Validation of Mixed Signal-Alpha Real-Time Systems through Affine Calculus on Clock Synchronisation Constraints

Irina M. Smarandache[1] and Thierry Gautier[2]

[1] The University of Reading, Department of Computer Science
Whiteknights, PO Box 225, Reading RG6 6AY, United Kingdom
Tel.: (44) 118 931 8611 (7626), Fax: (44) 118 975 1994
I.M.Smarandache@reading.ac.uk
[2] IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France
Thierry.Gautier@irisa.fr

Abstract. In this paper we present the affine clock calculus as an extension of the formal verification techniques provided by the Signal language. The affine clock calculus has immediate applicability in real-time system codesign using languages Signal and Alpha. The refinement of mixed Signal-Alpha specifications and their subsequent validation must be supported by the formal clock calculus implemented in the Signal compiler. A Signal program describes a system of clock synchronisation constraints the consistency of which is verified by compilation (clock calculus). Well-adapted in control-based system design, the clock calculus has to be extended in order to enable the validation of Signal-Alpha applications with important numerical calculi. The new affine clock calculus is based on the properties of affine relations induced between clocks by the refinement of Signal-Alpha specifications. Affine relations enable the derivation of a new set of synchronisability rules which represent conditions against which synchronisation constraints on clocks can be assessed. Properties of affine relations and synchronisability rules are derived in the semantical model of traces of Signal. A prototype implementing a subset of the synchronisability rules has been integrated in the Signal compiler and used for the validation of a video image coding application specified using Signal and Alpha.

1 Introduction

Real-time systems, and more generally reactive systems [1], are in continuous interaction with their environment. Therefore, they must respond *in time* to external stimuli. Moreover, real-time systems must be safe, thus one would wish to prove their correctness. Time constraints and safety are two important aspects to be considered in the design of a real-time application.

In order to ensure compliance with time constraints, a real-time system would be implemented entirely in hardware. However, considering the high cost associated with the implementation and maintenance of a hardware system, in some situations a mixed hardware/software implementation may be envisaged. For example, consider an application in which the most time consuming parts, consisting generally of numerical calculi, are conveyed to hardware, while the control which relates these parts is implemented in software and is therefore accessible for further modifications.

Generally speaking, designing a mixed hardware/software real-time system requires a rigorous methodology that comprises methods and tools addressing, among others, specification, formal verification, optimal code generation and hardware synthesis. These aspects are dealt with in *codesign* [2] [3] which denotes the specification, validation and implementation of an application which consists both of a hardware part, in the form of a set of specialised integrated circuits, and a software part implemented on general programmable processors. The idea is to explore various possible implementations of hardware/software systems in order to improve their performance and to ensure the respect of cost constraints.

1.1 Real-Time System Codesign

System codesign is a complex process which can be decomposed into three main activities [2]: 1. The specification of an application at various levels of abstraction, denoted by the term *cospecification*; 2. The validation of the above specifications by formal verification or simulation, also known as *cosimulation*; 3. The hardware/software partitioning of the application, the evaluation of the derived implementation from the point of view of time constraints and cost, executable code generation and hardware synthesis and the production of the interface between hardware and software, i.e *cosynthesis*. Previous work [2] [4] [5] was centred around defining a well-structured methodology for codesign. The starting point has been to describe both hardware and software systems in a unique context, defined by the same language. For this purpose, existing hardware description languages have been extended with facilities for expressing software. For example, the VHDL language has been enhanced with a mechanism for calls of C functions [6]. Conversely, high-level languages like C, C++ or FORTRAN have been extended with mechanisms for the description of hardware systems [7]. The above approaches are interesting as they enable the programming of both the hardware and software parts of a system in a unique framework and their validation by simulation. However, even extensive simulation cannot guarantee that the system is error free. Formal languages are thus an interesting alternative to the specification of a system, the refinement of specifications towards lower levels of abstraction and their validation by formal verification.

Defining a complete methodology of codesign necessitates addressing other relevant problems, most of them concerning system cosynthesis. Among these problems, the automatic partitioning of an application corresponding to its hardware/software implementation, the synthesis of hardware and the generation of optimal code for software implementation.

The work presented in this paper addresses the problems related to the implementation of a platform for the cooperative use of the languages SIGNAL [8]

[9] and ALPHA [10] in real-time system codesign. The complementary properties of SIGNAL and ALPHA can be exploited efficiently if the two languages are used together for the specification, validation and implementation of real-time systems consisting of important numerical treatments related through control parts. As detailed below, SIGNAL and ALPHA offer solutions to some of the problems arising in codesign. The solutions concern the use of formal methods in system cospecification and cosimulation, the automatic synthesis of integrated circuits for algorithms on regular data or the generation of optimal code for the software implementation of the entire system.

