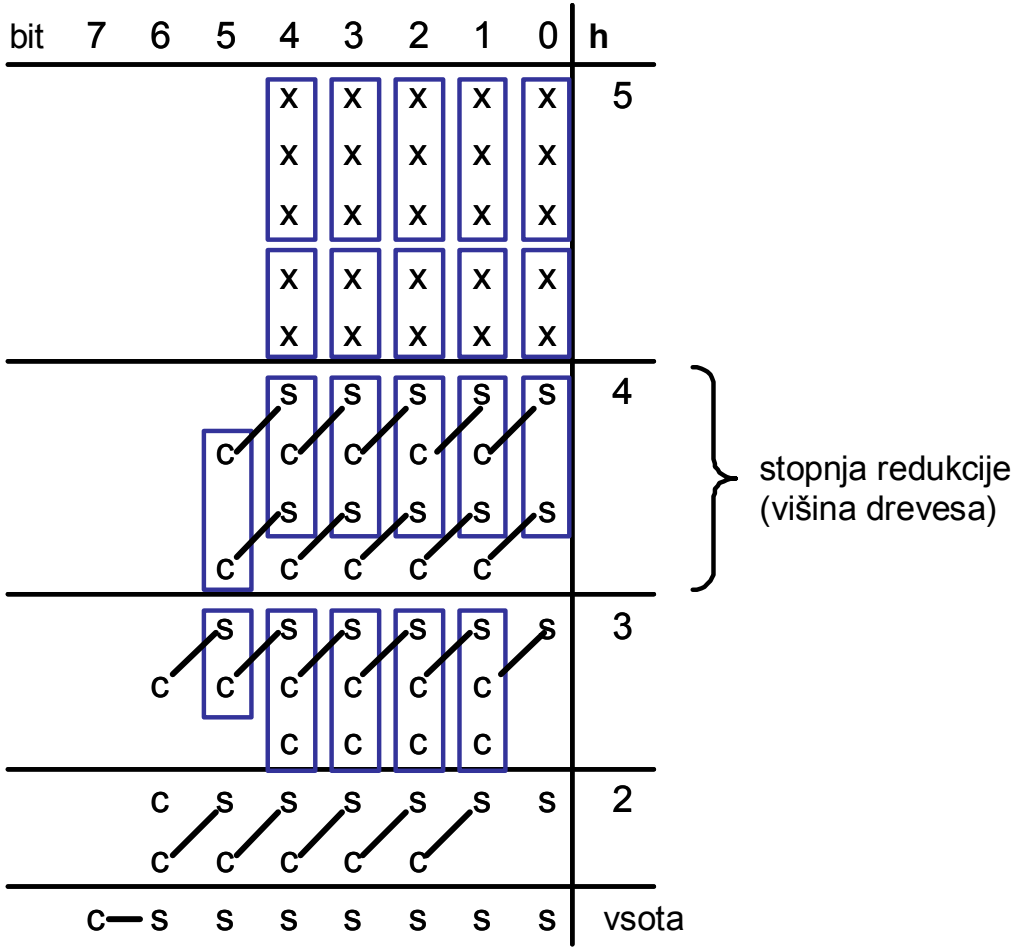
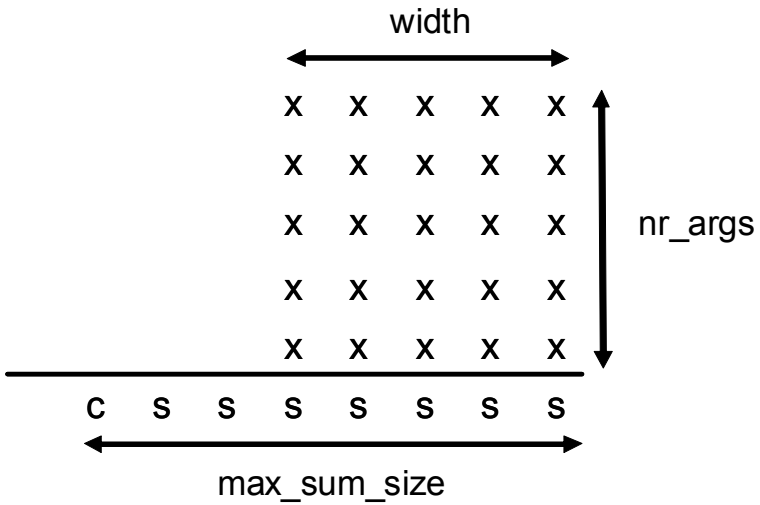
# Vzporedno seštevanje s splošno Wallace−ovo drevesno strukturo




Pri Wallace−ovi drevesni strukturi reduciramo začetno višino drevesa tako, da postavimo polni in polovični seštevalnik tam kjer je možno (čimprej). Postavljamo jih tako, da pri tem zaobjamemo karseda največje število bitov v stolpcu drevesne strukture. Primer vzporednega seštevanje petih nepredznačenih 5−bitnih števil prikazuje spodnja slika. Na začetnem nivoju redukcije postavimo na posamezno utež seštevanja (stolpec) en polni in en polovični seštevalnik (obkrožena). V naslednji redukciji se zato v LSB stolpcu pojavita dva bita vsote. Prenosa s seštevanja LSB stopnje se pojavita na LSB+1 mestu. Na sliki sta prenosa, ki izvirata iz določenega seštevanja povezana z vsoto s črto.

