TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gustav Kirsipuu 155770IASB

Aritmeetika-algoritmide realiseerimine FPGA-l

Bakalaureusetöö

Juhendaja: Peeter Ellervee
Professor / Ph.D

Tallinn 2020

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: Gustav Kirsipuu

04.01.2021

Annotatsioon

Käesoleva lõputöö raames kirjeldab autor lahti Xilinx Vivado projekti koostamise protsessi, mille abil on tudengitel võimalik enda loodud aritmeetika-algoritmi realiseerida Digilent Basys 3 prototüüpimisplaadil. Põhjalikumalt keskendutakse töös juhtautomaadi ja kasutatud moodulite tutvustamisele. Samuti on lahti seletatud loodud Vivado projekti funktsionaalsused kasutades Basys 3 prototüüpimisplaati.

Lõputöö kõige mahukamaks osaks osutus juhtautomaadi loogika disainimine algoritmi töö juhtimiseks. Suurimad probleemid tekkisid selles etapis algoritmi sammhaaval lahendamise loogikat paika pannes. Probleemide lahendamiseks programmeeriti Basys 3 prototüüpimisplaat Vivado projekti bitstreamiga, rakendades reaalajas testimist.

Lõputööga on kaasas kasutusjuhend Vivado projekti realiseerimiseks FPGA'l. Juhendis on lahti seletatud, kuidas pakendatud projektile lisada uus algoritm ning viisid algoritmi *debugimiseks*. Samuti on kirjeldatud Basys 3 prototüüpimisplaadil kasutatud sisend- ja väljundseadmete loogika.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 24 leheküljel, 5 peatükki, 22 joonist, 1 tabelit.

Abstract

Implementing Arithmetic Algorithms on FPGA

In this thesis, the author describes the process of creating a packaged Xilinx Vivado project, which allows students to implement their own arithmetic algorithms on a Digilent Basys 3 prototyping board. The research focuses on the introduction of the control unit and the modules used, furthermore, the functionalities of the created Vivado project are also explained.

The most time consuming part of the thesis turned out to be the design for the control unit logic that controls the workflow of the algorithm. The biggest problems arose at this stage of the project while establishing the logic for step by step solving of the algorithm. To solve the problems, a Basys 3 prototyping board was used for real-time testing. The prototyping board was programmed with the bitstream of the Vivado project.

The thesis is accompanied by instructions for the implementation of the Vivado project on the FPGA. The user manual explains how to add a new algorithm to the packaged project and introduces ways to debug the algorithm. The logic for the input and output devices used on the Basys 3 prototyping board is also described.

The thesis is in Estonian and contains 24 pages of text, 5 chapters, 22 figures, 1 tables.

Lühendite ja mõistete sõnastik

FPGA	*Field-programmable gate array*, korduvprogrammeeritav loogika
IA	Arvutisüsteemide instituut
LED	*Light-emitting diode*, valgusdiood
VHDL(VHSIC-HDL)	*Very High Speed Integrated Circuit Hardware Description Language*, riisvara kirjelduskeel
GCD	*Greatest common divisor*, suurim ühistegur
USB	*Universal Serial Bus*, USB-port
JTAG	*Joint Test Action Group*, digitaalse disaini testimise standard

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

1.1 Taust

TalTech õppeaine „Digitaalsüsteemid” läbimise käigus lahendavad tudengid kaks kodutööd, mille käigus tudengid õpivad digitaalsüsteemide disaini, projekteerimist ja optimeerimist. Teises kodutöös on tudengite eesmärgiks realiseerida neile määratud aritmeetika algoritm VHDL riistvara kirjelduskeeles. Kodutöö viimaseks osaks on loodud algoritmi implementeerimine FPGA prototüüpimisplaadil, et tudengid saaksid enda loodud algoritmi näha reaalsuses töötamas. See annab hea ülevaate riistvara kirjeldamisel kaasnevatest iseärasustest ja võimalikest probleemidest.

1.2 Eesmärgid

Lõputöö eesmärgiks on koostada pakendatud Xilinx Vivado projekt, mis on realiseeritud kasutades VHDL riistvara kirjelduskeelt. Projekt koosneb mitmest moodulist, millest osad on korduvkasutusel vigade vältimiseks. Samuti luuakse lõputöö käigus uus moodul, mis haldab kogu projekti tööd. Üheks mooduliks on tudengite poolt loodud teise kodutöö VHDL kood, mille nad saavad lisada Vivado projekti sisse. Lõputöö tulemiks on pakendatud Vivado projekt, mida saavad tudengid tulevikus kasutada õppeaine „Digitaalsüsteemid” läbimisel enda teise kodutöö implementeerimiseks FPGA prototüüpimisplaadil. Tulemi realiseerimiseks on kasutatud Xilinx Vivado tarkvara ja prototüüpimisplaati Digilent Basys 3.

Teiseks tulemiks on loodud Vivado projekti kasutamiseks mõeldud kasutusjuhend. Kasutusjuhendis on põhjalikult lahti kirjeldatud protsess pakendatud projekti lisamiseks Vivado’sse ja lahti seletatud kuidas kasutaja peab enda loodud algoritmi sinna sisestama. Lisaks on seal kirjeldatud, kuidas on võimalik projektile luua bitivoog, et seda saaks Digilent Basys 3 prototüüpimisplaadile programmeerida.

1.3 Sisend- ja väljundmoodulite projekteerimine aritmeetika-algoritmide prototüüpimiseks FPGA arendusplaadil

Loodud projekt võimaldab kasutajal sisestada kuni 16-bitiseid numbreid, juhtida algoritmi tööd ja kuvada algoritmi väljundeid 7-segmendilisel indikaatoril. Algoritmi on võimalik juhtida kahel viisil. Esimene variant annab algoritmile prototüüpimisplaadi taktsageduse ja nii kuvatakse koheselt tulemus. Teine variant lubab kasutajal algoritmi tööd juhtida sammhaaval, andes seeläbi parema võimaluse jälgida vahetulemusi ning algoritmi olekute vahelist liikumist.

Lõputöö käigus loodud projekt annab tudengitele edaspidiseid töid lihtsustava vahendi, mille abil on võimalik prototüüpida aritmeetika-algoritme FPGA arendusplaadil.

Lisaks oli antud töö eesmärgiks varem sarnasel teemal tehtud bakalaureusetöö „Aritmeetika algoritmide prototüüpimine FPGA'l" teatud aspektide täiendamine [12].

2 Ülevaade

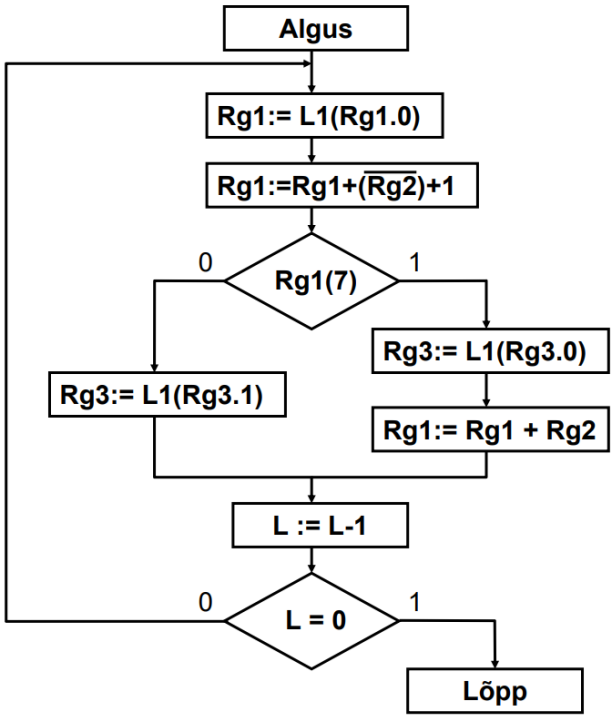
Töö käigus luuakse Vivado projekt erinevatest moodulitest ja disainitakse juhtautomaat, mida saavad tudengid edaspidi kasutada „Digitaalsüsteemid” aines teise kodutöö sünteesimiseks. Olemasolevate moodulite korduvkasutamine on lubatud vigade vältimiseks. Samuti on täiendatud sarnasel teemal varem tehtud bakalaureusetöö „Aritmeetika algoritmide prototüüpimine FPGA’l” mõningaid osasid.

2.1 Digitaalsüsteemide aritmeetika-algoritmide kodutöö

TalTech õppeaine „Digitaalsüsteemid” käigus peavad tudengid esitama kodutöö, mille eesmärk on realiseerida neile määratud algoritm VHDL kirjelduskeeles. Lõputöö käigus sünteesitavaks algoritmiks sai jagamine jäägi taastamisega.

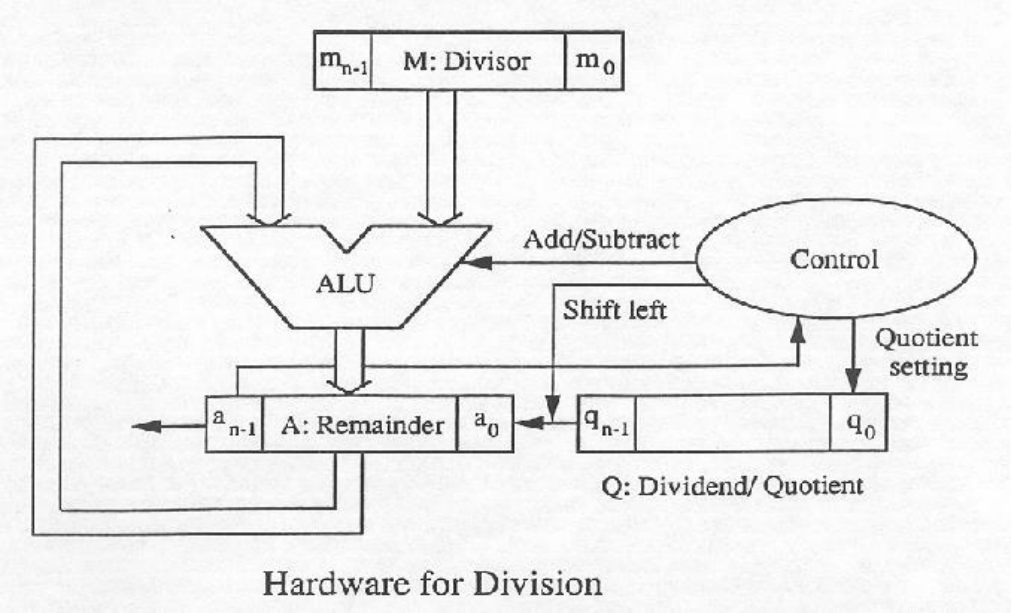
Õppeaine „Digitaalsüsteemid” kodutöö 2 nõuded [10]:

1. Luua algoritmi graafskeem koos käskude / juhtsignaalide tähistega.



Joonis 1 Graafskeem: jagamine jäägi taastamisega [11]

2. Operatsiooniseadme struktuurskeemi loomine, koos rakendatud juhtsignaalidega.



Joonis 2 Jäägi taastamisega jagamise struktuurskeem [13]

3. Vabatahtlik osa, algoritmi „käsitsi simuleerimine". Registrite muutuste jada.

4. Simuleeriv ja modelleeriv VHDL kood.

5. Algoritmi implementeerimine Basys 3 prototüüpimisplaadil kasutades loodud Vivado projekti.

2.2 Ülevaade lõputööst

Lõputöö käigus luuakse Vivado projekt, mida tudengid saavad tulevikus kasutada „Digitaalsüsteemid" õppeaines teise kodutöö implementeerimiseks Basys 3 prototüüpimisplaadil.

Lõputöö käigus läbitavad sammud:

1. VHDL kirjelduskeelega tutvumine.

2. Tutvumine Xilinx Vivado tarkvara funktsionaalsustega ja õppida seda kasutama projekti loomiseks ning Basys 3 FPGA'l implementeerimiseks.

3. Basys 3 prototüüpimisplaadi sisend ja väljund loogika planeerimine.

4. Leida juba olemasolevad moodulid, mida töö käigus võib vaja minna.
5. Juhtautomaadi disain, luua juhtautomaat kasutades sünteesitava algoritmina juhendaja tehtud GCD algoritmi.
6. Õppeaine „Digitaalsüsteemid" kodutöö sammude läbimine, mis on välja toodud punktis 2.1
7. Vivado projekti pakendamine kasutamise lihtsustamiseks.
8. Juhendi loomine Vivado projekti kasutamise seletamiseks.

3 Projekteerimine

Antud peatükk tutvustab lõputöö käigus loodud projekti realiseerimiseks kasutavatele vahenditele seatud nõudeid, koos lühitutvustusega. Samuti on välja toodud loodud projektile seatud realiseerimise nõuded.

3.1 Projekteerimiseks kasutatud vahendid

Juhendaja poolt määratud nõuded lõputöö realiseerimiseks:

1. Kasutatav prototüüpimisplaat on Digilent Basys 3 [1].
2. Riistvara kirjelduskeelena kasutada VHDL'i [3].
3. Projekteerimisel kasutada Xilinx Vivado tarkvara [4].

3.2 Kasutatud vahendite kirjeldus

Basys 3 plaat on komplektne, kasutusvalmis digitaallülituste arendamiseks mõeldud platvorm, mis põhineb viimasel Artix-7 FPGA'l. Sellel on piisavalt lüliteid, LED'e ja teisi sisend/väljund seadmeid, mis võimaldavad suure hulga disainide implementeerimist, ilma, et oleks vaja lisa riistvara [1]. Arvestades eelpool välja toodud põhjuseid ja võimalust juhendaja käest Basys 3 prototüüpimisplaati laenata, siis sobis see hästi lõputöö eesmärgi saavutamiseks.

Vivado Design Suite on Xilinx poolt loodud tarkvara riistvara kirjeldus disainide sünteesiks ja analüüsiks. Erinevalt eelkäiast Xilinx ISE, pakub uus lahendus paremaid võimalusi kõrg-taseme sünteesiks ja sisseehitatud disainide loomiseks [4]. Xilinx pakub ülikoolis õppivatele tudengitele tasuta Vivado litsentsi, mida selle töö käigus ka kasutati. Kuna lõputöö käigus kasutatav Basys 3 prototüüpimisplaat on loodud kasutamiseks koos Vivado tarkvaraga, siis oli see ainuke valik projekteerimiseks.

VHDL on üks enim kasutatud riistvara kirjelduskeeltest digitaalsete trükkplaatide disainis [2]. Kuna see on loetav nii masinatele kui ka inimestele, siis toetab see riistvara disainide

arendust, kontrollimist, sünteesimist ja testimist [3]. VHDL osutus valituks lõputöö realiseerimiseks kuna seda kasutatakse õppeaine „Digitaalsüsteemid” käigus ning töö autor omas varasemaid kogemusi selle käsitlemisel.

