VSDF : Synchronous Data Flow for VLSI

Alain Kerihuel, Roderick Mcconnell, Sanjay Rajopadhye

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

VSDF: Synchronous Data Flow for VLSI

Alain Kerihuel, Roderick McConnell, Sanjay Rajopadhye

N° 2237

Juin 1994

PROGRAMME 1

ISSN 0249-6399

VSDF: Synchronous Data Flow for VLSI*

Alain Kerihuel, Roderick McConnell, Sanjay Rajopadhye **

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet API

Rapport de recherche n ° 2237 — Juin 1994 — 31 pages

Abstract: This article describes a modified Synchronous Data Flow model which is suitable for modeling synchronous VLSI circuits. The target model takes advantage of the synchronous nature of the operations to eliminate buffering between circuits where possible. We introduce a temporal notation, and define functions relevant to constructing a system.

Key-words: digital signal processing, synchronous data flow, VLSI simulation

(Résumé : tsvp)

*This work was partially funded by the French Coordinated Research Program ANM of the French Ministry of Research and Space, by the Esprit BRA project No 6632 NANA-2, by the Doctoral-candidate Network for System and Machine Architecture of the DRED, and by NSF grant MIPS 910852.

**{Alain.Kerihuel}{Roderick.McConnell}{Sanjay.Rajopadhye}@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 84 71 71

Des graphes de flots de données synchrones pour le VLSI

Résumé : Cet article présente des graphes de flots de données synchrones adaptés pour la conception d'application dédiées en VLSI. Une notation temporelle ainsi que les fonctions nécessaires à son utilisation pour la modélisation des événements dans un système périodique sont données. Des contraintes sont alors ajoutées aux graphes en vue de la détermination d'horloges périodiques et régulières capables d'effectuer les tranferts de données nécessaires aux différentes opérations. Quelques exemples d'utilisation de la notation sont également proposés.

Mots-clé : conception de systèmes spécialisés, VLSI, traitement digital de signaux

1 Introduction

The Synchronous Data Flow (SDF) graph [9] [10] offers an elegant representation for certain real-time Digital Signal Processing (DSP) applications. If, in order to implement a certain application, high throughput will also be required, specialized VLSI circuits are often targeted. Typically, these circuits are synchronous, and are linked by dedicated communications paths - the wires of a PCB or Multi-Chip Module, for example. Although there are clear similarities between the SDF graph and a system of synchronous circuits, there is also a large gap in terms of modeling the implementation details. To help bridge this gap, we propose a component model which can represent a target application both at an SDF level and at the level of synchronous circuits.

There exist several systems which compile Synchronous Data Flow graphs for multiprocessor implementations, such as PTOLEMY [1] or Comdisco's "Signal Processing WorkStation"[3]. These systems suffer from the disadvantage for hardware implementation that blocks of data are buffered between operations. Our approach also differs from existing systems for developing periodic circuits, such as CATHEDRAL [2], PHIDEO [18], Mentor Graphic's "DSP Station" [15] or GAUT [13], which use knowledge of the internal operations to perform allocation and optimization. We emphasize that our model is targeted at system-level RTL design, and not at individual component design.

In this article we present a mathematical notation and a simulation model positioned between Synchronous Data Flow graphs, and a synchronous system built of synchronous circuits. As in SDF, the number of input and output values per operation per period are known. In addition, the moments at which they exist relative to a common clock are specified, in order to properly model synchronous circuits [16]. The temporal specification permits the designer to better define input and output functions, and in particular to model pipelined operators at the Register Transfer level. In most cases this eliminates buffers between pipelined operators. The model is targeted at synchronous VLSI circuits, with the intention that a designer can easily insert an existing synchronous circuit given its timing diagram.

This report is organized in a way that introduces VSDF in ever increasing detail. We begin with a brief review of Synchronous Data Flow. We then motivate our choice of a slightly different model for a VLSI hardware target. We introduce our notation for temporal expressions, and then discuss operations possible with these temporal expressions. We also define necessary conditions for functional hardware, which are an extension to the necessary conditions given in [9], [10]. We complete our model with initialization conditions and delay considerations. We finish by applying VSDF to synchronous circuits and system design. Examples, in general taken from video coding, are given throughout the article to illustrate key points.

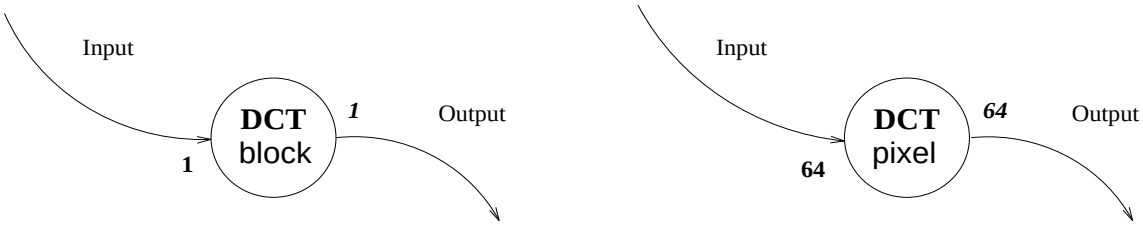

Figure 1: SDF node at block and pixel level

2 Synchronous Data Flow

In the model proposed by Lee and Messerschmitt [9] [10], a signal processing application is described in terms of *static* transfers and operations. The number of *consume* elements needed for an operation to commence, and the number of *produce* elements resulting from an operation, are fixed at design time. Furthermore, the nature of these produce and consume elements is not fixed - their granularity can vary from single words or pixels, to blocks of arbitrary size, representing for example entire images [8]. The consume elements are buffered at the input to an operator until an input threshold is reached; the operator is then *fired*, or started. In the abstract model, the results are then available immediately, i.e. the produce elements are produced with no delay.

The number of consume elements needed to launch an operation, depends on the operation and on the granularity of the elements. For example, a two-dimensional discrete cosine transform (DCT) which requires a block of 8x8 pixels as input, may be represented in coarse-grain data flow as consuming 1 block to start processing. Or, at a more fine grain, the DCT may be represented as consuming 64 pixels before starting. The first representation assumes that all the input values will arrive at the same instant, while the second representation implies a buffering of input values until all have been received. In Figure 1 an SDF node for a DCT is given at a block and at a pixel level.

2.1 Synchronous Data Flow Graphs

An SDF graph is a collection of connected nodes. Each node represents a specific operation, the complexity of which depends on the level of description of the associated functional block. The operator will operate on signals which contain an infinite stream of values. Operations must also be independent, i.e. all interactions between operators must be specified by means of static transfers. This yields a graph in which all operations are nodes, and all transfers are arcs between nodes. The signal-flow graphs presented by Lee and Messerschmitt are called *synchronous* to express the fact that an operator is invoked when a pre-specified number of new values are available at all inputs. We emphasize that the threshold for each input, as well as the number of inputs consumed and the number of outputs produced, is fixed in advance as shown in Figure 1.

2.2 Consistency

When developing a synchronous data flow application, it is important that the SDF graph has a bounded cyclic schedule. This is equivalent to the constraint that each node can be invoked a bounded integer number of times, after which input and output buffers will return to a previous state of "fullness". For example, if we have two nodes with a communication dependency between them, we can pose the question whether it is possible to call each node

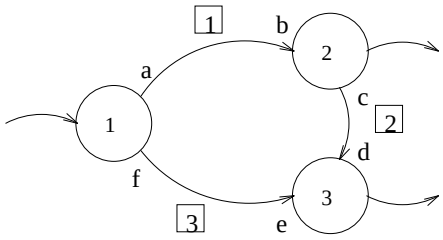

Figure 2: A generic three node dataflow graph

$$\begin{pmatrix} a & -b & 0 \\ 0 & c & -d \\ f & 0 & -e \end{pmatrix}$$

Figure 3: Topology Matrix associated with Figure 2

a finite number of times, allowing a complete exchange of the specified number of data, without accumulation or starvation on the part of either node. The property of existence of a bounded cyclic schedule is called *consistency* in [7]. Lee & Messerschmitt provide conditions for consistency, based on a topology matrix which represents the SDF graph.

Validity conditions for consistency

For a given SDF graph, we would like to find a periodic schedule $\vec{\mathbf{r}}$. The elements r_i represent the number of times that a node i is "*fired*" during one system period. If such a schedule exits, we can solve a system such as the one given in Figure 2. The number of elements produced and consumed for each link is represented by the expressions:

$$\left\{ \begin{array}{lcl} a * r_1 & = & b * r_2 \\ f * r_1 & = & e * r_3 \\ c * r_2 & = & d * r_3 \end{array} \right\}$$

We can then verify the equality between the number of elements produced and the number of elements consumed:

$$\left\{ \begin{array}{lcll} a * r_1 & - & b * r_2 & = 0 \\ f * r_1 & - & e * r_3 & = 0 \\ c * r_2 & - & d * r_3 & = 0 \end{array} \right\}$$

We can associate a *topology matrix* Γ as shown in Figure 3 with the graph given in Figure 2. The number of elements produced or consumed by each "firing" of a node. A column is associated with each node, and a row with each edge. The edges have been labeled in an arbitrary fashion, as shown in the little boxes in the Figure 2. The $(i, j)^{th}$ entry defines the number of data exchanged by node i on link j. If a node j consumes data on edge i the value is negative; otherwise it is positive, or equal to zero if edge j is not connected to node i.

