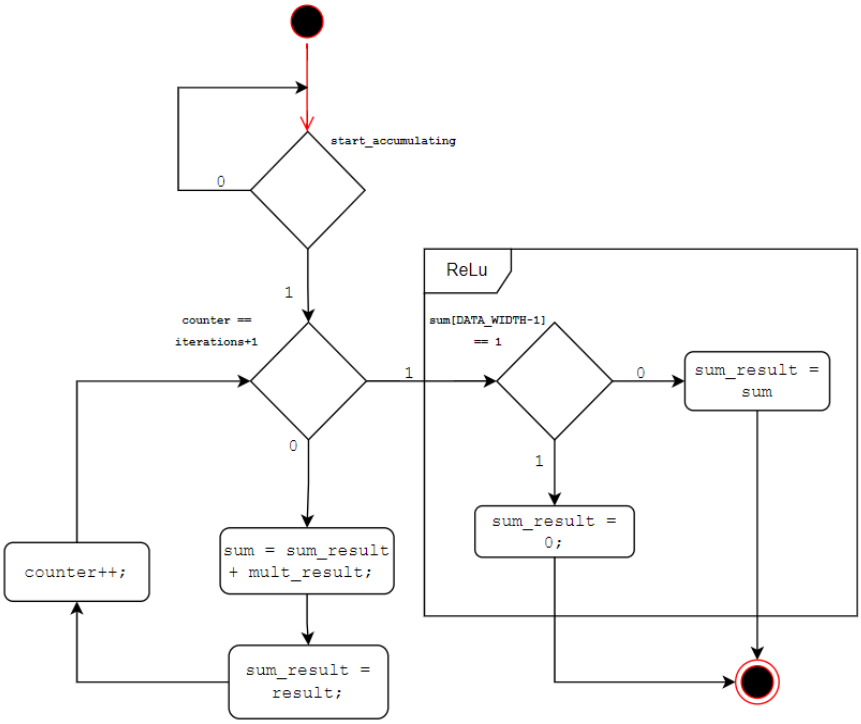
ReLu peamine ülesanne on kõrvaldada negatiivsed väärtused, et võrk saaks tõhusamalt treenida. Samuti aitab see vähendada treeninguaega ja vältida kaduvat gradiendi probleemi, mis ilmneb muude aktiveerimisfunktsioonide, nagu sigmod ja hüperboolne puutuja, puhul.



Joonis 11 32-bitise registri jaoks ReLu funktsiooni eest vastutav plokkskeem [7]

Moodul tagastab registri *result*, mis on võrdne *sum_result* väärtusega.

Joonis 12 näitab diagrammi, mis kirjeldab “Accumulator” moduli tööd.



Joonis 12 “Accumulator” mooduli diagramm

3.4 PE

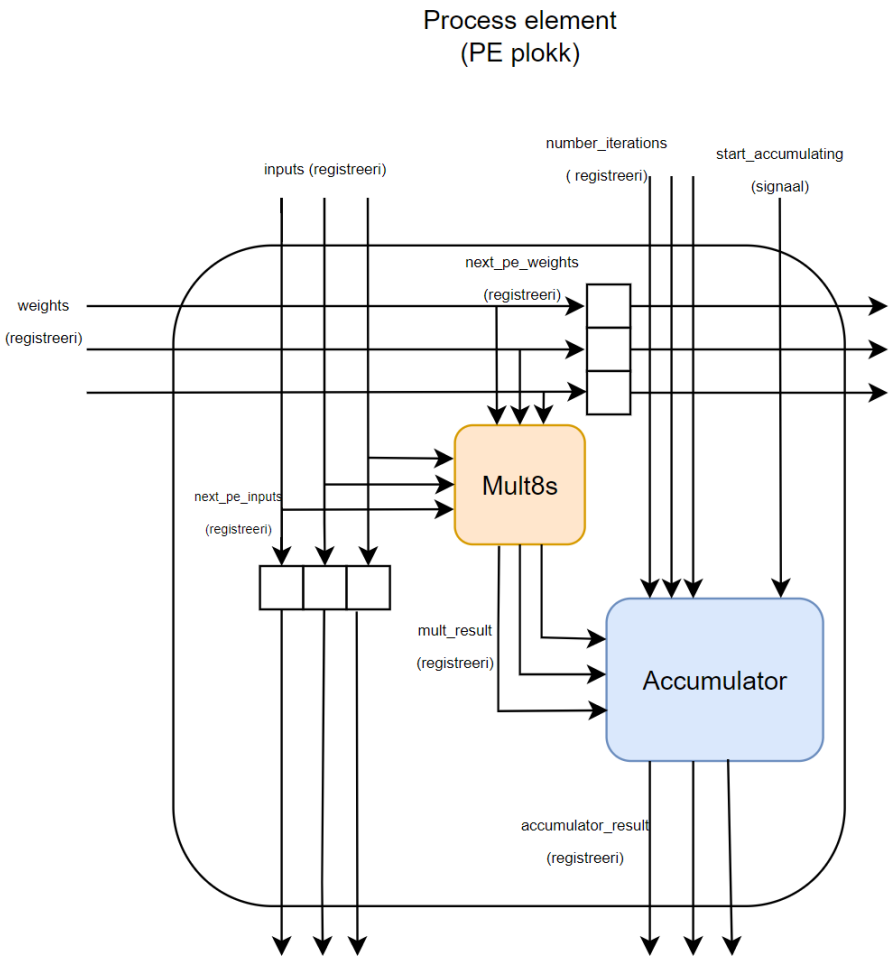
See moodul on süstoolse maatriksi arvutusüksus. See on “Systolic_Array” alammoodul. Sellel on 2 alammoodulit “Mult8s” ja “Accumulator”.

Moodul võtab vastu 2 sisendregistrit *weights* ja *inputs*, mis korrutatakse üksteisega “Mult8s” moodulis. Moodul tagastab 3 registrit: *next_pe_weights*, *next_pe_inputs* ja *acumulator_result*. Registrid *next_pe_weights* ja *next_pe_inputs* kannavad väärtusi *inputs* ja *weights*. Register *accumulator_result* väljastab mooduli “Accumulator” tulemuse.

Kaks sisendit *weights_data_ready* ja *inputs_data_ready* annavad märku, et moodulid *weights_register* ja *inputs_register* on valmis andmete laadimiseks “PE” moodulisse.

