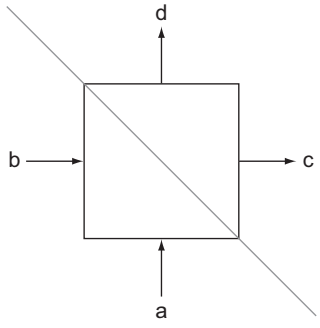
5 Regular 4-Sided Composition

This tutorial shows how regular circuits with 4-sided elements can be described in Lava. The type of regular circuits that are discussed in this tutorial are those where data flows in two or more directions, including directions perpendicular to each other. Two classes of circuits that have these data flow characterics are systolic arrays and arithmetic operators. We will demonstrate the features of Lava for composing regular circuits with 4-sided elements using the example of a constant adder/subtractor design. Before describing this design, we first introduce the model of 4-sided elements used by the standard Lava combinators. Because 4-sided elements are connected together in a 2-dimensional array, they are referred to here as 4-sided tiles.

5.1 4-Sided Tiles

Lava includes a set of combinators that interpret the inputs and outputs of circuit blocks according to a two-dimensional model. The relationship between this two-dimensional model and the input and output types of the circuit is shown in Figure 5.1. A circuit is considered a four sided tile if it has a type which takes a two element tuple as its input and returns a two element tuple.

Figure 5.1 Mapping of inputs and outputs to the sides of a 4-sided circuit element. The type of the 4-sided tile is `(a,b) -> (c,d)`.



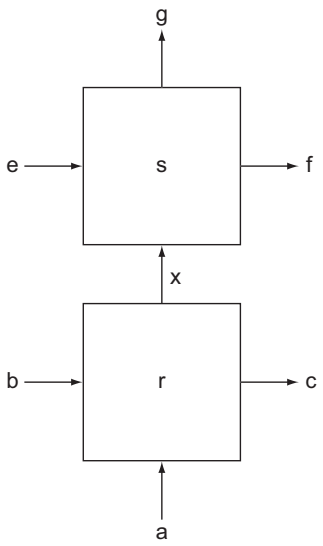
A four sided tile is divided into the input signal and output signal portions by the gray diagonal line. The inputs are represented by a tuple (a, b) in which the a signal is connected to the bottom of the tile and the b signal is connected to the left of the tile. These signals can be of any type and so can consist of more than one wire. The outputs are represented by a tuple (c, d) in which the c signal is connected to the right hand side of the tile and the d signal is connected to the top of the tile. This model is not applicable to all

circuits with 4-sided tiles, but Lava is not limited to this model. A user can create any model they wish to suit their applications by creating their own combinators.

To combine and place two 4-sided tiles vertically, Lava provides the `below` combinator. Figure 5.2 shows the result of the composition:

```
r `below` s
```

Figure 5.2 Connecting two 4-sided tiles together with the below operator.



The circuit r has the type (a,b) -> (c, x) and the circuit s has the type (x,e) -> (f, g) and the composite circuit has the type (a, (b, e)) -> ((c, f), g) where x is the type of the intermediate signal that r and s communicate along. To combine two 4-sided tiles horizontally, Lava provides the `beside` combinator. The composition

```
r `beside` s
```