Lee and Messerschmitt show that a necessary condition for the existence of a valid solution schedule, is that $rank(\Gamma) = s - 1$, where matrix Γ is the matrix of outputs and

inputs between the different nodes of one graph, and s its number of nodes [9] [10]. In fact, if a solution $\vec{\mathbf{r}}$ exits, it will be a null vector for the transfer matrix Γ:

$$\Gamma\vec{\mathbf{r}} = \vec{0}.$$

Likewise any multiple of this null vector will also be a solution, yielding a family of solutions. These solutions correspond to a single or multiple iterations of the system period. Intuitively, one can see that there must exist a single family of solution vectors, because the system must be able to run for one or more periods, while at the same time, the absence of conditional communications implies that there is only one possible execution scheme. The system itself may be a subsystem of a larger graph, in which case the minimum period of the larger graph will be a multiple of the periods of all subgraphs. This can be seen intuitively from the fact that if Γ_i and Γ_j are two sub-matricies representing subsystems of Γ, if they share a link, they will share a period which must satisfy both null vectors.

In order to render the calculations more manageable, we will generally consider only the least of the solution vectors of the system (or subsystem) under consideration.

2.3 The Balance Equation

The SDF approach allows one to prove that a periodic system can operate indefinitely with neither starvation nor buffer overflow, as demonstrated by means of the Topology Matrix. This same constraint is expressed in [7] [8] as satisfying the *balance equations*. If output i is connected to input j, the arc which connects them must satisfy:

$$r_i O_i = r_j I_j$$

where r_i and r_j are *repetition* counts, i.e. the number of times that node i or node j will be repeated during one system period, O_i is the number of output elements *produced* per repetition of $node_i$, and I_j the number of input elements *consumed* per repetition of $node_j$. The vector $\vec{\mathbf{r}}$ of r_i's gives the number of repetitions of each node during one system period. The balance equation specifies an equivalence of the count of values, i.e. that the same number of values are produced and consumed during one system period. This in turn guarantees that the system can continue for any number of system periods with neither starvation nor buffer overflow.

We emphasize that the balance equations are merely an alternate notation for the Topology Matrix Γ. Each equation is a row of Γ with all the zero elements removed. We will in general choose the balance equation format for our expressions, as it is usually easier to understand.

2.4 Limitations of SDF

The SDF representation has limitations, the most fundamental being that it can only be applied to a limited sub-class of signal processing applications, namely those where the data transfers can be statically specified. In addition, breaking the complete application into sub-problems, as represented by the nodes of an SDF graph, may limit the possibilities for global optimization. Approaches such as that of PHIDEO [18], however, allow automated optimization of systems which fit the Synchronous Data Flow model, particularly in terms of memory allocation [19].

Given a suitable application, there is another limitation which to us appears relevant: SDF as presented by Lee & Messerschmitt, and implemented in the SDF domain of the design system PTOLEMY [1], is oriented towards an implementation at least partially in software (for example, hardware/software co-design). This is due to the block nature of the transfers

and operations, which correspond closely to an input or output buffer and a subroutine call, respectively. While buffers and subroutines are well-suited for implementation using one or more programmable DSP's, they are poorly suited to a dedicated VLSI hardware implementation.

It is clear that there are points of similarity between an SDF graph and an RTL model of a hardware-based signal processing system. In an application such as motion video compression in real time, complex operations such as a discrete cosine transform (DCT) are often performed by a dedicated circuit, which in the abstract takes a block of data at the input and produces a block of data at the output. The principal difference for the designer is in the timing of data transfers. The simple count of elements consumed in SDF must be replaced by a specification for a stream of data, including word by word timing, to reflect the operation of the hardware at a Register Transfer level. It is basically a bandwidth and storage problem - there aren't enough pins on a chip to pass blocks of data at a time, nor does one want to store data unnecessarily.

3 SDF for VLSI

As indicated in the preceeding section, the SDF model is a powerful tool for applications which meet its rather strict constraints. We propose that it can be a powerful tool for modeling dedicated VLSI systems as well.

The number of consume elements needed to launch an operation, depends on the operation and on the granularity of the elements. For example, a two-dimensional discrete cosine transform (DCT) which requires a block of 8x8 pixels as input, may be represented in coarse-grain data flow as consuming 1 block to start processing. Or, at a more fine grain, the DCT may be represented as consuming 64 pixels before starting. The first representation assures that all the input will arrive at the same instant, while the second representation implies a buffering of input elements until all have been received. In Figure 1 an SDF node for a DCT is given at a block and at a pixel level.

At the pixel level, there is a clear similarity between the SDF DCT operator, and a dedicated VLSI DCT processor such as the SGS-Thomson IMS-A121 8x8 DCT chip [4], or the LSI Logic L64735 8x8 DCT chip [11]. There are 64 input pixels consumed per operation, and 64 coefficients produced. The principal differences are in the *timing* of the input and output events, and in the *latency* of the operation.

3.1 VLSI timing

The VLSI circuits have timing requirements, unlike the software operators used in SDF. Even if we consider only the Register Transfer Level (RTL) constraints, VLSI circuits require that the inputs be synchronized with the system clock, and that they arrive one value per clock cycle, without interruption, until a complete block has been received. In the SDF model, there are no constraints whatsoever on the timing of the input events. Likewise, the output values are available immediately when the operation has finished. The VLSI circuits, on the other hand, will produce the outputs after a certain latency, typically one per clock cycle, until a complete block has been emitted.

We choose to express RTL timing as the "instant" at which an input or output occurs. We take advantage of the fact that our system is synchronous to formalize the information of a datasheet. If, in the specification, an input must be present at the clock rising edge, we specify that the input event and the clock edge must occur at the same "instant", and likwise for the output. Given this description of inputs and outputs, we develop formal expressions for correct operation in the following subsection.

3.2 Constraints for Hardware

As presented in [7], an SDF graph can be statically analyzed to determine if it is consistent. We would like, in addition, to assure that all transfers occur at the "right" moment, i.e. that the destination accepts a transfer at the same moment that the source generates the transfer. This is in accordance with our hypothesis that a transfer can be considered instantaneous if it terminates in one clock cycle.

Thus we start with the *balance equations*, and add the constraint that, if output i is connected to input j, a value is produced and consumed at the same instant. In section 8 we will also demand that the circuit have a constant latency, corresponding to our vision of a component that repeats exactly the same operation with a constant period. In the following section we introduce a notation for the moments, or instants in time, at which a signal exists. In the circuit sense, this corresponds to the instants at which an output is valid, or an input is latched. With this notation, we will formalize our constraint, and introduce a method for static analysis.

4 A Temporal Notation

In order to formalize the constraints on the instants introduced above, we first introduce a notation for temporal expressions. We begin with a notation for describing an individual periodic event, then we extend this notation to represent a set of periodic events with a common period and starting instant. This section finishes with the definition of a canonical form which eliminates possible redundancy in our temporal expressions.

4.1 A Periodic Event

Our temporal expression for a periodic event defines a mapping from integers to integers. The resulting values denote the *temporal index* (or simply the *time*) at which events occur. A temporal expression is denoted by (θ, ϕ, h), where θ is the period, ϕ the phase, and h the initial delay (akin to a start-up delay). It specifies a clock function:

$$(\theta, \phi, h) : I\!N \longmapsto I\!N$$

$$t \longmapsto \theta t + \phi + h$$

$$\theta, h \ \in I\!N$$

$$\phi \ \in \{0 \ldots \theta - 1\}$$

We limit ϕ to be in the range $0 \leq \phi < \theta$ in order to keep the periodic event ϕ within the period θ. The presence of both ϕ and h is redundant in the above function, but we will later use h exclusively to describe an initial delay, and we will introduce a canonical form to resolve any possible ambiguity. If t is taken as time, $t \in I\!N$, then the affine expression (θ, ϕ, h) specifies an infinite series.

The most rapid periodic event τ which can be described using this notation is that which occurs every clock cycle:

$$(\theta, \phi, h) = (1, 0, 0)$$

$$(1, 0, 0) : t \mapsto t$$

$$\tau = \theta t + \phi + h = t.$$

Hence t may be viewed as the clock expression that specifies the fundamental clock of a system. Likewise an event τ which occurs every fourth event of a period of 64 (such as the fourth input pixel to our DCT) might be expressed as:

$$(\theta, \phi, h) = (64, 3, 0)$$

$$(64, 3, 0) : t \mapsto 64t + 3$$

$$\tau = \theta t + \phi + h = 64t + 3.$$

If there are k *distinct* events ϕ during a period θ, these events will be expressed as a union of individual periodic events. For events to be distinct, $\forall i \neq j \ \phi_i \neq \phi_j$. Periodic events may be combined into a periodic union if they have the same period and different phases, i.e. they denote distinct time instants:

$$\theta t + \phi_0 \cup \theta t + \phi_1 \cup \cdots \cup \theta t + \phi_{k-1} = \bigcup_{0 \leq i < k} \theta t + \phi_i.$$

4.2 A Set of Periodic Events

We will often work with a union of events ϕ which represent the phases of all events of a period. Therefore we introduce Φ to represent the set of k ϕ's of a period. We note:

$$\Phi = \{\phi_0, \phi_1, \ldots \phi_{k-1}\}$$

with

$$|\Phi| = k.$$

The affine expressions specified by a non-negative integer θ, and a set of non-negative integers Φ such that $\forall \phi_i \in \Phi, \phi_i < \theta$ denotes a function over the integers:

$$(\theta, \Phi, h) : t \longmapsto \{\theta t + \phi_0 + h, \theta t + \phi_1 + h, \ldots, \theta t + \phi_{k-1} + h\}$$

$$\theta, h \ \in I\!N$$

$$\phi \ \in \{0 \ldots \theta - 1\}$$

To summarize, we use an affine expression (θ, ϕ, h) for a single periodic event, and (θ, Φ, h) when there are a set of periodic events. The value θ determines the period, ϕ or Φ the phase(s), and h the initial delay.