1.2 Cospecification and Cosimulation of SIGNAL-ALPHA Systems

In its present form, the SIGNAL language is well-adapted for the design of real-time applications which consist mainly of control. ALPHA is a functional language used for the description and implementation of numerical algorithms on regular multidimensional data. Therefore, SIGNAL and ALPHA can be used jointly for the specification of an application consisting of treatments on multi-dimensional data related by an important control part.

SIGNAL is a *synchronous* language developed for the specification, validation and implementation of real-time systems. The synchronous programming paradigm [11] is based on the following hypotheses: 1. All actions (communications and calculi) in a system have zero *logical* duration; 2. Two or more actions can take place at the same logical instant, such actions being termed 'simultaneous'. Thus, from the point of view of the temporal properties of a system, only succession and simultaneity of instants are of interest, their exact time values have no signification. The temporal properties of a SIGNAL variable are summarised by its *clock*, that is, the set of logical instants where the variable is present and has a value. With each SIGNAL program can be associated a system of synchronisation constraints on clocks, encapsulating the logical properties of the program with respect to time. Properties of values of variables are described through a data dependency graph obtained from the SIGNAL program.

The abstraction of a SIGNAL program into a system of synchronisation constraints on clocks and a data dependency graph enable the verification of the logical properties of an application before taking into consideration the physical properties (time, resources) of a particular implementation.The SIGNAL compiler performs the validation of a SIGNAL specification by verifying the consistency of the clock constraints (clock calculus) and by checking the absence of cycles in the data dependency graph. The *synchronised data flow graph* (SDG) which is constructed during compilation represents the basic data structure for the SIGNAL environment. The latter consists, among others, of a C generator for the functional simulation [12], a formal tool for the verification of dynamic properties [13] and a VHDL code generator to access hardware synthesis tools [14].

The SIGNAL language is well-adapted for control-based applications. The clock calculus reposes on boolean equation resolution methods [12] [15] which have been successfully used for the validation of numerous real-time systems where control is an important aspect. However, validation of numerical applications is not convenient using the clock calculus in its present form. Moreover, SIGNAL does not treat adequately computations on regular data like matrices and vectors. Only simple algorithms can be expressed in SIGNAL and there are no optimisations at the level of the associated VHDL code for hardware synthesis or the generation of code for general architectures.

The ALPHA language constitutes an alternative to SIGNAL concerning the latter aspect. ALPHA has been developed as a functional language based on the formalism of affine recurrence equations over convex polyhedral domains [10]. The language enables the high-level description of regular algorithms on multi-dimensional data, and the transformation of programs towards optimal hardware and/or software implementations. A particular feature of ALPHA represents the transformational environment built around the language for the synthesis of parallel and regular architectures [16].

Given the complementary properties of the languages SIGNAL and ALPHA, we have projected a hybrid framework [3] in which SIGNAL and ALPHA are combined for the codesign of an application consisting of treatments on multi-dimensional data related by an important control part. This is the case for video image processing applications in which similar calculi are applied successively to different sequences of images. An application of this type can be described using ALPHA for the numerical part and SIGNAL for the control. The interfacing between the SIGNAL and ALPHA specifications is realised through C code. The overall specification can be validated by formal verification and/or simulation using jointly tools that are available in the SIGNAL and ALPHA environments.

Two levels of interfacing – functional and architectural – have been defined between SIGNAL and ALPHA for the cospecification and cosimulation of a mixed system. The refinement of a functional SIGNAL-ALPHA specification towards the architectural level introduces a refinement of the temporal indices in SIGNAL. The new time indices are obtained through *affine transformations* on the instants of time of the initial SIGNAL specification. Verification of the re-synchronisation of the clocks obtained by affine transformation necessitates the introduction of the *affine clock calculus* as an an extension of the boolean clock calculus implemented currently by the compiler. The main goal of the work presented in this paper consists in defining and implementing the affine clock calculus. Our study is based on the observation that the existing clock calculus can be expressed as a set of *synchronisability rules*, i.e. conditions against which synchronisation constraints on clocks can be assessed. Affine transformations induce *affine relations* between clocks with essential properties for defining additional synchronisability rules and increase the power of the existing clock calculus. Another original aspect of the work is the use of the semantical model of traces of SIGNAL [8] [17] in order to carry out a complete study of the properties of affine relations.

1.3 Organisation of the Paper

Interfacing SIGNAL and ALPHA in a codesign framework has raised the problem of the validation of refined mixed specifications and the definition of an adequate clock calculus for this purpose, as it is detailed in Section 2. The main contribution of the work herein is presented in Section 3. The main concepts useful in the definition of the affine clock calculus are progressively introduced, there are, the model of traces of the SIGNAL language, the properties of affine relations on clocks, the set of synchronisability rules induced by the latter, and finally the necessary elements for the integration of the affine clock calculus in the compiler. The affine clock calculus represents an important element in defining a complete codesign methodology based on the languages SIGNAL and ALPHA. This aspect is illustrated in Section 4 with the cospecification and cosimulation of a video image coding application. In the same section we discuss in which way the SIGNAL and ALPHA environments may further contribute to the development of a complete codesign methodology based on the two languages. Finally, in Section 5 we present conclusions and perspectives of our work.