Igal taktil, kui signaalid *weights_data_ready* ja *inputs_data_ready* on võrdsed ühega, omistatakse väärtus *next_pe_inputs* ja *next_pe_weights*, mis on ühendatud järgmiste "PE"-plokkidega.

Ning moodulisse "Accumulator" saadetakse signaal sisendandmete summeerimise alustamiseks. Joonis 13 näitab visuaalset diagrammi selle kohta, kuidas sisend mõjustab sisemise loogikaga. Andmeid moodulist "Mult8s" hakatakse "Accumulator" moodulisse koguma alles peale *start_acumulating* signaali, mis annab märku õigetest andmetest sisendregistritesse.



Joonis 13 "PE" ploki visuaalne diagramm

3.5 Systolic_Array

See moodul on kontroller puhvrite ("wheights_register" ja "inputs_register") ja "PE" arvutusmoodulite vahel.

Selle mooduli alammoodulid on: “wheights_register”, “inputs_register” ja “PE”.

Moodul “PE” asub genereerimisplokis, kus olenevalt MATRIX_SIZE-st genereeritakse “PE” moodulite maatriks, kus sisendandmeteks on sisemised kahemõõtmelised registrite massiivid *next_inputs* ja *next_weights*. Joonisel 14 on kujutatud koodiosa, mis vastutab maatriksi genereerimise eest “PE”-moodulitest.

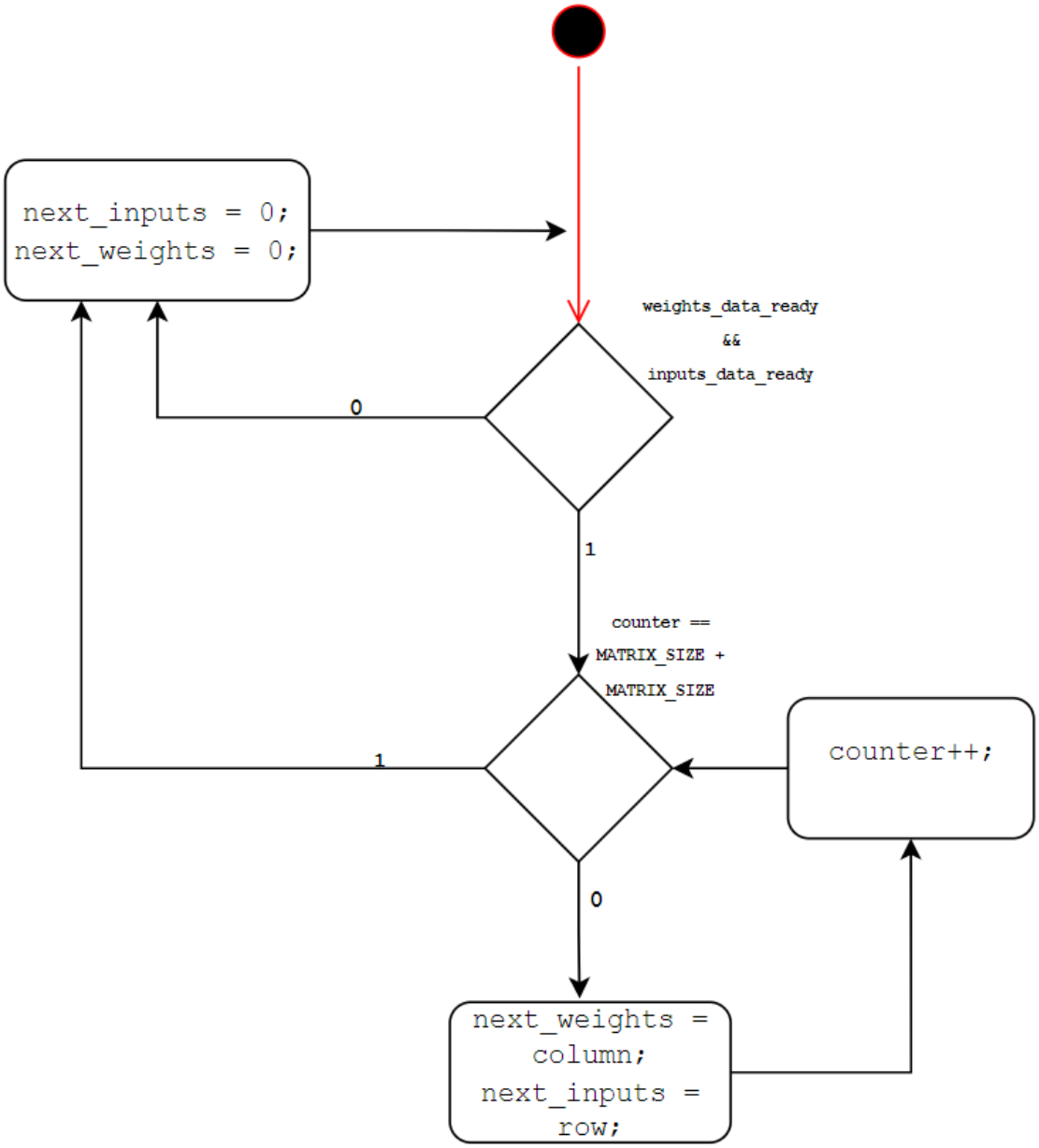
Üksteise sees olevad for tsüklid (ingl. k. Nested for loops) abil ühendame oma moodulid üksteisega vertikaalselt ja horisontaalselt. Joonis 6 (a) kujutab “PE”-moodulite struktuuri.

```
    genvar i, j;
    generate
        for (i = 0; i < MATRIX_SIZE; i++) begin
            for (j = 0; j < MATRIX_SIZE; j++) begin
                PE #(.DATA_WIDTH(DATA_WIDTH)) pe (
                .number_iterations      (MATRIX_SIZE  +  j  +
(MATRIX_SIZE * i)),
                .rstn                   (rstn),
                .clk                    (clk),
                .weights_data_ready     (weights_data_ready),
                .inputs_data_ready      (inputs_data_ready),
                .weights                (next_weights[i][j]),
                .inputs                 (next_inputs[i][j]),
                .next_pe_weights        (next_weights[i][j+1]),
                .next_pe_inputs         (next_inputs[i+1][j]),
                .accumulator_result     (array_results[i][j]));
            end
        end
    endgenerate
```

Joonis 14 Generate plokk “Systolic_Array” mooduli jaoks

Moodul “Systolic_Array” aktsepteerib *inputs* ja *weights* registrite väärtusi ja saadab need vastavatesse moodulitesse “wheights_register” ja “inputs_register”, kuni saab nende moodulite valmisoleku signaalid andmete saatmiseks.