Na LSB+1 mestu začetne iteracije se podobno seštejeta polni in polovični seštevalnik. Vsoti se vpišeta v LSB+1 stolpec, prenosa pa v LSB+2 stolpec. Če opisani postopek izvedemo na vseh mestih (LSB do LSB+4) dobimo zmanjšano višino Wallace−ovega drevesa. (h=4). Postopek ponavljamo, dokler višina drevesa ni enaka 2. Mesta zadnje iteracije seštejemo s splošnim seštevalnikom s širjenjem prenosa CPA (ang. Carry Propagate Adder), ki je lahko izveden kot seštevalnik z valovljenjem prenosa (RC - ang. Ripple Carry) ali seštevalnik z vnaprejšnim izračunom prenosa (CLA - ang. Carry Look Ahead Adder).

V VHDL programirajte arhitekturo splošne strukture Wallace−ovega drevesnega seštevalnika, ki vzporedno sešteva polje nepredznačenih operandov enake velikosti. Število operandov je (**nrargs**), širina posameznega operanda je (**width**). Dobljena vsota je velikosti (**max_sum_size**). Izhodišče problema vzporednega seštevanja povzema spodnja slika na primeru seštevanja petih 5−bitnih števil:




**Slika 1: Izhodišče vzporednega seštevanja petih 5−bitnih števil.**

Entiteta izdelane strukture **wallace_addition** ima priključke:

```
ENTITY wallace_addition IS
GENERIC (
      width : INTEGER := 4;   --širina posameznega operanda
      nrargs : INTEGER := 4   --število operandov
);
PORT (
      x     :     IN ArrayOfAddends(width - 1  DOWNTO 0, nrargs - 1  DOWNTO 0);    -- polje operandov
      sum   :     OUT STD_LOGIC_VECTOR(sizeof( nrargs * ( 2**width - 1)) - 1 DOWNTO 0)   -- vsota
);
END wallace_addition;
```

Naloge:

1. Ustvarite VHDL datoteko **Wallace_tree_functions.vhd** v kateri boste znotraj VHDL paketa (**PACKAGE**) programirali funkcije, potrebne za izvedbo sinteze drevesne strukture splošnega Wallace−ovega vzporednega seštevalnika. V datoteki se nahaja tudi definicija vhodnega tipa dvodimenzionalnega polja (**ArrayOfAddends**). Podani so prototipi funkcij z vhodnimi parametri in opisom delovanja:

```
PACKAGE Wallace_tree_functions IS

    --    @Type name: sizeof
    -- @Parameters:
    --    argument 1: x dimenzija polja
    --    argument 2: y dimenzija polja
    --    @Description:
    --    definicija tipa splošnega dvodimenzionalnega polja (x, y) bitov tipa STD_LOGIC
    Type ArrayOfAddends is array (natural range <>, natural range <>) of STD_LOGIC;

    --    @Function name: sizeof
    -- @Parameters:
    --    a: vhodno število
    --    @Return:
    --    Vrne število bitov, potrebnih za zapis števila a
    FUNCTION sizeof (a: NATURAL) RETURN NATURAL;    --

    --    @Function name: prev_lvl_carry_rect
    -- @Parameters:
    --    height: višina Wallaceove drevesne strukture na danem nivoju redukcije
    --    arg_width: velikost operanda Wallaceove drevesne strukture na danem nivoju redukcije
    --    this_weight: Utež (stolpec) Wallaceove drevesne strukture na danem nivoju redukcije
    --    this_lvl: nivo redukcije Wallaceove drevesne strukture
    --    @Return:
    --    Število bitov prenosa za dani stolpec podanega nivoja redukcije Wallaceove drevesne strukture (this_lvl)
    FUNCTION prev_lvl_carry_rect (height: NATURAL; arg_width: NATURAL; this_weight: NATURAL; this_lvl: NATURAL)
RETURN NATURAL;

    --    @Function name: this_lvl_bits_rect
    -- @Parameters:
    --    height: višina Wallaceove drevesne strukture na danem nivoju redukcije
    --    arg_width: velikost operanda Wallaceove drevesne strukture na danem nivoju redukcije
    --    this_weight: Utež (stolpec) Wallaceove drevesne strukture na danem nivoju redukcije
    --    this_lvl: nivo redukcije Wallaceove drevesne strukture
```

```
    --    @Return:
    --    Število bitov v danem stolpcu podanega nivoja redukcije Wallaceove drevesne strukture (this_lvl)
    FUNCTION this_lvl_bits_rect (height: NATURAL; arg_width: NATURAL; this_weight: NATURAL; this_lvl: NATURAL)
RETURN NATURAL;

    --    @Function name: num_full_adders_rect
    -- @Parameters:
    --    height: višina Wallaceove drevesne strukture na danem nivoju redukcije
    --    arg_width: velikost operanda Wallaceove drevesne strukture na danem nivoju redukcije
    --    this_weight: Utež (stolpec) Wallaceove drevesne strukture na danem nivoju redukcije
    --    this_lvl: nivo redukcije Wallaceove drevesne strukture
    --    @Return:
    --    Število polnih seštevalnikov v danem stolpcu podanega nivoja redukcije Wallaceove drevesne strukture
(this_lvl)
    FUNCTION num_full_adders_rect (height: NATURAL; arg_width: NATURAL; this_weight: NATURAL; this_lvl: NATURAL)
RETURN NATURAL;

    --    @Function name: num_half_adders_rect
    -- @Parameters:
    --    height: višina Wallaceove drevesne strukture na danem nivoju redukcije
    --    arg_width: velikost operanda Wallaceove drevesne strukture na danem nivoju redukcije
    --    this_weight: Utež (stolpec) Wallaceove drevesne strukture na danem nivoju redukcije
    --    this_lvl: nivo redukcije Wallaceove drevesne strukture
    --    @Return:
    --    Število polnih seštevalnikov v danem stolpcu podanega nivoja redukcije Wallaceove drevesne strukture
(this_lvl)
    FUNCTION num_half_adders_rect (height: NATURAL; arg_width: NATURAL; this_weight: NATURAL; this_lvl: NATURAL)
RETURN NATURAL;
END Wallace_tree_functions;
```

