Arithmetic, Relational, and Logical Functions 9

This chapter discusses the XIL library's arithmetic, relational, and logical functions. These functions can be used to lighten or darken an image, to increase or decrease contrast in an image, or to produce the "negative" of an image.

Note – All the functions discussed in this chapter can be performed in place; that is, the source and destination images can be the same image.

Arithmetic Functions

If you have two source images and a destination image, the XIL library enables you to:

- Add the two source images and store the results in the destination (`xil_add()`)
- Subtract one source image from the other and store the results in the destination (`xil_subtract()`)
- Multiply the two source images and store the results in the destination (`xil_multiply()`)
- Divide one source image into the other and store the results in the destination (`xil_divide()`)

For a single source image, the library lets you find the absolute value of pixels in an image and store the result in a destination image (`xil_absolute()`).

For each of these operations to work, its source and destination images must have the same data type: `XIL_BIT`, `XIL_BYTE`, or `XIL_SHORT`. Also, the images must have the same number of bands. The images do *not* have to have the same width and height. For an explanation of which pixels an operation will affect if the images have different sizes, see "Region of Interest" on page 48.

When you add two images, you take the value at location 0,0 in one source image, add it to the value at location 0,0 in a second source image, and write the sum at location 0,0 in a destination image. You then follow the same procedure for all other points in the images. Subtraction, multiplication, and division are handled similarly. When multiband images are involved, the arithmetic operation is performed on corresponding bands in the source images; that is, band 0 in the first source image is added to band 0 in the second source image, and so on.

This all seems very straightforward. But there are a few points you should bear in mind when using these functions:

- **If the result of an operation is out of range for a particular data type, the result is not truncated, but is clamped to the minimum or maximum value for the data type.** Thus, if you're working with `XIL_BYTE` images, adding a 200 and a 200 gives a result of 255 because 400 cannot be represented in 8 bits and the greatest valid value for an `XIL_BYTE` image is 255. Likewise, if you subtract 200 from 100, the result is 0. Table 9-1 indicates the valid range of values for each XIL data type.

  *Table 9-1*Valid Values for Each XIL Data Type

Data Type	Lowest Value	Greatest Value
`XIL_BIT`	0	1
`XIL_BYTE`	0	255
`XIL_SHORT`	-32768	32767

- **Division by 0 is permitted.** If the image serving as the divisor contains 0's, you will receive *one* error message indicating that division by 0 occurred. However, the division operation will be performed. If a 0 in an image is divided by 0, the result is 0, and if any other value is divided by 0, the result is the greatest valid value for the data type (or possibly the lowest valid value if the numerator is a negative value of type `XIL_SHORT`).

- **All division is performed using floating-point operands.** The `xil_divide()` function casts each dividend and divisor to a `float` before doing the division. It then rounds off each quotient and casts it to the type of the images involved in the division.

The `xil_absolute()` function finds the absolute value of pixels in a source image and stores the result in a destination image. Since `XIL_BIT` and `XIL_BYTE` images don't have negative values, `xil_absolute()` is useful only for `XIL_SHORT` images. For each band in the image, the absolute value at location 0,0 in the source is written to location 0,0 in the destination; the same procedure is then followed for all other points in the image.

Note – On `XIL_BIT` and `XIL_BYTE` images, `xil_absolute` is effectively `xil_copy`.

Relational Functions

If you have two source images and a destination image, the XIL library enables you to:

- Find the larger of pixels in the two source images and store the results in the destination (`xil_max()`)
- Find the lesser of pixels in the two source images and store the results in the destination (`xil_min()`)

As with the arithmetic operations, `xil_max()` and `xil_min()` require the source and destination images to have the same data type and the same number of bands. The images do not have to have the same width and height.

Taking the maximum or minimum values from two images performs a band by band comparison. For example, if `src1` and `src2` images are 4-banded images, comparing the value at location 0,0 writes to location 0,0 in the destination image the maximum (or minimum) for each source band; thus, the first band at 0,0 in the destination might receive its value from `src1`, whereas the remaining three bands might receive their values from `src2`. This band by band comparison is repeated for all other points in the source images.

Note – Don't confuse these relational functions with finding the maximum or minimum pixel value within each band of a single image. To find those values, use `xil_extrema()`, discussed in "Finding the Minimum and Maximum Values in an Image" on page 178.

Logical Functions

If you have two source images and a destination image, the XIL library also enables you to:

- Take the bitwise AND of the two source images and store the results in the destination (`xil_and()`)
- Take the bitwise OR of the two source images and store the results in the destination (`xil_or()`)
- Take the bitwise XOR of the two source images and store the results in the destination (`xil_xor()`)

As with the arithmetic and relational operations, for these operations to work, all three images must have the same data type and the same number of bands. The images do not have to have the same width and height.

The XIL library also contains a bitwise NOT operator (`xil_not`). This function works on a single source image. It looks at the values in that image as binary values and changes all the 1's in those values to 0's, and all the 0's to 1's. The function then writes this one's complement version of the source image to the destination.

