ANALYSING PARAMETRISED DESIGNS
BY NON-STANDARD INTERPRETATION

WAYNE LUK

Programming Research Group,
Oxford University Computing Laboratory,
11 Keble Road, Oxford, England OX1 3QD

Abstract. We examine the use of non-standard interpretation to analyse parametrised circuit descriptions, in particular for array-based architectures. Various metrics are employed to characterise the performance trade-offs of generic designs. The objective is to facilitate the evaluation of such metrics for estimating design quality, so that feasible design alternatives can be compared at an early stage of development.

INTRODUCTION

Constructing digital systems involves two challenges: to develop one or more circuits that perform the desired function, and to analyse design alternatives in order to select the optimal design. Our previous work [3], [4], [5] has described an algebraic framework and the associated computer-based tools for developing array-based architectures. We have shown how such a framework can be used to simplify the parametrisation, structuring and refinement of designs.

This paper builds on this algebraic framework and examines the analysis of parametrised descriptions by non-standard interpretation. The objective is to facilitate the comparison of feasible design alternatives at an early stage of development. Our research centres on techniques for extracting various performance attributes, such as critical path and latency, from a *single* generic design representation. The features of this approach include:

- uniformity. Algebraic representations are succinct and simplify the production of composite designs. They provide a common structure for a range of metrics for estimating design quality;

- modularity. The method is hierarchical and allows blocks of components to be analysed. It is also straightforward to incorporate boundary conditions in the analysis;

- reusability. The metrics characterise performance trade-offs of entire classes of designs, allowing different configurations to be chosen depending on the requirements. Moreover, one may be able to distinguish between implementation-specific, technology-specific and process-specific parameters to facilitate adapting a generic design to different implementation media;

- flexibility. Depending on the information available the designer can use the appropriate procedure to obtain either a rough numerical estimation or a detailed symbolic analysis. This enables designs to be incrementally developed;

- computerised support. The techniques proposed have been implemented in a software package, thereby automating detailed numerical or symbolic calculations. This will also provide a basis for driving a design transformation system [5].

1

THE LANGUAGE AND ITS STANDARD INTERPRETATION

Our approach will be illustrated on a simple functional language, derived from the language μFP [7], for describing hardware. We shall begin by introducing the standard semantics of this language as functions on input data. This interpretation is clearly capable of representing the behaviour of combinational circuits; later we shall adopt the same style to explain the algorithms for extracting useful design properties.

A distinct feature of our language is the use of higher-order functions, or combinators, to capture common patterns of computations as parametrised expressions. For instance, sequential composition is a combinator which corresponds to connecting the output of one component to the input of the other:

$$(F \; ; \; G) \, x \;\; = \;\; G \, (F \, x).$$

Note that we use reverse functional composition (;) to conform with the convention that signals flow from left to right and also to preserve compatibility with relational description of circuits [8]. To avoid confusion function application will be replaced, where appropriate, by sequential composition: instead of writing $Inv \; 1 = 0$, we write $1 \; ; \; Inv = 0$ where 1 and 0 now denote constant functions delivering respectively a bit one and a bit zero. A constant function like 0 or 1 belongs to the set of signal generators, $SigGen$. A symbolic signal generator will usually be denoted by a single lower case letter, so that we can define the squaring operation by $x \; ; \; Square = x^2$. This style of definition follows the form

$$(\text{signal generator}) \; ; \; (\text{circuit expression}) \;\; = \;\; \text{circuit behaviour},$$

and clearly circuit behaviour is itself expressed as a signal generator, since the left-hand side of the equation involves composing a signal generator with a circuit expression, which gives a signal generator. When $(x \; ; \; F) = (x \; ; \; G)$, we shall just write $F = G$.

For components with multiple inputs and outputs we need the combinator *construction*, which corresponds to broadcasting a signal to each component of a composite circuit:

$$x \; ; \; [F, G] \;\; = \;\; [(x \; ; \; F), \, (x \; ; \; G)].$$

Hence an adder can be specified as $[x, y] \; ; \; Add = x + y$. We shall use $[x_i \,|\, 0 \leq i < N]$ to denote $[x_0, x_1, \ldots, x_{N-1}]$.

Another common form of composing circuits involves two components operating independently on a pair of signals:

$$[x, y] \; ; \; (F \parallel G) \;\; = \;\; [(x \; ; \; F), \, (y \; ; \; G)]. \tag{1}$$

It is simple to show that $(A \; ; \; B) \parallel (C \; ; \; D) = (A \parallel C) \; ; \; (B \parallel D)$. Our language contains a set of such algebraic theorems, which equate distinct expressions with identical behaviour, and can be used to transform an obvious but inefficient design to make it more complex but efficient [3].

