Frontend dio internet aplikacije za simuliranje rada PicoBlaze procesora

Bernatović, Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij računarstva

FRONTEND DIO INTERNET APLIKACIJE ZA SIMULIRANJE RADA PICOBLAZE PROCESORA

Završni rad

Matej Bernatović

Osijek, 2024.

SADRŽAJ

1.UVOD

Svrha ovog završnog rada je razvoj React.js sučelja za simulaciju rada PicoBlaze procesora implementiranog na Nexys A7 FPGA(engl: *Field-Programmable Gate Array*) maketi.[1][2][3] [4][5] Glavni cilj rada je omogućiti korisnicima upravljanje procesorom putem intuitivnog grafičkog sučelja koje koristi SVG(engl. *Scalable Vector Graphics*) sliku za simulaciju fizičkih elemenata makete, poput LED dioda, RGB LED-a i tipkala. [6] Ova interakcija korisnicima omogućava realističan doživljaj rada s ulaznim i izlaznim uređajima FPGA ploče. Jedan od ključnih elemenata ovog rada je Monaco Editor, napredni tekstualni uređivač koji pruža funkcionalnosti poput isticanja sintakse, automatskog dovršavanja sintakse i umetanja točaka prekida (engl. *break-point*). [7] Ove značajke su neophodne za efikasno pisanje i uređivanje PicoBlaze asemblerskog koda. Korištenje Monaco Editora omogućava korisnicima pisanje i testiranje koda unutar istog sučelja, uz napredne mogućnosti za kontrolu izvršavanja programa. Monaco Editor podržava operacije kao isticanje sintakse i automatsko dovršavanje(engl. *autocomplete*) sintakse koje treba implementirati u projektni zadatak. Pored toga, u sklopu sučelja potrebno je implementirati kontrolu programa koja omogućava kreiranje, učitavanje i preuzimanje programa na računalo, kao i pokretanje programa s mogućnošću pauziranja kod točaka prekida. Na ovaj način korisnik ima punu kontrolu nad simulacijom rada procesora.Rad je strukturiran u pet poglavlja. Prvo poglavlje daje uvod u koncept web aplikacija, njihove značajke i ciljeve. U drugom poglavlju bit će prikazana tema u sklopu važnosti postojanja slične aplikacije. Treće poglavlje usredotočuje se na tehnologije korištene u razvoju aplikacije, objašnjavajući razloge za njihov odabir. Četvrto poglavlje razrađuje ključne komponente aplikacije, opisujući njihove funkcije i procese koji se odvijaju unutar nje. Na kraju, peto poglavlje donosi pregled vizualnog izgleda aplikacije, uključujući detaljno objašnjenje funkcionalnosti svakog dijela.

1.1. Zadatak završnog rada

Zadatak završnog rada je izraditi React.js sučelje koje omogućava simulaciju rada PicoBlaze procesora implementiranog na Nexys A7 FPGA maketi. Korištenjem SVG slike makete treba omogućiti interakciju s ulaznim i izlaznim uređajima te omogućiti unos i uređivanje programa pomoću Monaco Editora. Dodatno, potrebno je implementirati kontrolne funkcije za izvođenje programa, uključujući pokretanje, pauziranje i postavljanje točaka prekida

2.PREGLED PODRUČJA TEME

U današnje vrijeme, razvoj web aplikacija predstavlja ključnu komponentu digitalizacije različitih procesa, posebno u području obrazovanja i simulacija. Internet aplikacije omogućuju korisnicima interakciju s kompleksnim sustavima putem jednostavnih i intuitivnih sučelja. U kontekstu simulacije rada PicoBlaze procesora, postoji značajna potreba za razvojem aplikacija koje ne samo da omogućuju kodiranje i testiranje programa, već i vizualizaciju procesa na interaktivan način. Jedna od ključnih karakteristika ovih aplikacija je pristupačnost, koja omogućava korisnicima svih dobnih skupina da lako koriste platformu. U ovom slučaju, implementacija SVG slike makete Nexys A7 FPGA pruža korisnicima vizualni alat za razumijevanje funkcionalnosti ulaznih i izlaznih uređaja unutar simulacije. Ova vizualizacija poboljšava iskustvo učenja i olakšava korisnicima shvaćanje kompleksnih koncepata programiranja i digitalne logike. Uz to, integracija Monaco Editora omogućuje korisnicima pisanje i uređivanje koda u stvarnom vremenu, uz podršku za isticanje sintakse. Ova funkcionalnost je posebno važna jer olakšava učenje i identifikaciju grešaka u kodu, čime se potiče samostalno istraživanje i eksperimentiranje. Trenutno ne postoji sličan simulator na internetu koji kombinira SVG vizualizaciju s interaktivnim uređivanjem koda. Ovaj rad ima za cilj popuniti tu prazninu i pružiti sveobuhvatnu platformu koja ne samo da simulira rad PicoBlaze procesora, već i omogućava korisnicima aktivno sudjelovanje u procesu programiranja i testiranja. Kroz ovu internet aplikaciju, korisnici će steći praktična znanja i vještine koje su ključne za razumijevanje rada digitalnih sustava.

