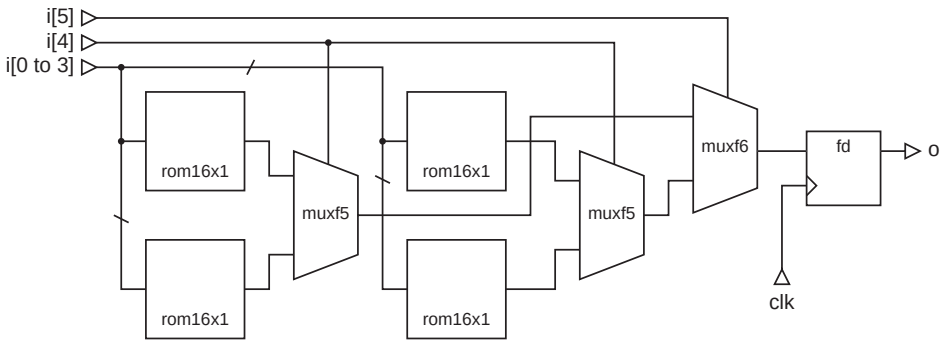
# 3 Placement with Overlay

Tutorial 2 showed how combinators can be used to place circuit blocks beside and below each other. This tutorial shows how combinators can be used to place blocks so that they are overlayed on to the same resources. Overlay can be used to achieve efficient implementations. To demonstrate this, the steps in describing a registered 6-input lookup table (LUT) design are given. In the case of Virtex, using overlay can result in a reduction in resource usage and an increase in performance as it allows designs that use multiple slice resources to be described.

**Figure 3.1** Registered Lut6 implementation using multiplexers and 4-input ROMs.




A schematic for the registered 6-input lookup table design is shown in Figure 3.1. This circuit works by addressing four 4-input lookup tables with the least-significant input bits and then selecting between their outputs with the most-significant input bits. The output is registered to provide pipelining when the 6-input lookup table is used to form larger functions.

The components in this circuit are particular to the Virtex architecture. The function generators of the CLB are configured as rom16x1 components. They have the same function as LUTs, but are not optimised automatically by the Xilinx implementation tools in terms of their contents. This is advantageous in the context of this tutorial because it allows us to observe the expected layout without any optimisations occuring. The muxf5 and muxf6 blocks are multiplexors that are implemented as dedicated logic in the CLB. They provide better performance and resource utilisation than multiplexors implemented in LUTs, when combining the outputs from the LUTs.

The composition and placement of this design is shown in two parts. The first shows how to compose the design without providing any placement. The second shows how this can be used to derive a description that provides placement with overlay. This is then compared to a LUT only implementation that does not use placement or overlay to show the advantages of this approach.

## 3.1 Composition Without Placement

The first step is to compose the final multiplexer (muxf6) in series with the flip-flop (fd). This is written in Lava as follows:

```
muxf6Reg clk = muxf6 >=> fd clk
```

Next we compose the two `rom16x1` blocks in parallel and connect their inputs together as follows:

```
romsPar init0 init1 =
  fork2 >=> rom16x1 init0 `par2` rom16x1 init1
```

The type of this composition is:

```
Int -> Int -> (Bit, Bit, Bit, Bit) -> (Bit, Bit)
```

The two `Int` arguments are the initialisation values of the ROMs. Each rom takes four wires and its type is a four element tuple. The `fork2` splits its input into a pair, in this case giving the type:

```
((Bit, Bit, Bit, Bit), (Bit, Bit, Bit, Bit))
```

which is the correct type of the parallel composition of the ROMs. The output of each ROM is a single wire so the output type of the composition is a pair. To simplify these types, we can use a type synonym:

```
type Lut4Addr = (Bit, Bit, Bit, Bit)
```

The type of `romsPar` now becomes:

```
romsPar :: Int -> Int -> Lut4Addr -> (Bit, Bit)
```

The parallel ROMs can now be composed in series with the multiplexer (muxf5). The select input of the multiplexer is not wired up during the series composition. The input type of the multiplexer is a pair with another pair as the second element. We cannot use partial application to combine the correct wires this time.