An example: a simple periodic function

An example of a simple periodic expression might be $(P, \{0, \ldots, P-1\}, 0)$, with period P and $k = P$ events in a period, that is:

$$\forall i \in 0 \ldots (P-1), \ \theta = P, \phi_i = i, h = 0$$

The input to the DCT at a pixel level, shown in Figure 1, would have the parameter $P = 64$. The clock expression would be:

$$(\theta, \Phi, h) = (64, \{0, 1, \ldots 63\}, 0)$$

$$(\theta, \Phi, h) : t \mapsto \{64t + k | 0 \leq k < 64\}.$$

Since this also represents an event at every clock tick, it is equivalent to the expression $(\theta, \phi, h) = (1, 0, 0)$:

$$t \mapsto \{64t + k | 0 \leq k < 64\} \equiv t \mapsto t.$$

This example shows that there are multiple expressions which represent the same set of clock events. Hence it is necessary to develop certain rules for manipulating clock expressions. As a prelude to the introduction of these rules, we propose the following cannonical form, which eliminates ambiguity in the notation.

4.3 A Canonical Form

We eliminate redundancy in temporal expressions by differentiating between ϕ or Φ, and h. We will use h exclusively to establish the delay before the periodic behavior commences. We define the beginning of a period by the initial event in a period. In other words, we set the minimum of the $\phi_i = 0$, and use h to specify the initial delay. For temporal expressions, this yields:

Definition 1 (Canonical Form) *An expression is in canonical form* **iff**

$$(\theta, \phi, h) \Rightarrow \phi = 0$$

$$(\theta, \Phi, h) \Rightarrow \phi_{MIN} = 0.$$

Proposition 1 *An expression can be put into canonical form by subtracting ϕ or ϕ_{MIN} from the phase, and adding it to the initial delay:*

$$(\theta, \phi, h) \mapsto (\theta, 0, h + \phi)$$

$$(\theta, \Phi, h) \mapsto (\theta, \Phi^{'}, h + \phi_{MIN})$$
$$\Phi^{'} = \{\phi_i^{'} | \phi_i^{'} = \phi_i - \phi_{MIN}\}.$$

Henceforth, we will use expressions in a canonical form to avoid any possible confusion.

5 Operations on Temporal Expressions

In order to manipulate temporal expressions, we use three operations and a condition of equivalence. The operations are *scaling*, *dilation* and *delay*. Scaling corresponds to extending the period of observation of a temporal expression, while dilation corresponds to a change-of-basis for a given clock expression and delay corresponds to adding a constant to the initial delay h. We also introduce a condition for equivalence between two temporal expressions, in the case where they describe the same instants. We detail these three operations and the condition of equivalence below.

5.1 Repeating the Period: Scaling

Because our systems are periodic, the events of a period will repeat, with the same phase, in following periods. If a clock expression (θ, Φ, h) describes the events of a system during one base period, it is also possible to view the events as periodic with the period any multiple of the base period. We have already encountered an example where $(64, \{0, 1, \ldots 63\}, 0)$ and $(1, 0, 0)$ represented the same instants. We now introduce a more formal specification for the generation of a new periodic system with longer periods. One new period will now contain r repetitions of the base period. This is done by calculating the phases of events of r base

periods which make up one new period, given the k phases of one base period, $\{\phi_0 \ldots \phi_{k-1}\}$. Likewise, if the base period contains a set of k phases Φ, we define the scaling $r \circ (\theta, \Phi, h)$ as the union of the scalings of the k individual phases ϕ_i. The initial delay h does not change:

Definition 2 (Scaling) *scaling of a unitary clock expression by a* repetition *factor r yields*

$$r \circ (\theta, \phi, h) \equiv (r\theta, \Phi^{'}, h)$$

where

$$\Phi^{'} = \{\phi, \theta + \phi, 2\theta + \phi, \ldots, (r-1)\theta + \phi\}$$
$$|\Phi^{'}| = r$$

Scaling $r \circ (\theta, \Phi, h)$ is the union of the scalings of the k individual phases $\phi_i \in \Phi$

$$r \circ (\theta, \Phi, h) \equiv (r\theta, \Phi^{'}, h)$$

where

$$\begin{aligned} \Phi^{'} &= \bigcup_{0 \le i < k} \{\phi_i, \theta + \phi_i, \ldots, (r-1)\theta + \phi_i\} \\ |\Phi^{'}| &= r|\Phi|. \end{aligned}$$

The instants of the scaled period can be calculated from t with:

Proposition 2

$$r \circ (\theta, \phi, h) = \bigcup_{0 \le l < r} (r\theta)t + (l\theta + \phi) + h.$$

$$r \circ (\theta, \Phi, h) = \bigcup_{0 \le i < rk} (r\theta)t + \phi_i + h$$

where

$$\phi_i = \left\lfloor \frac{i}{k} \right\rfloor \theta + \phi_{i \bmod k}$$

Proof: *We note that the Proposition is just a substitution of a union for the enumeration of the phases $\Phi^{'}$. In the case of a union of events Φ, we distribute the $\circ$ operator across the union of events Φ:*

$$\begin{aligned} r \circ (\theta, \Phi, h) &= \bigcup_{0 \le i < k} r \circ (\theta, \phi_i, h) \\ &= \bigcup_{0 \le i < k} (r\theta, \{\phi_i + h, \theta + \phi_i + h, \ldots, (r-1)\theta + \phi_i + h\}) \\ &= \bigcup_{0 \le i < k, 0 \le l < r} (r\theta)t + (l\theta + \phi_i) + h. \end{aligned}$$

□

Corollary

$$r_1 \circ r_2 \circ (\theta, \Phi, h) = (r_1 r_2) \circ (\theta, \Phi, h)$$

Examples

We can now demonstrate the equivalence of the clock expressions $(1,0,0)$ and $(64,\{0\ldots63\},0)$, the second being merely a scaling of the first by 64 (i.e. 64 repetitions of the period):

$$r \circ (\theta,\phi,h) \equiv (r\theta,\Phi^{'},h)$$
$$64 \circ (1,0,0) \equiv (64,\Phi^{'},0)$$
$$\Phi^{'} = \bigcup_{0\leq k<64} k.$$

We might then continue to scale this new expression $(64,\{0\ldots63\},0)$ by 2, to get a period which is twice as long:

$$2 \circ (64, \bigcup_{0\leq k<64} k, 0) = (128, \bigcup_{0\leq k<128} k, 0).$$

5.2 Equivalence of Clock Expressions

Two clock expressions are equivalent if they describe the same set of instants. Clearly two expressions are equivalent if they are identical:

Definition 3 (Equality)
two temporal expressions $(\theta_1,\phi_1,h_1),(\theta_2,\phi_2,h_2)$ *are equal, denoted by* $(\theta_1,\phi_1,h_1) = (\theta_2,\phi_2,h_2)$ **iff***:*

$$\theta_1 = \theta_2 \wedge \phi_1 + h_1 = \phi_2 + h_2.$$

two temporal expressions $(\theta_1,\Phi_1,h_1),(\theta_2,\Phi_2,h_2)$ *are equal, denoted by* $(\theta_1,\Phi_1,h_1) = (\theta_2,\Phi_2,h_2)$ **iff***:*

$$\theta_1 = \theta_2 \wedge \{\phi_i + h_1 | \phi_i \in \Phi_1\} = \{\phi_i + h_2 | \phi_i \in \Phi_2\}.$$

Two expressions may also describe the same set of instants, but over a different period. We consider two clock expressions to be *equivalent* if they can each be scaled in such a way that their events all "match", i.e. they both share a common scaled period, and all of the scaled phases (events) are the same. We remark that two expressions are equivalent when they define the same instants, even if this happens over different periods. As we illustrated earlier, the expressions (64,{0,..., 63},0) and (1,0,0) are equivalent. In general, expressions are equivalent if there exists a common multiple of their periods, over which the two expressions always define the same instants. More formally, two clock expressions are equivalent if there exist scalings r_1, r_2 such that the two scaled expressions are equal:

Definition 4 (Equivalence)

$$(\theta_1,\Phi_1,h_1) \equiv (\theta_2,\Phi_2,h_2) \Leftrightarrow$$
$$r_1 \circ (\theta_1,\Phi_1,h_1) = r_2 \circ (\theta_2,\Phi_2,h_2)$$

Proposition 3 *Two canonical-form clock expressions can be equivalent only if they share the same initial delay h:*

$$(\theta_1,\Phi_1,h_1) \equiv (\theta_2,\Phi_2,h_2) \Rightarrow h_1 = h_2$$

Proof: *The scaling operation* $\circ$ *does not change the initial delay h:*

$$\begin{aligned} h_1 \neq h_2 &\Rightarrow \nexists r_1, r_2 \;\; r_1 \circ (\theta_1,\Phi_1,h_1) = r_2 \circ (\theta_2,\Phi_2,h_2) \\ &\Rightarrow (\theta_1,\Phi_1,h_1) \not\equiv (\theta_2,\Phi_2,h_2) \end{aligned}$$