3.3 Projekteerimise nõuded

Koostöös juhendajaga määratud nõuded Vivado projektile:

1. Kasutaja peab saama sisestada kaks operandi:
    a. Operandid sisestatakse kasutades Basys 3 plaadil olevaid lüliteid. Joonisel 3 välja toodud kastis number 1.
    b. Sisestatud operandid on kahendkoodis ja kuni 16 bitised.
    c. Vasakut nuppu vajutades loetakse sisse esimene operand. Joonisel 3 kastis number 2 nupp „L”.
    d. Paremat nuppu vajutades loetakse sisse teine operand. Joonisel 3 kastis number 2 nupp „R”.
2. Kasutajal on võimalik algoritm käivitada nupu vajutusega:
    a. Keskmist nuppu vajutades antakse algoritmile operandide väärtused. Joonisel 3 keskmine nupp ehk „C”.
    b. Algoritmile antakse Basys 3 plaadi taktsignaal.
3. Kasutaja saab algoritmi käivitada sammhaaval:
    a. Ülemist nuppu vajutades antakse algoritmile operandide väärtused. Joonisel 3 nupp „U”.
    b. Algoritmile antakse taktsignaal ainult ülemise nupu vajutamisel.
    c. Keskmist nuppu vajutades antakse algoritmile Basys 3 plaadi taktsignaal.

4. Kasutajal on võimalik näha registrite väärtusi 7-segmendilisel indikaatoril kuueteistkümnendsüsteemis, joonisel 3 välja toodud kastis number 3:

    a. Sisestatud operandi väärtus kuvatakse vasaku ja parema nupu vajutusel.

    b. Algoritmi töö lõpus kuvatakse väljund.

    c. Võimalusel saab kuvada vahesammude tulemusi kasutades kolme kõige vasakpoolsemat lülitit.

5. Kasutajal on võimalik jälgida algoritmi tööd kasutades lülitite kohal olevaid LED’e:

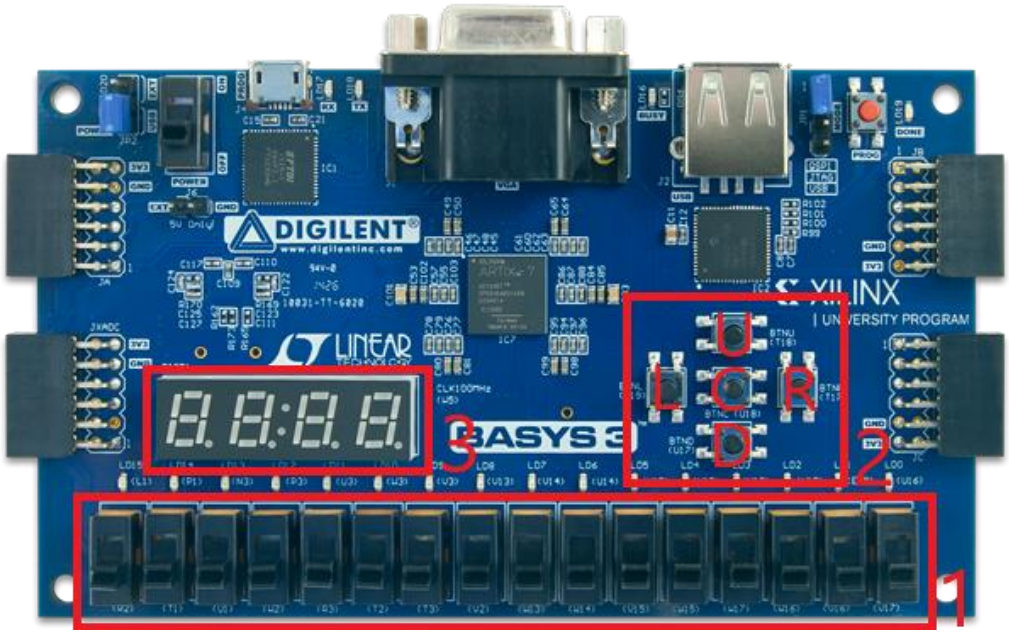
    a. Juhtautomaadi olek on kuvatud vasakpoolsematel LED’del.

    b. Kasutaja algoritmi olekute kuvamiseks saab kasutada parempoolseid LED’e.

6. Kasutaja saab algoritmile anda *reset* signaali:

    a. Alumist nuppu vajutades antakse algoritmile *reset* signaal. Joonisel 3 nupp „D”.

    b. Juhtautomaat ja algoritm peavad mõlemad minema algolekusse.



Joonis 3 Ülevaade Basys 3 prototüüpimisplaadist [14]

16-nd süsteem osutus valituks projekti jaoks kuna Basys 3 prototüüpimisplaat võimaldab *switchide* abil 16-bitiste arvude sisestamist ja 4-kohalisel 7-segmendilisel indikaatoril on võimalik dekoodri abil kuvada väljundit heksadetsimaal kujul. Oletame, et 2-nd süsteemis arv on “1111111111111111”, siis Basys 3 ekraanile mahub dekoodri abil kuvamiseks sama arv 16-nd süsteemis “FFFF”.

Nõue algoritmi sammhaaval lahendamiseks sai määratud seetõttu, et tudengitel oleks võimalik jälgida loodud algoritmi tööd ja vahetulemusi. See määras ka multipleksori kasutuse, mille abil saab vahetada erinevate vahetulemuste vahel.

Debugimise eesmärgil on kasutusel Basys 3 prototüüpimisplaadil olevad LED’id, et oleks võimalik leida vigu algoritmi juhtautomaadi töös.

4 Tulemused

See peatükk keskendub töö lahendamise protsessile ja Vivado projektis kasutatud moodulite kirjeldusele koos näidetega.

4.1 Töö käik

Käesoleva lõputöö autor alustas tööd VHDL riistvara kirjelduskeele meelde tuletamisega ja paremini tundma õppimisega. VHDL'i oli varasemalt kasutatud ülikoolis erinevate ainete läbimise käigus, kuid siis piirdus see lihtsamate probleemide lahendamisega. Xilinx'l on mitmeosaline seeria *Youtubes*, milles seletatakse põhjalikult lahti uue Vivado projekti loomine erinevate näidete abil.

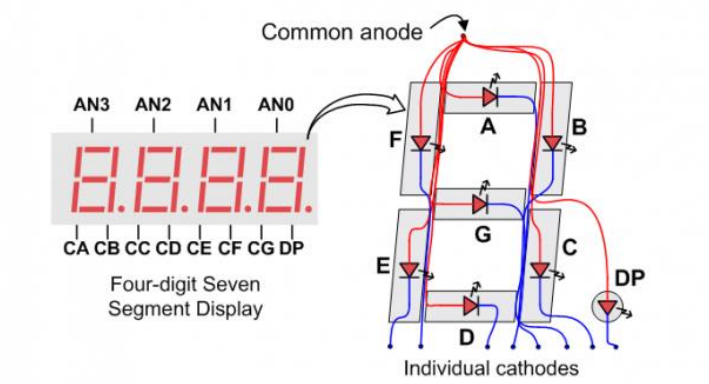
Projekti alguses seadis autor esialgseks eesmärgiks luua sünteesitav VHDL kood, mis salvestab vasaku ja parema nupu vajutamisel 16-bitised operandid vastavatesse registritesse ning seejärel kuvab salvestatud väärtust Basys 3 7-segmendisel indikaatoril. Valdav enamus informatsioonist, mida projekti baasi loomiseks kasutati on leitav „fpga4student" veebilehelt. Seal on põhjalikult välja toodud kuidas implementeerida dekooder 7-segmendilise indikaatori jaoks ja õpetus Basys 3 nuppude *debouncer'i* loomiseks [5] [6]. Põhi funktsionaalsuste testimiseks kasutati lihtsaid aritmeetika algoritme, kontrollimaks kas algoritm saab korrektsed sisestatud operandid. Samuti aitas see testida väljundi kuvamist.

Peale esialgse eesmärgi saavutamist pani töö autor koostöös juhendajaga paika juhtautomaadi loogika ja liikumise erinevate olekute vahel. Juhtautomaadi loomisel osutus kõige mahukamaks osaks algoritmi sammhaaval läbimise implementeerimine. Põhiliseks probleemiks osutus signaalide võistlus, mille tõttu algoritmi töö polnud korrektselt juhitav.

4.2 7-segmendiline indikaator

Basys 3 plaadil olev 7-segmendiline indikaator on kasutajale põhiliseks väljundseadmeks, millel kuvatakse sisendeid, vahetulemusi ja vastust kuueteistkümnendsüsteemis. Kuva

koosneb neljast osast, mis on seotud ühisanoodiga. Numbrikoht koosneb seitsmest segmendist, mis on paigutatud number 8 kujul ja iga segmendi jaoks on eraldi LED [1].



Joonis 4 Ühisanoodiga 7-segmendilise indikaatori segmentide kirjeldus [1]

Kuna 16-bitise kahendarvu teisendamine kuueteistkümnendsüsteemi on sama, mis nelja erineva 4-bitise kahendarvu puhul, siis saab dekooder anda igale kahendkoodi osale otse vastava heksadetsimaal väärtuse. Näiteks, kui soovida Basys 3 kuval näidata arvu “1234”, siis kahendkujul tuleb dekooderile anda arv “0001001000110100”. Seda kahendarvu saab vaadata nelja osana: “0001”, “0010”, “0011” ja “0100”. Andes need väärtused dekooderile, siis tulemusena kuvatakse väärtused “1”, “2”, “3” ja “4”.

```
process(LED_BCD)
begin
    case LED_BCD is
    when "0000" => LED_out <= "1000000"; -- "0"
    when "0001" => LED_out <= "1111001"; -- "1"
    when "0010" => LED_out <= "0100100"; -- "2"
    when "0011" => LED_out <= "0110000"; -- "3"
    when "0100" => LED_out <= "0011001"; -- "4"
    when "0101" => LED_out <= "0010010"; -- "5"
    when "0110" => LED_out <= "0000010"; -- "6"
    when "0111" => LED_out <= "1111000"; -- "7"
    when "1000" => LED_out <= "0000000"; -- "8"
    when "1001" => LED_out <= "0010000"; -- "9"
    when "1010" => LED_out <= "0100000"; -- a
    when "1011" => LED_out <= "0000011"; -- b
    when "1100" => LED_out <= "1000110"; -- C
    when "1101" => LED_out <= "0100001"; -- d
    when "1110" => LED_out <= "0000110"; -- E
    when "1111" => LED_out <= "0001110"; -- F
    when others => LED_out <= "1111111";
    end case;
end process;
```

Joonis 5 Segmentide dekooder [5]

Kuna kõik 4 numbrikohta kasutavad sama ühisanoodi, siis tuleb selgelt nähtava arvu kuvamiseks saata igale numbrikohale 1 kuni 16ms jooksul uus signaal. Tänu Basys 3 plaadi kiirele taktsagedusele pole vaja LED'ide väreluse pärast muretseda. Selleks, et määrata milline numbrikoht põlema läheb on tehtud loendur, mis iga taktitsükkli läbides kasvab 1 võrra. Joonisel 6 on toodud välja VHDL kood, mis otsustab vastavalt loenduri väärtusele, milline numbrikoht peab põlema.

```
process(LED_activatingCounter, displayedNumber)
begin
    case LED_activatingCounter is
    when "00" =>
        Anode_Activate <= "0111";
        LED_BCD <= displayedNumber(15 downto 12);
    when "01" =>
        Anode_Activate <= "1011";
        LED_BCD <= displayedNumber(11 downto 8);
    when "10" =>
        Anode_Activate <= "1101";
        LED_BCD <= displayedNumber(7 downto 4);
    when "11" =>
        Anode_Activate <= "1110";
        LED_BCD <= displayedNumber(3 downto 0);
    when others =>
        Anode_Activate <= "0000";
    end case;
end process;
```

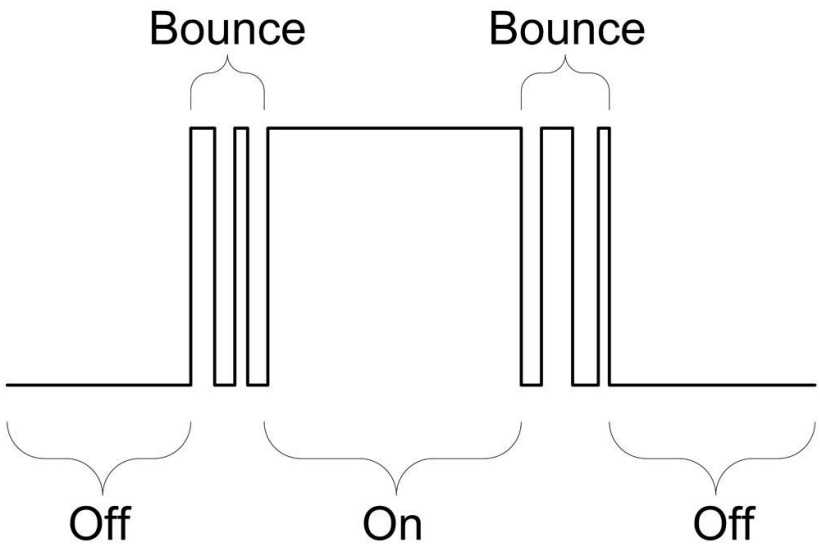
Joonis 6 Ühisanoodi aktiveerimine [5]

Loenduri väärtused "00", "01", "10" ja "11" määravad ära, millisele numbrikohale parajasti signaal saadetakse. Varem välja toodud näite põhjal näeme, et kui loenduri väärtus on "00", siis kuvatakse esimesel numbrikohal "0001" ehk number "1". Esimene numbrikoht ehk "00" asub kõige vasakul ja viimane "11" kõige paremal. "Anode_Activate" antud väärtuses määratakse 0-ga ära, milline numbrikoht signaali saab. Näiteks "0111" tähendab, et esimene numbrikoht põleb.

4.3 Värelusvastane moodul (*Debouncer*)

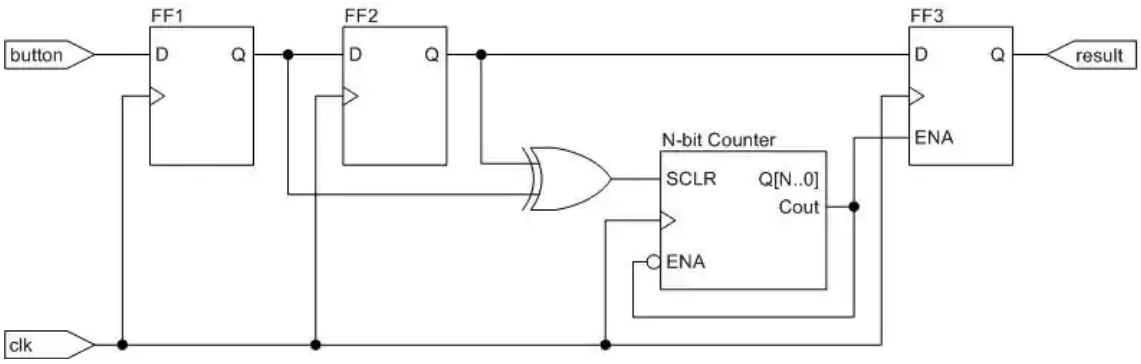
Teoorias võiks nuppude kirjeldamisel riistvara tasandil eeldada, et nupule vajutades muutub signaal "1" ja seda signaali saaks omakorda kasutada vastavalt vajadusele. Tegelikkuses toimub nupu vajutusel ja lahti laskmisel hoopis mitmeid signaali hüppeid "1" ja "0" vahel. Selle vältimiseks on vaja kasutada riistvara kirjeldamisel *debouncer'i*,

mis eemaldab signaali hüppamise ja tekitab ühe stabiilse signaali. Joonisel 7 on välja toodud nupu vajutamise signaal, millel pole rakendatud *debouncer'i*.