Given that Id is the identity function such that $(Id \; ; \; x) = (x \; ; \; Id) = x$, $F \parallel Id$ and $Id \parallel G$ will be abbreviated to fst F and snd G. The first and the second element of a pair can be extracted by the projection functions π_1 and π_2,

$$[x, y] \; ; \; \pi_1 \;\; = \;\; x,$$
$$[x, y] \; ; \; \pi_2 \;\; = \;\; y.$$

Their respective inverses, $\pi_1{}^{-1}$ and $\pi_2{}^{-1}$, pair an item with an undefined object, so that $(\pi_1{}^{-1} ; \pi_1) = (\pi_2{}^{-1} ; \pi_2) = Id$.

Next, examples of combinators that capture common cases of spatial iteration will be given. Repeated sequential composition (Figure 1a) is given by

$$
\begin{aligned}
F^0 &= Id, \\
F^{n+1} &= F ; F^n,
\end{aligned}
$$

while repeated parallel composition (Figure 1b) is given by

$$
[x_i \,|\, 0 \le i < N] ; \propto F = [(x_i ; F) \,|\, 0 \le i < N].
$$

($\propto F$ is often pronounced as "map F"). Triangular arrays of latches often arise at the boundaries of pipelined circuits, so we have the $\triangle$ combinator (Figure 1c),

$$
[x_i \,|\, 0 \le i < N] ; \triangle F = [(x_i ; F^i) \,|\, 0 \le i < N].
$$

Continued sums and similar kinds of computations can be achieved by *left reduction* (Figure 1d):

$$
[0, [x_0, x_1, x_2]] ; \mathsf{rdl}\, Add = (((0 + x_0) + x_1) + x_2).
$$

This corresponds to the recursion pattern

$$
[u_0, [x_i \,|\, 0 \le i < N]] ; \mathsf{rdl}\, F = u_N \tag{2}
$$

where $[u_i, x_i] ; F = u_{i+1}$ for $0 \le i < N$.
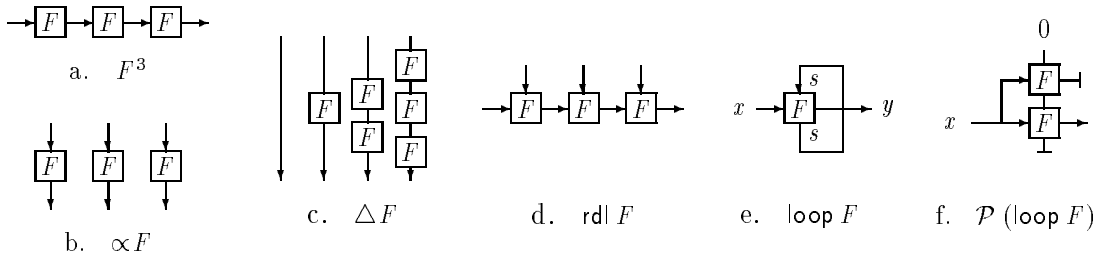

Figure 1 Pictures of some combinators.

NON-STANDARD INTERPRETATION

The purpose of non-standard interpretation is to provide an alternative meaning of representations expressed in a formal language. This technique can be used either to extend the language in order to capture a wider range of entities, or to provide additional information about the properties of an expression.

Devising a non-standard interpretation for a language involves two steps. First, data structures in the standard interpretation are altered to encapsulate the information required for the new interpretation; second, non-standard versions of operations are defined to transform the new data structures to achieve the desired effect. Essentially the method exploits the syntax for operators in the language to provide a scheme for evaluating an

expression to a range of values corresponding to its various properties. The following equation provides a general form for defining a non-standard interpretation for our language:

(non-standard data structure generator) ; $\mathcal{M}$ (circuit expression) $=$ property of circuit,

where $\mathcal{M}$ is a "meaning function" that characterises the non-standard interpretation in terms of the standard interpretation. As before, the property of the circuit is expressed as another generator producing the appropriate non-standard data structure.

The rest of this section will be dedicated to a simple example. One way to accommodate the description of sequential circuits is to adopt the stream data structure which consists of an infinite sequence of data representing values at successive clock cycles [7]. A circuit expression can then be interpreted as a function transforming an input stream to an output stream. A combinational circuit F will perform the same operation in every cycle, so its meaning, $\mathcal{S} F$, is obtained by repeated parallel composition of F, $\propto F$, on the input stream. That is,

$$\mathcal{S}\, F \;\; = \;\; \propto F, \quad \text{if } F \in CombinCirc$$

where $CombinCirc$ is the set of combinational circuits. The meaning of a composite expression can usually be given as a function h of the meaning of its components, so for example

$$\mathcal{S}\,(F\,;\,G) \;\; = \;\; h\,(\mathcal{S}\,F)\,(\mathcal{S}\,G).$$

In this case h is the same as that for the standard interpretation, so the above equation becomes