□

Proposition 4 *The smallest scalings to determine equivalence can be determined from the least common multiple of the two periods:*

$$\begin{aligned}(\theta_1, \Phi_1, h_1) \equiv (\theta_2, \Phi_2, h_2) \Leftrightarrow \\ \frac{\theta}{\theta_1} \circ (\theta_1, \Phi_1, h_1) = \frac{\theta}{\theta_2} \circ (\theta_2, \Phi_2, h_2), \\ \theta = \mathbf{lcm}\{\theta_1, \theta_2\}.\end{aligned}$$

Proof: *We first show that this is a necessary condition, and then use contradiction to show that it yields the smallest scaling. We know that*

$$r_1\theta_1 = r_2\theta_2$$

Let

$$r_1 = \frac{\theta}{\theta_1}$$

$$r_2 = \frac{\theta}{\theta_2},$$

and indeed

$$\frac{\theta}{\theta_1}\theta_1 = \frac{\theta}{\theta_2}\theta_2.$$

Assume that there exists a smaller $\theta^{'}, \theta = k\theta^{'}$ *such that*

$$\frac{\theta^{'}}{\theta_1} \circ (\theta_1, \Phi_1, h_1) = \frac{\theta^{'}}{\theta_2} \circ (\theta_2, \Phi_2, h_2).$$

$$r_1\theta_1 = r_2\theta_2 = k\theta^{'}.$$

We write

$$r_1 = k\frac{\theta^{'}}{\theta_1}$$

$$\frac{\theta^{'}}{\theta_1} = k\frac{\theta^{'}}{\theta_1}$$

$$k = 1.$$

□

5.3 Changing the Timing Resolution: Dilation

In a synchronous system, all events are calibrated by a common clock. Nevertheless, there will be events which are more frequent than others, necessitating a clock with a "finer" resolution, i.e. a faster clock, or a higher system clock frequency. This becomes important when connecting nodes which have different needs in terms of the calibration of the clock. The node with the "finest" resolution will determine the resolution necessary for the system clock, i.e. the system clock frequency.

In order to match a given clock expression to an arbitrarily "fine" system clock, we introduce *dilation*. Dilation corresponds to a change-of-basis for a given clock expression. If one regards the period θ as the unit of measure, the dilated period θ will contain more "ticks" of the fundamental clock $(1, 0, 0)$. We dilate an expression by multiplying by a constant:

Definition 5 (Dilation)
An expression (θ, ϕ, h) *dilated by a constant* $d \in \mathbb{N}$ *yields:*

$$d.(\theta, \phi, h) \longmapsto (d\theta, d\phi, dh).$$

An expression (θ, Φ, h) *dilated by a constant* $d \in \mathbb{N}$ *yields:*

$$d.(\theta, \Phi, h) \longmapsto (d\theta, \{d\phi_i\}, dh).$$

Use of Dilation

Using this definition we can show, for example, that dilating a scaled expression and scaling a dilated expression yield the same results:

$$r \circ (d.(\theta, \phi, h)) = d.(r \circ (\theta, \phi, h))$$

Proof:

$$\begin{aligned} r \circ (d.(\theta, \phi, h)) &= r \circ (d\theta, d\phi, dh) \\ &= (rd\theta, \{d\phi, \theta + d\phi, \ldots, (r-1)\theta + d\phi\}, dh) \\ &= d.(r\theta, \{\phi, \theta + \phi, \ldots, (r-1)\theta + \phi\}, h) \\ &= d.(r \circ (\theta, \phi, h)) \end{aligned}$$

□

An example: dilating an expression

In a motion video coding system, the luminance information may arrive four times as fast as the chrominance blue information. If a DCT is used for the chrominance blue, but the system clock is the luminance pixel clock, the clock expression for the chrominance blue DCT would be the basic expression as given in section 4.2, dilated by a factor of four:

$$\begin{aligned} (\theta_{Cb}, \Phi_{Cb}, h_{Cb}) &= 4.(\theta_{DCT}, \Phi_{DCT}, h_{DCT}) \\ &= 4.(64, \{0, 1, \ldots, 63\}, 0) \\ &= (256, \{0, 4, \ldots, 252\}, 0). \end{aligned}$$

Atomic Clock Expressions

Just as with scaling, there are families of clock expressions that can be generated via dilation. We consider these families to be multiples of *atomic* clock expressions; an atomic clock expression, in other words, is one that cannot be generated by dilating and/or scaling another clock expression:

Definition 6 (Atomic Clock Expressions)
An expression (θ, ϕ, h) *is* atomic **iff***:*

$$\begin{aligned} &(\theta, \phi, h) \Rightarrow \nexists d \in \mathbb{N}, \\ &\quad (\theta, \phi, h) = d.(\theta', \phi', h'). \end{aligned}$$

An expression (θ, Φ, h) *is* atomic **iff***:*

$$\begin{aligned} &(\theta, \Phi, h) \Rightarrow \nexists d, r \in \mathbb{N}, \\ &\quad (\theta, \Phi, h) = d.r \circ (\theta', \Phi', h'). \end{aligned}$$

Proposition 5 *Any expression (θ, ϕ, h) can be expressed as a dilation of the fundamental clock $(1, 0, 0)$ plus an initial delay δ.*
Proof: *We first convert the expression to canonical form:*

$$(\theta, \phi, h) = (\theta, 0, \phi + h).$$

If we dilate the fundamental clock by theta, and add an initial delay $\delta = \phi + h$, we get:

$$\theta.(1, 0, 0) + (\phi + h) = (\theta, 0, \phi + h).$$

□

5.4 Increasing the Initial Delay

Our notation is intended to model real circuits, which require time to calculate their results. There is necessarily a latency between the instant when the data enters a circuit, and the instant when the results are available at the output. For a system composed of multiple circuits, the latency as the data passes through the system will be *cumulative*, i.e. a circuit which takes its input from the output of another circuit, must wait until the output is available, which will in turn dictate the instant at which its own output will be available. We model this latency by adding a constant delay to a temporal expression.

All events are calibrated by a common clock, and all instants when a transfer occurs are identified by the temporal expressions. Additional latency, therefore, is simply a constant which defines the number of additional system clock cycles during which no events occur. This latency added to a temporal expression corresponds to adding a (positive) constant to the initial delay h:

Definition 7 (Delay)
An expression (θ, ϕ, h) delayed by a constant $\delta \in \mathbb{N}$ yields:

$$(\theta, \phi, h) + \delta \longmapsto (\theta, \phi, h + \delta).$$

An expression (θ, Φ, h) delayed by a constant $\delta \in \mathbb{N}$ yields:

$$(\theta, \Phi, h) + \delta \longmapsto (\theta, \Phi, h + \delta).$$

An example: delaying an expression

In a motion video coding system, an 8x8 discrete cosine transform (DCT) may be performed by a circuit which implements one-dimensional transforms on all rows and then on all columns. If the rows and then the columns are transformed sequentially, this operation will have a latency of at least 128 cycles. Assuming the the circuit performs the transform in 128 cycles, the temporal expression for the output will be delayed by $\delta = 128$ with respect to the input:

$$\begin{aligned} h^{out} &= h^{in} + 128 \\ (\theta_{DCT}, \Phi^{in}, h^{in}) &= (64, \{0, 1, \ldots, 63\}, 0) \\ (\theta_{DCT}, \Phi^{out}, h^{out}) &= (64, \{0, 1, \ldots, 63\}, 128). \end{aligned}$$

We remark that as a result of our restrictions on the application domain, a given circuit, or node in the graph, will always have a *constant latency.*

In the following section, we will show how these operations and the notion of equivalence can be used to strengthen the constraints of SDF, in order to correctly model the synchronization needed for VLSI circuits.

6 The Augmented Balance Equation

Synchronous Data Flow requires that each "actor", or node in the processing graph Γ, be synchronous and periodic, with one "firing" per base period of that node. This "firing" represents one execution of the operation of that node, including the consumption of a fixed number of input elements, and the production of a fixed number of output elements. As presented in [7], an SDF graph can be statically analyzed to determine if it is consistent, and to find a solution to the Topology Matrix. We would like, in addition, to ensure that all transfers occur at the "right" moment, i.e. that the destination accepts a transfer at the same moment that the source generates the transfer. We start with the *balance equations*, and add the constraint that, if output i is connected to input j, a value is produced and consumed at the same instant. We call our new constraint the *augmented balance equations*; in section 7 we will show how to find a solution for these new constraints.