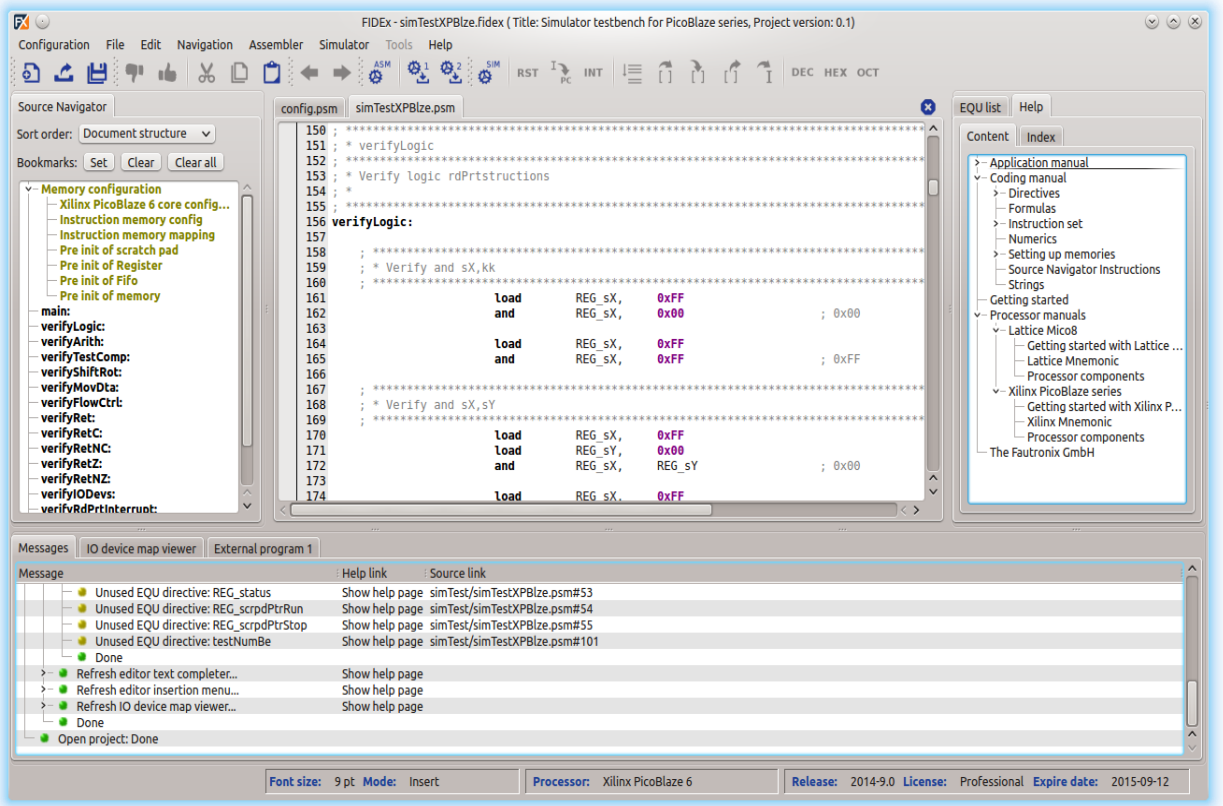
2.1. Pregled sličnih rješenja

FIDEx

FIDEx simulator za PicoBlaze ima određene nedostatke jer je zatvorenog koda, što ograničava prilagodljivost i mogućnost njegovog daljnjeg razvoja od strane zajednice. Jedan od glavnih problema današnjih simulatora za PicoBlaze je potreba za instalacijom na računalo kako bi mogli funkcionirati. Nije moguće pokrenuti ih izravno u internetskom pregledniku niti ih jednostavno preuzeti kao komprimiranu datoteku, raspakirati i koristiti bez dodatnih instalacija.

Mnogi današnji simulatori također pokušavaju simulirati PicoBlaze na razini VHDL(engl. *VHSIC Hardware Description Language*) detalja, što često povlači za sobom korištenje zatvorenog koda, budući da PicoBlaze nije kompatibilan s GHDL-om(engl. *Generating Hardware Description Language*), kompajlerom za VHDL otvorenog koda. Iako simulacija na razini VHDL-a može biti korisna u specifičnim tehničkim primjenama, za potrebe laboratorijskih vježbi iz predmeta poput Arhitekture računala, takva razina detalja nije potrebna. U takvim slučajevima, jednostavniji i pristupačniji alati bili bi daleko korisniji za studente. [8]

Uz sve navedeno, FIDEx simulator nema grafički prikaz, što dodatno ograničava njegovu upotrebljivost, osobito u edukativne svrhe, gdje bi vizualna interakcija s komponentama poput LED-ica, tipkala i 7-segmentnih prikaza mogla olakšati razumijevanje rada PicoBlaze procesora. Nedostatak ovakvog korisničkog sučelja predstavlja značajan nedostatak za one koji žele brzo i jednostavno testirati svoje programe i vidjeti rezultate u stvarnom vremenu. Izgled sučelja simulatora je prikazan na slici 2.1.



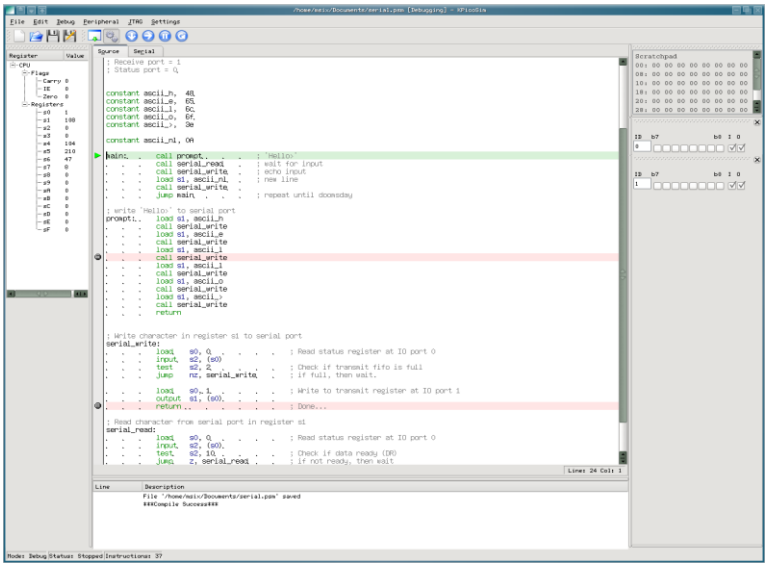
Slika 2.1. *sučelje FIDEx simulatora*

PicoBlaze IDE

Tvrtka Mediatronix nekada je proizvodila simulator PicoBlaze-a zvan PicoBlaze IDE. Bio napisan u Delphiju. Simulirao je rad PicoBlaze 1, PicoBlaze 2, PicoBlaze 3 i CoolBlaze. Velika prednost ovog simulatora je to što on označava linije asemblerskog koda u kojima su greške, dok simulator zadan kao zadatak ovog rada ne najavljuje greške. Kao i simulator PicoBlazea 6 zadatak napravljen za ovaj rad, podržava aritmetičke izraze u konstantama, uključujući i izraze sa zagradama. Danas se PBlaze IDE nigdje na internetu ne može preuzeti, što je veliki nedostatak. [9]

KPicoSim

KPicoSim je simulator otvorenog koda, to predstavlja njegovu značajnu prednost. Razvio ga je Mark Six, a osim heksadecimalnih datoteka, može izvoziti i VHDL i MEM datoteke(odnosi na datoteke povezane s memorijskim sadržajem). Nasuprot tome, PicoBlaze Simulator razvijen u sklopu ovog rada podržava izvoz programa u tekstualnoj datoteci, što omogućava jednostavno spremanje i dijeljenje koda. Datoteke se ne izvoze kao heksadecimalne budući da u nekim preglednicima, poput WebPositive, izvoz heksadecimalna datoteka nije uvijek pouzdan, a postoji mogućnost problema i u Safariju. Glavni nedostatak KPicoSima je njegova složena instalacija, jer zahtijeva kompajliranje iz izvornog koda. Po funkcionalnosti je usporediv s PBlaze IDE-om, no posljednja verzija KPicoSima objavljena je 2009. godine i ne podržava PicoBlaze 6, što znači da ne podržava naprednije instrukcije poput “jump@”. KPicoSim je napisan u C++ jeziku. Sličnost koju KPicoSim dijeli s projektnim zadatkom je to što ima grafičko sučelje, što olakšava prikaz i interakciju. Iako je posljednja verzija stara više od desetljeća, otvoreni kod omogućuje korisnicima da prilagođavaju i proširuju simulator prema vlastitim potrebama, čime ostaje fleksibilniji od zatvorenih rješenja. [10] Sučelje KPicoSim simulatora je prikazano na slici 2.2.