Joonis 7 Nupu signaal ilma *debouncerita* [9]

Joonisel 8 on välja toodud *debouncer'i* loogika, nupu signaali väärtus loetakse pidevalt „FF1" ja „FF2". Juhul kui „FF1" ja „FF2" väärtused on võrdsed määratud ajavahemikus, siis saadetakse aktiveerimis signaal „FF3". Joonisel välja toodud „FF" elemendid on *Delay flip-flop*'id, mis annavad sisestatud signaali edasi kui „clk" on 1. Sealt edasi läheb väljundisse *debouncetud* nupu signaal [7].



Joonis 8 Debouncer'i loogikaskeem [7]

Ajavahemik, mis määrab ära kui kaua peab nupp olema alla vajutatud, et signaal liiguks edasi „FF3" on kirjeldatud loenduri maksimum väärtuse abil. Antud projekti jaoks oli määratud loenduri maksimum väärtuseks 250 000. Arvestades Basys 3 prototüüpimisplaadi taktsagedust 100MHz, siis teame et nupp peab olema alla vajutatud 2,5ms. Kui loenduri väärtus on võrdne või suurem kui 250 000, siis loendur nullitakse.

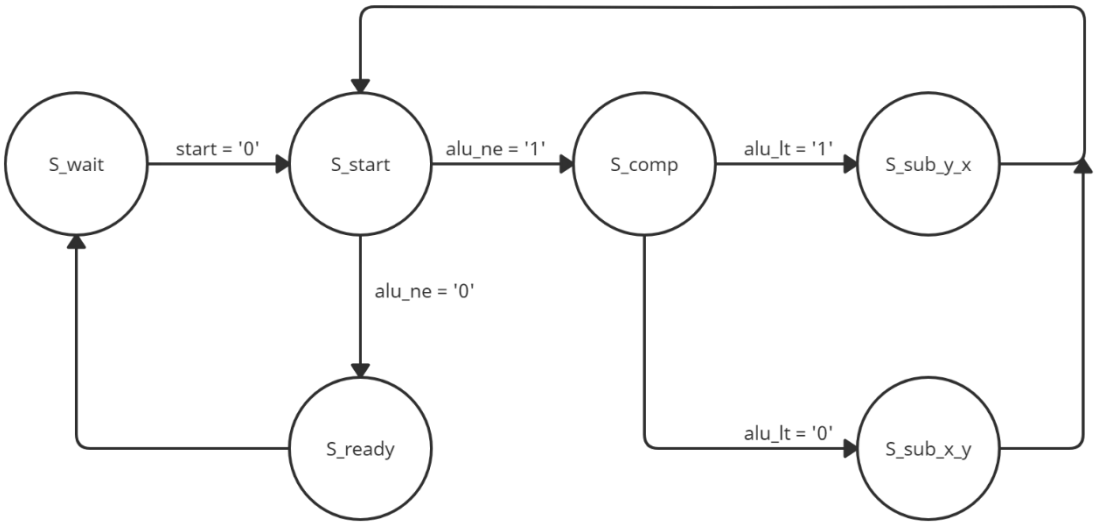
4.4 GCD algoritm

Vivado projekti juhtautomaadi testimiseks kasutati juhendaja poolt varem valmistatud GCD algoritmi. Algoritmi töö käigus leitakse kahe arvu suurim ühistegur. Suurim ühistegur leitakse sisestatud arvude võrdlemisel ja seejärel suuremast operandist lahutatakse teine.

```
while ( x != y ) {
    if ( x < y ) y = y - x;
    else x = x - y;
  }
```

Joonis 9 GCD algoritm [8]

Näiteks on kaks sisestatud arvu x = 27 ja y = 18, siis algoritm ootab *S_wait* olekus kuni *start* signaal muutub 0-ks ning liigub edasi *S_start* olekusse. Sealt edasi liigub algoritm *S_comp* olekusse, kus leitakse kumb arvudest on suurem. Algoritmi töö käib kuni x ja y on võrdsed ehk kuni on leitud suurim ühistegur. Kuna x on suurem kui y, siis liigub algoritm *S_sub_x_y*, kus leitakse uus x väärtus 27-18 = 9. Peale uue x väärtuse leidmist liigub algoritm uuesti *S_start* olekusse ja tehakse kontroll kas x = y. Nüüd on y suurem kui x ja algoritm liigub *S_sub_y_x* olekusse, kus leitakse y uus väärtus 18-9 = 9. *S_start* olekus leitakse, et x ja y on võrdsed ning algoritm liigub edasi *S_ready* olekusse, kus saadetakse juhtautomaadile *ready* = “1” signaal. Sellega on algoritmi töö lõppenud ja kuvatakse 7-segmendilisel indikaatoril kahe arvu suurim ühistegur 9 [8]. Joonisel 10 on välja toodud GCD algoritmi siirdeolekudiagramm.



Joonis 10 GCD algoritmi siirdeolekudiagramm

Juhendaja GCD algoritmi testimise käigus on *debugimise* eesmärgil tehtud mõned muudatused. Esiteks on võetud kasutusele lülitite kohal olevad LED'id, et jälgida millises olekus GCD algoritm parajasti on. See muudatus aitab jälgida sammhaavul algoritmi läbimist. Teise muudatusena on lisatud algoritmile protsess, mille abil saab kuvada 7-segmendilisel indikaatoril vahetulemusi, kasutades selleks kolme kõige vasakpoolsemat lülitit.

```
process(sel, x, y, xo_bf)
begin
  case sel is
      when "00" => dbg <= xo_bf;
      when "01" => dbg <= x;
      when "10" => dbg <= y;
      when others => dbg <= (others => '0');
  end case;
end process;
```

Joonis 11 Protsess vahetulemuste kuvamiseks

Joonisel 11 on välja toodud protsess algoritmi vahetulemuste kuvamiseks. Näiteks kui Basys 3 lülitid 14 ja 13 omavad vastavalt väärtusi '0' ja '1', siis kuvatakse 7-segmendilisel indikaatoril *y'i* väärtus sellel aja hetkel.

Lüliti 15 asend	Lüliti 14 asend	Lüliti 13 asend	Kuvatav number
0	0	0	Lõpptulemus
1	0	0	Registri 'xo_bf' väärtus
1	0	1	Registri 'x' väärtus
1	1	0	Registri 'y' väärtus
1	1	1	Pole kasutusel GCD algoritmis

Tabel 1 *Debug* lülititele vastavad väärtused

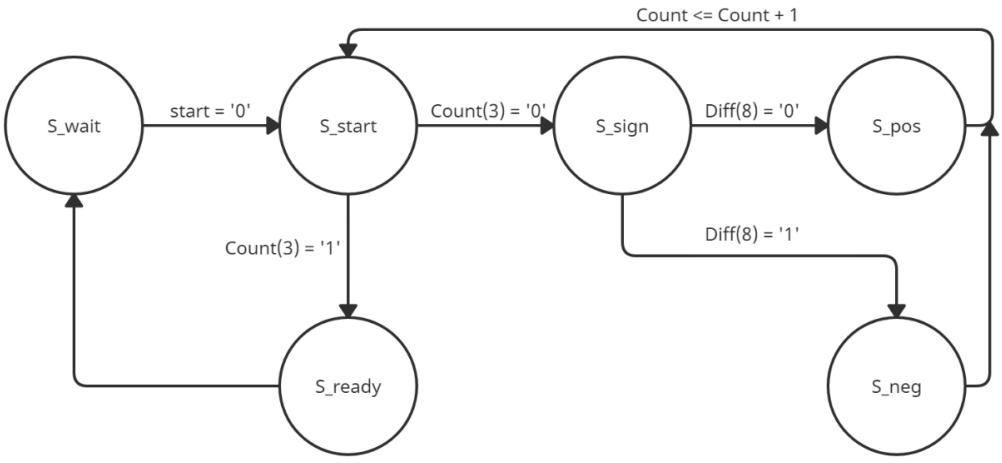
4.5 Kodutöö algoritm

Lõputöö käigus sünteesitava „Digitaalsüsteemid" teise kodutöö variandiks osutus jagamine jäägi taastamisega. Eesmärk oli luua graafskeemi põhjal algoritm, mis teostab kahe arvu jagamise koos jäägi taastamisega ja on sünteesitav prototüüpimisplaadil.

Jäägi taastamisega jagamise protsess:

1. Kõigepealt toimub registrite täitmine algsete väärtustega. Reg1 – jagatav, Reg2 – jagaja ja Reg3, mille algväärtus on 0. Loendur on võrdne operandide pikkusega.
2. Toimub nihe vasakule.
3. Registrist 1 lahutatakse register 2.
4. Kontrollitakse registri 1 märki. Kui kõige suurema järgu bit on ‘0’, siis registrisse 3 lisatakse vasakule nihkega ‘1’. Vastasel juhul lisatakse ‘0’ ja toimub registri 1 väärtuse taastamine registri 2 liitmisel.
5. Loendurist lahutatakse 1.
6. Kontrollitakse ega loendur ei võrdu 0-ga. Kui võrdub, siis on algoritmi töö lõppenud, vastasel juhul liigub algoritm tagasi teise sammu juurde.
7. Registris 3 on jagatis ja registris 1 on jagamise jääk.

Kodutöö algoritmi implementeerimiseks Basys 3 prototüüpimisplaadil oli vaja lisada sellele juhtautomaat. Lisatud juhtautomaat on ülesehituselt üsnagi sarnane GCD algoritmi omaga. Joonisel 12 välja toodud automaadi suurimaks erinevuseks on see, et kasutusel on loendur, mille abil tehakse kindlaks kuna algoritm on töö lõpetanud.



Joonis 12 Jagamine jäägi taastamisega siirdeolekudiagramm

Sarnaselt GCD algoritmile lisati ka kodutöö käigus sünteesitavale algoritmile juurde joonisel 11 välja toodud protsess vahetulemuste kuvamiseks. Samuti lisati LED väljundid, et jälgida millises olekus algoritm parajasti asub.

4.6 Vivado projekt

Vivado projekti tähtsaim osa on *top.vhd* fail, kus on kirjeldatud terve projekti poolt kasutatavad sisend ja väljund signaalid ning nende liikumine erinevate komponentide vahel. Samuti on kõik projekti komponendid ükshaaval välja toodud *entity*'tena, kus on kirjeldatud komponendi poolt kasutatavad sisend ja väljund signaalid. Käsitletavas projektis oli kolm põhilist komponenti – *debouncer*, 7-segmendilise indikaatori dekooder ja sünteesitav algoritm.

```
component debouncingButton
    Port ( button          : in STD_LOGIC;
           clk             : in STD_LOGIC;
           debouncedButton : out STD_LOGIC
           );
end component;
```

Joonis 13 Debouncer komponent

Debouncer'i komponendi sisenditeks on puhas nupu signaal ja Basys 3 prototüüpimisplaadi taktsagedus ning väljundiks on *debouncetud* nupu signaal.

```
component decoder
    Port ( clk             : in STD_LOGIC;
           reset           : in STD_LOGIC;
           displayedNumber : in STD_LOGIC_VECTOR;
           LED_out         : out STD_LOGIC_VECTOR(6 downto 0);
           Anode_Activate  : out STD_LOGIC_VECTOR(3 downto 0)
           );
end component;
```

Joonis 14 7-segmendilise indikaatori dekooderi komponent

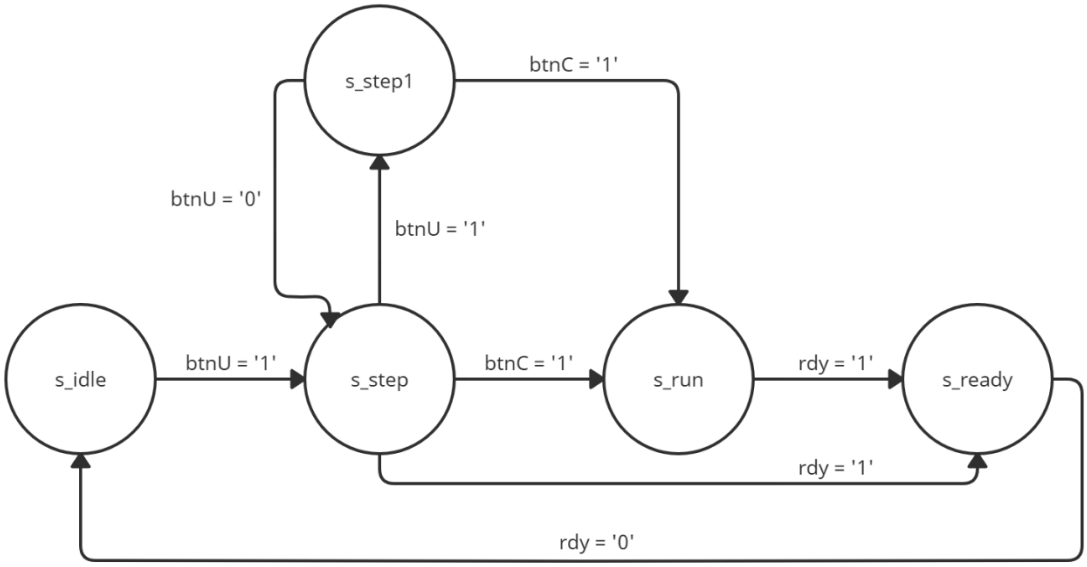
7-segmendilise dekooderi komponendi sisend signaalideks on Basys 3 taktsagedus, *reset* signaal nullimiseks ja kuvatav number kahendkoodis. Väljund signaalideks on ühisanoodi aktiveerimise signaal ja vastaval numbrikohal kuvatava arvu signaal.

Joonisel 15 on välja toodud VHDL kood, kus toimub operandide sisestamine ja kuvamine. Vasaku nupu vajutusel loetakse sisse esimene väärtus ja parema nupu vajutamisel teine. Väärtus, mida soovitakse kuvada 7-segmendilisel indikaatoril hoitakse „regC" registris.

```
    process(clk, debounced_pb1, debounced_pb2, debounced_btnC,
debounced_btnU)
    begin
        if(rising_edge(clk)) then
            if(debounced_pb1 = '1') then
                regA <= dipSW;
                regC <= dipSW;
            elsif(debounced_pb2 = '1') then
                regB <= dipSW;
                regC <= dipSW;
            else
                if(dipSW(15) = '1') then
                    regC <= dbg;
                elsif(debounced_btnC = '1') then
                    regC <= resultGCD;
                elsif(debounced_btnU = '1') then
                    regC <= resultGCD;
                elsif(reset = '1') then
                    regC <= "0000000000000000";
                end if;
            end if;
        end if;
    end process;
```

Joonis 15 VHDL kood numbrite sisse lugemiseks ja kuvamiseks

Lõputöö käigus loodud juhtautomaat paikneb samuti Vivado projekti *top.vhd* failis. Juhtautomaat kontrollib sünteesitava algoritmi lahendamise käiku vastavalt kasutaja soovile. Keskmisele nupule vajutades annab juhtautomaat algoritmile Basys 3 taktsignaali ja algoritm lahendatakse lõpuni ning kuvatakse vastus. Ülemisele nupule vajutades läheb juhtautomaat sammhaavul lahendamise režiimi ehk *s_step* olekusse. Ülemist nuppu uuesti vajutades liigub juhtautomaat *s_step1* olekusse ja püsib seal kuni nupu lahti laskmiseni. Iga järgneva nupu vajutusega antakse sünteesitavale algoritmile 1 taktsageduse tsükkli signaal ehk algoritm liigub edasi ainult ühe oleku võrra. Sammhaaval režiimist on võimalik igal ajahetkel vahetada ümber *s_run* olekusse, kasutades selleks keskmist nuppu. Algoritmi töö lõppedes saadetakse juhtautomaadile *ready* signaal, mis annab märku, et algoritm on töö lõpetanud. *Ready* signaali saamisel läheb juhtautomaat *s_idle* olekusse ja jääb uusi juhiseid ootama. Joonisel 16 on näha juhtautomaadi siirdeolekudiagrammi, kus on kirjeldatud automaadi loogika erinevate olekute vahel liikumiseks ja kuidas erinevad signaalid juhtautomaadi tööd mõjutavad.