6.1 Application to SDF

In order to model our constraints, we associate with a node a constant θ_i, corresponding to the basic periodic operation of that node. During one repetition of the period θ_i, there will be I_i inputs and O_i outputs. Each individual input or output instant corresponds to a certain phase ϕ. For the moment we present each node as if it has only one input and one output; this is only to avoid a clutter of indices in our notation. The periodic set of instants of any one input of a node is assigned the temporal expression:

$$InputInstants = (\theta_i, \Phi^{in}, h^{in})$$

with

$$|\Phi^{in}| = I_i.$$

Likewise, the periodic set of outputs of a node is assigned the temporal expression:

$$OutputInstants = (\theta_i, \Phi^{out}, h^{out})$$

with

$$|\Phi^{out}| = O_i.$$

6.2 Additional Constraints

We now apply our notation to augment the balance equation, so that it will include temporal constraints, and in particular that every *produce instant* is matched by a *consume instant*. Let $Output_i$ and $Input_j$ be connected. Node i has a period θ_i, during which O_i distinct emissions occur; node j has a period θ_j, during which I_j distinct receptions occur (as in Figure 4). We express the set of output phases of the O_i emissions as Φ_{ij}^{out}, and the set of input phases of the I_j receptions as Φ_{ij}^{in}. We wish to augment the balance equation to express the fact that the emissions from i should occur at the same instants as the receptions at j, i.e. that their timing expressions must be equivalent. We write an *augmented balance equation* as follows:

Definition 8 (Augmented Balance Equation)

$$r_i O_i = r_j I_j \tag{1}$$

$$(\theta_i, \Phi_{ij}^{out}, h_{ij}^{out}) \equiv (\theta_j, \Phi_{ij}^{in}, h_{ij}^{in}). \tag{2}$$

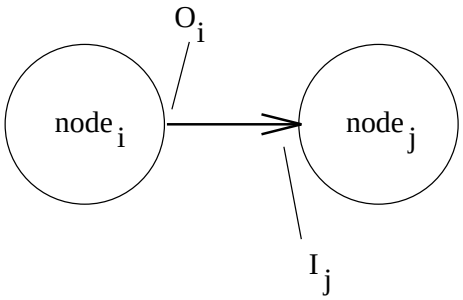

Figure 4: Two SDF Nodes

In other words, not only are the number of items *produced* and *consumed* equal, but also the *produce instants* and the *consume instants* are identical.

Proposition 6 *If r_i and r_j are the solutions to the balance equation determined from the application graph Γ [9] [10], then the augmented balance equation 2 is a sufficient condition for the balance equation 1:*

$$r_i \circ (\theta_i, \Phi_{ij}^{out}, h_{ij}^{out}) = r_j \circ (\theta_j, \Phi_{ij}^{in}, h_{ij}^{in}) \Rightarrow r_i O_i = r_j I_j$$

Proof:

$$|\Phi_{ij}^{out}| = O_i$$
$$|\Phi_{ij}^{in}| = I_j$$
$$r_i |\Phi_{ij}^{in}| = r_j |\Phi_{ij}^{out}|$$

□

The techniques for finding the repetition vector $\vec{\mathbf{r}}$ of all r_i's for an application graph Γ are given in [9] [10]; we do not repeat them here. Our interest will be to verify that the θ's and ϕ's satisfy the augmented balance equation, and to determine appropriate h's.

An example: a DCT

If we take a source node which emits a value every clock cycle, and connect it to a DCT which consumes 64 values in 64 cycles, we have:

$$O_{src} = 1, I_{DCT} = 64$$
$$r_{src} O_{src} = 64 * 1 = 1 * 64 = r_{DCT} I_{DCT}$$
$$(\theta_{src}, \phi^{out}, h^{out}) = (1, 0, 0)$$
$$(\theta_{DCT}, \Phi^{in}, h^{in}) = (64, \bigcup_{0 \le k < 64} k, 0)$$
$$r_{src} \circ (\theta_{src}, \phi^{out}, h^{out}) = 64 \circ (1, 0, 0)$$
$$r_{DCT} \circ (\theta_{DCT}, \Phi^{in}, h^{in}) = 1 \circ (64, \bigcup_{0 \le k < 64} k, 0)$$
$$(64, \bigcup_{0 \le k < 64} k, 0) = (64, \bigcup_{0 \le k < 64} k, 0).$$

We note that this is once again the scaling by 64 demonstrated in the previous section.

An example: the System Clock

In order to build a synchronous system, we desire that all nodes share a common system clock θ_{sys}. We know that each node will repeat r_i times during one system period defined by the repetition vector $\vec{\mathbf{r}}$. Therefore, the period of each node must be such that when repeated r_i times, one arrives at the system period θ_{sys}. If the system period is known:

$$\theta_i = \frac{\theta_{sys}}{r_i}.$$

7 Determining Timing Functions for a System

The timing of the example in section 6.2 is a trivial case, where both input and output nodes have an event every clock cycle. In the general case, it is not as easy to determine the appropriate values for θ and Φ. We shall now determine specific θ's and Φ's, which are amenable to a hardware implementation yet still satisfy the augmented balance equation. We will use dilation of the fundamental clock $(1, 0, 0)$, as well as scaling, to derive appropriate timing functions.

The clock expressions which we will derive use dilation of the fundamental clock $(1, 0, 0)$, which yields a sequence of events which are *regular*, i.e. the events of a period happen at *regular intervals* during the period. This desirable for two reasons. The first is that synchronous circuits tend to produce values at regular multiples of their synchronous clock: a DCT typically calculates one result per pixel clock cycle, while a motion estimation processor may calculate one vector per 128 pixels. The second is that if the values are produced and consumed at regular intervals, it is easy to establish whether the *augmented balance equation* is satisfied. We also note that dilation can easily be implemented for circuits using a counter together with the system clock.

7.1 Regular Clock Expressions

A *regular* clock expression is one in which the time between events, expressed as a function of the fundamental clock t, is constant. In other words, a regular clock expression is a scaling of a single *atomic* clock expression:

Definition 9 (Regular Clock Expressions) *A temporal expression* (θ, Φ, h) *is* regular *if the set of phases* Φ *is of the form:*

$$\Phi = \{\frac{\theta}{|\Phi|} \times k, 0 \leq k < |\Phi|\}$$

$$(\theta, \Phi, h) = |\Phi| \circ (\theta/|\Phi|, 0, h).$$

Proposition 7 *A regular expression* (θ, Φ, h) *in canonical form can always be expressed as a scaling and dilation of the fundamental clock* $(1, 0, 0)$*, plus an initial delay.*
Proof: *We remark that by proposition 5, the expression* $(\theta/|\Phi|, 0, h)$ *can be generated from the fundamental clock. We need only scale this expression by:*

$$r = |\Phi|$$

to obtain the desired form.
□

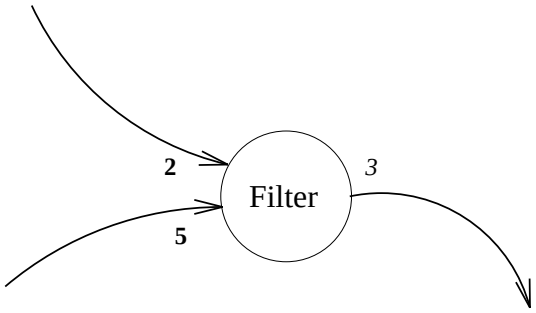

Figure 5: An SDF node

An example: a regular clock function

An example of a regular clock function is the DCT function given earlier, $(\theta, \Phi, h) = (64, 0 \ldots 63, 0)$. In this case,

$$\phi_{i+1} - \phi_i = 1.$$

7.2 A *Regular Clock* for a Node

As a prelude to determining regular clock expressions for a complete system, we show how a regular clock may be determined for an arbitrary node in isolation. Our approach will be to determine the minimum node period which is sufficiently fine, or "fast", to be able to represent all input and output events using dilation and scaling of the fundamental clock.

We use Definition 2 to define the clock expression for each input or output. In order to do this, we first find a node period θ_i^* which is the least common multiple of all inputs I_i and outputs O_j, in order to then divide this by the number of input or output events per period:

$$\theta_i^* = \mathbf{lcm}\{\forall l, m, I_i, O_j\}.$$

For each input I_i, we have the clock expression:

$$\begin{aligned}(\theta_i^*, \Phi_i^{in}, h_i^{in}) &= I_i \circ (\frac{\theta_i^*}{I_i}, 0, h_i^{in}) \\ &= I_i \circ (\frac{\theta_i^*}{I_i}.(1,0,0) + h_i^{in}).\end{aligned}$$

For each output O_j, we have the clock expression:

$$\begin{aligned}(\theta_i^*, \Phi_j^{out}, h_j^{out}) &= O_j \circ (\frac{\theta_i^*}{O_j}, 0, h_j^{out}) \\ &= O_j \circ (\frac{\theta_i^*}{O_j}.(1,0,0) + h_j^{out}).\end{aligned}$$

An example: a regular clock for a node

Consider a filter (figure 5) which takes two values on input I_1, and five values on input I_2, yielding three values on output O_1. We find θ_i^* as the least common multiple of the count of input and output events:

$$\theta_i^* = \mathbf{lcm}\{\forall m, I_i, O_j\}$$

$$\begin{aligned} &= \mathbf{lcm}\{2,3,5\} \\ &= 30 \end{aligned}$$

We use the regular function resulting from dilation by θ/k and scaling by k to generate the input and output event sets:

$$(\theta_i^*, \Phi_i^{in}, h_i^{in}) = I_i \circ \frac{\theta_i^*}{I_i}.(1,0,0) + h_i^{in}.$$

For the first input, we have:

$$2 \circ (15.(1,0,0)) + h_1^{in} = (30, \{0,15\}, h_1^{in}),$$

for the second input:

$$5 \circ (6.(1,0,0)) + h_2^{in} = (30, \{0,6,12,18,24\}, h_2^{in}),$$

and for the output:

$$3 \circ (10.(1,0,0)) + h_1^{out}) = (30, \{0,10,20\}, h_1^{out}).$$

7.3 Finding a common θ_{sys}

In the example given in Section 6.2, the system period can be easily determined from the common fundamental clock (the sample clock) and the number of samples in a block. In the general case it may be necessary to determine the minimum system period θ_{sys} capable of representing all transfer events using dilation and scaling of the fundamental clock $(1,0,0)$, i.e. a system clock with enough precision to represent all events. When working with regular clocking functions, this is related to the practical problem of determining a minimum system clock frequency such that all events can be calculated by *dividing* the system clock by an integer value. This constraint in practical terms implies that all node clocks can be calculated from the common system clock using only simple counters, i.e. without the use of clock multipliers, etc.

