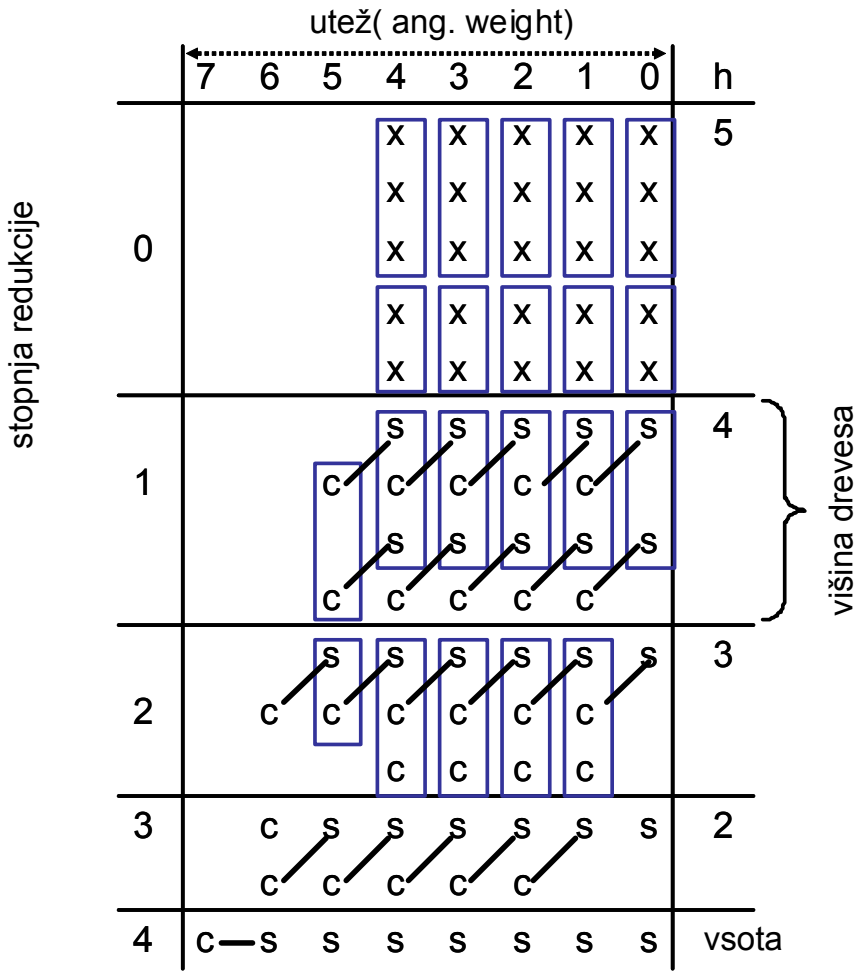
Vzporedno seštevanje s splošno Wallaceovo drevesno strukturo