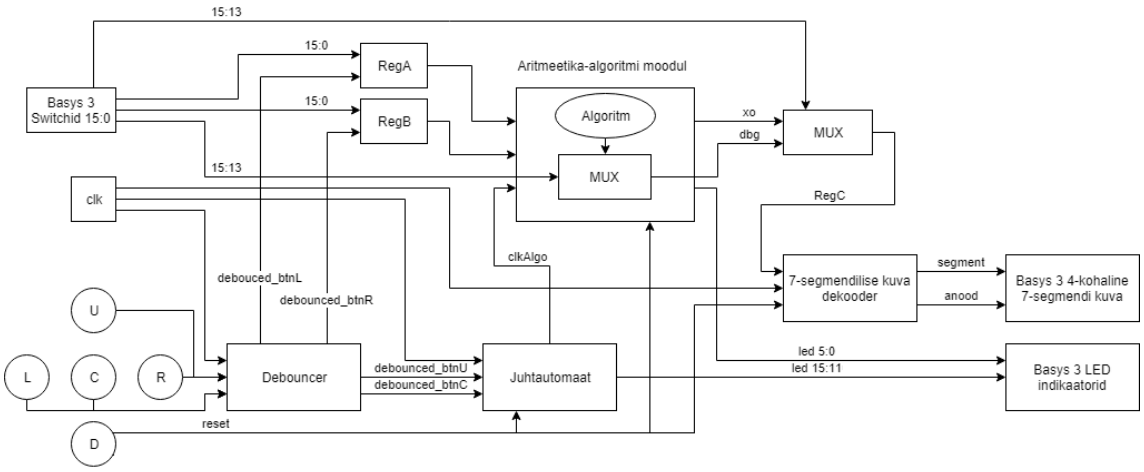
Joonis 16 Juhtautomaadi siirdeolekudiagramm

Projekti loomise käigus osutus suurimaks probleemiks algoritmi sammhaaval lahendamise loogika paika panemine. Samuti tekkis algselt sammhaaval juhtimise käigus mitme signaali vaheline vastuolu, mis takistas juhtautomaadi korrektset tööd.

Juhtautomaadi olekute jälgimiseks on sarnaselt GCD ja kodutöö algoritmidele kasutatud lülitite kohal paiknevaid LED väljundeid. Erinevalt neist on kasutatud vasakpoolsemaid LED’e, et oleks võimalik samaaegselt jälgida nii juhtautomaadi, kui ka sünteesitava algoritmi olekuid. Juhtautomaadi olekute kuvamiseks kasutatakse LED’e 15 kuni 11 ja algoritmi jälgimiseks on kasutusel LED’id 5 kuni 0. Vastavad LED’id juhtautomaadi olekutele on järgmised:

1. s_idle – LED 15
2. s_step – LED 14
3. s_step1 – LED 13
4. s_run – LED 12
5. s_ready – LED 11

Algoritmi olekute jälgimiseks saab kasutaja ise määrata, milline LED teatud olekus põleb.

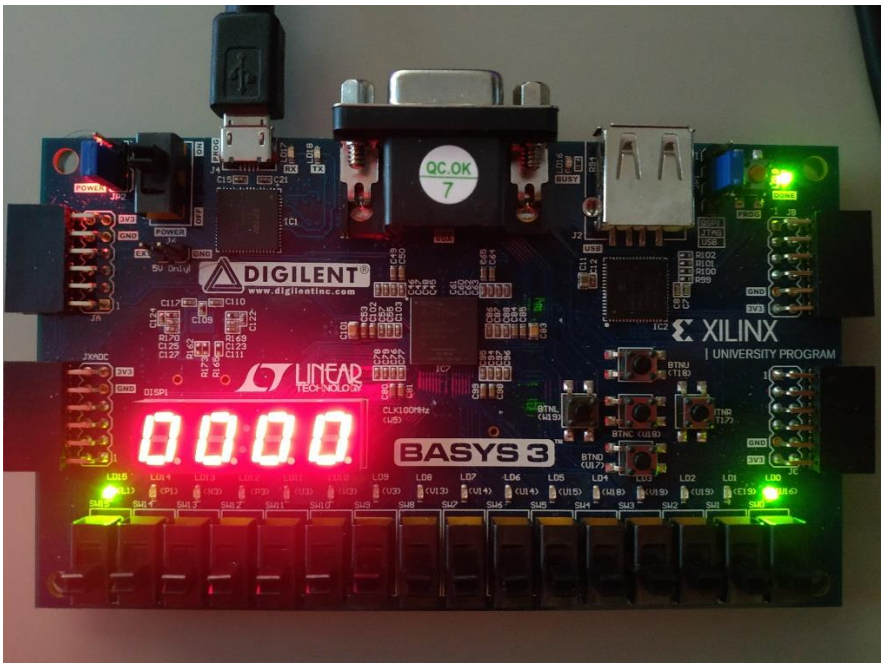


Joonis 17 Vivado projekti struktuurskeem

Struktuurskeem kirjeldab kogu projekti üldist loogikat ja annab hea ülevaate, kuidas erinevad moodulid üksteist mõjutavad. Joonisel 17 on näha Vivado projekti struktuurskeem, millel on välja toodud signaalide liikumine *debounceri*, 7-segmendilise indikaatori dekooderi ja juhtautomaadi vahel. Struktuurskeemil on välja toodud projekti jaoks kasutatud sisendid vasakul ja väljundid paremal. Sisenditeks on Basys 3 prototüüpimisplaadi 16 lülitit, protsessori taktsagedus ja 5 nuppu. Väljunditeks on 4-kohaline 7-segmendiline indikaator ja *debugimisel* kasutatavad LED'id.

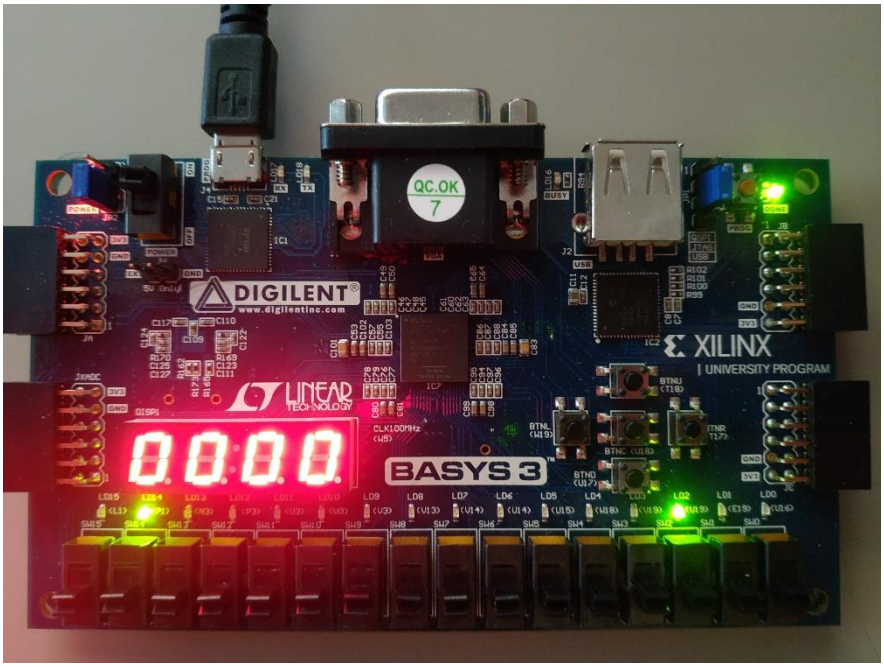
4.7 Testimine

Lõputöö käigus projekti funktsionaalsuste testimine oli teostatud täielikult Basys 3 prototüüpimisplaadi peal. Uue mooduli või muudatuse testimiseks genereeriti *bitstream*, millega programmeeriti FPGA. Testimise käigus kasutati FPGA programmeerimiseks Basys 3 prototüüpimisplaadil olevat JTAG ühendust üle *micro*-USB-pordi. JTAG ühenduse kaudu programmeerimiseks peab veenduma, et Basys 3 prototüüpimisplaadil olev „JP1" hüpiklühisti on pandud JTAG asendisse. Üle JTAG ühenduse toimub Basys 3 prototüüpimisplaadi ajutine programmeerimine ehk Basys 3 väljalülitamisel kaob programmeeritud bitstream fail prototüüpismisplaadilt. Samuti võimaldab JTAG meetodi kasutamine kiiremat testimist.



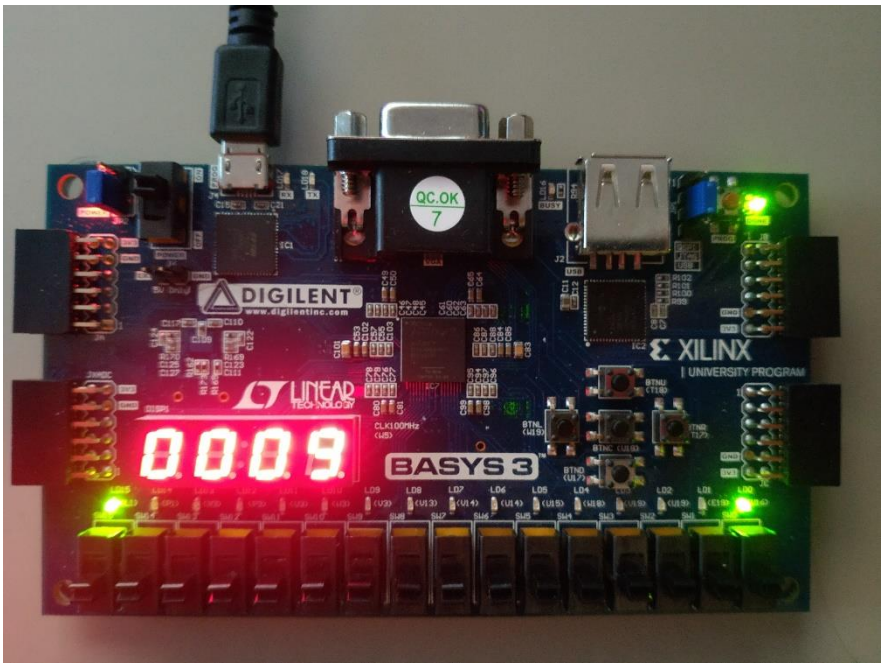
Joonis 18 Foto Basys 3 prototüüpimisplaadist peale programmeerimist

Joonisel 18 on näha, milline Basys 3 prototüüpimisplaat peale programmeerimist välja näeb. 7-segmendilisel indikaatoril kuvatakse hetkel „0000” kuna dekooderile pole väärtust antud. Juhtautomaadi ja algoritmi olekuid kuvavad LED’id on algolekutest ehk juhtautomaadil põleb kõige vasakpoolsem LED ning algoritmil põleb kõige parempoolsem LED. Tähistades vastavalt *s_idle* ja *s_wait* olekuid. Antud näite puhul on FPGA’le programmeeritud GCD algoritm ning sisestatud operandid on 27 ja 18.



Joonis 19 Foto algoritmi sammhaaval lahendamisest Basys 3 prototüüpimisplaadil

Joonisel 19 on välja toodud Basys 3 prototüüpimisplaat algoritmi sammhaaval lahendamise jooksul. Vaadates 14-ndat LED'i näeme, et juhtautomaat asub *s_step* olekus ja paremalt poolt kolmas LED kuvab algoritmi olekut *s_comp* olekus. Kuna *debugimis* lülitid pole kasutusel, siis 7-segmendilisel indikaatoril vahetulemusi ei kuvata.



Joonis 20 Foto Basys 3 prototüüpimisplaadist peale algoritmi lahendamist

Basys 3 prototüüpimisplaadi lõppolek on välja toodud joonisel 20. 7-segmendilisel indikaatoril kuvatakse vastus „0009" peale GCD algoritmi lahendamist, mis on arvude 27 ja 18 suurim ühistegur.

5 Kokkuvõte

Käesoleval bakalaureusetööl oli kaks eesmärki. Esimeseks eesmärgiks oli pakendatud Xilinx Vivado projekti loomine, mida tudengid saaksid kasutada TalTech õppeaine „Digitaalsüsteemid" teise kodutöö jooksul loodud aritmeetika-algoritmi realiseerimiseks FPGA'l. Töö teine eesmärk oli luua kasutusjuhend Vivado projekti implementeerimiseks Digilent Basys 3 prototüüpimisplaadil.

Projekti alustuseks sai loodud Vivado projekt ilma juhtautomaadita, kasutades juba olemasolevaid mooduleid 7-segmendilise indikaatori jaoks ja *debouncer'i* nuppude korrektse töö tagamiseks. Selle osa testimiseks kasutati lihtsamaid artimeetika funktsioone, kuhu oli võimalik sisestada 16-bitiseid operande. Testimine oli vajalik selleks, et veenduda kasutatud moodulite korrektses töös, enne juhtautomaadi disainimise alustamist.

Töö järgmiseks etapiks oli juhtautomaadi loogika paika panemine koostöös juhendajaga ning seejärel selle realiseerimine. Juhtautomaadi disainimiseks kasutati varasemalt juhendaja poolt loodud GCD algoritmi, mis sobis potensiaalsete probleemide lahendamiseks. Juhtautomaadi loomisel osutus raskeimaks osaks algoritmi sammhaaval läbimine ja sellele kulus lõputöö jooksul kõige rohkem aega. Juhtautomaadi testimise käigus täiendati GCD algoritmi *debugimiseks* vajaliku protsessiga ja lisati algoritmi oleku kuvamine LED'idel.

„Digitaalsüsteemid" teise kodutöö realiseerimisel kasutati algoritmi: jagamine jäägi taastamisega. Vajalikud graafskeemid algoritmi loogika kirjeldamiseks olid leitavad loengumaterjalides ja nende põhjal sai loodud ka VHDL kood. Kodutöö realiseerimise käigus tuli välja mitu võimalikku parandust, mida saaks pakendatud projektile lisada. Näiteks, võiks implementeerida *generic* väärtuste kasutuse, mis muudaks pakendatud projekti paindlikumaks ja kasutajasõbralikumaks. Võimaldades kasutajal ära määrata sünteesitava algoritmi operandide ja tulemuse pikkused. Hetkel tuleb selleks käsitsi muuta projektis kasutusel olevate vektorite pikkuseid.

