VISUAL SIMULATION OF A DIGITAL HARDWARE MODEL

Petar Minev
Technical University of Gabrovo, Bulgaria
pminev@tugab.bg

ВИЗУАЛНА СИМУЛАЦИЯ НА МОДЕЛ НА ЦИФРОВ ХАРДУЕР

Abstract

Visual simulation in circuit design involves creating a software application that provides a dynamic visual representation of a digital system modeled with Hardware Description Languages (HDLs). This allows users to gain a deeper understanding of the system's behavior by observing the simulation in real-time. Additionally, the simulation data facilitates easier identification and correction of malfunctions in digital circuits models.

This report explores the capabilities of the Visual Debug (VIZ) feature in the Makerchip IDE for creating a visual simulation of a stack-based calculator model. This model is specifically designed for online learning in digital hardware design courses offered by the Computer Systems and Technologies department at the Technical University of Gabrovo.

Keywords: *Visual Simulation; Digital Hardware Model; Stack-based Calculator; Makerchip IDE; Visual Debug.*

ВЪВЕДЕНИЕ

В настоящия доклад се разглежда дефинирането на визуализация за симулация на работата на модел на калкулатор със стекова архитектура. Целта е да се представи разработката на приложения за визуална симулация (Visual Debug) в онлайн платформата за проектиране на цифрови схеми Makerchip [1]. За реализация на визуалните симулации в Makerchip се използва библиотеката Fabric.js. Това е JavaScript библиотека, която предоставя обектен модел върху методите на ниско ниво, използвани в HTML5 режим canvas. Така, програмистът не се грижи за състоянието на чертожното поле и рендерирането, а работи директно с „обекти“ [2].

Симулационните визуализации подпомагат процеса на отстраняване на грешки в моделите на цифрови схеми, разработени с език за описание на хардуер. Необходимо е средата за проектиране да може да генерира стандартните за индустрията .vcd файлове за трасиране [3].

Моделът на калкулатор е разработен с помощта на езика за хардуерно описание TL-Verilog [4]. Езикът за описание TL-Verilog е съвременно средство за проектиране на цифров хардуер. Той позволява ефективно създаване на цифрови системи чрез надграждане на възможностите на езика SystemVerilog [5]. В него се използва методология на проектиране, основана на „транзакции“. Моделирането с транзакции е по-високо ниво на абстракция спрямо моделирането с наложилата се от десетилетия абстракция „трансфер между регистри“, използвана при традиционното проектиране с езиците за хардуерно описание VHDL или Verilog [5]. Това води до подобряване четливостта на кода, намаляване на грешките и съкращаване на времето за разработка на TL-Verilog модели [5]. Без да имат познания и опит в проектирането на цифрови схеми, студентите могат бързо в сравнително кратък срок да се запознаят и да започнат да използват TL-Verilog за разработка на собствени модели на цифрови системи [6]. Онлайн

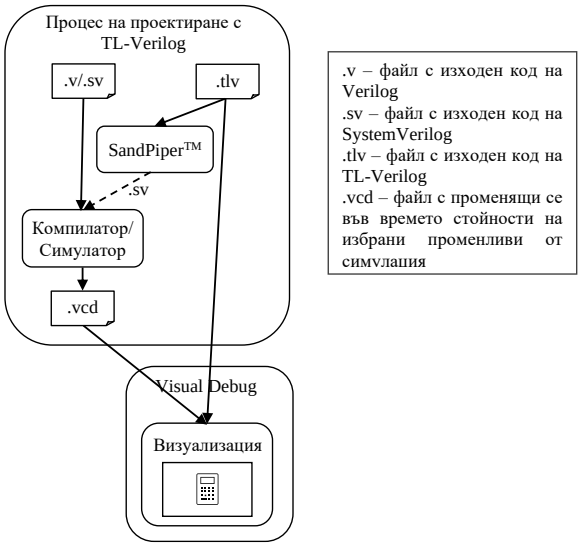
платформата Makerchip, която предлага интегрирана среда за разработка на хардуер с TL-Verilog, също улеснява обучението по автоматизираното проектиране на цифрови системи.

В TL-Verilog модела на стек-базирания калкулатор се използват масиви, които моделират памет за инструкции и стек за данни. Аритметичният израз се намира в масива „памет за инструкции" (фиг. 3). Вместо префиксно, изразът е представен в обратен полски запис. Елементите от него се извличат последователно, като операндите се записват в стека. Моделираният калкулатор поддържа четири основни целочислени операции – събиране, изваждане, умножение и деление [7]. Визуализацията на TL-Verilog модела представя текущото състояние на стека по време на изпълнение на изчисленията. Тази симулационна визуализация позволява не само да се наблюдава изпълнението на стековите операции, но дава и познания за това как е моделирана хардуерната архитектура.

Разработката има потенциал да се използва като учебен инструмент за студенти в областта на технологиите за хардуерно проектиране. Симулационната визуализация на модела, подпомагаща откриването и отстраняването на грешки, представлява важен методичен ресурс, който може да бъде интегриран в учебни курсове по автоматизирано проектиране на хардуер. Освен това, тя може да бъде основа за бъдещи проекти, които да включват по-сложни изчисления и допълнителна функционалност.

