A Model for Online Learning in Digital Hardware Design

Valentina Kukenska, Petar Minev, Ilian Varbov, Matyo Dinev

Technical University of Gabrovo, Bulgaria

vally@tugab.bg; pminev@tugab.bg; ivarbov@tugab.bg;
mat_s@abv.bg

Модел за онлайн обучение по проектиране на цифров хардуер

Abstract: *The report presents a model of a calculator with a stack architecture. It is developed in the hardware description language TL-Verilog. The model is designed for online learning in a development environment for digital hardware design. It is part of a set of models used for training students and doctoral students at Technical University - Gabrovo.*

Keywords: *TL-Verilog; Model; Online Learning; Stack-based Calculator.*

Въведение

Езикът за описание на хардуер TL-Verilog дава възможност да се проектират цифрови системи, като се използват концепциите, свързани с паралелизма в работата на компютърните архитектури. Така отпада необходимостта да се изучават езици за описание на хардуер, като VHDL, Verilog, System Verilog, System C и други [3], [4]. Произхода на тези езици е свързан с езиците за програмиране Ada и C. Първоначалното им предназначение е само за симулация. Освен присъщите им последователни оператори, са добавени и паралелни такива. Едновременното използване на последователни и паралелни оператори води до усложняване на моделите. Освен това са необходими и много по-комплексни инструменти за синтез, чрез които моделите да се трансформират в схеми с логически елементи и тригери [2].

Без да имат познания и опит с програмирането, студентите могат в сравнително кратък срок (от пет до десет дни) да се запознаят и да започнат да използват TL-Verilog за разработка на собствени модели на компютърни системи[1], [8]. Това прави този език подходящо средство за практическо обучение по дисциплини като Организация на компютъра, Компютърни архитектури и други.

Наличието на онлайн среда за разработка [7] с TL-Verilog също е предпоставка, улесняваща внедряването на езика в практическото обучение по дисциплини свързани с проектиране на цифрови системи. Онлайн средата за разработка е достъпна на адрес: makerchip.com. Тя е безплатна и не се изисква регистрация.

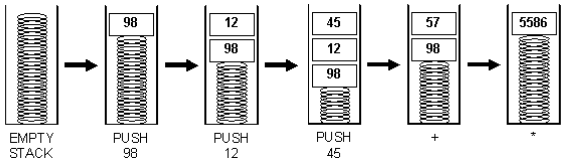
В този доклад е представен модел на калкулатор със стекова архитектура [6]. Той е разработен със средствата на TL-Verilog [5]. Моделът позволява да се изследват възможностите за моделиране на памет и по конкретно на памет с организация от тип LIFO (Last In First Out). От друга страна, използването на стек, за съхранение на входните операнди и на резултатите от изчисленията, опростява и улеснява логиката, реализираща достъпа до тях. Така, моделът става разбираем и може по-лесно да се интегрира в учебния процес.

Изложение

Принцип на действие на стековите калкулатори

Стековите калкулатори изпълняват операции за манипулиране на данни, използвайки постфиксен запис или обратната полска нотация. Постфиксният запис се отличава с факта, че операндите се записват преди операцията.

На фиг. 1 е показано използването на стек при изчислението на математическия израз „***(12 + 45) * 98***". Еквивалентът на този израз, представен в постфиксна нотация е „***98 12 45 + ****". В постфиксната нотация операторът действа спрямо последно записаните операнди и използва същия стек за запис на резултата. В този пример, числата ***98***, ***12*** и ***45*** ще бъдат поставени в стека, както е показано на фиг. 1. Операторът + ще действа върху най-горните два елемента в стека (***45*** и ***12***). Резултатът ***57*** ще бъде записан във върха на стека, като преди това участвалите в операцията елементи ще бъдат премахнати. Операторът * ще работи върху новите два най-горни елемента от стека ***57*** и ***98***, записвайки ***5586***.