For this reason, the prelude provides the combinators `fsT` and `snD` to help navigate through pairs when doing series composition. `fsT` combines a block in parallel with a wire, and `snD` combines a wire in parallel with a block. We want to connect the outputs of the ROMs to the second element of the multiplexer input pair. The connecting outputs and inputs are pairs themselves. This composition is written in Lava as follows:

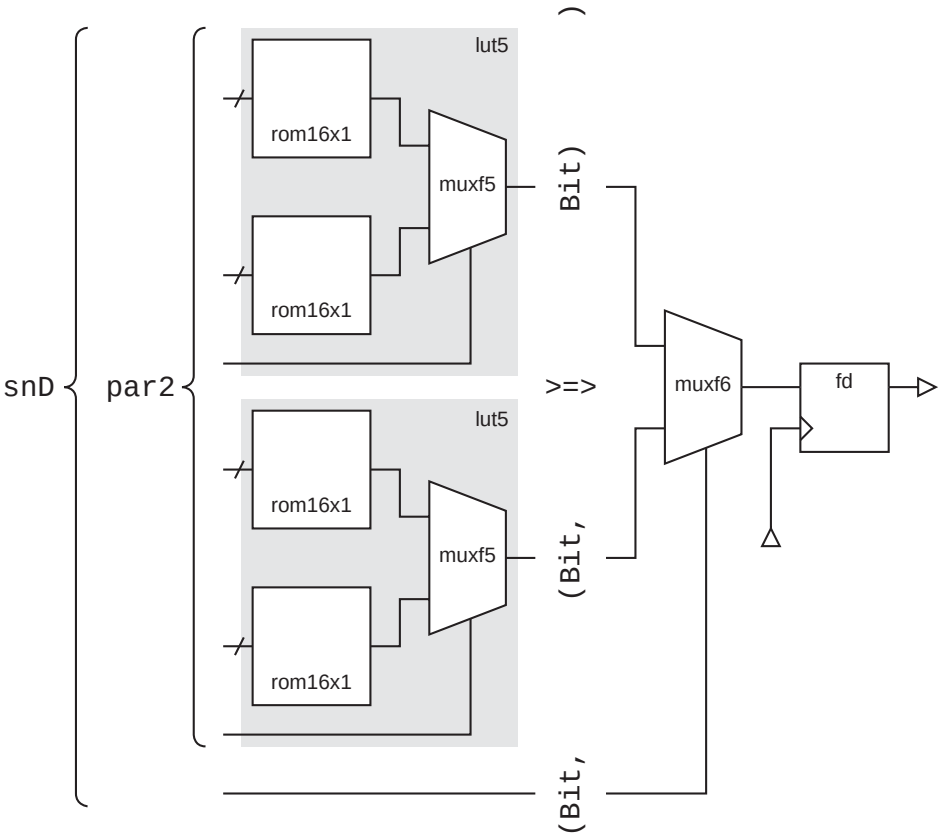
```
snD(romsPar init0 init1) >=> muxf5
```

It helps to tidy up the wiring at this stage. We have effectively composed a 5-input LUT and it is clearer if it takes a 5-element tuple where the MSB controls the select input of the multiplexer. We can do this easily by writing a custom wiring function and composing it in series with the main composition:

```
lut5 init0 init1 =
  wires >=> snD(romsPar init0 init1) >=> muxf5
  where
  wires (a, b, c, d, e) = (e, (a, b, c, d))
```

We can now compose the entire circuit by applying the same techniques. This complete composition is shown in Figure 3.2.