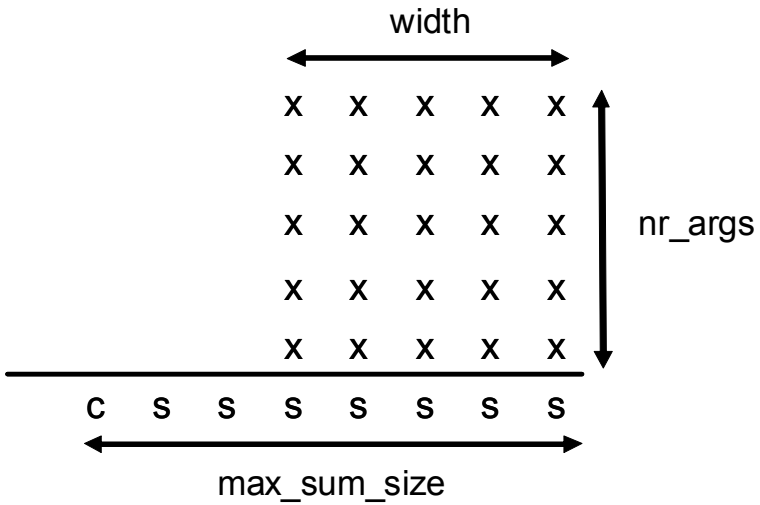


Pri Wallaceovi drevesni strukturi reduciramo začetno višino drevesa tako, da postavimo polni in polovični seštevalnik tam kjer je možno (čimprej). Postavljamo jih tako, da pri tem zaobjamemo čim večje število bitov na posamezni uteži (stolpcu) drevesne strukture. Primer vzporednega seštevanja petih nepredznačenih 5−bitnih števil prikazuje leva slika. Na začetni stopnji redukcije 0 postavimo na ustrezno utež (0 do 4) en polni in en polovični seštevalnik (obkrožena). Na stopnji redukcije 1 se zato v utežeh 0 do 4 pojavita dva bita vsote (s). Prenosa seštevanja (c) uteži 0 na stopnji 0 se pojavita na uteži 1, stopnje redukcije 1. Na sliki so prenosi, ki izvirajo iz določenega seštevanja, povezani z vsoto s črto.

Na začetni stopnji redukcije 0, se na utežeh 0 do 4 *naenkrat* (od tod ime vzporedno seštevanje) seštejeta polni in polovični seštevalnik; vsoti se vpišeta v isti stolpec, prenosa pa na naslednji stolpec. Na utežeh 0 do 4 tako dobimo zmanjšano višino h=4 na stopnji redukcije 1. Višina začetne redukcije 0 je bila h=5. Postopek redukcije višine ponavljamo po sliki na levi strani, dokler višina drevesa ni enaka 2.

Mesta zadnje iteracije (h=2) 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 vnaprejšnjim izračunom prenosa (CLA - ang. Carry Look Ahead Adder).