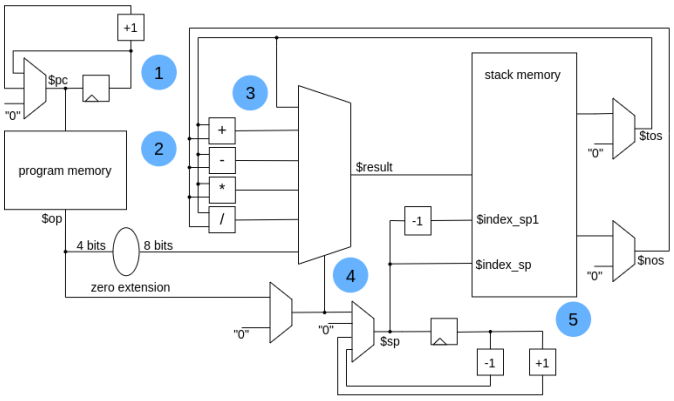


Фигура 1. Последователност на операциите при изчислението на „***98 12 45 + ****" [6]

Структура на модела

Разработеният модел на калкулатор позволява работа с цели едноцифрени числа. Върху тях могат да се извършват операции събиране, изваждане, умножение и деление. Изразите, които следва да се изчислят, се записват предварително в програмна памет.

На фиг. 2 е представена структурната схема на проектирания калкулатор. Блоковете в тази структура са: 1) програмен брояч; 2) програмна памет; 3) аритметично-логическо устройство (АЛУ); 4) логика за декодиране на прочетените елементи от програмната памет и за адресиране на стека; 5) стек.



Фигура 2. Обобщена блокова схема на модела на стек-базирания калкулатор

1) Програмен брояч

Програмният брояч посочва адреса на следващия елемент от входния израз в програмната памет. Тя съдържа операциите и данните за тях. Във всички случаи, елементите от тази памет се четат последователно, което означава, че по подразбиране стойността на програмния брояч се увеличава с единица при всеки тактов период.

2) Програмна памет

Програмната памет съдържа операндите и изпълняваните от калкулатора операции, записани в постфиксен ред. Четенето от тази памет (извличането на операнд или операция) се извършва спрямо посочения от програмния брояч адрес.

3) Аритметично-логическо устройство

След като са получени операнди и е извлечена операция от паметта, тя трябва да се изпълни. Операциите, които се изпълняват от аритметично-логическото устройство са събиране, изваждане, умножение и деление. Всички операции са целочислени.

4) Декодер на операции

Извлеченият от паметта за инструкции елемент, трябва да се интерпретира или декодира. Това се извършва като се прави проверка дали елементът е цифра от 0 до 9 или е символ от A до F. В първия случай елементът е операнд, който трябва да се запише в стека, а във втория – операция, чрез която се посочва какво действие трябва да се извърши от аритметично-логическото устройство.

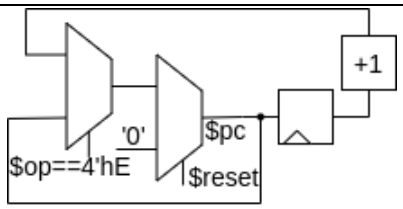
5) Стек. Адресиране на стека за четене и запис

Стекът е малко локално хранилище на стойности, с които калкулаторът работи активно. Декодирането на операцията, определя дали в стека ще се записва входен операнд или ще се четат двата елемента от неговия връх за да се изпълни аритметично-логическа операция.