$$\mathcal{S}\,(F\,;\,G) \;\; = \;\; (\mathcal{S}\,F)\,;\,(\mathcal{S}\,G).$$

To derive the meaning of parallel composition, $\mathcal{S}\,(F \parallel G)$, we need the matrix transposition function $tran$ to manufacture a stream by pairing the corresponding elements from the streams generated by x and y:

$$[[x_t \mid t \in T], [y_t \mid t \in T]]\,;\,tran \;\; = \;\; [[x_t, y_t] \mid t \in T],$$

where T may, for example, be the set of natural numbers. The effect of this operation should be the same as sequentially composing x with the stream version of F, $\mathcal{S}\,F$, and similarly composing y with $\mathcal{S}\,G$, and then pairing the corresponding elements for these two streams by $tran$ to form a stream,

$$
\begin{aligned}
& [x, y]\,;\,tran\,;\,\mathcal{S}\,(F \parallel G) \\
& = \;\; [(x\,;\,\mathcal{S}\,F),\,(y\,;\,\mathcal{S}\,G)]\,;\,tran && \text{(by definition of } \mathcal{S}\,(F \parallel G)) \\
& = \;\; [x, y]\,;\,(\mathcal{S}\,F) \parallel (\mathcal{S}\,G)\,;\,tran && \text{(from Equation 1)} \\
& = \;\; [x, y]\,;\,tran\,;\,tran\,;\,(\mathcal{S}\,F) \parallel (\mathcal{S}\,G)\,;\,tran && \text{(since } tran\,;\,tran = Id).
\end{aligned}
$$

Hence $\mathcal{S}\,(F \parallel G) = tran\,;\,(\mathcal{S}\,F) \parallel (\mathcal{S}\,G)\,;\,tran$ (since x and y are arbitrary). The meaning of other combinators can be obtained in a similar way.

A latch, $\mathcal{D}$, can be modelled by appending a "don't care" value, generated by $\bot$, to a stream of signals:

$$x\,;\,\mathcal{S}\,\mathcal{D} \;\; = \;\; [\bot, x]\,;\,apl$$

where apl, short for "append left", is given by $[a, [x_0, x_1, x_2, \ldots]]\,;\,apl \;=\; [a, x_0, x_1, x_2, \ldots]$. We shall also have $\mathcal{D}^{-1}$, a fictitious element which can predict its next input, such that

4

$\mathcal{D} \; ; \; \mathcal{D}^{-1} = Id$. Although $\mathcal{D}^{-1}$ is not implementable, it can be useful in reasoning about a design.

A loop construct can also be defined (Figure 1e),

$$x \; ; \; \mathcal{S} \, (\mathsf{loop} \; F) \quad = \quad y$$

where $[s, y] = [x, s] \; ; \; tran \; ; \; \mathcal{S} \, F \; ; \; tran$, and s is a stream containing the "state" of F. To avoid asynchronous loops, F must have at least one latch on its feedback path.

Jones and Sheeran [1] provide further discussions on giving a stream semantics to an algebraic language.

CIRCUIT METRICS

The preceding section illustrates the use of non-standard interpretation to cover a wider range of designs – namely to describe sequential circuits. We shall now elucidate how non-standard interpretation can be used to compute circuit metrics. As before the two steps are (a) to determine an appropriate data structure that facilitates the representation and manipulation of information required for a given property, and (b) to formulate a meaning function that allows the designated property of a composite design to be deduced from the properties of its components. The metrics introduced in this section include cell count, latency, and critical path evaluation.

Cell count

The data structure for evaluating the number of a given cell in a composite design consists of two components. The first component contains information for instantiating a parametrised representation, such as deciding the number of F in $\propto F$. The simplest representation for this component is to adopt the same data structure used in the standard interpretation, although the actual numerical or symbolic values of atomic expressions are not needed. The second component in the data structure is a counter to accumulate the number of the given cell.

Let us consider the meaning function $\mathcal{N}_F \, G$ for counting the number of F's in a given expression G. The first rule states that, given instantiation information x and counter n, if $F = G$ then increment the counter n and evaluate $(x \; ; \; G)$ according to the standard interpretation to propagate the instantiation information; otherwise if G does not contain combinators then preserve the value of n and just evaluate $(x \; ; \; G)$.

$$
\begin{aligned}
[x, n] \; ; \; \mathcal{N}_F \, G \quad &= \quad [(x \; ; \; G), \; n + 1] \quad \text{if } F = G, \\
&= \quad [(x \; ; \; G), \; n] \qquad \text{if } F \neq G \text{ and } G \text{ does not contain combinators.}
\end{aligned}
$$

Since the standard interpretation is originally intended for describing combinational circuits, sequential constants such as $\mathcal{D}$ are not defined; it is, however, evident that we should take $\mathcal{D}$ as Id in this non-standard interpretation to propagate the instantiation information.