2 SIGNAL and ALPHA in Real-Time System Codesign

Figure 1 illustrates the motivation of using SIGNAL and ALPHA in a codesign framework based on the main elements of the environments built around the two languages. Note that SIGNAL and ALPHA programs represent mathematical descriptions of the properties of the corresponding processes. A SIGNAL program can be assimilated to a system of constraints (relations) on clocks and values of the variables. The consistency of the system is verified by compilation. The synchronised data flow graph (SDG) constructed during compilation is used as a support for C or VHDL code generation, or as an entry point for the formal tool SIGALI [13] in charge with verification of dynamic properties.

An ALPHA program represents a system of affine recurrence equations on variables defined as functions over convex polyhedra. The compiler is enhanced with a powerful type checking mechanism based on the complex structure of an ALPHA variable. The result of the compilation is a data structure – a syntax tree – which constitutes the basis for further manipulations in the ALPHA environment. An ALPHA system can be directly translated into C code for functional simulation, or it can be transformed according to a parallel or sequential implementation in hardware or software. As illustrated in Fig. 1, the transformed system in ALPHA0 (a subset of ALPHA) can be translated in C or VHDL, or directly mapped on a netlist [16].

SIGNAL and ALPHA are thus perfectly adapted for the cospecification and cospecification of an application which consists of both control and numerical calculi: the control (sequencing, preemption, etc.) on the numerical treatments is expressed in SIGNAL, while the individual tasks associated with the calculi on regular multi-dimensional data are specified in ALPHA. It is thus necessary to define an interface between the two languages which would enable the communication between the initial SIGNAL and ALPHA specifications, or their refined forms according to the details of a particular implementation.

Two levels of interfacing – functional and architectural — have been considered as detailed below (see Figs. 2 and 3). The approach is based on the fact that both SIGNAL and ALPHA can be translated into C code and executed for functional simulation. Also, SIGNAL offers the possibility to call *external processes*: an external process can be the specification of an algorithm in a language other than SIGNAL. A particular type of external process is a *function* the execution of which is considered instantaneous from the point of view of SIGNAL. A SIGNAL function can be a predefined or a user-defined C function.

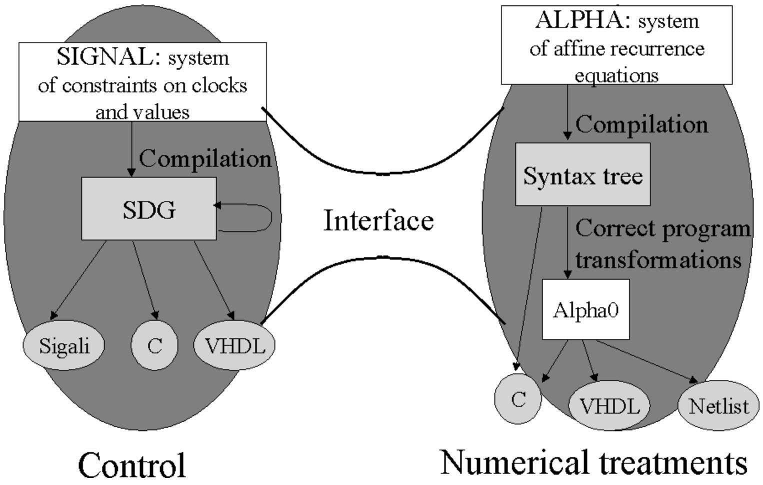

Fig. 1. SIGNAL and ALPHA in system codesign.

2.1 Functional Cospecification and Cosimulation

As detailed in Section 3 when introducing the SIGNAL semantics, the process associated with a SIGNAL program represents a succession of logical instants, with each instant being associated one or more actions involving process variables present at that instant. Figure 2 illustrates the case of a simple process which manipulates matrices A, B and C present at each instant t_i: the values A_i and B_i of the input matrices are multiplied in order to produce C_i; actually, the calculus is considered to be implemented by an external process. Matrix multiplication is a time consuming algorithm, particularly for large matrices. In order to improve the overall performance of the SIGNAL process, one would wish to execute each instance of the matrix multiplication $A_i * B_i$ on a parallel integrated architecture as derived by the ALPHA transformational environment.

