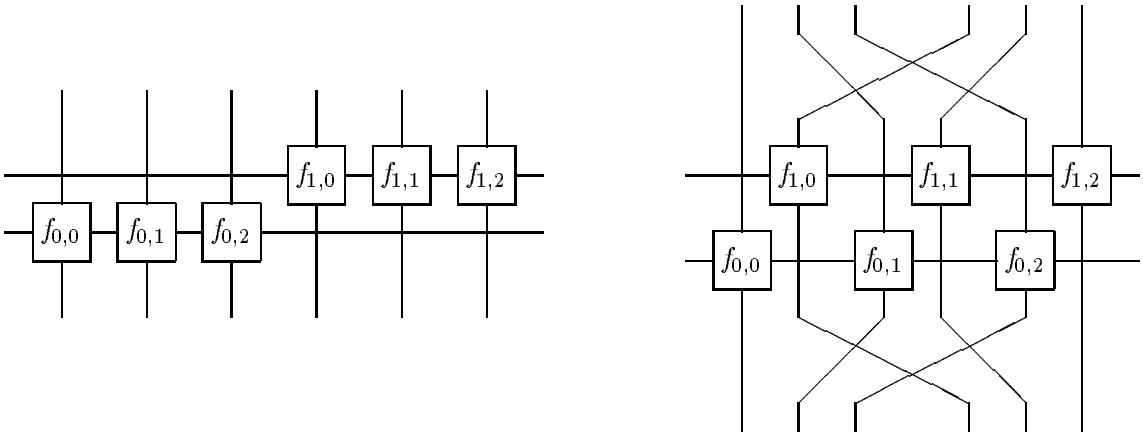
Optimising designs by transposition

Wayne Luk*

The purpose of this paper is fourfold: first, to describe some observations about how array-based designs can be optimised by transposition – a method of rearranging components and their interconnections; second, to provide concise parametric representations of such designs; third, to present simple equations that correspond to correctness-preserving transformations of these parametric representations; and finally, to suggest quantitative measures of design trade-offs involved in this kind of transformation.

Motivation

Consider the two designs in Figure 1. Notice that despite the structural differences of the circuits, the relative positions of the external connections are the same. This is achieved by the interleaving wires at the top and bottom of Design B. Transposition is the name for this kind of interleaving and also for the subsequent structure of components, since the rearrangement is similar to transposing matrices. It is clear that the functional behaviour (but not timing characteristics) of the two designs is indistinguishable; we shall show later how this situation can be expressed mathematically as a correctness-preserving transformation.



Design A Design B

Figure 1: An example of transposing designs.

*Programming Research Group, Oxford University Computing Laboratory, 11 Keble Road, Oxford, England OX1 3QD; *wayne@prg.oxford.ac.uk*

Why are these designs interesting? In our experience Design A is a generic pattern that arises frequently in the development of circuits. For instance in developing an arithmetic and logic unit, it is more convenient to specify separately the arithmetic and logic operations: $f_{0,0}, f_{0,1}$ and $f_{0,2}$ may represent full adders that process interleaved representations of integers, and $f_{1,0}, f_{1,1}$ and $f_{1,2}$ may perform a specific logic operation, depending on the horizontal input, on their vertical inputs. Figure 2(a) shows a typical configuration in which the outputs of the full adders and the logic circuits are interleaved – transposed – before feeding into an array of multiplexers that select either output. It is, however, more advantageous to perform the transposition at the outset using the transformation described in Figure 1, since we would then be able to form a single array of composite cells each consisting of a full adder, a logic circuit and a multiplexer as shown in Figure 2(b). It is the aim of the circuit designer to use as few types of cell as possible in order to reduce the design and validation effort, because attention can be focused on optimising and testing the basic units which can then be replicated with confidence.

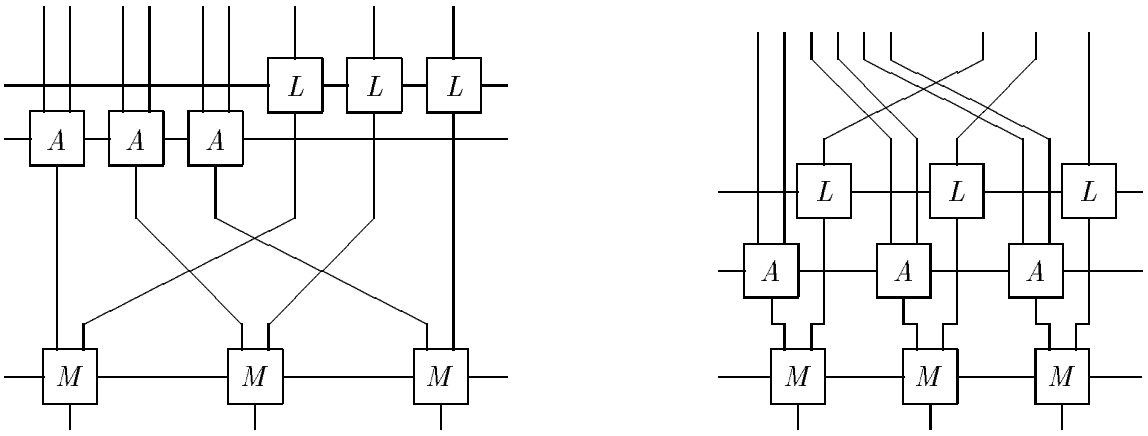
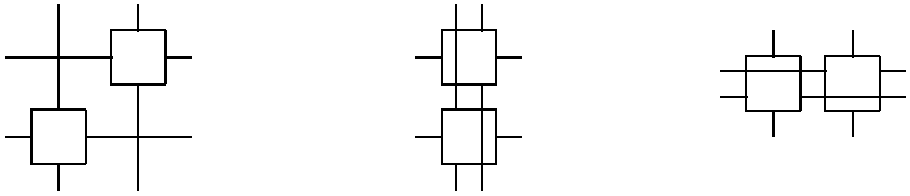


Figure 2: Two implementations of an ALU. A, L and M correspond respectively to a full adder, a logic circuit, and a multiplexer.

Another situation in which Design B in Figure 1 offers an improvement over Design A is as follows. Suppose that every component in Design A is latched. However, instead of depending on the propagation delay of the components, the clock speed of the system may be restricted by the propagation delay associated with the two long horizontal wires. In Design B the components are more evenly distributed and the problem of propagation delay is transferred to the wires implementing the transpositions. We shall see later the circumstances in which these peripheral transpositions can be eliminated to produce an efficient layout.

A further benefit of transposing components can be derived from altering the aspect ratio of a design. Given that one can overlay a wire on a component and that the connections of a component can be shifted as long as their direction and order are maintained, the configuration in Figure 3(a) can be implemented as described in Figure 3(b) or Figure 3(c). Hence if every component in Figure 1 has the same height h and the same width w, and

ignoring the contribution of the peripheral interleaving wires, then Design B may have an aspect ratio of either $2h \times 3w$ or $h \times 6w$. Design A is not as flexible; the same assumptions will result in an aspect ratio of $h \times 6w$.



(a) (b) (c)

Figure 3: Implementing transposition.

So much for motivation. The rest of the paper will describe a framework for expressing transposition parametrically, and will present a catalogue of such transformations together with quantitative measures of the trade-offs involved.

Notation

The formalism we use is based on Sheeran's relational framework and the author's heterogeneous combinators. The background and the details of this approach have been described elsewhere (see [2], [3]), and only the definitions and concepts relevant to our discussion will be introduced here.

A design will be described by a binary relation of the form $x\, f\, y \stackrel{\text{def}}{=} \mathcal{P}(x, y)$ where x, y represent the interface signals and belong to $dom(f)$ and $rng(f)$ (domain and range of f) respectively, and $\mathcal{P}$ is a predicate describing the intended behaviour. For example, an inverter can be specified as $x\ inv\ y \stackrel{\text{def}}{=} x = -y$.

The converse f^{-1} of a relation f is defined by $x\,(f^{-1})\,y \stackrel{\text{def}}{=} y\, f\, x$, and the identity relation is given by $x\ id\ y \stackrel{\text{def}}{=} x = y$. If f is a function, then $f\ x$ represents the value of f for the argument x.

Objects in our notation are either atoms (such as numbers or relations) or tuples of objects: for instance the object $\langle 0, \langle 1, 2\rangle\rangle$ is a 2-tuple containing the number 0 and the tuple $\langle 1, 2\rangle$. A tuple is an ordered collection of elements, with the empty tuple denoted by $\langle\,\rangle$. Tuples are concatenated by '$\hat{}$' (pronounced 'append'), so that

$$\langle a\rangle\hat{}\langle b, c, d\rangle = \langle a, b, c\rangle\hat{}\langle d\rangle = \langle a, b, c, d\rangle.$$

Given that x is a tuple, $\#x$ represents the number of elements in it, and x_i (where $0 \le i < \#x$) is its i-th element. Let us call a tuple of tuples a two-level tuple, a tuple of tuples of tuples a three-level tuple and so on. If x is a tuple with two or more levels, then $x_{i,j} \stackrel{\text{def}}{=} (x_i)_j$. Some operations on tuples are given below:

$$\begin{array}{rcll} x\ \pi_{n+1}\ z & \stackrel{\text{def}}{=} & z = x_n & \text{(select } n\text{-th } x,\ 0 \le n < \#x), \\ \langle x, y\rangle\ swap\ z & \stackrel{\text{def}}{=} & z = \langle y, x\rangle, & \text{(swap components of 2-tuple),} \end{array}$$

$$x \; tran \; z \quad \stackrel{\text{def}}{=} \quad \begin{array}{ll} \forall i, j : 0 \leq i < \#x, 0 \leq j < \#z. & \text{(transpose } x\text{, given that} \\ (\#x_i = \#z) \wedge (z_{j,i} = x_{i,j}) & x \text{ is a tuple of tuples).} \end{array}$$

For example, $tran\,\langle\langle 1,2,3\rangle,\langle 4,5,6\rangle\rangle = \langle\langle 1,4\rangle,\langle 2,5\rangle,\langle 3,6\rangle\rangle$. Notice that *tran* is only defined for rectangular tuples – tuples in which all sub-tuples have the same length.

A rectangular circuit with connections on every side is modelled by a relation that relates 2-tuples, with the components in the domain corresponding to signals for the west and north side and those in the range corresponding to signals for the south and east side. In general, composite signals are represented as tuples with the position of a particular signal corresponding to its relative position, and with its structure – the grouping of signals – reflecting the logical organisation of adjacent signals.

Given a circuit f with connections on all four sides, we can use the generic reverse function *recrev* which recursively reverses a tuple and all its component tuples

$$\begin{array}{rcl} recrev\; x & \stackrel{\text{def}}{=} & x \text{ if } x = \langle\,\rangle \text{ or if } x \text{ is an atom,} \\ recrev\;(\langle x\rangle\hat{}\,xs) & \stackrel{\text{def}}{=} & (recrev\; xs)\hat{}\langle recrev\; x\rangle \text{ otherwise,} \end{array}$$

to define $f^{\mathcal{V}}$ and $f^{\mathcal{H}}$ which denote respectively the reflection of f in a vertical and in a horizontal axis, and $f^{\mathcal{VH}}$ which denotes the reflection of f both vertically and horizontally:

$$\begin{array}{rcl} \langle x, y\rangle\, f^{\mathcal{V}}\, \langle x', y'\rangle & \stackrel{\text{def}}{=} & \langle y', recrev\; y\rangle\, f\, \langle recrev\; x', x\rangle, \\ \langle x, y\rangle\, f^{\mathcal{H}}\, \langle x', y'\rangle & \stackrel{\text{def}}{=} & \langle recrev\; x, x'\rangle\, f\, \langle y, recrev\; y'\rangle, \\ \langle x, y\rangle\, f^{\mathcal{VH}}\, \langle x', y'\rangle & \stackrel{\text{def}}{=} & \langle recrev\; y', recrev\; x'\rangle\, f\, \langle recrev\; y, recrev\; x\rangle. \end{array}$$

To deal with sequential circuits an expression is considered as relating a stream (an infinite tuple of data) in its domain to a stream in its range. In most cases, such as in the absence of conditionals, the same algebraic theorems can be applied to expressions representing both combinational and sequential systems [1].

Combinators

Combinators are higher-order functions that capture common patterns of computation as parametrised expressions. These patterns can represent behaviour, in which case the behaviour of a composite device is expressed in terms of the behaviour of its components; or they can represent the spatial organisation of a circuit, in which case they describe the connection of components to form the composite device. The following binary combinators are adapted from Ruby [2] to define various ways that two components can be connected together:

$$\begin{array}{rcll} x\,(f;g)\,z & \stackrel{\text{def}}{=} & \exists y.\,(x\,f\,y) \wedge (y\,g\,z) & \text{(relational composition),} \\ \langle x, y\rangle\,(f \parallel g)\,\langle x', y'\rangle & \stackrel{\text{def}}{=} & (x\,f\,x') \wedge (y\,g\,y') & \text{(parallel composition),} \\ \langle a, \langle b, c\rangle\rangle\,(f +\!\!+ g)\,\langle\langle p, q\rangle, r\rangle & \stackrel{\text{def}}{=} & \exists s.\,\langle a, b\rangle\, f\, \langle p, s\rangle \wedge \langle s, c\rangle\, g\, \langle q, r\rangle & \text{(beside),} \\ \langle\langle a, b\rangle, c\rangle\,(f \mathbin{+\!\!+} g)\,\langle p, \langle q, r\rangle\rangle & \stackrel{\text{def}}{=} & \exists s.\,\langle a, s\rangle\, f\, \langle p, q\rangle \wedge \langle b, c\rangle\, g\, \langle s, r\rangle & \text{(below).} \end{array}$$

Since $(f \mathbin{+\!\!+} g)^{-1} = f^{-1} +\!\!+ g^{-1}$, we only need to work out the properties of either *beside* or *below*.

We shall use the abbreviations $\mathsf{fst} f \stackrel{\text{def}}{=} f \parallel id$, $\mathsf{snd} f \stackrel{\text{def}}{=} id \parallel f$, $\mathsf{fsth} f \stackrel{\text{def}}{=} f +\!\!+ swap$, $\mathsf{sndh} f \stackrel{\text{def}}{=} swap +\!\!+ f$, $\mathsf{fstv} f \stackrel{\text{def}}{=} f \mathbin{+\!\!+} swap$, $\mathsf{sndv} f \stackrel{\text{def}}{=} swap \mathbin{+\!\!+} f$, $f \backslash g \stackrel{\text{def}}{=} g^{-1}; f; g$ and $f \backslash\backslash g \stackrel{\text{def}}{=}$

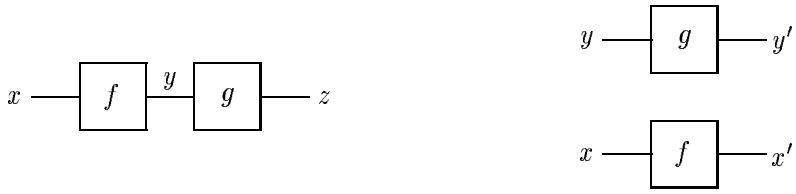


Figure 4: Relational and parallel composition.

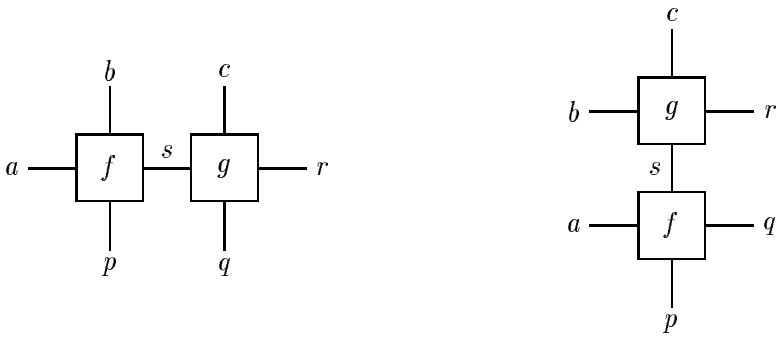
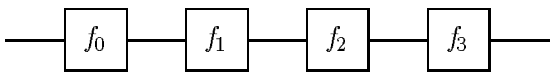


Figure 5: Beside and below.

$g^{-1}\backslash swap;f;g$. These abbreviations have interesting properties like $\mathsf{fst}f;\mathsf{fst}g = \mathsf{fst}(f;g)$, $\mathsf{fsth}(\mathsf{sndv}f) = \mathsf{sndv}(\mathsf{fsth}f)$, $(f \backslash\backslash g) \backslash\backslash h = f \backslash\backslash (g;h)$, $\mathsf{snd}f = (\mathsf{fst}f)\backslash swap$, and $\mathsf{snd}f^{-1};g;\mathsf{fst}f = g \backslash\backslash (\mathsf{fst}f)$. Note that prefix combinators have a higher precedence than infix ones, and that relational composition has the lowest precedence. The combinators $\backslash$, $\backslash\backslash$ have a lower precedence than all other binary combinators except relational composition.

One can check that in general none of the above binary combinators is commutative, and that only relational composition is associative. For instance, since $\langle\langle x, y\rangle, z\rangle$ and $\langle x, \langle y, z\rangle\rangle$ are distinct tuples, $(f \parallel g) \parallel h \neq f \parallel (g \parallel h)$. We now introduce a class of prefix combinators, called heterogeneous combinators, each of which takes a tuple of components and returns a binary relation corresponding to the composite circuit; and the components that are wired together can be different from one another. For instance, the parallel composition of three distinct components can be expressed as $(\ \|\|\ \langle f, g, h\rangle)$ which relates signals of the form $\langle x, y, z\rangle$. Given that $\#f = \#x = \#y = N$, the four common flavours of heterogeneous combinators are described below:

$$a \left(\begin{array}{c} \circ \\ \S \\ \circ \end{array} f \right) b \quad \equiv \quad \exists s.\, (s_0 = a) \wedge (s_N = b) \wedge \forall i : 0 \leq i < N.\, s_i\, f_i\, s_{i+1} \quad \text{(het. chain)},$$



$$\begin{array}{c} \circ \\ \S \\ \circ \end{array} \langle f_0, f_1, f_2, f_3\rangle$$

Figure 6: Heterogeneous chain.

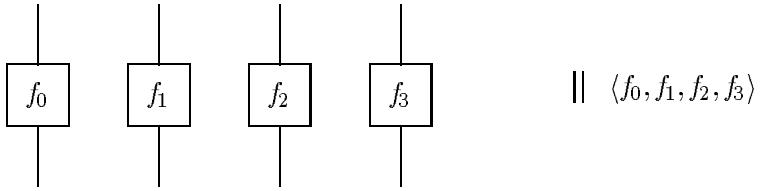


Figure 7: Heterogeneous map.

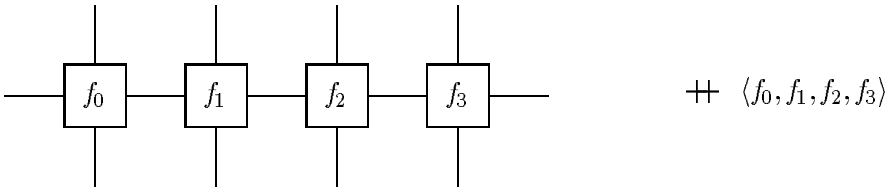


Figure 8: Heterogeneous row.

$$x \left(\| f \right) y$$
$$\equiv \forall i : 0 \le i < N . x_i \, f_i \, y_i \qquad \text{(het. map)},$$

$$\langle a, x \rangle \, (+\!\!+ f) \, \langle y, b \rangle$$
$$\equiv \exists s . (s_0 = a) \wedge (s_N = b) \wedge \forall i : 0 \le i < N . \langle s_i, x_i \rangle \, f_i \, \langle y_i, s_{i+1} \rangle \qquad \text{(het. row)},$$

$$\langle x, a \rangle \left(\ddagger f \right) \langle b, y \rangle$$
$$\equiv \exists s . (s_0 = b) \wedge (s_N = a) \wedge \forall i : 0 \le i < N . \langle x_i, s_{i+1} \rangle \, f_i \, \langle s_i, y_i \rangle \qquad \text{(het. column)}.$$

Given $\Psi \in \left\{ \fatsemi , \| , +\!\!+ , \ddagger \right\}$ and $0 \le N \le \# f$, we shall adopt the abbreviation

$$\mathop{\Psi}_{i<N} f_i \quad \stackrel{\text{def}}{=} \quad \Psi \, \langle f_i \mid 0 \le i < N \rangle .$$

The following combinator will be used to describe a rectangular grid of heterogeneous components:

$$\mathop{\#\!\#}_{i,j<M,N} f_{i,j} \quad \stackrel{\text{def}}{=} \quad \mathop{\ddagger}_{i<M} \left(\mathop{+\!\!+}_{j<N} f_{i,j} \right) .$$

For many applications the generality of heterogeneous combinators is not required. If f does not depend on i, then the following homogeneous combinators [2] can be used to

describe arrays of identical components.

$$f^N \stackrel{\text{def}}{=} \mathop{⨟}_{i<N} f \qquad \text{(chain)}, \tag{1}$$

$$\propto_N f \stackrel{\text{def}}{=} \mathop{\|}_{i<N} f \qquad \text{(map)}, \tag{2}$$

$$\mathsf{row}_N\, f \stackrel{\text{def}}{=} \mathop{+\!\!+}_{i<N} f \qquad \text{(row)}, \tag{3}$$

$$\mathsf{col}_N\, f \stackrel{\text{def}}{=} \mathop{\#}_{i<N} f \qquad \text{(column)}, \tag{4}$$

$$\mathsf{grid}_{M,N}\, f \stackrel{\text{def}}{=} \mathsf{col}_M\, (\mathsf{row}_N\, f) \qquad \text{(rectangular grid)}. \tag{5}$$

The subscripts associated with these combinators correspond to the size of the arrays, often omitted when they can be deduced from context.

Properties of transposition

Let us look at some properties of *tran* that will prove useful. If we write $\langle x_i | 0 \le i < \#x \rangle$ as $\langle x_i | i \in I \rangle$ where I is the set of subscripts $\{i | 0 \le i < \#x\}$, then the previous definition of *tran* can be rewritten as

$$tran\ \langle \langle x_{i,j} \mid j \in J \rangle \mid i \in I \rangle \quad = \quad \langle \langle x_{i,j} \mid i \in I \rangle \mid j \in J \rangle. \tag{6}$$

This shows that transposing a tuple with two or more levels corresponds to permuting the positions of the two subscript generators $i \in I$ and $j \in J$.

From Equation 6 we can conclude that transposing an expression twice leaves it unchanged:

$$tran^2 \quad = \quad id_2, \tag{7}$$

since permuting the positions of the subscript generators $i \in I$ and $j \in J$ twice will return the original expression. Here id_n is the identity relation on n-level rectangular tuples. Equation 7 also shows that $tran^{-1} = tran$.

Now consider the effect of the function $(tran;\ \propto tran)^3$ on tuples with three or more levels:

$$\begin{aligned}
&((tran\,;\propto tran)^3)\ \langle\langle\langle x_{i,j,k} \mid k \in K\rangle \mid j \in J\rangle \mid i \in I\rangle \\
&\quad = \quad (\propto tran\,;(tran\,;\propto tran)^2)\ \langle\langle\langle x_{i,j,k} \mid k \in K\rangle \mid i \in I\rangle \mid j \in J\rangle \\
&\quad = \quad ((tran\,;\propto tran)^2)\ \langle\langle\langle x_{i,j,k} \mid i \in I\rangle \mid k \in K\rangle \mid j \in J\rangle \\
&\quad = \quad (\propto tran\,;(tran\,;\propto tran))\ \langle\langle\langle x_{i,j,k} \mid i \in I\rangle \mid j \in J\rangle \mid k \in K\rangle \\
&\quad = \quad ((tran\,;\propto tran))\ \langle\langle\langle x_{i,j,k} \mid j \in J\rangle \mid i \in I\rangle \mid k \in K\rangle \\
&\quad = \quad (\propto tran)\ \langle\langle\langle x_{i,j,k} \mid j \in J\rangle \mid k \in K\rangle \mid i \in I\rangle \\
&\quad = \quad \langle\langle\langle x_{i,j,k} \mid k \in K\rangle \mid j \in J\rangle \mid i \in I\rangle,
\end{aligned}$$

so we obtain

$$(tran\,;\propto tran)^3 \quad = \quad id_3. \tag{8}$$

Notice that at each step of the above derivation, the positions of two of the subscript generators are permuted. One can generalise this result for an M-level tuple where $M > N$ by applying a cyclic permutation to the first $N+1$ subscript generators, so that

$$(tran\,;\propto tran\,;\propto^2 tran\,;\;\ldots\;;\propto^{N-1} tran)^{N+1} \quad = \quad id_{N+1}.$$

In other words,

$$\left(\mathop{\fatsemi}_{i<N} \propto^i tran\right)^{N+1} \quad = \quad id_{N+1}.$$

With $N = 1$ we obtain Equation 7, and with $N = 2$ we obtain Equation 8.

Next, we shall present three ways of eliminating *tran* from an expression. The purpose is to remove unnecessary transpositions which are wasteful of area, or to make the description more uniform so that further optimisations can be applied.

Theorems with preconditions

The first method of eliminating *tran* involves using theorems with preconditions. For example, with the precondition $h\,;g = id$, it can be shown that

$$\mathop{\fatsemi}_{i<N} (g\,;f_i\,;h) \quad = \quad g\,;\left(\mathop{\fatsemi}_{i<N} f_i\right);h,$$

$$\mathop{+\!\!+}_{i<N} (\mathsf{fst}\,g\,;f_i\,;\mathsf{snd}\,h) \quad = \quad \mathsf{fst}\,g\,;\left(\mathop{+\!\!+}_{i<N} f_i\right);\mathsf{snd}\,h,$$

$$\mathop{\#\!\#}_{i,j<M,N} (g \parallel g\,;f_{i,j}\,;h \parallel h) \quad = \quad \propto g \parallel \propto g\,;\mathop{\#\!\#}_{i,j<M,N} f_{i,j}\,;\propto h \parallel \propto h,$$

since the g's and h's within the array cancel one another. Instances of these theorems include

$$\begin{aligned}(g\,;f\,;h)^N &= g\,;f^N\,;h,\\ \mathsf{row}\,(\mathsf{fst}\,g\,;f\,;\mathsf{snd}\,h) &= \mathsf{fst}\,g\,;\mathsf{row}\,f\,;\mathsf{snd}\,h,\\ \mathsf{grid}\,(g \parallel g\,;f\,;h \parallel h) &= \propto g \parallel \propto g\,;\mathsf{grid}\,f\,;\propto h \parallel \propto h.\end{aligned}$$

Now we can use the result obtained in the last section: the above theorems will be applicable for $g = tran$ and $h = tran$, or $g = (tran\,;\propto tran)$ and $h = (tran\,;\propto tran)^2$, and so on. In this way transpositions in the wiring between adjacent cells can be eliminated.

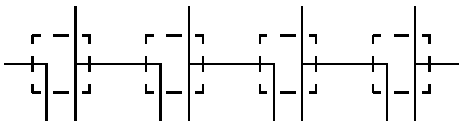
Array of wiring cells

The second method of eliminating *tran* involves developing an array of wiring cells that corresponds to an expression involving *tran*. For instance, given that

$$\begin{aligned}x\ fork\ y &\stackrel{\text{def}}{=} y = \langle x,x\rangle && \text{(duplication)},\\ \langle x,xs\rangle\ apl\ y &\stackrel{\text{def}}{=} y = \langle x\rangle\hat{}xs && \text{(append left)},\\ \langle xs,x\rangle\ apr\ y &\stackrel{\text{def}}{=} y = xs\hat{}\langle x\rangle && \text{(append right)},\\ \langle\langle x\rangle\hat{}xs,y\rangle\ shl\ u &\stackrel{\text{def}}{=} u = \langle x,xs\hat{}\langle y\rangle\rangle && \text{(shift left)},\\ \langle x,xs\hat{}\langle y\rangle\rangle\ shr\ u &\stackrel{\text{def}}{=} u = \langle\langle x\rangle\hat{}xs,y\rangle && \text{(shift right)},\end{aligned}$$

it can be shown that

$$fork\,;(apr^{-1};\pi_1)\parallel(apl^{-1};\pi_2)\,;tran \quad = \quad apl^{-1}\,;\mathsf{row}\,(\mathsf{snd}\,fork;\,shr)\,;\pi_1,$$



$\mathsf{row}_4\,(\mathsf{snd}\,fork;shr)$

Figure 9: A row of wiring cells.

The right-hand side of the above equation is in the form of a row (Figure 9) and can be optimised further, such as combining its components with those of adjacent arrays. Other examples include

$$\begin{aligned}(f\parallel g\;\setminus\;tran)\mathbin{\backslash\!\backslash}\mathsf{snd}\,swap &= (\mathsf{sndv}\,f)\mathbin{+\!\!+}(\mathsf{fstv}\,g),\\ f\parallel g\;\setminus\;tran &= (\mathsf{fstv}\,f)\mathbin{+\!\!+}(\mathsf{sndv}\,g).\end{aligned}$$

Duplication and truncation

A further possibility of eliminating *tran* arises when signals are being duplicated or discarded. For instance, given the function dup_N which duplicates its argument N times,

$$x\;dup_N\;y \quad \stackrel{\text{def}}{=} \quad (\#y = N)\wedge(\forall i : 0\le i<N.\,y_i = x),$$

it can be shown that

$$dup_N\,;tran \quad = \quad \propto dup_N.$$

Similarly, for rectangular tuples of length less than or equal to i, we have

$$tran\,;\pi_i \quad = \quad \propto \pi_i.$$

Transposition theorems

Transposition theorems belong to one of several classes of theorems useful for circuit optimisation [4]. They relate circuits interleaved in different ways. In this section we give a number of transposing theorems that have been found useful in developing designs. Theorems involving columns can be derived from the corresponding theorems for rows, since $\mathsf{col}\,f \;=\; (\mathsf{row}\,(f^{-1}))^{-1}$.

Linear structures will be discussed first. The theorems obtained will then be extended to cover rectangular structures.

Linear arrays

First, let us consider a linear array that can be described as a map of maps. The components of the array can be transposed if the domain and the range signals are transposed accordingly

(Figure 10):

$$\|_{i<M} \left(\|_{j<N} f_{i,j} \right) \quad = \quad \|_{j<N} \left(\|_{i<M} f_{i,j} \right) \setminus tran. \tag{9}$$

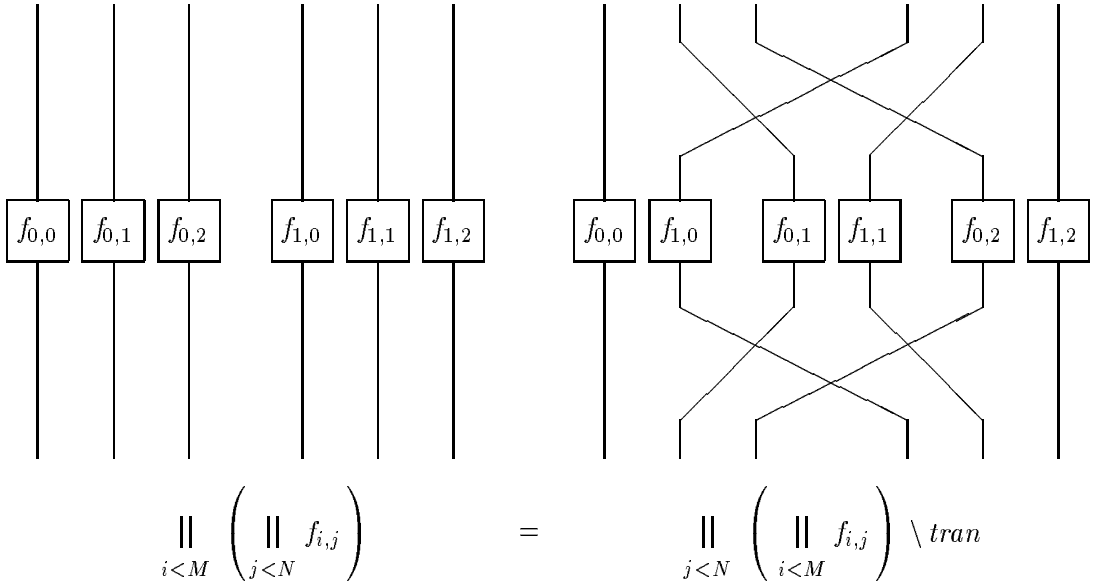


Figure 10: A theorem for transposing a map of maps ($M = 2$, $N = 3$).

A common special case of this theorem is

$$\propto (f \parallel g) \quad = \quad (\propto f \parallel \propto g) \setminus tran.$$

Next, we consider a collection of rows displaced horizontally and vertically from each other. They can be combined into a single row with the displacement transferred to the components; this usually results in a more compact layout as described in the introductory section:

$$\|_{i<M} \left(+\!\!+_{j<N} f_{i,j} \right) \setminus tran \quad = \quad +\!\!+_{j<N} \left(\|_{i<M} f_{i,j} \setminus tran \right) \setminus\!\!\setminus \mathsf{fst}\ tran. \tag{10}$$

An example of the circuits represented by this equation, with $M = 2$ and $N = 3$, is given in Figure 1.

Special cases of Equation 10 include

$$\begin{aligned} \propto (f +\!\!+ g) \setminus tran &= (\propto f \setminus tran) +\!\!+ (\propto g \setminus tran) \setminus\!\!\setminus \mathsf{fst}\ tran, \\ (\mathsf{row} f \parallel \mathsf{row} g) \setminus tran &= \mathsf{row} (f \parallel g \setminus tran) \setminus\!\!\setminus \mathsf{fst}\ tran. \end{aligned}$$

Rectangular arrays

The theorem for transposing linear arrays can be extended to deal with rectangular arrays:

$$\|_{k<K} \left(\#_{i,j<M,N} f_{k,i,j} \right) \setminus tran \quad = \quad \#_{i,j<M,N} \left(\|_{k<K} f_{k,i,j} \setminus tran \right) \setminus (tran \parallel tran). \qquad (11)$$

Special cases of this theorem include

$$\begin{aligned} \propto(\mathsf{grid}\, f) \setminus tran &= \mathsf{grid}\,(\propto f \setminus tran) \setminus (tran \parallel tran), \\ (\mathsf{grid} f \parallel \mathsf{grid} g) \setminus tran &= \mathsf{grid}\,(f \parallel g \setminus tran) \setminus (tran \parallel tran). \end{aligned}$$

Distributing components in arrays with bends

In this section we present some theorems for reasoning about arrays with bends in them. The idea is to distribute the components as evenly as possible throughout the network. The circuit *bend* is a piece of wire that bends backwards:

$$\langle x, y \rangle \; bend \; z \quad \stackrel{\text{def}}{=} \quad x = y.$$

Note that x and y can themselves be tuples. In such cases cross-overs are needed to maintain the order of the signals (Figure 11). z is a dummy signal that corresponds to an unused connection.

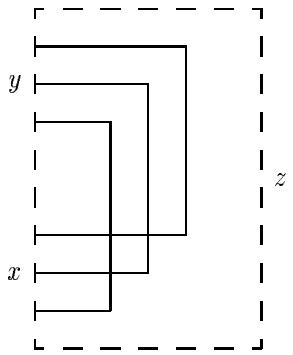


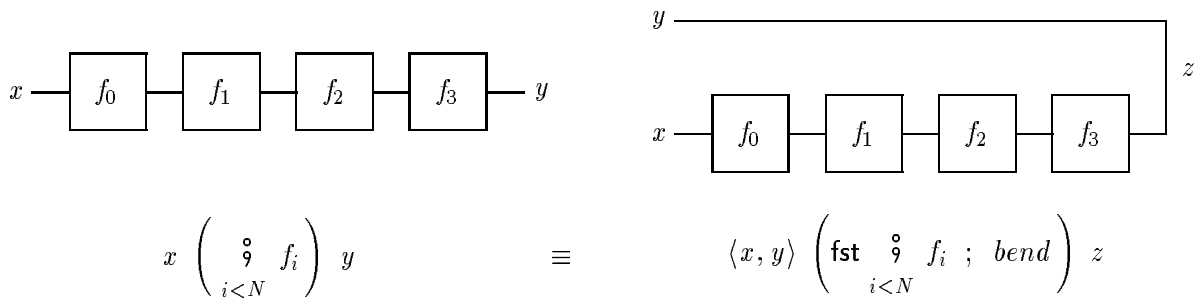
Figure 11: A vertical instance of the circuit *bend*.

We shall start by looking at how components can be distributed in linear arrays with a single bend. The results will then be extended to include linear and rectangular arrays with multiple bends.

Linear arrays with a single bend

A bend can be added to a chain of N components (Figure 12):

$$x \left(\fatsemi_{i<N} f_i \right) y \quad \equiv \quad \langle x, y \rangle \left(\mathsf{fst} \fatsemi_{i<N} f_i \; ; \; bend \right) z.$$

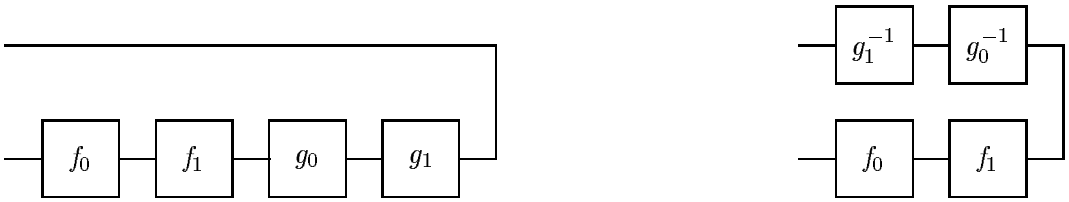


$$x \left(\mathop{⨟}_{i<N} f_i \right) y \quad \equiv \quad \langle x, y \rangle \left(\mathsf{fst} \mathop{⨟}_{i<N} f_i \; ; \; bend \right) z$$

Figure 12: Adding a vertical bend to a chain ($N = 4$).

We can now move some of the components from the lower branch to the upper branch (Figure 13):

$$\begin{aligned} \mathsf{fst} \left(\mathop{⨟}_{i<N} f_i \; ; \; \mathop{⨟}_{i<N} g_i \right) ; \; bend &= \left(\mathop{⨟}_{i<N} f_i \right) \parallel \left(\mathop{⨟}_{i<N} g_{N-i-1}^{-1} \right) ; \; bend \\ &= \mathop{⨟}_{i<N} (f_i \parallel g_{N-i-1}^{-1}) ; bend. \end{aligned} \tag{12}$$



$$\mathsf{fst} \left(\mathop{⨟}_{i<N} f_i \; ; \; \mathop{⨟}_{i<N} g_i \right) ; \; bend \quad = \quad \mathop{⨟}_{i<N} (f_i \parallel g_{N-i-1}^{-1}) ; bend$$

Figure 13: Transforming a chain with a vertical bend ($N = 2$).

For homogeneous arrays Equation 12 becomes

$$\mathsf{fst}\,(f^N ; g^N) ; bend \quad = \quad (f \parallel g^{-1})^N ; bend.$$

Similarly, a bend can be added to a component with connections on every side:

$$\langle x, y \rangle \; f \; \langle u, v \rangle \quad \equiv \quad \langle \langle x, v \rangle, y \rangle \; \left(\mathsf{fstv}\, f \; ; \; \mathsf{snd}\, bend \; ; \; \pi_1 \right) \quad u.$$

One can move the component to the upper branch provided that it is reflected vertically,

$$\mathsf{fstv}\, f \; ; \; \mathsf{snd}\, bend \quad = \quad (\mathsf{sndv}\, f^{\mathcal{V}}) \setminus\!\setminus (\mathsf{fst}\, recrev) \; ; \; \mathsf{snd}\, bend.$$

From this, it is easy to derive an equation which introduces transposition to a circuit with two components:

$$\mathsf{fstv}\,(f \mathbin{+\!\!+} g)\ ;\ \mathsf{snd}\,bend \quad = \quad (f \parallel g^{\mathcal{V}} \setminus tran) \mathbin{\backslash\!\backslash} \mathsf{fst}\,(\mathsf{snd}\,recrev)\ ;\ \mathsf{snd}\,bend.$$

If both components are themselves rows of N components, then we can move half of the components from the lower branch to the upper branch and transpose the resulting circuit (Figure 14):

$$\begin{aligned}
&\mathsf{fstv}\left(\mathop{+\!\!+}_{i<N} f_i \mathbin{+\!\!+} \mathop{+\!\!+}_{i<N} g_i\right)\ ;\ \mathsf{snd}\,bend \\
&\quad = \quad \left(\left(\mathop{+\!\!+}_{i<N} f_i \parallel \mathop{+\!\!+}_{i<N} g^{\mathcal{V}}_{N-i-1}\right) \setminus tran\right) \mathbin{\backslash\!\backslash} \mathsf{fst}\,(\mathsf{snd}\,recrev)\ ;\ \mathsf{snd}\,bend \\
&\quad = \quad \mathop{+\!\!+}_{i<N} (f_i \parallel g^{\mathcal{V}}_{N-i-1} \setminus tran) \mathbin{\backslash\!\backslash} \mathsf{fst}\,transrev\ ;\ \mathsf{snd}\,bend, \qquad (13)
\end{aligned}$$

where

$$transrev \quad \stackrel{\text{def}}{=} \quad tran\ ;\ \mathsf{snd}\,recrev.$$

If the two arrays are homogeneous then Equation 13 becomes

$$\mathsf{fstv}\,(\mathsf{row}\,f \mathbin{+\!\!+} \mathsf{row}\,g)\,;\mathsf{snd}\,bend \quad = \quad \mathsf{row}(f \parallel g^{\mathcal{V}} \setminus tran) \mathbin{\backslash\!\backslash} \mathsf{fst}\,transrev\ ;\ \mathsf{snd}\,bend.$$

Linear arrays with multiple bends

We now explore how multiple bends can be introduced in linear arrays. First, observe that a bend can be twisted to form a zig-zag piece of wire containing three bends:

$$bend \quad = \quad \mathsf{snd}\,(\pi_2^{-1}\ ;\ shl \mathbin{\backslash\!\backslash} (\mathsf{snd}\,bend)\ ;\ \pi_1)\ ;\ bend$$

This process can be continued so that a single bend can be twisted to form a piece of wire containing an odd number of bends:

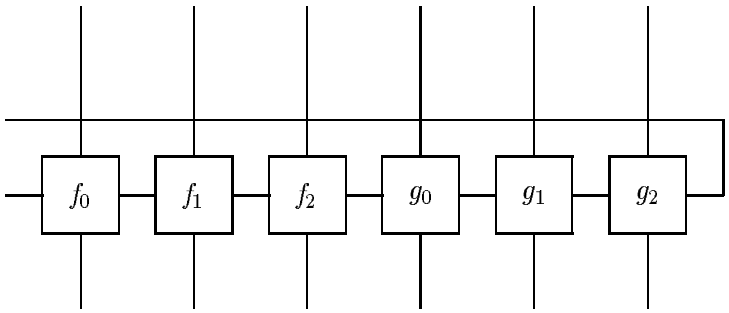
$$bend \quad = \quad \mathsf{snd}\,(\pi_2^{-1}\ ;\ (\mathsf{col}_N\,shl) \mathbin{\backslash\!\backslash} \mathsf{snd}\,(\propto_N bend)\ ;\ \pi_1)\ ;\ bend.$$

In the same vein one can add $2N-1$ bends to a chain of $2N$ components by the following theorem:

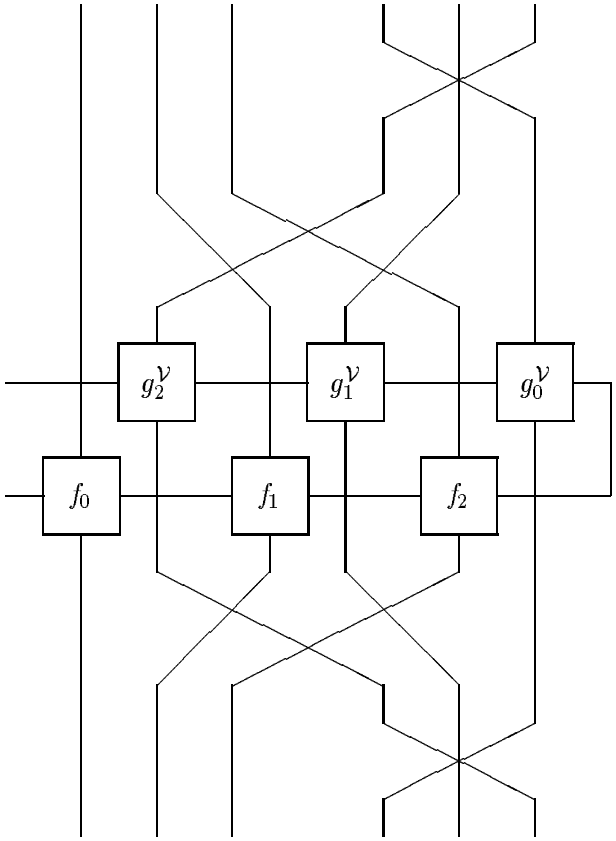
$$\mathsf{fst}\left(\mathop{⨾}_{i<2N} f_i\right)\ ;\ bend \quad = \quad sndbends_N\,;\,\mathsf{fst}\left(\mathsf{fst}\mathop{⨾}_{i<2N} f_i\right) \setminus apl\,;\,\propto_N bend$$

where

$$sndbends_N \quad \stackrel{\text{def}}{=} \quad \mathsf{snd}\,(\pi_2^{-1}\ ;\ \mathsf{fst}\,(\propto_{N-1} bend^{-1})\ ;\ \mathsf{col}_{N-1}\,shl)\ ;\ shr\ ;\ apl.$$

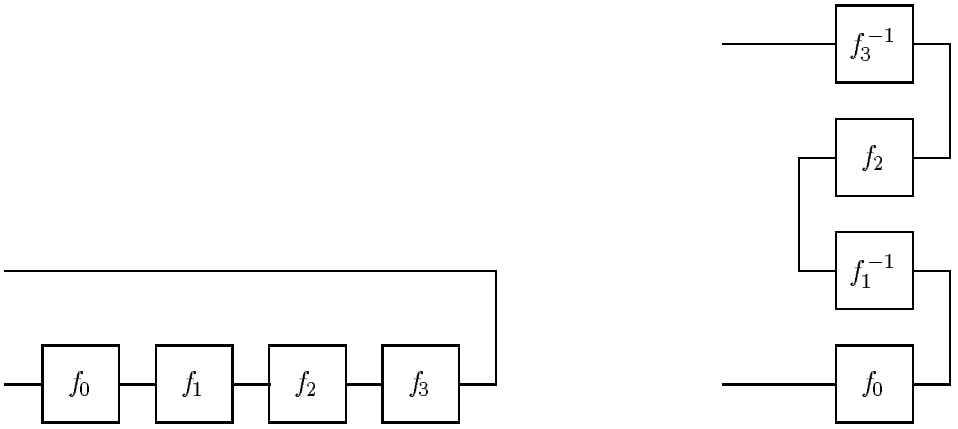


$$\mathsf{fstv}\left(\mathop{\#\!\#}_{i<N} f_i \;\#\; \mathop{\#\!\#}_{i<N} g_i\right) ;\ \mathsf{snd}\ \mathit{bend}$$



$$= \mathop{\#\!\#}_{i<N} (f_i \parallel g_{N-i-1}^{\mathcal{V}} \setminus \mathit{tran}) \mathbin{\backslash\!\backslash} \mathsf{fst}\ \mathit{transrev}\ ;\ \mathsf{snd}\ \mathit{bend}$$

Figure 14: Transforming a row with a vertical bend ($N = 3$).



$$\mathsf{fst}\left(\mathop{\fatsemi}_{i<2N} f_i\right)\,;\,bend \qquad = \qquad sndbends_N\,;\,\mathop{\|}_{i<N}(f_{2i}\,\|\,f_{2i+1}^{-1})\,;\,\propto_N bend$$

Figure 15: Adding multiple bends to a chain ($N = 2$).

We then move some components from the bottom branch onto the other branches (Figure 15):

$$\mathsf{fst}\left(\mathop{\fatsemi}_{i<2N} f_i\right)\,;\,bend \quad = \quad sndbends_N\,;\,\mathop{\|}_{i<N}(f_{2i}\,\|\,f_{2i+1}^{-1})\,;\,\propto_N bend. \tag{14}$$

The same procedure can be applied to a row of components (Figure 16):

$$\mathsf{fstv}\left(\mathop{+\!\!+}_{i<N}(f_{2i} +\!\!+ f_{2i+1})\right)\,;\,\mathsf{snd}\,bend \quad = \quad \mathsf{fst}\,sndbends_N\,;\,\Phi_N\,;\,\mathsf{snd}\propto_N bend$$

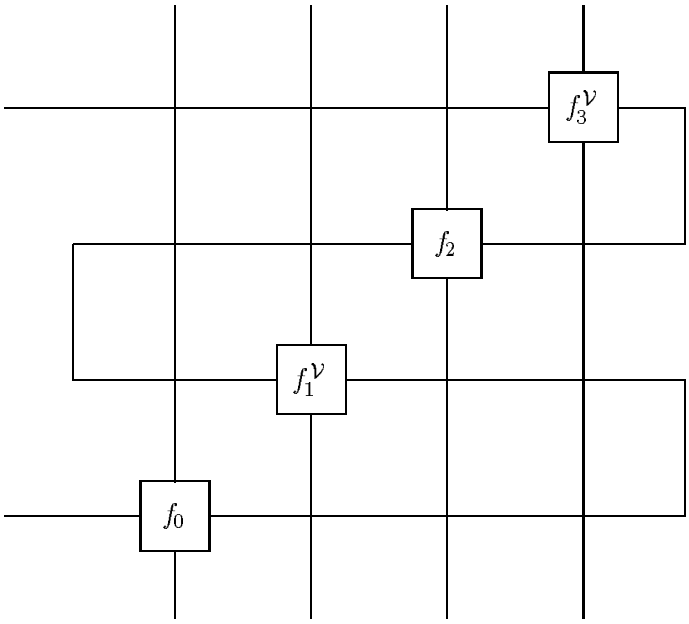
where

$$\Phi_N \quad \stackrel{\text{def}}{=} \quad \left(\mathop{\|}_{i<N}(f_{2i}\,\|\,f_{2i+1}^{\mathcal{V}}\setminus tran)\setminus tran\right) \;\setminus\!\!\setminus\; \mathsf{fst}\propto(\mathsf{snd}recrev). \tag{15}$$

The above two theorems can be specialised to deal with homogeneous arrays:

$$\begin{aligned}&\mathsf{fst}\,f^{2N}\,;\,bend\\ &\quad = \quad sndbends_N\,;\,\propto_N(f\,\|\,f^{-1})\,;\,\propto_N bend,\end{aligned}$$

$$\begin{aligned}&\mathsf{fstv}\,(\mathsf{row}_N\,(f +\!\!+ f))\,;\,\mathsf{snd}\,bend\\ &\quad = \quad \mathsf{fst}\,sndbends_N\,;\,(\propto_N(f\,\|\,f^{\mathcal{V}}\setminus tran)\setminus tran)\setminus\!\!\setminus\mathsf{fst}\propto(\mathsf{snd}recrev)\,;\,\mathsf{snd}\propto bend_N.\end{aligned}$$



$$\mathsf{fst}\, sndbends_N \,;\, \left(\mathop{\|}_{i<N} (f_{2i} \parallel f_{2i+1}^{\mathcal{V}} \setminus tran) \setminus tran \right) \;\setminus\!\!\setminus\; \mathsf{fst} \propto (\mathsf{snd} recrev) \,;\, \mathsf{snd} \propto_N bend$$

Figure 16: Adding multiple bends to a row of components ($N = 2$).

Rectangular arrays with multiple bends

In this section we extend the results for linear arrays to cover rectangular arrays. First, let us state two theorems for rearranging vertical bends in rectangular arrays:

$$\mathsf{fstv} \left(\mathop{\#\!\#}_{i,j<M,N} f_{i,j} \;+\!\!+\; \mathop{\#\!\#}_{i,j<M,N} g_{i,j} \right) \;;\, \mathsf{snd}\, bend$$

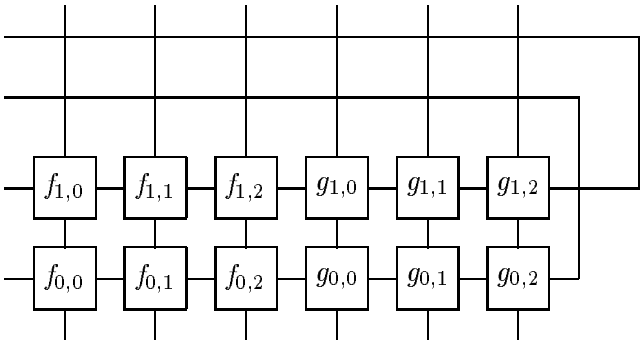
$$= \mathsf{fst}\, tran \,;\, \mathop{\#\!\#}_{i,j<M,N} (\mathsf{fstv}\, f_{i,j}) \;+\!\!+\; \mathop{\#\!\#}_{i,j<M,N} (\mathsf{fstv}\, g_{i,j}) \,;\, \mathsf{snd}\, (\propto bend) \qquad (16)$$

$$= \mathsf{fst}\, tran \,;\, \mathop{\#\!\#}_{i,j<M,N} (f_{i,j} \parallel g_{i,N-j-1}^{\mathcal{V}} \setminus tran) \;\setminus\!\!\setminus\; \mathsf{fst}\, transrev \;;\, \mathsf{snd}\, (\propto bend) \qquad (17)$$

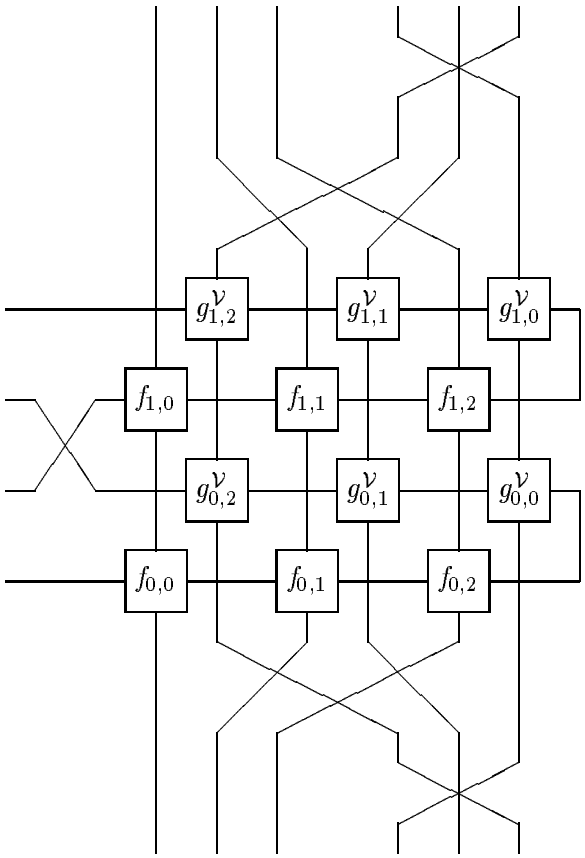
where $transrev \stackrel{\text{def}}{=} tran \,;\, \mathsf{snd}\, recrev$ as before. Equation 16 corresponds to implementing the feedback paths as internal wiring cells, and Equation 17 corresponds to a transposition forming composite cells each consisting of an $f_{i,j}$ and a reflected version of $g_{i,N-j-1}$ (Figure 17).

Again these can be specialised for homogeneous grids, for example:

$$\mathsf{fstv}\, (\mathsf{grid} f \;+\!\!+\; \mathsf{grid} g) \,;\, \mathsf{snd}\, bend$$
$$= \mathsf{fst}\, tran \,;\, \mathsf{grid}\, (f \parallel g^{\mathcal{V}} \setminus tran) \;\setminus\!\!\setminus\; \mathsf{fst}\, transrev \;;\, \mathsf{snd}\, (\propto bend).$$



$$\mathsf{fstv}\left(\mathop{\#\!\#}_{i,j<M,N} f_{i,j} \;\#\; \mathop{\#\!\#}_{i,j<M,N} g_{i,j}\right) \;;\; \mathsf{snd}\, bend$$



$$= \qquad \mathsf{fst}\, tran \;;\; \mathop{\#\!\#}_{i,j<M,N} (f_{i,j} \parallel g^{\mathcal{V}}_{i,N-j-1} \setminus tran) \mathbin{\backslash\!\backslash} \mathsf{fst}\, transrev \;;\; \mathsf{snd}\,(\propto bend)$$

Figure 17: Rectangular arrays with multiple vertical bends ($M = 2$, $N = 3$).

Next, consider a grid consisting of four heterogeneous arrays:

$$grid4\,\langle p,q,r,s\rangle \stackrel{\text{def}}{=} \left(\mathop{\#\!\#}_{i,j<M,N} p_{i,j} \mathbin{+\!\!+} \mathop{\#\!\#}_{i,j<M,N} q_{i,j}\right) \ddagger \left(\mathop{\#\!\#}_{i,j<M,N} r_{i,j} \mathbin{+\!\!+} \mathop{\#\!\#}_{i,j<M,N} s_{i,j}\right).$$

A rectangular array with vertical and horizontal bends (the top diagram of Figure 19) can be described by

$$grid4bends\,\langle p,q,r,s\rangle \stackrel{\text{def}}{=} \mathsf{fsth}\,(\mathsf{fstv}(grid4\,\langle p,q,r,s\rangle))\,;\,bend \parallel bend.$$

The first optimisation, as before, is to implement the feedback paths as wiring cells within the array. This is captured by the following theorem:

$$grid4bends\,\langle p,q,r,s\rangle = map2tran\,;\,grid4\,\langle \Omega\,p,\,\Omega\,q,\,\Omega\,r,\,\Omega\,s\rangle\,;\,map2bend \qquad (18)$$

where

$$\begin{aligned}
\Omega\,f &\stackrel{\text{def}}{=} \mathop{\#\!\#}_{i,j<M,N} (\mathsf{fsth}\,(\mathsf{fstv}\,f_{i,j})),\\
map2tran &\stackrel{\text{def}}{=} (tran\,;\propto tran) \parallel (tran\,;\propto tran),\\
map2bend &\stackrel{\text{def}}{=} (\propto bend \parallel \propto bend) \parallel (\propto bend \parallel \propto bend).
\end{aligned}$$

However, there are still long combinational paths in the circuit which can be eliminated by a further rearrangement of components. Before doing that, let us define the combinator representing the wiring pattern that will be required (Figure 18):

$$transwap\,\langle a,b,c,d\rangle \stackrel{\text{def}}{=} ((\mathsf{sndh}\,a \parallel \mathsf{sndh}\,b) \setminus tran) \ddagger ((\mathsf{fsth}\,c \parallel \mathsf{fsth}\,d) \setminus tran).$$

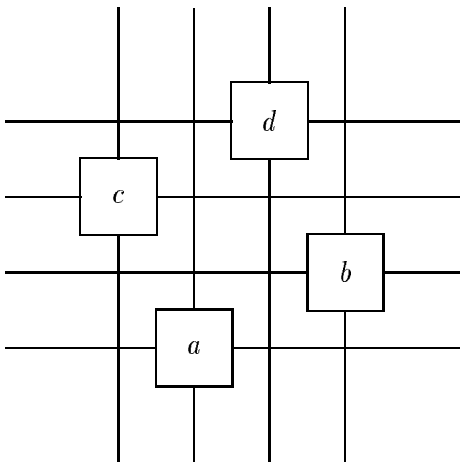
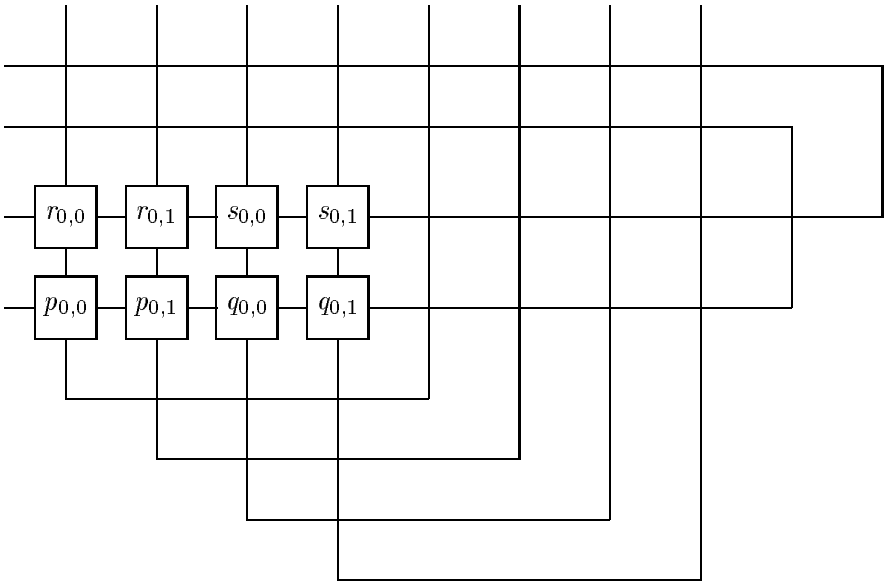
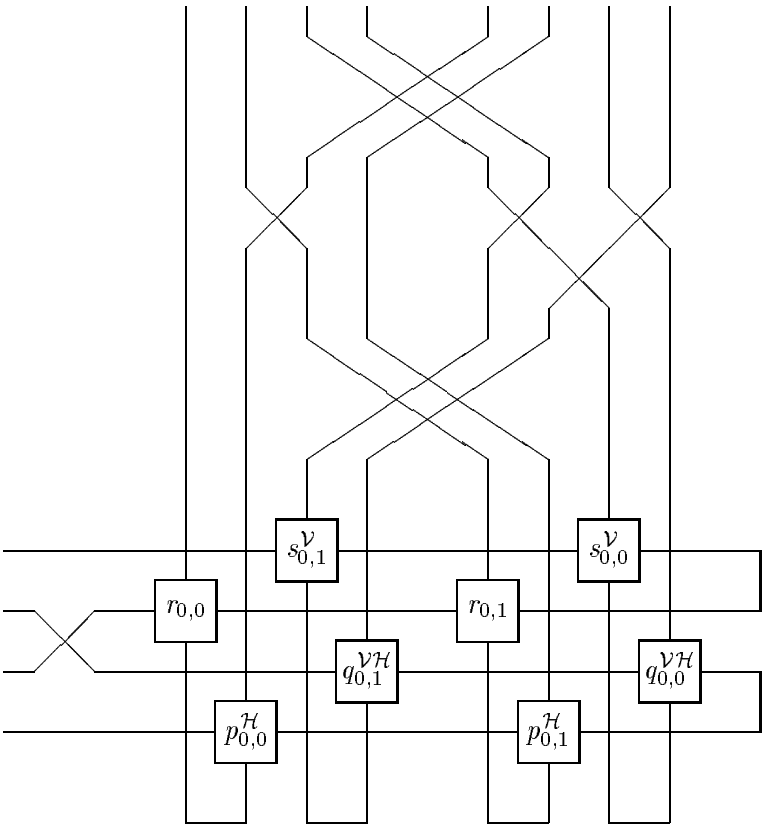


Figure 18: *transwap* $\langle a,b,c,d\rangle$.



$$\mathsf{fsth}\,(\mathsf{fstv}(\mathit{grid4}\,\langle p, q, r, s\rangle))\,;\,\mathit{bend}\;\parallel\;\mathit{bend}$$



$$= \mathit{maprevs}\,;\; \#_{i,j<M,N}\; \left(\mathit{transwap}\,\langle p^{\mathcal{H}}_{M-i-1,j}, q^{\mathcal{VH}}_{M-i-1,N-j-1}, r_{i,j}, s^{\mathcal{V}}_{i,N-j-1}\rangle\right)\;;\; \mathit{mapbend2}$$

Figure 19: Rectangular arrays with vertical and horizontal bends ($M = 1$, $N = 2$).

A useful result concerning *transwap* is

$$\mathsf{fstv}\,(\mathsf{fsth} \;\#\!\!\#\; \langle\langle a, b\rangle, \langle c, d\rangle\rangle)\,;\, bend \parallel bend \;=\; revs\,;\, transwap\,\langle a^{\mathcal{H}}, b^{\mathcal{VH}}, c, d^{\mathcal{V}}\rangle\,;\, parbend2,$$

where

$$\begin{aligned} revs &\stackrel{\text{def}}{=} (tran;\mathsf{fst}\,(recrev \parallel recrev)) \parallel (tran;\mathsf{snd}\,(recrev \parallel recrev)),\\ parbend2 &\stackrel{\text{def}}{=} (bend \parallel bend) \parallel (bend \parallel bend). \end{aligned}$$

Using this we can obtain a theorem that produces a more even distribution of components for $grid4bends\,\langle p, q, r, s\rangle$, an instance of which is depicted in Figure 19:

$$\begin{aligned} &grid4bends\,\langle p, q, r, s\rangle\\ &\quad= maprevs\,;\; \#\!\!\#_{i,j<M,N} \left(transwap\,\langle p^{\mathcal{H}}_{M-i-1,j}, q^{\mathcal{VH}}_{M-i-1,N-j-1}, r_{i,j}, s^{\mathcal{V}}_{i,N-j-1}\rangle\right)\;;\\ &\qquad mapbend2 \end{aligned} \tag{19}$$

where

$$\begin{aligned} maprevs &\stackrel{\text{def}}{=} (tran\,;\,\mathsf{fst} \propto recrev\,;\, \propto tran\,;\, tran) \parallel (tran\,;\,\mathsf{snd} \propto recrev\,;\, \propto tran\,;\, tran),\\ mapbend2 &\stackrel{\text{def}}{=} \propto(bend \parallel bend) \parallel \propto(bend \parallel bend). \end{aligned}$$

These theorems can be specialised for homogeneous networks as well; for instance one can show that

$$\begin{aligned} &\mathsf{fstv}\,(\mathsf{fsth}\,(\mathsf{grid}_{2,2}\,(\mathsf{grid}_{M,N}\,f)))\,;\, bend \parallel bend\\ &\quad= maprevs\,;\, \mathsf{grid}_{M,N}\,(transwap\,\langle f^{\mathcal{H}}, f^{\mathcal{VH}}, f, f^{\mathcal{V}}\rangle)\,;\, mapbend2. \end{aligned}$$

Trade-off analysis

This section suggests some quantitative measures of the trade-offs involved in the transformations discussed in the preceding sections. As explained in the introductory section, transposition usually results in a more uniform distribution of components in the array at the expense of increasing the complexity of the interface. In the following estimation of size and performance the contribution of the interface will not be included since it is usually application- or implementation-dependent. Hence the reader must check for the particular situation whether the improvement brought by the transformation is offset by the increased complexity of the interface.

Table 1 summarises the effects of various transposition strategies introduced earlier. The calculation is based on the simplified case in which all components have the same aspect ratio $h \times w$. We also assume, as explained in the introductory section, that wires can be overlaid on components in assessing the alteration of aspect ratio, and that all components are latched in estimating the reduction of long combinational paths.

Table 1: Summary of trade-offs in transposition strategies, showing the features of the designs represented by the left- and the right-hand side of a given equation in the text.

Equation	LHS aspect ratio	RHS aspect ratio	LHS longest path	RHS longest path
9	$h \times MNw$	$h \times MNw$ or $Mh \times Nw$	0	h or 0
10	$h \times MNw$	$h \times MNw$ or $Mh \times Nw$	h or Nw	h or w
11	$KMh \times KNw$	$KMh \times Nw$ or $Mh \times KNw$	$(K-1)Mh$ or $(K-1)Nw$	$(K-1)h$ or $(K-1)w$
14	$h \times 2Nw$	$2Nh \times w$	$2Nw$	0
15	$h \times 2Nw$	$2Nh \times w$	$2Nw$	$(2N-1)h$
16	$Mh \times 2Nw$	$Mh \times 2Nw$	$(M-1)h + 2Nw$	$2Nw$
17	$Mh \times 2Nw$	$Mh \times 2Nw$ or $2Mh \times Nw$	$(M-1)h + 2Nw$	h or w
18	$2Mh \times 2Nw$	$2Mh \times 2Nw$	$2(M-1)h + 2Nw$ or $2Mh + 2(N-1)w$	$2Mh$ or $2Nw$
19	$2Mh \times 2Nw$	$4Mh \times Nw$ or $Mh \times 4Nw$	$2(M-1)h + 2Nw$ or $2Mh + 2(N-1)w$	$4h$ or $4w$

Concluding remarks

We have developed a number of techniques for optimising array-based designs by component transposition. The major innovations include the description of properties of *tran* that are useful for design optimisation and the collection of theorems for transforming expressions with *tran* and *bend*. Let us summarise the main features of our work and examine its design implications.

Summary

A principal challenge in providing a theory for transforming designs is to find useful design abstractions. We have identified the parametrised building blocks *tran* and *bend* and showed how they correspond to common ways of interconnecting components. These generic building blocks allow concise descriptions of complex wiring patterns.

The use of *tran* and *bend* also enables us to follow a simple equational style of presenting transformations. Several schemes for transposing components of regular array circuits have been discussed. Although the more refined versions require a higher degree of sophistication in interface arrangements, they can still be captured succinctly using the appropriate combinators and primitives. These transformations form a valuable part of a coherent framework for developing and optimising designs (see [2], [3], [4], [6], [7]).

Design implications

The increase in performance brought by transposing components comes mainly from localising interconnections by wiring cells. The elimination of long wires reduces the area and power required for a circuit. We have also indicated how further improvements can be obtained by wiring over the computational blocks, a technique which is eased by the adoption of multi-layer interconnections in some CMOS technologies. An advantage of our approach is that the transformations preserve the regularity of the architecture. This simplifies both the implementation of regular array circuits and the comparison of alternative designs.

Our work, however, is not restricted to custom integrated circuit designs; for instance, transposition can also be applied to vary the array dimensions of programmable cellular structures in order to maximise cell utilisation.

The introduction of bends provides a further opportunity for meeting design constraints, such as satisfying requirements for an array to conform to a given aspect ratio or for the input and output ports to be arranged in a specific manner. In particular, our work provides a possible optimisation for systolic implementations involving long feedback paths [5]. The optimised design can be achieved with little additional effort as the transformations only involve the reflection of components. A more complicated interface may be required, however, although this may not be an issue if the data come in the right format or if the resulting increase in performance justifies the complication.

Further work is needed in three areas. First, a deeper understanding of the scope and the value of transposition requires the application of this method to circuits of greater complexity than those discussed in this paper. Second, it is important to study how transposition fits in with other techniques – such as pipelining – in order to provide an overall strategy for optimising designs. Third, the implementation of transposition should be facilitated by appropriate computer-based tools which are compatible with other circuit design aids and cell libraries.

Acknowledgements. I thank Geraint Jones for contributing the bottom array in Figure 19 and for providing useful suggestions. I also thank Michael Jampel, Mary Sheeran and the referees of this paper for their comments. The support of the UK Alvey Programme (Project ARCH 013), the Croucher Foundation and Rank Xerox (UK) Limited is gratefully acknowledged.

References

[1] G. Jones and M. Sheeran, *Timeless truths about sequential circuits*, in S. K. Tewksbury, B. W. Dickinson and S. C. Schwartz (eds.), *Concurrent computations: algorithms, architectures and technology*, pp. 245–259, Plenum Press, 1988.

[2] G. Jones and M. Sheeran, *Circuit design in Ruby*, in J. Staunstrup (ed.), *Formal methods for VLSI design*, North-Holland, 1990.

[3] W. Luk, *Specifying and developing regular heterogeneous designs*, in L. Claesen (ed.), *Formal VLSI specification and synthesis*, pp. 391–409, North-Holland, 1990.

[4] W. Luk and G. Jones, *The derivation of regular synchronous circuits*, in K. Bromley, S. Y. Kung and E. Swartzlander (eds.), *Proceedings of International Conference on systolic arrays*, pp. 305–314, IEEE Computer Society Press, 1988.

[5] J. Moreno and T. Lang, On *partitioning the Faddeev algorithm*, in K. Bromley, S. Y. Kung and E. Swartzlander (eds.), *Proceedings of International Conference on systolic arrays*, pp. 125–134, IEEE Computer Society Press, 1988.

[6] L. Rossen, *HOL formalisation of Ruby*, Technical Report ID-TR 1989–61, Department of Computer Science, Technical University of Denmark, 1989.

[7] S. Singh, *An application of non-standard interpretation: testability*, in L. Claesen (ed.), *Formal VLSI correctness verification*, pp. 235–244, North-Holland, 1990.