**Figure 3.2** Composition of the Lut6 design using series and parallel combinators.



The complete Lava code for this design is as follows:

```
type Lut4Addr = (Bit, Bit, Bit, Bit)
type Lut5Addr = (Bit, Bit, Bit, Bit, Bit)
type Lut6Addr = (Bit, Bit, Bit, Bit, Bit, Bit)

muxf6Reg :: Bit -> (Bit, (Bit, Bit)) -> Bit
muxf6Reg clk = muxf6 >=> fd clk

romsPar :: Int -> Int -> Lut4Addr -> (Bit, Bit)
romsPar init0 init1 =
  fork2 >=> rom16x1 init0 `par2` rom16x1 init1

lut5 :: Int -> Int -> Lut5Addr -> Bit
lut5 init0 init1 =
  wires >=> snD(romsPar init0 init1) >=> muxf5
  where
  wires (a, b, c, d, e) = (e, (a, b, c, d))

lut6Reg :: Int -> Int -> Int -> Int -> Bit -> Lut6Addr -> Bit
lut6Reg init0 init1 init2 init3 clk =
  wires >=> snD(fork2 >=> lut5 init0 init1 `par2` lut5 init2 init3)
    >=> muxf6Reg clk
  where
  wires (a, b, c, d, e, f) = (f, (a, b, c, d, e))
```

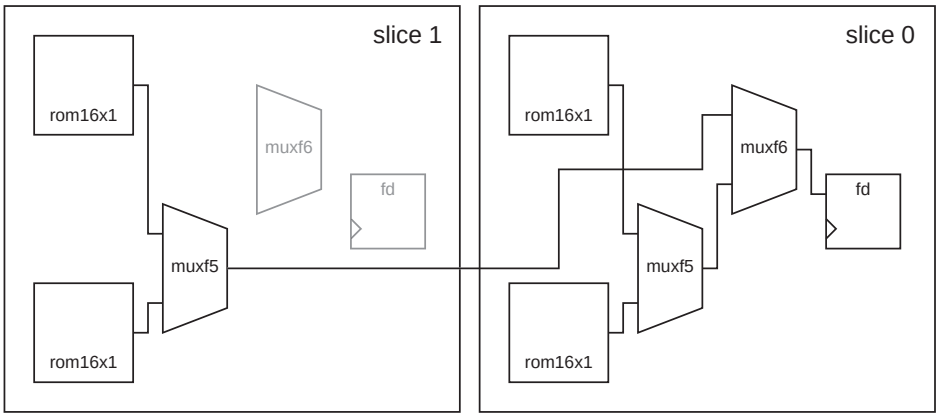
Note that type synonyms have been defined for both the 5-input LUT and 6-input LUT functions to simplify the type signature declarations.

## 3.2 Composition Using the Overlay Combinators

In addition to combinators that place circuit blocks beside and below each other, Lava provides combinators to place circuit blocks in the same location. This allows blocks to be mapped to the different resources within a single slice. Figure 3.3 shows a possible mapping of the 6-input LUT circuit blocks to a Virtex CLB. The CLB contains two slices, each of which contain a set of programmable logic blocks. These include two LUTs (which the `rom16x1` maps to), a `muxf5`, a `muxf6` and a register. The Virtex RLOC constraint specifies the CLB row, column and slice to place a circuit block. To

place the circuit blocks that map to the LUTs, they are RLOCed to the same slice and the Xilinx implementation tools allocate them to the physical resources.

**Figure 3.3** Layout of the Lut6 design within a Virtex CLB. The external connections are not shown.




To compose the 6-input LUT with the above placement, we start again by composing the final multiplexer (`muxf6`) is series with the flip-flip (`fd`). To specify that they are placed in the same slice, we use the >|> series combinator. In a similar way to before, this is written in Lava as:

```
muxf6RegPl clk = muxf6 >|> fd clk
```

The rom16x1 blocks can be composed in each slice as before, replacing the `par2` combinator with `vpar2`:

```
romsParPl init0 init1 =
  fork2 >=> rom16x1 init0 `vpar2` rom16x1 init1
```

The `muxf5` block are composed with the `rom16x1` blocks in each slice by again using the >|> series combinator:

```
lut5Pl init0 init1 =
  wires >=> snD(romsParPl init0 init1) >|> muxf5
  where
  wires (a, b, c, d, e) = (e, (a, b, c, d))
```

To compose the `muxf6RegPl` and `lut5Pl` blocks together is more tricky. This is because the blocks in slice 0 must be composed to share the same location. If they are composed as before with `lut5Pl` blocks in parallel and the `muxf6RegPl` block composed using >|>, then the `muxf6` and register would be placed in slice 1. This is because >|> places blocks in the same location with their origins aligned in the lower left. To do this we need to compose the block in slice 0 in series using >|> and then compose these in series horiztonally with the slice 1 blocks. We can the following rule to break the parallel composition of the `lut5Pl` blocks:

```
A hpar2 B = fst A >-> snd B
```