Describing matrix multiplication in ALPHA is straightforward. Furthermore, the validation of the overall system consisting of both SIGNAL and ALPHA specifications is performed by interfacing the two languages at the *functional* level as following: 1. The ALPHA specification (a *system* of affine recurrence equations) is translated in executable C code; 2. The C function obtained at step 1 (function *ALPHA_C* in Fig. 2) replaces the initial external process implementing matrix multiplication. Function *ALPHA_C* is considered instantaneous in SIGNAL: the instants of time corresponding to its "real" execution are not visible at the level of the SIGNAL specification. The overall system is thus represented as a SIGNAL specification executing instantaneously the functional description of the ALPHA specification. The system can be validated in the SIGNAL environment by formal verification (compilation, SIGALI) and/or simulation.

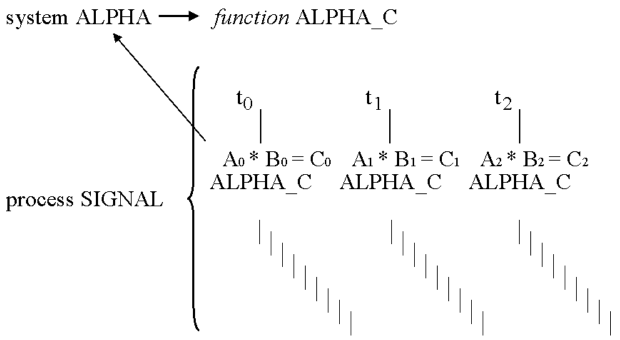

Fig. 2. SIGNAL-ALPHA interface at functional level.

2.2 Implementation-oriented Cospecification and Cosimulation

In order to refine the functional specification of a system corresponding to the details of a particular implementation, it is necessary to consider a finer level of interfacing between the languages SIGNAL and ALPHA as represented in Fig. 3. At the *architectural* level, the specification *ALPHA'* has been derived from *ALPHA* through a sequence of *correct* transformations provided by the ALPHA environment. The transformations guarantee the equivalence of the specifications *ALPHA* and *ALPHA'*. The system *ALPHA'* contains the time index corresponding to a particular scheduling of the operations described by *ALPHA*. In case *ALPHA* exhibits parallelism, the specification *ALPHA'* associates with each operation the index of the processor allocated for execution from a particular parallel architecture.

The SIGNAL specification has to be refined accordingly in order to enable the validation of the overall system given the details of the implementation provided by the ALPHA environment. Therefore, the micro-instants of time in *ALPHA'* are introduced in SIGNAL and the new C function *ALPHA'_C* describes the sequence of operations performed at each micro-instant (see Fig. 3).

Besides taking into consideration the micro-instants of time, the management of communications between the refined SIGNAL and ALPHA specifications intro-

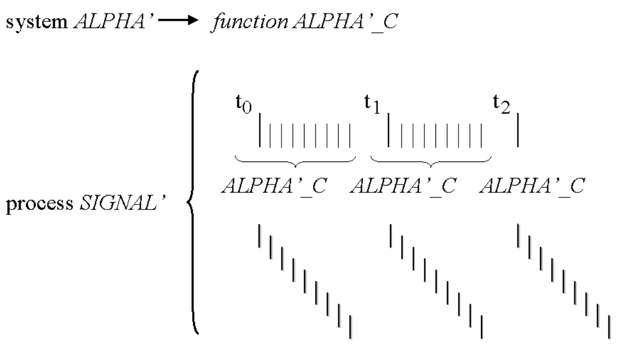

Fig. 3. SIGNAL-ALPHA interface at architectural level.

duces further modifications in the initial SIGNAL program. Furthermore, the C function generated automatically from the *ALPHA'* system has been adapted according to the communications and in order to produce the function *ALPHA'_C*. These aspects are irrelevant for the work presented here and are not detailed further. The rest of the paper is devoted to the *affine clock calculus* introduced in SIGNAL for the validation of mixed SIGNAL-ALPHA specifications at an architectural level of abstraction as described above.

Data manipulated in ALPHA are regular and this property remains valid on micro-instants: in Fig. 3 the number of micro-instants present between any two successive macro-instants t_i and t_{i+1} is constant. In Fig. 4 the refined SIGNAL specification from Fig. 3 is represented as the program $PrgS_1$ with a particular emphasis on micro-instants 6, 15, ..., where the results of the computations performed in ALPHA are made available in SIGNAL. These instants, defining clock c_1, can be described by an *affine* function on the total number of logical instants, i.e. macro and micro-instants. By associating the positive index t with the set of macro and micro-instants, the affine function is $\lambda t(9t + 6)$ in the case of clock c_1 in Fig. 3. As detailed later, the clock c_1 can be obtained by an *affine transformation* applied to clock $c = \{t_1 = 0, t_2 = 9, t_3 = 18, ...\}$ which is equivalent to saying that clocks c_1 and c are in *affine relation.*