Above we indicated how one might determine a node period θ^* for a node considered in isolation. In the case of a system, it is no longer sufficient to consider each node in isolation; one must consider the effect of the whole system, as the smallest time interval t will be the interval necessary to express the fastest event in the whole system. In order to find a minimum system period θ_{sys} for regular clock expressions, we need to find a system period such that all events can occur at regular intervals during the system period. In other words, we need a θ_{sys} which is the *least common multiple* of all transfer events:

$$\begin{aligned} \forall i,\ & \frac{\theta_{sys}}{r_i O_i}, \frac{\theta_{sys}}{r_i I_i} \in I\!N \Rightarrow \\ & \theta_{sys} = \mathbf{lcm}\{r_i O_i, r_i I_i, \forall i \in I\!N\}. \end{aligned}$$

We note that it is sufficient to express this condition either over all outputs, or over all inputs, because for any link in a system which satisfies the balance equations, there will be a matching input and output:

$$\begin{aligned} \forall j, \exists i, r_j I_j &= r_i O_i \\ \theta_{sys} &= \mathbf{lcm}\{r_i O_i, \forall i \in I\!N\}. \end{aligned}$$

The resulting θ_{sys} will necessarily be as fine as or finer than the finest θ_i^* calculated for each node in isolation.

7.4 Application to the Balance Equation

If we connect inputs and outputs which have regular clock expressions, the augmented balance equations will always be satisfied, given that the following three conditions are met:

- the balance equations are satisfied;
- the nodes share a common system period θ_{sys} calculated from the transfer matrix Γ and $\vec{\mathbf{r}}$ (as given in in Section 7.3);
- the transfer events start at the same time, i.e. the start-up delays h^{in} and h^{out} are equal when the expressions are in canonical form.

We express this more formally:

Proposition 8 *If, for any edge$_{ij}$ in the SDF graph, $(\theta_i, \Phi_{ij}^{out}, h_{ij}^{out})$ and $(\theta_j, \Phi_{ij}^{in}, h_{ij}^{in})$ are regular canonical expressions, and*

$$r_i O_i = r_j I_j$$
$$h_{ij}^{out} = h_{ij}^{in},$$
$$r_i \theta_i = r_j \theta_j = \theta_{sys}$$

then

$$(\theta_i, \Phi_{ij}^{out}, h_{ij}^{out}) \equiv (\theta_j, \Phi_{ij}^{in}, h_{ij}^{in}).$$

Proof: *We will show that*

$$r_i \circ (\theta_i, \Phi_{ij}^{out}, h_{ij}^{out}) = r_j \circ (\theta_j, \Phi_{ij}^{in}, h_{ij}^{in}).$$

We begin by expressing the input and output events using dilation and scaling of the fundamental clock $(1,0,0)$. By Proposition 7, we know that there exist atomic clock expressions $(\theta_i^{'}, \phi_{ij}^{out}, h_{ij}^{out})$ and $(\theta_j^{'}, \phi_{ij}^{in}, h_{ij}^{in})$ such that

$$(\theta_i, \Phi_{ij}^{out}, h_{ij}^{out}) = |\Phi_{ij}^{out}| \circ (\theta_i^{'}.(1,0,0) + h_{ij}^{out})$$

$$(\theta_j^{'}, \phi_{ij}^{in}, h_{ij}^{in}) = |\Phi_{ij}^{in}| \circ (\theta_j^{'}.(1,0,0) + h_{ij}^{in}).$$

We recall that $|\Phi_{ij}^{out}| = O_i$, and $|\Phi_{ij}^{in}| = I_j$. This in turn implies:

$$\theta_i^{'} = \frac{\theta_i}{O_i} = \frac{\theta_{sys}}{r_i O_i}$$

$$\theta_j^{'} = \frac{\theta_j}{I_j} = \frac{\theta_{sys}}{r_j I_j}$$

and because $r_i O_i = r_j I_j$:

$$r_i \circ (O_i \circ (\frac{\theta_{sys}}{r_i O_i}.(1,0,0) + h_{ij}^{out})) = r_j \circ (I_j \circ (\frac{\theta_{sys}}{r_j I_j}.(1,0,0) + h_{ij}^{in})).$$

□

We can now demonstrate that, if we ignore initial delay, any SDF system which is *consistent* has a representation in steady-state as a VSDF system using only regular clocks.

Proposition 9

$$\begin{aligned}&\exists \vec{\mathbf{r}}, \Gamma\vec{\mathbf{r}} = \vec{0} \Rightarrow \\ &\quad \forall i, j,\ \exists \theta_i, \theta_j, \Phi_{ij}^{out}, \Phi_{ij}^{in} \\ &\quad r_i \circ (\theta_i, \Phi_{ij}^{out}, 0) = r_j \circ (\theta_j, \Phi_{ij}^{in}, 0).\end{aligned}$$

Proof: *We ignore initial delay,* $\forall h,\ h = 0$. *We use*

$$\theta_{sys} = \mathbf{lcm}\{\forall i,\ r_i O_i\} = \mathbf{lcm}\{\forall j,\ r_j I_j\}$$

$$\theta_i = \frac{\theta_{sys}}{r_i}$$

and from Proposition 8, the augmented balance equations will be satisfied.
□

8 Latency and Initial Delay h

The equations presented in the above subsections make no assumptions about the relation between the input events and the output events of a given node. In Synchronous Data Flow, the results of a calculation are available immediately after a node is "fired", i.e. there is no latency for calculations. Time, insofar as it exists, is related to the number of times the different nodes in a system obtain all their inputs and generate their outputs i.e. the number of times the *sources* are "fired". In the case of a synchronous VLSI circuit, this is not realistic, as the output events are a direct function of the common clock, and in addition only exist after the start-up latency of the circuit. In the remainder of this section, we develop a model of latency, based on a per-node latency, which we use to calculate the initial delay h.

It is important to note that this per-node latency has no impact on the augmented balance equations; we still insist that inputs be received at the same time as the corresponding outputs are emitted.

8.1 Determining Per-node Latency

In the SDF model, the input data to a node waits in an input buffer until the node is "fired". Thus the balance equations can be used to determine an execution order for the nodes of a processing graph, provided that the *precedence* of data is respected. We prefer to take advantage of the physical latency of a given VLSI processor to determine at exactly which instant a node should begin its execution, i.e. be "fired". An important advantage which stems from our temporal constraints is that a node may begin execution before all of its input has arrived. This is typical of VLSI circuits, where processing often begins with the first input value. For example, a DCT circuit usually begins its Multiply-Accumulate sequence with the first input pixel, and we would like to model this.

In the following we develop an expression for the latency which will be associated with a given node. The principle of our expression is that, as in SDF, all the inputs for an operation must be present before the output may be generated. Note that this does not mean that no calculations will begin before all the data has arrived - unlike the case of SDF, our temporal expressions allow us to calculate when successive data will arrive, and thus to begin the calculations early; in other words, the node can get a "head start" on the arriving data. Our constraint is rather that of *data dependency*, i.e. if the output depends on an input, then it cannot be calculated before that input arrives.

Let the input expressions be regular canonical expressions of the form (θ, Φ, h). We introduce a second index m for multiple inputs and outputs. We assume that each input Φ^{in}_{mi} will be timed by a regular canonical expression. Clearly, $|\Phi^{in}_{mi}|$ and θ must be related such that:

$$\frac{\theta}{I_{mi}} \in \mathbb{N}.$$

We assume in the following that an appropriate θ has been found. We begin with regular input expressions:

$$\Phi^{in}_{mi} = \{0 \leq l < I_{mi} |\ l\theta_i / I_{mi}\}.$$

Solving for the instant in time at which the last of the $\{0 \ldots I_{mi} - 1\}$ inputs arrives, we have

$$\begin{aligned} \phi_{I_{mi}-1} &= \frac{(I_{mi} - 1)\theta_i}{I_{mi}} \\ &= \theta_i - \frac{\theta_i}{I_{mi}}. \end{aligned}$$

The last of all inputs for any given period will arrive on the input

$$\mathbf{max}\{\forall m | I_{mi}\}.$$

Let us denote the last input to node i for a period θ_i, by η:

$$\eta = \mathbf{max}\{\forall m | I_{mi}\}.$$

Since the output cannot be determined without all the inputs on which it depends, the moment at which all inputs have arrived, and the calculation can *finish* (it may have begun earlier), will be:

$$\phi_\eta = \theta_i - \frac{\theta_i}{\eta} = \frac{\eta - 1}{\eta}\theta_i.$$

We express the latency δ_i of a given node i as:

$$\delta_i \quad = \quad \frac{\eta - 1}{\eta}\theta_i + \Delta_i$$

where Δ_i is the calculation latency. The input time $\frac{\eta-1}{\eta}\theta_i$ functions somewhat like the *input threshold* of an SDF node, in that it describes the events needed to collect all the inputs necessary for one "firing" of the node. We note that in the case of a VLSI component, the node latency δ_i depends on the details of the implementation.