If `A = B = lut5Pl`, `C = muxf6RegPl` and `F = Fork2`, then the form of the composition we are looking for is (ignoring initial values and the clock):

```
snD(F >=> A hpar2 B) >|> C
```

Substituting for the LHS of the above rule:

```
snD(F >=> fsT A >-> snD B) >|> C
```

Distributing the `snD` and bracketing the terms we want to place together:

```
snD(F) >=> snD(fsT A) >-> (snD(snD B) >|> C)
```

Substituting `A`, `B`, `C` and `F` completes the Lava description for the placed circuit:

```
muxf6RegPl :: Bit -> (Bit, (Bit, Bit)) -> Bit
muxf6RegPl clk = muxf6 >|> fd clk

romsParPl :: Int -> Int -> Lut4Addr -> (Bit, Bit)
romsParPl init0 init1 =
  fork2 >=> rom16x1 init0 `vpar2` rom16x1 init1

lut5Pl :: Int -> Int -> Lut5Addr -> Bit
lut5Pl init0 init1 =
  wires >=> snD(romsParPl init0 init1) >|> muxf5
  where
  wires (a, b, c, d, e) = (e, (a, b, c, d))

lut6RegPl :: Int -> Int -> Int -> Int -> Bit -> Lut6Addr -> Bit
lut6RegPl init0 init1 init2 init3 clk =
  wires >=> snD(fork2) >=>
  (snD(fsT(lut5Pl init0 init1)) >-> (snD(snD(lut5Pl init2 init3)) >|> muxf6RegPl clk))
  where
  wires (a, b, c, d, e, f) = (f, (a, b, c, d, e))
```

If the 6-input LUT is described without the placement combinators, the Xilinx implementation tools will come up with a similar placement themselves (probably with the slices reversed, which makes no difference in terms of the Virtex architecture). So why go to the trouble of placing such a small circuit? The advantage is, that this can now be used to build bigger, fully placed circuits. This composition might seem complicated for the size of the circuit, but this is because it is irregular and is complicated by the need for placing circuit blocks on top of each other.

## 3.3 Comparison with a LUT Only Design

TO DO:

1) Show how to Implement the designs.

2) Show floorplans.

A 6-input lookup table can also be created using LUTs only. The following Lava code describes a LUT only version with no placement composed using named wires:

```
lut6RegNet :: Int -> Int -> Int -> Int -> Bit -> Lut6Addr -> Bit
lut6RegNet init0 init1 init2 init3 clk (i0, i1, i2, i3, i4, i5) =
  muxRegOut
  where
  lutOut0 = rom16x1 init0 (i0, i1, i2, i3)
  lutOut1 = rom16x1 init1 (i0, i1, i2, i3)
  lutOut2 = rom16x1 init2 (i0, i1, i2, i3)
  lutOut3 = rom16x1 init3 (i0, i1, i2, i3)
  muxOut0 = muxBit i4 (lutOut0, lutOut1)
  muxOut1 = muxBit i4 (lutOut2, lutOut3)
  muxRegOut = fd clk (muxBit i5 (muxOut0,muxOut1))
```

Table 1 shows the delay and resource usage of the design in Section 3.2 and the LUT only design above. By using overlay placement and the dedicated resources of the Virtex architecture, it is possible to double the speed of the design and halve the number of slices used. It would be possible to describe the design in Section 3.2 using named wires, but again, it is not possible to then place this design when composing larger designs.

**TABLE 1.** Delay and resource usage of 6-input lookup table designs.

| Design | Critical Path Delay (ns) | Number of slices |
|---|---|---|
| muxfX and overlay | 3.9 | 2 |
| LUTs only | 8.1 | 4 |

## 3.4 Exercises

1. Provide a Lava description of the 6-input LUT circuit placed using combinators for the Virtex II architecture.

2. Determine the critical path delay of the Virtex II design.