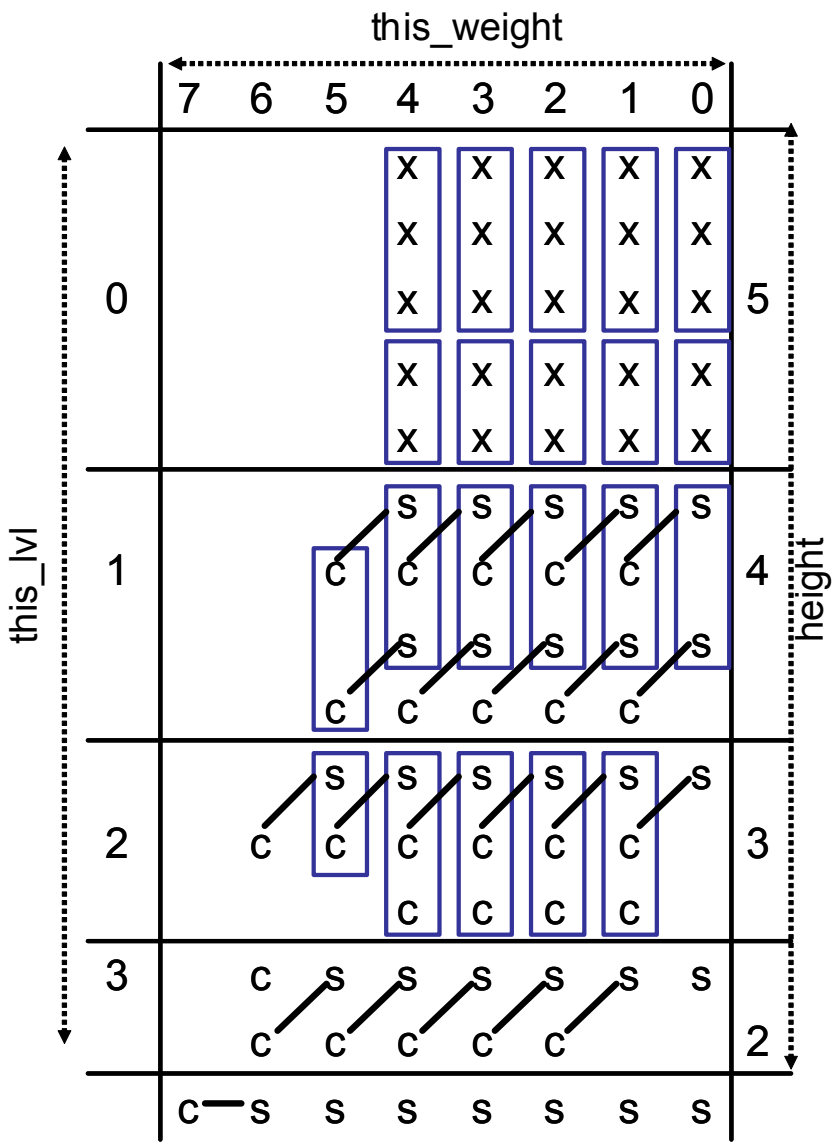
V VHDL programirajte arhitekturo splošne strukture Wallaceovega drevesnega seštevalnika, ki vzporedno sešteva polje nepredznačenih operandov enake velikosti. Število operandov je (**nrargs**), širina posameznega operanda je (**width**). Dobljena vsota je velikosti (**max_sum_size**). Izhodišče problema 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;
```



Preden bomo začeli programirati entiteto naloge, bomo pripravili nekaj funkcij, s katerimi bomo izračunali potrebne spremenljivke redukcije. Opazujmo začetno redukcijo (`this_lvl` = 0). Na dani uteži bita (`this_weight`) redukcije (`this_lvl`) moramo iz višine drevesa (`height`=5) in števila bitov argumenta seštevanja (`arg_width`=5) določiti:

1.) število FA (`full_adder_sum_bits`)

2.) število HA (`half_adder_sum_bits`)

3.) število prenosov prejšnje uteži prejšnje redukcije (`prev_lvl_carry_bits`)

4.) število bitov redukcije na dani uteži (`this_lvl_num_bits`).

Na redukciji 0, na uteži 0 imamo: 1 FA, 1 HA, 0 prenosov iz prejšnje stopnje in 5 bitov redukcije. Podobno velja za preostale uteži (od 1 do 4). Za uteži 5 do 7 je število bitov redukcije (`this_lvl_num_bits`=0), kljub temu da je (`height`=5).

Če opazujemo redukcijo 1, bomo imeli na uteži 0 *vedno samo bite vsote* redukcije 0 (prenosov iz nižje uteži ne more biti, saj je utež 0 najnižja). Na višjih utežeh (1 do 7) se lahko pojavijo prenosi na višjo utež. Za utež 0 redukcije 1 imamo samo dva bita vsote iz prejšnje redukcije, medtem imamo na utežeh 1 do 4 dva bita vsote in dva bita prenosa iz nižje uteži. Na uteži 5 ni več bitov vsote, ampak samo prenosa iz uteži 4. Na utežeh 6 in 7 na redukciji 1 sploh ni več bitov (`this_lvl_num_bits`=0). Število bitov trenutne redukcije (`this_lvl_num_bits`) dobimo tako, da od števila bitov prejšnje redukcije iste uteži odštejemo zaobjeto število bitov (3*FA) in (2*HA) - dobimo število nevezanih bitov, ki mu prištejemo število bitov vsot FA in HA ter število prenosov iz