Slika 2.2. *sučelje KPicoSim simulatora*

Xilinxov PicoBlaze Simulator

Xilinxov PicoBlaze Simulator, kao službeni simulator, nije otvorenog koda. On je dio Xilinxovih razvojnih alata, koji su komercijalni softver. Ovaj simulator se fokusira na simulaciju asemblerskog koda za PicoBlaze procesor, ali ne pruža napredne grafičke prikaze FPGA uređaja ili njihovu arhitekturu.U Xilinxovim alatima, kao što su Vivado ili ISE, korisnici mogu koristiti FPGA uređaje i raditi s njihovim grafičkim sučeljima, no sam simulator ne uključuje detaljan grafički prikaz FPGA uređaja kao što bi to neki drugi simulatori mogli imati. [11]

Tablica 2.1. *prikaz prednosti i mana simulatora*

Simulator	**Open Source**	**Grafički prikaz na FPGA maketi**	**Dostupnost preuzimanja**
Xilinx	-	-	+
PicoBlaze	+	+	+
PicoBlaze IDE	-	+	-
FIDEx	-	-	+
Projektni zadatak	+	+	+

3. TEHNOLOGIJE WEB APLIKACIJ

3.1. HTML

HTML (engl. *HyperText Markup Language*) predstavlja jedan od temeljnih gradbenih blokova interneta. On definira značenje i strukturu, čime poboljšava i olakšava organizaciju te prikaz web sadržaja na WWW-u (eng. *World Wide Web*). HTML koristi označavanje (engl. *markup*) za tekst, slike i drugi sadržaj, koji se prikazuju u web pregledniku putem oznaka (engl. *tags*). Ovim pristupom omogućava se prikaz raznih elemenata, uključujući odlomke, slike, poveznice i ostali multimedijski sadržaj. Elementi se u dokumentu razlikuju od običnog teksta pomoću znakova „<“ i „>“. Imena elemenata unutar oznaka nisu osjetljiva na velika i mala slova; ipak, preporučuje se da se oznake pišu malim slovima. Jedna od najvažnijih karakteristika HTML-a je njegova semantička priroda, što znači da oznake ne služe isključivo za prezentaciju sadržaja, već također dodaju značenje različitim dijelovima dokumenta. Oznake pomažu preglednicima u tumačenju svake oznake; na primjer, oznaka za naslov označava da je određeni tekst ključni naslov stranice. Ova semantička struktura omogućuje pretraživačima i alatima za pristupačnost bolje razumijevanje sadržaja stranice, čime se poboljšava korisničko iskustvo i optimiziraju rezultati pretraživanja. [12] Također, HTML je neovisan o platformi, što znači da se može koristiti na bilo kojem operativnom sustavu i web pregledniku, zbog čega je ključni alat za stvaranje univerzalno dostupnog web sadržaja. Web preglednici interpretiraju HTML kod i prikazuju sadržaj korisnicima u skladu s pravilima koja su definirana standardima.

3.2. Javascript

JavaScript je interpretirani programski jezik koji nudi podršku za funkcije prvoklasnog tipa. Najpoznatiji je kao skriptni jezik za web stranice, no također se koristi u raznim okruženjima koja nisu preglednici, poput Node.js-a i Adobe Acrobata. Ovaj jezik je prototipski dinamički jezik koji podržava objektno-orijentirane, imperativne i deklarativne stilove programiranja. JavaScript, u kombinaciji s povezanim značajkama preglednika, predstavlja tehnologiju za unapređenje internetskog sadržaja. Kada se koristi na klijentskom računalu, jezik omogućuje transformaciju statičnih stranica u zanimljivo, interaktivno i dinamično iskustvo za korisnike. [13]

JavaScript omogućuje programerima stvaranje bogatih korisničkih sučelja, obradu podataka u stvarnom vremenu i interakciju s korisnicima na način koji nije moguć isključivo pomoću statičnih jezika poput HTML-a i CSS-a(engl. *Cascading Style Sheets*). Jedna od ključnih karakteristika

JavaScripta je mogućnost izravnog izvršavanja koda u pregledniku, bez potrebe za prethodnom interpretacijom. Ovo omogućava brže izvršavanje skripti i stvaranje interaktivnog sadržaja u stvarnom vremenu. Također, JavaScript je dinamički tipizirani jezik, što znači da se vrste podataka mogu mijenjati tijekom izvođenja programa, što olakšava rad s raznolikim tipovima podataka. Osim toga, JavaScript omogućuje rad s DOM-om (engl. *Document Object Model*), što programerima omogućava manipulaciju HTML i CSS elementima na stranici, promjenu stilova, dodavanje ili uklanjanje elemenata te stvaranje potpuno dinamičkih korisničkih sučelja. U kombinaciji s tehnologijama kao što su AJAX (engl. Asynchronous JavaScript and XML), JavaScript omogućava slanje i primanje podataka s poslužitelja u stvarnom vremenu bez potrebe za ponovnim učitavanjem stranice, čime se značajno poboljšava korisničko iskustvo.

3.3. React

React je okruženje za izradu korisničkih sučelja i razvoj aplikacija na jednoj stranici. Razvijen od strane Facebooka, React omogućava izgradnju učinkovitih i sofisticiranih web aplikacija koristeći komponentni pristup. Kao platforma, uključuje bogat ekosustav alata i biblioteka koje podržavaju širok spektar funkcionalnosti, uključujući upravljanje stanjem, usmjeravanje, asinkronu komunikaciju s poslužiteljem i još mnogo toga. React nudi fleksibilne i ponovo upotrebljive komponente koje olakšavaju razvoj, održavanje i ažuriranje koda web aplikacija, čime se poboljšava produktivnost programera i korisničko iskustvo. U kombinaciji s drugim tehnologijama, poput Redux-a za upravljanje stanjem ili React Router-a za usmjeravanje, React postaje moćna platforma za stvaranje modernih web rješenja. Glavna značajka ovog okruženja je komponentni pristup organizaciji strukture web aplikacije. Komponente su neovisni dijelovi aplikacije koje se mogu ponovno upotrebljavati, što predstavlja veliku prednost prilikom razvoja složenih web aplikacija s različitim funkcionalnostima i stranicama. Svaka komponenta u Reactu može sadržavati JavaScript (ili TypeScript), JSX(engl. *JavaScript XML*) predložak i CSS stilove. JSX predložak opisuje samu strukturu komponente, dok JavaScript klasa definira logiku i ponašanje komponente. CSS stilovi dodaju vizualni aspekt aplikaciji, poput boje fontova ili razmaka između HTML elemenata. Jedna od najčešćih primjena Reacta je razvoj jednostranih aplikacija (engl. *Single Page Applications* - SPA), gdje cijela aplikacija funkcionira unutar jedne stranice, a promjene u sadržaju se odvijaju dinamički bez ponovnog učitavanja stranice. Ovaj pristup omogućava brže i fluidnije korisničko iskustvo. React je također vrlo popularan u poslovnim okruženjima zbog svoje sposobnosti skalabilnosti kod velikih projekata te pruža robusne alate za upravljanje stanjem, testiranje i modularizaciju.

3.4. CSS

CSS (engl. *Cascading Style Sheets*) je jezik za stilizaciju koji se koristi za kontrolu prezentacije i izgleda web stranica. U ovom projektu, CSS se koristi za stiliziranje komponenti, osiguravajući vizualno privlačno i korisnički prijateljsko sučelje. Omogućava prilagodbu fontova, boja, razmaka i cjelokupne estetike, što doprinosi boljem korisničkom iskustvu. [14]

3.5. Monaco editor

Monaco Editor je komponenta za uređivanje koda koja pruža značajke poput isticanja sintakse, automatskog dovršavanja i navigacije kroz kod. To je isti uređivač koji se koristi u Visual Studio Code i osmišljen je za izradu web aplikacija koje zahtijevaju mogućnosti uređivanja koda. U ovom projektu, služi kao primarno sučelje za korisnike za pisanje i uređivanje PicoBlaze asemblerskog koda.

