TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Artjom Juškov 164414

ARITMEETIKA ALGORITMIDE PROTOTÜÜPIMINE FPGA'L

Bakalaureusetöö

Juhendaja: Peeter Ellervee
Professor / Ph.D

Tallinn 2019

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: Artjom Juškov

20.05.2019

Annotatsioon

Lõputöö raames pakendatakse Vivado projekt erinevatest moodulitest. Vivado projekt võimaldab uue aine raames tudengitel enda sünteesitava algoritmi integreerida Vivado projekti. Samuti on võimalik genereerida projektist *bitstream* , laadida selle prototüüpimisplaadile Basys 3, et tudengil oleks võimalik vaadelda enda algoritmi tööd praktiliselt.

Lõputöö raames luuakse Basys 3 prototüüpimisjuhendi, kus kirjeldatakse kuidas seadistada tarkvara, kuidas laadida Basys 3 paadile. Samuti kirjeldatakse ära, mis nuppudega teha, mis lülititega teha saab, mida saab kuvada ekraanil ning mida kuvatakse valdusdioodidel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 6 peatükki, 25 joonist, 2 tabelit.

Abstract

Prototyping of Arihmetic Algorithms on FPGA

In the thesis will be packed Vivado project from different modules. Students can integrate own built synthesizable algorithms into the Vivado project. Students can generate a bitstream of the project and load bitstream to the prototyping plate Basys 3, so students can see the working algorithm in real-time.

In the thesis will be written Basys 3 prototyping guide for students, which describes how to configure the software, how to load a bitstream to the Basys 3 board. It also describes what you can do with buttons and switches, what can be displayed on the screen and what LED indicators to indicate.

The thesis is in Estonian and contains 26 pages of text, 6 chapters, 25 figures, 2 tables.

Lühendite ja mõistete sõnastik

ATI	TTÜ Arvutitehnika instituut
FPGA	*Field-programmable gate array*, korduvprogrammeeritav loogika
VHDL	Riistvara kirjeldus keel
LED	Valgusdiood
GCD	*Greatest common divisor*
LUT	*Look-up-table* ehk tõeväärtus tabel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

1.1 Taust

TalTech'is ühendati kaks ainet kokku üheks aineks: „Digitaalsüsteemid" ja „Arvutite aritmeetika ja loogika". Mõlema aine läbimiseks oli vaja teha mitu kodutööd. Ainete ühendamine andis mitme omavahel raskesti seostatava või näiliselt ebahuvitava kodutöö üheks suuremaks, mis algab algoritmist ja lõpeb töötava riistvaraga. Kuna ajaliselt ei olnud võimalik realiseerida mõlema aine kodutöid, siis ainsaks variandiks oli kodutööde ühendamine ühte. Samuti väikesed kodutööd on tihti reaalsusest kaugemal. Suuremad kodutööd aitavad paremini siduda erinevaid aspekte väikestest kodutöödest eriti siis, kui kõik lõpeb praktilise realisatsiooniga.

Uues kodutöös on tudengi eesmärk realiseerida aritmeetika algoritm kasutades riistvara kirjeldus keelt VHDL'i [11]. Algoritmi realisatsiooni ülesande paremaks sidumiseks riistvara projekteerimise õppimisega on viimaseks alamülesandeks algoritmi demonstratsioon FPGA'l põhineval prototüüpimisplaadil. Prototüüpimisplaat sisaldab lisaks FPGA'le sisendeid lülitite ja väljundeid LED-indikaatorite näol.

1.2 Eesmärgid

Lõputöö eesmärgiks on koostada valmis töötav VHDL moodulite komplekt pakendatud Vivado projektina, kus üheks mooduliks on tudengi poolt kirjutatud algoritm. Lõputöö tulemiks on praktiline väljund, mida tudengid saavad kasutada uue koodiga aines „Digitaalsüsteemid". Tulemit võib menetleda kui teeki, kuna moodulid moodustavad ühe terviku. Tulemi sünteesiks on kasutatud Xilinx Vivado, lõputöö loomise ajal kasutati 2018.3 versiooni ning prototüüpimiseks Digilent Basys 3.

Xilinx Vivado projekt sisaldab mooduleid, mis lubavad andmete sisestamist, algoritmi juhtimist ja tulemuste kuvamist. Algoritmi juhtimise võimalusi on kaks. Üks võimalus on kuvada otse tulemus ning teine – kuvada vahetulemusi ja võimaldada algoritmi samm-sammuline töö. Algoritmil on kaks 16-bittist sisendit, mis on numbrid

kahendkoodis ja üks-kaks 16-bittist väljundit, mida kuvatakse 4-numbrilisel 7-segmendilisel ekraanil. Sisendite puhul ei ole oluline, kas on märgiga või märgita.

Teiseks tulemi osaks on kasutusjuhend, kus on ära kirjeldatud, mida saab lülititega teha, mis nuppu tuleb vajutada, et sisendeid sisse lugeda ja nii edasi. Samuti on ära kirjeldatud, kuidas õieti moodulit lisada ning mida teha, et saada see Digilent Basys 3 prototüüpimisplaadil tööle.

2 Põhisammud

Valitud oli algoritm arvude vahetu korrutamisest ühe bitti järgi ehk ühest arvust võetakse bitt ja selle alusel otsustatakse, mis algoritm peab edasi tegema.

2.1 Kodutöö sammud

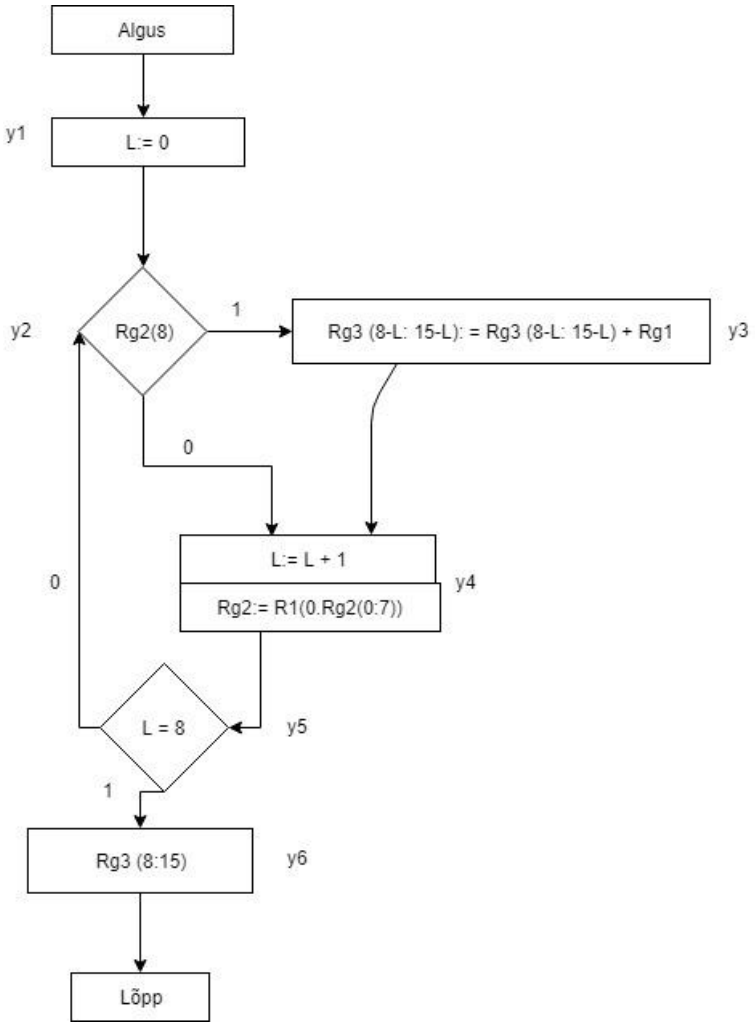
Tudengi poolt tehtavad sammud on järgmised:

1. Graafskeemi loomine. [5]
2. Operatsioonseadme skeemi loomine. [5]
3. Näidisalgoritmi modelleerimine valitud programmeerimise keeles ja testpink algoritmi testimiseks. Näiteks Java programmeerimise keel [12].
4. Näidisalgoritmi taktitõene modelleerimine VHDL'is (simuleeritav VHDL) ja testpink algoritmi testimiseks.
5. Basys 3 ja Vivado kasutamist õpetatakse praktikumide käigus. Tudengitele antakse lõputöö käigus loodud kasutusjuhend.
6. Tudengitele antakse valmis pakendatud Vivado projekt, kuhu lisatakse tudengi poolt loodud sünteesitavat algoritmi või hakkab tudeng Vivado projektis kirjutama sünteesitavat algoritmi.

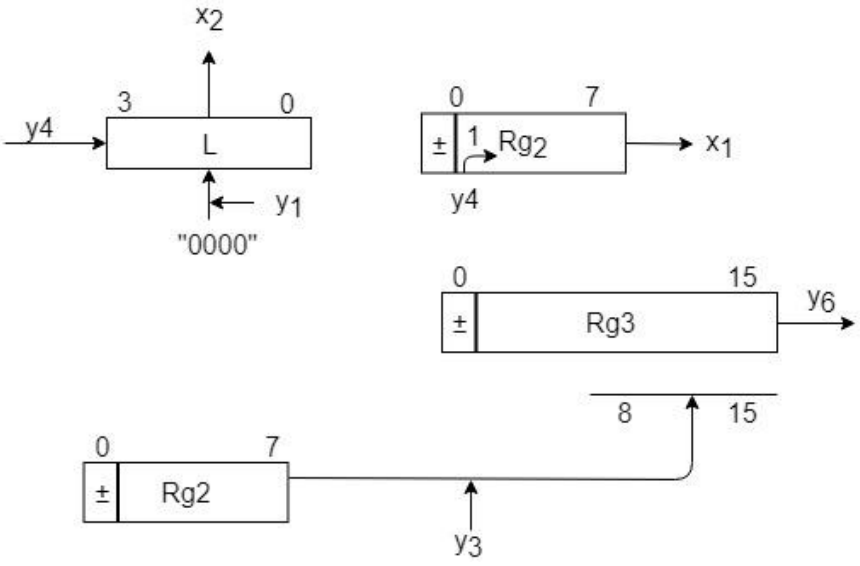
2.2 Lõputöö sammud

Lõputöö käigus olid tehtud kõik sammud, mida läbivad tudengid. Lõputöö käigus luuakse „Juhend protoüüpimiseks Basys 3 kasutades" ja pakendatud Vivado projekt, mida tudengid saavad kasutada peatükkis 2.1 punktides 5 ja 6.

1. Lõputöö käigus luuakse graafskeem ja operatsioonseadme skeem. [5]



Joonis 1 Graafskeem



Joonis 2 Operatsioonseadme skeem

2. Näidisalgoritmi samm-sammuline modelleerimine Javas ja VHDL’is. Samuti oli loodud testpink algoritmi testimiseks.

3. Basys3 ja Vivado kasutamise õppimine, samuti olemasolevate näidiste ja moodulite korduvkasutamise võimaluste uurimine. [1]

4. Basys3 sisendite ehk lülitite ja nuppude koos väljundite ehk LED indikaatorite ja 4-numbrilise 7-segmendilise ekraani kasutamise planeerimine.

5. Liidesmoodulite projekteerimine, modelleerimine ja süntees. Riistvarakeele kasutamise korral on klassikalised sammud järgmised [11]:

    a. koodi kirjutamine;

    b. modelleerimine;

    c. süntees;

    d. prototüüpimine;

6. Vivado projektina pakendamine.

7. Kasutusjuhendi koostamine tudengitele.

3 Java ja VHDL võrdlus

Lõputöö käigus selgus, et kirjutades VHDL'is ei tohi kasutada sama loogikat, mida kasutatakse Java programmeerimisel. Tuleb järgida mitmeid reegleid, kõige tähtsam on järgida VHDL'is taktsignaali. Samuti oli ka mitmeid sarnasusi struktuuri poolelt.