prejšnje uteži (za utež 0 redukcije 1 velja: (5-1*3-1*2)+1+1+0=2). Na redukciji 1 se namreč pojavijo tudi prosti (nevezani) biti, ki niso vezani ne na FA, ne na HA. Te bite bomo pozneje *v entiteti* vezali (z žico) na stopnjo redukcije 2. Primeri prostih bitov so biti prenosa na utežeh 1 do 4 redukcije 1. Ti biti so z žico (ang. wire) povezani na čisto spodnje bite stolpca dane uteži redukcije 2. Ostali biti prenosa na redukciji 2 izvirajo iz vsote z nižje uteži (s-c črta).

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 Wallaceovega 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:

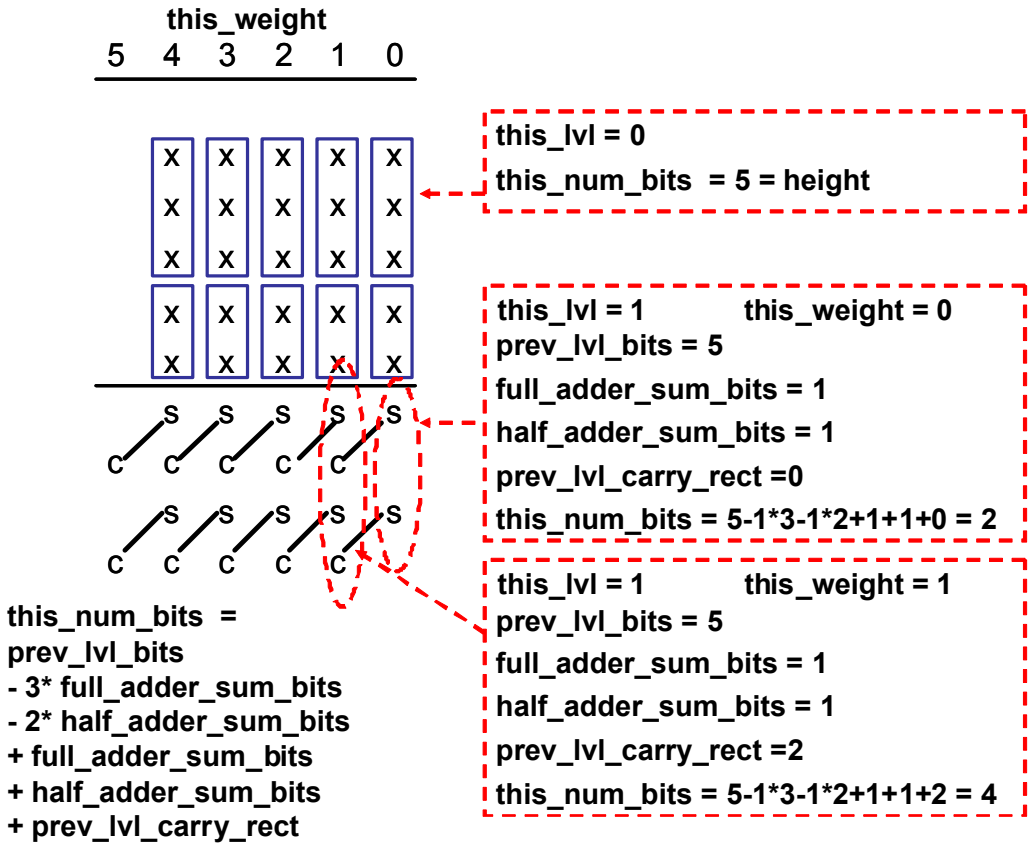
1. Funkcijo `sizeof` izdelajte tako, da definirate interno spremenljivko (`VARIABLE nr : NATURAL := a;`), ki jo znotraj (`FOR ... LOOP`) zanke iterativno delite z 2 dokler je rezultat deljenja večji od 0. Število deljenj shranite v spremenljivko, ki jo funkcija po izteku zanke (`FOR ... LOOP`) vrne.

2. Funkcija (`this_lvl_bits_rect`) vrne število bitov (`this_num_bits`) ob redukciji stolpca iz prejšnje stopnje redukcije Wallaceovega drevesa (`this_lvl - 1`) na opazovano stopnjo redukcije (`this_lvl`). Najprej izračunamo število nevezanih bitov, ki ga dobimo tako, da od števila bitov uteži na prejšnjem nivoju (`prev_lvl_bits`) odštejemo število bitov, ki jih obsega en FA (3*`full_adder_sum_bits`) in en HA (2*`half_adder_sum_bits`). Temu številu nato prištejemo število bitov vsote FA in HA ter število prenosov iz prejšnje stopnje (`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 prejšnji redukciji (`prev_lvl_bits`) dobite z uporabo rekurzivnega klica funkcije (`this_lvl_bits_rect`) na redukciji (`this_lvl - 1`). Če gre za redukcijo 0, mora biti rezultat funkcije (`this_num_bits`) enak višini drevesa (`height`) za uteži od 0 do `arg_width`-1 in 0 za uteži `arg_width` in višje.
   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.

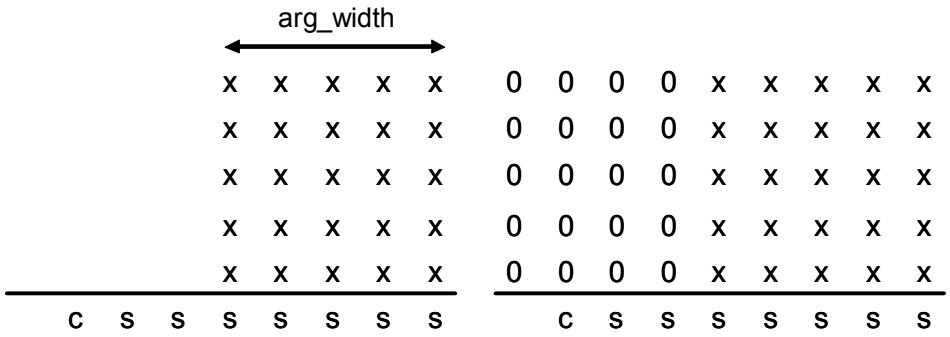
Slika 2 prikazuje primer redukcije 0 na redukcijo 1 pri seštevanju petih 5 bitnih operandov za uteži 0 in 1.