Изложение

Системата за визуализация на симулациите в Makerchip, наречена Visual Debug (или VIZ), предоставя възможности за графично представяне на изпълнението на TL-Verilog кода на един хардуерен модел. Тя е тясно интегрирана с инструментите за разработка на Redwood EDA. Работният процес при създаване на TL-Verilog модели е изобразен на фиг. 1. Този процес може да бъде изцяло капсулиран в SandStorm или Makerchip на Redwood EDA [8].



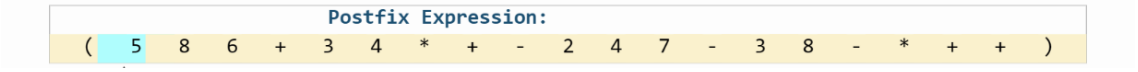
Фиг. 1. Място на Visual Debug в работния процес с TL-Verilog

За да се реализира графично приложение, което да се използва за визуална симулация, в кода на TL-Verilog модела се добавят блокове \viz_js. Структурата на един програмен /viz_js блок е представена на фигура 2.

```
/instr_mem[20:0]
   \viz_js
      all: {
               init() {
                        let imem_header = new fabric.Text("Postfix Expression:", {
                             top: 10,
                             left: 140,
                             fontSize: 22,
                             fontWeight: 800,
                             fontFamily: "monospace",
                             fill: "#1b4f72"
                           })
                           return {imem_header}
                         },
               render() {// Highlight instruction.
                           let pc = '|example$pc'.asInt(1)
                           this.highlighted_addr = pc
                           instance = this.getContext().children[pc%20]
                           instance1 = this.getContext().children[(pc-1)%20]
                           instance.initObjects.instr_asm_box.set({fill: "#b0ffff"})
                           instance1.initObjects.instr_asm_box.set({fill: "#fcf3cf"})
                         },
             },
         box: {strokeWidth: 0},
         where0: {left: -150, top: 40},
         layout: {left: 50}, //scope's instance stacked horizontally
         init() {
                  let instr_str = new fabric.Text("" , {
                              left: 40,
                              fontSize: 22,
                              fontFamily: "monospace"
                            })
                  let instr_asm_box = new fabric.Rect({
                              left: 30,
                              fill: "#fcf3cf",
                              width: 80,
                              height: 30
                            })
                 return {instr_asm_box, instr_str}
                },
         render() { // Instruction memory is constant, so just create it once.
                     let instr_str = '$instr_strs'.asString("?")
                     this.getObjects().instr_str.set({text: `${instr_str}`})
                  },
```

Фиг. 2. Примерен viz_js блок за визуализация на елементите в масив от TL-Verilog модел

Фигура 3 представя визуализацията на едномерния масив instr_mem[20:0], дефиниран в кода от фиг. 2. Клетките са в един цвят с изключение на текущата, която се адресира от сигнала $pc. В рамките на блока \viz_js, стойността на TL-Verilog сигнала $pc в текущия момент на изгледа се извлича чрез '|example$pc'. Получената стойност се интерпретира като цяло число чрез метода .asInt(1).



Фиг. 3. Резултат от визуализацията на масив, съдържащ аритметичен израз, представен в обратен полски запис

Кодът в блока \viz_js е JavaScript обект със съответните свойства. Всяка „йерархия" в TL-Verilog модела, с други думи, /hier или |pipeline, може да има най-много един \viz_js блок. В случая с кода от фиг. 2 свойството all:{…} представлява обект, който се отнася за колекцията от всички 21 инстанции на йерархично ниво /instr_mem[20:0]. Свойството all:{…} е полезно, когато трябва да се приложат действия или да се зададат настройки, които засягат всички инстанции на това ниво. Вместо да се извикват индивидуално за всяка инстанция, операциите могат да бъдат зададени глобално. В този случай, функциите init() и render() се изпълняват еднократно за всички инстанции на компонента.

В примера, с функцията init() се създава текстовият обект imem_header, който представлява общ заглавен текст за всички инстанции. Това е подходящо за елементи, които не се променят индивидуално за всяка инстанция, а са общи за всички.

Функцията render() се извиква на всяка симулационна стъпка (цикъл) и се използва за обновяване на визуализацията на всички инстанции в зависимост от текущият симулационен момент. В този случай render() задава общо поведение за маркиране (highlight) на елементите в масива /instr_mem[20:0] в зависимост от стойността на сигнала $pc. Така се постига централизирано управление на визуалните елементи за всички инстанции, вместо да се налага да се дефинират отделни render() функции за всяка инстанция.

Други свойства, освен посоченото свойство all:{…}, дефинирано в тялото на \viz_js, което само по себе си се явява JavaScript обект, са:

- where: {left: …, top: …} : позиция на обект (включително неговите инстанции) в родителския компонент.
- box: {…} : включват се свойствата: width, high, left и top, които определят границите на този компонент в неговата собствена координатна система. Ако това поле, не се използва, границите на компонента се определят от по-тесните граници на template и init обектите и box, template и init обектите на всички дъщерни компоненти.
- layout: { horizontal | vertical } : Хоризонтално или вертикално подреждане на инстанциите на компонента.
- where0: {left: …, top: …} : може да се използва заедно с all() за указване на вграждането на инстанцията с индекс нула във всички обекти на компонента, който е родител на инстанцията.