Реализация на логиката на програмния брояч

Реализирано е последователно извличане на елементите от програмната памет. На фиг. 3 нарастването на програмния брояч е представено с "***+1***" (+ един елемент). Извличането на елементи трябва да започне от нулевия адрес на паметта. Това се реализира със сигнала за нулиране *$reset*. На сигнала *$pc* се присвоява предишната

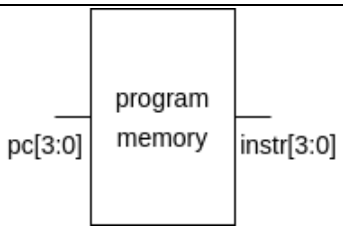
му стойност увеличена с единица. Максималната стойност, до която може да достигне брояча, е ограничена до броя на елементите във входния израз.




```
m4_define(M4_NUM_INSTRS, 11)
$pc[3:0] = $reset ? 4'h0  :
    (>>1$op == 4'hE) ? >>1$pc :
    (>>1$pc + 1) % M4_NUM_INSTRS;
```

Фигура 3. Схема и код на логиката на програмния брояч

Реализация на програмната памет




```
logic [3:0] instrs [0:M4_NUM_INSTRS-1];
assign instrs = '{
    4'hF, //   First
    4'h5, //    5
    4'h8, //    8
    4'h6, //    6
    4'hA, //    +
    4'h3, //    3
    4'h4, //    4
    4'hC, //    *
    4'hA, //    +
    4'hB, //    -
    4'hE  //    End
  };
```

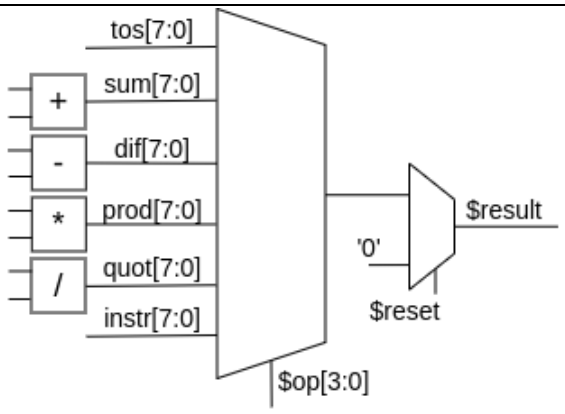
Фигура 4. Схема и код на програмната памет

За реализацията на програмната памет е използван предварително създаден Verilog масив, в който е записан входният израз, изчисляван от калкулатора. Тази

памет приема адрес като вход и произвежда 4-битовите данни за четене като изход (фиг. 4). От така реализираната памет може само да се чете. Обикновено една памет има вход за разрешаване на четенето, който показва дали в даден тактов период може да се четене от нея. Програмната памет се реализира, чрез статична памет с произволен достъп или SRAM. При тази памет адресирането предшества четенето на данните. Първо се подава адресът, а данните се извеждат в следващия тактов цикъл. В настоящата разработка е възприет подходът, всички действия в калкулатора да се изпълняват в рамките на един такт. В посочения масив изходните данни се извеждат в същия тактов цикъл, в който се подава входният адрес.

Реализация на АЛУ и логиката за декодиране на операции

Прочетеният елемент от програмната памет трябва да се декодира, за да се установи резултатът от коя от четирите операции извършвани от АЛУ да се използва (фиг. 5). Могат да се четат само 4-битови стойности. Това прави 16 възможни двоични комбинации. Елементите от *0* до *9* са цифрите, използвани във входния израз. Елементите от *A* до *D* са четирите математически операции, където *A* се използва за събиране, *B* – за изваждане, *C* – за умножение и *D* – за деление. Елементите E и F се използват съответно за край и начало на израза, подлежащ на изчисление от калкулатора.