Navodila za izdelavo funkcij:

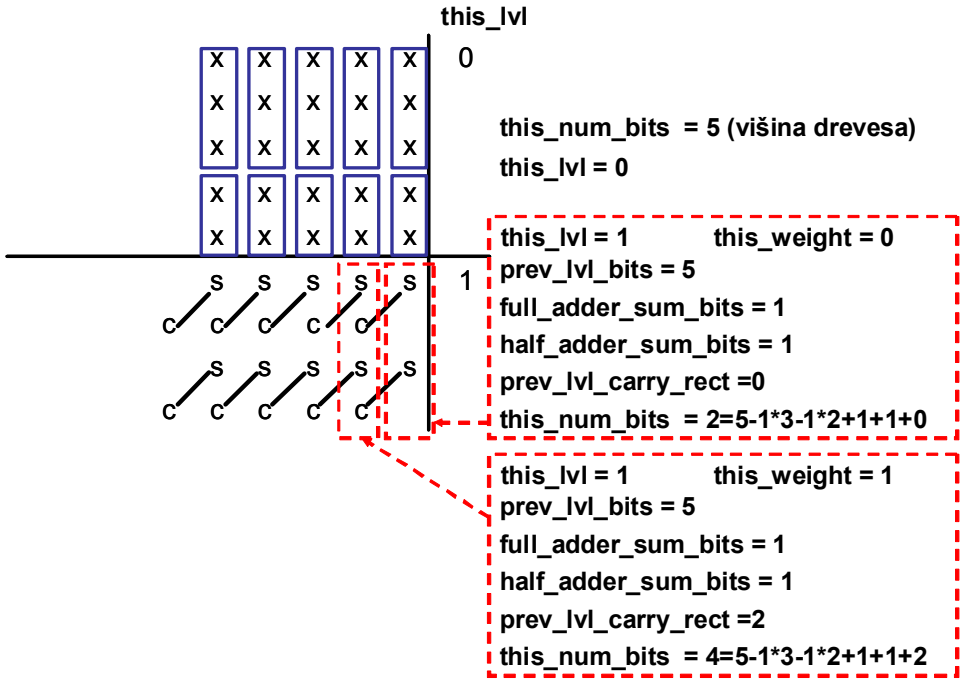
1. Funkcijo `sizeof` izdelajte tako, da definirate interno spremenljivko (`VARIABLE nr : NATURAL := a;`), ki jo znotraj (`FOR ... LOOP`) zanke iterativno delite z 2 dokler je rezultat deljenja večji od 0. Število deljenj shranite v spremenljivko, ki jo funkcija po izteku zanke (`FOR ... LOOP`) vrne.

2. Funkcija (`this_lvl_bits_rect`) vrne število bitov (`this_num_bits`) ob redukciji stolpca iz prejšnje stopnje redukcije Wallace−ovega drevesa (`this_lvl - 1`) na opazovano stopnjo redukcije (`this_lvl`). Število bitov (`this_num_bits`) ob redukciji stopnje stolpca dobimo, če od števila bitov stolpca na prejšnjem nivoju (`prev_lvl_bits`) odštejemo število polnih (`full_adder_sum_bits`) in polovičnih (`half_adder_sum_bits`) seštevalnikov ter število prenosov iz prejšnje stopnje. Slednje dobite s klicem funkcije (`prev_lvl_carry_rect`).

   Za hranjenje podatkov definirajte interne spremenljivke (`VARIABLE`) kot naravna števila (`NATURAL`):
   − (`prev_lvl_bits`), ki hrani število bitov prejšnje stopnje redukcije (`this_lvl - 1`),
   − (`full_adder_sum_bits`), ki hrani število polnih seštevalnikov na prejšnji stopnji redukcije,
   − (`half_adder_sum_bits`), ki hrani število polovičnih seštevalnikov na prejšnji stopnji redukcije,
   − (`this_num_bits`), ki hrani število bitov na trenutni stopnji redukcije.

