1 Exercises: Fixed Point Numbers

1.1 Integral Numbers

There are two ways to interpret a bit vector as integral number: *unsigned* and *signed*, corresponding to the *IEEE* VHDL libraries *std_logic_unsigned* and *std_logic_signed*, resp.

Unsigned interpretation: A bit vector of w bits represents the integer range . . . . . .

Signed interpretation: A bit vector of w bits represents the int. range . . . . . .

1.2 Fixed Point Numerical Representation: The Q Number Format

Unsigned: UQg.f with *g* integral (deutsch: ganze) and *f* fractional bits. Width *w=g+f*.
Signed: Qg.f with 1 sign bit plus *g* integral and *f* fraction bits. Width *w=1+g+f*.

Exercise: The bit string **110.1011** can be interpreted...

... as UQ3.4 format representing

. . . . . .

... as Q2.4 delivering

. . . . . .

Unsigned: Range: Resolution: r =

Singed Range: Resolution: r

- You can append an arbitrary number of . . . . . after the point.
- You can precede an arbitrary number of . . . . . before an unsigned number.
- You can precede an arbitrary multiple of . . . . . before a signed number.

Summation an subtraction of fixed-point numbers is easy as they can be treated like integer numbers when they are written such that the points are over each other. Example:

Given numbers	Unsigned treatment	Signed treatment
11011011.11011...	11011011.11011...	11011011.11011...
± 101.11101101	± 101.11101101	± 101.11101101

1.3 Multiplication of Fixed-Point Numbers

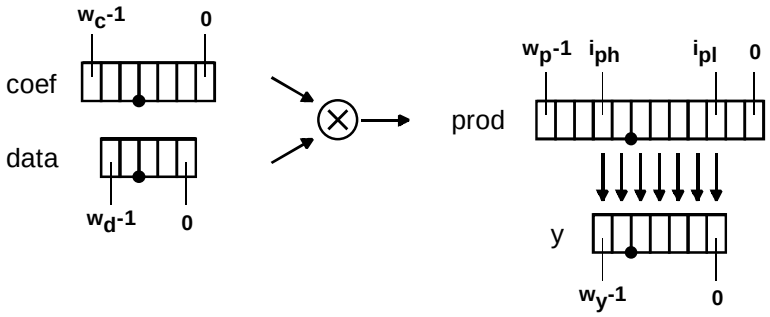

Fig. 2.3:
Reducing the product length to the length of its factors with indices $\mathbf{i_{ph}}$ and $\mathbf{i_{pl}}$.

We compute **prod = coef** * **data** with **coef** and **data** having $\mathbf{w_c}$ and $\mathbf{w_d}$ binary places, respectively, $\mathbf{f_c}$ and $\mathbf{f_d}$ of them fractional.

Then **prod** has . . . . . binary places, ... of them fractional.

Reducing the length of products: $\mathbf{y = prod(i_{ph} : i_{pl})}$

Let **coef** have $\mathbf{w_c}$ binary places, $\mathbf{f_c}$ of them fractional. Signal **data** has $\mathbf{w_d}$ binary places, $\mathbf{f_d}$ of them fractional. The product has

$\mathbf{w_p}$ = ... binary places, $\mathbf{f_p}$ = ... of them fractional.

Fig. 2.3 illustrates the multiplication of the coefficient **coef** with $\mathbf{w_c}$ = ... , $\mathbf{f_c}$ = ...

and the data sample **data** with $\mathbf{w_d}$ = ... , $\mathbf{f_d}$ = ... The product **prod** has

$\mathbf{w_p}$ = ... binary places,

$\mathbf{f_p}$ = ... of them fractional.

We want to take result vector **y** out of **prod** preserving the point. For all bit vectors the LSB has index 0.

In Fig. 2.3 **y** has $\mathbf{w_y}$ = ... , binary places $\mathbf{f_y}$ = ... of them fractional.

To apply the VHDL command **`y<=prod(iph DOWNTO ipl)`** we have to compute

$\mathbf{i_{pl}}$ = ...

$\mathbf{i_{ph}}$ = ...

1.4 Real → Binary Conversion

Exercises (for solutions see → chapter 8) **:**
Convert π=3.14159 into a signed bit vector with 8 binary places, 4 of them fractional.

. . . . . .

. . . . . .

Convert -π=-3.14159 into a signed bit vector with 8 binary places, 4 of them fractional.

. . . . . .

. . . . . .

2 Exercise Based on Executable VHDL

Listing 4: Code with gaps

```
(1)  LIBRARY ieee; USE ieee.std_logic_1164.ALL;
(2)  PACKAGE pk_filter IS
(3)    CONSTANT cDataInWidth:POSITIVE:=4;    -- Input-Data BitWidth
(4)    CONSTANT cDataInFract:POSITIVE:=2;    -- No of Input-Data fract. Bits
(5)    CONSTANT cDataOutWidth:POSITIVE:=5; -- Output-Data BitWidth
(6)    CONSTANT cDataOutFract:POSITIVE:=3; -- No of Output-Data fract Bits
(7)    CONSTANT cCoefWidth:POSITIVE:=4;     -- Coefficient's BitWidth
(8)    CONSTANT cCoefFract:POSITIVE:=2;     -- No of Coef's fractional Bits
(9)    SUBTYPE  t_DataIn  IS std_logic_vector(cDataInWidth-1  DOWNTO 0);
(10)   SUBTYPE  t_DataOut IS std_logic_vector(cDataOutWidth-1 DOWNTO 0);
(11)   SUBTYPE  t_coef IS std_logic_vector(cCoefWidth-1 DOWNTO 0);
(12) END PACKAGE pk_filter;
(13)
(14) LIBRARY ieee; USE ieee.std_logic_1164.ALL,
(15)                   ieee.std_logic_signed."+", ieee.std_logic_signed."*";
(16) USE WORK.pk_filter.ALL;
(17) ENTITY TestBitslice IS
(18) END ENTITY TestBitslice;
(19)
(20) ARCHITECTURE rtl_TestBitslice OF TestBitslice IS
(21)   SIGNAL DataIn :t_DataIn;
(22)   SIGNAL coef   :t_coef;
(23)   SIGNAL DataOut:t_DataOut;

(24)   SIGNAL product:std_logic_vector(.....................................
     ..........................................................................

(25)   CONSTANT iPl:NATURAL:= ..........................................
     ..........................................................................

(26)   CONSTANT iPh:NATURAL:= ..........................................
     ..........................................................................
(27) BEGIN
(28)   DataIn  <= "0101", "0100" AFTER 10 ns; -- 1.25,   1.00 AFTER 10 ns
(29)   coef    <= "0101";                     -- 1.25
(30)   product <= coef * DataIn;              -- 1.5625, 1.25 AFTER 10 ns

(31)   DataOut <= product(iPh DOWNTO iPl) ..............................


     ..........................................................................
(32) END ARCHITECTURE rtl_TestBitslice;
```

Correspondences with chapter 2.3: f_c=cCoefFract, f_d=cDataInFract, f_y=cDataOutFract, w_c, w_d, w_p, w_y: cCoefWidth, cDataInWidth, cProdWidht, cDataOutWidth, respectively.

Exercises:

- Complete line (24) to get a *product* signal that fits to the multiplication of line (30).
- Compute *iPl* und *iPh* in lines (25), (26) to fit the bit-slice operation of line (31).
- Extend line (31) to get the bit-slice by bit-vector easy rounding.
- Verify the product, bit-slice and rounding operation of lines (39), (49) by hand.