Мастер рад

Хардверска дигитална мрежа за сортирање

Студент: Алекса Величковић 2020/3358

Ментор: доц. др Живојин Шуштран

28. септембар 2023.

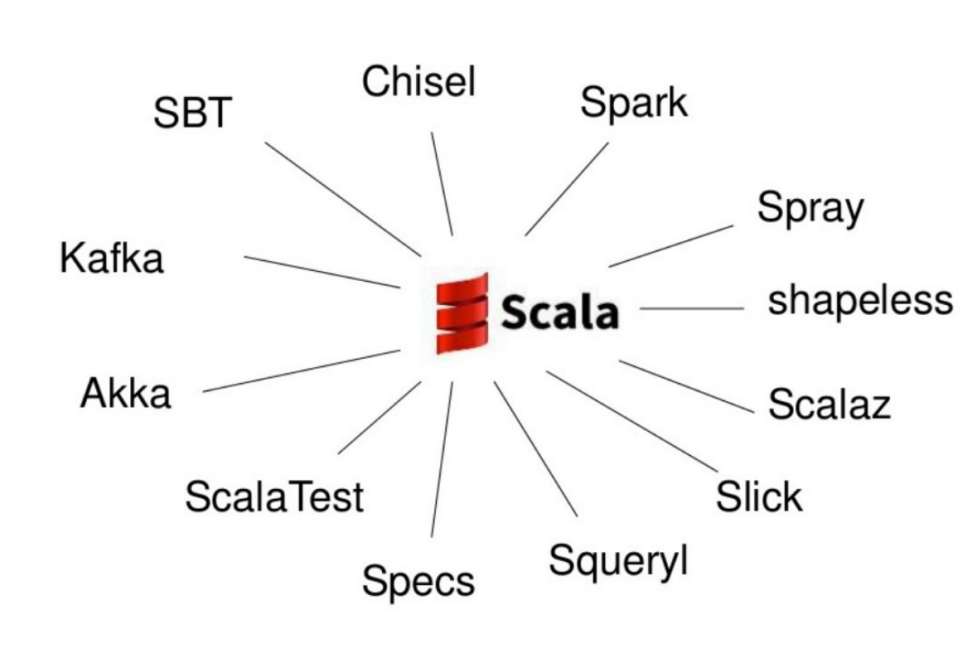
Универзитет у Београду
Електротехнички факултет

Рекурзивна имплементација битоник сортера у Чизелу

- Описана је рекурзивна имплементација хардверске дигиталне мреже за сортирање битоник сортер
- Имплементација је урађена коришћењем језика за опис хардвера Чизел, базиран на програмском језику Скала
  - Омогућава једноставније дефинисање хардвера од других језика и прегледнији програмски код
- Приказана су постојећа решења, са којим се врше поређења

Скала

- Скала (енг. *Scala*) је програмски језик који подржава и објектно-оријентисано и функционално програмирање
- Дизајниран је да буде концизан и да надомести недостатке које има програмски језик Јава (енг. *Java*)
- Постоје многи софтверски алати који су написани у Скали или су надограђени на овом програмском језику