   Število bitov na prejšnjem nivoju (`prev_lvl_bits`) dobite z uporabo rekurzivnega klica funkcije (`this_lvl_bits_rect`) na prejšnji stopnji redukcije (`this_lvl - 1`). Če gre za prvo redukcijo Wallace−ovega drevesa, mora biti rezultat funkcije (`this_num_bits`) enak višini drevesa (`height`).
   Število polnih seštevalnikov (`full_adder_sum_bits`) dobite tako, da število bitov na prejšnjem nivoju delite s številom vhodov, ki jih sešteva posamezen polni seštevalnik (3). Število polovičnih seštevalnikov (`half_adder_sum_bits`) dobite tako, da od števila bitov prejšnje stopnje (`prev_lvl_bits`) odštejete število vhodov, ki jih seštevajo polni seštevalniki (`full_adder_sum_bits*3`). Dobljeno razliko delite s številom vhodov, ki jih sešteva polovični seštevalnik (2). Preostanek bitov pri redukciji stopnje (`this_num_bits`) dobite tako, da od števila bitov prejšnje stopnje odštejete število bitov, ki so jih na stopnji sešteli polni seštevalniki in polovični seštevalniki. Tej razliki je potrebno prišteti še bite vsote polnih in polovičnih seštevalnikov ter prenose prejšnjih stopenj.

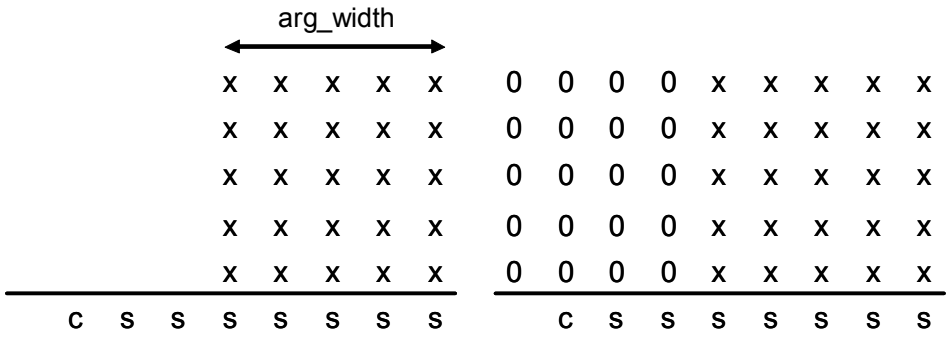
Slika 2 prikazuje primer prve stopnje redukcije seštevanja petih 5 bitnih operandov za LSB in LSB+1 utež (stolpec).




**Slika 2: Prva stopnje redukcije seštevanja petih 5−bitnih operandov za LSB in LSB+1 stolpec.**

Za uteži, ki so večje od MSB operanda (`arg_width`) mora funkcija pred prvo stopnjo redukcije (`this_lvl=0`) vrniti (`this_num_bits=0`), saj seštevamo nepredznačena števila, kot je prikazano na sliki 3.

Slika 3 prikazuje seštevanje petih 5−bitnih operandov, ki imajo razširjen pozitivni predznak z '0'.




**Slika 3: Razširjanje bitov predznaka seštevanja na izhodiščni stopnji redukcije.**

3. Funkcija (`prev_lvl_carry_rect`) vrne število bitov prenosa (`num_carry`) ob redukciji stolpca iz prejšnje stopnje redukcije Wallace−ovega drevesa (`this_lvl - 1`). Število bitov prenosa dobimo, če od najprej ugotovimo števila bitov stolpca na prejšnjem nivoju (`prev_lvl_bits`). Število prenosov polnih seštevalnikov dobite tako, da število bitov stolpca na prejšnjem nivoju delimo s 3. Število prenosov polovičnih seštevalnikov dobimo tako, da preostanek bitov stolpca delimo z 2. Na LSB stolpcu je število prenosov vedno '0'. Na začetni stopnji redukcije (`this_lvl=0`) je izračun števila prenosov enak, le namesto števila bitov prejšnje stopnje upoštevamo kar višino drevesa (`height`).

   Za hranjenje podatkov definirajte interne spremenljivke (`VARIABLE`) kot naravna števila (`NATURAL`):
   − (`num_carry`), ki hrani število prenosov na prejšnji stopnji redukcije,
   − (`prev_lvl_bits`), ki hrani število bitov na prejšnji stopnji redukcije.
   Število bitov na prejšnjem nivoju (`prev_lvl_bits`) izračunate s klicem prej opisane funkcije (`this_lvl_bits_rect`) na prejšnji stopnji redukcije (`this_lvl - 1`).

4. Funkcija (`num_full_adders_rect`) vrne število polnih seštevalnikov ob na dani stopnji redukcije (`this_lvl`) stolpca (`this_weight`). Število polnih seštevalnikov dobite tako, da ugotovite število bitov stolpca na trenutnem nivoju (`this_num_bits`) s klicem funkcije (`this_lvl_bits_rect`). Dobljeno število delite s 3 in rezultat vrnete.

Za hranjenje podatkov definirajte interne spremenljivke (**VARIABLE**) kot naravna števila (**NATURAL**):

− (**this_num_bits**), ki hrani število bitov na trenutni stopnji redukcije.

Število bitov stolpca (**this_num_bits**) izračunate s klicem prej opisane funkcije (**this_lvl_bits_rect**) na opazovani stopnji redukcije (**this_lvl**).