Kui andmed on valmis, viime need moodulitest “wheights_register” ja “inputs_register” siseregistrite massiividesse *row* ja *column*. Seejärel paigutatakse iga register nendest massiividest maatriksite *next_weights* ja *next_inputs* algusesse. Joonisel 15 on diagramm süstolüütilise maatriksi tööst pärast seda, kui on saabunud signaalid puhvri valmisoleku kohta. Nagu näha, töötab süstoolne maatriks peatumata ja ootab pidevalt andmeid, kui neid pole, siis saadab “PE” moodulitest maatriksile alati nullid.



Joonis 15 “Systolic_Array” mooduli diagramm

3.6 wheights_register ja inputs_register

Mõlemad moodulid vastutavad andmete salvestamise ja ettevalmistamise eest nende laadimiseks süstoolsesse maatriksisse.

Need on “Systolic_Array” alammoodulid.

Registrid *inputs* ja *weights* tuleb nihkeregistritega nihutada, et iga sisend ühest maatriksist jõuaks õige “PE”-ni täpselt õigel ajal, et seda teise maatriksi õige sisendiga korrutada-akumuleerida.

Need moodulid võtavad “Systolic_Array” sisendväärtused ja panevad need registrite sisemisse maatriksisse, kus ridade või veergude arvu, olenevalt moodulist, suurendatakse 2*MATRIX_SIZE-2 võrra – see on suurus, millesse kogu nihutatud maatriks mahub.

Pärast signaali andmist laaditakse andmed järjekorras registrite massiivi, mis seejärel saadetakse registrite väljundmassiivi.

Joonisel 16 on näidatud, kuidas mõlema mälupuhvri andmed määratakse “Systolic_Array” moodulis leiduvale registrite massiivile. Joonisel 16 (a) on näidatud, kuidas on määratud veergude väärtused registrite väljundmassiivile, mis tähendab, et registris *sisend_register* salvestatud andmed liiguvad süstoolses maatriksis ülalt alla. Vastupidi, joonisel 16 (b) on väljundandmete massiivile määratud reaväärtused, mis tähendab, et *weights* registrid andmed liiguvad piki süstoolset maatriksit vasakult paremale.

```
if(!rstn) begin
    row <= 0;
    column <= 0;
    inner_data_ready <= 0;


    for (int i = 0; i < MATRIX_SIZE; i++) begin
         prepared_output_data[i] <= 0;
    end

    for (int i=0; i < (MATRIX_SIZE + MATRIX_SIZE)-1; i++) begin
        for (int j=0; j < MATRIX_SIZE; j++) begin
            inputs_matrix[i][j] <= 0;
        end
    end
end
else if (row == MATRIX_SIZE) begin
    inner_data_ready <= 1;
    if (weights_data_ready) begin
        for (int i = 0; i < MATRIX_SIZE; i++) begin
             prepared_output_data[i] <= inputs_matrix[colum][i];
        end
        column += 1;
     end
end
```

(a) inputs_register kood

```
if(!rstn) begin
    row <= 0;
    column <= 0;
    inner_data_ready <= 0;


    for (int i = 0; i < MATRIX_SIZE; i++) begin
         prepared_output_data[i] <= 0;
    end

    for (int i=0; i < MATRIX_SIZE; i++) begin
        for (int j=0; j < (MATRIX_SIZE + MATRIX_SIZE)-1; j++) begin
            weights_matrix[i][j] <= 0;
        end
    end

end
else if (colum == MATRIX_SIZE) begin
    inner_data_ready <= 1;
    if (inputs_data_ready) begin
        for (int i = 0; i < MATRIX_SIZE; i++) begin
             prepared_output_data[i] <= weights_matrix[i][row];
        end
        row += 1;
   end
end
```

(b) weights_register kood

Joonis 16 Kood andmete mälupuhvrist mahalaadimiseks

4 Testimine

Peatükk kirjeldab algoritmi testimise etappi reaalsel arendusplaadil. Testimiseks mõeldud plaat oli Basys 3. Selle projektiga plaadil töötamiseks kirjutati eraldi moodul Satic_memory, mis võimaldab koodi mitte muuta ja ainult selle plaadi jaoks kohandada.

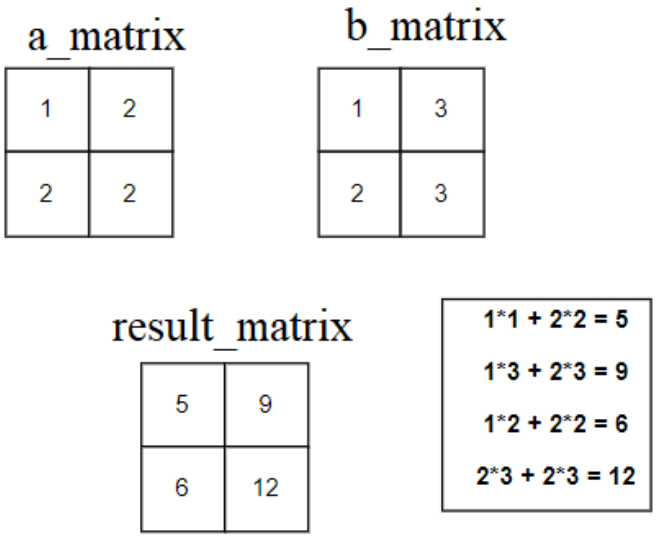
“Satic_Memory” moodul, kuvab 7-segmendilisel ekraanil maatriksi väärtust, millest plaadil on 4, mis võimaldab kuvada 2x2 maatriksi väärtusi.