3.1 Erinevused

Kõige suuremaks erinevuseks tooks välja selle, et VHDL'is kõik protsessid ja tegevused on enamasti seotud taktsignaaliga. Taktsignaalil on kaks olekut *low* ehk '0' ja *high* ehk '1'. Järjest erinevaid protsesse ja tegevusi ei tehta enne kui taktsignaal on '1' [6].

Ühes ja samas protsessis ei saa omistada muutujale *X* väärtust ning seejärel omistada muutujale *Y* muutuja *X* väärtust [6]. Ei tohi mõelda sellest nagu Java programmeerimisest, kus on võimalik teostada kaks omistamist järjest ja nad toimivad korrektselt. Esimene omistamine toimub esimese taktiga ning teine omistamine teise taktiga. VHDL'is toimub omistamine nii öelda paralleelselt.

Vivado projekti tegemisel tuleb jälgida, et ei oleks *multi-driven* porte ehk topeltomistamist. Omistamine peaks toimuma ühes protsessis või tuleb luua vahemuutujaid ehk puhvreid.

3.2 Sarnasused

VHDL ja Java on sarnased struktuuri poolelt:

1. Javas on meetod ning VHDL'is võib välja tuua protsessi.
2. Javas on olemas klassid ning VHDL'is võib välja tuua *entity*.

Javas on võimalik kasutada loodud klasse teistes klassides. VHDL'is on võimalik kasutada loodud *entity*'d teistes *entity*'des deklareerides neid komponendina. Näiteks Javas on võimalik luua klass ning omistada sellele väärtused hiljem. Samuti VHDL'is peale komponendi deklareerimist mäppitakse väärtused ehk omistatakse.

4 Realiseerimise nõuded

4.1 Funktsionaalsed nõuded

Pakendatud Vivado projekti funktsionaalsed nõuded on järgmised:

1. Kasutajal on võimalik sisestada kaks numbrit kasutades lüliteid.
    a. Sisestatud numbrid on kahendkoodis.
    b. Numbrit loetakse sisse nuppu vajutamisel.
    c. Vasaku nuppu vajutamisel loetakse sisse esimene number.
    d. Keskmise nuppu vajutamisel loetakse sisse teine number.
2. Kasutajal on võimalik näha sisestatud numbrit 4-märgilisel 7-segmendilisel ekraanil.
    a. Nii kaua kuni hoitakse numbri sisestamise nuppu all, ekraanil on võimalik näha sisestatavat numbrit.
3. Kasutajal on võimalik kuvada vajalikud registrid 4-märgilisel 7-segmendilisel ekraanil.
4. Kasutajal on võimalik peale numbrite sisestamist käivitada algoritmi.
5. Kasutajal on võimalik käivitada algoritmi samm haaval.
    a. Vajutades paremat nuppu antakse algoritmile sisse sisestatud numbreid.
    b. Algoritmile antakse taktsignaal ainult nuppu vajutamisel.
6. Kasutajal on võimalik käivitada algoritmi otse.
    a. Vajutades ülemist nuppu antakse algoritmile sisse sisestatud numbreid.
    b. Algoritmile antakse prototüüpimisplaadi taktsignaali.
7. Kasutajal on võimalik näha algoritmi vahesammude tulemusi.

    a. Vahesammude tulemuste kuvamiseks kasutatakse 4-märgilist 7-segmendilist ekraani.

    b. Valikut teostakse vasakul pool asuvate kolme lüliti abil.

8. Kasutajal on võimalik nuppu vajutamisel saata *reset* signaal.

    a. *Reset* signaali saab sisestada vajutades alumist nuppu.

4.2 Mittefunktsionaalsed nõuded

Lõputöö mittefunktsionaalsed nõuded:

1. Prototüüpimisplaat peab olema Digilent Basys 3. [2]
2. Projekteerimisvahendiks on Xilinx Vivado 2018.3. [1]
3. Moodulid peaksid olema kirjutatud riistvara kirjelduskeeles VHDL.
4. Gitlab projekti litsenseerimine *OpenCore/OpenHardware* abil.

4.3 Projekteerimise vahendite valik

Vivado 2018.3 võimaldab kirjutada mooduleid kasutades VHDL'i või kasutades Verilog'i. Peamised põhjused, miks oli valitud VHDL:

1. Aines „Digitaalsüsteemid" kasutatakse terve aine raames VHDL'i.
2. VHDL on kasutatud kodutööde simuleerimiseks ehk on olemas kogemus VHDL'i kasutamises.
3. Juhendaja poolt määratud realiseerimise nõue.

Xilinx on loonud kaks tarkvara ISE ja Vivado, mõlemad on loodud projekti tegemiseks prototüüpimisplaatide jaoks ning projekti laadimiseks plaadile. ISE kasutatakse peamiselt vanemate prototüüpimisplaatide jaoks. Vivado on uuem tarkavara, mille jaoks oli väljatöötatud Basys 3, mis on uusim versioon Basys toodangust, mis sobib suurepäraselt FPGA õppimiseks tudengitele [3].

Sünteesitava algoritmi kuvamiseks oli valitud 16-nd süsteem, sest see on kompaktsem ning riistavara programmeerimisel tavapärasem. 16-nd süsteemiga on võimalik kuvada kõik numbreid, mis on 16-bitti suurused. Kasutades 10-nd süsteemi kuvamiseks tuleb kasutada 5-numbrilist ekraani või luua protsess, mis liigutaks numbreid koguaeg. See tõstaks lõputöö raskust, mis oleks juba magistritöö lähedane. Näiteks 2-nd süsteemis on „1111111111111111“, 10-nd süsteemis oleks „65535“ ja 16-nd süsteemis on „FFFF“.

Sünteesitavas algoritmis ja *main entity*'s realiseeritakse juhtautomaati. Juhtautomaati on kahte tüüpi Mealy juhtautomaat ja Moore juhtautomaat. Samuti on võimalus ka kombineerida kahte juhtautomaati. Mealy juhtautomaadis on asünkroonse tagasiside oht. Valituks osutus Moore juhtautomaat, sest ta on kindlam. [10]

Näidisalgoritmi kirjutamiseks oli kasutatud Java sellepärast, et sellega oli kõige rohkem kokku puutumisi terve ülikooli programmi vältel. Samuti olin õppinud Pythonit [13], kuid need teadmised olid hajunud ning Pythonis kirjutamine võtaks rohkem aega.

5 Prototüüpimine

5.1 Taust

Esialgu uurisin Basys 3 dokumentatsiooni ning olin vaatanud läbi mitmeid *tutorial'*e, põhilised allikad olid prototüüpimisplaadi veebilehel endal [4]. Digilent'il on olemas ka *youtube*'s õpetus video, kuid seal oli kasutatud Verilog'i, kuid põhilised sammud, mis on vajalikud Vivadoga töötamiseks on võimalik sealt saada.

Märtsis alustasin samasugust tööd või vähemalt sarnast sellele, mida hakkavad tudengid uue kodutöö raames tegema ning otsus langes läbida kõik need sammud, mis on vajalikud kodutöö valmimiseks. Peale seda oli alustatud graafskeemi ja operatsiooniseadme struktuurskeemi loomisega. Graafskeemi alusel oli kirjutatud Javas näidisalgoritm. Aja kokku hoidmiseks oli kasutatud õpetaja poolt 2009. aastal loodud näidis, mis sobis hästi, kuid tuli teha paar muudatust [7]. Eesmärk oli algoritmi luua maksimaalselt sarnane graafskeemil kirjeldatud tegevustele, et saaks võrrelda Java's ja VHDL'is kirjutatud näidisalgoritme. Loomulikult neid tehteid ja arvutusi on võimalik teha lihtsamalt, kuid see ei olnud eesmärgiks. Samuti näidisalgoritmidele oli kirjutatud testpink, mille abil anti algoritmile paaris väärtused ning kuvati nende arvude tulemused.

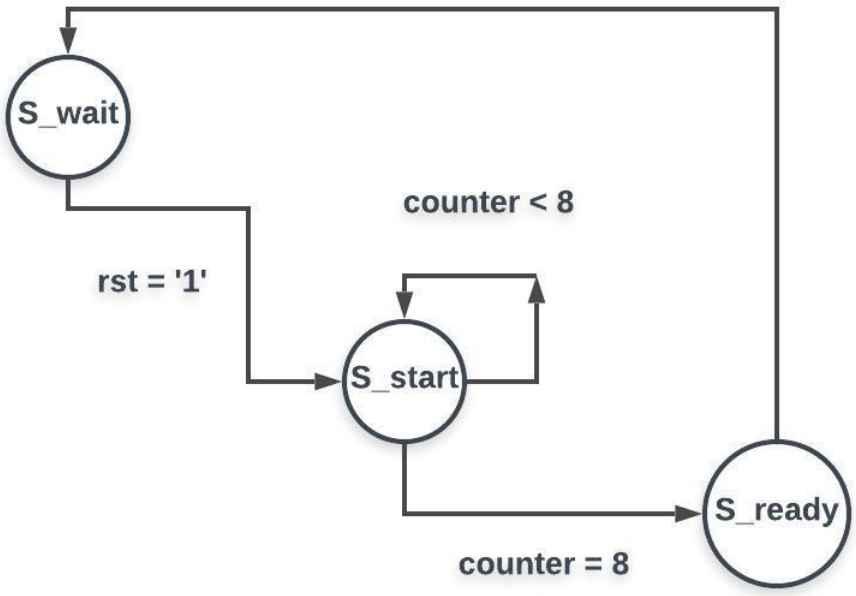
Peale seda hakkas kõige raskem osa, kus tuli otsida infot, kuidas käituvad nupud, lülitid, kuidas saab neid väärtusi sisse lugeda ja salvesatada, kuidas taktsignaal mõjutab protsesse ja andmete sisse lugemist ja salvestamist. Nuppude ja lülititega katsetamine otse plaadi peal ning takti mõju uurimine võttis umbes kaks nädalat aega. Uurida tuli plaadi peal, sest nuppu vajutuste ja lülitite signaali simuleerimine oleks vale, kuna plaadilt ei pruugi puhtad signaalid tulla.

Iga nädalaste juhendajaga kohtumiste käigus arutasime, mida võiks veel uurida, mida võiks vaadata. Samuti juhendaja poolt olid mõned seletused VHDL'i tööst. Märtsi lõpus oli juhendaja poolt antud Digilent Basys 3 prototüüpimisplaat, et oleks võimalik juba praktiliselt rakendada teadmisi, mida olin omandanud mitu kuud.

5.2 Sünteesitav algoritm

Graafskeemi alusel tuli teha simuleeritavat algoritmi. Simuleeritavaks algoritmiks sobis õpetaja poolt varem kirjutatud korrutamise algoritm [7]. Simuleeritavast algoritmist tuli teha sünteesitavat algoritmi. Selleks tuli lisada olemasolevale algoritmile olekud, mis muutuvad, kui takt on ’1’ ning tingimused on täidetud – seda nimetatakse juhtautomaadiks [10]. Joonisel 3 on näha olekutediagrammi, kus on näha olekute vahel liikumist ning tingimused olekute vahelise liikumiseks. Olekust *s_wait* on võimalik jõuda ainult olekusse *s_start*, kus on juba kaks võimalust. Niikaua, kui loendur on väiksem kui 8 viiakse *s_start* olekusse tagasi, kui loendur on võrdne 8-ga, siis on võimalik liikuda järgmisesse olekusse, milleks on *s_ready*.