8.2 Conditions on h

As indicated in proposition 3, the initial delays h at both ends of a link must be equal in order for the augmented balance equations to be satisfied. Therefore, in all cases, the initial delay on a link between nodes i and j must be identical:

$$h^{in}_{ij} = h^{out}_{ij}$$

In addition, we can assign some "reasonable" constraints on h based on the circuits which we wish to model. We make the assumption that the circuits have a fixed architecture and treatment sequence, i.e. that the operations performed by a given circuit are not data-dependant.

Our principal constraint is that the relationship between the initial delays of all the inputs of a given node is fixed by the caracteristics of the circuit. Thus the initial delay of any input can be calculated from the initial delay of any other input:

$$\forall l, m \in Inputs, \exists k \in \mathbb{Z} \quad h_l = h_m + k$$

An example

Let output O_i of node i with operational latency δ_i be connected to input I_j of node j, and the two nodes share a common system clock. We will assume that node i is connected to a source, i.e. that $h_i^{in} = 0$ If we express the produce instants as τ^{out} and the consume instants as τ^{in}, we would like node j to wait before it begins to consume:

$$\tau_i^{in} = (\theta_i, \Phi_i^{in}, hi^{in})$$
$$\tau_i^{out} = (\theta_i, \Phi_{ij}^{out}, hi^{out})$$
$$hi^{out} = hi^{in} + \delta_i$$

We apply scaling to determine the input function:

$$r_i \circ (\theta_i, \Phi_{ij}^{out}, hi^{out}) = r_j \circ (\theta_j, \Phi_{ij}^{in}, hj^{in})$$

and by Proposition 3:

$$hj^{in} = hi^{out} = hi^{in} + \delta_i$$
$$\tau_j^{in} = (\theta_j, \Phi_{ij}^{in}, hi^{in} + \delta_i)$$

Thus if node j only begins expecting input at time δ_i after node i gets its input, there will be no synchronization error.

8.3 Periodicity: restriction to the Steady State

We finish this section with a restriction on the temporal expressions that makes it possible to express the events of a system only when it is in a steady state, i.e. the start-up latency of all nodes has been satisfied, and the shut-down latency has not yet begun. Typically this is of interest when working with systems that include a feedback loop. A feedback loop usually has a start-up phase, when the feedback is not yet active, and then a steady-state phase, when the feedback loop functions as normal. For the application of the augmented balance equations, only the steady state is relevant.

We therefore redefine a periodic system to be one in which the events repeat themselves *after an initialization period.* We do this without any loss of generality by starting our Augmented Balance Equation from instant $\tau \geq T_{init}$ instead of from $\tau \geq 0$:

$$\{r_i \circ (\theta_i, \Phi_{ij}^{out}, h)\} \geq T_{init} = \{r_j \circ (\theta_j, \Phi_{ij}^{in})\} \geq T_{init}.$$

In the typical case of an output O_i associated with a feedback value, which has output events only after time $T_{feedback}$, the augemented balance equation will not be satisfied during start-up. In this case, we will have the unbalanced Balance Equation:

$$\{\forall \theta_i t + \phi_{ij}^{out} + h < T_{feedback}\}$$
$$\emptyset \neq \theta_j t + \phi_{ij}^{in} + h_{ij}^{in}.$$

We remark that, given the timing expressions for the input and output associated with a feedback link, we can solve for the exact value of T_{init} associated with the input:

$$h_{ij}^{out} = h_{ij}^{in} - T_{init}.$$

In Synchronous Data Flow, start-up feedback values are "inserted" into the input buffers, so that they are available when needed to "fire" a node. These initialization values may be necessary for the correct operation of a system, but we will not calculate their synchronization.

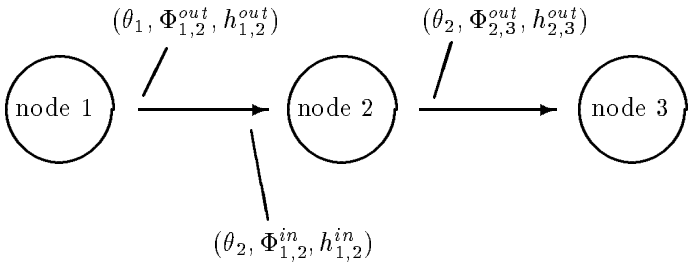

Figure 6: Clock expressions for VSDF nodes

9 Specifying a Synchronous Circuit

In the preceeding sections we introduced a notation for describing periodic events. We then showed how the notation can be applied to an SDF node, and used the notation to define an *augemented balance equation*. In this section we attempt to bridge the gap between temporal expressions, and a VLSI circuit. We first give a complete specification for a VSDF node, and then indicate how this applies to an example with actual circuits (in Section 7 we show how timing functions in the general case can be determined).

9.1 A VSDF Node

A VSDF node is specified by its latency δ, the set of all its inputs $(\theta, \Phi^{in}, h^{in})$, the set of all its outputs $(\theta, \Phi^{out}, h^{out})$, and the period θ:

$$VSDF_i = (\delta, \{\forall l \in Inputs, (\theta, \Phi^{in}_l, h^{in}_l)\}, \{\forall m \in Outputs, (\theta, \Phi^{out}_m, h^{out}_m)\}, \theta)$$

.

We will identify the Φ's and the h's by their termination nodes. Thus if nodes $1, 2, 3$ are connected in that order, as in Figure 6, node 2 will have an input specification connected to node 1:

$$(\theta_2, \Phi^{in}_{1,2}, h^{in}_{1,2}).$$

and an output specification connected to node 3:

$$(\theta_2, \Phi^{out}_{2,3}, h^{out}_{2,3}),$$

9.2 h_in and h_out

In order to simplify the process of assembling VSDF nodes, we synthesize a "smallest initial delay" h_in such that all the inputs h^{in} are either at the same time or later:

$$\forall l \in Inputs, \exists n \in I\!N \quad h^{in}_l = \texttt{h\_in} + n$$

We remark that this synthetic input h_in is often very close in practical terms to a RESET pin.

We make the assumption that the initial delay between input and output is fixed by the characteristics of the circuit. We call this delay the *latency* of a node, marked δ. If we synthesize a "smallest initial delay" h_out such that all the outputs h^{out} are either at the

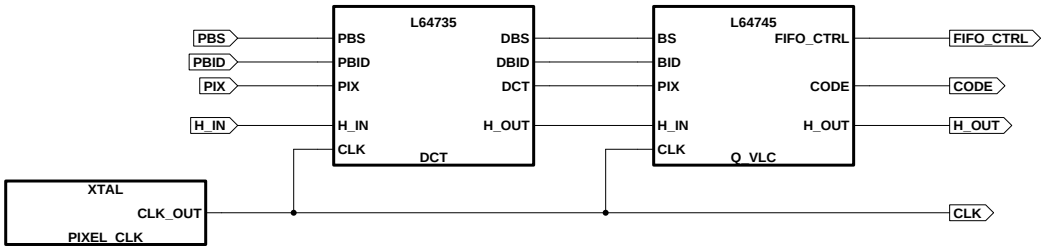

Figure 7: LSI Logic chipset for DCT and Quantification/VLC

same time or later, we can express all outputs as a function of h_out plus a non-negative constant:

$$\forall m \in Outputs, \exists n \in I\!N \quad h_m^{out} = \texttt{h\_out} + n$$

We also know that the latency is a constant, and thus we can calculate the outputs from h_in plus a non-negative constant:

$$\forall m \in Outputs, \exists k \in I\!N \quad h_m^{out} = \texttt{h\_in} + k$$

Replacing h_out by h_in+δ:

$$\texttt{h\_out} = \texttt{h\_in} + \delta$$

$$\forall m \in Outputs, \exists n \in I\!N \quad h_m^{out} = \texttt{h\_in} + \delta + n$$

We remark that the input h_in and the output h_out make it easy to compose a series of nodes, the initial output delay of one being the initial input delay of the next.

9.3 A Discrete Cosine Transform application

An 8x8 Discrete Cosine Transform (DCT), followed by quantization with compression (information loss) is used in a wide variety of image and video standards such as JPEG, MPEG, H.261 and ETSI 34-megabit coding.

Here we develop the *agumented balance equations* for a DCT connected to the input of a quantizer, as they might be used for JPEG image compression; for reference we use two VLSI circuits from LSI Logic, and the ETSI european image size. The LSI logic circuits are the L64735 DCT and the L64745 quantizer and run-length coder. The L64735 performs a transform on 8-bit pixels to yield 12-bit coefficients with a latence of 168 cycles. The L64745 takes these coefficients, quantizes them (with information loss), and applies a run-length coding (without information loss). These two circuits form part of a three-chip chipset intended for JPEG image compression and decompression [11].