To count the number of F in $(G \; ; \; H)$, we simply accumulate the number of F in G and in H one after the other,

$$\mathcal{N}_F \, (G \; ; \; H) \quad = \quad (\mathcal{N}_F \, G) \; ; \; (\mathcal{N}_F \, H).$$

5

For parallel composition, the number of F in G and H can be accumulated independently and the results are then combined,

$$[[x, y], n] \; ; \; \mathcal{N}_F \; (G \parallel H) \quad = \quad [[u, v], n + p + q]$$

where $[x, 0] \; ; \; (\mathcal{N}_F \; G) = [u, p]$ and $[y, 0] \; ; \; (\mathcal{N}_F \; H) = [v, q]$. The meaning of other combinators can be derived in the same manner: for instance, $\mathcal{N}_F \, (G^N) = (\mathcal{N}_F \, G)^N$ and, given that $x = [x_i \,|\, 0 \leq i < N]$,

$$[x, n] \; ; \; \mathcal{N}_F \, (\propto G) \quad = \quad \left[(x \; ; \; \propto G), \, n + \sum_{0 \leq i < N} ([x_i, 0] \; ; \; \mathcal{N}_F \, G \; ; \; \pi_2) \right],$$

$$[x, n] \; ; \; \mathcal{N}_F \, (\triangle G) \quad = \quad \left[(x \; ; \; \triangle G), \, n + \sum_{0 \leq i < N} ([x_i, 0] \; ; \; \mathcal{N}_F \, G^i \; ; \; \pi_2) \right].$$

These definitions can be used to derive results like

$$[x, n] \; ; \; \mathcal{N}_G \, (\triangle G) \; ; \; \pi_2 \quad = \quad n + N(N-1)/2.$$

As for the loop construct, we disregard the feedback path and use π_1^{-1} and π_2 to match the types in the first component of the data structure: $\mathcal{N}_F \, (\textsf{loop } G) = \textsf{fst } \pi_1^{-1} \; ; \; \mathcal{N}_F \; G \; ; \; \textsf{fst } \pi_2$.

Of course, one must remember to initialise the counter to zero before the evaluation. A simple example showing how our method works is given below.

$$\begin{aligned} [x, 0] \; ; \; \mathcal{N}_F \, (G \; ; \; F \parallel F) \quad &= \quad [[y, y], 0] \; ; \; \mathcal{N}_F \, (F \parallel F) \quad (\text{given } x \; ; \; G = [y, y]) \\ &= \quad [[(y \; ; \; F), (y \; ; \; F)], 2] \quad (\text{given } [y, 0] \; ; \; \mathcal{N}_F \, F = [(y \; ; \; F), 1]). \end{aligned}$$

It is obvious that the cell count metric can be used to give a lower bound of the area and power required for a composite cell, if the area and power of each of its components are known.

Latency evaluation

To extract the latency of a design, we compute the maximum number of latches for all paths from input to output. The data structure is the same as that for the standard interpretation except that numerical expressions represent counters for accumulating the number of latches. Constant numerical functions are used to generate boundary conditions, such as the latency of a component whose output is connected to the input of the circuit under evaluation. However, this representation of boundary conditions may necessitate changing those signal generators embedded in a circuit expression for this non-standard interpretation.

Given that $max \; x$ returns either the maximum of x or x itself depending on whether x is composite (e.g. $max \; [2, [1, 3]] = 3$ and $max \; 4 = 4$), the meaning function $\mathcal{L}$ for expressions not containing combinators can be summarised as follows:

$$\begin{aligned} x \; ; \; \mathcal{L} \; F \quad &= \quad F \qquad \text{if } F \in SigGen, \\ &= \quad x + 1 \quad \text{if } F = \mathcal{D}, \\ &= \quad x - 1 \quad \text{if } F = \mathcal{D}^{-1}, \\ &= \quad max \; x \quad \text{otherwise.} \end{aligned}$$

The meaning of combinators is the same as that for the standard interpretation, so for instance $\mathcal{L} \, (G \; ; \; H) = (\mathcal{L} \; G) \; ; \; (\mathcal{L} \; H)$ and $\mathcal{L} \, (G \parallel H) = (\mathcal{L} \; G) \parallel (\mathcal{L} \; H)$. The exception is the loop construct, the meaning of which is not defined in this interpretation since there is no universal model for initialising the latch on the feedback path.

Critical path evaluation

There are two components in our data structure for estimating the critical path of a design. The first component has the same structure as that for standard interpretation, and is used to accumulate the combinational delay of the current path. The second component records the maximum combinational delay for paths evaluated so far.