S_wait olekus loetakse andmed sisse ja määratakse järgmine olek. *S_start* olekus toimub algoritmi kõige tähtsam osa ehk arvutamine. *S_ready* olekus määratkse algolek ehk *s_wait* ja vajalikud väärtused, mis annavad teada, et algoritm on töö lõpetanud. Olekutes ei salvestata andmeid otse registritesse, vaid salvestatakse puhvritesse. Eraldi seisvas protsessis toimub väärtuste puhvritest salvestamine registritesse.



Joonis 3 Sünteesitud algoritmi olekudiagramm

Lisaks juhtautomaadile tuli lisada protsessi, mis lülitite sisendi alusel otsustab, millist registrit kuvada. Kuna algoritmis on kasutatud 8-bittised numbrid, aga väljundis peab olema 16-bitti, siis tuleb lisada ette 8-bitti, iga bitti väärtus on ’0’.

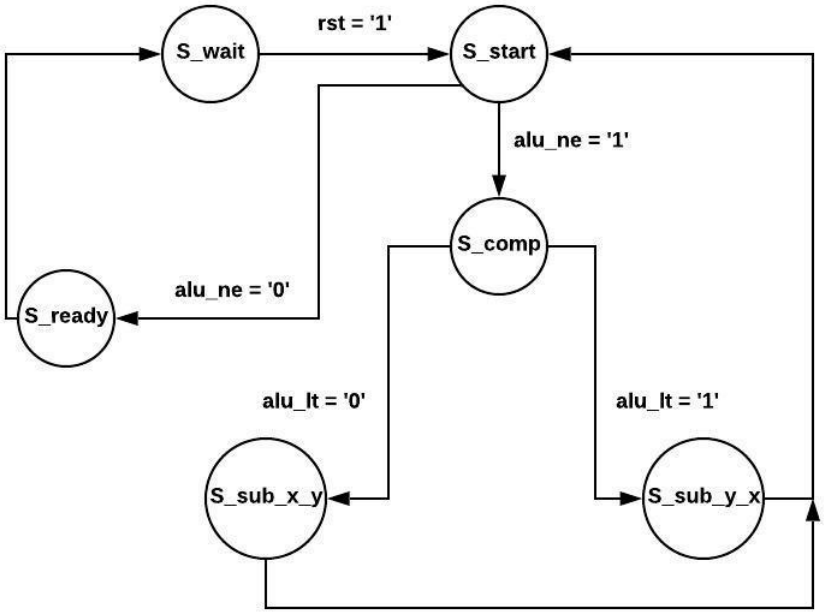
```
process(sel, ar, br, cr, counter)
begin
        case sel is
                when "00" => dbg <= z8 & counter;
                when "01" => dbg <= z8 & ar;
                when "10" => dbg <= z8 & br;
                when "11" => dbg <= cr;
                when others => dbg <= (others => '0');
        end case;
 end process;
```

Joonis 4 Protsess, kus otsustatakse, mis registrit algoritmist ekraanil kuvada

5.3 Juhendaja GCD algoritm

GCD algoritm [14] töötab senikaua, kuni kahe numbri kõige suurimat ühistegurit ei leita. *S_wait* olekust jõutakse *S_start* olekusse siis, kui algoritm saab *reset* signaali. S_start olekus on numbrid puhvritesse sisse loetud. *S_start* olekust liigutakse *S_comp* olekusse niikaua, kuni suurimat ühistegurit ei leita. Selleks, et liikuda *S_comp* olekust teostatakse paar võrdlust, mille alusel algoritm otsustab, kas lahutada esimesest numbrist teist või vastupidi. See on lihtne võrdlus, kui esimene number suurem kui teine, siis liigutakse *S_sub_x_y* olekusse. *S_sub_y_x* olekusse liigutakse siis, kui teine number on suurem kui esimene. *S_ready* olekusse jõuab siis, kui esimene ja teine number võrdne.

Algoritmile antakse sisenditeks kaks numbrit näiteks 2 ja 9. Teostatakse võrdlus, ning liigutakse *S_sub_y_x* olekusse ning peale seda teostatakse arvutus 9-2=7 ja määratakse teiseks numbriks 7. See toimub niikaua, kuni on jõutud arvudega nii kaugele, et toimub tehe 3-2= 1. Nüüd on esimene number suurem kui teine ning *S_comp* olekust liigutakse *S_sub_x_y* olekusse ning teostatakse vastupidine tehte ehk lahutatakse esimesest numbrist teist numbrit 2-1=1. Jõutakse *S_start* olekusse, kus selgub, et esimene number on võrdne teise numbriga. Algoritm lõpetab enda töö, sest ühistegur on leitud ja jõuab *S_ready* olekusse ning väljastab lõpptulemuse.

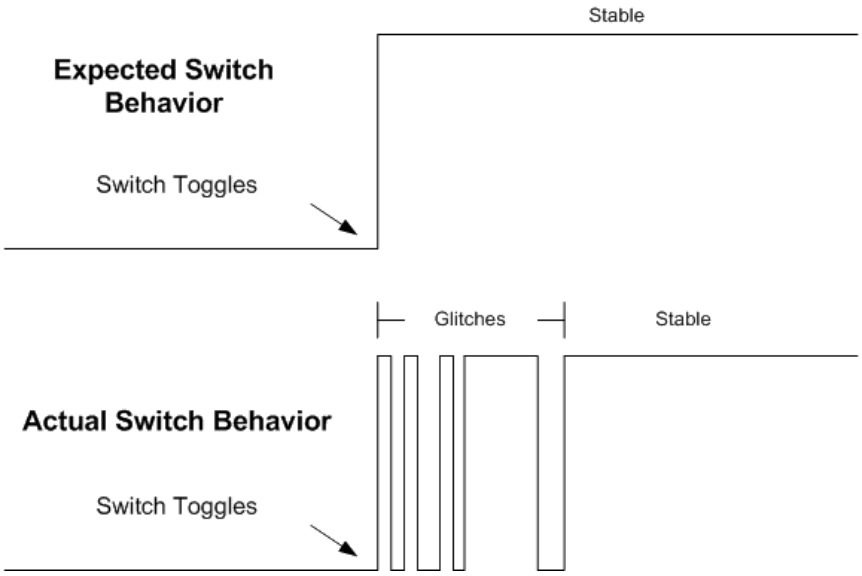


Joonis 5 GCD algoritmi olekudiagramm

Õpetaja algoritm oli samuti täiendatud joonis 4 koodiga, et saaks kuvada algoritmi siseseid registreid.

5.4 Värelusvastane lülitus (*debouncer*)

Inimene ei taju seda, et toimub nii öelda nuppu mitme kordset vajutamist, sest see toimub riistvara tasemel. Sel ajal, kui kontaktid on läheduses, siis võib tekkida mitu sisend signaale ühe asemel. Mõnel prototüüpimisplaadil on sisseehitatud *debouncer*, mis viib signaali soovitud olekuni ehk puhta signaalini [8]. Joonisel 6 alumises osas on näha, kuidas tegelikult signaal käitub. Nuppu vajutusel toimuvad *glitches* ehk üleliigsed signaali võnked, mida meie ei soovi enda tarkvaras saada. Selle vältimiseks oli kasutatud *debouncer*, sinna sisse läheb ebasobilik signaal ning väljundiks on puhas signaal.



Joonis 6 Signaali käitumine nuppu vajutusel [9].

Vivado projektis *debouncer*'is on kasutatud loendur, mis loendab 250 000-ni, mis on ¼ prototüüpimisplaadi sagedusest. Loendurile liidetakse 1 iga taktiga, kui on täidetud kaks tingimust. Loendur peab olema väiksem 250 000-st ning signaal peab olema '1' ehk nupp peab olema vajutatud. Kui loendur on 250 000, siis tagastatakse nuppu vajutuse signaal ning nullitakse loendur, kui signaal ei ole '1' ja loendur on väiksem 250 000-st, siis nullitakse ära loendur.

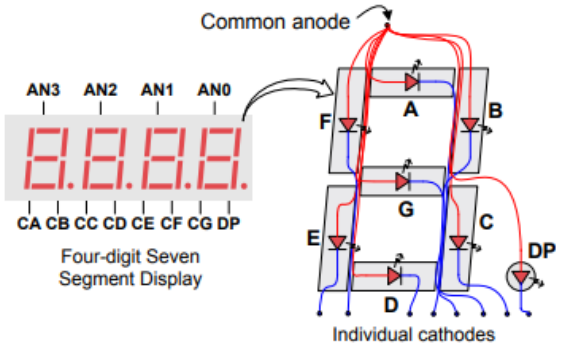
```
if rising_edge(clk) then
    if(in_button_signal/=state_register and count_register<c_DEBOUNCE_LIMIT) then
       count_register <= count_register + 1;
    elsif count_register = c_DEBOUNCE_LIMIT then
      state_register <= in_button_signal;
      count_register <= 0;
    else
      count_register <= 0;
    end if;
end if;
```

Joonis 7 Debounceri kood [9].

5.5 4-numbriline 7-segmendiline ekraan

4-numbrilise 7-segmendilise ekraanil kuvatakse 16-nd süsteemis numbreid. Numbriteks on vahetulemused, sisestatud numbrid ja vastused. Lülititega on võimalik valida, kas kuvada vastuseid või sisestatud numbreid või lülitada sisse *debug* režiim, millega saab kuvada registreid/vahetulemusi algoritmist.

Number on 16-bittine. Selleks, et määrata, mis numbrit tuleb kuvada on vaja 4-bitti ehk numbri saab jagada 4 võrdseks osaks 4-bitti kaupa. Näiteks number on „1000000100100100“, seda võib jagada neljaks „1000“, „0001“, „0010“, „0100“. Vastavalt neljale osale, peab ekraanil olema kuvatud ’8’, ’1’, ’2’ ja 4. Joonisel 9 välja toodud protsessi alusel määratakse segmendile 7-bitti. Iga bitile vastab üks segment, kui biti väärtus on ’0’, siis valgustatakse segmenti, kui biti väärtus on ’1’, siis seda segmenti ei valgustata. Joonisel 8 on antud segmentidele tähed. Näiteks numbri ’0’ kuvamiseks peab muutujale *seg* määrama „1000000“. Kõik segmendid peale ’G’ peavad olema valgustatud. Selle alusel võib määrata malli „GFEDCBA“. [2]



Joonis 8 Ühisanoodiga 7-segmendiline indikaator (numbrikoht) [2].

```
process(digit)
begin
    case digit is
        when "0000" => seg <= "1000000"; -- "0"
        when "0001" => seg <= "1111001"; -- "1"
        when "0010" => seg <= "0100100"; -- "2"
        when "0011" => seg <= "0110000"; -- "3"
        when "0100" => seg <= "0011001"; -- "4"
        when "0101" => seg <= "0010010"; -- "5"
        when "0110" => seg <= "0000010"; -- "6"
        when "0111" => seg <= "1111000"; -- "7"
        when "1000" => seg <= "0000000"; -- "8"
        when "1001" => seg <= "0010000"; -- "9"
        when "1010" => seg <= "0100000"; -- "a"
        when "1011" => seg <= "0000011"; -- "b"
        when "1100" => seg <= "1000110"; -- "C"
        when "1101" => seg <= "0100001"; -- "d"
        when "1110" => seg <= "0000110"; -- "E"
        when "1111" => seg <= "0001110"; -- "F"
    end case;
end process;
```

Joonis 9 Segmentide määramine [15]

Numbrite kuvamise raskeks teeb asjaolu, et ei saa igale numbrikohale määrata kindlat numbrit, mida seal kuvatakse. Segmendile määratud 7-bitti kuvatakse kõikidel numbrikohtadel. Abiks on prototüüpimisplaadi kiire sagedus 100 Mhz. Selleks, et ära kasutada taktisagedust oli tehtud loendur, mis liidab 1 juurde siis, kui takt on ’1’. Nelja taktiga tehakse tsükkel ära. Loendur on algul „00“ => „01“ => „10“ => „11“ ja jõuab uuesti „00“-ni. Loenduri alusel otsustatakse, millist numbrikohta kuvada. Joonisel 8 on näha, et numbrikohad on ära märgistatud AN3 kuni AN0. Nelja bitiga määratakse ära, millised numbrikohad on valgustatud ja millised mitte, kui bitt on ’1’, siis seda kohta ei valgustata ja vastupidi. Kuna tegu on suure sagedusega, siis silmaga pole võimalik märgata, et mingi numbrikoht ei ole valgustatud. Joonis 10 on toodud 7-segmendilise ekraani koodi osa, kus teostatakse numbrikohtade korda mööda valgustamine ja samuti selles osas määratakse ära millisel numbrikohal, mis väärtust tuleb kuvada.