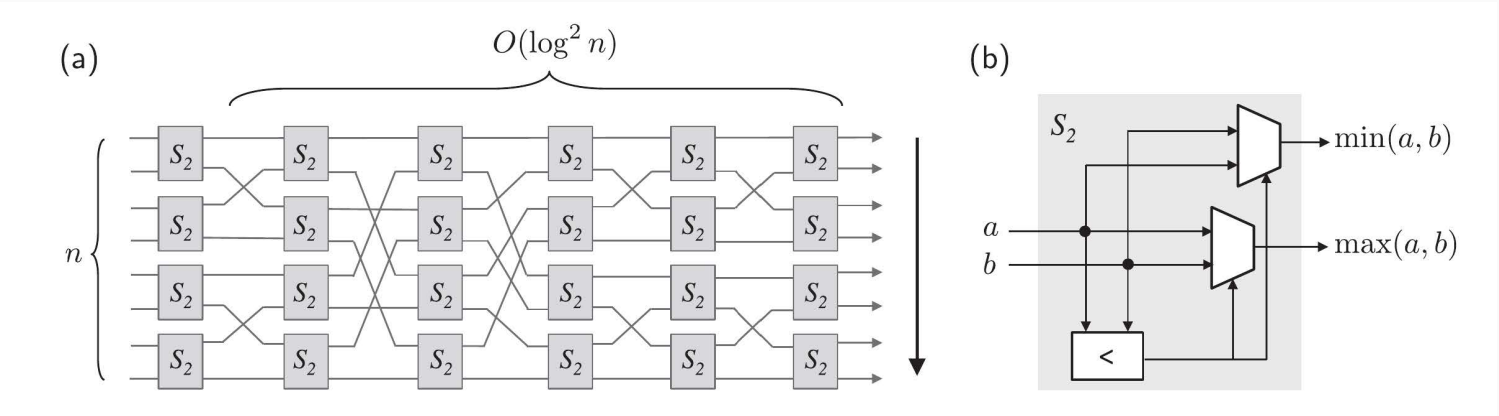
Чизел

- Чизел је језик за опис хардвера заснован на програмском језику Скала
  - *Chisel - Constructing Hardware in a Scala Embedded Language*
  - Наслеђује аспекте ООП и ФП Скале
  - Битно је разликовати који типови, оператори, условни изрази припадају Скали, а који Чизелу
  - Чизел програмски код може бити претворен у Верилог за синтезу и симулацију
  - Захтева много мање кода од Верилога

```
class RegisterModule extends Module {
  val io = IO(new Bundle {
    val in  = Input(UInt(12.W))
    val out = Output(UInt(12.W))
  })
  val register = Reg(UInt(12.W))
  register := io.in + 1.U
  io.out := register }
```

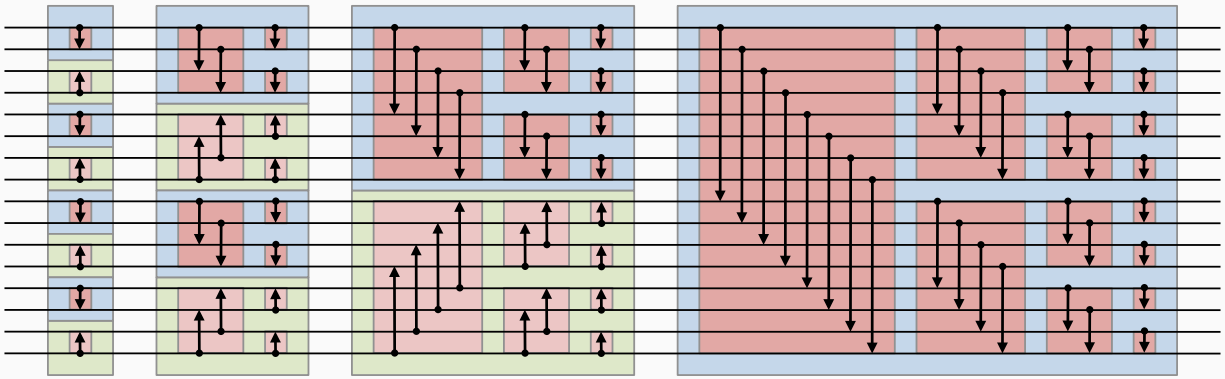
Мреже за сортирање

- Мреже за сортирање паралелно обрађују листу n улазних елемената кроз више нивоа тако да је на последњем нивоу улазна листа сортирана
- Број улазних вредности је степен броја два: $n = 2^t$
- Број нивоа и број поређења по нивоу одређују цену мреже
  - Битоник сортер на слици има $\log_2(n)(\log_2(n)+1)/2$ нивоа са $n/2$ паралелних S_2 операција, укупно $\mathcal{O}(n \log^2 n)$ оп.