Consider first the meaning function $\mathcal{P}\ F$ for evaluating the critical path of a non-composite circuit F. For each latch in the circuit the first component of the data structure is cleared and its previous value is compared with that of the second component so that the greater of the two will be stored in the second component. As discussed in the preceding section, signal generators are used to capture boundary conditions – in this case the critical path of components whose outputs are connected to the inputs of the circuit under evaluation. For each combinational cell in the circuit with a composite input, the delay of the cell is added to the maximum of the input delays to give the new value of the current delay path; the second component of the data structure remains unchanged. The critical path delay corresponds to the maximum of all components at the output.

So given that δF denotes the combinational delay of F, the rules in the preceding paragraph can be expressed as:

$$
\begin{aligned}
[x, n]\ ;\ \mathcal{P}\ F\ &=\ [0,\ max\ [x, n]] & \text{if } F = \mathcal{D},\\
&=\ [F,\ max\ [x, n]] & \text{if } F \in SigGen,\\
&=\ [(\delta F + max\ x),\ n] & \text{otherwise.}
\end{aligned}
$$

As far as projection functions are concerned, we need to check whether the eliminated output is linked to the critical path; so for instance

$$
[[x, y], n]\ ;\ \mathcal{P}\,\pi_2\ =\ [y,\ max\ [x, n]].
$$

The meaning of sequential and parallel composition is reasonably obvious and is given below:

$$
\begin{aligned}
\mathcal{P}\,(G\ ;\ H)\ &=\ (\mathcal{P}\ G)\ ;\ (\mathcal{P}\ H),\\
[[x, y], n]\ ;\ \mathcal{P}\,(G\ \|\ H)\ &=\ [[u, v],\ max\ [p, q]]
\end{aligned}
$$

where $[x, n]\ ;\ (\mathcal{P}\ G) = [u, p]$ and $[y, n]\ ;\ (\mathcal{P}\ H) = [v, q]$. Similarly, given that $x = [x_i \mid 0 \le i < N]$, Equation 2 is altered to become

$$
[[u_0, x], c_0]\ ;\ \mathcal{P}\,(\mathsf{rdl}\ F)\ =\ [u_N, c_N] \tag{3}
$$

where $[[u_i, x_i], c_i]\ ;\ \mathcal{P}\ F = [u_{i+1}, c_{i+1}]$ for $0 \le i < N$.

The meaning of other combinators can be developed in a similar way – except for the loop construct, whose meaning is derived by "unwinding" the iteration once (Figure 1f):

$$
[x, m]\ ;\ \mathcal{P}\,(\mathsf{loop}\ F)\ =\ [[x, s], n]\ ;\ \mathcal{P}\,(F\ ;\ \pi_2)
$$

where $[x, m]\ ;\ \mathcal{P}\,(\pi_1^{-1}\ ;\ F\ ;\ \pi_1) = [s, n]$. According to this model, if the feedback path is not latched properly then the loop construct will contribute an incremental delay equal to the sum of the incremental delays at the vertical and the horizontal output of F. The absence of such asynchronous loops can be assured by checking that the incremental delay at the bottom truncated output in Figure 1f does not exceed the open-loop delay for the corresponding vertical output of F.

DERIVING PARAMETRISED EXPRESSIONS FOR METRICS

The application of this approach will be illustrated by an adaptive convolver design. Given N coefficients $w_1 \ldots w_N$, the circuit is to calculate $\sum_i w_{i,t-N} \times x_{t-i}$, with $1 \le i \le N$. Our architecture consists of a linear array of M identical clusters of cells, with each cluster itself consisting of a linear array of K latched multiply-accumulators (Figure 2, with latches represented by heavy dots). Given that the total number of cells is fixed such that $K \times M = N$, by varying K one can obtain a range of designs with different trade-offs in speed, latency and the number of latches as shown in Table 1. It will be shown how to compute the formulae in this table using the techniques outlined in the preceding section.
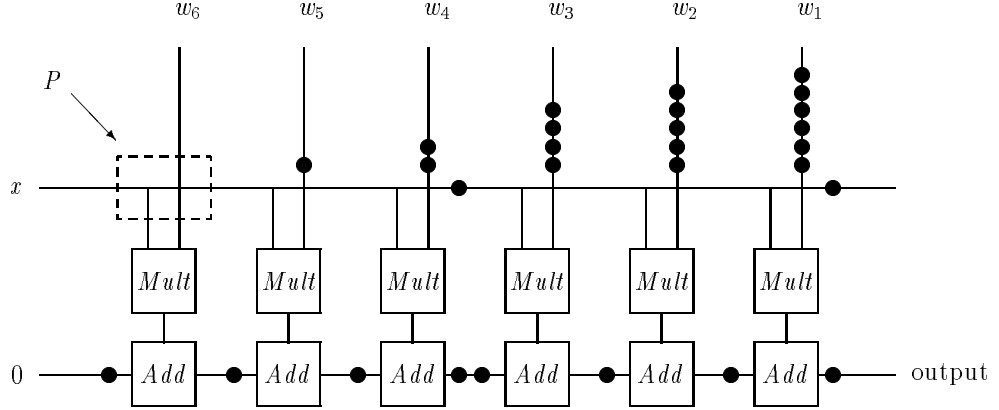

Figure 2 adaptive convolver design ($K = 3$, $M = 2$, $N = 6$).