Viimase osana loodi kasutusjuhend, kus on kirjeldatud kuidas Vivado projekti lisada algoritm ja seejärel seda rakendada Digilent Basys 3 prototüüpimisplaadil. Samuti on juhendis lahti kirjeldatud Basys 3 lülitite, nuppude ja kuvamise loogika.

Kasutatud kirjandus

[1] „Basys 3 Reference Manual” [Võrgumaterjal]. Saadaval: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual (04.01.2021)

[2] S. Arar, „What Is VHDL? Getting Started with Hardware Description Language for Digital Circuit Design” [Võrgumaterjal]. Saadaval: https://www.allaboutcircuits.com/technical-articles/hardware-description-langauge-getting-started-vhdl-digital-circuit-design/ (04.01.2021)

[3] IEEE, „IEEE Standard VHDL Language Reference Manual” [Võrgumaterjal]. Saadaval: https://ieeexplore.ieee.org/document/4772740 (04.01.2021)

[4] „Vivado Design Suite HLx Editions – Acceleration High Level Design” [Võrgumaterjal]. Saadaval: https://www.xilinx.com/products/design-tools/vivado.html#overview (04.01.2021)

[5] „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 (04.01.2021)

[6] „VHDL code for debouncing buttons on FPGA” [Võrgumaterjal]. Saadaval: https://www.fpga4student.com/2017/08/vhdl-code-for-debouncing-buttons-on-fpga.html (04.01.2021)

[7] S. Larson, „Debounce Logic Circuit (with VHDL example)” [Võrgumaterjal]. Saadaval: https://www.digikey.com/eewiki/pages/viewpage.action?pageId=4980758 (04.01.2021)

[8] P. Ellervee, „Suurima ühisteguri leidmine (GCD)” [Võrgumaterjal]. Saadaval: http://mini.pld.ttu.ee/~lrv/gcd/ (04.01.2021)

[9] „Debouncing” [Võrgumaterjal]. Saadaval: https://blog.mbedded.ninja/electronics/circuit-design/debouncing/ (04.01.2021)

[10] „ARVUTUSALGORITM ja selle MODELEERIMINE (simuleerimine) kirjelduskeeles VHDL” [Võrgumaterjal]. Saadaval: http://www.diskmat.ee/algoritm.htm (04.01.2021)

[11] M. Kruus, „Jagamine” [Võrgumaterjal]. Saadaval: https://ained.ttu.ee/pluginfile.php/30363/mod_resource/content/5/Jagamine.pdf (04.01.2021)

[12] A. Juškov, „Aritmeetika algoritmide prototüüpimine FPGA’l”, Tallinn, 2019

[13] R. Wang, „Multiplication and Division” [Võrgumaterjal]. Saadaval: http://fourier.eng.hmc.edu/e85_old/lectures/arithmetic_html/node8.html

[14] „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/

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

Mina, Gustav Kirsipuu

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Aritmeetika-algoritmide realiseerimine FPGA-l", mille juhendaja on Peeter Ellervee.
   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.

04.01.2021

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 – Vivado projekti ja kodutöö VHDL kood

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

entity top is
    Port ( dipSW                  : in STD_LOGIC_VECTOR(15 downto 0);
           btnL, btnR, btnC, btnU : in STD_LOGIC;
           clk                    : in STD_LOGIC;
           reset                  : in STD_LOGIC;
           LED_out                : out STD_LOGIC_VECTOR(6 downto 0);
           Anode_Activate         : out STD_LOGIC_VECTOR(3 downto 0);
           led                    : out STD_LOGIC_VECTOR(15 downto 0)
           );
end top;


architecture Behavioral of top is
    type state_type is (s_idle, s_step, s_step1, s_run, s_ready);
    signal state, next_state   : state_type;
    signal regA, regB, dbg    : STD_LOGIC_VECTOR(15 downto 0);
    signal regC : STD_LOGIC_VECTOR(15 downto 0);
    signal resultGCD      : STD_LOGIC_VECTOR(15 downto 0);
    signal debounced_pb1       : STD_LOGIC;
    signal debounced_pb2       : STD_LOGIC;
    signal debounced_btnC      : STD_LOGIC;
    signal debounced_btnU      : STD_LOGIC;
    signal clkAlgo, manual     : STD_LOGIC;
    signal start               : STD_LOGIC;
    signal rdy                 : STD_LOGIC := '0';

    component debouncingButton
        Port ( button          : in STD_LOGIC;
               clk             : in STD_LOGIC;
               debouncedButton : out STD_LOGIC
               );
    end component;

    component decoder
        Port ( clk             : in STD_LOGIC;
               reset           : in STD_LOGIC;
               displayedNumber : in STD_LOGIC_VECTOR;
               LED_out         : out STD_LOGIC_VECTOR(6 downto 0);
               Anode_Activate  : out STD_LOGIC_VECTOR(3 downto 0)
               );
```

```
    end component;

    component gcd
        Port ( xi, yi          : in UNSIGNED(15 downto 0);
               clk             : in STD_LOGIC;
               start, reset    : in STD_LOGIC;
               sel             : in STD_LOGIC_VECTOR(1 downto 0);
               xo, dbg         : out UNSIGNED(15 downto 0);
               rdy             : out STD_LOGIC;
               led             : out STD_LOGIC_VECTOR(15 downto 0)
               );
    end component;

    component division
        Port ( xi, yi            : in STD_LOGIC_VECTOR(7 downto 0);
               clk, start, reset : in STD_LOGIC;
               sel               : in STD_LOGIC_VECTOR(1 downto 0);
               dbg               : out STD_LOGIC_VECTOR(7 downto 0);
               led               : out STD_LOGIC_VECTOR(15 downto 0);
               rdy               : out STD_LOGIC
               );
    end component;

begin

    save1 : debouncingButton PORT MAP
        ( button => btnL,
          clk => clk,
          debouncedButton => debounced_pb1
          );

    save2 : debouncingButton PORT MAP
        ( button => btnR,
          clk => clk,
          debouncedButton => debounced_pb2
          );

    start1 : debouncingButton PORT MAP
        ( button => btnC,
          clk => clk,
          debouncedButton => debounced_btnC
          );

    start2 : debouncingButton PORT MAP
        ( button => btnU,
          clk => clk,
          debouncedButton => debounced_btnU
          );

    display : decoder PORT MAP
        ( clk => clk,
```

```
        reset => reset,
        displayedNumber => regC,
        LED_out => LED_out,
        Anode_Activate => Anode_Activate
        );


    process(clk, debounced_pb1, debounced_pb2, debounced_btnC,
debounced_btnU)
    begin
        if(rising_edge(clk)) then
            if(debounced_pb1 = '1') then
                regA <= dipSW(15 downto 0);
                regC <= dipSW;
            elsif(debounced_pb2 = '1') then
                regB <= dipSW(15 downto 0);
                regC <= dipSW;
            else
                if(dipSW(15) = '1') then
                    regC <= dbg;
                elsif(debounced_btnC = '1') then
                    regC <= resultGCD;
                elsif(debounced_btnU = '1') then
                    regC <= resultGCD;
                elsif(reset = '1') then
                    regC <= "0000000000000000";
                end if;
            end if;
        end if;
    end process;

    process(clk, reset)
    begin
        if(reset = '1') then
            state <= s_idle;
        elsif(rising_edge(clk)) then
            state <= next_state;
        end if;
    end process;

    process(state, debounced_btnC, rdy, btnU)
    begin
        case state is
            when s_idle =>
                manual <= '1';
                start <= '1';
                led(15 downto 10) <= "100000";
                if(debounced_btnC = '1') then
                    next_state <= s_run;
                elsif(btnU = '1') then
                    next_state <= s_step;
```

```
            else
                next_state <= s_idle;
            end if;

        when s_step =>
            manual <= '1';
            start <= '0';
            led(15 downto 10) <= "010000";
            if(rdy = '1') then
                next_state <= s_ready;
            elsif(btnU = '1') then
                next_state <= s_step1;
            elsif(debounced_btnC = '1') then
                next_state <= s_run;
            else
                next_state <= s_step;
            end if;

        when s_step1 =>
            manual <= '1';
            start <= '0';
            led(15 downto 10) <= "001000";
            if(btnU = '0') then
                next_state <= s_step;
            elsif(debounced_btnC = '1') then
                next_state <= s_run;
            else
                next_state <= s_step1;
            end if;

        when s_run =>
            manual <= '0';
            start <= '0';
            led(15 downto 10) <= "000100";
            if(rdy = '1') then
                next_state <= s_ready;
            else
                next_state <= s_run;
            end if;

        when s_ready =>
            start <= '1';
            manual <= '0';
            led(15 downto 10) <= "000010";
            if(rdy = '0') then
                next_state <= s_idle;
            else
                next_state <= s_ready;
            end if;
    end case;
end process;
```

```
    process(clk, manual, debounced_btnU)
    begin
        if(manual = '0') then
            clkAlgo <= clk;
        else
            clkAlgo <= debounced_btnU;
        end if;
    end process;


    GCD_algoritm : gcd PORT MAP
        ( xi => unsigned(regA),
          yi => unsigned(regB),
          clk => clkAlgo,
          start => start,
          reset => reset,
          sel => dipSW(14 downto 13),
          std_logic_vector(xo) => resultGCD,
          std_logic_vector(dbg) => dbg,
          rdy => rdy,
          led => led
          );

--    restoring_division : division PORT MAP
--        ( xi => regA,
--          yi => regB,
--          clk => clkAlgo,
--          start => start,
--          reset => reset,
--          sel => dipSW(14 downto 13),
--          dbg => dbg,
--          led => led,
--          rdy => rdy
--          );

end Behavioral;
```

Joonis 21 Vivado projekti top.vhd VHDL kood

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

entity division is
    port ( xi, yi            : in STD_LOGIC_VECTOR(7 downto 0);
           clk, start, reset : in STD_LOGIC;
           sel               : in STD_LOGIC_VECTOR(1 downto 0);
           dbg               : out STD_LOGIC_VECTOR(7 downto 0);
           led               : out STD_LOGIC_VECTOR(15 downto 0);
           rdy               : out STD_LOGIC);
end division;

architecture Behavioral of division is
    type state_type is (S_wait, S_start, S_sign, S_neg, S_pos, S_ready);
    signal state, next_state: state_type;
    signal Q, Qn : STD_LOGIC_VECTOR(15 downto 0);
    signal Diff  : STD_LOGIC_VECTOR(8 downto 0);
    signal Count : STD_LOGIC_VECTOR(3 downto 0);
    signal set_rdy, count_add, count_rst : STD_LOGIC;

begin

    Diff <= ('0' & Q(14 downto 7)) - ('0' & yi);

    process(state, start, Q, xi, Diff, Count)
    begin
        set_rdy <= '0';
        count_add <= '0';
        count_rst <= '0';
        next_state <= state;
        Qn <= Q;
        case state is
            when S_wait =>
                led(5 downto 0) <= "000001";
                if(start = '0') then
                    Qn <= "00000000" & xi;
                    next_state <= S_start;
                end if;

            when S_start =>
                led(5 downto 0) <= "000010";
                if(Count(3) = '0') then
                    next_state <= S_sign;
                else
                    next_state <= S_ready;
                end if;

            when S_sign =>
                led(5 downto 0) <= "000100";
```

```
            if(Diff(8) = '1') then
                next_state <= S_neg;
            else
                next_state <= S_pos;
            end if;

        when S_neg =>
            led(5 downto 0) <= "001000";
            count_add <= '1';
            Qn <= Q(14 downto 0) & '0';
            next_state <= S_start;

        when S_pos =>
            led(5 downto 0) <= "010000";
            count_add <= '1';
            Qn <= Diff(7 downto 0) & Q(6 downto 0) & '1';
            next_state <= S_start;

        when S_ready =>
            led(5 downto 0) <= "100000";
            set_rdy <= '1';
            count_rst <= '1';
            next_state <= S_wait;
    end case;
end process;

process
begin
    wait on clk until clk = '1';
    if(reset = '1') then
        state <= S_wait;
    else
        state <= next_state;
    end if;
    if(count_rst = '1') then
        Count <= "0000";
    elsif(count_add = '1') then
        if(Count(3) = '0') then
            Count <= Count + 1;
        end if;
    end if;
    Q <= Qn;
    rdy <= set_rdy;
end process;

process(sel, Q, Count, Diff)
begin
    case sel is
        when "00" => dbg <= Q(7 downto 0);
        when "01" => dbg <= Q(15 downto 8);
        when "11" => dbg <= "0000" & Count;
```

```
            when "10" => dbg <= "0000000" & Diff(8);
            when others => dbg <= (others => '0');
        end case;
    end process;
end Behavioral;
```

Joonis 22 Jäägi taastamisega jagamise VHDL kood

Lisa 3 – Vivado projekti kasutusjuhend prototüüpimiseks kasutades Basys 3

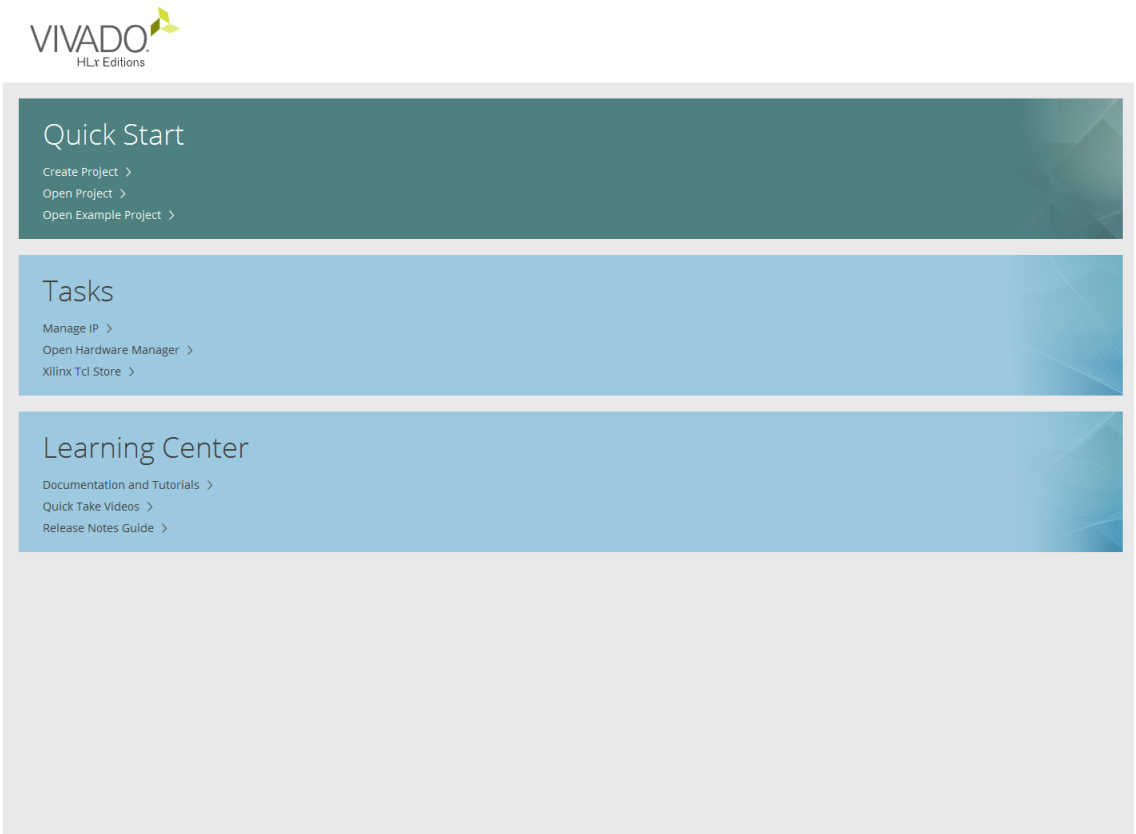
Basys 3 kasutades on prototüüpimiseks vajalik Xilinx Vivado tarkvara. Saadaval: https://www.xilinx.com/support/download.html

Vivado allalaadimiseks on vajalik ennast kasutajaks registreerida, tudengitel on võimalik saada tasuta litsents õppimise eesmärgil. Proovimiseks on neil olemas ka 30 päevane litsents Vivado Design Suite HL versioonile.