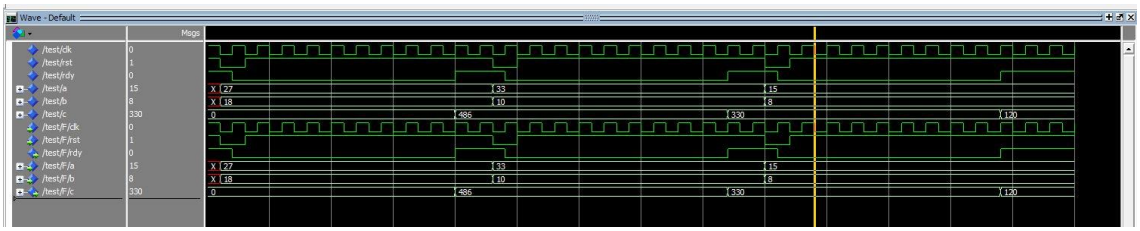
```
process(led_activation_order)
begin
   case led_activation_order is
            when "00" =>
               an <= "0111";
               -- activate LED1 and Deactivate LED2, LED3, LED4
               digit <= displayed_number(15 downto 12);
               -- the first hex digit of the 16-bit number
            when "01" =>
               an <= "1011";
               -- activate LED2 and Deactivate LED1, LED3, LED4
               digit <= displayed_number(11 downto 8);
               -- the second hex digit of the 16-bit number
            when "10" =>
               an <= "1101";
               -- activate LED3 and Deactivate LED2, LED1, LED4
               digit <= displayed_number(7 downto 4);
               -- the third hex digit of the 16-bit number
            when "11" =>
               an <= "1110";
               -- activate LED4 and Deactivate LED2, LED3, LED1
               digit <= displayed_number(3 downto 0);
               -- the fourth hex digit of the 16-bit number
   end case;
end process;
```

Joonis 10 Numbrikohtade valgustamise protsess [15]

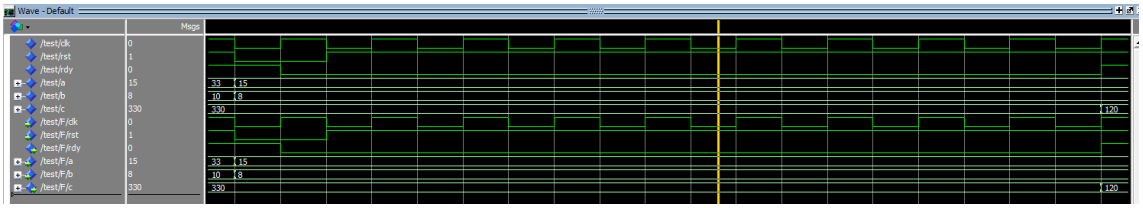
5.6 Testimine

Testimine oli teostatud reaalajas. Peale muudatusi koodis oli teostatud bitstreami genereerimine ning laetud Basys 3 peale. Reaalajas testimine oli valitud ainult sellepärast, et ei ole otstarbekas simuleerida või ette anda teatuid signaale. Programsel tasemel võivad signaalid käituda erinevalt sellest, kuidas nad käituvad riistvara tasemel. Peatükkis 5.2 on ära kirjeldatud nuppu käitumine, kui nuppu simuleerimiseks annaks signaali, siis see signaal ei vastaks reaalsuses tõele.

Algoritmi töö testimiseks oli kirjutatud testpink, mis annab algoritmile sisse kahe numbri kaupa ning kuvatakse kasutajale kõikide ette antud numbri paaride vastused. Sarnaselt toimib ka simuleeritava VHDL'i testpink. Joonis 11 on näha VHDL simulatsiooni tulemus, kus on loetavuse mõttes jäetud numbrid 10-nd süsteemis. Algoritm töötab korrektselt, sest näiteks 33 x 10 = 330. Esimesel taktil on punane joon, kuna esimesel taktil ei ole antud mitte mingeid väärtusi, mida on võimalik algoritmist ka jälgida.



Joonis 11 ModelSIM'is simuleeritud näidisalgoritmi signaalid



Joonis 12 Kolmanda numbripaari suurendatud signaalid

5.7 Vivado projekt

Vivado projekti kõige tähtsam osa on *main.vhd* failis, mis juhib sisend ja väljund signaalide liikumist *entity*'te vahel ning väljundeid nendest. *Main entity*'s on ära defineeritud komponendid, mida võib vaadelda kui ühe klassi välja kutsumist teises klassis Java programmeerimise keeles. Samuti komponentide määramiseks tuleb pordid

ära mäppida ehk määrata signaalid, mis lähevad komponendi sisse ning määrata registrid, mis võtavad signaalid vastu, mis tulevad komponentidest.

Joonise 13 on ekraani komponendi näidis ning joonis 14 on sama komponendi portide mäppimine. Joonisel 14 on näha, et nimetused võivad olla samasugused nii komponendis, kui ka signaalide registrite nimed.

```
component seven_segment_display
port(
   clk : in STD_LOGIC;
   displayed_number: in STD_LOGIC_VECTOR(15 downto 0);
   an : out STD_LOGIC_VECTOR(3 downto 0);
   seg : out STD_LOGIC_VECTOR(6 downto 0)
);
end component;
```

Joonis 13 4-numbrilise 7-segmendilise ekraani komponent

```
sev_segment : seven_segment_display
port map (
   clk => clk,
   displayed_number => displayed_number,
   an => an,
   seg => seg
);
```

Joonis 14 4-numbrilise 7-segmendilise ekraani komponendi mäppimine

Joonisel 15 protsessis loetakse lülitite signaalid registritesse nuppu vajutusel. Vasaku nuppu vajutusel loetakse sisse esimese numbri registrisse ning keskmise nuppu vajutusel loetakse sisse teisse registrisse.

```
process(btnc, btnl, clk)
begin
   if rising_edge(clk) then
      if btnl = BTN_ACTIVE then
         first_number <= sw;

      elsif btnc = BTN_ACTIVE then
         second_number <= sw;

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

Joonis 15 Numbrite sisse lugemise protsess

Joonisel 16 on protsess, mille abil otsutatakse, millist registrit tuleks kuvada ekraanil. Vasaku nuppu vajutusel kuvatakse sisse loetud numbrit. Keskmise nuppu vajutusel kuvatakse sisse loetud numbrit. Kui neid nuppe ei vajutata, siis kuvatavate registreid vastavalt kolme lüliti asendist. Esimese lüliti sisse lülitamisel toimub algoritmi debugimine, kuna sellel ajal kuvatakse algoritmi siseseid registreid, mida tudeng võib ise määrata. Saab vaadata, mis tulemused on sinna salvestatud ning võrrelda, kas nad on soovitud väärtusega. Tabel 1 on välja toodud lülitite asendid ning mida kuvatakse.

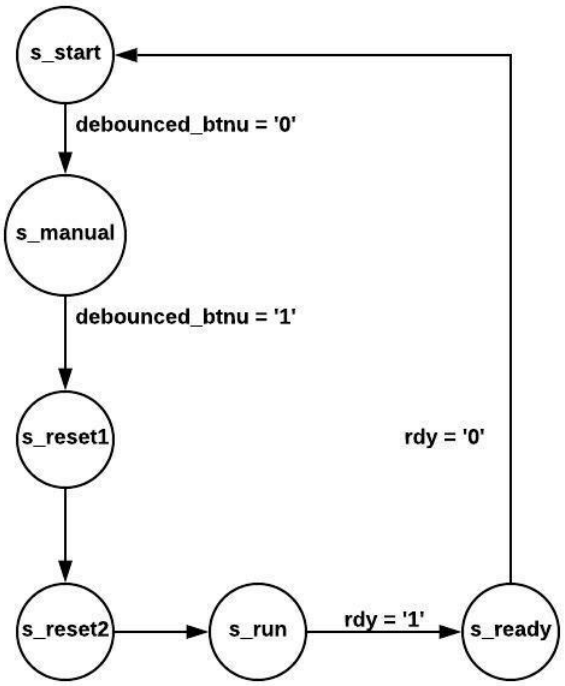
Tabel 1 Kuvatavate registrite valik vastavalt kolme lülitile '0' - lüliti väljas, '1' - lüliti sees

Vasakult esimene	Vasakult teine	Vasakult kolmas	Kuvatav register
0	0	0	Esimene number
0	0	1	Teine number
0	1	0	Esimene tulemus
0	1	1	Teine tulemus
1	0	0	Loenduri väärtus algoritmis
1	0	1	Esimene number algoritmis
1	1	0	Teine number algoritmis
1	1	1	Tulemus algoritmis