Slika 2: Prehod redukcije 0 na redukcijo 1 pri seštevanju petih 5−bitnih operandov za uteži 0 in 1.

Za uteži nad MSB operanda (`arg_width`) mora funkcija na redukciji 0 (`this_lvl=0`) vrniti (`this_num_bits=0`), saj seštevamo nepredznačena števila, kot je prikazano na sliki 3. V prikazanem primeru bi na redukciji 0 veljalo (`this_num_bits=height=5`) za uteži 0 do 4, medtem ko za uteži 5, 6, 7 velja (`this_num_bits=0`)

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 prenosov vseh seštevalnikov (`num_carry`) uteži (`this_weight`). Število bitov prenosa dobimo, če najprej določimo števila bitov prejšnje uteži (`this_weight – 1`) na prejšnje redukcije (`this_lvl – 1`). To število naj bo (`prev_lvl_bits`). Število prenosov FA (`num_carry_FA`) dobite tako, da (`prev_lvl_bits`) delite s 3. Število prenosov HA dobimo tako, da preostanek bitov uteži (`prev_lvl_bits-3*num_carry_FA`) delimo z 2. Končno število prenosov (`num_carry`) je vsota prenosov FA in HA. Na uteži 0 je število prenosov vedno 0. Na redukciji 0 (`this_lvl=0`) predpostavimo, da je število bitov (`prev_lvl_bits`) enak višini drevesa (`height`).

   Za hranjenje podatkov definirajte interne spremenljivke (`VARIABLE`) kot naravna števila (`NATURAL`):
   − (`num_carry`), ki hrani število prenosov vseh seštevalnikov,
   − (`prev_lvl_bits`), ki hrani število bitov na prejšnji stopnji redukcije. Število bitov (`prev_lvl_bits`) izračunate s klicem funkcije (`this_lvl_bits_rect`) na prejšnji redukciji (`this_lvl – 1`) prejšnje uteži (`this_weight – 1`).

4. Funkcija (`num_full_adders_rect`) vrne število polnih seštevalnikov ob na dani stopnji redukcije (`this_lvl`) uteži (`this_weight`). Število polnih seštevalnikov dobite tako, da ugotovite število bitov uteži 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 redukcije.
   Število bitov stolpca (`this_num_bits`) izračunate s klicem funkcije (`this_lvl_bits_rect`) na redukciji (`this_lvl`).