9.3.1 The L64735 DCT

We first determine I, O, θ^*, Φ and δ from the parameters specific to this chip. The inputs to the L64735 DCT consist of the image pixels PIX, a block synchronization signal PBS, and a line synchronization signal PBID. The block synchronization signal exists at every clock cycle, whether it be true or false. The line signal is not used by the L64735, but is delayed

in order to be used to synchronize the L64745 (see Figure 7). The outputs of the L64735 DCT consist of the transform coefficients DCT, and the delayed block and line signals DBS and DBID. The clock input, labeled CLK, is connected to a global system clock. We note also the existance of a latency initialization input H_IN, and a latency initialization output H_OUT, which are used to compose the initial delays h in a system with several nodes. We therefore have the following parameters:

$$I_{PIX} = I_{PBS} = I_{PBID} = 64$$

$$O_{DCT} = O_{DBS} = O_{DBID} = 64$$

$$\theta^*_{L64735} = 64$$

$$\Phi_{PIX} = \bigcup_{0 \le k < 64} k$$

$$\Phi_{PBS} = \bigcup_{0 \le k < 64} k$$

$$\Phi_{PBID} = \bigcup_{0 \le k < 64} k$$

$$\Phi_{DCT} = \bigcup_{0 \le k < 64} k$$

$$\Phi_{DBS} = \bigcup_{0 \le k < 64} k$$

$$\Phi_{DBID} = \bigcup_{0 \le k < 64} k$$

$$\delta_{L64735} = 168$$

We can also determine expressions for h^{out}_{DCT}, h^{out}_{DBS}, and h^{out}_{DBID}, based on `h_out`; in fact, all outputs have the same initial delay, which is exactly the latency of the L64735 DCT:

$$h^{out}_{DCT} = h^{out}_{DBS} = h^{out}_{DBID} = \texttt{h\_out}$$

$$\texttt{h\_out} = \texttt{h\_in} + \delta_{L64735} = \texttt{h\_in} + 168.$$

9.3.2 The L64745 Quantizer and Run-length Coder

We shall only determine the inputs to the L64745, in order to demonstrate the augmented balance equation; the outputs of the L64745 are data-dependant, and thus cannot be modeled by Synchronous Data Flow. As shown in Figure 7, the L64745 has the inputs PIX BS and BID, all three of which can be connected directly to the outputs of the L64735 DCT. Different from the L64735 DCT, the L64745 functions stripe-by-stripe, where a stripe is a line of 8x8 pixel blocks. We shall use a line of 720 pixels, which when multiplied by the height of 8 yields a period of 5760 cycles of the pixel clock. We have therefore:

$$I_{PIX} = I_{BS} = I_{BID} = 5760$$

$$\Phi_{PIX} = \Phi_{BS} = \Phi_{BID} = \bigcup_{0 \le k < 5760} k$$

$$\theta_{L64745} = 5760.$$

9.3.3 The Balance Equation

We use a repetition vector $\vec{\mathbf{r}}$ determined by the parameters of the ETSI standard broadcast images: 248 lines/field * 720 pixels per line = 178560 pixels per image. For the two circuits given here, a repetition vector per line would suffice, but we prefer to consider the circuits as part of a hypothetical complete system. The values for r_{L64735} and r_{L64745} are therefore $r_{L64735} = 2790$ and $r_{L64745} = 31$. Both circuits run at exactly the pixel clock, so we assign $\theta^* = \theta$, i.e. the period determined from the circuit specifications is the period in the system. These parameters, combined with those determined from the characteristics of the circuits given above, yield:

$$r_{L64735} = 2790; O_{DCT} = 64$$

$$r_{L64745} = 31; I_{PIX} = 5760$$

$$r_{L64735} \circ (\theta_{L64735}, \Phi^{out}_{DCT}, h^{out}_{DCT}) = r_{L64745} \circ (\theta_{L64745}, \Phi^{in}_{PIX}, h^{in}_{PIX})$$

If we assign $h^{in}_{PIX} = h^{out}_{DCT}$, as indicated in (Figure 7), the augmented balance equation can be verified:

$$2790 \circ (64, \bigcup_{0 \le k < 64} k, 168) = 31 \circ (5760, \bigcup_{0 \le k < 5760} k, 168).$$

$$\begin{aligned} r\theta: &\quad 2790 * 64 = 31 * 5760 \\ \Phi: &\quad \bigcup_{0 \le k < 178560} k = \bigcup_{0 \le k < 178560} k \\ h: &\quad 168 = 168. \end{aligned}$$

The equations for BS and BID are identical.

9.4 Applications

The parameters θ, Δ, and the ϕ's for input and output can be encoded using the tools introduced in [14], and the component model presented in [6] [5]. These tools permit verification of correct timing, after the initialization period has passed. The system displayed in Figure 7 was captured from a Synopsys simulation [17] using these tools coded in VHDL [12].

10 Conclusion

We have presented a notation and a model which can ease the transition between a block-level Synchronous Data Flow graph, and an implementation-level synchronous system using specialized VLSI circuits. Our notation can be used to formally specify a system, and statically verify correct synchronization at a Register-transfer level. Our model provides a consistent format for both Synchronous Data Flow graphs, and a hardware-level RTL description of synchronous circuits. We have also presented several examples, taken from video coding. Tools based on this model, written in SIGNAL or VHDL, allow us to check that proper synchronization is maintained.

Limitations

Of course, our model and tools have many limitations. We do not yet have a tool to automatically perform the transition between an SDF-level and a circuit-level synchronization; such a tool should be possible for at least the simple cases. Our starting point is not, strictly speaking, an SDF graph as defined by Lee and Messerschmitt, as we assume a "unitary"

delay for each operation (although we note that this does not change the input/output matrix). Nor do we offer the automatic optimization capabilities of the Silicon Compilers. In order to simulate our system using SIGNAL, we also require a 'C'/Unix description for each processor, or node in the SDF graph, whose behavior conforms to the model detailed in [6] [5].

A more fundamental limitation of our model, is that we assume all input events occur at known instants relative to the known clock of a given node. This is in keeping with our decision to consider synchronous systems, but nevertheless implies that timings are carefully controlled. We point out that our equivalence expressions do not require a system-wide clock, only that the input and output expressions common to a given link are properly timed. Nevertheless, for practical implementations, a common system clock (which can be determined as presented in this article) seems indicated.

References

[1] Joseph Buck, Soonhoi Ha, Edward Lee, and David Messerschmitt. *Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems.* Technical Report, University of California, Berkely, August 1992.

[2] D.Lanneer, S.Note, F.Depuydt, M.Pauwels, F.Catthoor, G.Goossens, and H.De Man. *Architectural synthesis for medium and high throughput signal processing with the new CATHEDRAL environment.* Kluwer, Boston, April 1991. In R.Camposano and W.Wolf, *Trends in High-level Synthesis.*

[3] Comdisco Systems Inc. SPW - The DSP Framework. Data Sheet, 1993.

[4] inmos. IMS A121 2-D Discrete Cosine Transform Image Processor. Data Sheet, 1991.

[5] Dominique Lavenier and Roderick McConnell. *From Behavioral to RTL Models: an approach.* Technical Report 822, IRISA, Campus de Beaulieu, Rennes, FRANCE, May 1994.

[6] Dominique Lavenier and Roderick McConnell. From Behavioral to RTL Models: an approach. In *Proceedings of the 5th IEEE International Workshop on Rapid System Prototyping*, Grenoble, France, June 1994.

[7] Edward A. Lee. Consistency in Dataflow Graphs. In *IEEE Transactions on Parallel and Distributed Systems*, pages 355–369, IEEE Computer Society, April 1991.

[8] Edward A. Lee. Multidimensional Streams Rooted in DataFlow. In *IFIP Working Conference on Architectures and Compilation Techniques of Fine and Medium Grain Parallelisim*, North-Holland, January 1993. Orlando, Florida.

[9] Edward A. Lee and David G. Messerschmitt. Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. *IEEE Transactions on Computers*, C-36(1), January 1987.

[10] Edward A. Lee and David G. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, 75(9), September 1987.

[11] LSI Logic. L64735 dicrete cosine transform processor. Data Sheet, 1990.

[12] *IEEE Standard VHDL Language Reference Manual.* Institute of Electrical and Electronics Engineers, New York, 1987. IEEE Std 1076-1987.

[13] E. Martin, O. Sentieys, H. Dubois, and J. Philippe. GAUT: an architectural tool for dedicated signal processors. In *Euro-DAC '93*, pages 14–19, IEEE, Hamburg, September 1993.

[14] Roderick McConnell, Alain Kerihuel, and Frédéric Raimbault. Tools for Correct DSP Synchronization. In *Proceedings of the 1993 IEEE Workshop on VLSI Signal Processing*, pages 206–214, Koningshof, Velhoven, The Netherlands, October 1993.

[15] Mentor Graphics Corporation, EDC. DSP Station. Data Sheet, 1993.

[16] Charles Seitz. *System Timing*, chapter 7, pages 218–254. Addison-Wesley, 1980. In Mead and Conway, *Introduction to VLSI Systems.*

[17] *VHDL System Manual V3.0.* Synopsys, Inc., 1992.

[18] J. van Meerbergen, P. Lippens, B. McSweeny, W. Verhaegh, and A. van der Werf. *PHIDEO: High-Level Synthesis for High Throughput Consumer Applications.* Technical Report, Philips Research Laboratories Eindhoven, November 1992.

[19] J. van Meerbergen, P. Lippens, B. McSweeny, W. Verhaegh, A. van der Werf, and A. van Zanten. Architectural strategies for high-throughput applications. *Journal of VLSI Signal Processing*, 5(2/3):201, April 1993.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399