```
process(sw, dbg, result1, result2, first_number, second_number, btnl, btnc)
   begin
      if btnl = BTN_ACTIVE then
        displayed_number <= first_number;
      elsif btnc = BTN_ACTIVE then
        displayed_number <= second_number;
      else
         if sw(15) = '1' then
            displayed_number <= dbg;
         else
            case sw(14 downto 13) is
               when "00" => displayed_number <= first_number;
               when "01" => displayed_number <= second_number;
               when "10" => displayed_number <= result1;
               when "11" => displayed_number <= result2;
               when others => displayed_number <=(others => '0');
            end case;
         end if;
      end if;
   end process;
```

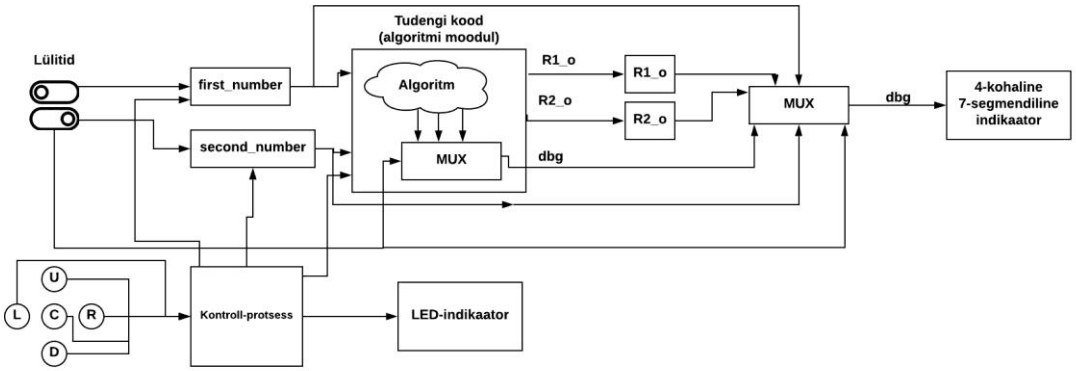
Joonis 16 Lülitite järgi numbrite kuvamine ekraanil

Üks protsessidest on juhtautomaat, sest ülemise nuppu vajutusel peab algoritm automaatselt lõpuni jõudma, kuid probleem oli selles, et kiire sageduse pärast läbiti algoritmi mitu korda. Ülemise nuppu lahti laskmisel peatus algoritm suvalises kohas. S_start olekust liigutakse s_manual olekusse siis, kui ei vajutata ülemist nuppu. Ülemise nuppu vajutusega liigutakse s_reset1 olekusse, kus määratakse reset ja sealt s_reset2 olekusse, kus määratakse topelt resetile väärtuseks ’1’. Peale seda liigutakse olekusse s_run ehk algoritmi jooksutatakse ja oodatakse niikaua, kuni algoritm annab *rdy* signaaliga teada, et algoritm on oma töö lõpetanud peale mille jääb algoritm oote olekusse niikaua, kuni ülemine nupp ei ole lahti lastud ning alumine nupp pole vajutatud. Hoides alumist nuppu ja vajutades paremat nuppu on võimalik aktiveerida samm-sammulist režiimi. Lastes lahti alumist nuppu ning vajutades paremale nuppule on võimalik algoritmi arvutusi vaadelda nii öelda takti kaupa.



Joonis 17 Vivado projekti juhtautomaadi olekudiagramm

Joonisel 18 on esitatud üldine struktuur skeem, mis kaustab eelpool kirjeldatud mooduleid. Kontroll-protsessi kuuluvad nuppude *debouncer*’id ja algoritmi juhtautomaat. MUX on protsess, mis kuvab valitud registri sisu 4-kohalisele 7-segmendilisele indikaatorile.



Joonis 18 Struktuurskeem

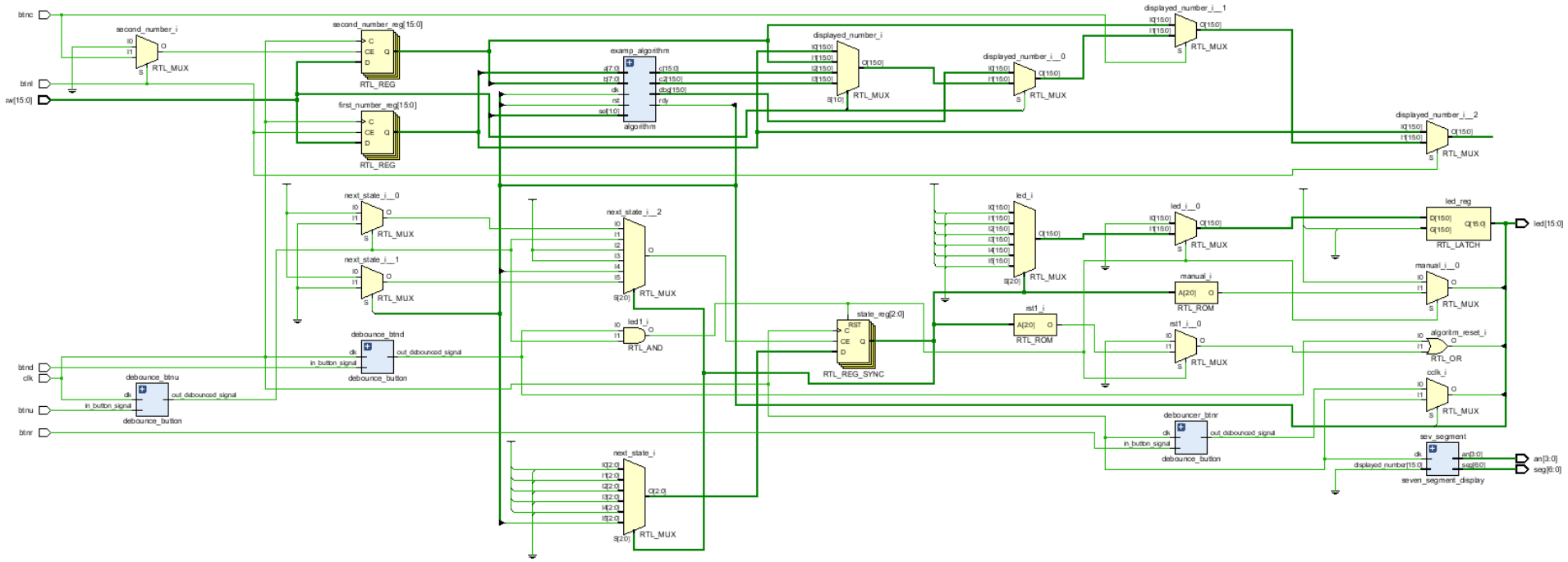
Sünteesitud algoritmi paigutust FPGA'l on võimalik näha LISA 1 joonis 25. Joonisel 25 on näha, et FPGA kasutus on tühine, millest võib järeldada, et võib palju mahukamaid ja keerukamaid algoritme luua. Samuti on võimalik sünteesitud projekti raportidest täpsemalt vaadata FPGA kasutusest. Tabel 2 on näha raportite tulemused. *Latch'*ide ehk lukk-trigerite arv on 0, mis viitab korrektsele sünkroonsele disainile, mis viitab disaini puhtusele. Kombinatoorse loogika moodulites on kõik tingimuslikud harud kaetud ja ei teki ebavajalikke lukk-registreid/lukk-trigereid, mis on ajastuse analüüsil suureks mureks. LUT on FPGA sisesed tõeväärtustabelid, mis on genereeritud automaatselt. Tõeväärtustabel määratleb ära, kuidas kombinatoorne loogika projektis käitub. Samuti väljundeid iga sisse antud sisendi kohta. *Flip-flop'*e võib vaadelda kui hoidlat. Põhilisteks FPGA plokkideks on *flip-flop*'id ning *look-up-tabel*'id. [16] Viimane tabeli argument on FPGA sisemine pisi-moodul, mis on peamine ehitusplokk. *Slice* jaguneb sisemiselt väiksemateks osadeks nagu LUT, *flip-flop* ja nii edasi.

Tabel 2 Raportide tulemus

Argument	Kasutusel (tk)	Kokku (tk)	Protsent (%)
LUT as logic	136	20800	0.65
Registers as Flip Flop	164	41600	0.39
Registers as Latch	0	41600	0
Slice (LUT & FF)	74	8150	0.91

Samuti on võimalik näha sünteesitud disaini LISA 1 asuval joonisel 24. Samuti on võimalik näha joonisel 19, kust saadakse signaalid sisse ning kuhu nad lõpuks jõuavad terve Vivado projekti vältel. Vasakul pool antakse sisendid ja paremal pool on näha,

mis on väljunditeks. Sisenditeks on nupud, lülitid ja *clk*. Väljunditeks on ekraan ja LED-indikaatorid.



Joonis 19 Vivado poolt genereeritud registersiirete taseme skeem

6 Kokkuvõte

Töö esimeseks eesmärgiks oli luua pakendatud Vivado projekt, kuhu tuli lisada sünteesitav näidisalgoritm. Teiseks eesmärgiks oli luua Basys 3 prototüüpimisjuhend tudengitele, kus on kõik sammud ära kirjeldatud alustades projekti kloonimisest ning lõpetades koodi laadimisega Basys 3 prototüüpimisplaadile.

Esialgu oli loodud „iapb“ repositoorium Gitlabis. Seejärel oli loodud projekt kasutades Vivado 2018.3. Samuti oli loodud Gitlabis eraldi repositoorium Java algoritmi jaoks, kuna see on erladi seisev ülesanne.

Vivado projekti kõrgeima taseme üksusesse (*main entity*) oli lisatud kaks komponenti. Esimene lisatud komponent võimaldab 16-bittist numbri kuvamist ekraanil. Teine lisatud komponent eemaldab signaali värelemist ehk nuppu vajutamise signaal on puhas. Seejärel oli lisatud 4 protsessi, mis aitavad lülitite signaalid sisse lugeda nuppu vajutusel, aitavad kuvada erinevaid registreid sõltuvalt lülitite asendist ning lisatud oli juhtautomaat, mis juhib algoritmi automaatset tööd.

Näidisalgoritmi jaoks oli loodud graafskeem ja operatsiooniseadme skeem, kui graafskeem oli loodud, siis selle alusel oli kirjutatud Javas algoritm ja testpink algoritmile. Graafskeemi alusel oli valitud sobilik simuleeritav korrutamisealgoritm ja testpingi näide, mis oli kirjutatud juhendaja poolt, et hoida aega kokku. Simuleeritavast algoritmist oli loodud sünteesitav algoritm. Samuti oli lisatud juhendaja poolt loodud GCD sünteesitav algoritm, mis oli täiendatud mitme argumendiga ja protsessiga, et algoritm saaks korrektsed signaalid Vivado projekti sees ja väljund signaalid jõuaksid õigesse registrisse.

Viimaseks sammuks oli juhendi koostamine. Juhendis on ära kirjeldatud, mis lülititega, millist registrit saab kuvada, mis nupuga saab sisse lugeda esimest ja teist numbrit, värskendada, algoritmi käivitada automaatselt, algoritmile ise takti sisse anda. Juhendis on ära kirjeldatud protsess, et projekt enda vivados tööle saada.