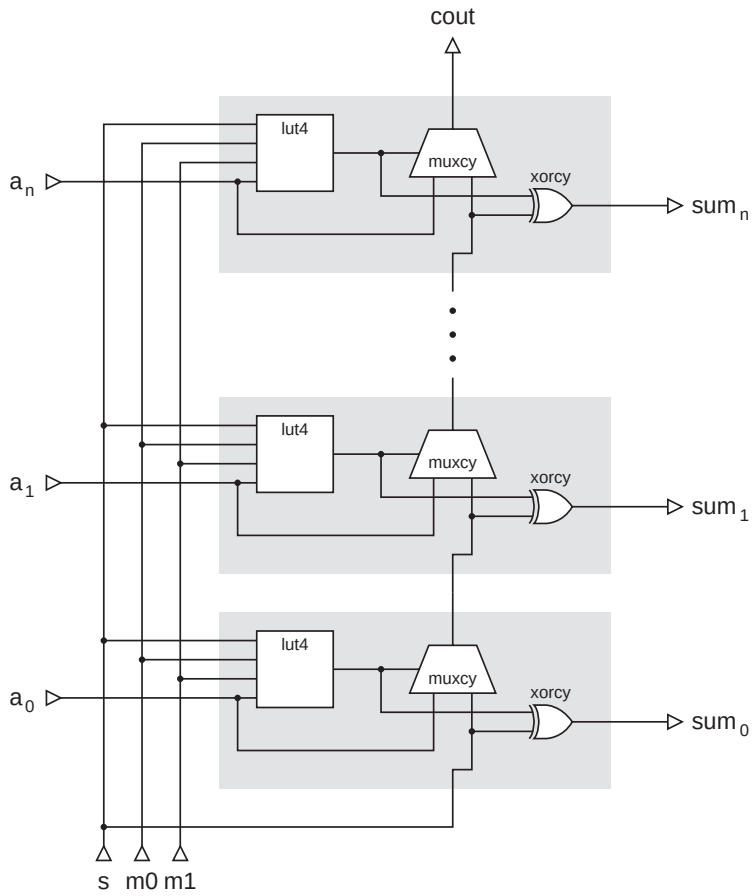
places s to the right of r. Lava provides parametrized combinators to replicate and compose four sided tiles in row and columns. The `row` combinator replicates and composes a tile horizontally from left to right and takes a parameter that determines number of times a tile is to be replicated. The `col` combinator replicates and composes a tile vertically from bottom to top. The resulting type of applying the `col` combinator to the tile in Figure 5.1 is `(a, [b]) -> ([c], d)` and the resulting type of applying the `row` combinator is `([a], b) -> (c, [d])`. The `row` and `col` combinators can be used together to form a 2-dimensional array of 4-sided tiles. The use of these operators is demonstrated in the next section by the constant adder/subtractor design.

5.2 Constant Adder/Subtractor Design

This section demonstrates the use of 4-sided tile combinators by providing the steps required to describe a constant adder/subtractor design in Lava. Figure 5.3 shows a constant adder/subtractor implementation that uses circuit blocks specific to the Xilinx Virtex architecture. The repeating element is a 4-sided tile that implements a 1-bit constant adder/subtractor. This adds or subtract one of four constant bits to the input bit a. This is combined using a ripple-carry architecture to implement an n-bit adder/subtractor. The input s selects between the add mode when low, and the subtract mode when high. The constants are selected by the signals m0 and m1. The main steps required to describe this circuit are to firstly describe the repeating element, and then secondly compose them whilst distributing the inputs and outputs accordingly.

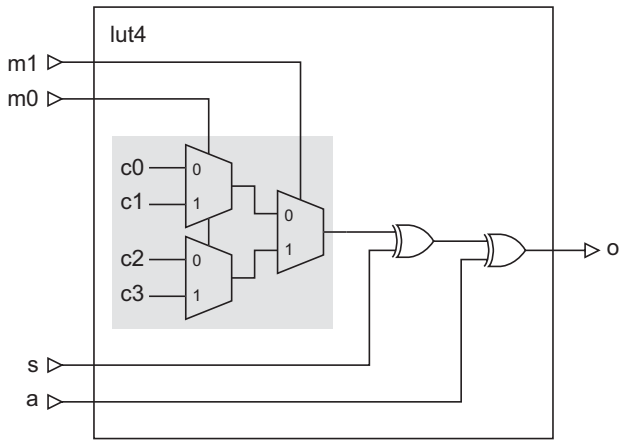
Figure 5.3

Constant adder/subtractor design. The repeating element is a 4-sided tile that is composed in a column. The Virtex dedicated carry chain components are used to optimise the performance and resource usage of the design. The input s selects between the add mode when low, and the subtract mode when high. Four constants are stored in the LUTs; they are selected using the signals m0 and m1.