3.6. SVG

SVG (engl. *Scalable Vector Graphics*) je format vektorske slike temeljen na XML-u koji se koristi za stvaranje dvodimenzionalnih grafika. Neovisnost o razlučivosti čini ga idealnim za web aplikacije. U ovom projektu, SVG se koristi za prikazivanje Nexys A7 FPGA makete, omogućavajući interaktivne simulacije značajki makete kao što su tipke, LED-ice i sklopke. SVG grafike su lako skalabilne i manipulativne putem CSS-a i JavaScripta, što poboljšava interaktivnost simulacije.

3.7. XMLHttpRequest

XMLHttpRequest (XHR): JavaScript API za slanje HTTP zahtjeva poslužitelju. Omogućuje web aplikacijama interakciju s udaljenim resursima bez ponovnog učitavanja stranice, što omogućava dinamička ažuriranja sadržaja. [15]

4. POSTAVLJANJE RAZVOJNOG OKRUŽENJA I KOMPONENTE WEB APLIKACIJE

4.1. Postavljanje razvojnog okruženja web aplikacije

Razvojna okolina za React uključuje instalaciju potrebnih alata i konfiguraciju projekta. Koraci potrebni za postavljanje React razvojne okoline su idući:

Korak 1: Instalacija Node.js

React zahtijeva Node.js i njegov paketni upravitelj *npm*. Node.js se preuzima preko interneta, te nakon instalacije potrebno je provjeriti je li instalacija uspješno završila pokretanjem sljedećih naredbi u naredbenom retku (engl. *command prompt*) prikazanim u programskom kodu 4.1:

```
node –v
npm –v
```

Programski kod 4.1. *Provjera verzije Node.js*

Korak 2: Instalacija create-react-app

Create-react-app koristi se za generiranje, izgradnju i upravljanje React projektima. Instalira se koristeći *npm* paket putem naredbe prikazane u programskom kodu 4.2:

```
npm install -g create-react-app
create-react-app ime-aplikacije
```

Programski kod 4.2*. Instalacija create-react-app globalno*

Korak 3: Pokretanje aplikacije

Nakon instalacije potrebnih alata, pokreće se novi React projekt naredbom prikazanom u programskom kodu 4.3:

```
cd ime-aplikacije
npm start
```

Programski kod 4.3*. Pokretanje aplikacije na "http://localhost:3000"*

Korak 4: Naknadno pokretanje React razvojnih alata

Potrebno je preuzeti razvojne alate i instalirati ih prikazano u programskom kodu 4.4. i dodati s programskog koda 4.5. u zaglavlje HTML koda.

```
npm install -g react-devtools
```

Programski kod 4.4. *Instalacija razvojnih alata*

```
<script src="http://localhost:8097"></script>
```

Programski kod 4.5. *pokretanje slušanja na "http://localhost:8097*

Aplikacija se pokreće na lokalnom poslužitelju na adresi *http://localhost:8097/* te je React aplikacija spremna za daljnje razvijanje i izgradnju.

Korak 5: Instalacija Monaco Editora i Monaco React paketa prikazanog programskim kodom 4.6.

```
npm install monaco-editor --save
npm install react-monaco-editor –save
npm install monaco-editor-webpack-plugin --save-dev
```

Programski kod 4.6. *postavljanje okruženja za Monaco Editor*

Izborni korak: React podržava rad s SVG slikom, za napredne kontrole SVG slike nekada je potreban kod 4.7.

```
npm install @svgr/webpack --save-dev
```

Programski kod 4.7. *napredni @svgr/webpack paket*

4.2. Izbornik za upravljanje

Ovaj kod definira React komponentu nazvanu *ControlMenu*, koja omogućuje korisniku interakciju s tekstualnim editorom i simulaciju programa za kontrolu LED dioda na Nexys A7 ploči. Izbornik je služi za upravljanje simulatorom, posjeduje funkcije kao što su: spremanje, učitavanje, brisanje i pokretanje koda. Pojedinačni dijelovi koda slijede u nastavku:

Stanja i reference:

- *useState:* se koristi za praćenje stanja programa *(program)* koji se učitava, sprema ili izvršava.
- *useRef* se koristi za stvaranje reference na HTML element *input* koji služi za učitavanje datoteka, a sakriven je s korisničkog sučelja.

Funkcije:

- *newProgram*: Postavlja program na praznu vrijednost i briše sadržaj iz povezanog editora, ako je prisutan. To omogućuje stvaranje novog programa od nule.
- *loadProgram*: Funkcija za učitavanje programa iz lokalne datoteke. Koristi *FileReader* da pročita sadržaj datoteke i postavi ga kao trenutno aktivni program. Ako je editor prisutan, on također ažurira sadržaj editora s tim programom.
- *triggerFileInput*: Ova funkcija simulira klik na skriveni HTML input element kako bi korisnik mogao odabrati datoteku za učitavanje.

- *saveProgram*: Omogućuje spremanje programa u datoteku. Program se sprema u .kc ili .txt datoteku i automatski se preuzima kada korisnik klikne na gumb "Save Program". To se postiže stvaranjem HTML a elementa koji generira i preuzima datoteku pomoću *URL.createObjectURL*.
- *runProgram*: Ova funkcija pokreće simulaciju programa, poziva *simulateProgramExecution* funkciju za obradu programa i ažurira SVG s rezultatima simulacije pomoću updateSVG callback funkcije.
- *simulateProgramExecution*: Simulira izvršavanje programa. Analizira svaku liniju programa i traži naredbe tipa "LEDx on" ili "LEDx off", gdje x označava broj LED diode, a “on/off” označava stanje. Rezultat je niz objekata koji opisuju status LED dioda. Ove rezultate komponenta dalje koristi za ažuriranje SVG prikaza. Sve navedeno je prikazano u programskom kodu 4.8.

Renderiranje komponenti:

Komponenta vraća nekoliko gumba:

- "New Program": za stvaranje novog programa.
- "Load Program": otvara prozor za učitavanje datoteke.
- Skriveni *input* element služi za učitavanje datoteke u .kc ili .txt formatu.
- "Save Program": omogućuje spremanje trenutnog programa u datoteku.
- "Run": pokreće simulaciju trenutnog programa i ažurira LED-ove u SVG.

```
const ControlMenu = ({ editor, updateSVG }) => {
  const [program, setProgram] = useState('');
  const fileInputRef = useRef(null);
  const newProgram = () => {
    setProgram('');
    if (editor) {
      editor.setValue('');
    }
  };
  const loadProgram = (event) => {
    const file = event.target.files[0];
    const reader = new FileReader();
    reader.onload = (e) => {
      const programContent = e.target.result;
      setProgram(programContent);
      if (editor) {
        editor.setValue(programContent);
      }
    };

    if (file) {
      reader.readAsText(file);
```

```
    }
  };
const triggerFileInput = () => {
    if (fileInputRef.current) {
      fileInputRef.current.click();
    }
  };
  const saveProgram = () => {
    const element = document.createElement('a');
    const file = new Blob([program], { type: 'text/plain' });
    element.href = URL.createObjectURL(file);
    element.download = 'program.kc';
    document.body.appendChild(element);
    element.click();
    document.body.removeChild(element);
  };
  const runProgram = () => {
    if (program) {
      const results = simulateProgramExecution(program);
      updateSVG(results); // Send the results to the SVG
    }
  };

const simulateProgramExecution = (program) => {
    const lines = program.split('\n');
    const results = lines.map((line) => {
      const ledCommandMatch = line.match(/LED(\d+)\s+(on|off)/i); // Match "LEDx on"
or "LEDx off"
      if (ledCommandMatch) {
        const ledNumber = parseInt(ledCommandMatch[1], 10); // Extract the LED number
        const isOn = ledCommandMatch[2].toLowerCase() === 'on'; // Determine the
state
        return { type: 'led', index: ledNumber - 1, state: isOn }; // Adjust for 0-
indexed array
      }
      // Add more command handling as needed (e.g., RGB control)
      return null;
    }).filter(result => result !== null); // Filter out null entries
    return results;
  };
```

Programski kod 4.8.

4.3. NexysA7

Ovaj kod definira React komponentu nazvanu NexysA7, koja simulira interakciju s Nexys A7 FPGA pločom, uključujući diode, RGB LED i gumbe. Komponenta koristi SVG za grafički prikaz ploče i omogućuje korisniku interakciju s elementima na ploči. Korištenje *useState*, *useEffect*, *useImperativeHandle*, i *forwardRef* u ovoj komponenti omogućuje praćenje stanja i omogućuje komunikaciju s drugim komponentama kroz reference.

Stanje i *useState*

- *ledStatus*: Praćenje statusa 16 LED dioda. Početno stanje je niz od 16 elemenata postavljenih na *false* (sve LED diode su ugašene).
- *rgbStatus*: Praćenje statusa RGB LED dioda, s tri boje (crvena, zelena, plava), svaka postavljena na *false* (isključene).
- *buttonStatus*: Praćenje statusa 5 gumba na ploči, svaki od njih može biti pritisnut ili ne (*false* znači da nije pritisnut).

useEffect:

- Ovaj *hook* omogućuje ažuriranje stanja LED i RGB dioda na temelju podataka dobivenih preko *props.data* (podataka koji dolaze od roditeljske komponente). Koristi se *requestAnimationFrame* kako bi se ažuriranja izvršila na učinkovit način i smanjila opterećenja na pregledniku.

useImperativeHandle:

- Koristi se s *forwardRef* kako bi se omogućilo roditeljskoj komponenti da poziva *updateDisplay* funkciju unutar NexysA7 komponente. Ova funkcija ažurira prikaz LED i RGB dioda na temelju podataka proslijeđenih kao argument.

Interakcija s LED i RGB diodama te gumbima:

- *toggleLed*: Funkcija koja se poziva kada korisnik klikne na diodu. Mijenja stanje određene diode.
- *toggleRgb*: Funkcija koja se poziva kada korisnik klikne na jednu od RGB dioda. Prebacuje stanje određene boje (crvena, zelena, plava).
- *toggleButton*: Funkcija koja se poziva kada korisnik klikne na jedan od gumba na ploči. Mijenja stanje gumba (pritisnut/nepritisnut).

Grafički prikaz (SVG):

- Komponenta koristi SVG element za prikaz Nexys A7 ploče i interaktivnih elemenata (LED diode, RGB diode, gumbi).
- LED diode: Prikazuju se kao krugovi na određenim koordinatama, a njihova boja se mijenja ovisno o stanju (crvena za uključene LED diode, crna za isključene).
- RGB diode: Prikazane su na različitim pozicijama, a njihova boja ovisi o stanju crvene, zelene i plave boje.

- Gumbi: Prikazani su kao kvadrati i smješteni su u formaciju plus znaka (sjever, jug, istok, zapad, centar).

Prikaz statusa LED dioda, RGB dioda i gumba:

- Nakon SVG dijela, ispod se prikazuje status svake LED diode ("On" ili "Off"), kao i status RGB dioda (svaka boja može biti uključena ili isključena).
- Također se prikazuje status svakog gumba (pritisnut ili ne).

Sve prethodno pometnuto se može naći u priloženom programskom kodu 4.9.

```
const NexysA7 = forwardRef((props, ref) => {
  const [ledStatus, setLedStatus] = useState(Array(16).fill(false));
  const [rgbStatus, setRgbStatus] = useState({ r: false, g: false, b: false });
  const [buttonStatus, setButtonStatus] = useState(Array(5).fill(false));

useEffect(() => {
    if (props.data) {
      // Use requestAnimationFrame for efficient updates
      requestAnimationFrame(() => {
        console.log('Props data received:', props.data);
        const newLedStatus = Array(16).fill(false);
        let newRgbStatus = { r: false, g: false, b: false };

        props.data.forEach((entry) => {
          if (entry.type === 'led') {
            newLedStatus[entry.index] = entry.state;
          } else if (entry.type === 'rgb') {
            newRgbStatus = { ...newRgbStatus, ...entry.state };
          }
        });

        setLedStatus(newLedStatus);
        setRgbStatus(newRgbStatus);
      });
    }
  }, [props.data]);

  useImperativeHandle(ref, () => ({
    updateDisplay(data) {
      console.log("updateDisplay called with data:", data);
      if (data) {
        const newLedStatus = Array(16).fill(false);
        let newRgbStatus = { r: false, g: false, b: false };

        data.forEach((entry) => {
          if (entry.type === 'led') {
            newLedStatus[entry.index] = entry.state;
          } else if (entry.type === 'rgb') {
            newRgbStatus = { ...newRgbStatus, ...entry.state };
          }
```

```
        });

        setLedStatus(newLedStatus);
        setRgbStatus(newRgbStatus);
      }
    }
  }));

const toggleLed = (index) => {
    const newStatus = [...ledStatus];
    newStatus[index] = !newStatus[index];
    setLedStatus(newStatus);
  };

 // slična logika za toggleRGB i toggleLED
return (
    <div>
      <h2>Nexys A7 Board</h2>
      <div>
        <svg width="800" height="600">
          <image href={NexysA7SVG} width="800" height="600" />

          {/* Render LED circles based on positions */}
          {ledPositions.map((pos, index) => (
            <circle
              key={index}
              cx={pos.cx}
              cy={pos.cy}
              r={15}
              style={getLedStyle(index)}
              onClick={() => toggleLed(index)}
            />
          ))}
    // ostatak koda je samo renderiranje slike
```

Programski kod 4.9.

4.4. Editor

Ovaj dio definira React komponentu pod nazivom *Editor*, koja koristi Monaco Editor za prikaz i uređivanje koda napisanog u PicoBlaze jeziku. Komponenta registrira jezik, definira prilagođenu temu, učitava uzorak koda, i osigurava da se editor ispravno inicijalizira i čisti nakon korištenja.