5. Funkcija (**num_half_adders_rect**) vrne število polovičnih seštevalnikov ob na dani stopnji redukcije (**this_lvl**) stolpca (**this_weight**). Število polovičnih seštevalnikov dobite tako, da ugotovite število bitov stolpca na trenutnem nivoju (**this_num_bits**) s klicem funkcije (**this_lvl_bits_rect**) na opazovani stopnji redukcije (**this_lvl**). S klicem prej opisane funkcije (**num_full_adders_rect**) izračunate število polnih seštevalnikov na opazovani stopnji (**num_full_adds**). Razliko števila bitov (**this_num_bits**) in števila bitov, ki jih seštevajo polni seštevalniki (**num_full_adds*3**) delite z 2 in rezultat vrnete.

   Za hranjenje podatkov definirajte interne spremenljivke (**VARIABLE**) kot naravna števila (**NATURAL**):

   − (**this_num_bits**), ki hrani število bitov na trenutni stopnji redukcije.

   − (**num_full_adds**), ki hrani število polnih seštevalnikov na trenutni stopnji redukcije.

2. Ustvarite datoteko **wallace_addition_unsigned.vhd** v kateri programirajte entiteto in arhitekturo splošne strukture Wallace−ovega drevesnega seštevalnika, ki vzporedno sešteva (**nrargs**) nepredznačenih operandov enake velikosti (**width**). Ime entitete mora biti: **wallace_addition**. Podana je entiteta strukture:

```
ENTITY wallace_addition IS
GENERIC (
      width : INTEGER := 4;   --širina operanda
      nrargs : INTEGER := 4   --število operandov
      );
PORT (
x     :     IN ArrayOfAddends(width - 1  DOWNTO 0, nrargs - 1  DOWNTO 0);    -- polje bitov operandov
sum   :     OUT STD_LOGIC_VECTOR(sizeof( nrargs * ( 2**width - 1)) - 1 DOWNTO 0)   -- vsota
);
END wallace_addition;
```

Vhod strukture je splošno dvodimenzionalno polje elementov tipa (**ArrayOfAddends**). S tem tipom je podanih (**nrargs**) (**width**) bitnih števil kot dvodimenzionalno polje elementov (**STD_LOGIC**). Izhod strukture je vsota (**sum**) tipa (**STD_LOGIC_VECTOR**), ki ima toliko mest, da lahko shrani vsoto (**nrargs**) nepredznačenih (**width**) bitnih števil. V tem VHDL modulu bomo uporabljali prej izdelano paketno datoteko zato jo vključimo z ukazom:

```
USE work.Wallace_tree_functions.all;
```

Definirajte celoštevilski naštevni tip (**nr_stages_type**), v katerem definirate število stopenj redukcije drevesne strukture Wallace−ovega drevesa. Vrednosti povzema spodnja tabela:

| **n** | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **h** | 9 | 9 | 9 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 5 | 5 | 4 | 3 | 3 |

Višina (**h**) oz. število stopenj Wallace−ovega drevesa, ki je potrebna za redukcijo **n** operandov na vhodu na 2−bitni izhod je podana s spodnjo rekurzivno relacijo. Za redukcijo dveh operandov (n=2) ni potrebno imeti Wallace−ovega drevesa (h=0). Z višino drevesa h=1 lahko reduciramo največ n=3 števila na vhodu.

$$h(n) = 1 + h(\lceil 2n/3 \rceil) \qquad h(2) = 0$$

Formulo lahko tudi obrnemo, tako da izrazimo največje število operandov (**n**), ki jih lahko opišemo z določeno višino drevesa (**h**).

$$n(h) = \lfloor 3n(h\text{-}1)/2 \rfloor \qquad n(0) = 2$$

Z višino drevesa h=9 je sicer možno opisati seštevanje največ n=63 vhodov. V našem primeru se bomo omejili na seštevanje največ 32 argumentov. Izdelajte konstanto naštevnega tipa (**nr_stages**) Ena možnost opisovanja tovrstnega podatkovnega tipa je rekurzivna funkcija. Namesto nje bomo raje uporabili naštevni tip, ki izhaja iz zgornje tabele.

Za indeks naštevnega tipa definirajte enodimenzionalno polje celoštevilskih elementov z imenom (**nr_stages_type**). Indeks polja teče od 32 do 3.

Definirajte konstanto (**nr_stages**) tipa (**nr_stages_type**), katere vrednosti predstavljajo višine (h) iz zgornje tabele.

Končno definirajte celoštevilsko konstanto (**stages**), ki izračuna število nivojev redukcije Wallace−ovega drevesa:

```
CONSTANT stages: INTEGER := nr_stages( nrargs );  --število stopenj redukcije Wallace-ovega drevesa
```

Definirajte konstanto (**max_sum_size**), ki predstavlja število mest vsote s pomočjo prej izdelane funkcije (**sizeof**), kot je prikazano na Slika 1.

```
CONSTANT max_sum_size: INTEGER := sizeof( nrargs * ( 2**width - 1));
```

Definirajte tip (**cell_type**), ki predstavlja razširjeno dvodimenzionalnem polje mest števil. Dimenzije polja so (**max_sum_size, nrargs**).

Definirajte tip (**W_type**), ki predstavlja enodimenzionalnem polje tipa(**cell_type**). Dimenzija polja je število stopenj redukcije (višina drevesa) (**stages**).