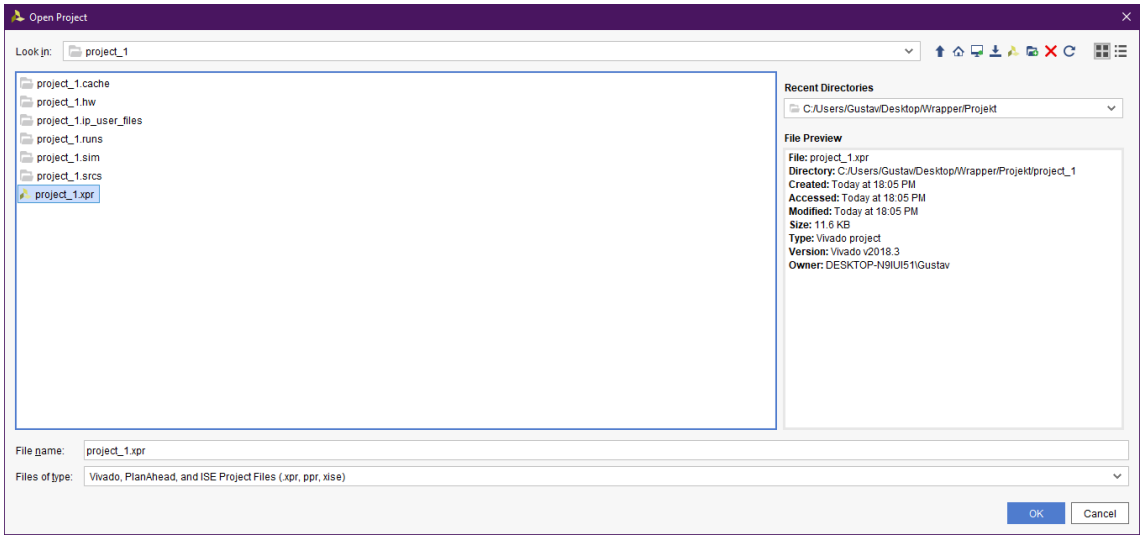
Projekti kood on saadaval:

Projekti lisamine ja uue faili loomine

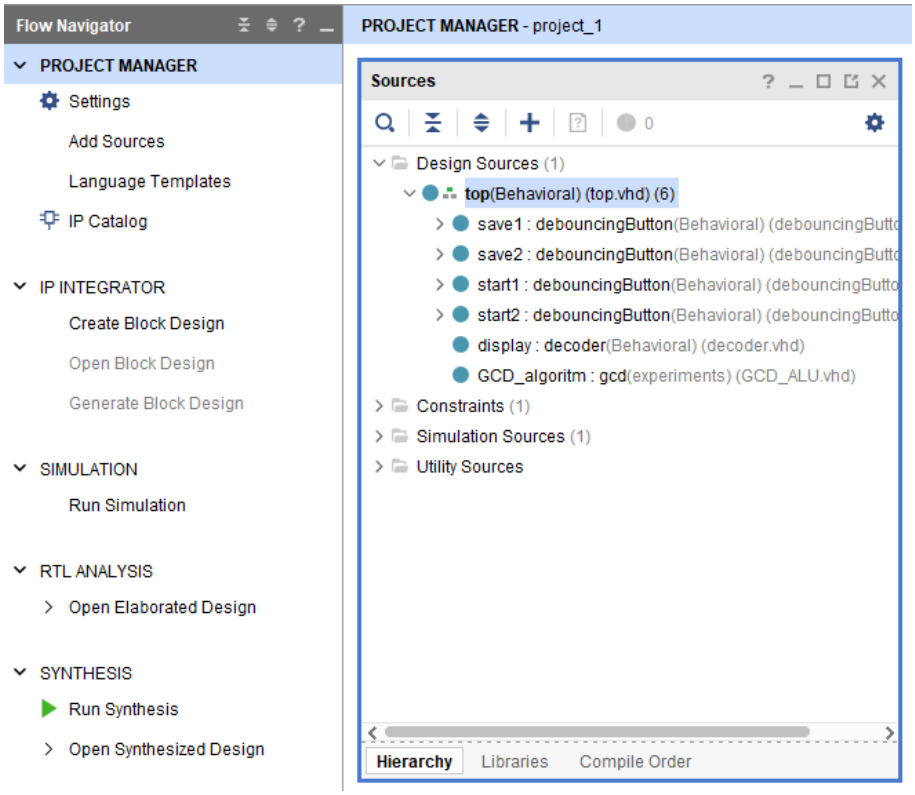
Vivado avamisel tuleb ette avakuva, kus tuleb projekti lisamiseks valida „Open Project”.



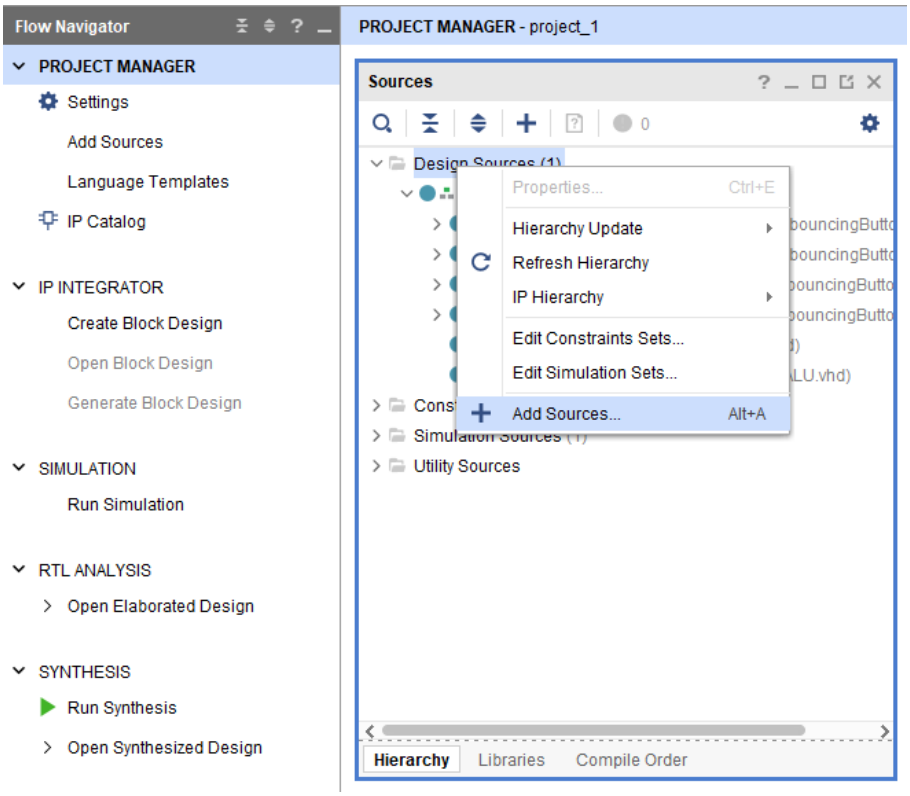
Avanenud aknas tuleb liikuda kausta, kuhu alla laetud projekt on salvestatud. Projekti kaustast tuleb üles leida fail nimega „project_1.xpr” ja seejärel vajutada „OK”.

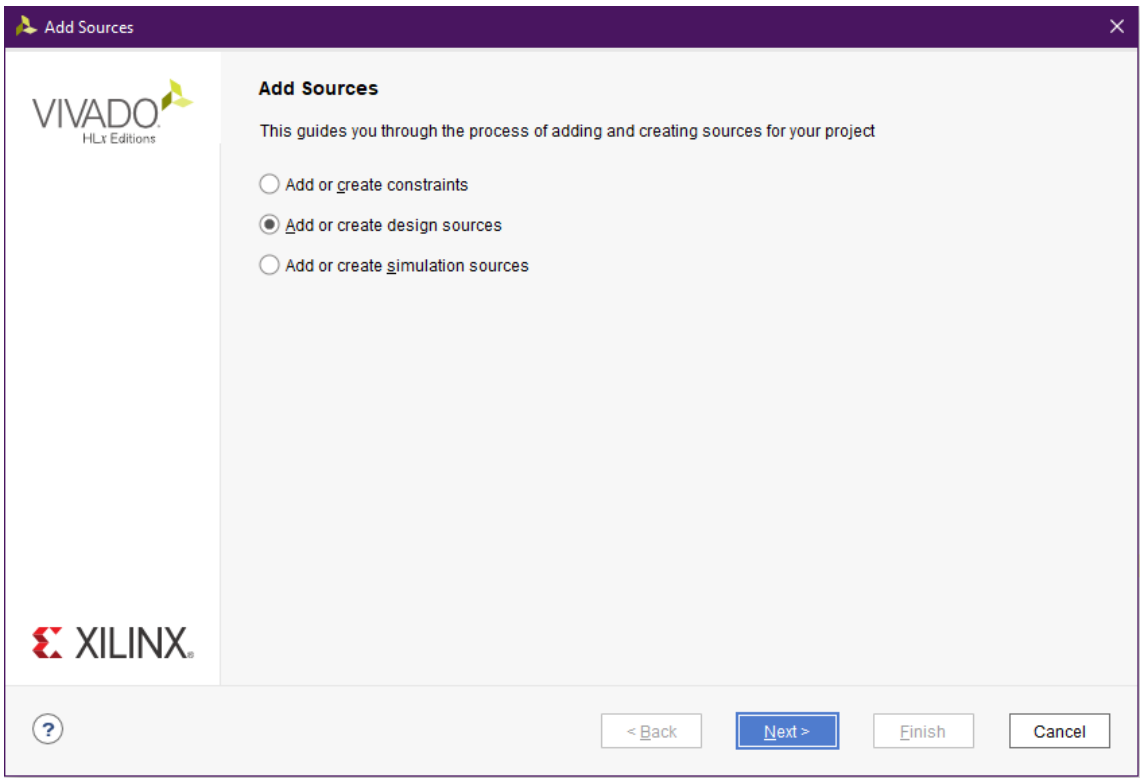


Projekti avamisel tuleb ette uus aken, kus on näha kõik kasutusel olevad failid.

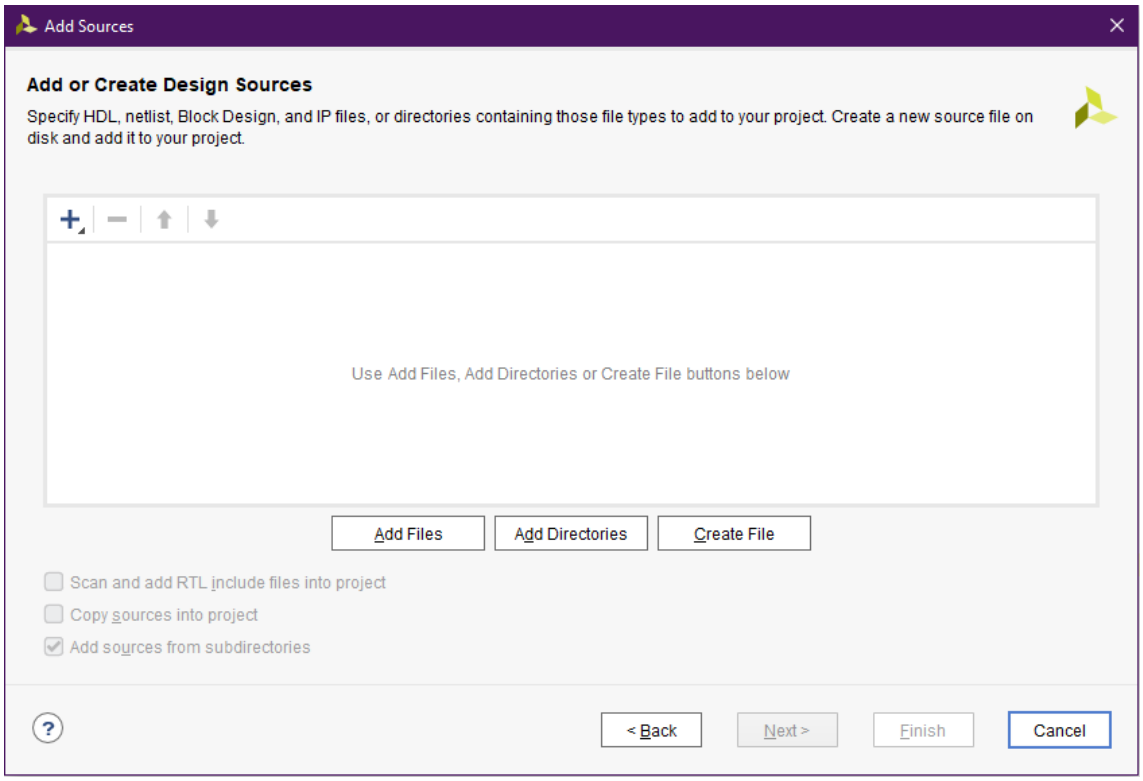


Algoritmi lisamiseks tuleb „Sources” aknas parema hiire klahviga vajutada „Design Sources” peale ja valida sealt „Add Sources...”.