- Ova funkcija prikazana programskim kodom 4.10. omogućuje asinkrono učitavanje podataka s weba putem XMLHttpRequest. Vraća *Promise* koji se ispunjava kada je zahtjev uspješan ili odbija kada dođe do pogreške.

```
function xhr(url) {
  const req = new XMLHttpRequest();
  return new Promise((resolve, reject) => {
    req.onreadystatechange = function () {
      if (req.readyState === 4) {
        if ((req.status >= 200 && req.status < 300) || req.status === 1223) {
          resolve(req);
```

```
        } else {
          console.error(`Error loading ${url}: ${req.statusText}`);
          reject(req);
        }
      }
    };
    req.open('GET', url, true);
    req.responseType = '';
    req.send(null);
  }).catch(() => {
    req.abort();
  });
}
```

Programski kod 4.10. XMLHttpRequest

- Programski kod 5.11. omogućava izvođenje rada u pozadini što omogućava funkcionalnosti kao označavanje sintakse.

```
window.MonacoEnvironment = {
  getWorkerUrl: function (workerId, label) {
    return `data:text/javascript;charset=utf-8,${encodeURIComponent(`
      self.MonacoEnvironment = {
        baseUrl: '${window.location.origin}'
      };
      importScripts('${window.location.origin}/monaco-
editor/min/vs/base/worker/workerMain.js');`
    )}`;
  },
};
```

Programski kod 4.11. Konfiguracija radne okoline

Komponenta editor je prikazana na programskom kodu 5.12.:

- *useEffect* se koristi za inicijalizaciju editora nakon što se komponenta montira.
- Registrira se PicoBlaze jezik i definiraju se pravila za njegovu sintaksu.
- Prilagođena tema se definira s različitim pravilima za boje i stilove tokena (npr. ključne riječi, brojevi, komentari).
- *editorInstance* se stvara i prosljeđuje funkciji *onMount* ako je definirana.

```
const Editor = ({ onMount }) => {
  useEffect(() => {
    const container = document.getElementById('container');

    // Registracija jezika PicoBlaze
    monaco.languages.register({ id: 'picoblaze' });
    monaco.languages.setMonarchTokensProvider('picoblaze', picoBlazeLanguage);

    // Definiranje i primjena prilagođene teme
    monaco.editor.defineTheme('myCoolTheme', {
      // Definicija pravila za boje i stilove tokena
```

```
    });

    // Inicijalizacija editora s prilagođenom temom
    const editorInstance = monaco.editor.create(container, {
      theme: 'myCoolTheme',
      value: getCode(),
      language: 'picoblaze',
      automaticLayout: true,
    });
   // Proslјeđivanje instance editora roditeljskoj komponenti
    if (onMount && typeof onMount === 'function') {
      onMount(editorInstance);
    }

    // Čišćenje prilikom demontaže
    return () => {
      if (editorInstance) {
        editorInstance.dispose();
      }
    };
  }, [onMount]);
```

Programski kod 4.12. Editor

- Funkcija koja prikazuje uzorak koda u editoru koji je zadan da se prikazuje u editoru je prikazana na programskom kodu 4.13.

```
function getCode() {
  return [
    '10xy0 ADD s0 s1',
    '02xy0 AND s0 s1',
    '; This is a comment',
    // Dodaj više uzoraka koda PicoBlaze ovdje...
  ].join('\n');
}
```

Programski kod 4.13.

- Za renderiranje editora se koristi *div* element koji služi kao kontejner za editor, programski kod 4.14.

```
return <div id="container" className="editor-container" style={{ height: '600px',
width: '800px' }} />;
```

Programski kod 4.14.

4.5. Isticanje sintakse i automatsko dopunjavanje

Ovo je JavaScript objekt koji definira jezik za PicoBlaze unutar Monaco editora, poznat kao "tokenizer". Tokenizer razdvaja kod na različite elemente poput ključnih riječi, brojeva, registara i komentara, te im dodjeljuje određene stilove.

Tokenizer: Ovo je ključna komponenta koja razdvaja kod na "tokene" na temelju određenih pravila.

U programskom kodu 4.15. se primjećuje pravilo za svaki token, a u programskom kodu 4.16. se za svaki token dodjeljuje tema.

```
const picoBlazeLanguage = {
  tokenizer: {
    root: [
      // Instructions
      [

/\b(?:ADD|ADDCY|AND|CALL|CALL@|COMPARE|COMPARECY|DISABLE|ENABLE|FETCH|HWBUILD|INPUT|J
UMP|JUMP@|LOAD|LOAD&RETURN|OR|OUTPUT|OUTPUTK|REGBANK|RETURN|RETURNI|RL|RR|SL0|SL1|SLA
|SLX|SR0|SR1|SRA|SRX|STAR|STORE|SUB|SUBCY|TEST|TESTCY|XOR)\b/,
        'keyword'
      ],
      // Preprocessor Directives
      [
        /\b(?:ADDRESS|NAMEREG|CONSTANT)\b/,
        'preprocessor'
      ],
      // Registers
      [
        /(?<=[^\s,()])[sS][0-9A-F](?=[, )]|$)/,
        'register'
      ], // i tako dalje za ostatak sintakse
```

Programski kod 4.15.

```
import * as monaco from 'monaco-editor/esm/vs/editor/editor.api';

monaco.editor.defineTheme('myCoolTheme', {
    base: 'vs', // Can be 'vs', 'vs-dark', or 'hc-black'
    inherit: true,
    rules: [
      { token: 'keyword', foreground: 'FF0000', fontStyle: 'bold' }, // Red color for
keywords
      { token: 'address', foreground: '007F7F' }, // Teal color for addresses
      { token: 'number', foreground: '0000FF' }, // Blue color for numbers
      { token: 'number_system', foreground: '800080' }, // Purple color for number
systems
      { token: 'register', foreground: 'FF8C00' }, // Dark orange for registers
      { token: 'label', foreground: 'A52A2A' }, // Brown color for labels
      { token: 'comment', foreground: '008000', fontStyle: 'italic' }, // Green for
comments
```

```
      { token: 'string', foreground: 'FF4500' }, // OrangeRed for strings
    ],
    colors: {
      'editor.background': '#FFFFFF', // White background
      'editor.foreground': '#000000'  // Black text
    }
  });
```

Programski kod 4.16.

Logika za rješavanje automatskog dopunjavanja pojmova je riješena na način prikazan u programskom kodu 4.17. Uzima prvo slovo kao okidač, a moguće je dodati i nekoliko sljedećih i na taj način pokreće automatsko dopunjavanje.

```
monaco.languages.registerCompletionItemProvider('picoblaze', {
  triggerCharacters: ['R'],  // Autocomplete will trigger on typing 'R'

  provideCompletionItems: (model, position) => {
    const word = model.getWordUntilPosition(position);
    const range = {
      startLineNumber: position.lineNumber,
      endLineNumber: position.lineNumber,
      startColumn: word.startColumn,
      endColumn: word.endColumn,
    };

    // lista instrukcija
    const suggestions = [
      {
        label: 'RETURN',
        kind: monaco.languages.CompletionItemKind.Keyword,
        insertText: 'RETURN',
        documentation: 'RETURN',
        range: range,
      },
       // dodavanje ostatka
    ];

    return { suggestions: suggestions };
  },
});
```

Programski kod 4.17

5. IZGLED INTERNET APLIKACIJE

U ovom poglavlju su predstavljene slike funkcionalnosti aplikacije.

5.1. Izgled kontrolne ploče

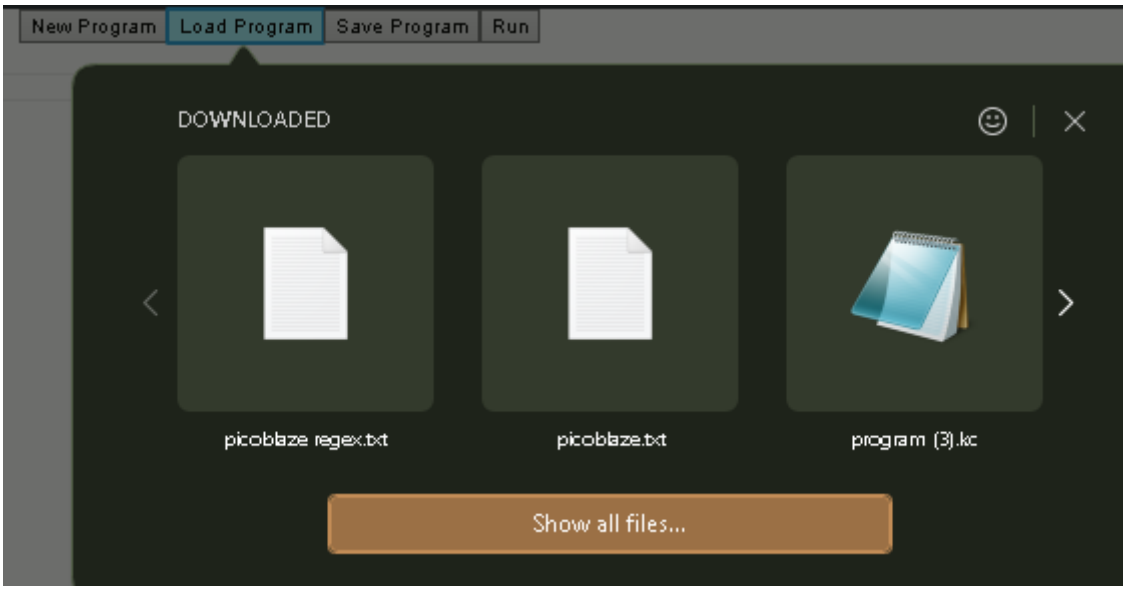
U ovom poglavlju su predstavljene slike funkcionalnosti aplikacije.

5.1. Izgled kontrolne ploče

Izgled kontrolne ploče za upravljanje simulatorom prikazan je na slici 5.1.

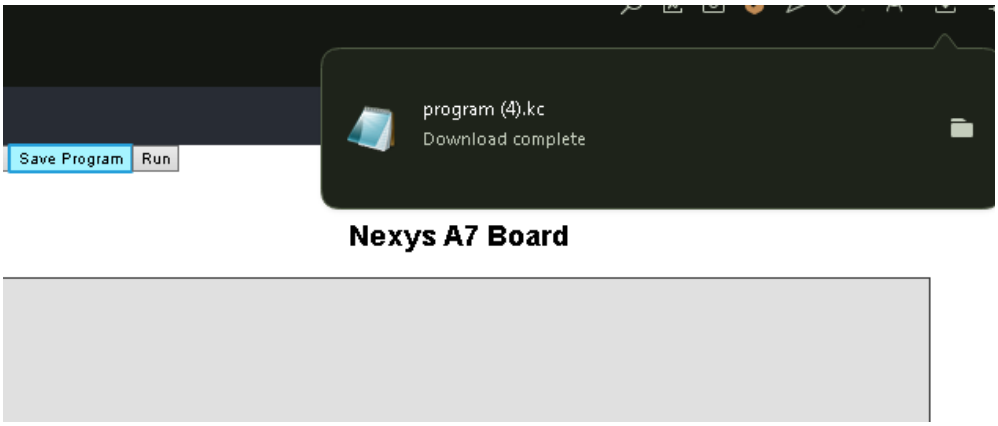
Slika 5.1. *Kontrolna ploča*

Na slici 5.2. prikazane su opcije za učitavanje programa.



Slika 5.2. *učitavanje programa*

Slika 5.3 prikaz preuzimanja programa.



Slika 5.3. *preuzimanje programa*

5.2. Editor

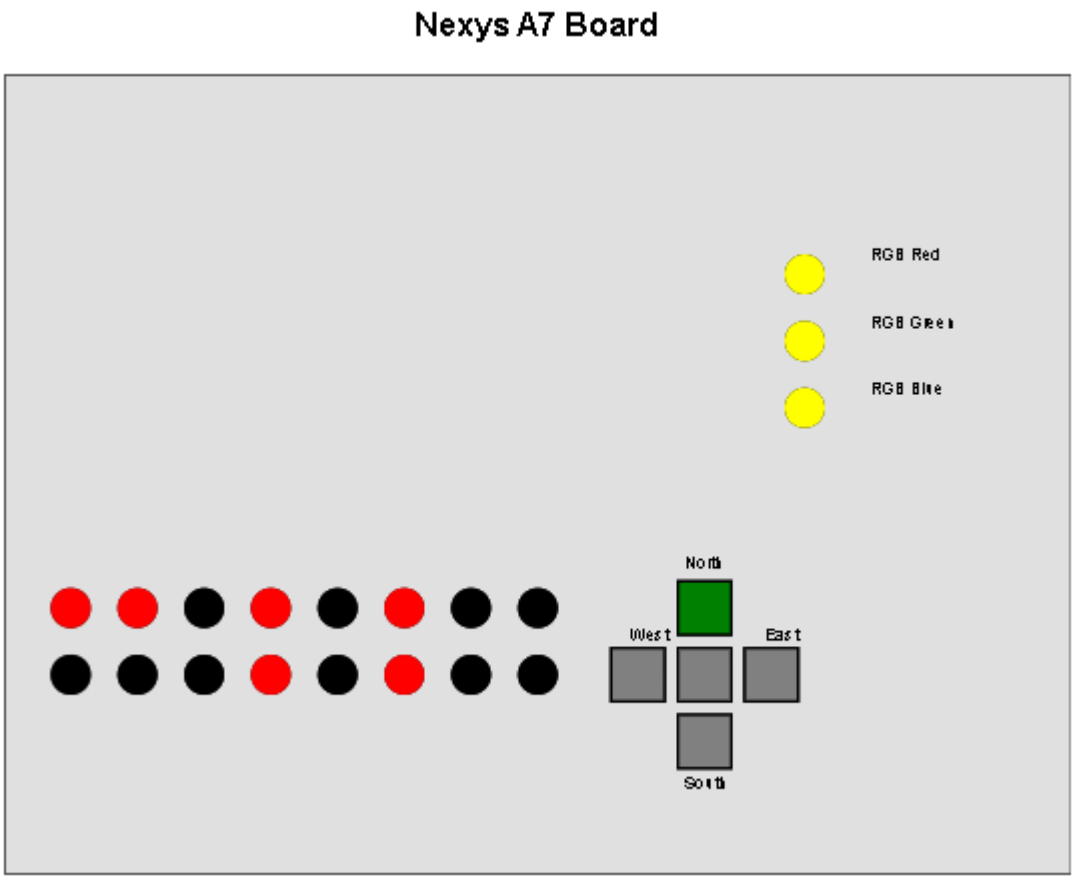
Na slici 5.4. se vidi isticanje sintakse i atomatskog dovršavanja na djelu. Također na slici 5.4. se vidi umetnuta točka prekida.

```
1   ; Simple program to test LOAD, ADD, OUTPUT, JUMP, and CALL instructions
2
3   LOAD s0, 1          ; Load the value 1 into register s0
4   LOAD s1, 2          ; Load the value 2 into register s1
5   ADD s0, s1          ; Add the value in s1 to s0 (s0 = 3 now)
6   OUTPUT 0, 0         ; Turn off all LEDs on port 0
7   OUTPUT 0, s0        ; Output the value in s0 to port 0 (LED 1 and LED 2 should
8   CALL wait_loop      ; Call the wait loop subroutine
9
10  JUMP $              ; Infinite loop to prevent program from running out
11
12  wait_loop:
13      LOAD s2, 255    ; Load 255 into register s2 (used for delay)
14  wait_loop_1:
15      SUB s2, 1       ; Decrement s2
16      JUMP NZ, wait_loop_1 ; Repeat until s2 is zero
17      RE
18      RETURN
```