2.3 Prelude to the Affine Clock Calculus

Properties on affine relations can be exploited in order to verify that clocks are *synchronisable*, that is, their sets of instants can be identified. This concept is illustrated in Fig. 4. The SIGNAL specification $PrgS_2$ has been obtained through a similar process of refinement as $PrgS_1$. The clock c_2 is in affine relation with clock c which is identical to the clock c in $PrgS_1$. A typical situation is that in which specifications $PrgS_1$ and $PrgS_2$ have been obtained by refinement from different subsystems of the same initial SIGNAL specification. Therefore, initially clocks c_1 and c_2 were identical and the same as clock c (c_1 and c_2 have been derived from ALPHA clocks assumed identical with the clock c at the functional level). In the refined overall specification containing $PrgS_1$ and $PrgS_2$ one would wish to re-identify clocks c_1 and c_2. Whether their identification is possible or not can be decided by analysing the properties of the affine relations existing between c_1 and c, and respectively c_2 and c. In the case of Fig. 4 the affine relations satisfy the necessary conditions for the identification of c_1 and c_2. A formal motivation is provided in Section 3; the intuition behind is that clocks c_1 and c_2 are in the same *relative position* with respect to clock c.

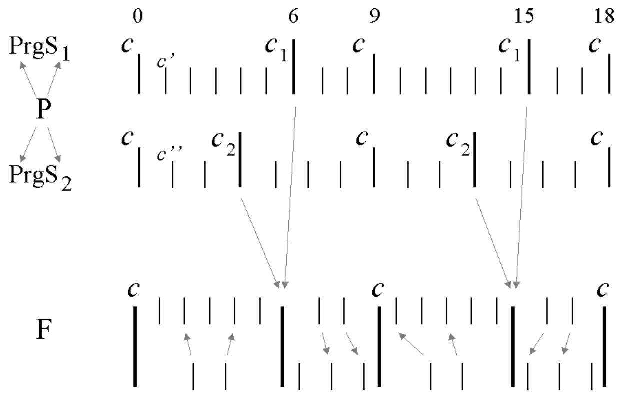

Fig. 4. Synchronisable clocks in the context of codesign with SIGNAL and ALPHA.

The aim of the affine clock calculus discussed in Section 3 is to define necessary and sufficient conditions for clock synchronisability based on the properties of affine relations on clocks. The affine clock calculus has been defined as an extension to the existing *boolean* clock calculus. It consists of a set of *synchronisability rules* derived in the semantical model of *traces* of SIGNAL. Section 3 introduces the semantics of traces in SIGNAL. The most relevant properties of the affine relations are then gradually presented, together with the motivation of each step of the study performed. Finally, we present a subset of the synchronisability rules induced thereafter and we conclude with the main elements necessary for their implementation in the SIGNAL compiler.

3 Affine Calculus on Clocks in SIGNAL

Figure 4 illustrates a different perspective of a SIGNAL process with the intention of introducing the reader to the semantics of traces (flows) [8] [17] of the SIGNAL language. SIGNAL variables are finite or infinite sequences of values called *signals*. With each signal is associated a sequence of logical instants where the values of the signal are present, denoted as signal's *clock*. SIGNAL programs manipulate signals implicitly or explicitly by imposing constraints on their values and clocks. Clocks can be identical (also known as *synchronous*), subsets of other clocks defined as samplings by boolean conditions or clock intersections ($\wedge$), or unions ($\vee$) and/or differences ($\backslash$) of clocks. When no condition is explicitly or implicitly stated on a pair of clocks, they are independent.

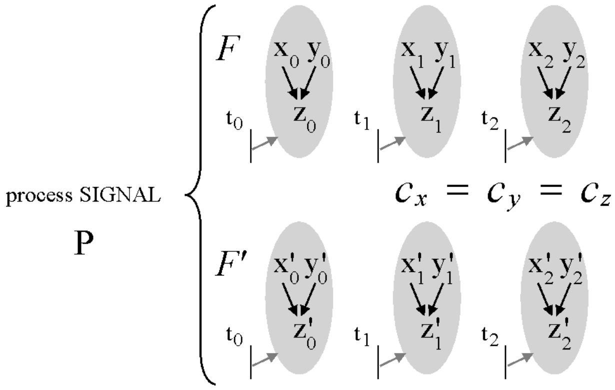

Fig. 5. Illustration of SIGNAL semantics of traces.

The set of clock constraints is resumed in a system of equations the consistency of which is verified by compilation. During this process, known as *clock calculus*, a total order is derived on the set of clocks whenever possible. The maximal clock, if unique, contains the other clocks of the program and is used for the sequencing of the actions associated with each logical instant. A program with more than one maximum is under-specified from the point of view of its clocks and cannot be executed.