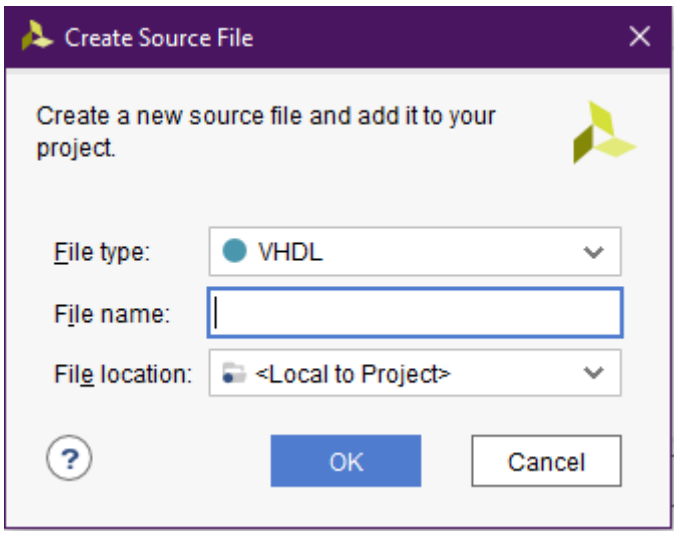


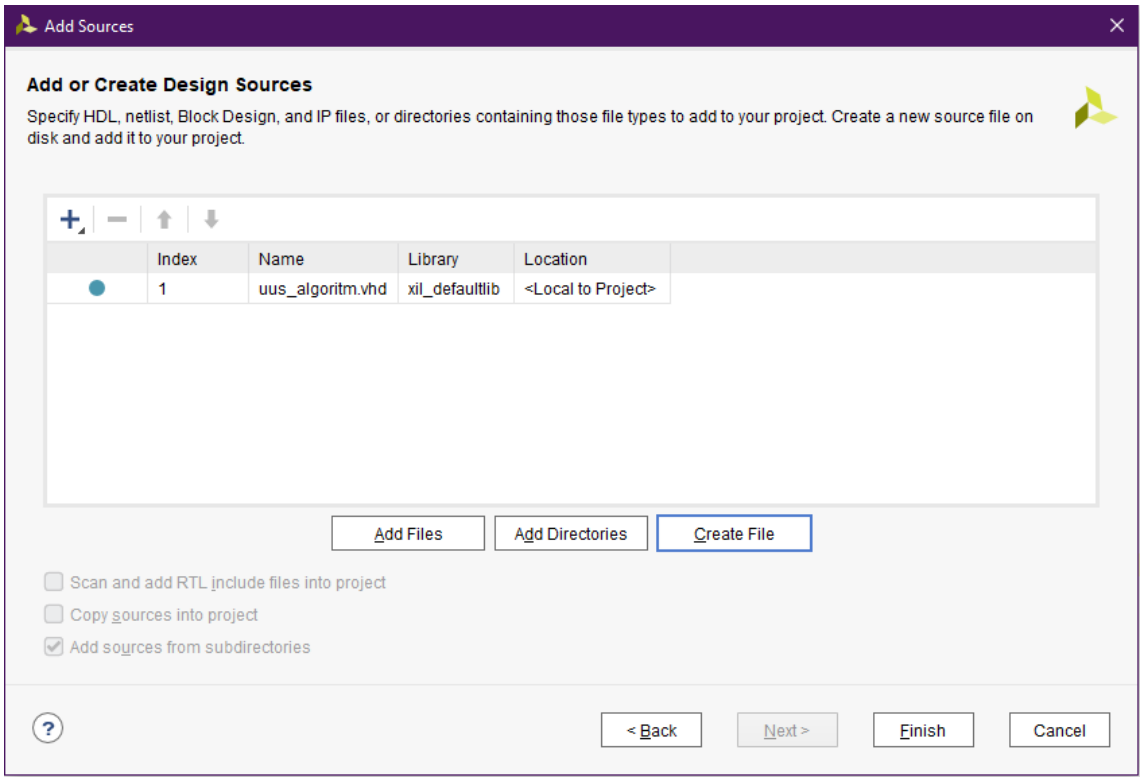
Uues aknas tuleb valida „Add or create design sources” ja vajutada nupule „Next >”.



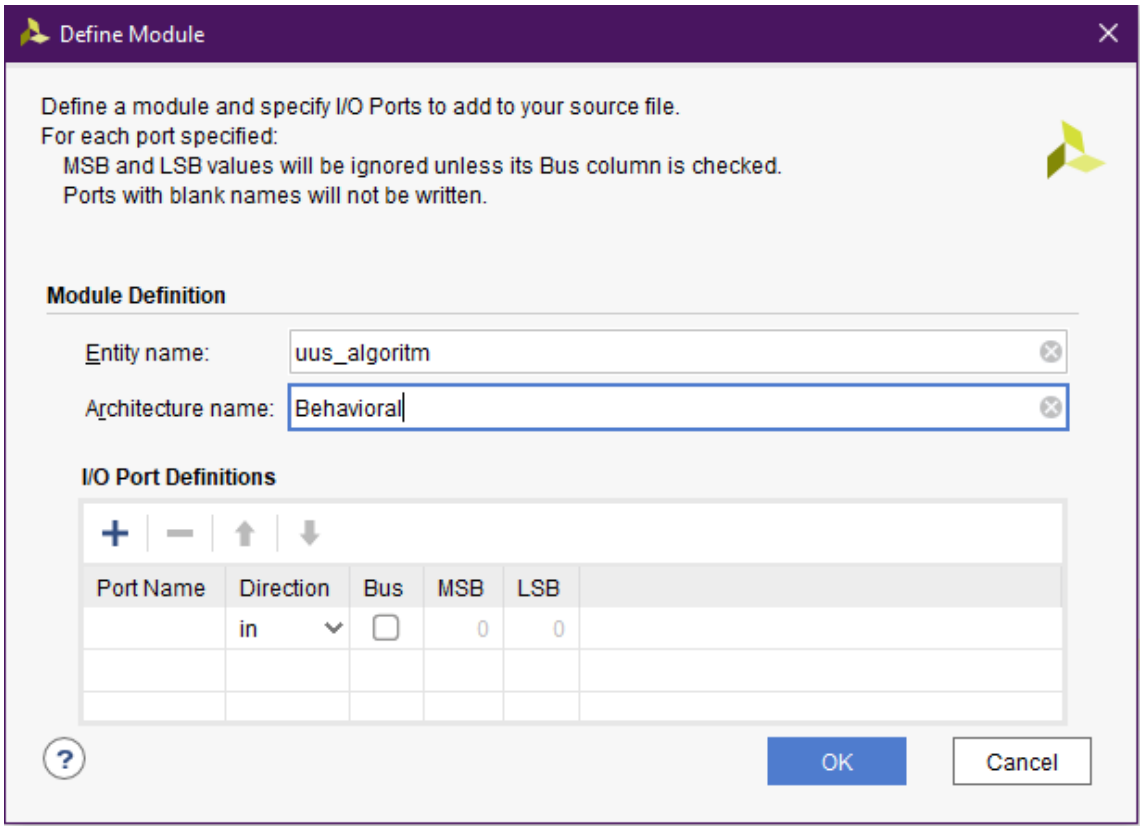
Seejärel tuleb luua uus fail vajutades nupule „Create File”. Avanenud aknas vaadata, et „File type” oleks VHDL ja sisestada uuele failile nimi. Vajutada nupule „OK”.

NB! Faili nimes ei tohi olla tühikuid, soovitatav kasutada ‘_’.

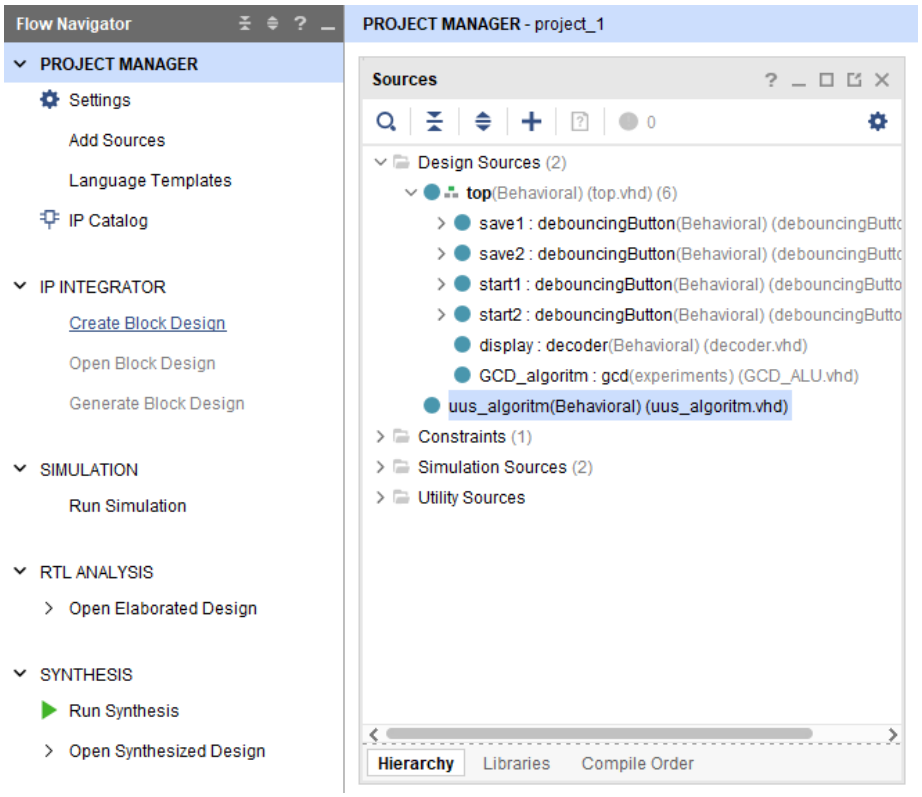




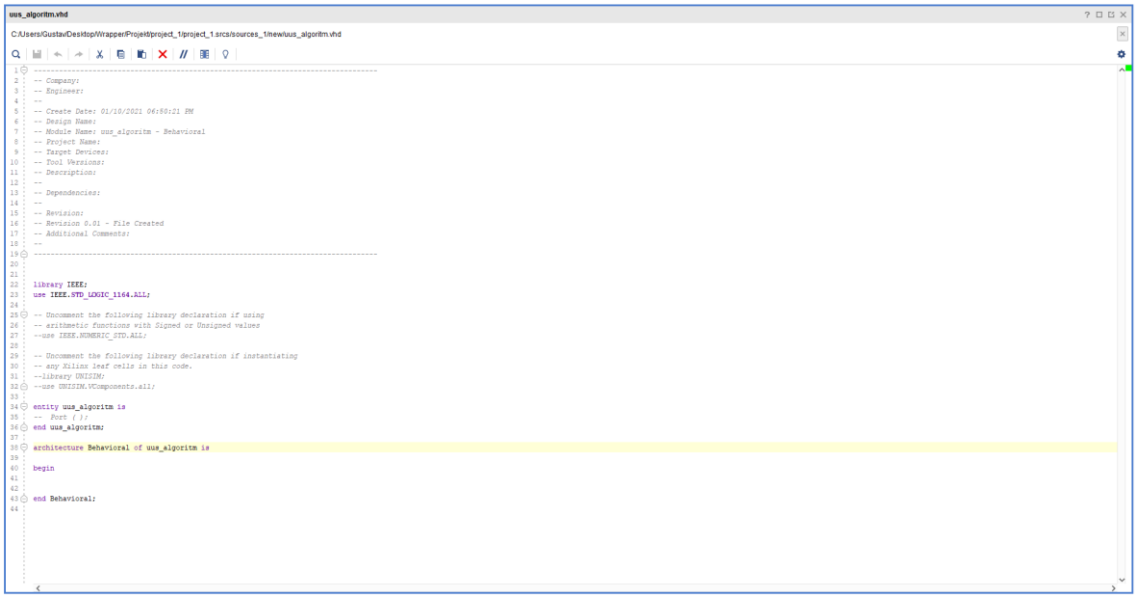
Peale uue faili loomist kuvatakse see „Add Sources” aknas. Vajutada nupule „Finish”.



Vivado’s uue faili loomisel antakse võimalus defineerida „Entity name”, mis määrab mooduli nime VHDL koodis. Samuti saab siin lisada algoritmi moodulile sisendid ja väljundid, kuid seda on võimalik ka hiljem VHDL koodis kirjutada.



Lisatud fail tekib „Project Manager” vaates „Design Sources” alla, kus on võimalik see topeltklõpsuga avada.



Avanenud aknas saab kirjutada VHDL koodi algoritmi jaoks.

Soovitatavad lisad algoritmile

Debugimise eesmärgil on soovitatav algoritmile lisada järgnev protsess:

```
process(sel, x, y, xo_bf)
begin
  case sel is
      when "00" => dbg <= xo_bf;
      when "01" => dbg <= x;
      when "10" => dbg <= y;
      when others => dbg <= (others => '0');
  end case;
end process;
```

Antud protsess on vajalik algoritmi vahetulemuste kuvamiseks kasutades Basys 3 lüliteid 15 kuni 13. Muutujaid ‘xo_bf’, ‘x’ ja ‘y’ tuleb muuta vastavalt soovitud vahetulemustele. Kokku on võimalik kuvada 4 erinevat vahetulemust kasutades väärtusi “00”, “01”, “10” ja “11”. Väärtuste kontrollimiseks on Basys 3 *switchid* 13 ja 14 ning *switch* 15 lülitab sisse *debug* režiimi.

Samuti on soovitatav lisada algoritmi olekutele LED väärtused, et jälgida algoritmi tööd sammhaaval lahendamise käigus.

Esimesse olekusse lisada:

```
led(5 downto 0) <= "000001";
```

Teise olekusse:

```
led(5 downto 0) <= "000010";
```

Projektis on näitena olemas GCD algoritm, mille pealt saab vaadata kuidas *debug* meetodeid on rakendatud.

Algoritmi komponendi lisamine top.vhd faili

Uue algoritmi loomisel tuleb kirjeldada sisend ja väljund signaalid *entity's*. Allpool on toodud näide GCD algoritmi põhjal.

```
entity gcd is
  port (xi, yi : in unsigned(15 downto 0);
        clk    : in STD_LOGIC;
        start  : in STD_LOGIC;
        reset  : in STD_LOGIC;
        sel    : in STD_LOGIC_VECTOR(1 downto 0);
        xo,dbg : out unsigned(15 downto 0);
        rdy    : out STD_LOGIC;
        led    : out STD_LOGIC_VECTOR(15 downto 0));
end gcd;
```

Soovitatav on kasutada signaali tüüpe: *unsigned*, *std_logic_vector(n downto 0)* ja *std_logic*.