Table 1 metrics for parametrised adaptive convolver design.

Minimum clock period	Latency (cycles)	Number of skewing latches	Number of latches in array
$(K-1)\delta P + \delta M + \delta A$	$\dfrac{N(K+1)}{K}$	$\dfrac{N(N+NK-2K)}{2K}$	$\dfrac{N(K+2)}{K}$

$\delta M, \delta A$: the combinational delay of cell *Mult*, *Add*,
δP: the propagation delay of broadcasting horizontally across cell P.

We first capture this design as a parametrised representation, Cv, in our language:

$$
\begin{aligned}
Cv &= \text{snd } InSkew \text{ ; rdl } (CvCells \text{ ; } \mathcal{D} \parallel \mathcal{D}) \text{ ; } \pi_1, \\
InSkew &= \triangle\mathcal{D} \text{ ; } Group_M \text{ ; } \triangle(\propto\!\mathcal{D}), \\
[x_i \,|\, 0 \le i < KM] \text{ ; } Group_M &= [[x_{i,j} | 0 \le j < K] \,|\, 0 \le i < M], \\
CvCells &= \text{rdl (fst (fst } \mathcal{D}) \text{ ; } CvCell), \\
[[y, x], w] \text{ ; } CvCell &= [y + (x \times w), x].
\end{aligned}
$$

Cell count. Let us first check the number of skewing latches in the design. This step involves sequentially composing $\mathcal{N}_{\mathcal{D}}$ *InSkew* with an appropriate input,

$$[[x_i \,|\, 0 \le i < KM], 0] \,;\, \mathcal{N}_{\mathcal{D}} \,(\triangle \mathcal{D} \,;\, Group_M \,;\, \triangle(\propto\mathcal{D}))$$

$$= \left[[x_i \,|\, 0 \le i < KM], \left(\sum_{0 \le i < KM} i \right) \right] \,;\, \mathcal{N}_{\mathcal{D}} \,(Group_M \,;\, \triangle(\propto\mathcal{D}))$$

$$= \left[[[x_{i,j} \,|\, 0 \le j < K] \,|\, 0 \le i < M], \left(\sum_{0 \le i < KM} i \right) \right] \,;\, \mathcal{N}_{\mathcal{D}} \,(\triangle(\propto\mathcal{D}))$$

$$= \left[[[x_{i,j} \,|\, 0 \le j < K] \,|\, 0 \le i < M], \left(\sum_{0 \le i < KM} i \right) + \left(K \sum_{0 \le i < M} i \right) \right].$$

Now the sum of the two continued sums is equal to $KM(KM + M - 2)/2 = N(N + NK - 2K)/2K$, which is the result given in Table 1.

Latency. Given that $w = [w_i | 0 \le i < N]$ and $w_{i+1} \ge w_i$ for $0 \le i < N - 1$,

$$[[y, 0], w] \,;\, \mathcal{L} \, CvCells \;=\; [[y, 0], w] \,;\, \mathcal{L} \,(\mathsf{rdl} \,(\mathsf{fst} \,(\mathsf{fst} \, \mathcal{D}) \,;\, CvCell))$$
$$= \; [max \,[y + N, \, last \, w], 0]$$

where $last \; w = w_{N-1}$. Let $\vec{0}_L = [0 \,|\, 0 \le i < L]$. To calculate $\mathcal{L} \, InSkew$, we sequentially composed it with $\vec{0}_{KM}$,

$$\vec{0}_{KM} \,;\, \mathcal{L} \,(\triangle\mathcal{D} \,;\, Group_M \,;\, \triangle(\propto\mathcal{D})) \;=\; [i \,|\, 0 \le i < KM] \,;\, \mathcal{L} \,(Group_M \,;\, \triangle(\propto\mathcal{D}))$$
$$= \; [[Ki + j \,|\, 0 \le j < K] \,|\, 0 \le i < M] \,;\, \mathcal{L} \,(\triangle(\propto\mathcal{D}))$$
$$= \; [[(K + 1)i + j \,|\, 0 \le j < K] \,|\, 0 \le i < M]$$
$$= \; w', \quad \text{say}.$$

Now consider

$$[[0, 0], \vec{0}_{KM}] \,;\, \mathcal{L} \, Cv \;=\; [[0, 0], w'] \,;\, \mathcal{L} \,(\mathsf{rdl} \,(CvCells \,;\, \mathcal{D} \parallel \mathcal{D}) \,;\, \pi_1)$$
$$= \; [[0, 0], w'] \,;\, \mathsf{rdl} \,(\mathcal{L} \,(CvCells \,;\, \mathcal{D} \parallel \mathcal{D})) \,;\, \pi_1$$
$$= \; max \,[M(K + 1), 1 + last \, w']$$
$$= \; M(K + 1)$$

since $last \; w' = (K + 1)(M - 1) + K - 1 = M(K + 1) - 2$. By definition $M = N/K$, so the latency of Cv is $N(K + 1)/K$ as given in Table 1.