Panime kahe maatriksi andmed eelnevalt registrite *a_matrix* ja *b_matrix* maatriksisse, nagu on näha Jooniselt 17.

```
    logic     [DATA_WIDTH-1:0]   a_matrix   [MATRIX_SIZE   -
1:0][MATRIX_SIZE - 1:0] = '{'{2, 2}, '{2, 1}};

    logic     [DATA_WIDTH-1:0]   b_matrix   [MATRIX_SIZE   -
1:0][MATRIX_SIZE - 1:0] = '{'{3, 2}, '{3, 1}};

    logic      [DATA_WIDTH-1:0] result_matrix [MATRIX_SIZE
- 1:0][MATRIX_SIZE - 1:0] = '{'{0, 0}, '{0, 0}};
```

Joonis 17 Sisendmaatriksite initsialiseerimine moodulis “Satic_Memory”

Vajame andmekogumit, mis ei annaks tulemust üle 15, kuna 7-segmendiline ekraan ei suuda selliseid numbreid kuvada. Samuti on Joonisel 18 näha tulemus, mille peaksime saama korrutamise tulemusena.



Joonis 18 Maatriksi korrutamise tulemus “Satic_Memory” moodulis

Kui signaal *writing_signal* on 1, hakkab programm kahest maatriksist andmeid laadima moodulisse “Systolic_Array”, pärast tulemuse valmimist saame moodulist “Systolic_Array” signaali *start_transerf_data_from_Systolic_Array*, mis saab käivitajaks protsessile, mis laadib kõik väärtused maha “Systolic_Array”-st *matix_result* maatriksi jaoks. Joonisel 19 on kujutatud koodi osa, mis vastutab andmete laadimise eest moodulist “Systolic_Array”. LED-i eest vastutab *result_matrix_ready* signaal, kui see on 1, siis LED süttib. Registreeri *number_for_c_matrix* - edastab valmis tulemuse väärtused moodulist “Systolic_Array”, mis teisaldab valmis maatriksi *result_matrix*-sse.

```
logic ir, jr;
always_ff @ (posedge clk) begin
   if (!rstn) begin
       ir <= 0;
       jr <= 0;
   end
   else if (result_ready) begin
        result_matrix[ir][jr] <= number_for_c_matrix;
        ir += 1;
        if (ir == MATRIX_SIZE) begin
            jr += 1;
            ir <= 0;
        end
    end
end
```

Joonis 19 Maatriksi andmetega *result_marix* ülekandmiseks moodulist “Systolic_Array” moodulisse “Static_Memory”

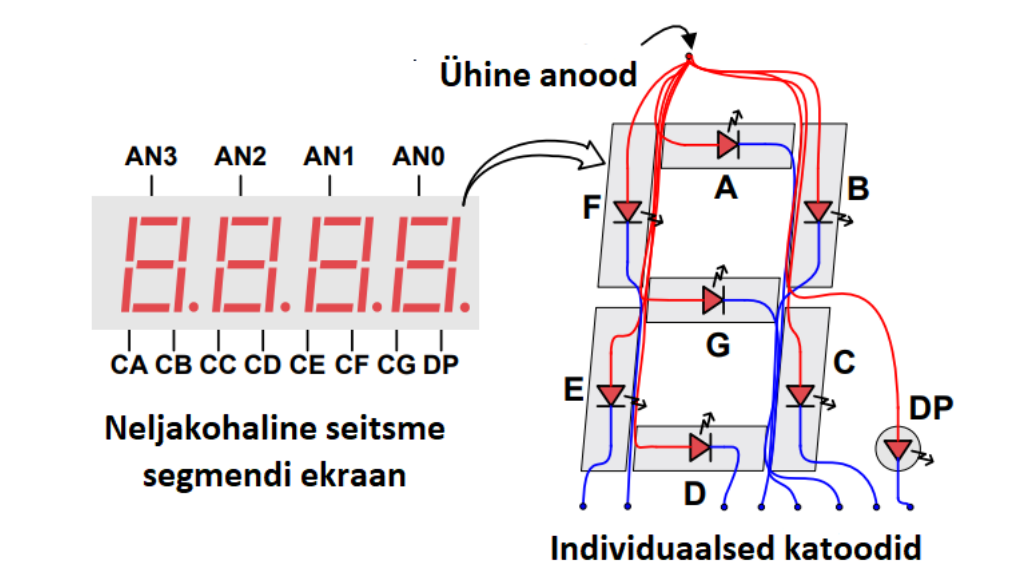
Matiks *display_matrix* vastutab maatriksi väljundi eest, mis sõltuvalt lülitist võtab erinevate maatriksite väärtused, nagu on näidatud Joonisel 20.

```
  always_ff @ (posedge clk) begin
        if (!rstn) begin
            display_matrix <= '{'{0, 0}, '{0, 0}};
        end
        else begin
            case(buttoms)
                1 : display_matrix = a_matrix;
                2 : display_matrix = b_matrix;
                4 : display_matrix = result_matrix;
                default    display_matrix = '{'{0, 0}, '{0,
0}};
            endcase
        end
  end
```

Joonis 20 Väärtuse *display_matix* muutumine sõltuvalt lülitist väärtusest

Basys 3 plaadil on üks neljakohaline ühise anoodiga seitsmesegmendiline LED-ekraan. Kõik neli numbrit koosnevad seitsmest segmendist, mis on paigutatud "joonis 8" mustri järgi ja igasse segmenti on integreeritud LED.
Iga numbrit moodustava seitsme LED-i anoodid on ühendatud üheks "ühise anoodi" ahela sõlmeks, kuid LED-katoodid jäävad eraldiseisvaks, nagu on näidatud joonisel 21. Ühise anoodi signaalid on saadaval nelja "numbri lubamise" sisendsignaalina. 4-kohaline ekraan. Sarnaste segmentide katoodid kõigil neljal kuvaril on ühendatud seitsmesse ahela sõlme, millel on sildid CA kuni CG (näiteks neli "D" katood neljast numbrist on rühmitatud üheks ahela sõlmeks, mida nimetatakse "CD"). Need seitse katoodsignaali on saadaval 4-kohalise ekraani sisenditena. See signaaliühendusskeem

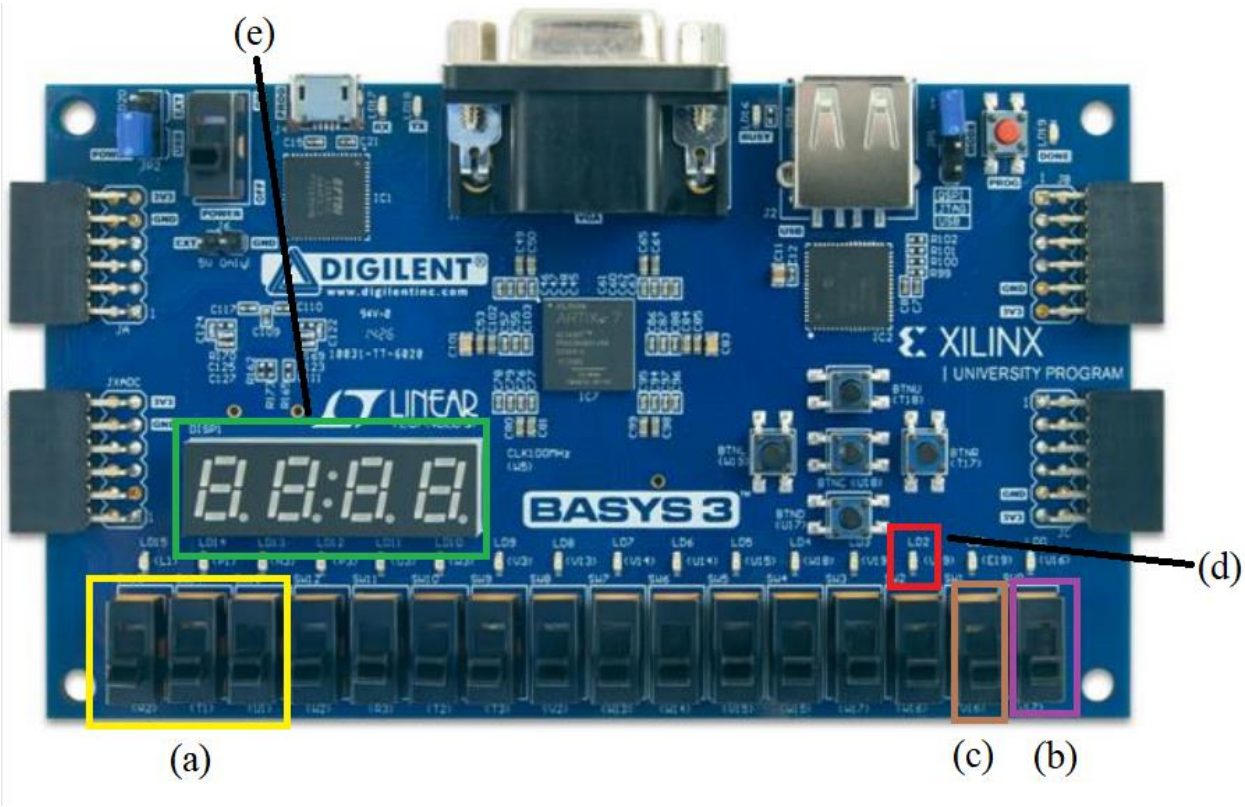
loob multipleksitud kuva, kus katoodsignaalid on ühised kõikidele numbritele, kuid need võivad valgustada ainult neid numbri segmente, millele vastav anoodisignaal on kinnitatud [2].



Joonis 21 Ühine anoodiahela sõlm [2]

Joonisel 22 on Basys 3 arendusplaat.

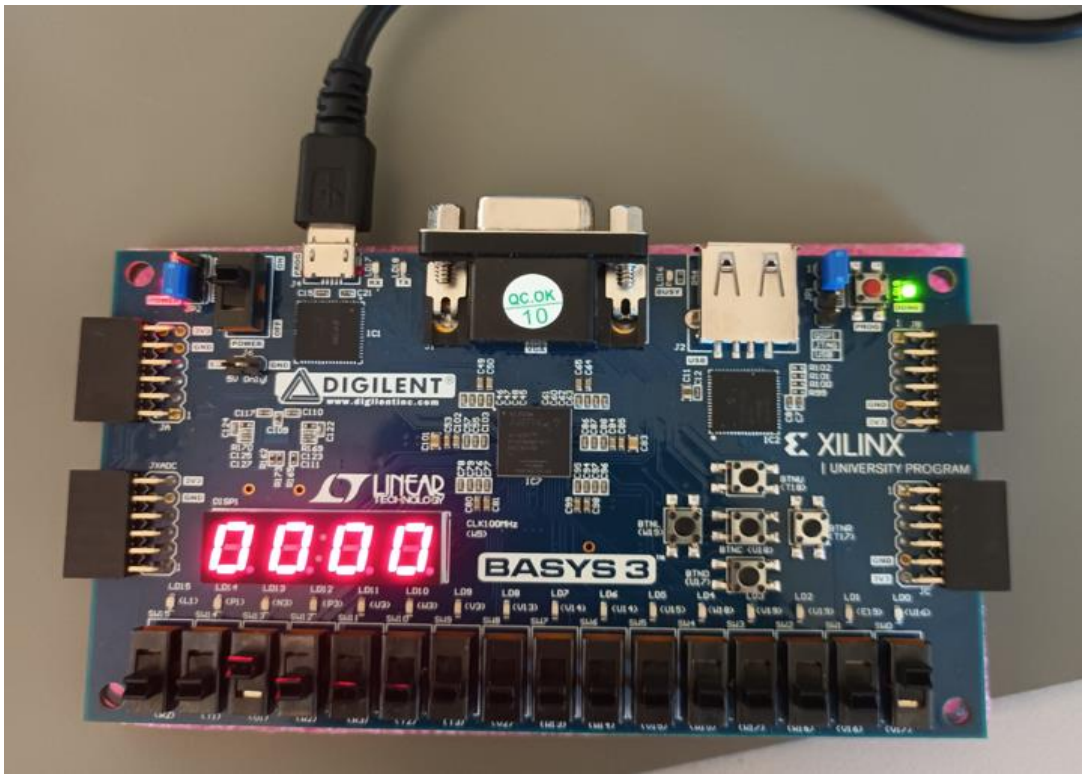
Joonisel 22 (a) on kujutatud maatriksilülitit, mis valib, millist maatriksit näidata. Lüliti r2 näitab maatriksit a, lüliti t1 näitab maatriksi b väärtust ja lülitab välja u1 näitab maatriksi c väärtust (tulemus). Joonis 22 (b) lüliti vastutab v17 lähtestussignaali (ingl. k. reset) eest. Joonisel 22 (c) on näidatud *start_writing* signaali saatmise eest vastutav lüliti v16, mis käivitab maatriksi korrutamise protsessi. Joonis 22 (d) näidatud LED v19 vastab signaalile *result_ready*, mis annab teada, et korrutamise tulemus on valmis. Joonisel 22 (e) on kujutatud 7-segmendiline ekraan, mis kuvab 2x2 maatriksi väärtused.



Joonis 22 Basys 3 arendusplaat [2]

Projekti testimiseks valiti parameetrid MATRIX_SIZE = 2 ja DATA_WIDTH = 4, mis tähendab, et algmaatriksite ja tulemuse suurus on 2x2 ning kõigi registrite suurus on 4 bitti. Sellised parameetrid sobivad ideaalselt nende kuvamiseks 7-l segmendil.

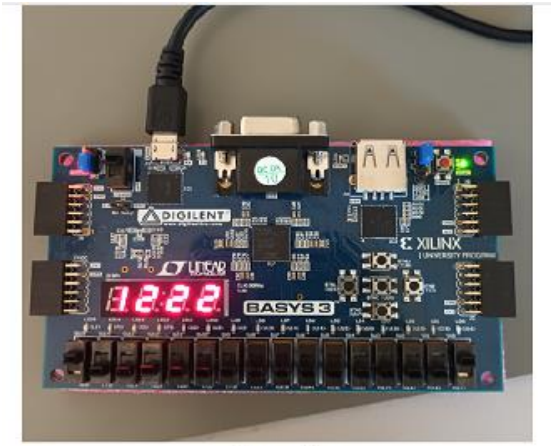
Pärast koodi edukat tahvlile üleslaadimist peate projekti uuesti laadimiseks ja kõigi loendurite lähtestamiseks kohe määrama *rstn*-i väärtused väärtusele 1. Kui kõik ekraanid näitavad nullväärtusi, on kõik koras. Kui lülitate maatriksi c (u1) sisse, näete ekraanil ka nulle, kuna signaali *writing_signal* pole veel saadetud. Joonisel 23 on plaat pärast taaskäivitamist.



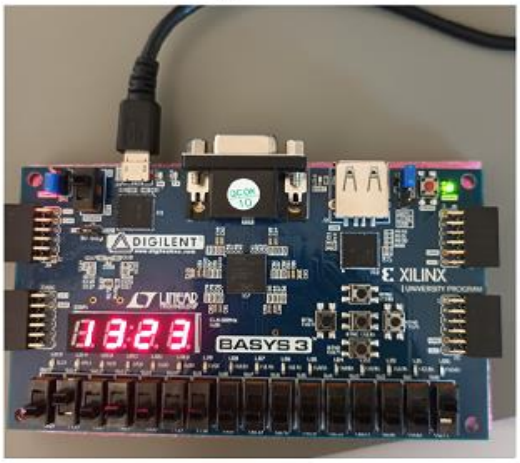
Joonis 23 Plaat pärast lähtestussignaali

Lisaks kuvame näiteks esimese maatriksi väärtused ja seejärel teise maatriksi väärtused, kui rohkem kui üks lüliti on sisse lülitatud, on väljundmaatriksi väärtus võrdne nulliga. Kolmandas maatriksis pole seni ühtegi tulemust peale nulli.

Lülitite sisselülitamiseks saate vaadata algsete maatriksite väärtusi: r2 vastutab maatriksi a sisselülitamise eest, nagu on näha Joonisel 24 (a). t1 vastutab maatriksi b kaasamise eest, nagu on näha Jooniselt 24 (b).



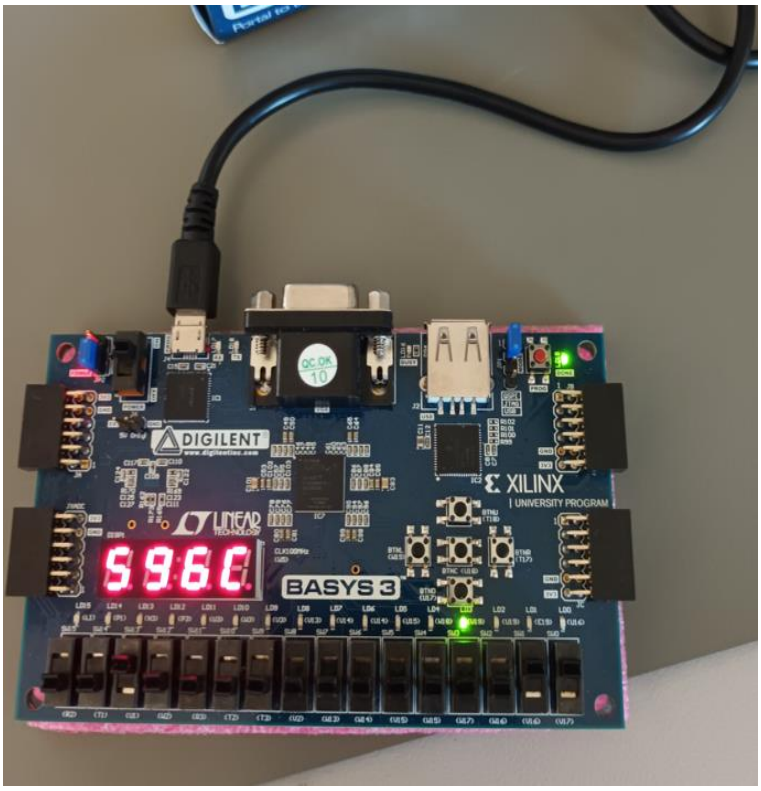
(a) maatriks a



(b) maatriks b

Joonis 24 Maatriksi a ja maatriksi b väärtus

Korrutamise tulemuse saamiseks tuleb lülitada *write_signal* lüliti ja koheselt sütib LED, mis vastutab *result_ready* eest, andes meile teada, et tulemus on valmis. Joonis 25 näitab meie korrutamise tulemust 7-segmendilisel kuval.



Joonis 25 Maatrikskorrutamise tulemus

5 Kokkuvõte

Käesoleva töö ülesanne koosnes kahest osast. Esimene osa oli süstoolse maatriksi koodi kirjutamine, mida kasutatakse närvivõrgu kiirendi arvutusliku tuumana, mida minu juhendaja Jaan Raiki meeskond kasutab edasiseks uurimiseks. Teine osa seisnes selle projekti testimises Basys 3 arendusplaadil ja selle projekti jaoks lisamooduli kirjutamises, mis ei peaks muutma projekti sisemist loogikat, vaid olema vaid sellele nagu "lisand" ja peale selle eemaldamist projektist ei mõjuta see projekti ennast.

Esimene osa ülesandest on täidetud ja hetkel on olemas 7 moodulist koosnev arvutusmoodul. Projektil on reguleeritavad parameetrid süstoolse maatriksi ja registrite suuruse jaoks. Projekt ise võib töötada nii negatiivsete kui ka positiivsete arvudega. Projektil on ka sisemälu sissetulevate massiivide salvestamiseks.

Teise osa käigus loodud moodul on võimeline töötama Basys 3 välisseadmetega ja kuvama 3 maatriksi andmeid 7-segmendilisel ekraanil. Seda moodulit on projektis lihtne rakendada ja see mõjutab põhiprojekti minimaalselt.

Kasutatud kirjandus

[1] Ahmad Shawahna, Sadiq M. Sait, Aiman El-Maleh. (2019. a.). *FPGA-Based Accelerators of Deep Learning*. [Võrgumaterjal]. Allikas: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8594633. [Kasutatud 11 05 2023].

[2] *Basys 3 Board Manual*. (10. Juuli 2019. a.). [Võrgumaterjal]. Allikas: https://moodle.taltech.ee/pluginfile.php/624152/mod_resource/content/2/Basys3_Board_Manual.pdf. [Kasutatud 11 05 2023].

[3] *Carry Lookahead Adder*. (30. Detsember 2021. a.). [Võrgumaterjal]. Allikas: https://www.watelectronics.com/carry-lookahead-adder/. [Kasutatud 13 05 2023].

[4] Cheryala, N. (30. Oktoober 2020. a.). *Systolic Arrays and the TPU.* [Võrgumaterjal]. Allikas: https://www.linkedin.com/pulse/systolic-arrays-tpu-neeraj-cheryala. [Kasutatud 13 05 2023].

[5] Elliott Delaye, P. E. (5. Mai 2022. a.). *Exploration and Tradeoffs of Different Kernels.* [Võrgumaterjal]. Allikas: https://www.ispd.cc/slides/2018/s2_3.pdf. [Kasutatud 13 05 2023].

[6] Ganapathi Hegde, C. P. (2009. a.). Implementation of Systolic Array Architecture for Full Search. [Võrgumaterjal]. Allikas: https://static.aminer.org/pdf/PDF/000/343/667/algorithms_for_high_speed_multi_dimensional_arithmetic_and_dsp_systolic.pdf. [Kasutatud 13 05 2023].

[7] H.T. Kung, B. M. (Aprill 2019. a.). *Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization.* [Võrgumaterjal]. Allikas: http://www.eecs.harvard.edu/~htk/publication/2019-asplos-kung-mcdanel-zhang.pdf . [Kasutatud 16 05 2023].

[8] Mahajan, P. (23. Octoober 2020. a.). *Fully Connected vs Convolutional Neural Networks*. [Võrgumaterjal]. Allikas: https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5. [Kasutatud 16 05 2023].

[9] Mahendra Vucha, A. R. (Juuli 2011. a.). *Design and FPGA Implementation of Systolic Array Architecture for Matrix Multiplication.* [Võrgumaterjal]. Allikas: https://www.researchgate.net/profile/Mahendra-Vucha-2/publication/252951671_Design_and_FPGA_Implementation_of_Systolic_Array_Architecture_for_Matrix_Multiplication/links/5440f4ae0cf251bced617b40/Design-and-FPGA-Implementation-of-Systolic-Array-Architectur. [Kasutatud 16 05 2023].

[10] Saha, S. (15. Detsember 2018. a.). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. [Võrgumaterjal]. Allikas: https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/. [Kasutatud 16 05 2023].

[11] *Systolic CNN AcceLErator Simulator (SCALE Sim) v2*. [Võrgumaterjal]. Allikas: https://github.com/scalesim-project/scale-sim-v2/blob/main/README.md. [Kasutatud 16 05 2023].

[12] *Wallace tree*. (Detsember 2009. a.). [Võrgumaterjal]. Allikas: https://en.wikipedia.org/wiki/Wallace_tree. [Kasutatud 16 05 2023].

[13] Yeh, T. T. *Reconfigurable Dataflow on DNN Systolic Accelerator.* [Võrgumaterjal]. Allikas: https://dpeecs.nycu.edu.tw/df_ufiles/017/111A%20Seminar%20第一場-葉宗泰教授.pdf. [Kasutatud 16 05 2023].

[14] Fasih Ud Din Farrukh, T. X. (2018. a.). *Optimization for Efficient Hardware.* [Võrgumaterjal]. Allikas: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8706067&casa_token=L5VeMNk6lUcAAAAA:SWI02mbfyA7ttsHumf2aDnARxy2icMCk46qslSawU8M6q-90uAT8VIETymOmyXOua3O4oT05gw. [Kasutatud 16 05 2023].

[15] Uniqtech. (30. Jaanuar 2018. a.). *Understand the Softmax Function in Minutes.* [Võrgumaterjal]. Allikas: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d. [Kasutatud 18 05 2023].

[16] Holzbauer, L. (30. Detsember 2019. a.). *Convolutional Neural Networks Explained…with American Ninja Warrior.* [Võrgumaterjal]. Allikas: Blog.insightdatascience: https://blog.insightdatascience.com/convolutional-neural-networks-explained-with-american-ninja-warrior-c6649875861c. [Kasutatud 02 04 2023].

[17] Kaiyuan Guo, S. Z. *A Survey of FPGA-Based Neural Network Inference.* [Võrgumaterjal]. Allikas: https://arxiv.org/pdf/1712.08934.pdf. [Kasutatud 06 05 2023].

[18] Keiron O'Shea, R. N. *An Introduction to Convolutional Neural Networks.* [Võrgumaterjal]. Allikas: https://arxiv.org/pdf/1511.08458.pdf. [Kasutatud 14 05 2023].

[19] S.H. Shabbeer Basha, S. R. (8. Jaanuar 2020. a.). *Impact of fully connected layers on performance of convolutional neural networks for image classification.* [Võrgumaterjal]. Allikas: https://www.sciencedirect.com/science/article/pii/S0925231219313803. [Kasutatud 02 04 2023].

[20] Saad Albawi, T. A.-Z. (2017. a.). *Understanding of a Convolutional Neural Network.* [Võrgumaterjal]. Allikas: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8308186. [Kasutatud 16 05 2023].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Nikita Budovey

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Süstoolsel maatriksil põhineva närvivõrgu kiirendi realisatsioon FPGAs", mille juhendaja on Jaan Raik.