Algoritmi kasutamiseks projektis tuleb see kirjeldada komponendina top.vhd failis:

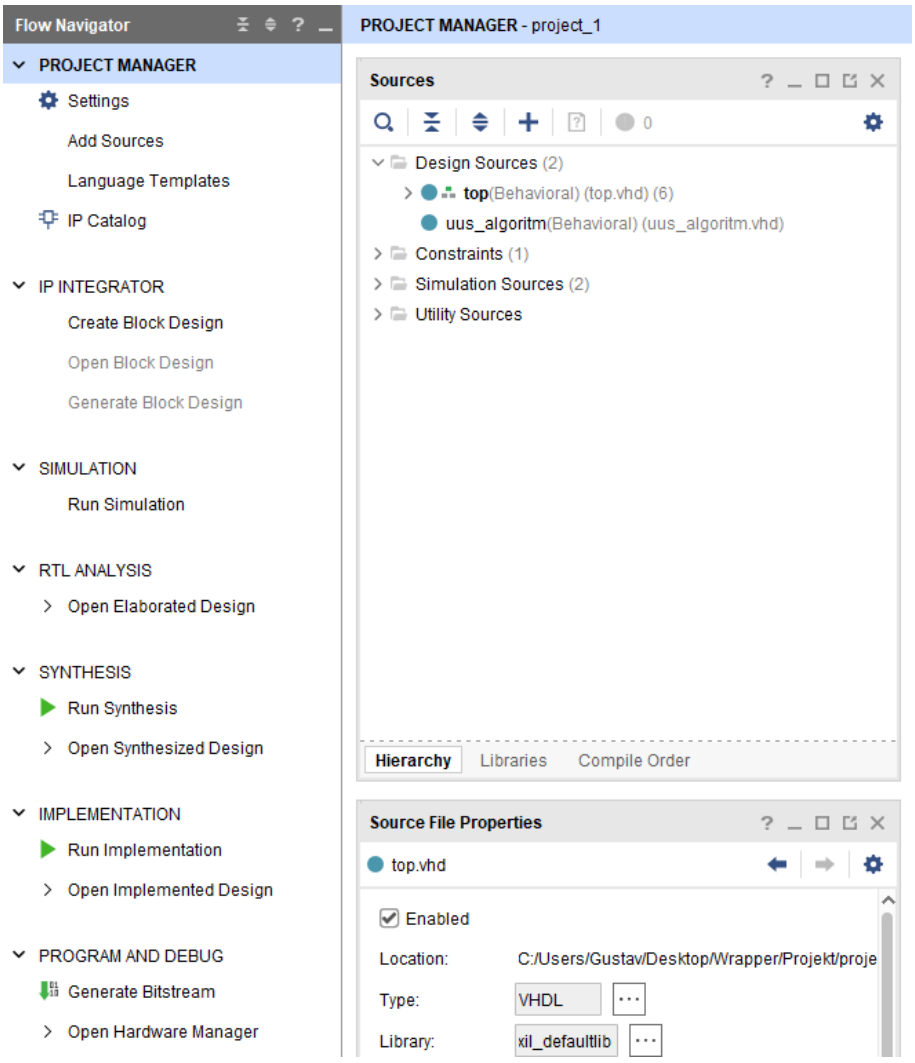
```
    component gcd
        Port ( xi, yi          : in UNSIGNED(15 downto 0);
               clk             : in STD_LOGIC;
               start, reset    : in STD_LOGIC;
               sel             : in STD_LOGIC_VECTOR(1 downto 0);
               xo, dbg         : out UNSIGNED(15 downto 0);
               rdy             : out STD_LOGIC;
               led             : out STD_LOGIC_VECTOR(15 downto 0)
               );
    end component;
```

Peale komponendi lisamist tuleb teha signaalidele „PORT MAP", kirjeldatakse signaalide liikumine komponendi ja top.vhd vahel:

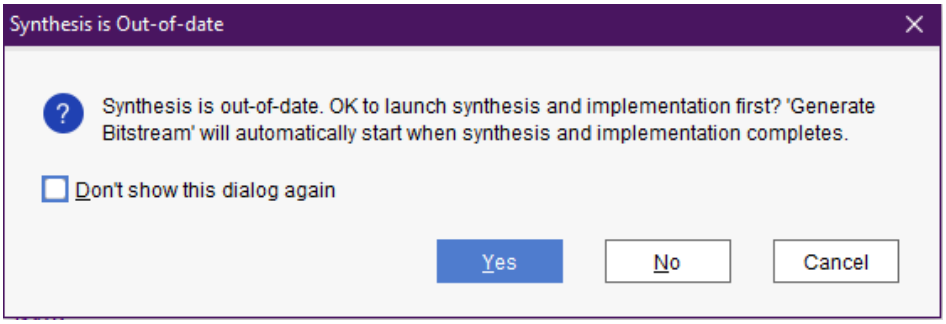
```
    GCD_algoritm : gcd PORT MAP
        ( xi => unsigned(regA),
          yi => unsigned(regB),
          clk => clkAlgo,
          start => start,
          reset => reset,
          sel => dipSW(14 downto 13),
          std_logic_vector(xo) => resultGCD,
          std_logic_vector(dbg) => dbg,
          rdy => rdy,
          led => led
          );
```

Projekti programmeerimine Basys 3 peale

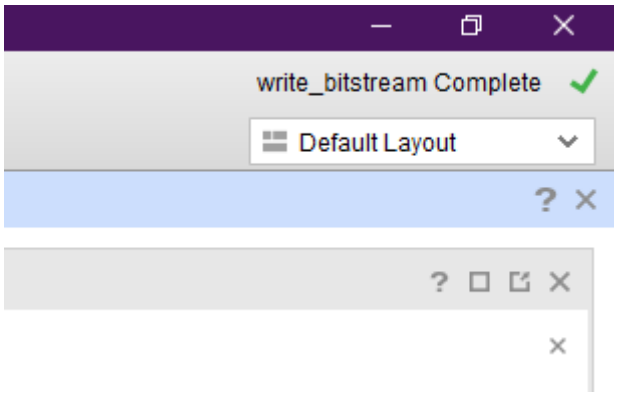
Kui algoritm on valmis ja on soov seda testima hakata, siis tuleb Vivado projektile genereerida *bitstream*. Selle jaoks on Vivado vasakpoolses menüüs „Program and Debug" all valik „Generate Bitstream".



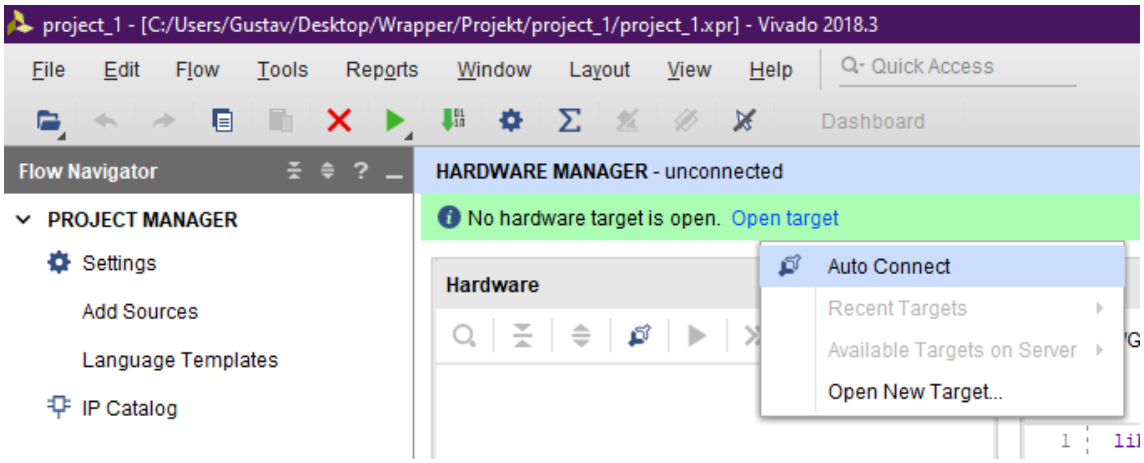
Peale „Generate Bitstream" nupule vajutamist annab Vivado teada, et projekti süntees on aegunud. Vajutada nupule „Yes".



Vivado hakkab kõigepealt looma uut sünteesi ja seejärel genereerib *bitstreami* Basys 3 programmeerimiseks. Üleval paremas nurgas on näha, kui kaugel Vivado *bitstreami* genereerimisega on. Lõpetades kuvatakse sinna „write_bitstream Complete".

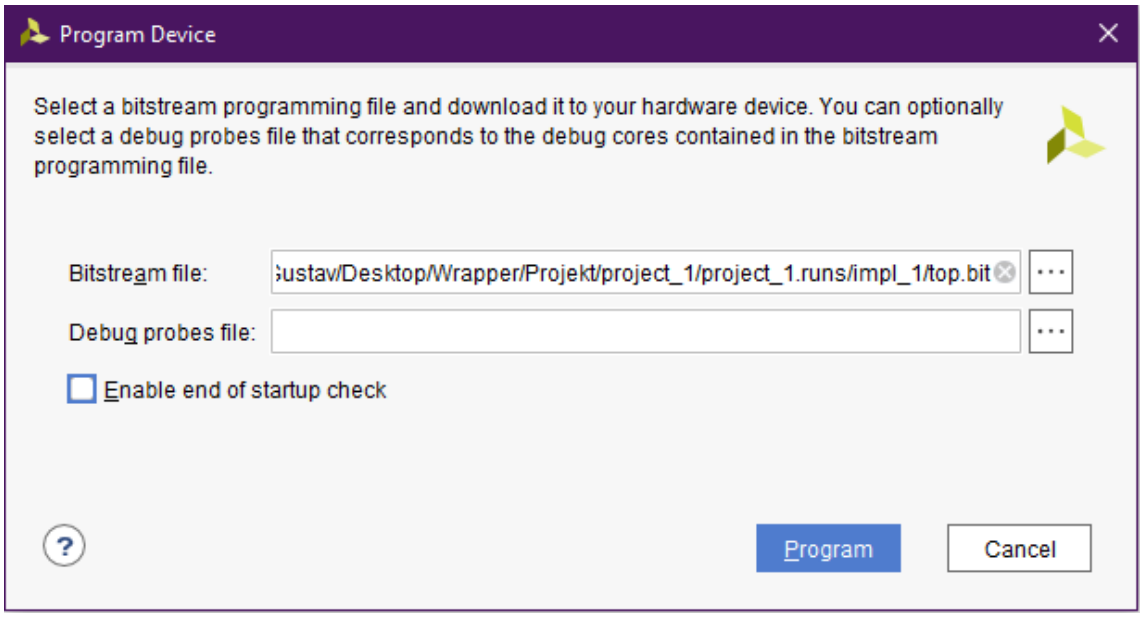


Peale bitstreami genereerimist tuleb Basys 3 ühendada Vivado'ga. Selleks tuleb Basys 3 prototüüpimisplaat ühendada micro-USB kaudu arvutisse ja veenduda, et „JP1" hüpiklühisti on ühendatud valikule JTAG. Seejärel võib Basys 3 plaadi sisse lülitada ja avada Vivado „Hardware Manager" kasutades vasakpoolset menüüd.



Kui „Hardware Manager" on avatud, siis vajutada nupule „Open target" ja rippmenüüst valida „Auto Connect". Basys 3 esmakordsel ühendamisel võib see tavalisest rohkem aega võtta erinevate driverite paigaldamiseks.

Peale Basys 3 ühendamist tekib rohelisele ribale uus valik „Program device", mida vajutades tekib uus aken *bitstreami* valimiseks.

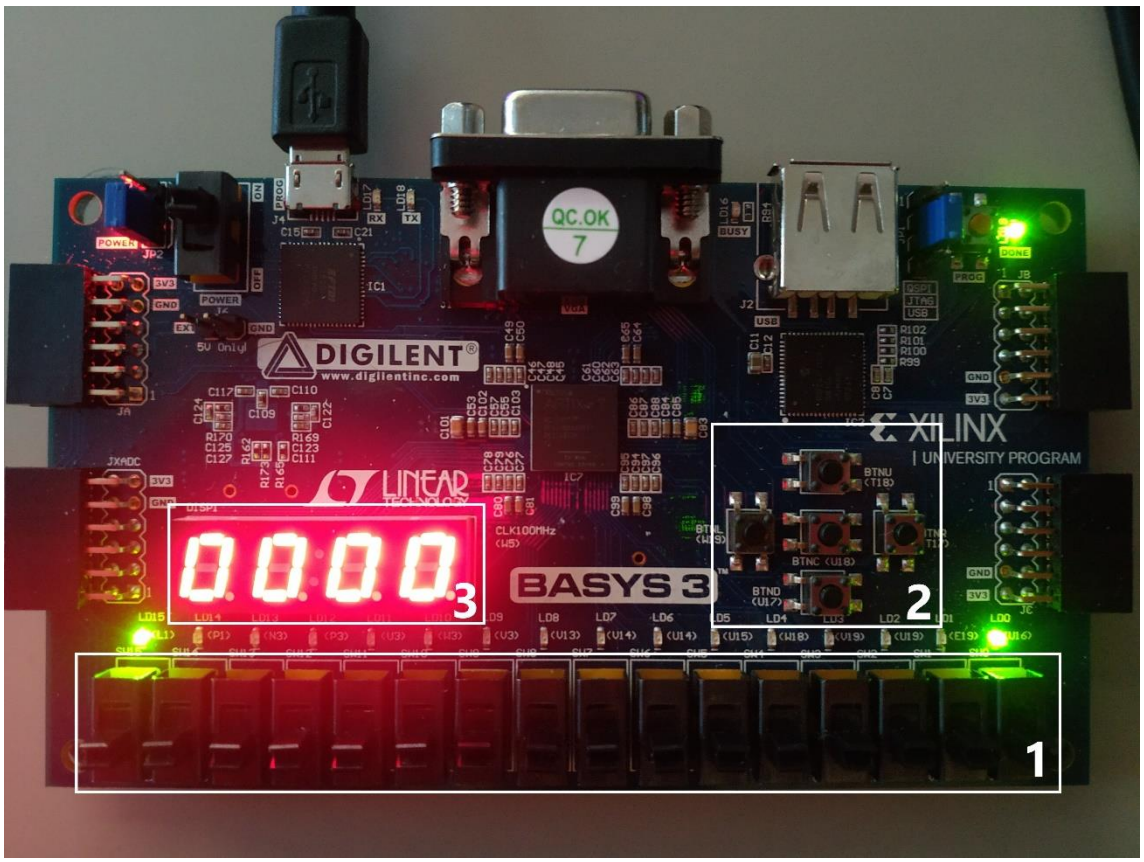


Vivado valib ise viimati genereeritud *bitstreami* faili ehk siis tuleb lihtsalt vajutada nupule „Program”. Programmeerimine võtab natukene aega, aga peale seda on Basys 3 testimiseks valmis.

Projekti funktsionaalsused kasutades Basys 3

Pildil on välja toodud Basys 3 ja tehtud marked põhiliste sisend ja väljund seadmete jaoks.

1. 16 *switchi*
2. 5 nuppu
3. 4-kohaline 7-segmendiline indikaator



Algoritmile sisestavate operandide väärtused tuleb määrata kasutades Basys 3 *switche*, iga lüliti tähistab ühte numbrikohta 16-bitises kahendarvus. Pildil on kõik lülitid alumises asendis ehk väärtus on „0000000000000000”.

Väärtuste sisestamiseks tuleb kasutada vasakut ja paremat nuppu. Esimese operandi sisestamiseks tuleb lülitid panna soovitud väärtusele vastavatesse asendisse ja seejärel vajutada vasakpoolsemat nuppu, teise väärtuse sisestamiseks tuleb kasutada parempoolset nuppu. Nupule vajutades kuvatakse sisestatud väärtus 7-segmendilisel indikaatoril 16-nd süsteemis.

Algoritmi käivitamiseks on 2 erinevat võimalust:

1. Keskmist nuppu vajutades antakse algoritmile Basys 3 taktsagedus ehk algoritm lahendatakse lõpuni ja kuvatakse vastus 7-segmendilisel indikaatoril.

2. Ülemist nuppu vajutades toimub algoritmi sammhaaval lahendamine. Algoritmile antakse iga nupu vajutusega ühe taktitsükkli signaal ehk algoritmi lahendatakse ühe oleku kaupa.

Sammhaaval lahendamise käigus on võimalik jälgida algoritmi vahetulemusi, kui VHDL koodi sai lisatud selleks vajalik protsess (vaata „Soovitatavad lisad algoritmile" peatükki).

Algoritmi on võimalik viia algolekusse kasutades alumist nuppu, mis annab *reset* signaali. Selleks hoida all alumist nuppu ja seejärel vajutada ülemist nuppu 1 korra ning peale seda lahti lasta alumine nupp.