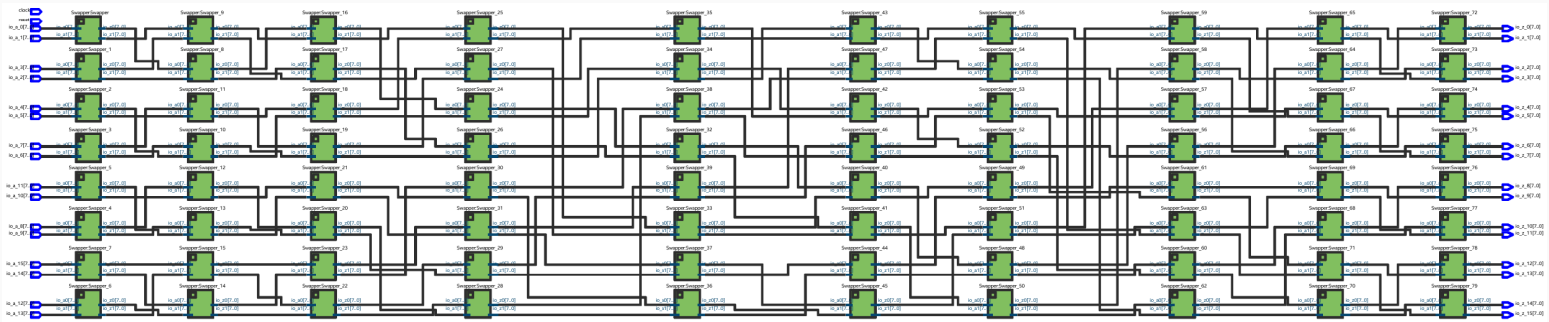
Битоник сортер

- *Bitonic mergesort* алгоритам је објавио Кенет Едвард Бечер 1968. године. Такође је развио и *пар-непар* (*odd-even*) мреже за сортирање
  - Најпопуларнија решења, због једноставности, и близине оптималној цени и перформансама
- Битоник низ се састоји из два монотона низа, растућег и опадајућег
- Унутар црвених правоугаоника се пореде елементи из горње половине са одговарајућим из доње. Излаз из плавог је растући, из зеленог опадајући



Итеративна имплементација битоник сортера у Чизелу

- Програмски код искоришћен за имплементацију рекурзивног решења
- Класа `Swapper` дефинише хардверски модул за замену два улазна елемента на основу функције поређења.
- `insertSorter` је помоћна функција која врши замену елемената задатим са `lo` и `hi`
- Главна логика се извршава у петљи, где `i` представља битну позицију која се разматра, `j` варира од `i` до `0`, и `k0` и `k1` итерирају кроз елементе који треба да се пореде
- Генерисан је Верилог код и извршена синтеза у Quartus-у



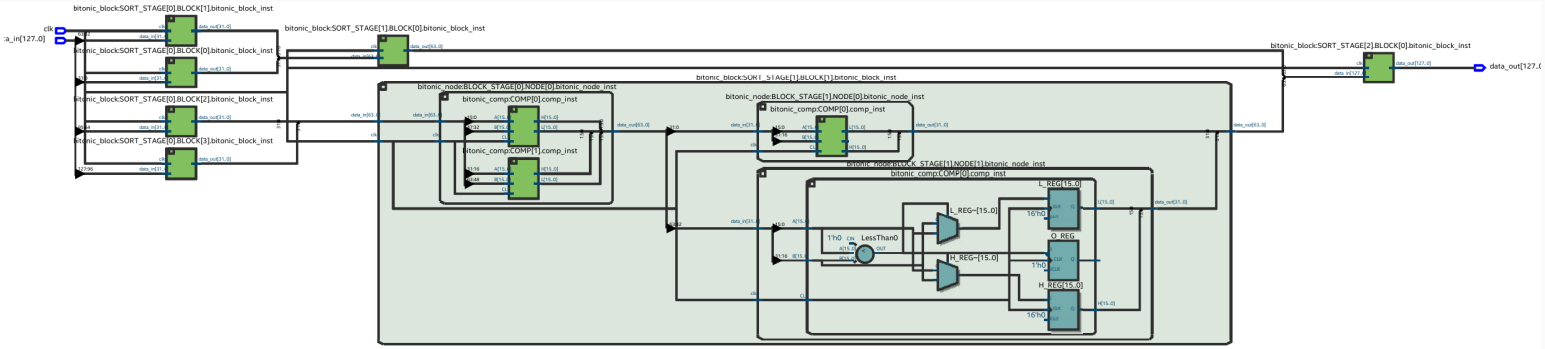
Рекурзивна имплементација у Пајтону

- `bitonicMerge(a, low, cnt, dire)` функција рекурзивно сортира битоник низ у растућем или опадајућем редоследу, у зависности од вредности параметра `dire`. Низ се сортира почевши од позиције `low`, а параметар `cnt` представља број елемената који се сортирају.
- `bitonicSort(a, low, cnt, dire)` функција прво производи битоник низ рекурзивним сортирањем његових две половине у супротним редоследима, а затим позива функцију `bitonicMerge` да их уједини у исти редослед

```
def bitonicMerge(a, low, cnt, dire):
  if cnt > 1:
    k = cnt//2
    for i in range(low , low+k):
        compAndSwap(a, i, i+k, dire)
    bitonicMerge(a, low, k, dire)
    bitonicMerge(a, low+k, k, dire)
```