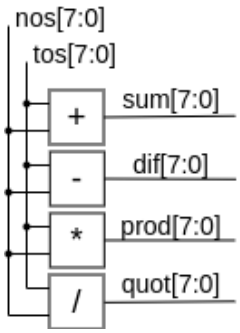



```
$op[3:0] = $reset ? 8'h0 : *instrs\[$pc\];
result =  $reset ? 8'h0 : ($op == 4'hA) ? $sum :
      ($op == 4'hB) ? $dif :
      ($op == 4'hC) ? $prod :
      ($op == 4'hD) ? $quot :
      ($op == 4'hE) ? $tos :
      ($op == 4'hF) ? 8'h0 : {4'h0, *instrs\[$pc\]};
```

Фигура 5. Схема и код на логиката за декодиране

Ако *$op* не съвпада с никоя от операциите (*A, B, C, D*) и не е нито край (*E*), нито начало (*F*) на израза, в стека ще се запише поредната цифра в израза от програмната памет. Аритметично-логическото устройство извършва паралелно

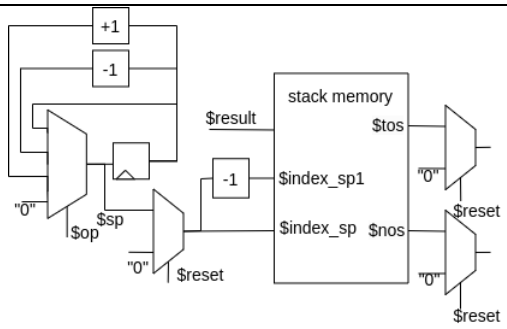
четири операции (фиг. 6). Работи се с прочетените от предходния тактов цикъл стойности от двата съседни елемента във върха на стека.




```
$sum[7:0] = >>1$tos + >>1$nos;
$dif[7:0] = >>1$tos - >>1$nos;
$prod[7:0] = >>1$tos * >>1$nos;
$quot[7:0] = >>1$tos / >>1$nos;
```

Фигура 6. Схема и код на аритметично-логическото устройство

Стек. Адресиране, четене и запис



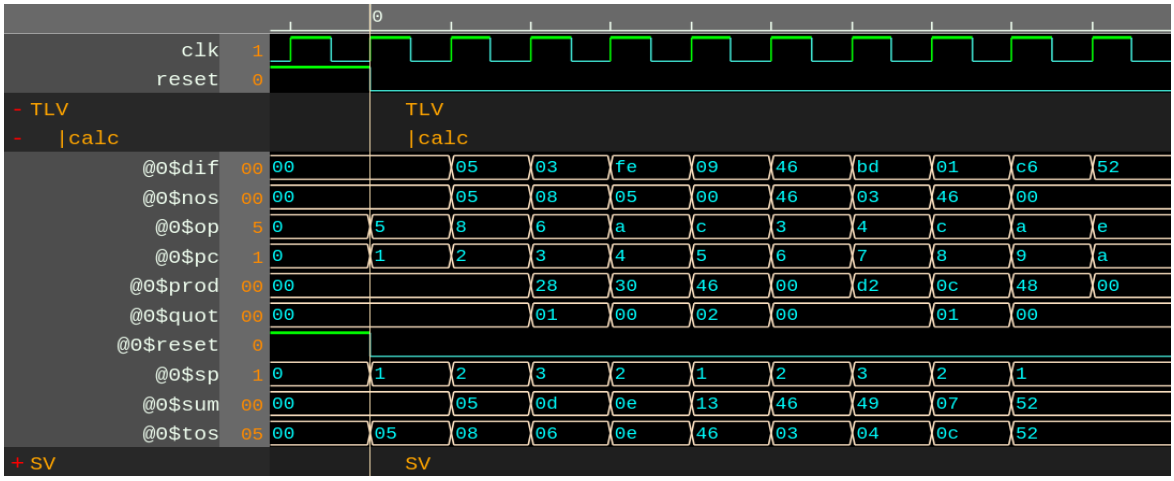

```
logic [7:0] elements[7:0];
$sp[3:0] = $reset ? 8'h0 : (($op == 4'hA) ? >>1$sp - 1:
      ($op == 4'hB) ? >>1$sp - 1 :
      ($op == 4'hC) ? >>1$sp - 1 :
      ($op == 4'hD) ? >>1$sp - 1 :
      ($op == 4'hE) ? >>1$sp :
      ($op == 4'hF) ? 4'h0 : >>1$sp + 1);
*elements\[$sp\] = result;
$tos[7:0]  = $reset ? 8'h0 : *elements\[$sp\];
$nos[7:0]  = $reset ? 8'h0 : *elements\[$sp-1\];
```