Slika 5.4. *editor i označavanje sintakse*

5.3. Prikaz SVG slike

Na slici 5.5. su prikazana stanja gumba i dioda s prethodne slike. Na prikazu se vidi da su *Red i Green* upaljene tako da RGB sjaji žutom bojom. Slika 6.6 prikazuje tekstom opisane statuse elemenata.



Slika 5.5. *SVG s upaljenim LED i RGB sija žutom bojom dok je sjeverni gumb pritisnut*

Na slici 5.6. se nalazi prikaz stanja komponenti na simulatoru.

Slika 5.6. *prikaz stanja komponenti*

6. ZAKLJUČAK

U današnjem svijetu, simulacija hardverskih sustava postaje sve važnija, a interaktivnost i točnost simulacija igraju ključnu ulogu u obrazovanju i istraživanju. Ovaj završni rad fokusira se na izradu React.js sučelja za simulaciju PicoBlaze procesora implementiranog na Nexys A7 FPGA ploči. Projekt koristi moderne web tehnologije, uključujući React.js, Monaco Editor, SVG za prikaz ploče, te niz prilagođenih funkcionalnosti za upravljanje prekidačima, gumbima i LED-ovima. Korištenjem React.js omogućena je dinamična i responzivna web aplikacija koja omogućuje korisnicima interakciju s virtualnom FPGA pločom. SVG prikaz omogućuje preciznu vizualizaciju hardverskih elemenata, dok Monaco Editor pruža podršku za unos i simulaciju koda napisanog za PicoBlaze procesor. Ova aplikacija omogućuje izvršavanje koda, postavljanje točaka prekida, te vizualizaciju stanja komponenti na ploči, što olakšava korisnicima razumijevanje rada procesora. Zahvaljujući korištenju modernih tehnologija i prilagođenih komponenti, ova web aplikacija pruža efikasan alat za simulaciju i istraživanje rada FPGA sustava, što može biti od velike koristi u obrazovnim i istraživačkim okruženjima.

LITERATURA

[1] React Documentation [online], Meta, 2024, dostupno na: https://react.dev [stranica posjećena 20.05.2024.].

[2] D. Petreus, Digital Electronic System-on-Chip Design: Methodologies, Tools, Evolution, and Trends, Electronics, vol. 15, No. 2, str. 247, veljača 2024.

[3] K. Mueller, VHDL Reference Guide [online], Hochschule Bremerhaven, 2006, dostupno na: https://www1.hs-bremerhaven.de/kmueller/VHDL/ug129.pdf [stranica posjećena: 20.05. 2024.].

[4] Digilent, Nexys A7 Reference Manual, Digilent, Inc., 2018, dostupno na: https://digilent.com/reference/_media/programmable-logic/nexys-a7/nexys-a7_rm.pdf [stranica posjećena: 20.05. 2024.].

[5] All About Circuits, Field Programmable Gate Arrays (FPGAs): Introduction [online], EETech Media, 2024, dostupno na: www.allaboutcircuits.com/technical-articles/field-programmable-gate-arrays-fpgas-introduction [posjećeno: 20.05. 2024.].

[6] W3C, Scalable Vector Graphics (SVG) [online], World Wide Web Consortium (W3C), 2024, dostupno na: https://www.w3.org/Graphics/SVG/ [stranica posjećena 20.05.2024.]

[7] Microsoft, Monaco Editor Documentation [online], Microsoft, 2024, dostupno na: https://microsoft.github.io/monaco-editor/ [stranica posjećena 20.05.2024.]

[8] Fautronix, FIDEx [online], Fautronix, 2024, dostupno na:, https://www.fautronix.com/en/en-fidex [stranica posjećena 20.05.2024.]

[9] Fautronix, PicoBlaze IDE [online], Fautronix, 2024, dostupno na: https://www.fautronix.com/en/en-projects-assembler-ide [posjećeno: 20.05. 2024.].

[10] M. Six, KPicoSim [online], GitHub, 2024, dostupno na: https://github.com/MarkSix/KPicoSim [stranica posjećena 20.05.2024.]

[11] E. Ali, W. Pora, A deterministic branch prediction technique for a real-time embedded processor based on PicoBlaze architecture, Electronics, vol. 11, No. 21, str. 3438, listopad 2022.

[12] D. Goodman, Dynamic HTML: The Definitive Reference, 2nd ed., O'Reilly & Associates, Sebastopol, CA, 2002.

[13] GeeksforGeeks, Introduction to JavaScript [online], GeeksforGeeks, 2024, dostupno na: https://www.geeksforgeeks.org/introduction-to-javascript/

[14] W3Schools, CSS Introduction [online], W3Schools, 2024, dostupno na: https://www.w3schools.com/css/css_intro.asp [stranica posjećena 20.05.2024.]

[15] Mozilla, XMLHttpRequest [online], Mozilla, 2024, dostupno na: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest [stranica posjećena 20.05.2024.].

SAŽETAK

Ovaj rad prikazuje način izrade web aplikacije za simulaciju PicoBlaze procesora implementiranog na Nexys A7 FPGA ploči koristeći tehnologije poput React.js-a, Monaco Editora i SVG-a. Implementacija ovih tehnologija omogućuje funkcije poput interaktivne simulacije hardverskih komponenti, izvršavanje koda u koracima, postavljanje točaka prekida te vizualizaciju stanja LED-ova, tipki i prekidača na FPGA ploči.

Ključne riječi: Monaco Editor, NexysA7 FPGA, PicoBlaze simulator, React.js, SVG.

ABSTRACT

Frontend of an internet application for simulating the operation of the PicoBlaze processor

This paper presents the development of a web application for simulating the PicoBlaze processor implemented on the Nexys A7 FPGA board, using technologies such as React.js, Monaco Editor, and SVG. The implementation of these technologies enables features such as interactive simulation of hardware components, step-by-step code execution, setting breakpoints, and visualizing the state of LEDs, buttons, and switches on the FPGA board.

Keywords: Monaco Editor, NexysA7 FPGA, PicoBlaze simulator, React.js, SVG.