The muxcy and xorcy circuit blocks are dedicated resources provided in the Virtex slice. They are connected in the configuration shown in Figure 5.3 and provide an efficient way to build ripple-carry arithmetic circuits. To construct a full adder in a single slice, the LUT is configured as an XOR gate. We wish to provide some additional functionality within the LUT; a subtract mode and 4 constants. Figure 5.4 shows a way in which this can be achieved.

Figure 5.4 Constant adder/subtractor LUT. The constants c0-c3 are encoded in the table by partially applying them to the LUT initialisation function.



The final XOR provides the ‘add’ functionality, the preceding XOR inverts the constant operand when in subtract mode and the multiplexors select between the four constants. The contents of the 4-input LUT can be provided, as before, using an initialising function. The constants can be then be encoded by partial application. The gray area denotes the part of the circuit that is transformed to a constant 2-input function by partial application. The initialising function is written in Lava as follows:

```
addSubConst4Func c0 c1 c2 c3 a s m0 m1 =
  (((muxFn m1 (muxFn m0 c0 c1) (muxFn m0 c2 c3)) == s) == a)
```

It is used to initialise the LUT, partial applying the constants c0, c1, c2 and c3, as follows:

```
lut4 (addSubConst4Func c0 c1 c2 c3)
```

The type of the constants c0, c1, c2 and c3 is `Bool`.

The repeating tile fits within one half of a Virtex slice and so can composed using named wires as follows:

```
oneBitAddSubConst4 :: Bit -> Bit -> Bit ->
                      (Bit, (Bit, Bool, Bool, Bool, Bool)) ->
                      (Bit, Bit)

oneBitAddSubConst4 s m0 m1 (cin, (a, c0, c1, c2, c3))
  = (sum, cout)
    where
    part_sum = lut4 (addSubConst4Func c0 c1 c2 c3) (a, s, m0, m1)
    sum = xorcy (part_sum, cin)
    cout = muxcy (part_sum, (a, cin))
```

Since the inputs c, m0 and m1 fan-out across each of the tiles then they are not included in the 4-sided composition. They therefore are partially applied to give a 2-element tuple, the correct type for 4-sided composition. The carry input `cin` is the lower input, and the operand input `a` and the constants `c0`-`c3` are the left inputs. To compose the tiles into a column of size n we use the `col` combinator. Since the type of the left input is `(Bit, Bit, Bit, Bit, Bit)` the `col` combinator results in the type `(Bit, [(Bit, Bit, Bit, Bit, Bit)]) -> ([Bit], Bit)`.

It is convient to represent each of the left multi-bit inputs as lists. In this case, we need a function to give the type `[(Bit, Bit, Bit, Bit, Bit)]`. As for the example in last tutorial we can use a zip function. We need a five element zip function and since it is not in the standard library, we need to define it ourselves:

```
zip5 :: [a] -> [b] -> [c] -> [d] -> [e] -> [(a, b, c, d, e)]
zip5 [] [] [] [] [] = []
zip5 (a:as) (b:bs) (c:cs) (d:ds) (e:es) =
  (a, b, c, d, e) : zip5 as bs cs ds es
```

Now, the composition of the column can be completed as follows:

```
addSubConst4 :: Int -> Bit -> Bit ->
                [Bool] -> [Bool] -> [Bool] -> [Bool] ->
                (Bit, [Bit]) -> ([Bit], Bit)

addSubConst4 n m0 m1 c0 c1 c2 c3 (s, a)
  = col n (oneBitAddSubConst4 s m0 m1) (s, (zip5 a c0 c1 c2 c3))
```

5.3 Exercises

1. Describe the circuit `oneBitAddSubConst4` using combinators.

2. Using `oneBitAddSubConst4` and the `row` combinator, design a circuit to implement the function:

$$sum = a + k_0 s_0 + k_1 s_1 + \ldots + k_{m-1} s_{m-1}$$

where $k_i = \{c_0, c_1, c_2, c_3\}$ and $s_i = \{1, -1\}$