Tako smo definirali tridimenzionalno polje v VHDL, katerega prva dimenzija opisuje dano stopnjo redukcije, preostali dimenziji pa definirata položaj posameznega bita (vrstica, stolpec) v stopnji redukcije. Posamezne celice drevesa definirajte kot signal (**W**):

```
SIGNAL W : W_type := (others => (others =>(others =>'0'))); -- Wallace-ovo drevo
```

Uporabite komponento CLA seštevalnika iz predloge vaje:

```
ENTITY cla_add_n_bit IS
   generic(n: natural := 8);
   PORT (      Cin   :     in    std_logic ;
               X, Y  :     in    std_logic_vector(n-1 downto 0);
               S     :     out   std_logic_vector(n-1 downto 0);
               Gout,
               Pout,
               Cout  :     out   std_logic);
END cla_add_n_bit;
```

CLA seštevalnik bo seštel (**max_sum_size**) bitov na zadnji stopnji redukcije Wallace−ovega drevesa.

Definirajte signale (**add_a, add_b, add_sum**) tipa (**STD_LOGIC_VECTOR**) velikosti (**(max_sum_size**), ki so potrebni za povezovanje vhodnih argumentov in izhodne vsote CLA seštevalnika.

V arhitekturi definirajte dva procesa: Prvi bo opisoval povezovanje polovičnih in polnih seštevalnikov v Wallace−ovem drevesu, drugi bo povezoval izhod zadnje redukcije Wallace−ovega drevesa na CLA seštevalnik.

Prvi proces z imenom (**wallace_proc**) je odvisen od polja vhodnih operandov (**X**) in tridimenzionalnega polja bitov (**W**). Znotraj procesa definirajte še naslednje interne spremenljivke (**VARIABLE**) kot naravna števila (**NATURAL**):

| | |
|---|---|
| this_carry_bits | Število prenosov stolpca trenutne stopnje redukcije drevesa |
| this_stage_bits | Število bitov stolpca trenutne stopnje redukcije drevesa |
| num_full_adds | Število polnih seštevalnikov stolpca trenutne stopnje redukcije drevesa |
| num_half_adds | Število polovičnih seštevalnikov stolpca trenutne stopnje redukcije drevesa |
| num_wires | Število prostih bitov stolpca trenutne stopnje redukcije drevesa |

Prvi del procesa predstavlja povezovanje bitov polja vhodnih operandov (**X**) na ustrezna mesta začetne (ničte) redukcije Wallace−ovega drevesa (**W**). Do mesta na koordinatah (**i, j**) začetne redukcije dostopate tako: **W(0)( i, j) .** Povezovanje mest izvedite v dvojni gnezdeni (**FOR ... LOOP**) zanki po obeh koordinatah.

Po izvedenem povezovanju začetne redukcije izvedemo preostale redukcije znotraj dvojne gnezdene (`FOR ... LOOP`) zanke – najprej po indeksu (`k`), nato po indeksu (`i`). Na k–ti stopnji redukcije se izračuna:

- število prenosov na tej stopnji z uporabo funkcije (`prev_lvl_carry_rect`):

```
this_carry_bits := prev_lvl_carry_rect( nrargs, width, i, k + 1);
```

- število polnih seštevalnikov na k–ti stopnji.

```
num_full_adds := num_full_adders_rect( nrargs, width, i, k);
```

Definirate (`FOR ... LOOP`) zanko, ki teče po indeksu bitov posameznega stolpca (`j`). Položaj (`j`) pomeni absolutno lego prvega bita polnega seštevalnika v stolpcu stopnje redukcije. V zanki se povežejo mesta naslednje (`k+1`) stopnje redukcije za vsoto (`W( k+1 )( i, this_carry_bits + j)`) in prenos (`W( k+1 )( i + 1, j)`) polnega seštevalnika.

- število polovičnih seštevalnikov na k–ti stopnji.

```
num_half_adds := num_half_adders_rect( nrargs, width, i, k);
```

Definirate (`FOR ... LOOP`) zanko, ki teče po indeksu bitov posameznega stolpca (`j`). Tokrat položaj (`j`) pomeni absolutno lego prvega bita polovičnega seštevalnika v stolpcu stopnje redukcije. V zanki se povežejo mesta naslednje (`k+1`) stopnje redukcije za vsoto (`W( k+1 )( i, this_carry_bits + num_full_adds + j)`) in prenos (`W( k+1 )( i + 1, num_full_adds + j)`) polovičnega seštevalnika.

- celotno število bitov (`this_stage_bits`) k–te stopnje redukcije z uporabo funkcije (`this_lvl_bits_rect`).

- Število prostih bitov (`num_wires`), ki predstavljajo mesta, ki jih ne zajamejo ne polni, ne polovični seštevalniki na dani stopnji redukcije. Število prostih bitov je razlika med celotnim številom bitov v stolpcu (`this_stage_bits`) in številom bitov, ki so vezani na polne (`num_full_adds * 3`) in polovične seštevalnike (`num_half_adds * 2`).

Definirate (`FOR ... LOOP`) zanko, ki teče po indeksu bitov posameznega stolpca (`j`). Tokrat položaj (`j`) pomeni absolutno lego prostega bita v stolpcu stopnje redukcije. V zanki se povežejo mesta naslednje (`k+1`) stopnje redukcije za proste bite (`W( k+1 )( i, this_carry_bits + num_full_adds + num_half_adds + j)`).