Being a real-time system, a SIGNAL program is in continuous interaction with its environment. Input values are transformed corresponding to the actions of a given specification and the results are provided to the environment. Incoming values of input signals may vary from one execution of the SIGNAL program to another, and correspondingly different sequences of values are produced for the output signals. This situation is illustrated in Fig. 5 for the case of a program manipulating inputs x and y and providing output z depending on the values of x and y. In case z is the addition of x and y, signals x, y and z are implicitly constrained by the $+$ operator in SIGNAL to have the same clocks $c_x = c_y = c_z$.

The configurations F and F' illustrated in Fig. 5 correspond to two different executions of the SIGNAL program involving sequences x_i, y_i and z_i and respectively x'_i, y'_i and z'_i. The set of all possible configurations or *traces* define completely the *process* P associated with the SIGNAL program. A *flow* is a trace with the property that with each logical instant there is associated at least one action; the traces represented in Fig. 5 respect this condition. Terms 'process' and 'program' are used interchangeably in some situations. However, in the sequel, a process refers exclusively to the set of flows induced by a SIGNAL specification (program).

The semantics of a SIGNAL specification is defined by the corresponding process P. Each flow F in P is constrained by a system of equations (relations) on the clocks and values of signals manipulated by F. Equations on values can be further expressed in the abstract form of a data dependency graph. An example of a data dependency graph is illustrated in Fig. 5 for the $+$ operator: $z_i = x_i + y_i, \forall i$. The system of synchronisation constraints on clocks and the data dependency graph represent the abstraction of a SIGNAL specification on which any formal verification is based. The compiler verifies the absence of cycles in the data dependency graph and the consistency of the system of equations on clocks. The latter verification, called clock calculus, is based on a system of *synchronisability rules* as detailed in the next section.

3.1 Clock calculus & Synchronisability

The clock calculus is equivalent to the resolution of a system of clock equations, e.g.

$$\begin{array}{l} c = c_1 \\ c' = (c_1 \wedge c_2) \vee c_1 \\ c = c' \end{array} \tag{1}$$

This is a particular example of a system derived from a SIGNAL program which manipulates clocks c, c', c_1 and c_2. The resolution of the system is performed by *triangularisation* [12] [15]: each equation of (1) is oriented from right to left in order for the expression on the right to become the definition of the clock on the left. This orientation is not possible if the clock on the left side of an equation has already a definition. In this case, the formula on the right side must be proven to be equivalent to the previous definition. Considering the system (1) above, the first equation can be oriented and the clock c_1 becomes the definition of the clock c. The second equation is oriented in the same manner and the clock c' is defined by the formula $(c_1 \wedge c_2) \vee c_1$. The last equation of the system (1) cannot be oriented as c has already the formula c_1 as definition. Therefore, formulas c_1 and $(c_1 \wedge c_2) \vee c_1$ must be demonstrated to be equivalent.

The resolution of a system of clock equations derived from a SIGNAL program can thus be reduced to a set of equivalences like above. The space of clocks is a *boolean lattice* [18] the properties of which are extensively used for the proof of equivalences. These properties are resumed by the axioms of the boolean lattice:

$$\begin{array}{l} c_1 \wedge c_2 = c_2 \wedge c_1 \\ c_1 \vee (c_2 \wedge c_3) = (c_1 \vee c_2) \wedge (c_1 \vee c_3) \\ (c_1 \wedge c_2) \vee c_1 = c_1 \\ \ldots \end{array} \tag{2}$$

Note axiom $(c_1 \wedge c_2) \vee c_1 = c_1$ which guarantees the consistency of the system (1). Moreover, given a boolean signal Cd, its clock noted $\hat{Cd}$ can be partitioned into the clock $[Cd]$ where the signal Cd is present and true and the clock $[\neg Cd]$ where Cd is present and false. The relation between clocks $\hat{C}d$, $[Cd]$ and $[\neg Cd]$ can be resumed by the *partition equations* below:

$$\begin{array}{l} [Cd] \vee [\neg Cd] = \hat{C}d \\ [Cd] \wedge [\neg Cd] = \emptyset \end{array} \tag{3}$$

The axioms of the boolean lattice and the partition equations induce on the space of clocks a lattice of an order $\preceq$ "coarser" than the order $\leq$ of the boolean lattice [12]. Clocks can be boolean formulas constructed either with samplings by boolean conditions $[Cd]$, $[\neg Cd]$ or with free variables of the boolean lattice. The properties of the lattice of order $\preceq$ are actually used during the triangularisation of any system of clock equations.

The first step in defining the new affine clock calculus has been to establish the connection between the axioms of the lattice $\preceq$ and the synchronisability concept. Therefore, we have demonstrated that systems (2), (3) and equations induced by them are equivalent to a system of *synchronisability rules* which can be further extended using the properties of affine relations between clocks. Moreover, in their present form, the synchronisability rules used by the boolean clock calculus described above are actually *synchronism rules*; this is not the case for the affine clock calculus as detailed later.