Фигура 7. Схема и код на стек с два канала за четене и един за запис

Подобно на програмната памет, паметта на стека представлява масив, наречен *elements[7:0]* (фиг. 7). Тя съдържа осем 8-битови елемента. Стекът има един входен порт за данни, на който се подава стойността, която да се запише в текущия връх. Номерът на елемента, явяващ се текущ връх се определя от индексиращия входен порт *$index_sp*. Логиката за адресиране ще предостави сигналите, необходими за четене от стека и за запис в него. Тя ще определи дали е необходимо четене от стека, за да се получат входните операнди. Ако е необходимо, от стека се извличат данните от съседните елементи, посочени от *$index_sp* и *$index_sp1*. Изходите, на които се извеждат прочетени данни са означени с *$tos* и *$nos*.

Симулация на разработения модел

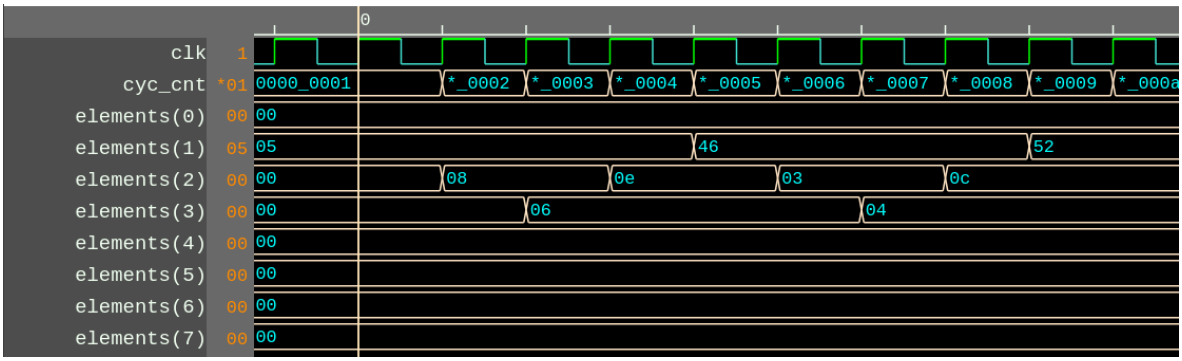
Резултатът от симулацията на модела е показана на фиг. 8. Стойностите в симулацията се изразяват с шестнадесетични цифри. Четирите операции се извършват паралелно, като за пресмятане се използват стойностите на сигналите $tos и $nos от предходния тактов период. В първия тактов период *$tos=08*, а *$nos=05*. Резултатите от операциите с тези две стойности за четирите операции са: *$sum=0d*$_{16}$*(13*$_{10}$*), $dif=03*$_{16}$*(3*$_{10}$*), $prod=28*$_{16}$*(40*$_{10}$*), $quot=01*$_{16}$*(1*$_{10}$*).*



Фигура 8. Резултат от симулацията на разработения модел на калкулатор

В сигнала *$op* се записват последователно елементите, извлечени от програмната памет. Сигналът *$sp* (указателят на стека) нараства с единица, когато извлечената стойност е число (от 0 до 9) и намалява с единица, когато извлечената стойност е между *a* и *f*. Сигналът *$tos* винаги съдържа последната записана в стека стойност, а *$nos* – предпоследната. От фиг. 8 се вижда че, в първия и втория тактов период, стойностите в *$tos* са *08*$_{16}$ и *06*$_{16}$, а стойностите в $nos са предходно постъпилите в стека числа *05*$_{16}$ и *08*$_{16}$.