Definirajte drugi proces z imenom (`final_stage_sum__proc`) je odvisen od tridimenzionalnega polja bitov (`W`).
Znotraj tega procesa z uporabo gnezdene zanke (`FOR ... LOOP`), ki teče po indeksu stolpcev Wallace−ovega drevesa (`i`), povežite rezultata zadnje stopnje redukcije (`W( stages - 1)( i, 0)`) in (`W( stages - 1)( i, 1)`) na vhoda argumenta komponente CLA seštevalnika (`add_a, add_b`). Dobljena signala nato z uporabo povezovalnega stavka (`PORT MAP`) povežete na (`max_sum_size`) bitno komponento CLA seštevalnika.

Za preverjanje uporabite priloženo VHDL datoteko testnih vrednosti (`wallace_addition_tb.vhd`) in s simulacijo preverite pravilnost delovanja seštevanja za opisane signale.

**Analiza procesa razhroščevanja na primeru seštevanja sheme osmih 8-bitnih števil**

**Table 1: Stage: 0 of 6**

| BIT/HEIGHT | 10/0 | 9/0 | 8/0 | 7/8 | 6/8 | 5/8 | 4/8 | 3/8 | 2/8 | 1/8 | 0/8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FA | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| HA | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| C | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 0 |
| W | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Na konec zanke, ki teče po stopnjah redukcije (`FOR k IN 0 TO stages - 2 LOOP`), vstavimo izraz za razhroščevanje, ki izpiše trenutno utež vsote (`BIT`), višino drevesa (`HEIGHT`) število porabljenih FA, HA ter bitov prenosa (`C`) in preostalih žic (`W`):

```
report "Bit#/Total " & integer'image(i) & "/" &
       integer'image(this_stage_bits) & HT &
       "FA: " & integer'image(num_full_adds) & HT &
       "HA: " & integer'image(num_half_adds) & HT &
       "C: " & integer'image(this_carry_bits) & HT &
       "W: " & integer'image(num_wires);
```

Izraz (`report`) bo v konzolnem oknu simulatorja iSim izpisoval vrednosti spremenljivk. Slika na desni prikazuje dejanske razmere na začetni (ničti) stopnji redukcije:

V naslovni vrstici zgornje tabele se nahajajo od uteži posameznega argumenta, razširjeni čez vsa mesta končne vsote (`max_sum_size`). V primeru seštevanja osmih 8-bitnih števil teče vsota od mesta 10 do 0, saj je največje število ($8*255=2040_{10}=7F8_{16}$). V stolpcu je navedeno število potrebnih HA (`HA`), FA (`FA`), prenosov (`C`) in povezav/žic (`W`) na naslednjo stopnjo redukcije. Dejansko postavitev FA in HA kaže slika na desni. Na uteži LSB (utež `0`) sta dva FA, en HA. Na uteži `1` je podobno, le da se od seštevalnikov na uteži `0` prenesejo trije prenosi. Podobno lahko razberemo za preostale uteži. Na uteži `8` so trije prenosi iz uteži `7`, na utežeh `9` in `10` pa na prvi stopnji redukcije ni nobenega FA, HA, ne prenosov, ne povezav. Višina drevesa (`h`) se na prvi stopnji reducira iz 8 na 6.