    1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

    1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

15.05.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – SystemVerilogis kirjutatud mooduli Static_Memory kood

```
`timescale 1ns / 1ps

module Static_memory(
    input        rstn,
    input        clk,
    input        [2:0] buttoms,
    input        writing_signal,

    output       ready,
    output       [10:0] seven_segments
    //output       [3:0] anode
    );

    parameter DATA_WIDTH = 4;
    parameter MATRIX_SIZE = 2;

    logic  [DATA_WIDTH-1:0] a_matrix [MATRIX_SIZE - 1:0][MA-
TRIX_SIZE - 1:0] = '{'{2, 2}, '{2, 1}};
    logic  [DATA_WIDTH-1:0] b_matrix [MATRIX_SIZE - 1:0][MA-
TRIX_SIZE - 1:0] = '{'{3, 2}, '{3, 1}};
    logic  [DATA_WIDTH-1:0] result_matrix [MATRIX_SIZE -
1:0][MATRIX_SIZE - 1:0] = '{'{0, 0}, '{0, 0}};

    logic signed [DATA_WIDTH-1:0] input_number;
    logic signed [DATA_WIDTH-1:0] weight_number;
    logic [DATA_WIDTH-1:0]        number_for_c_matrix;

    logic           result_ready, writing_signal, delay_signal,
start_fetching;
    logic           switch_detection;
    logic     [2:0]      current_position_of_switch;
    logic     [2:0]      previus_position_of_switch;
    integer         counter_for_dealy;


    logic [MATRIX_SIZE-1:0] colum, row;
    logic [10:0] output_for_seven_segments;

      Systolic_Array #(.MATRIX_SIZE(MATRIX_SIZE),
.DATA_WIDTH((DATA_WIDTH))) Systolic_Array (
       .rstn                (rstn),
       .clk                 (clk),
       .inputs              (input_number),
       .weights             (weight_number),
```

```
        .writing_signal      (start_fetching),
        .number_for_c_matrix (number_for_c_matrix),
        .result_ready        (result_ready));



    logic [2:0] ir, jr;
    logic result_matrix_ready;

     always_ff @ (posedge clk) begin
        current_position_of_switch <= buttoms;
        previus_position_of_switch <= current_posi-
tion_of_switch;
        switch_detection <= current_position_of_switch != pre-
vius_position_of_switch;
    end

    always_ff @ (posedge clk) begin
       if (!rstn) begin
           ir <= 0;
           jr <= 0;
           result_matrix_ready <= 0;
       end
       else if (number_for_c_matrix) begin

       if (ir == MATRIX_SIZE && jr == MATRIX_SIZE-1) begin
            result_matrix_ready <= 1;
       end else begin

            result_matrix[ir][jr] <= number_for_c_matrix;

            jr += 1;

            if (jr == MATRIX_SIZE) begin
                ir += 1;
                jr <= 0;
            end


        end
    end
  end

    always_ff @ (posedge clk) begin

        if (!rstn) begin
            input_number <= 0;
            weight_number <= 0;
            row <= 0;
            colum <= 0;
            start_fetching <= 0;

        end
        else if (writing_signal) begin
            start_fetching <= 1;
            input_number  <= a_matrix[row][colum];
```

```
            weight_number <= b_matrix[row][colum];
            colum += 1;
            if (colum == MATRIX_SIZE) begin
                row += 1;
                colum <= 0;
            end
        end

    end

  logic [DATA_WIDTH-1:0] display_matrix [MATRIX_SIZE-1:0][MA-
TRIX_SIZE-1:0];

  always_ff @ (posedge clk) begin
        if (!rstn) begin
            display_matrix <= '{'{0, 0}, '{0, 0}};

        end
        else begin
            case(buttoms)
                1 : display_matrix = a_matrix;
                2 : display_matrix = b_matrix;
                4 : display_matrix = result_matrix;
                default     display_matrix = '{'{0, 0}, '{0,
0}};
            endcase
        end
  end

  logic [3:0] number_of_segments, shifter;
  logic [MATRIX_SIZE-1:0] i, j;
  reg [6:0]    r_Hex_Encoding = 7'h00;

  always_ff @ (posedge clk) begin

        if (!rstn || switch_detection) begin
            r_Hex_Encoding <= 0;
            shifter <= 4'b0001;
            i <= 0;
            j <= 0;
        end
        else if (delay_signal) begin

            if (shifter == 4'b0001) begin
                shifter <= 4'b1000;
            end else begin
                shifter <= shifter >> 1;
            end



            case(display_matrix[i][j])
                4'b0000 : r_Hex_Encoding <= 7'h7E;
                4'b0001 : r_Hex_Encoding <= 7'h30;
                4'b0010 : r_Hex_Encoding <= 7'h6D;
                4'b0011 : r_Hex_Encoding <= 7'h79;
```

```
                4'b0100 : r_Hex_Encoding <= 7'h33;
                4'b0101 : r_Hex_Encoding <= 7'h5B;
                4'b0110 : r_Hex_Encoding <= 7'h5F;
                4'b0111 : r_Hex_Encoding <= 7'h70;
                4'b1000 : r_Hex_Encoding <= 7'h7F;
                4'b1001 : r_Hex_Encoding <= 7'h7B;
                4'b1010 : r_Hex_Encoding <= 7'h77;
                4'b1011 : r_Hex_Encoding <= 7'h1F;
                4'b1100 : r_Hex_Encoding <= 7'h4E;
                4'b1101 : r_Hex_Encoding <= 7'h3D;
                4'b1110 : r_Hex_Encoding <= 7'h4F;
                4'b1111 : r_Hex_Encoding <= 7'h47;
            endcase



            j += 1;
            if (j == MATRIX_SIZE) begin
                i += 1;
                j <= 0;
            end

            if (i == 2) begin
                i <= 0;
                j <= 0;
            end

        end
    end


    always_ff @ (posedge clk) begin

        if (counter_for_dealy != 40000) begin

            counter_for_dealy += 1;
            delay_signal <= 0;
        end
        else begin
            delay_signal <= 1;
            counter_for_dealy <= 0;
        end

    end


    assign ready = result_ready;
    assign seven_segments = {~r_Hex_Encoding, ~shifter};

endmodule
```

Joonis 26 Static_Memory mooduli kood