Java algoritmi repositoorium on saadaval TTÜ Gitlabis aadressil https://gitlab.cs.ttu.ee/arjusk/java_aritmeetika_algoritmi_realiseerimine. Simuleeritav VHDL'is kirjutatud näidisalgoritm koos testpingiga koos pakendatud Vivado projektiga on saadaval samuti TTÜ Gitlabis aadressil https://gitlab.cs.ttu.ee/arjusk/iapb.

Kasutatud kirjandus

[1] „Xilinx Vivado Design suite – Getting started,“ [Võrgumaterjal]. Saadaval: https://www.digikey.com/eewiki/display/LOGIC/Xilinx+Vivado+Design+Suite+-+Getting+Started (09.05.2019)

[2] „Basys 3™ FPGA Board Reference Manual,“ [Võrgumaterjal]. Saadaval: https://reference.digilentinc.com/_media/reference/programmable-logic/basys-3/basys3_rm.pdf?_ga=2.170500128.1287486364.1557321527-1378189227.1553101689 (09.05.2019)

[3] „Basys 3 Artix-7 FPGA Trainer Board: Recommended for Introductory Users,“ [Võrgumaterjal]. Saadaval: https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/ (09.05.2019)

[4] „Basys 3 Resource Center,“ [Võrgumaterjal]. Saadaval: https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/ (09.05.2019)

[5] „Korrutusalgoritmid,“ M. Kruus, [Võrgumaterjal]. Saadaval: http://mini.pld.ttu.ee/~lrv/IAS0150/Korrutamine.pdf (09.05.2019)

[6] „VHDL 3 – Sequintial Logic Circuits,“ [Võrgumaterjal]. Saadaval: http://www.eng.auburn.edu/~nelson/courses/elec4200/Slides/VHDL%203%20Sequential.pdf (09.05.2019)

[7] „Multiplier“, P. Ellervee, [Võrgumaterjal]. Saadaval: http://mini.pld.ttu.ee/~lrv/IAY0150/multiplier/multiplier1.vhd (09.05.2019)

[8] „Pushbutton DeBounce circuit in VHDL,“ [Võrgumaterjal]. Saadaval: http://vhdlguru.blogspot.com/2017/09/pushbutton-debounce-circuit-in-vhdl.html (16.05.2019)

[9] „Go Board Project – Debounce A Switch,“ [Võrgumaterjal]. Saadaval: https://www.nandland.com/goboard/debounce-switch-project.html (16.05.2019)

[10] „Automaadi süntees,“ [Võrgumaterjal]. Saadaval: http://www.tud.ttu.ee/im/Peeter.Ellervee/IAS0150/exercise-5.pdf (19.05.2019)

[11] D. Jansen, „Design using Standart Description Languages,“ *The Electronic Design Automation Handbook*, Boston, Kluwer Academic Publishers, 2003, pp.86-145.

[12] „The Java® Language Specification,“ J. Gosling, B.Joy, G. Steele, G. Bracha, A.Buckley, [Võrgumaterjal]. Saadaval: https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf (19.05.2019)

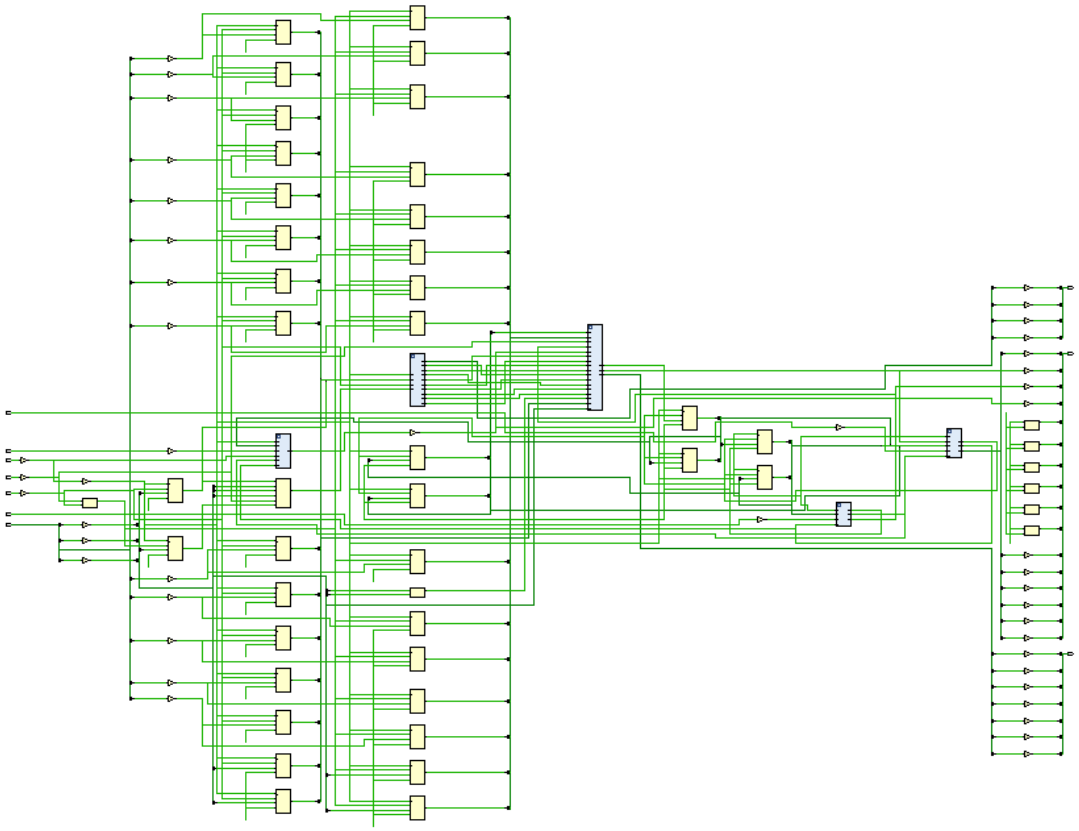
[13] [Võrgumaterjal]. Saadaval: https://www.python.org/ Saadaval: (20.05.2019)

[14] P. Ellervee, [Võrgumaterjal]. Saadaval: http://mini.pld.ttu.ee/~lrv/gcd/gcd-rtl1.vhdl (20.05.2019)

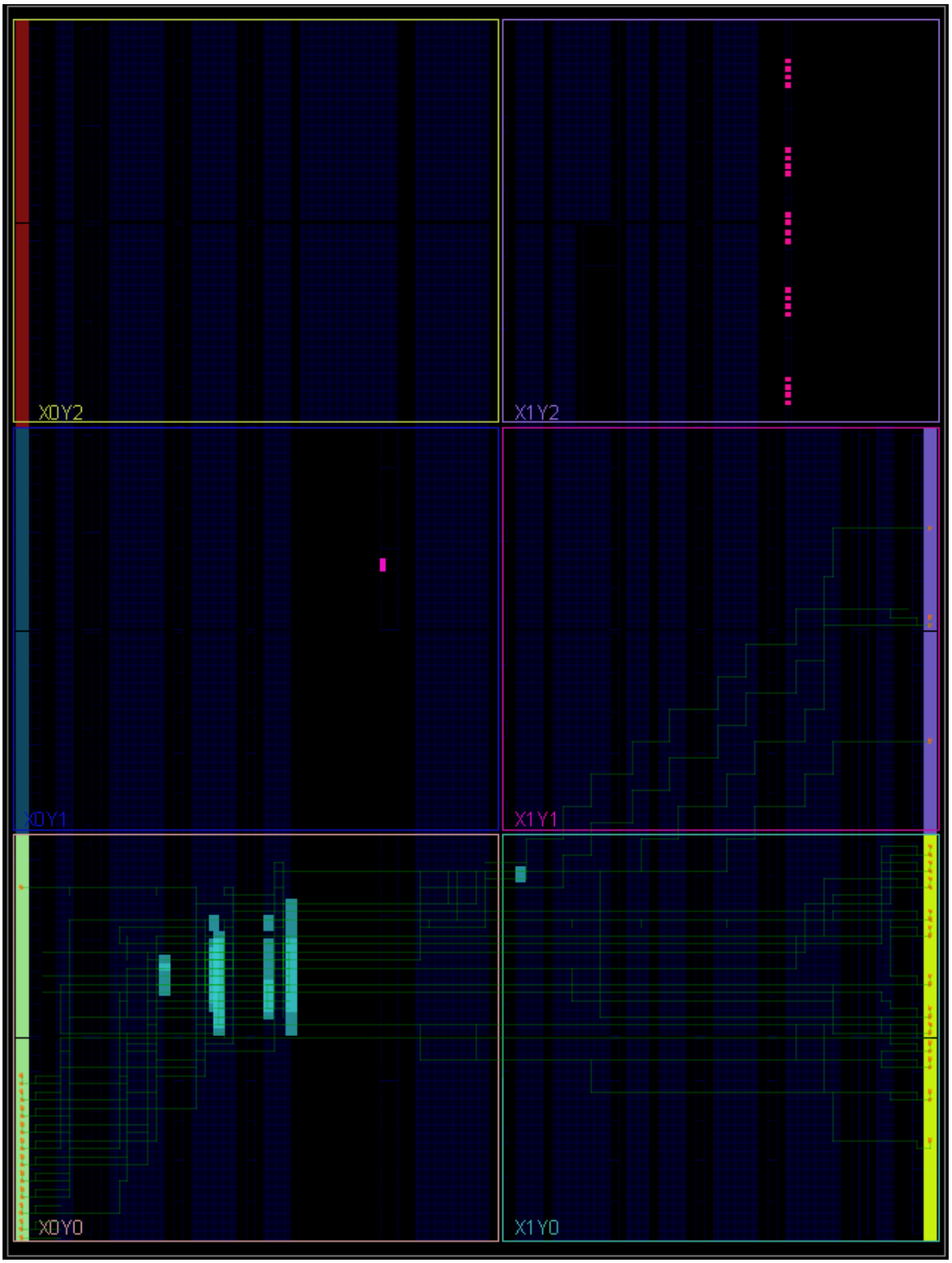
[15] „VHDL code for Seven-Segment Display on Basys 3 FPGA,“ [Võrgumaterjal]. Saadaval: https://www.fpga4student.com/2017/09/vhdl-code-for-seven-segment-display.html (20.05.2019)

[16] „Getting Started with FPGAs: Lookup Tables and Flip-Flops, “ [Võrgumaterjal]. Saadaval: https://www.allaboutcircuits.com/technical-articles/getting-started-with-fpgas-look-up-tables-and-flip-flops/ (20.05.2019)

LISA 1 – Skeemid

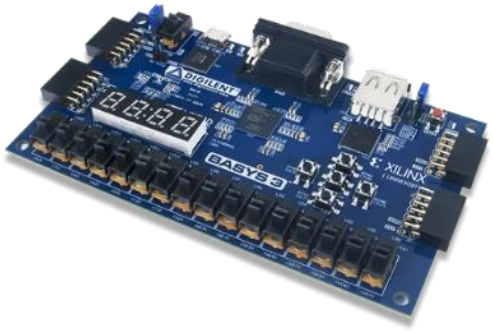


Joonis 20 Sünteesitud disaini skeem



Joonis 21 FPGA kasutamine

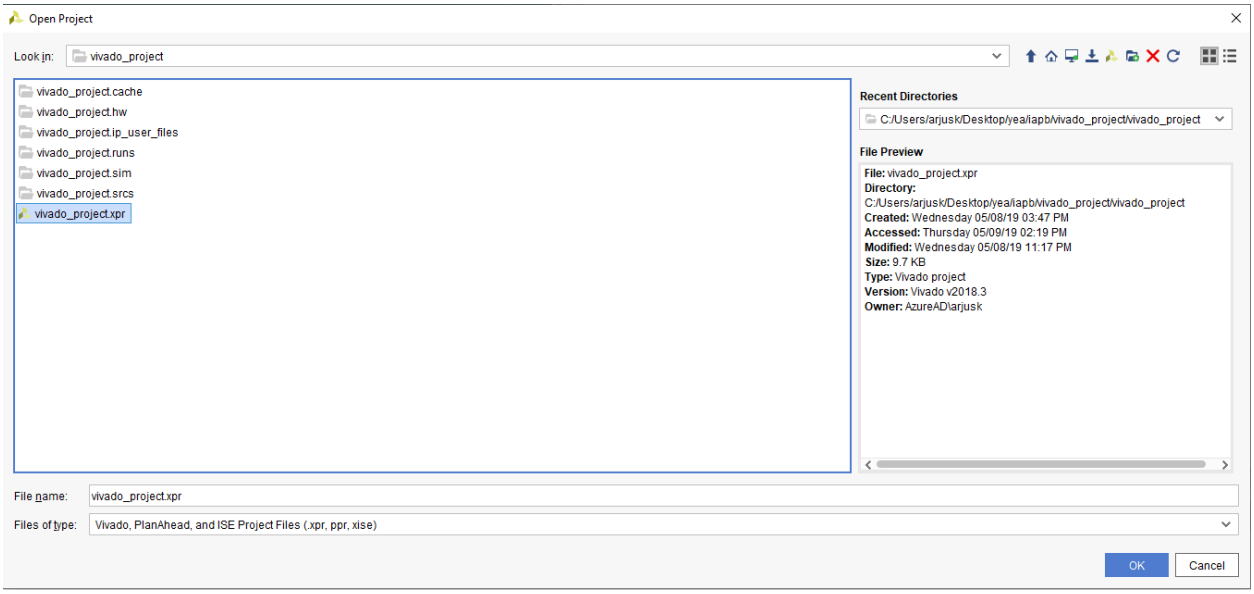
LISA 2 - Juhend prototüüpimiseks Basys 3 kasutades



Esialgu tuleb laadida enda arvutisse Vivado.
NB! Tasuta versioon on „WebPACK and Editions“ versioon.
Vivado koduleheküljelt tuleb laadida fail, kuid enne tuleb end seal ära registreerida:
https://www.xilinx.com/support/download.html

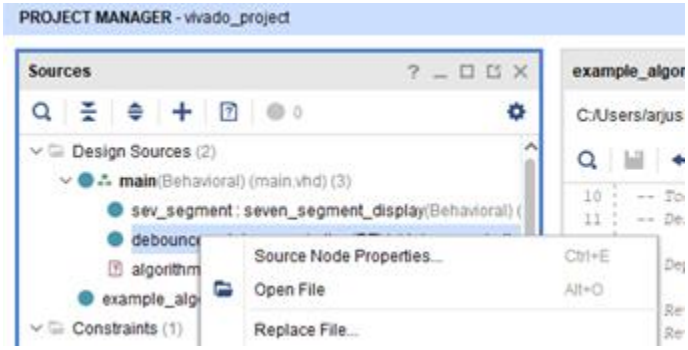
Seejärel tuleb kloonida projekti gitlabist enda arvutisse kasutades linki:
https://gitlab.cs.ttu.ee/arjusk/iapb.git

Peale seda, kui tarkvara Vivado on avatud(võib võtta mingit aega) tuleb vajutada „Open project“ ning liikuda kausta, kuhu on kloonitud projekt. *Vivado_project* kaustas tuleb leida fail „vivado_project.xpr“ ning vajutada sellele ja vajutada nuppu „OK“.



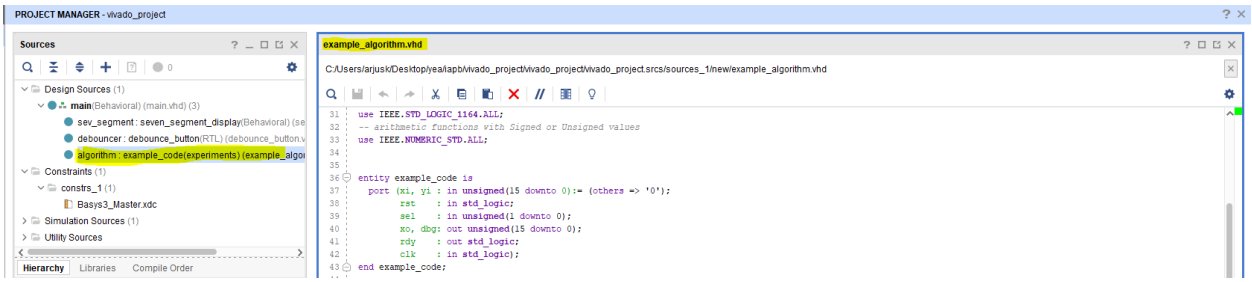
Laadimisel võib tekkida olukord, kus on punane hüüumärk. Sellest võib lihtsalt jagu saada. Tuleb teostada 2.a ja 2.b punktis olevad tegevused. Seejärel tuleb valida „Add files“ ning see

suunab automaatselt kausta, kus asuvad kõik vhd failid ning tuleb valida need failid, mille kõrval on hüüumärk. Peale seda toimingut vajalikud failid laetakse projekti.



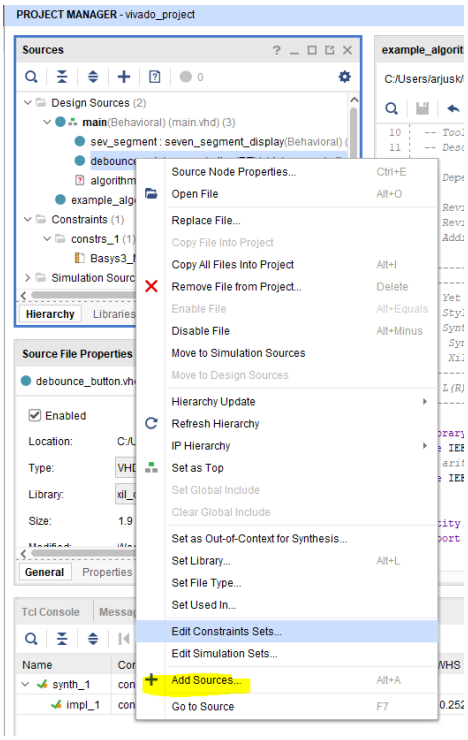
Algoritmi projekti lisamiseks on kaks võimalust:

1. Kõige lihtsam on olemasoleva algoritmi asendamine. Tuleb kopeerida kogu algoritmi osa *example_algorithm* faili.

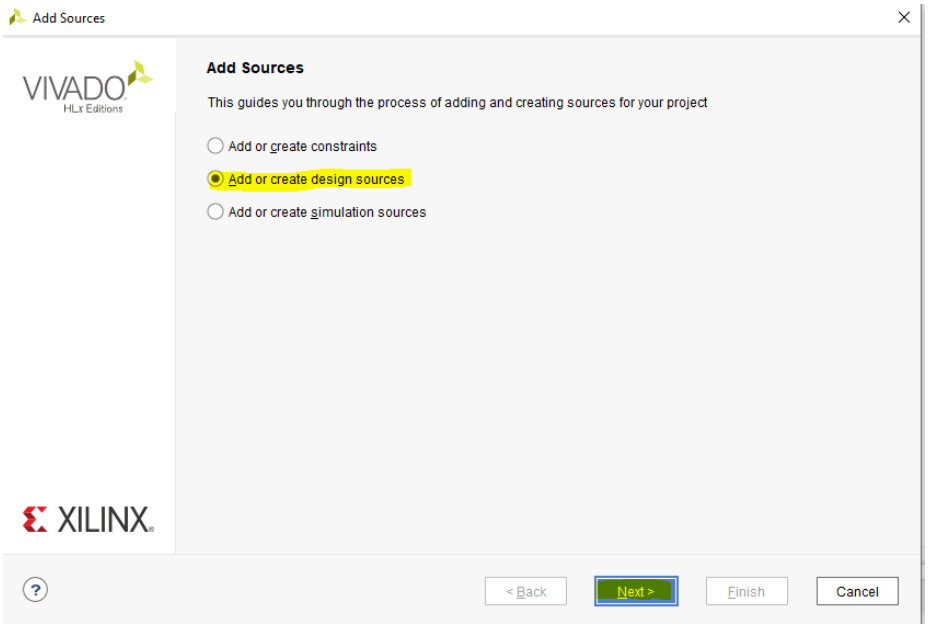


2. Teine variant on lisada uus fail.

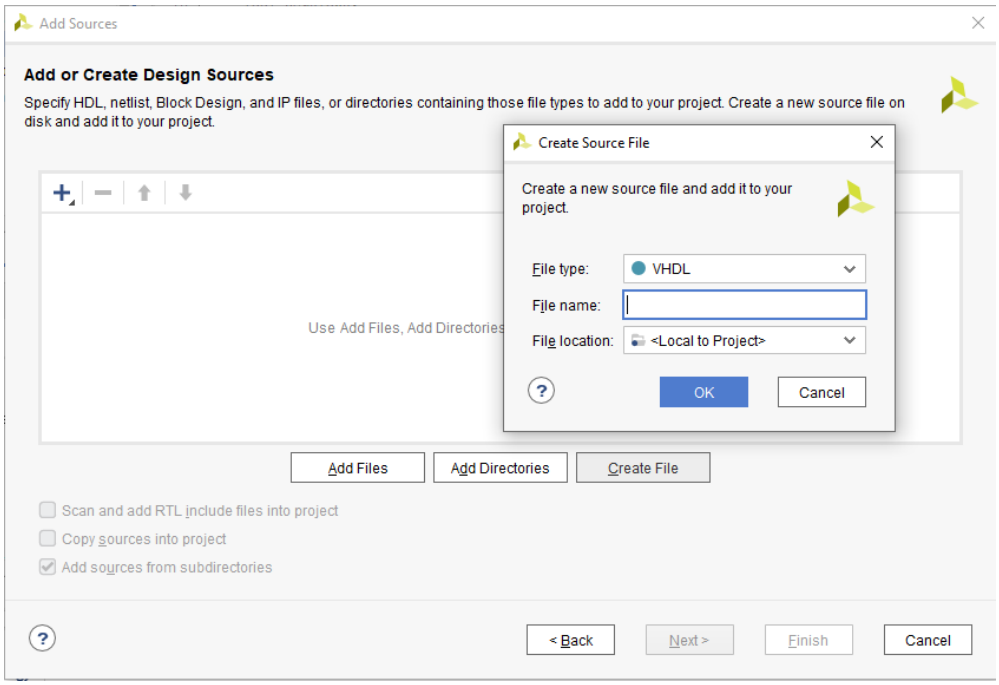
a. Tuleb vajutada „Sources“ aknas parempoolse hiireklahviga ning valida „Add Sources“ samuti võib kasutada kiirklahve Alt+A.



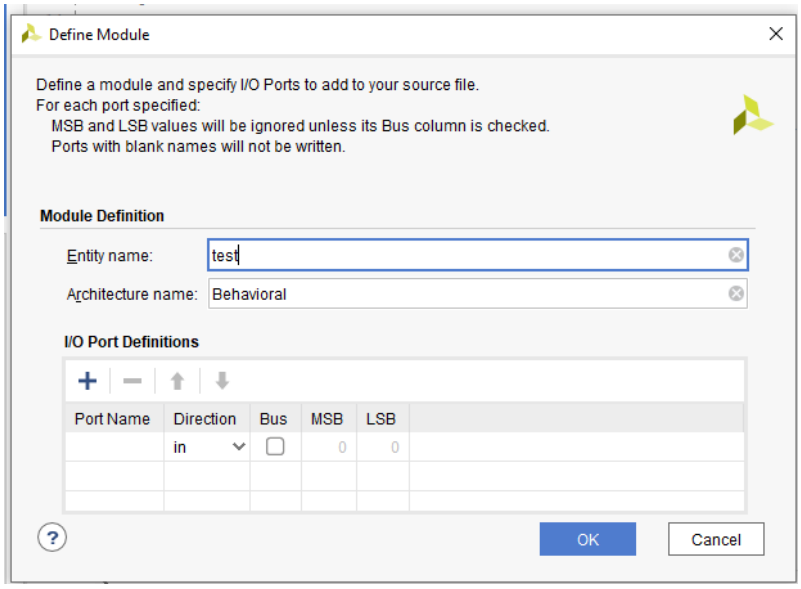
b. Peab olema valitud *design sources* ning seejärel vajutada nuppu „Next“.



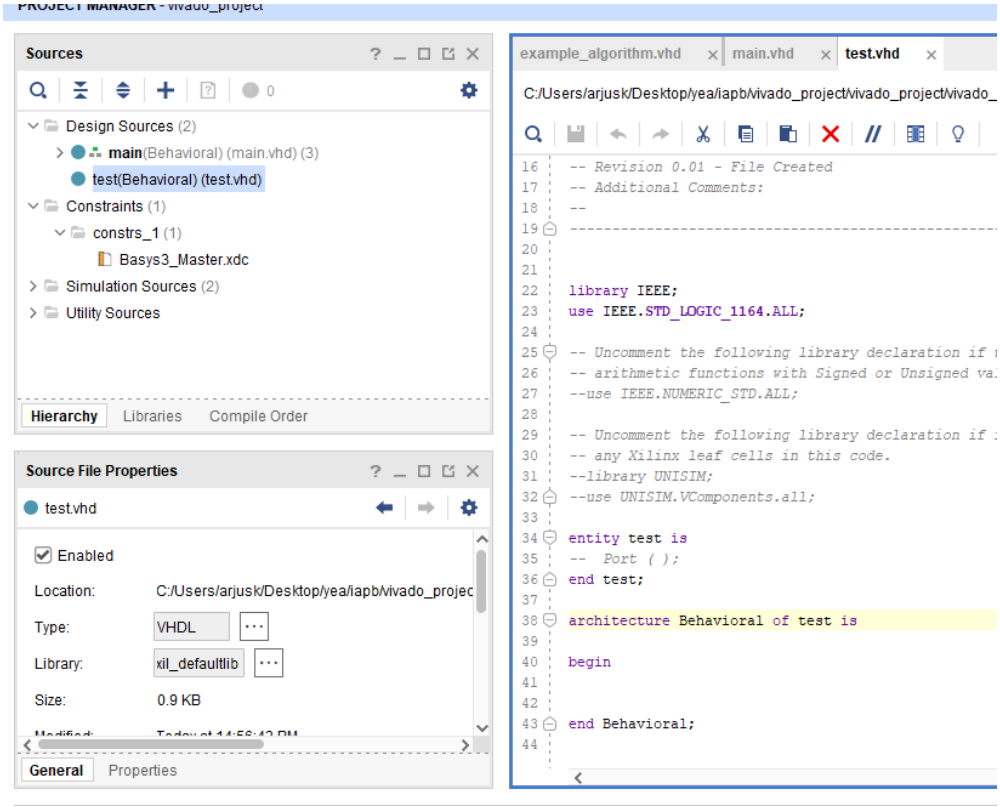
c. Ilmub aken, kus tuleb vajutada „Create file“. Sisestada failinimi ja vajutada „OK“. Peale seda muutub aktiivseks nupp „Finish“ ning tuleb vajutada seda.



d. Kui soovitakse algoritmi otse projektis kirjutada ning on teada, mis on sisend signaalid ja väljund signaalid saab teha selles aknas. Vajutada „OK“, kui kõik on valmis. Signaale saab hiljem juurde lisada, muuta või eemldada.



e. Nüüd on loodud fail „test.vhd“ *entity*’ga test. Siia võib kirjutada algoritmi või kopeerida seda faili.



f. Algoritmi tuleb lisada seda protsessi, et saaks algoritmi debugida ehk valida registreid algoritmist, mida kuvatakse ekranile. Muutujad võivad olla teised vastavalt sellele, mida soovid näha ekraanil. Kuna *ar*, *br* ja *counter* on 8-bittised signaalid, siis tuleb lisada 8-bitti nulle selle ette, sest ekraanile kuvatakse 16-bittiseid numbreid (seda saab teha ka main *entity*’s, kuid pole soovituslik).