Synchronisability and synchronism are defined in the semantics of flows of the SIGNAL language as following. Clocks c and c' are synchronisable in the process P, denoted by $c \overset{P}{\odot} c'$, if there *exists* a flow F in P in which c and c' are identical:

$$c \overset{P}{\odot} c' \Leftrightarrow \exists F \in P, c \overset{F}{=} c' \tag{4}$$

Whenever the above property is valid *for each* flow F in P, the clocks c and c' are said to be synchronous in P, which is noted by $c \overset{P}{=} c'$. Formally, this definition can be expressed as follows:

$$c \overset{P}{=} c' \Leftrightarrow \forall F \in P, c \overset{F}{=} c' \tag{5}$$

Unless explicitly constrained through the SIGNAL program, clocks c and c' are completely independent in the associated P process. Therefore, their relative position can be such that in some flows F in P they are identical, while in some other flows F' in P their instants interleave in an arbitrary manner. Obviously, if c and c' are independent in P, they are synchronisable. When the relative position of clocks c and c' is constrained by the SIGNAL specification, flows F in P are subsequently constrained and the synchronisability of c and c' depends on these constraints.

Consider a process P derived from a SIGNAL program *Prg* in which clocks c and c' are defined by

$$\begin{array}{l} c = c_1 \\ c' = (c_1 \wedge c_2) \vee c_1 \end{array} \tag{6}$$

(the first two equations of the system (1)). Program *Prg* may be transformed into *Prg*$'$ in which an additional constraint has been expressed on clocks c and c': $c = c'$. The system of clock equations associated with *Prg*$'$ and the associated process P' is (1). Verifying that clocks c and c' are equivalent in P' is the same as testing the synchronisability of c and c' in P. Clocks c and c' are synchronisable in P if there exists a flow $F \in P$ satisfying $c \stackrel{F}{=} c'$. This is true given the axiom $(c_1 \wedge c_2) \vee c_1 \stackrel{P}{=} c_1$ valid for an arbitrary process P, which is further equivalent to $(c_1 \wedge c_2) \vee c_1 = c_1$ when the process P is implicit. Consequently, for the process P considered above $(c_1 \wedge c_2) \vee c_1 \stackrel{P}{\odot} c_1$, as synchronism implies synchronisability. It can be concluded that the process P' in which clocks c and c' are constrained to be synchronous ($c = c'$ or $c \stackrel{P'}{=} c'$) is consistent from the point of view of the constraints expressed on its clocks. Process P' is actually defined by a subset {F'} of the set of flows $\{F\} = P$ satisfying $c \stackrel{F'}{=} c'$.

The synchronisability rule $(c_1 \wedge c_2) \vee c_1 \stackrel{P}{\odot} c_1$, with P arbitrary, is equivalent to $(c_1 \wedge c_2) \vee c_1 \stackrel{P}{=} c_1$ and can be further written $(c_1 \wedge c_2) \vee c_1 = c_1$. In the same manner one can demonstrate that systems (1), (2) and equations induced by them are equivalent to a set of synchronism rules which are also synchronisability rules by definition of the property of synchronism.

The existing set of synchronisability rules on which the boolean clock calculus is based can be extended using the properties of affine relations on clocks. Figure 4 illustrates this idea with the schematic representation of the process P associated with the SIGNAL specification consisting of subprograms Prg_1 and Prg_2. Flows in P are obtained by composition of flows in P_1 and P_2 (processes corresponding to Prg_1 and Prg_2) in which c and c_1, and respectively c and c_2, are in affine relation. c_1 and c_2 are synchronisable in P as there exists flows $F \in P$ in which c_1 and c_2 are identical ($c_1 \stackrel{F}{=} c_2$); a particular flow F satisfying this condition have been represented in Fig. 4.

Clocks c_1 and c_2 in Fig. 4 are synchronisable because of the particular characteristics of the affine relations on c and c_1, respectively c and c_2 *associated with* the *flows* F (in P) and with P. The next section defines the affine relation associated with a flow and a process and presents the motivation of studying affine relations in order to induce new synchronisability rules.