Operations with Constants

For each of the arithmetic and logical functions that operate on two source images—addition and so on—the XIL library includes a similar function that takes as input one source image and a constant. For instance, the library contains the function `xil_add()`, whose prototype is shown below.

```
void xil_add (XilImage src1, XilImage src2, XilImage dst);
```

It also contains the function `xil_add_const()`, whose prototype is shown below.

```
void xil_add_const (XilImage src1, float *constants,
    XilImage dst);
```

To add a constant to a single-band image, you declare a one-element array of type `float`, assign that element a value, and then call `xil_add_const()`. For instance, the code to add 8.0 to each value in a single-band image might look like this:

```
XilImage src1, dst;
float constants[1] = {8.0};

xil_add_const(src1, constants, dst);
```

This operation is roughly the equivalent of adding to `src1` a second source image, all of whose values are 8. The only difference between these two operations is that using a constant instead of a second source image enables you to use noninteger values in the operation. When you add a constant to an image, each sum is rounded and then cast to the data type of the source and destination images.

Note – As stated on page 142, if the result of an operation is out of range, the value is clamped and assigned the minimum or maximum value for the data type. It's important to realize that the operation is performed first, then the result is clamped. For example, if the operation is division and you divide a negative constant into the values for an `XIL_BYTE` image, the division is first performed. Thus, for a pixel value of 10 divided by the constant -2, the result is

10 / -2 = -5, which is clamped to 0.

Notice that clamping the -2 to 0 before the division is performed would yield 10 / 0, which would result in a value of 255 in the destination image.

In operations that involve a multiband source image and a constant, the constant must actually be an array of constants, and the number of constants in the array must equal the number of bands in the image. The following code adds a different constant to each band of a 3-band image.

```
XilImage src1, dst;
float constants[3] = {8.0, 12.0, 2.0};

xil_add_const(src1, constants, dst);
```

This operation adds 8.0 to each value in band 0 of the image, 12.0 to each value in band 1, and 2.0 to each value in band 2.

For operations that are not associative—subtraction and division—the XIL library enables you to specify the constant as the first operand or the second. That is, for a subtraction, the constant can be either the minuend or the subtrahend, and for a division, it can be either the dividend or the divisor.

Table 9-2 lists the arithmetic functions that can take constants.

Table 9-2 Arithmetic Operations Using a Source Image and a Constant

Function Name	What the Function Does
`xil_add_const`	Adds an image and a constant
`xil_subtract_const`	Subtracts a constant from an image
`xil_subtract_from_const`	Subtracts an image from a constant
`xil_multiply_const`	Multiplies an image by a constant
`xil_divide_by_const`	Divides an image by a constant
`xil_divide_into_const`	Divides an image into a constant

You can also perform logical operations using an image and a constant. For instance, besides the function `xil_and()`, the library contains the function `xil_and_const()`, whose prototype is shown below.

```
void xil_and_const (XilImage src1, unsigned int *constants,
    XilImage dst);
```

To take the logical AND of a constant and a single-band image, you declare a one-element array of type `unsigned int`, assign that element a value, and then call `xil_and_const()`. For instance, the code to find the logical AND of each value in an image and 8 might look like this:

```
XilImage src1, dst;
unsigned int constants[1] = {8};

xil_and_const(src1, constants, dst);
```

This operation is the equivalent of taking the logical AND of `src1` and a second source image, all of whose values are 8.

Table 9-3 lists the logical functions that can take constants.

Table 9-3 Logical Operations Using a Source Image and a Constant

Function Name	What the Function Does
`xil_and_const`	ANDs an image and a constant
`xil_or_const`	ORs an image and a constant
`xil_xor_const`	XORs an image and a constant

Note – The relational operations `xil_max()` and `xil_min()` always require two source images and cannot operate on one source image and one constant.

Arithmetic and Logical Operations with Bit Images

The arithmetic and logical operations discussed in this chapter *are* defined for `XIL_BIT` images. If you bear in mind that the values in the destination image produced by such an operation cannot fall below 0 or go above 1, the results to expect are usually obvious. For example, when you add two `XIL_BIT` images, the following calculations are used:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 1

Note that 1 plus 1 equals 1. Two is not a valid value for a 1-bit data element, so the result has been clamped at 1. Similarly, when you subtract one `XIL_BIT` image from another, 0 minus 1 equals 0 because -1 is not a valid value.

When performing division, bear in mind that 0 divided by 0 equals 0 and that a positive nonzero value divided by 0 equals the maximum valid value for the data type, in this case 1.

When you perform an arithmetic or logical operation that involves a bit image and a constant, similar rules apply. For arithmetic operations, think of the following sequence of steps taking place:

1. Addition, subtraction, multiplication, or division is performed using a 1-bit value (0 or 1) and a floating-point value.
2. The result is rounded and clamped to 0 or 1.
3. The 0 or 1 is cast to the data type `XIL_BIT`.

For logical operations involving a constant, the constant (an `unsigned int`) is set to 1 if it is greater than 0; then the operation is performed.