```
process(sel, ar, br, cr, counter)
begin
        case sel is
                when "00" => dbg <= z8 & counter;
                when "01" => dbg <= z8 & ar;
                when "10" => dbg <= z8 & br;
                when "11" => dbg <= cr;
                when others => dbg <= (others => '0');
        end case;
 end process;
```

i. *counter* loendur (debugimiseks)

ii. *ar* on esimene number.

iii. *br* on teine number.

iv. *cr* on vastus.

v. *dbg* on väljund, mida kuvatakse ekraanil

vi. *sel* lülitite asendi signaalid

g. Samuti tuleb jälgida, et in ja out signaalid on õiget tüüpi. Nimetused võivad olla erinevad. Kasutusel on peamiselt *std_logic*, *unsigned* ja *std_logic_vector*. Neid võib muuta, kuid tuleb muuta tüübid ka teistes failides.

```
entity example_algorithm is
  port (xi, yi : in unsigned(15 downto 0):= (others => '0');
        rst    : in std_logic;
        sel    : in unsigned(1 downto 0);
        xo, dbg: out unsigned(15 downto 0);
        rdy    : out std_logic;
        clk    : in std_logic);
end example_algorithm;
```

h. Selleks, et lisada loodud algoritmi *main*'i . Tuleb lisada komponenti.

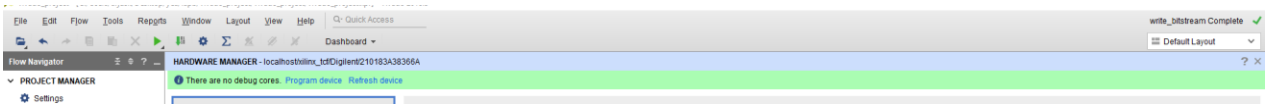
```
  component example_algorithm is
    port (xi, yi : in unsigned(15 downto 0);
          rst    : in std_logic;
          sel    : in unsigned(1 downto 0);
          xo, dbg: out unsigned(15 downto 0);
          rdy    : out std_logic;
          clk    : in std_logic);
  end component;
```

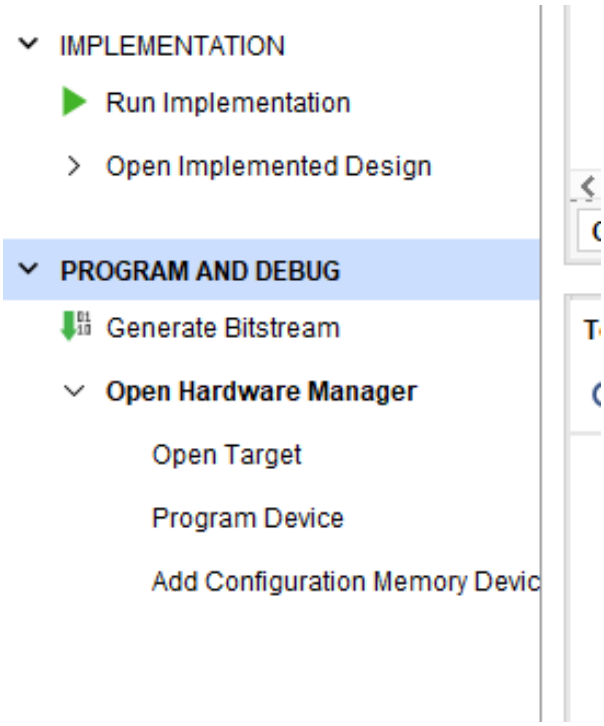
i. Seejärel tuleb mäppida signaalid ära.