Имплементација битоник сортера у Верилогу

- `bitonic_comp` - компаратор и логика замене
- `bitonic_node` - основна јединица битоник мреже
  - Производи сортиране излазе, састоји се од `bitonic_comp`
- `bitonic_block` представља степен (ниво) у битоник мрежи за сортирање
  - Унутар овог модула има више инстанци модула `bitonic_node` које чине битоник мрежу овог степена
- `bitonic_sort` модул је на највишем нивоу и представља потпуну битоник мрежу за сортирање.
  - Управља сортирањем података делећи га на степене и блокове користећи модуле `bitonic_block` и `bitonic_node`.



Рекурзивна имплементација битоник сортера у Чизелу

- Као основа за израду ове имплементације су искоришћене итеративна у Чизелу и рекурзивна у Пајтону
- Објекат `BitonicSorter` дефинише функције: `bitonicMerge()`, `bitonicSort()` и `sort()`.

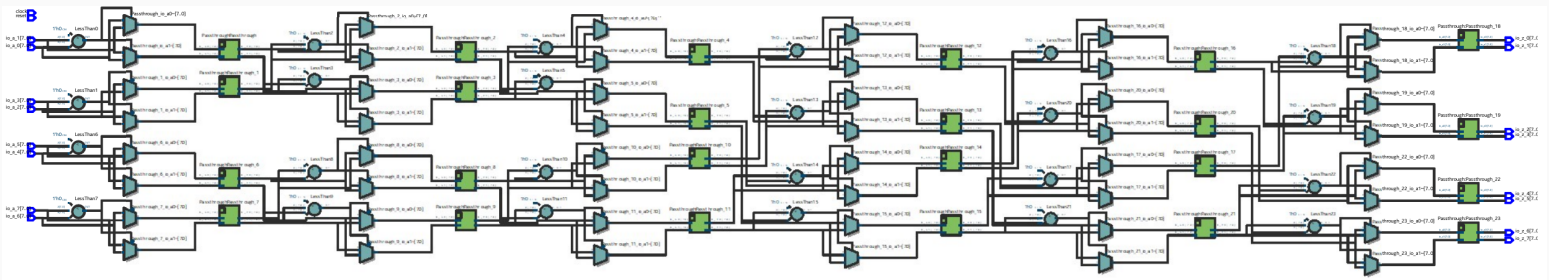
```
def swapIfNecessary(...): IndexedSeq[Option[T]] = {
  val j = i + length / 2
  val si = arr(i).getOrElse(0.U.asTypeOf(arr(i).get))
  val sj = arr(j).getOrElse(0.U.asTypeOf(arr(j).get))
  val m = Module(new Passthrough(si.cloneType))
  val swapNeeded = lt(sj, si) === ascending
  when(swapNeeded) {
    m.io.a0 := sj
    m.io.a1 := si
  }.otherwise {
    m.io.a0 := si
    m.io.a1 := sj }
  seq.updated(i, Some(m.io.z0)).updated(j, Some(m.io.z1)) }
```

Рекурзивна имплементација битоник сортера у Чизелу

- Потребно је генерисати нову листу са замењеним (свапованим) елементима у одговарајућем редоследу, у `bitonicMerge()` функцији, искоришћена је `foldLeft()` функција, која извршава функцију `swapIfNecessary()`