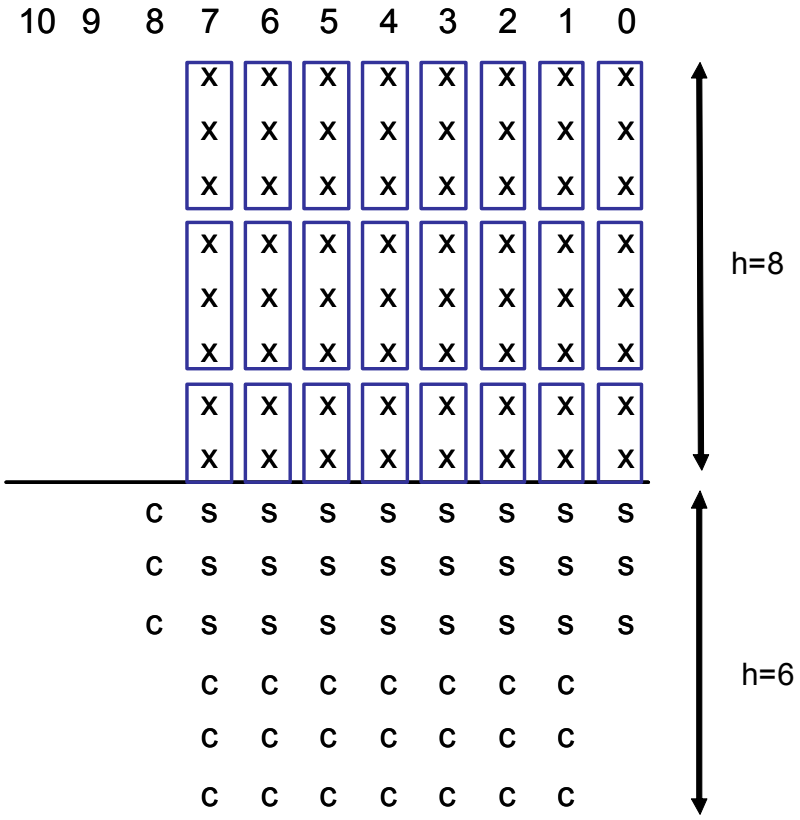
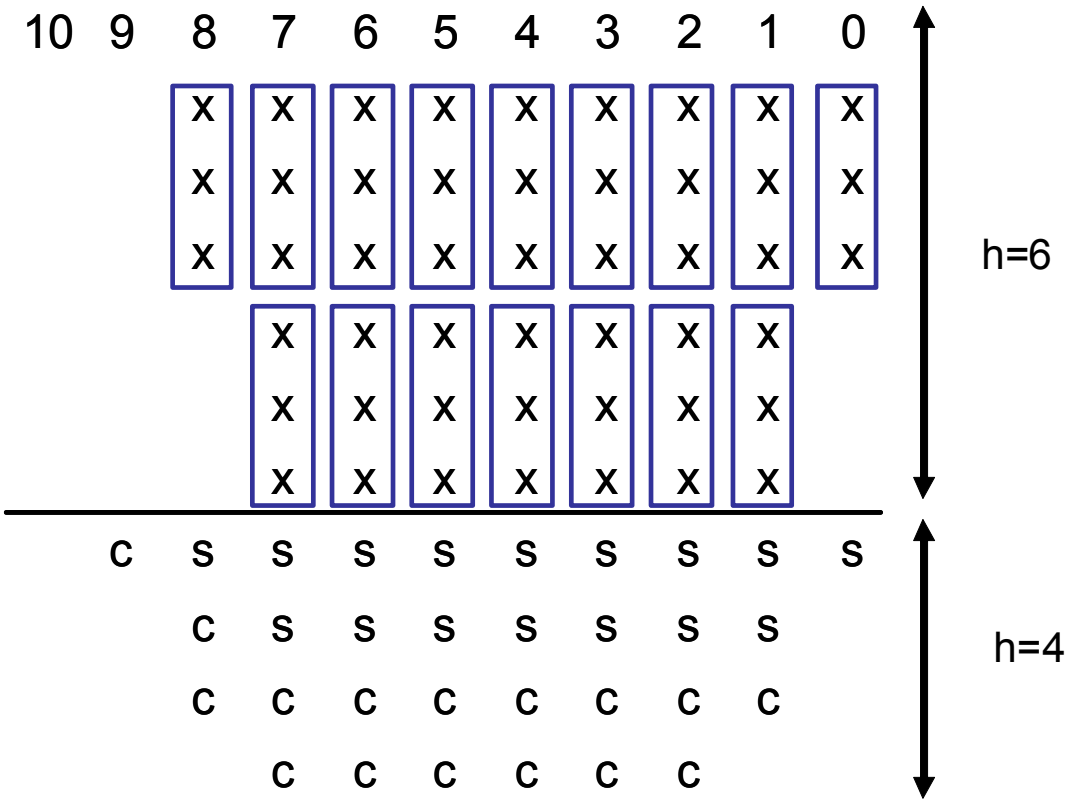


**Table 2: Stage: 1 of 6**

| BIT/HEIGHT | 10/0 | 9/0 | 8/3 | 7/6 | 6/6 | 5/6 | 4/6 | 3/6 | 2/6 | 1/6 | 0/3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FA | 0 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |
| HA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 0 |
| W | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Podobno se nadaljuje na prvi stopnji redukcije, s tem da se je višina drevesa (HEIGHT) ustrezno spremenila, glede na rezultat postavitve seštevalnikov na začetni stopnji redukcije.
Na uteži `0` se tako npr. pojavijo trije biti vsote iz začetne stopnje redukcije (zdaj so označeni z `x`, namesto z `s` kot na dnu slike na prejšnji strani).
Višina drevesa (`h`) se na prvi stopnji reducira iz 6 na 4.



Po zadnji stopnji redukcije 4 (`Stage: 4 of 6`) dobimo višino drevesa 2, kar lahko vodimo na CPA seštevalnik. Iz slike na desni strani sledi, da uteži 2 do 0 ni potrebno seštevati, kar zmanjšuje velikost končnega CPA na 8 mest. Naloga tako ni povsem optimalno rešena, saj predvideva poenostavljeno tvorbo 11-bitnega končnega CPA (`FOR i IN max_sum_size - 1 DOWNTO 0 LOOP`). To bi lahko nadgradili tako, da bi začeli zanko tvorbe argumentov končnega seštevalnika šele tam, kjer višina drevesa postane 2. Do tiste uteži, bi bite vsote vodili neposredno na rezultat seštevanja, kot je razvidno iz slike na desni strani.

**Table 3 Stage: 2 of 6**

| BIT/HEIGHT | 10/0 | 9/1 | 8/3 | 7/4 | 6/4 | 5/4 | 4/4 | 3/4 | 2/4 | 1/3 | 0/1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FA | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| HA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| W | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

**Table 4 Stage: 3 of 6**

| BIT/HEIGHT | 10/0 | 9/2 | 8/2 | 7/3 | 6/3 | 5/3 | 4/3 | 3/3 | 2/3 | 1/1 | 0/1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FA | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| HA | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| W | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

**Table 5 Stage: 4 of 6**

| BIT/HEIGHT | 10/1 | 9/2 | 8/2 | 7/2 | 6/2 | 5/2 | 4/2 | 3/2 | 2/1 | 1/1 | 0/1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| HA | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| C | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |