Prototyping of VLSI Components from a Formal Specification

Roderick Mcconnell

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Prototyping of VLSI Components from a Formal Specification

Roderick McConnell

N° 2363

septembre 1994

PROGRAMME 1

ISSN 0249-6399

Prototyping of VLSI Components from a Formal Specification *

Roderick McConnell **

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

Rapport de recherche n°2363 — septembre 1994 — 30 pages

Abstract: We present a trajectory from formal specification to component, which can be used to prototype applications which combine off-the-shelf components and custom hardware, provided they meet the constraints of Synchronous Data Flow. The formal specification allows us to prove that custom components will be correctly synchronized in a system context. We first introduce VLSI Synchronous Data Flow, elaborate the constraints it imposes, and define functions relevant to constructing a system. Then we describe the construction of a simulation model. Finally, we develop a component for motion video coding in VHDL, starting from a formal specification.

Key-words: motion video coding, 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, and by the Doctoral-candidate Network for System and Machine Architecture of the DRED.

**rod@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

Developpement systématic d'un composant VLSI à partir d'une spécification formelle

Résumé : Nous developpons un composant lié au codage d'image vidéo, en nous appuyant sur une spécification formelle avec le modèle flot de données synchrone VLSI. Cette spécification nous permet d'établir les bonnes propriétes de synchronisation, une fois le composant inséré dans son environnement. Une notation temporelle ainsi que les fonctions nécessaires à son utilisation pour la modélisation des événements dans un système périodique sont introduits. Ensuite nous décrivons la construction d'un composant de simulation. Enfin, nous donnons un exemple d'un système pour l'estimation de mouvement, ainsi que des résultats de synthèse avec les outils de Synopsys.

Mots-clé : conception de systèmes spécialisés, VLSI, codage d'image vidéo

1 Introduction

The transition between a behavioral model of a system and a Register Transfer Level (RTL) model is often a difficult step, involving a complete re-write of each component. In addition, it is often the source of errors: according to E.E. legend, "Fifty percent of ASIC prototypes don't work in the system environment because of chip specification errors, chip interaction problems or electrical effects related to board implementation." [3].

In this article we propose a systematic approach to prototyping a system at the Register Transfer level, starting from a formal specification of the system timing in VLSI Synchronous Data Flow (VSDF) [12] [14] [13]. Our approach is based on a formal specification notation and a simulation model, positioned between Synchronous Data Flow (SDF) 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 [28]. 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 Synchronous Data Flow graph [20] [21] 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 Formal Specification notation and 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 [4], GRAPE-II [15] or Alta/Cadence's "Signal Processing WorkStation"[9]. If a pipelined hardware implementation is envisaged, these systems suffer from the disadvantage that blocks of data are buffered between operations. Our approach also differs from existing systems for developing periodic circuits, such as CATHEDRAL [8], PHIDEO [30], Mentor Graphic's "DSP Station" [25] or GAUT [23]; these systems are based on synthesis, and use a description of the internal operations to determine the synchronization of the data from the data dependency. We instead use the formal specification of synchronization in VSDF to prove correctness.

The remainder of the article is organized as follows. In section 2 we review the principles of Synchronous Data Flow. We then introduce VLSI Synchronous Data Flow and its associated operators in section 3. We also define necessary conditions for functional hardware, which are an extension to the necessary conditions given in [20], [21], and apply these to components in section 4. A component model which aids in applying VSDF to the development of an RTL system simulation, is also presented. We finish by applying VSDF to synchronous circuits and system design in section 5, using VHDL and the Synopsys synthesis tools. Examples, in general taken from video coding, are given throughout the article to illustrate key points.

2 Synchronous Data Flow

For applications which meet its strict constraints on periodicity, the Synchronous Data Flow graph offers a powerful model for static analysis and simplified implementation. Data Flow graphs [1] [7] are often used in signal processing to model real-time applications because they offer an intuitive and visually-oriented approach [6]. SDF is a variant of the traditional Data Flow graph; the amount of data consumed and produced by each node for all its inputs and outputs, is fixed in advance.

2.1 The SDF model

In the SDF model, 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 [19]. 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.

2.2 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 Γ. This same constraint is expressed in [18] [19] 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.

2.3 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. 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 [4], appears better adapted to modeling software than hardware. 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 can be a great deal 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 are often performed by a single dedicated circuit, which 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. Produce and consume thresholds in SDF must be replaced by a stream of data, word by word, 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.

2.4 Approaches for Hardware

One approach to modeling a hardware system at the Register Transfer level, is to break down block-oriented operations into simpler sub-operations, until each operation works on a single word of data. We discard this approach, as it yields an unwieldy and counterintuitive graph, and is also a poor model for VLSI circuits.

We choose instead to model blocks of data at the Register Transfer level as being transferred serially, word by word. Our formal specification notation, allows us to express the timing of this serial transfer, and to statically verify that the transfers are properly synchronized between producer and consumer. Nodes in our graph, and components in our library, will be composed of two sub-elements, a timing component and an operator. We use the timing component to represent the timing of a cyclic transfer, whether it be

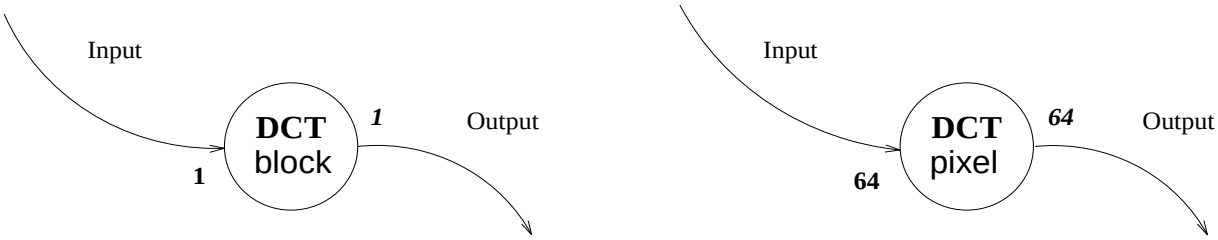

Figure 1: SDF node at block and pixel level

at a block level (native SDF) or serial stream (circuit-level). In the following section, we describe our formal specification notation.

3 VSDF: Synchronous Data Flow for VLSI

As indicated in the preceding section, the SDF model is a powerful tool for applications which meet its rather strict constraints. We propose that, with some minor modifications, 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 [10], or the LSI Logic L64735 8x8 DCT chip [22]. 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 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 likewise for the output. Given this description of inputs and outputs, we develop formal expressions for correct operation in the following subsection.

Constraints for Hardware

As presented in [18], 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 4.3 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 text 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.

3.2 A Temporal Notation

In order to formalize the constraints on the instants introduced above, we first introduce a notation for temporal expressions. A complete description of the notation, along with a more formal development of the associated proofs, is given in [14]. 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. We also define a canonical form which eliminates possible redundancy in our temporal expressions.

3.2.1 A Periodic Event or Set of Events

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 simple temporal expression is denoted by (θ, ϕ, h), where θ is the period, ϕ the phase, and h the initial delay. We also often work with a union of events ϕ which represent the phases of multiple (distinct) events during a period. Therefore we introduce the set Φ to represent a collection of k ϕ's during one period. We use the following notation for our temporal expressions:

Definition 1 (Temporal Expressions)

$$(\theta, \phi, h) : t \longmapsto \theta t + \phi + h$$

$$(\theta, \Phi, h) : t \longmapsto \left\{ \begin{array}{l} \theta t + \phi_0 + h \\ \theta t + \phi_1 + h \\ \ldots \\ \theta t + \phi_{k-1} + h \end{array} \right\}$$

$$\theta, h \ \in I\!N;\ \phi \in \{0 \ldots \theta - 1\}$$

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

with

$$|\Phi| = k.$$

If t is taken as time, $t \in I\!N$, then the affine expression (θ, ϕ, h) or (θ, Φ, h) specifies an infinite series.

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 canonical form, which eliminates ambiguity in the notation.

3.2.2 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 2 (Canonical Form) *An expression is in canonical form* **iff**

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

$$(\theta, \Phi, h) \Rightarrow \phi_0 = 0, \phi_i < \phi_{i+1}.$$

Proposition 1 *An expression (θ, ϕ, h) can be put into canonical form by subtracting ϕ from the phase, and adding it to the initial delay:*

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

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

3.3 Operations on Temporal Expressions

In order to manipulate temporal expressions, we will make use of two operations and a condition of equivalence. The operations are *scaling*, and *delay*. Scaling corresponds to extending the period of observation of a temporal expression, while 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 two operations and the condition of equivalence below.

3.3.1 Scaling: Repeating the Period

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

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

where

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

The 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

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

Examples: multi-event periodic operations

We can now demonstrate the equivalence of the clock expressions $(1, 0, 0)$ and $(64, \{0 \ldots 63\}, 0)$, the second being merely a scaling of the first by 64 (i.e. 64 repetitions of the period):

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

We might then continue to scale this new expression $(64, \{0 \ldots 63\}, 0)$ by 2, to get a period which is twice as long:

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

3.3.2 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 4 (Delay)
An expression (θ, ϕ, h) delayed by a constant $\delta \in I\!N$ yields:

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

An expression (θ, Φ, h) delayed by a constant $\delta \in I\!N$ yields:

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

An example: delaying an expression

In a motion video coding system, an 8x8 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 circuit performs the transform in 128 cycles, the temporal expression for the output will be delayed by $\delta = 128$ with respect to the input:

$$h^{out} = h^{in} + 128$$

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

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.*

3.3.3 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, term by term. 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 illustrated above, the expressions (64,{0,. . . , 63},0) and (1,0,0) are equivalent, because they define the same instants. 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, term by term [14] [12] :

Definition 5 (Equivalence)

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

In the following section, we show how this condition of equivalence can be applied to statically verify that the synchronization of transfers is correct.

4 Proving Correct Synchronization

A Synchronous Data Flow graph can be statically analyzed to determine if it is consistent, as presented in [18], and to derive a periodic schedule from 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*. We then show how the notation can be used to specify VLSI components, and sketch how this can be applied to static verification of synchronization. Finally, we discuss simulation components for simulating complete systems at a Register Transfer level.

4.1 Application to SDF

SDF requires that each "actor", or node in the processing graph, 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. In order to model our constraints, we associate with a node i 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

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

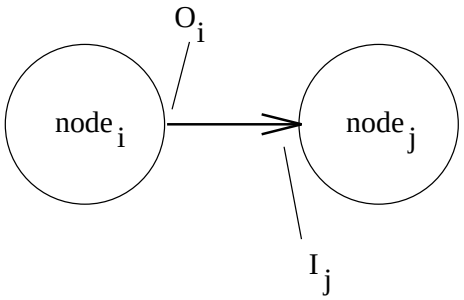

Figure 2: Two SDF Nodes

4.2 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 2). 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 6 (Augmented Balance Equation)

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

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

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. In order to simplify the task of static verification, we now show that, if the consume and produce instants are always identical, then their count is equal:

Proposition 2 *If r_i and r_j are solutions to the balance equation determined from the Topology Matrix Γ [20] [21], and there exists a suitable θ_i, θ_j such that the augmented balance equation (2) is satisfied, then (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:

$$|\Phi_{ij}^{out}| = O_i$$

$$|\Phi_{ij}^{in}| = I_j$$

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

□

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:

$$O_{src} = 1, I_{DCT} = 64$$

$$r_{src}O_{src} = 64 * 1 = 1 * 64 = r_{DCT}I_{DCT}$$

$$(\theta_{src}, \phi^{out}, h^{out}) = (1, 0, 0) \Rightarrow r_{src} \circ (\theta_{src}, \phi^{out}, h^{out}) = 64 \circ (1, 0, 0)$$

$$(\theta_{DCT}, \Phi^{in}, h^{in}) = (64, \bigcup_{0 \le k < 64} k, 0) \Rightarrow r_{DCT} \circ (\theta_{DCT}, \Phi^{in}, h^{in}) = 1 \circ (64, \bigcup_{0 \le k < 64} k, 0)$$

$$(64, \bigcup_{0 \le k < 64} k, 0) = (64, \bigcup_{0 \le k < 64} k, 0).$$

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

4.3 Specifying a Synchronous Circuit

The temporal expressions introduced in section 3 describe the synchronization of transfers between circuits. We now describe how these are applied to the inputs and outputs of a node, in order to model a given component. In particular, 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 only exist after the start-up latency of the circuit. In addition, we will only consider the case where the latency is constant, and in particular is not data-dependent.

4.3.1 A VSDF Node

A VSDF node is specified by its latency δ (*always* a constant), 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 3, node 2 will have an input expression on the link to node 1:

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

and an output expression on the link to node 3:

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

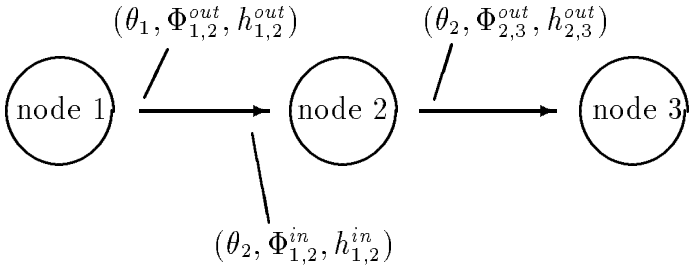

Figure 3: Temporal expressions for VSDF nodes

4.3.2 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 same time or later :

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

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.

4.4 Static Verification of Synchronization

Correct synchronization of transfers on every arc of the system graph, can be verified using the augmented balance equations. If, for every arc in the graph, the augmented balance equation 2 is satisfied, then the transfers will be correctly synchronized during the operation of the system. The equations ensure that not only will the number of valued consumed and produced on each arc be equal after a certain number of node firings, but also that the instants at which these values are produced or consumed are the same.

The techniques for finding the periodic schedule or repetition vector $\vec{\mathbf{r}}$ of all r_i's for a Topology Matrix Γ are given in [20] [21]; we do not repeat them here. Static verification of synchronization can be accomplished by verifying that the temporal expressions θ_i and ϕ_i on both ends of a link, when scaled by r_i, satisfy the augmented balance equation, and by determining appropriate initial delays h on inputs and outputs.

We remark that part of the constraint implied by the augmented balance equation 2 is that each node only begins processing when its input data is available. Because the operation of scaling does not change the initial delay h [14], 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:

$$h_{ij}^{in} = h_{ij}^{out} \tag{3}$$

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.

4.5 Constructing Simulation Components

We prototype components at the Register Transfer level using a synchronous timing sub-element. We completely separate timing aspects of a component from operational aspects, reflecting the usual conceptual distinction between the *function* of a circuit and the *timing* of a circuit. The timing sub-element determines the moments at which the inputs and outputs exist, and is used to transition between an SDF graph and an RTL model. The operation sub-element performs the actual calculation, i.e. the behavior of a component. The operator is treated as a "black box", without regard for its implementation.

In order to simulate entire systems at a Register Transfer level, we first construct RTL simulation components from the timing and behavioral sub-elements. We have chosen this composition because it permits us to preserve the behavioral aspect of components, while adding the necessary timing for RTL simulation. This approach differs from existing systems for developing periodic circuits, such as CATHEDRAL [27], PHIDEO [30], Amical [11] [26] or GAUT [23], which use knowledge of the internal operations to perform synthesis, allocation and optimization.

Initialization and termination sequences are an important part of a prototype, particularly when working with complex VLSI circuits. On the other hand, when working with a behavioral model, these timing considerations either are ignored or are expressed differently. We model the hardware initialization sequence using the timing sub-element of the component.

We use a three-part maquette as the starting point for each component. The maquette includes the timing sub-element, the operation sub-element, and interface "glue" between the input/output and the operation, as shown in Figure 4. The part labeled **operator** specifies the behavior of a component. The *timing diagram* of the component, i.e. the VSDF *temporal expressions*, is presented to the exterior by the **timing** sub-element. The timing includes a **check** on all inputs, and an explicit control of the instants when the output will **exist**. An **interface** is used to handle any differences in format between the timing sub-element and the operation; this interface, for example, handles the buffering of a stream of words for a block-oriented operator.

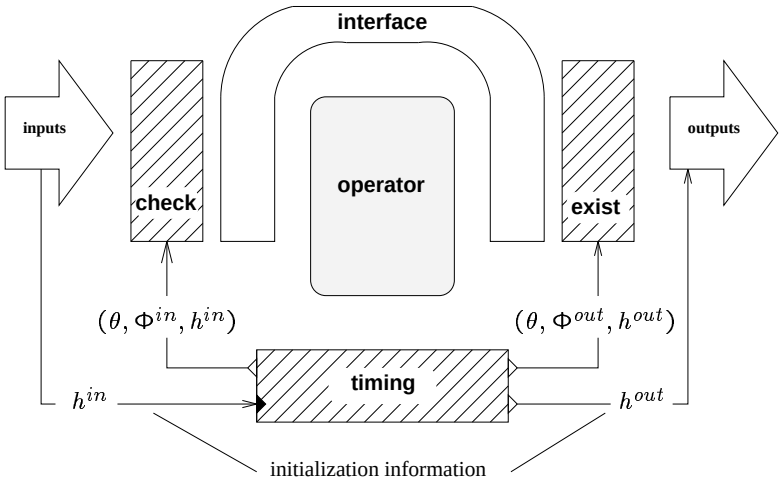

Figure 4: The Simulation Model

5 A Case Study: a motion-estimation subsystem

We now present a detailed case study of VSDF used to prototype a motion-estimation subsystem. We will apply two SGS-Thomson STI3220 motion-estimation processors to allow an enlarged search window. The STI3220 processor performs an exhaustive search for the best possible movement vector, trying all possible displacements for a given block of pixels; we will use two processors to evaluate two different search windows, together with a component to select the better of the two results.

The specifications and timing information for the STI3220 are taken from the data-book [29]. The component which selects the better of the two vectors does not exist, so we will follow a top-down methodology to develop a prototype from a formal specification in VSDF. The necessary synchronization will be specified in VSDF, and the *augmented balance equations* will be verified. A simulation component which meets the VSDF specification will then be developed in VHDL. After simulation to insure correct operation, the component is synthesized using the Synopsys tools.

The STI3220: principle of operation

The equation used to evaluate a given displacement vector (i, j) for an MxN reference block is given by (Equation 4) [2].

$$distortion_{i,j} = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} \mid SearchWin_{x+i,y+j} - RefWin_{x,y} \mid \qquad (4)$$

The STI3220 contains 256 systolic processors, which are used to simultaneously evaluate the 256 different displacement vectors.

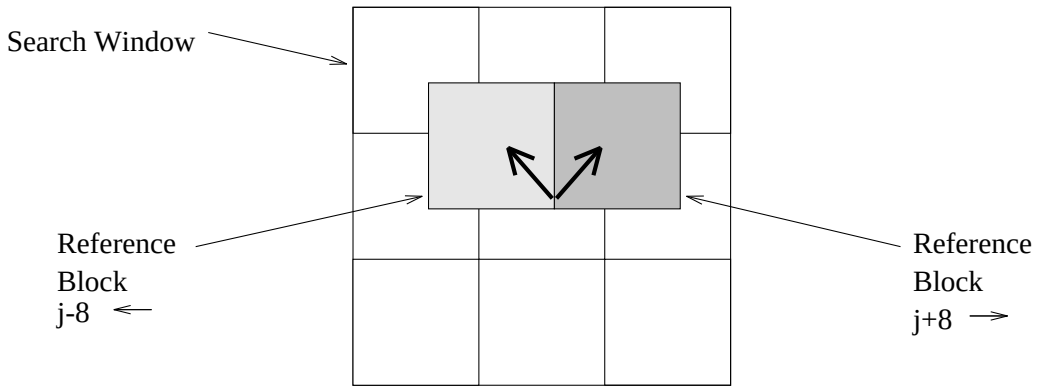

Figure 5: Displacement of the reference block

In order to keep our example simple, we do not consider interpolated pixels, nor do we generate synthetic pixels for a search window which extends outside the physical image; instead, we will use *edge control* signals to limit the search to the physical image.

5.1 Enlarging the search window

We would like to enlarge the search window in the horizontal direction, corresponding to enlarging the set of motion vectors to be evaluated. In video sequences, movement in the horizontal direction is more common than movement in the vertical direction (due both to the movement of objects across the screen, and to "panning" of the camera). If we double the length of the search window, we will require twice as many calculations in order to perform the exhaustive search. Since the STI3220 is limited to a horizontal displacement of -8/+7 pixels, we will require a second STI3220 circuit to evaluate the enlarged search window.

The principle of our approach is to use one Motion Estimation processor in the forward direction and the other in the backwards direction, and to then choose the better of the two vectors. We take advantage of the fact that the STI3220 provides as output not only the movement vector, but also a measure of its quality, the *distortion*. The distortion associated with each of the two vectors will be evaluated, and the vector associated with the lesser of the two distortions will be retained.

This example allows us to demonstrate the elaboration of a multi-component subsystem with relatively complex timing constraints. In addition, the selection of a motion vector based on the distortion (and edge controls) represents an operation of moderate complexity which is not available as a commercial circuit. We will develop this specialized component in VHDL, using tools from our library, and then synthesize it using Synopsys.

We apply two Motion Estimation processors, the one for evaluating the vectors with a negative displacement ($j \in \{-16 \ldots -1\}$), and the other for evaluating vectors with a positive displacement ($j \in \{0 \ldots 15\}$). The two resulting distortions will be compared, and the vector associated with the lessor of the two distortions will be selected. The block

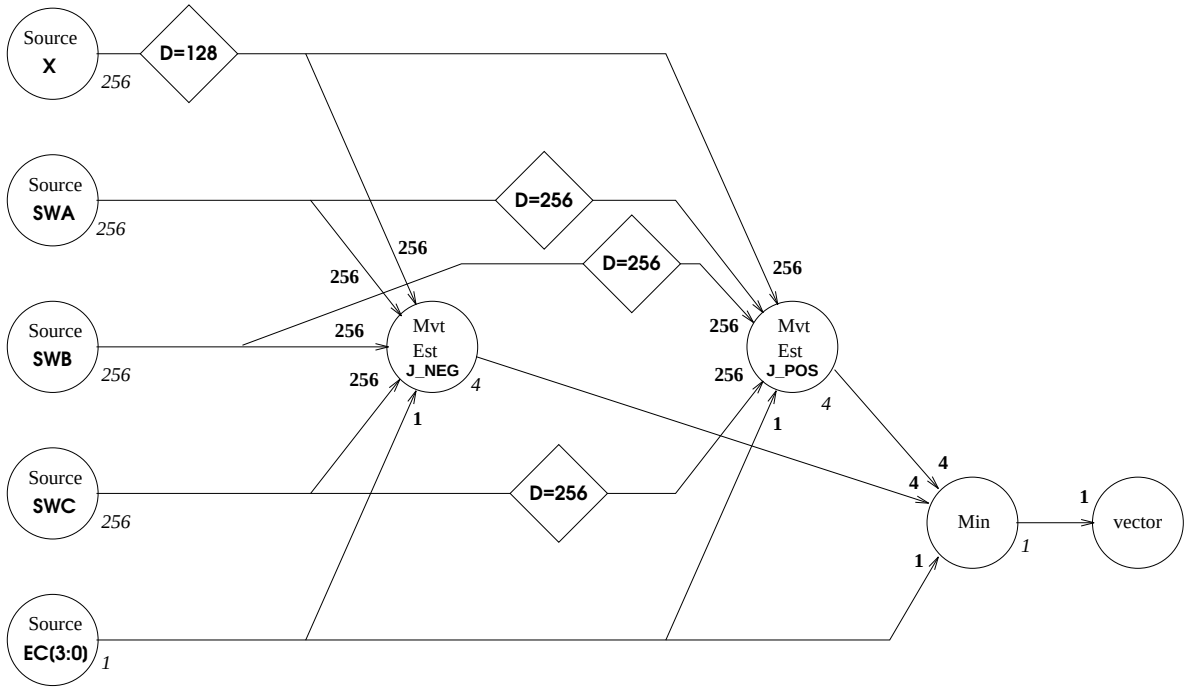

Figure 6: Estimation subsystem with two M.E. processors

size will be 16x16, for an image coded in format CCIR-601 4:2:2 [5]; we will use only luminance information for the motion estimation.

The search space for the processor which evaluates the negative displacement vectors, will be advanced by 8 columns with respect to the reference block. Likewise, the search window for the positive processor will be retarded by 8 columns with respect to the reference block. We present the view of the Motion Estimation processor in the *negative* direction in (Figure 5).

We can represent this subsystem using an SDF graph, as shown in (Figure 6).

5.2 Specification of the STI3220 in VSDF

The STI3220 performs exhaustive-search motion estimation, in this case in a configuration for MPEG. Only the luminance information will be used; it is in blocks of 16 by 16 pixels, i.e. 256 pixels (or 256 cycles of the clock) per macro-block.

The specifications of the STI3220 [29] require an initialization phase of 144 cycles of the pixel clock (equivalent to 9 columns of pixels), before the first pixel of the search window arrives. Then, 7 columns of the search window must arrive before the first column of the reference block. The total initialization time is therefore 256 cycles. The chronogramme in the datasheet specifies a latency of 292 cycles between the arrival of the first pixel of the reference block, and the output of the motion vector.

A simplification of the output

The output of the STI3220 appears as a sequence of values on a one-byte-wide bus noted IOB. The *distortion* is a value which requires 16 bits, and so it is presented in two steps, most significant byte (**msb**) followed by least significant byte (**lsb**). The three bytes of output (vector, distortion.**msb**, distortion.**lsb**) appear during cycles 1, 3, and 4 of the output sequence; during the second output cycle, no information is present on IOB. To simplify the control, we assume a dummy value during the second cycle, which is not used. This allows us to express the synchronization on IOB as a sequence of four events, without intervening "holes". The VSDF expressions for the STI3220 are thus:

$$I_X = 256$$

$$I_{EC0} = I_{EC1} = I_{EC2} = I_{EC3} = 1$$

$$I_{SWA} = I_{SWB} = I_{SWC} = 256$$

$$O_{IOB} = 4$$

$$\delta = 548$$

$$\theta = 256$$

Input:

$$\begin{aligned} X: &\quad \theta \circ (1,0,0) + (256 + h^{in}) \\ EC_0, EC_1, EC_2, EC_3: &\quad (\theta,0,0) + (256 + h^{in}) \\ SW_A, SW_B, SW_C: &\quad \theta \circ (1,0,0) + (144 + h^{in}) \end{aligned}$$

Output:

$$IOB: \quad (\theta, \{k | 0 \leq k < 4\}, \delta + h^{in}).$$

We recall that the motion estimation processor STI3220 requires a time $(256 + h^{in})$ before the first reference block arrives to complete its initialization.

5.3 Specifying the Selector

The motion vector selector is to our knowledge a component which is not commercially available. We will therefore specify its operation and its synchronization, and then develop the component using VHDL. We recall that this component should select the "better" of two motion vectors, based on a criterium of minimum distortion, and taking into account the horizontal edge controls. It should then provide the selected motion vector, adding the appropriate displacement (positive or negative) depending on the relative position of the search window of the Motion Estimation processor. The requirements are:

1. the component must accept on input both the movement vectors and their respective distortions;

2. the distortion values to be compared will arrive in two cycles (**msb** followed by **lsb**);

3. edge control should be supported in a way that is compatible with the STI3220.

Analyzing the movement vectors requires that the displacements be stored while the distortions are compared. In addition, the horizontal component of the movement vector, which appears on bits IOB[7:4], must be adjusted to correspond to the displacement of the search window: -8 for the negative displacement, and +8 for the positive displacement (this also increases the number of bits in the movement vector by one, from 8 to 9)

The actual comparison of distortions occurs during two cycles, **msb** followed by **lsb**. The priority is, in descending order, given to the edge controls, the **msb** of the distortion, and finally to the **lsb**. The VSDF model requires that the latency of a component be constant. Therefore, the result of the comparison will not be available until after the **lsb** of the distortion has been evaluated, even if it is irrelevant.

The left and right edge controls EC1 and EC3 can no longer be applied directly to the STI3220's, because they assume that the search window is centered around the reference block (we recall that the search window has been displaced, either to the left or to the right). We must therefore suppress invalid vectors during the selection. We consider two possible cases:

- EC1 = '1', which indicates the right edge of the image, and therefore the suppression of the result of the processor which evaluates the positive displacement *except in the case of a zero horizontal displacement.*

- EC3 = '1', which indicates the left edge of the image, and therefore the suppression of the result of the processor which evaluates the negative displacement.

A final consideration is that the edge control signals are synchronized to the beginning of the processing by the STI3220's. We must therefore save these values until the motion vectors have been calculated. We save these values EC1 and EC3 inside the selection component, in order to minimize the component count. The EC are synchronized with the first pixel of the reference block X, which arrives at time $h^{in} + 256$ as presented in section 5.2.

The vector selector component takes as input two sequences of four values (the simplified IOB sequence) as well as the edge controls EC1 and EC3. The result is available one cycle after the distortion **lsb**. The VSDF specification is therefore:

$$\delta = 4$$

$$\theta = 256$$

Input:

$$IOB_{POS}, IOB_{NEG}: \quad (\theta, \{k | 0 \leq k < 4\}, h^{in})$$
$$EC_1, EC_3: \quad (\theta, 0, h^{in})$$

Output:

$$(\theta, 0, h^{in} + \delta).$$

5.4 An RTL Model of the Subsystem

We now have a formal specification in VSDF of the different components which make up the motion estimation subsystem. The next step is to describe the entire subsystem at a register-transfer level, introducing the constraints relative to the system clock into the SDF model (Figure 6). We would like to displace the search windows relative to the reference block in each of the processors J_NEG and J_POS. We do this by changing the arrival time of the search window relative to the reference block.

5.4.1 Shifting the Search Window

For the processor J_POS, we delay the search window by 8 columns, or 128 cycles of the pixel clock, which corresponds to shifting the search window to the left relative to the reference block:

Search Window:

$$SW_A, SW_B, SW_C: \quad \theta \circ (1,0,0) + 144 + h^{in} + 128 = \theta \circ (1,0,0) + (400 + h^{in})$$

The motion estimation processor J_NEG will have its search window arrive earlier than the reference window, which corresponds to shifting the search window to the right relative to the reference block:

Search Window:

$$SW_A, SW_B, SW_C \quad \theta \circ (1,0,0) + 144 + h^{in} - 128 = \theta \circ (1,0,0) + (16 + h^{in})$$

5.4.2 The Complete System: Static Verification

We can now perform static verification of synchronization between the two Motion Estimation processors and the Vector Selector component. We will use the pixel clock to synchronize all components; the period for a block of 256 pixels is therefore:

$$\theta = 256.$$

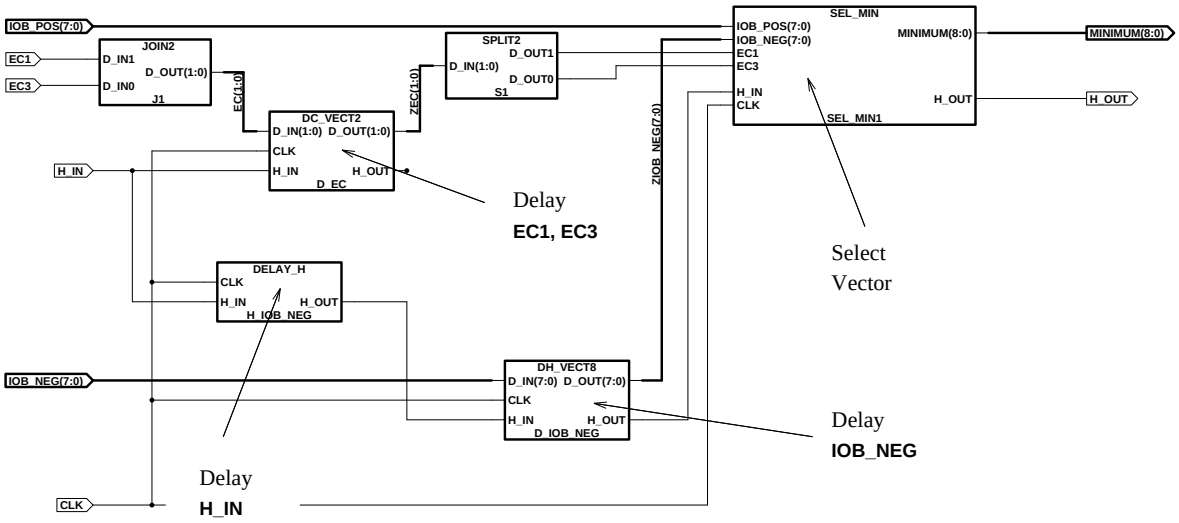

Figure 7: Complete Vector Selector component

The repetition counts will all be 1, corresponding to one block per period. We take the input `h_in` to the Vector Selector as `h_out` of a Motion Estimation processor. The Augmented Balance Equations are therefore:

$$\begin{array}{rcl} IOB^{out}_{POS}, IOB^{out}_{NEG}: & & IOB^{in}: \\ (\theta, \{k|0 \le k < 4\}, h^{out}_{STI}) & = & (\theta, \{k|0 \le k < 4\}, h^{in}_{Min}) \\ (256, \{k|0 \le k < 4\}, \delta_{STI} + h^{in}_{STI}) & = & (256, \{k|0 \le k < 4\}, \delta_{STI} + h^{in}_{STI}). \end{array}$$

This configuration assumes that the search window and reference block are appropriately synchronized with the initialization input `h_in`. We also assume that the values for the edge controls EC0 ... EC3 are calculated elsewhere. One could use the specification above to create a working motion-estimation subsystem. However, by simple displacement of the delays, we will show below how to create a system which requires less storage.

5.4.3 An Improved Subsystem

In the implementation specified above, every search-window input SW of the Motion Estimation processor `j_pos` is delayed, which in turn implies substantial delay storage. Since there are three times as many data in the search window as in the reference block, it is desirable to delay the reference block instead of the search window. In order to reduce the delay storage, we will modify the formal specification, while maintaining the synchronization across connections. This will imply a modification of the specification for the Vector Selector component (but not the STI3220). The resulting formal specification for the Vector Selector will still satisfy the *augmented balance equations*, and will also describe directly a usable component.

Changing the delay scheme can be accomplished by delaying all inputs (the search window SWA, SWB, SWC, the reference block X, the edge controls EC0 .. EC3 and the initialization input $H^{in}_{J\_NEG}$) to the processor j_neg by one block, i.e. 256 cycles of the pixel clock. The result is that the search windows of the two Motion Estimation processors are "aligned", eliminating the extra delay on SWA, SWB, SWC.

We express this operation in VSDF using Delay (addition of 256) on all the input expressions of the Motion Estimation processor j_neg. The new expressions are:

$$h^{in}_{J\_NEG} = h^{in}_{SYS} + 256$$

Inputs:

$$\begin{aligned} X: \quad & \theta \circ (1,0,0) + (256 + h^{in}_{J\_NEG}) \\ & = (\theta, \{0 \ldots \theta - 1\}, 512 + h^{in}_{SYS}) \\ EC_0, EC_1, EC_2, EC_3: \quad & (\theta, 0, 0) + (256 + h_{IN\_NEG}) \\ & = (\theta, 0, 512 + h^{in}_{SYS}) \\ SW_A, SW_B, SW_C: \quad & \theta \circ (1,0,0) + (16 + h^{in}_{J\_NEG}) \\ & = (\theta, \{0 \ldots \theta - 1\}, 272 + h^{in}_{SYS}) \end{aligned}$$

Outputs:

$$IOB \quad (\theta, \{k | 0 \leq k < 4\}, \delta) + (128 + h^{in}_{SYS}).$$

In the simulation model, the delays on the values are implemented by tools for synchronous delays presented in [24].

The advantage of the new configuration is that the volume of data to be delayed is much reduced: the edge controls are one bit each, once per block of 256 pixels; and the reference block X represents one-third the volume of the search window SWA, SWB, SWC.

5.4.4 Specifying the Modified Selector

We must modify the Vector Selector in order to take into account the delays displaced to the inputs. We use the *augmented balance equations* to determine the synchronization of the revised component. We take advantage of the delay introduced on the edge controls EC0 .. EC3, by taking them and h^{in}_{SEL} from the input to j_neg. We calculate all inputs relative to h^{in}_{SEL}:

$$\theta = 256$$

$$h^{in}_{SEL} = h^{in}_{J\_NEG} = h^{in}_{SYS} + 256$$

Inputs:

$$
\begin{aligned}
EC_1, EC_3: \quad & \theta \circ (1,0,0) + h^{in}_{SEL} \\
& = (\theta, 0, h^{in}_{SEL}) \\
IOB_{POS}: \quad & (\theta, \{k | 0 \le k < 4\}, 0) + h^{in}_{SYS} + \delta_{STI3220} \\
& = (\theta, \{k | 0 \le k < 4\}, 0) + (h^{in}_{SEL} - 512) + \delta_{STI3220} \\
& = (\theta, \{k | 0 \le k < 4\}, h^{in}_{SEL} + 36) \\
IOB_{NEG}: \quad & (\theta, \{k | 0 \le k < 4\}, 0) + h^{in}_{SYS} + (\delta_{STI3220} + 256) \\
& = (\theta, \{k | 0 \le k < 4\}, 0) + (h^{in}_{SEL} - 512) + (\delta_{STI3220} + 256) \\
& = (\theta, \{k | 0 \le k < 4\}, h^{in}_{SEL} + 292)
\end{aligned}
$$

The latency of this component will still be 4 cycles *after the arrival of the second vector*, in this case the input IOB_{NEG}:

$$
\begin{aligned}
\delta_{SEL} &= \mathbf{max}\{h^{in}_{EC}, h^{in}_{IOB\_POS}, h^{in}_{IOB\_NEG}\} + 4 \\
&= 296
\end{aligned}
$$

Since the inputs have been calculated from the augmented balance equation, it is trival to prove that the equations are satisfied for the subsystem.

5.4.5 Synthesizing the Modified Selector

As the operator of the Vector Selector takes all inputs at the same time, we add physical delays corresponding to the addition (+ 256) on the edge controls EC1 and EC3, and on the output of the Motion Estimation processor j_pos. The complete component with these improved specifications is shown in (Figure 7). We remark two boxes, "split" and "join", which are simply bus rippers. The result of automatic synthesis of the Vector Selector operator with Synopsys is shown in (Figure 8).

6 Conclusion

We have presented a way to rapidly prototype a synchronous system composed of specialized VLSI circuits, starting from a formal specification notation. Our notation can be used to formally specify a system, and statically verify correct synchronization at a Register-transfer level. Our component model provides a consistent format to ease the transition between a block-level Synchronous Data Flow graph, and an RTL description of synchronous circuits. Software tools for this approach, written in SIGNAL or VHDL, allow us to check that proper synchronization is maintained. We have also presented a prototyping example, taken from motion video coding.

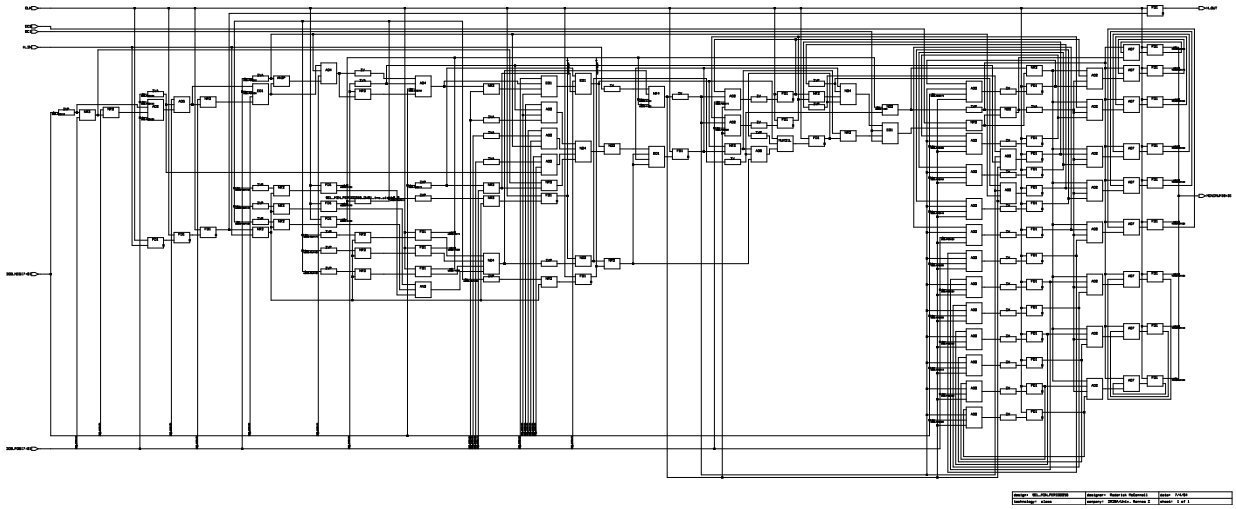

Figure 8: Vector Selector operator after synthesis

Our simulation model offers the advantages typical of multi-level simulations, both in the modularity of a component and in the efficiency of system-level simulation. The components need not be described internally at a gate level, speeding the prototyping process for a system. And the separation of functions make it easy to change the "technology" of the simulation library. It also adapts well to the typical stages in the design process, beginning at a block-level and finishing as circuit-specific.

We emphasize that our approach is targeted at system-level RTL design, and not at individual component design, with the capacity to integrate existing components using their functional and timing specification. In order to develop an RTL model, we require detailed timing information for the VLSI circuits (such as that given on a datasheet) but not the specifics of the internal implementation, making it practical to insert models of commercially available circuits.

The notation and simulation model presented also propose a transition mechanism for adding initialization and termination considerations to a Synchronous Data Flow graph. These considerations are an important part of the detail work needed for a dedicated-component implementation, particularly when working with complex VLSI circuits. The VSDF notation and the model assume that each component performs a certain cyclic operation independent of the data, and that the timing of input and output events can be calculated relative to this cyclic operation using a single clock. This can be seen as a major limitation; however, it simplifies our model and makes precise synchronization possible. We also benefit from the work done on Synchronous Data Flow [20][21] and synchronous circuit design.

Limitations

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 Topology Matrix Γ). 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 [17] [16].

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 (determined as presented in [14]) seems indicated.

Acknowledgments

We would like to thank Alain Kerihuel and Sanjay Rajopadhe for many fruitful discussions and numerous insightful comments, and Mohammed Belhadj and Dominique Lavenier for their assistance in giving this article its form. We would also like to thank Hamid Bougayouu, Jaques Mahe, Arnaud Maitrehenry, and Bruno Pillerel for their behavioral model of the SGS-Thomson STI3220 component.

References

[1] W.B. Ackerman. Data flow languages. *Computer*, 15:15–25, February 1982.

[2] A. Arteri and F. Jutand. A Versatile and Powerful chip for Real-time Motion Estimation. In *SPIE 88*, Boston, 1989.

[3] Bruce Bourbon. System-level Design. *Computer Design*, 19–21, December 1990.

[4] 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.

[5] CCIR. *Paramètres de codage de télévision numérique pour studios.* Technical Report, Committée Cosultatif International pour la Radiotéléphonie, 1990. Recommendation 601-2.

[6] A.L. Davis and R.M. Keller. Data flow program graphs. *Computer*, 15(15):26–47, February 1982.

[7] J.B. Dennis, J.B. Fosseen, and J.P. Linderman. Data FLow Schemas. *Lecture Notes in Computer Science*, 5:187–216, 1972.

[8] 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.*

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

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

[11] A. Jerraya, I. Park, and K. O'Brien. AMICAL: Interactive High-Level Synthesis Environment. In *EDAC 93*, Paris, February 1993.

[12] Alain Kerihuel, Roderick McConnell, and Sanjay Rajopadhye. Des graphes de flots de données synchrones pour le VLSI. Actes de 6ème rencontres francophones du paralláelisme, June 1994.

[13] Alain Kerihuel, Roderick McConnell, and Sanjay Rajopadhye. VSDF: Synchronous Data Flow for VLSI. In *Proceedings of the 37th Midwest Symposium on Circuits and Systems*, Lafayette, Louisiana, August 1994.

[14] Alain Kerihuel, Roderick McConnell, and Sanjay Rajopadhye. *VSDF: Synchronous Data Flow for VLSI.* Technical Report 843, IRISA, Campus de Beaulieu, Rennes, FRANCE, June 1994.

[15] R. Lauwereins, P. Wauters, M. Ad , and J.A. Peperstraete. Geometric Parallelism and Cyclo-static Data Flow in Grape-II. In *Proceedings of the 5th IEEE International Workshop on Rapid System Prototyping*, Grenoble, France, June 1994.

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

[17] 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.

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

[19] 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.

[20] 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.

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

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

[23] 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.

[24] Roderick McConnell, Alain Kerihuel, and Frédéric Raimbault. *Tools for Correct DSP Synchronization.* Technical Report 717, IRISA, April 1993.

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

[26] Kevin O'Brien. Compilation de silicium : du circuit au système. Thèse de l'Institut National Polytechnique de Grenoble, France, March 1993.

[27] Jan Rosseel, Guy Lauwers, and Francky Catthoor. *Array clustering in the context of the CATHEDRAL-IV environment.* Technical Report PPR 3, NANA-I, March 1992.

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

[29] SGS-Thomson. STI3220 motion estimation processor. Data Sheet, 1990.

[30] 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.

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