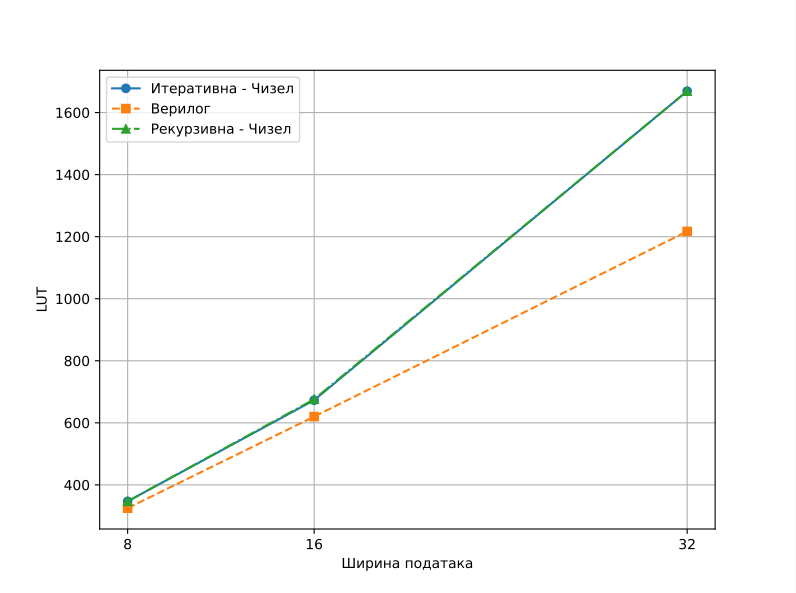
```
val indices = 0 until half
val swappedSeq = indices.foldLeft(arr) { (seq, i) =>
  swapIfNecessary(seq, i, ascending, lt) }
```

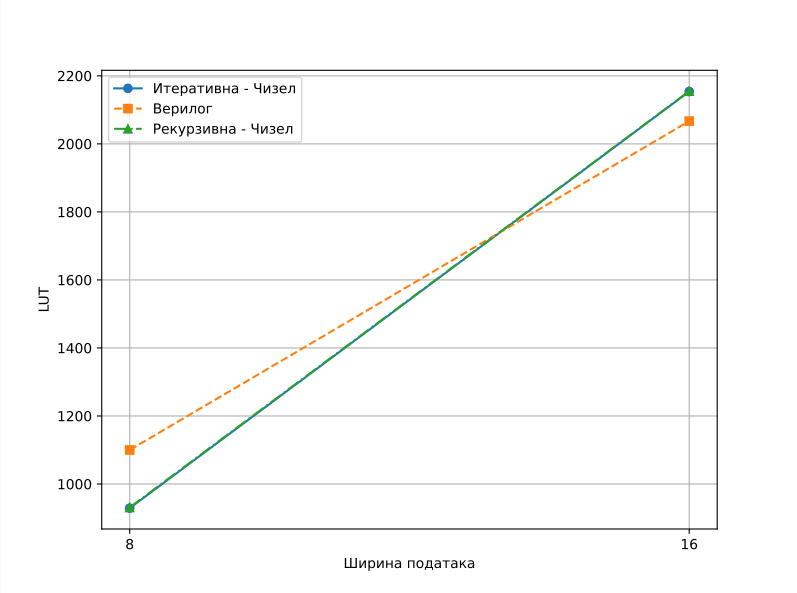
- Тестирање је извршено за низове дужине 1, 2, 4, 8, 16 који се састоје од насумичних бројева ширине 8, 16, 32, 64 бита
- Позивом `getVerilog(new BitonicSorterModule(...)` је добијен Верилог код и синтеза је извршена у Quartus-у



Поређење са претходним имплементацијама

- Следећи графици приказују број Look-up табела потребних за синтезу приказаних имплементација на FPGA уређају Cyclone V 5CGXFC9E7F35C8 када је број елемената за сортирање 8 и 16
- Итеративна и рекурзивна имплементација у Чизелу заузимају готово исти простор на чипу





Поређење са претходним имплементацијама

- Потрошња енергије изражена у [mW]

Број ел.	Ширина	Вер.	Итер.	Рек.
8	8	530.08	530.06	531.20
8	16	535.70	535.81	536.66
8	32	546.12	546.50	545.98
16	8	535.41	535.40	535.59
16	16	546.31	545.60	546.36

- Број линија Верилог програмског кода

Број ел.	Ширина	Рекурзивна	Итеративна
8	8	356	327
8	16	368	327
8	32	368	327
16	8	1158	1023
16	16	1158	1023

Закључак

- Чизел нуди могућност дизајнирања хардвера са начином размишљања као да је у питању програмирање у језику вишег нивоа
- Рекурзивна имплементација битоник сортера заузима исти простор на чипу као итеративна
  - Иако шема на први поглед делује компликованије, а сам програмски код је интуитивнији
- Шема би била прегледнија ако би имплементација била састављена од подмодула, а не појединачних оператора
  - Делује компликованије од итеративног решења, а сам програмски код је интуитивнији
- Анализирати битоник сорт алгоритам и уочити још неке начине рекурзивне имплементације чиме би се добила јаснија шема резултујућег кола