Critical path. It is assumed that the skewing circuit *InSkew* does not contribute to the critical path. Hence

$$[[[y, x], 0], c] \,;\, \mathcal{P} \, CvCell \;=\; [[max \,[x + \delta M, y] + \delta A, \; x + \delta P], c].$$

Let $CvCell' = \mathsf{fst} \,(\mathsf{fst} \, \mathcal{D}) \,;\, CvCell$. Recall that $[x, n] \,;\, \mathcal{P} \, \mathcal{D} \;=\; [0, max \,[x, n]]$. Hence

$$[[[y, x], 0], c] \,;\, \mathcal{P} \, CvCell' \;=\; [[x + \delta M + \delta A, \; x + \delta P], \; max \,[c, y]]. \tag{4}$$

Since there is no $\mathcal{D}^{-1}$ in $CvCells$, $\mathcal{P}\,(\mathsf{rdl}\,(CvCells\,;\,\mathcal{D}\,\|\,\mathcal{D})) = \mathcal{P}\,CvCells = \mathcal{P}\,(\mathsf{rdl}\,CvCell')$. We shall show by induction that

$$[[[0,0],\vec{0}_K],0]\,;\,\mathcal{P}\,(\mathsf{rdl}\,CvCell') \quad = \quad [[(K-1)\delta P + \delta M + \delta A,\,K\delta P],c_K] \qquad (5)$$

where $c_K = 0$ if $K = 1$, otherwise $c_K = (K-2)\delta P + \delta M + \delta A$.

The base case is straightforward: $[[a,[b]],c]\,;\,\mathcal{P}\,(\mathsf{rdl}\,F) = [[a,b],c]\,;\,\mathcal{P}\,F$, so we just use Equation 4 to check that Equation 5 is valid when $K = 1$. Now consider the induction case:

$[[[0,0],\vec{0}_{K+1}],0]\,;\,\mathcal{P}\,(\mathsf{rdl}\,CvCell')$
$\quad = \quad [[[(K-1)\delta P + \delta M + \delta A,\,K\delta P],0],c_K]\,;\,\mathcal{P}\,CvCell' \qquad$ (Equation 3)
$\quad = \quad [[K\delta P + \delta M + \delta A,\,(K+1)\delta P],\,max\,[c_K,(K-1)\delta P + \delta M + \delta A]] \quad$ (Equation 4)
$\quad = \quad [[K\delta P + \delta M + \delta A,\,(K+1)\delta P],(K-1)\delta P + \delta M + \delta A] \qquad$ (def. of c_K)
$\quad = \quad [[(K'-1)\delta P + \delta M + \delta A,\,K'\delta P],(K'-2)\delta P + \delta M + \delta A] \qquad$ ($K' = K+1$)

which corresponds to the hypothesis (Equation 5) when K' is substituted for K. If we assume that $\delta M + \delta A \geq \delta P$, then the formula for critical path in Table 1, $(K-1)\delta P + \delta M + \delta A$, is correct.

COMPUTERISED SUPPORT

The circuit metrics described in this paper have been incorporated into a prototype computer-based tool for regular array design [5]. They provided a numerical characterisation of a composite circuit given the numerical characterisation of its components. This section illustrates how the design system can be used in analysing the convolver architecture presented earlier.

First of all, we define the input to Cv and instantiate M,

```
> ws = [w6,w5,w4,w3,w2,w1]
> in = [[0,x],ws]
> M  = 2
```

The symbolic simulator can then be used to check the correctness of our design:

```
> sim in ; Cv

  0: ?
  1: ?
  2: ?
  3: ?
  4: ?
  5: ?
  6: ?
  7: ?
  8: ((((((x_1 * w6_1) + (x_2 * w5_1)) + (x_3 * w4_1)) + (x_4 * w3_1))
+ (x_5 * w2_1)) + (x_6 * w1_1))
  9: ((((((x_2 * w6_2) + (x_3 * w5_2)) + (x_4 * w4_2)) + (x_5 * w3_2))
+ (x_6 * w2_2)) + (x_7 * w1_2))
 10: ((((((x_3 * w6_3) + (x_4 * w5_3)) + (x_5 * w4_3)) + (x_6 * w3_3))
+ (x_7 * w2_3)) + (x_8 * w1_3))
 11: ((((((x_4 * w6_4) + (x_5 * w5_4)) + (x_6 * w4_4)) + (x_7 * w3_4))
+ (x_8 * w2_4)) + (x_9 * w1_4))
```