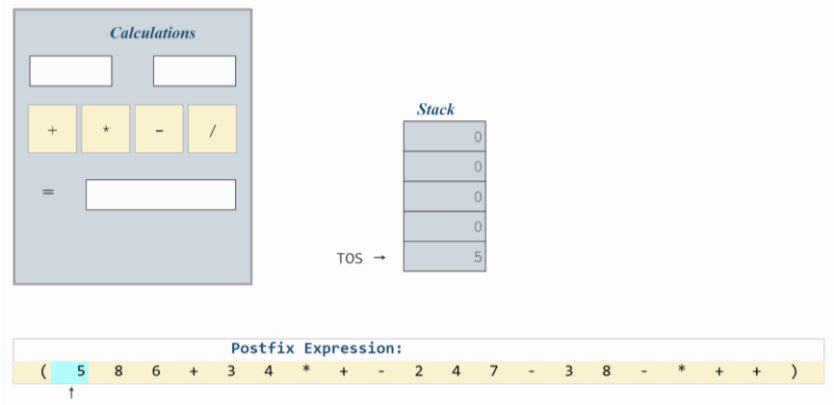
Подробно обяснение на всяка част от кода на фиг. 2, както и пълният програмен код, използван за дефиниране на симулационната визуализация на TL-Verilog модела на стек-базиран калкулатор са достъпни на адрес: https://github.com/pminev23/stackCalc.

Цялостната визуализацията на работата на стек-базирания калкулатор е представена на фиг. 4. Освен вече представения масив с аритметичния израз, подлежащ на изчисление в калкулатора, на фигурата се вижда структурата и съдържанието на стека. Стекът притежава указател, наречен TOS (Top Of the Stack), сочещ към последно записания елемент в него Също така, в интерфейса на калкулатора е включена секция с полета, в които се визуализират двата текущи операнда, операцията която се извършва над тях и полученият резултат.

Заключение

Представеният подход за визуализация на симулацията на хардуерни модели, прави по-лесно разбирането и проследяването на тяхната работа. Той може да бъде приложен в различни контексти, включително за демонстрации за образователни цели, при разработка на процесори и други хардуерни компоненти, визуализирайки изчислителния процес в учебна среда.

В бъдеще се предвижда да се разработят визуализации за симулация на модели на процесори с различни микроархитектури, които да послужат в учебната работа по дисциплината Компютърни архитектури за студенти от Технически университет – Габрово.



Фиг. 4. Общ изглед от визуализацията на симулацията за модела на стек-базирания калкулатор

Благодарности

Този доклад е подготвен и осъществен като част от проект № 2409Е „Приложение на съвременни IT технологии за представяне, опазване и съхраняване на културно-историческото наследство", финансиран от средствата по бюджета за научни изследвания на Технически университет – Габрово.

References // Литература

1. Makerchip, "Online platform for digital design," [Online]. Available at: https://www.makerchip.com (last view: 21-07-2024).
2. Fabric.js. (2024). "Introduction to Fabric.js – Part 1". Available at: http://fabricjs.com/fabric-intro-part-1 (last view: 08-08-2024)
3. "IEEE Standard for Verilog Hardware Description Language", IEEE Std 1364-2001, 2001.
4. Gosh, K., S. Hoover. (2018). "Pipelining RISC-V with Transaction-Level Verilog". [Video]. Available at: https://makerchip.com/resources (last view: 23-09-2024).
5. Hoover, S. (2017). "Timing-Abstract Circuit Design in Transaction-Level Verilog," IEEE International Conference on Computer Design (ICCD), Boston, MA, pp. 525-532, 2017. DOI: http://dx.doi.org/10.1109/ICCD.2017.91
6. Hoover, S., & Salman, A. (2018). "Top-Down Transaction-Level Design with TLVerilog," VSDOpen, 2018. Available at: https://arxiv.org/pdf/1811.01780.pdf (last view: 03-03-2024). DOI: https://doi.org/10.48550/arXiv.1811.01780
7. Kukenska, V., Minev, P., Varbov, I., & Dinev, M. (2023). "A Model for Online Learning in Digital Hardware Design", Fifth International Scientific Conference "Innovative STEM Education", Science Series, Volume 5, Veliko Tarnovo, Bulgaria, pp. 103-111, eISSN: 2683-1333. DOI: https://doi.org/10.55630/STEM.2023.0513
8. Redwood EDA. (2021). "Visual Debug User Guide (Draft)". 2021.

Received: 30-09-2024 Accepted: 03-11-2024 Published: 20-12-2024

Cite as:
Minev, P. (2024). "Visual Simulation of a Digital Hardware Model", Science Series "Innovative STEM Education", volume 06, ISSN: 2683-1333, pp. 115-119, 2024. DOI: https://doi.org/10.55630/STEM.2024.0612