5. Funkcija (`num_half_adders_rect`) vrne število HA na redukciji (`this_lvl`) uteži (`this_weight`). Število HA dobite tako, da ugotovite število bitov (`this_num_bits`) uteži dane redukcije (`this_lvl`) s klicem funkcije (`this_lvl_bits_rect`). S klicem funkcije (`num_full_adders_rect`) izračunate število FA (`num_full_adds`). Razliko števila bitov (`this_num_bits`) in števila bitov, ki jih seštevajo FA (`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 Wallaceovega 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 konstanto polja celoštevilskih vrednosti (**nr_stages_type**), v katerem definirate število redukcij (n). Vrednosti povzema spodnja tabela:

h	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
n	8	8	8	8	7	7	7	7	7	7	7	7	7	6	6	6	6	6	6	5	5	5	5	4	4	4	3	3	2	1

Drevo **h** operandov reduciramo v **n** redukcijah na dva operanda, ki ju seštejemo s končnim CPA seštevalnikom po spodnji rekurzivni relaciji. Za dva operanda (h=2) ni redukcije (imamo samo CPA seštevalnik, ki ga ne štejemo). Z eno redukcijo (n=1) reduciramo največ h=3 števila na vhodu (uporabimo FA, ki reducira drevo na h=2).

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

Formulo lahko tudi obrnemo, tako da izrazimo potrebno število redukcij **n**, da seštejemo **h** operandov.

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

Z n=9 koraki redukcije je možno seštevati največ h=63 operandov. Spodnja tabela povzema število operandov (h) v odvisnosti od števila redukcij (n)

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
h	2	3	4	6	9	13	19	28	42	63	94	141	211	316	474	711	1066	1599	2398	3597

V našem primeru se bomo omejili na seštevanje največ 32 operandov.

Izdelajte konstanto (**nr_stages**) tipa (**nr_stages_type**). Ena možnost opisovanja tovrstnega podatkovnega tipa je rekurzivna funkcija. Namesto nje bomo raje uporabili 1D polje iz tabele na prejšnji strani. Za indeks tipa definirajte 1D polje (**ARRAY**) 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 Wallaceovega drevesa:

```
CONSTANT stages: INTEGER := nr_stages( nrargs );  --število stopenj redukcije Wallaceovega 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 2D polje bitov vseh operandov posamezne redukcije. Dimenzije polja so (**max_sum_size, nrargs**).
Definirajte tip (**W_type**), ki predstavlja 1D polje polja (**cell_type**). Dimenzija 1D polja je (**stages+1**). Število redukcij je povečano za izhodno stopnjo, ko je višina drevesa h=2. Tako smo definirali tridimenzionalno polje polja v VHDL, katerega prva dimenzija opisuje redukcije, preostali dimenziji pa definirata položaj bita operanda (vrstica, stolpec). Posamezne celice drevesa definirajte kot signal (**W**):

```
SIGNAL W : W_type := (others => (others =>(others =>'0'))); -- Wallaceovo 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 Wallaceovega 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 Wallaceovem drevesu, drugi bo povezoval izhod zadnje redukcije Wallaceovega 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 redukcije 0 (**W**). Do mesta na koordinatah (**i, j**) redukcije 0 dostopate tako: `W(0)( i, j) .` Povezovanje mest izvedite v dvojni gnezdeni (**FOR ... LOOP**) zanki po koordinatah (**i, j**).

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 FA na k−ti stopnji.

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

Definirate (`FOR ... LOOP`) zanko, ki teče po indeksu (`j`) od `0` do (`num_full_adds-1`). Položaj (`3*j`) pomeni absolutno lego prvega bita FA. V iteraciji zanke se povežejo trije biti (`3*j, 3*j+1, 3*j+2`) na bite naslednje redukcije (`k+1`) za:

- vsoto (`W( k+1 )( i, this_carry_bits + j)`) in
- prenos (`W( k+1 )( i + 1, j)`) FA.

- število HA na k−ti stopnji.

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

Definirate (`FOR ... LOOP`) zanko, ki teče po indeksu (`j`) od `0` do (`num_half_adds-1`). Položaj (`2*j`) pomeni absolutno lego prvega bita HA. V iteraciji zanke se povežeta bita (`2*j, 2*j+1`) na bite naslednje redukcije (`k+1`) za :

- vsoto (`W( k+1 )( i, this_carry_bits + num_full_adds + j)`) in
- prenos (`W( k+1 )( i + 1, num_full_adds + j)`) HA.

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

- Število prostih bitov (`num_wires`), ki predstavljajo povezave (ang. wire) na naslednjo redukcijo. Število teh povezav/žic (oz. nevezanih, prostih bitov) je razlika med celotnim številom bitov v stolpcu (`this_stage_bits`) in številom bitov, ki jih zaobjamejo FA (`num_full_adds * 3`) in HA (`num_half_adds * 2`).
  Definirate (`FOR ... LOOP`) zanko, ki teče po indeksu (`j`) od `0` do (`num_wires-1`).. Tokrat položaj (`j`) pomeni absolutno lego prostega bita v stolpcu stopnje redukcije. V zanki se povežejo mesta naslednje redukcije (`k+1`) za proste bite:

  ```
  W( k+1 )( i, this_carry_bits + num_full_adds + num_half_adds + j) <=
  W(k)(i, num_full_adds * 3 + num_half_adds * 2 + 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 Wallaceovega drevesa (`i`), povežite rezultata izhoda zadnje redukcije (`W( stages )( i, 0)`) in (`W( stages )( 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 4

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 redukcijah (`FOR k IN 0 TO stages - 1 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 (h=8) prikazuje razmere na redukciji 0:
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.

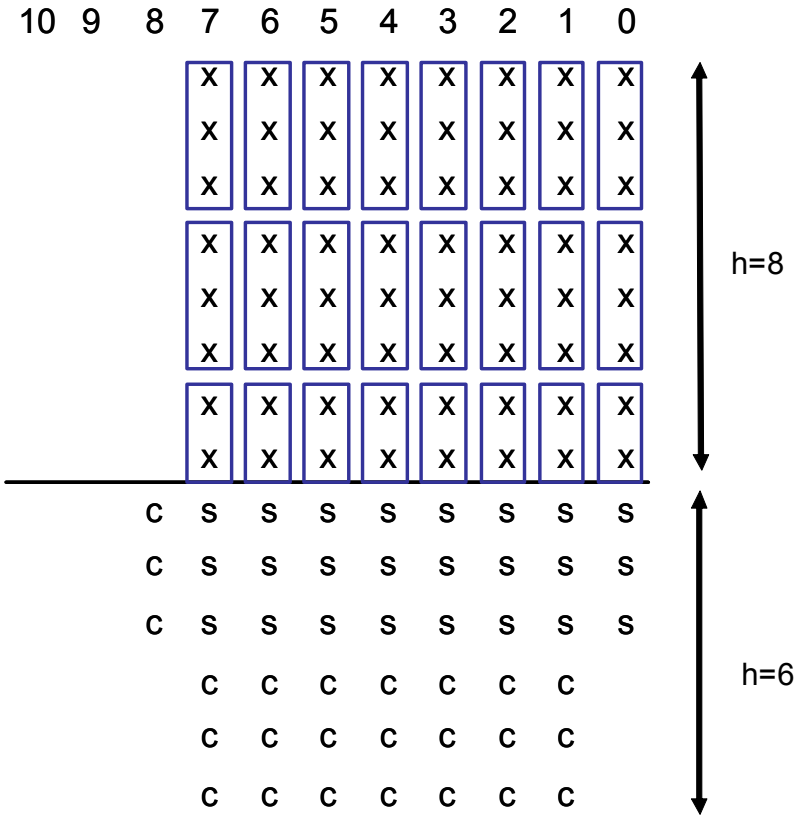
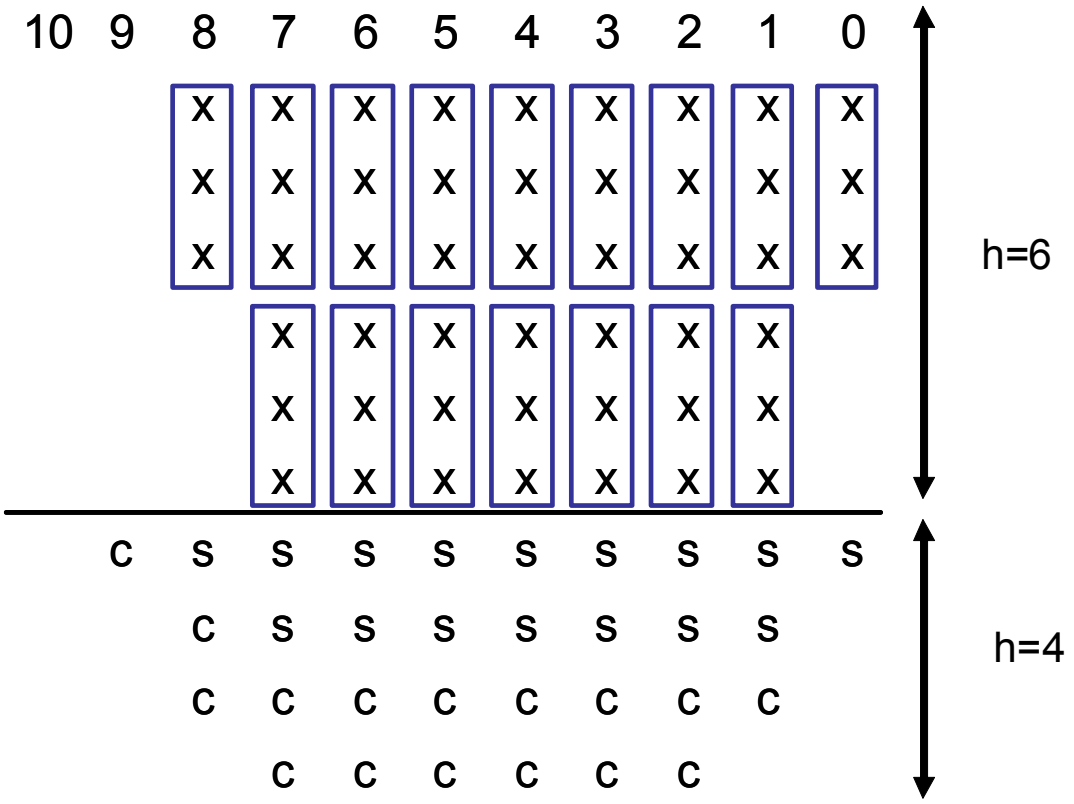


Table 2: Stage: 1 of 4

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 redukciji, s tem da se je višina drevesa (HEIGHT) zmanjšala iz h=8 na h=6 na utežeh 1 do 7. Na utežeh 0 in 8 je h=3.
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 *po* prvi redukciji zmanjša iz h=6 na h=4.



Po zadnji redukciji (`Stage: 3 of 4`) 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 4

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 4

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 Stanje po zadnji redukciji

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