Cell count. We can count the number of *CvCells* and $\mathcal{D}$ in this configuration,

```
> count CvCells  in ; Cv
2
```

```
> count D  in ; Cv
28
```

Composite expressions such as $\mathcal{D} \parallel \mathcal{D}$ can also be counted,

```
> count D||D  in ; Cv
2
```

Latency. Now let us check the latency of our design,

```
> latency  in ; Cv
8: 0 -> D -> Add -> D -> Add -> D -> Add -> D -> D -> Add -> D -> Add -> D
 -> Add -> D.
```

Notice that in addition to providing a numerical value, the system also displays the path with the maximum number of latches. This facility should be helpful if the designer wants to alter the design to reduce its latency.

If it happens that the input x comes from a circuit with a latency of 5, then we can incorporate this boundary condition into the evaluation of latency:

```
> latency  [[0,5],ws] ; Cv
12: 5 -> Mult -> Add -> D -> Add -> D -> Add -> D -> D -> Add -> D -> Add
 -> D -> Add -> D.
```

We can also add an arbitrary amount of latency to a component without altering its definition. For instance, if we assign a latency of 3 to the multiplier, then we get

```
> latency  in ; Cv
10: w5 -> Mult(3) -> Add -> D -> Add -> D -> Add -> D -> D -> Add -> D
 -> Add -> D -> Add -> D.
```

Of course, this result will no longer agree with the simulation of the circuit, which remains unchanged.

Critical path. Let $\delta P = 1$, $\delta A = 3$, and $\delta M = 6$. With these values, the critical path of Cv can be computed,

```
> crpath  in ; Cv
11: x -> P(1) -> P(1) -> P -> Mult(6) -> Add(3).
```

This shows that the critical path consists of two broadcasting delays. If we change M to 6 to obtain a fully pipelined design, then we get

```
> crpath  in ; Cv
9: x -> P -> Mult(6) -> Add(3).
```

On the other hand we may be satisfied with a non-pipelined design. This can be produced by instantiating M to 1 to get

```
> crpath  in ; Cv
14: x -> P(1) -> P(1) -> P(1) -> P(1) -> P(1) -> P -> Mult(6) -> Add(3).
```

In reality different size adders, with different delays, may be used in each *CvCell* to cope with word length growth. This situation can be modelled by using a heterogeneous version of rdl [2].

11

CONCLUSION

The purpose of this paper is to illustrate how non-standard interpretation can be used in analysing performance attributes of a parametrised design representation. We have illustrated that the proposed techniques can produce both parametrised expressions and numerical values for a variety of circuit metrics. The rest of this section reviews related work and suggests a number of extensions to our approach.

Non-standard interpretation has been employed to deduce various properties of a design description; for instance Sheeran [8] has shown how directional information can be obtained from a relational circuit expression, and Singh [9] has also presented interpretations for testability analysis and for deductive fault simulation. A similar technique, called abstract interpretation, has been used in strictness analysis for functional programming languages [6].

Providing multiple interpretations for a design description helps to eliminate possible inconsistencies that may arise if the circuit is represented in more than one way; it is a step towards an environment for developing parametrised hardware representations. Future work will include extending the class of circuits amenable to this treatment, improving the efficiency of data structures and algorithms that can be used, investigating how performance attributes (such as area and power estimates) can be captured more accurately, and linking our tools to synthesis systems, circuit design tools and cell libraries.

References

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

[2] W. Luk, "Specifying and developing regular heterogeneous designs," in: L.J.M. Claesen (ed.), *Formal VLSI specification and synthesis*, p. 391, North-Holland, 1990.

[3] 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*, p. 305, IEEE Computer Society Press, 1988.

[4] W. Luk and G. Jones, "From specification to parametrised architectures," in: G. Milne (ed.), *The fusion of hardware design and verification*, p. 267, North-Holland, 1988.

[5] W. Luk, G. Jones and M. Sheeran, "Computer-based tools for regular array design," in: J. McCanny, J. McWhirter and E. Swartzlander (eds.), *Systolic array processors*, p. 589, Prentice Hall, 1989.

[6] S. Peyton Jones, "The implementation of functional programming languages," Prentice Hall International, 1987.

[7] M. Sheeran, "μFP – a language for VLSI design," D.Phil. Thesis, Programming Research Group, Oxford University, November 1983.

[8] M. Sheeran, "Describing and reasoning about circuits using relations," in: K. McEvoy and J. Tucker (eds.), *Theoretical foundations of VLSI design*, Cambridge University Press, 1990.

[9] S. Singh, "An application of non-standard interpretation: testability," in: L.J.M. Claesen (ed.), *Formal VLSI correctness verification*, North-Holland, 1990.