```
algorithm : example_algorithm
port map (
    xi => unsigned(first_number),
    yi => unsigned(second_number),
    rst => btnd,
    sel => sel,
    std_logic_vector(xo) => answer,
    std_logic_vector(dbg) => choice,
    rdy => rdy,
    clk => cclk
);
```

Peale seda, kui algoritm on valmis, õige aeg on teda Basys 3 plaadile peale laadida. Seda võib teha samuti kahel viisil. Üks on kindel ning alati töötav, teine meetod on aga selline, mis

vahepeal ilmub ning vahel mitte. Mõlema meetodi jaoks on vaja genereerida bitstream, mis läheb plaadi peale ning tuleb ühendada Basys 3 arvutiga kasutades USB to *Micro-USB* ning plaadil *Micro-USB* pesa kõrval oleva lüliti vajutada. Kui plaat on sees, läheb ekraan tööle. Bitstreami genereerimise õnnestumist on näha paremal üleval nurgas.



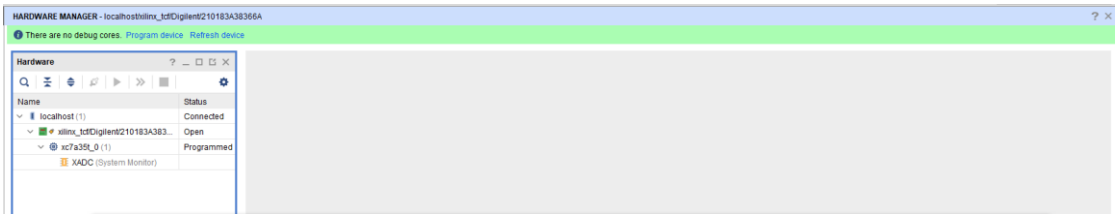


1. Peale plaadi ühendamist peab vajutama „Open Hardware Manager“. Seejärel avaneb võimalus vajutada „Open Target“, kust tuleb valida „Auto connect“. Kui seda teostatakse esimest korda, siis tuleb oodata, kuni kõik vajalikud driverid ja ühendused on ära tehtud. Peale seda tuleb vajutada „Program Device“ ja valida ühe ainsa võimaluse. Kui *bitstream*’i genereerimise kausta asukohta pole muudetud, siis on kergem. Ilmub aken, kus saab valida *bitstream*’i, seal ei pea midagi muuta võib vajutada nuppu „Program“.
2. Peale plaadi ühendamist ilmub ülesse roheline riba, kuid alati seda ei juhtu.

Kui vajutada vasakul menüüs „Open Hardware Manager“, siis ilmub ülesse roheline riba.

Kui „Open target“ on vajutatud, peab vajutama „Auto connect“ peale.

Kui on ühendatud, siis on siit näha.



Kui *bitstream* on genereeritud, siis saab seda peale laadida vajutades „Program device“ ja „OK“.

Selleks, et sisestada esimest numbrit, tuleb lülitid alumisest asendist ümber lülitada ülemisse, vastavalt sellele, mis numbrit soovitakse 2-nd süsteemis sisestada. Seejärel tuleb vajutada vasakut nuppu (*btnl*). Numbri sisestamisel ekraanil kuvatakse sisestatud numbrit 16-nd süsteemis.

Selleks, et sisestada teist numbrit, tuleb lülitid alumisest asendist ümber lülitada ülemisse, vastavalt sellele, mis numbrit soovitakse 2-nd süsteemis sisestada. Seejärel tuleb vajutada keskmist nuppu (*btnc*). Numbri sisestamisel ekraanil kuvatakse sisestatud numbrit 16-nd süsteemis.

Selleks, et käivitada algoritm ning kohe kuvada vastust tuleb vajutada ülemist nuppu (*btnu*).

Selleks, et käivitada algoritmi samm haaval, et debugida ehk kuvada algoritmi vahe tulemusi , tuleb hoida all alumist nuppu (*btnr*) ja vajutada (*btnr*). Seejärel võib mõlemad nupud lahti lasta ja iga parema nuppu (*btnr*) vajutusega antakse takt sisse ehk tehakse järgmine arvutuse samm.

Selleks, et kuvada erinevaid registreid saab kasutada vasakult kolm esimest lülitit. Lülitite asendi ja kuvatavate registrite kodeeringut võib leida järgmisest Tabelist 1. Kuvatavate registrite nimed on võetud näidisalgoritmist, mis asub failis „algorithm.vhd“.

Vasakult esimene	Vasakult teine	Vasakult kolmas	Kuvatav register
0	0	0	Esimene number
0	0	1	Teine number
0	1	0	Esimene tulemus
0	1	1	Teine tulemus
1	0	0	Loenduri väärtus algoritmis
1	0	1	Esimene number algoritmis
1	1	0	Teine number algoritmis
1	1	1	Tulemus algoritmis

LED’e kasutatakse selleks, et jälgida, mis olekus asub Vivado projektis asuv juhtautomaat ning mitme teise registri väärtuseid. Tabelis 2 on välja toodud, mis LED, mida näitab.

LED	Mida kuvab ?
led(0)	*manual* registri väärtus
led(1)	algoritmi *ready* väärtus
led(2)	*algoritmi_reset* väärtus
led(3)	*cclk* väärtus
led(10)	*state* on *s_ready* olekus
led(11)	*state* on *s_run* olekus
led(12)	*state* on *s_reset2* olekus
led(13)	*state* on *s_reset1* olekus
led(14)	*state* on *s_manual* olekus
led(15)	*state* on *s_start* olekus

Katsetamiseks võib genereerida alguses ühest algoritmist *bitstream* ja seejärel teisest. Tuleb vajaliku algoritmi mäppimise kommenteerida välja, mida ei soovita ja vastupidi. NB! Kuna algoritmis GCD on mitu vaheolekut, siis tuleb umbes kolm korda vajutada paremat nuput takti sisse andmisel, et toimuks arvutus, kuna liigutakse ühest olekust

teisse. Korrutusalgoritmis liigutakse arvutus olekust kohe arvutus olekusse, seepärast toimub seal iga vajutusega vahetulemuste muutus.