На фиг. 9 се вижда как се променя съдържанието на стека във времето. Първоначално се записват стойностите *05*$_{16}$, *08*$_{16}$ и *06*$_{16}$. Четвъртият записан елемент е *0e*$_{16}$*(08*$_{16}$*+06*$_{16}$*)*. Резултатът *0e*$_{16}$ презаписва предпоследно постъпилия в стека елемент (*08*$_{16}$). По аналогичен начин могат да се проследят и останалите записи в паметта на стека.



Фигура 9. Резултат от симулацията на стека

Заключение

В настоящия доклад е предложен модел на калкулатор със стекова архитектура. За реализацията е използван езикът за проектиране на хардуер TL-Verilog. Моделът съдържа: програмна памет, в която е записан изразът за изчисляване в обратен полски запис; стек, който се индексира от два указателя, сочещи два съседни елемента във върха на стека; аритметично-логическо устройство, което извършва операциите спрямо елементите на записания в програмната памет израз. Поддържат се четири операции, които са целочислени. Създаденият модел е работоспособен. Той се използва за обучение на студенти и докторанти в дисциплини, свързани с проектиране на цифров хардуер в Технически университет – Габрово. Като бъдещо развитие на представената разработка се предвижда изграждането на набор от модели, демонстриращи различни микроархитектурни аспекти от работата на компютърните системи. Това ще допълни средствата за обучение, използвани в дисциплината Компютърни архитектури от учебните планове на специалностите „Компютърни системи и технологии“ и „Софтуерно и компютърно инженерство“ в Технически университет – Габрово.

Благодарности

Този доклад е подготвен и осъществен като част от проект № 2209E „Виртуална лаборатория за обучение по проектиране на цифров хардуер - II етап“, финансиран от средствата по бюджета за научни изследвания на Технически университет – Габрово.

References // Литература

[1] Building a RISC-V CPU Core… (n. d.). Course “Building a RISC-V CPU Core”, Available at: https://training.linuxfoundation.org/training/building-a-riscv-cpu-core-lfd111x (last view: 24-03-2023).

[2] Coussy, P.; Meredith, M.; Takach, A.; Gajski, D. (2009). “An Introduction to High-Level Synthesis” in IEEE Design & Test of Computers, vol. 26, no. 04, pp. 8-17, 2009. DOI: https://doi.org/10.1109/MDT.2009.69

[3] 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

[4] Hoover, S.; Salman, A. (2018). "Top-Down Transaction-Level Design with TL-Verilog", VSDOpen, 2018. https://arxiv.org/pdf/1811.01780.pdf (last view: 24-03-2023). DOI: https://doi.org/10.48550/arXiv.1811.01780

[5] Hoover, S. et al. (2018). „TL-X 2a HDL Extension Draft Syntax Specification, Work in Progress", 2018. https://www.TL-X.org (last view: 24-03-2023), 2018.

[6] Koopman, Ph. (1997). "Stack Computers: the new wave", 1997. http://users.ece.cmu.edu/~koopman/stack_computers/index.html (last view: 24-03-2023)

[7] Makerchip (n. d.). "TL–Verilog Tutorial", Available at: https://makerchip.com (last view: 24-03-2023).

[8] VSD - Pipelining RISC-V… (2018). Course "VSD - Pipelining RISC-V with Transaction-Level Verilog", 2018. Available at: https://www.udemy.com/course/vsd-pipelining-risc-v-with-transaction-level-verilog/ (last view: 24-03-2023).

Received: 30-03-2023 Accepted: 29-06-2023 Published: 24-07-2023
Cite as:
Kukenska, V.; Minev, P.; Varbov, I.; Dinev, M. (2023). "A Model for Online Learning in Digital Hardware Design", Science Series "Innovative STEM Education", volume 05, ISSN: 2683-1333, pp. 103-111, 2023. DOI: https://doi.org/10.55630/STEM.2023.0513