3.2 Affine Relations in SIGNAL

Consider[1] $n, d \in \mathbb{N}^*$ and $\varphi \in \mathbb{Z}$ fixed. Clocks c and c' are in (n, φ, d)-affine relation in the flow F, noted $c \; \mathcal{R}^F_{(n,\varphi,d)} \; c'$ or $(c, c') \in \mathcal{R}^F_{(n,\varphi,d)}$, if the relative position of c and c' in F can be induced by an (n, φ, d)-*affine transformation*. Clock c' is the result of an (n, φ, d)-affine transformation applied to clock c if it can be obtained through steps 1,2 and 3 as following: 1. Inserting $n - 1$ fictive instants between any two successive instants of c and $-\varphi$ fictive instants before the first instant of c when φ is negative; 2. Grouping in T the set of instants

[1] The set of strictly positive integers is noted $\mathbb{N}^*$; the total set of integers is $\mathbb{Z}$.

of c and the set of fictive instants introduced at step 1; 3. Defining clock c' as the set of instants $\{dt + \varphi \mid t \in T\}$. In Fig. 6 we have represented flows F and F' exhibiting the same $(9, 6, 2)$-affine relation on clocks c and c_1, respectively c and c_2. It can be observed, for example, that clock c_1 can be constructed by an affine transformation parametrised by $(9, 6, 2)$ applied to clock c.

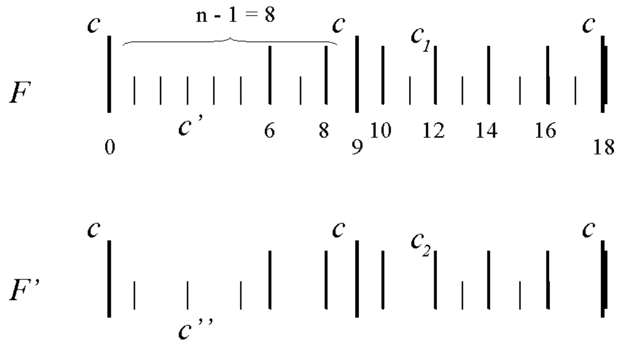

Fig. 6. $c\ \mathcal{R}^F_{(9,6,2)}\ c_1$ and $c\ \mathcal{R}^{F'}_{(9,6,2)}\ c_2$.

Clocks c and c' are in (n, φ, d)-affine relation in process P, noted $c\ \mathcal{R}^P_{(n,\varphi,d)}\ c'$ or $(c, c') \in \mathcal{R}^P_{(n,\varphi,d)}$, if they are in (n, φ, d)-affine relation in each flow F of P, i.e. $c\ \mathcal{R}^F_{(n,\varphi,d)}\ c'$, $\forall F \in P$. Flows and processes are defined formally over the set of variables they manipulate. For a given set A, a flow F on A is a member of the set of total flows on A noted $\mathcal{F}_A$ $(F \in \mathcal{F}_A)$; in a similar manner, a process P on A belongs to the set of processes on A, i.e. $P \in \mathcal{P}_A$. Associating a finite set of variables with flows and processes enables the characterisation of the corresponding affine relations as finite sets:

$$\forall F \in \mathcal{F}_A,\ \mathcal{R}^F_{(n,\varphi,d)} = \{(c, c') \in A \times A \mid c\ \mathcal{R}^F_{(n,\varphi,d)}\ c'\} \tag{7}$$

$$\forall P \in \mathcal{F}_A,\ \mathcal{R}^P_{(n,\varphi,d)} = \{(c, c') \in A \times A \mid c\ \mathcal{R}^P_{(n,\varphi,d)}\ c'\} \tag{8}$$

Consider a SIGNAL program manipulating exclusively clocks c, c_1 and c_2 such that c and c_1 are in (n_1, φ_1, d_1)-affine relation and c and c_2 are in (n_2, φ_2, d_2)-affine relation. The SIGNAL program defines the process $P \in \mathcal{P}_{\{c,c_1,c_2\}}$ as the set of flows $F \in \mathcal{F}_{\{c,c_1,c_2\}}$ satisfying $c\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ c_1$ and $c\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ c_2$:

$$P = \{F \in \mathcal{F}_{\{c,c_1,c_2\}} \mid c\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ c_1, c\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ c_2\} \tag{9}$$

By definition of process P it results $c\ \mathcal{R}^P_{(n_1,\varphi_1,d_1)}\ c_1$ and $c\ \mathcal{R}^P_{(n_1,\varphi_1,d_1)}\ c_1$. With the above hypothesis, c_1 and c_2 are synchronisable in P if there exists $F \in P$ satisfying $c_1 \stackrel{F}{=} c_2$. Clocks c_1 and c_2 being identical in F implies $c\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ c_2$,

$c\,\mathcal{R}^F_{(n_2,\varphi_2,d_2)}\; c_1$ and therefore, $\mathcal{R}^F_{(n_1,\varphi_1,d_1)} = \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \{(c,c_1),(c,c_2)\}$. In conclusion, it is necessary to determine the conditions triplets (n_1,φ_1,d_1) and (n_2,φ_2,d_2) must satisfy in order for the relations $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ and $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$ to be equivalent.

This example demonstrates that affine relations influence clock synchronisability. The most important step in defining the affine clock calculus consists thus in a complete exploration of the space of affine relations; this work is exposed in Section 3.3. The study is performed in the semantical model of flows of the SIGNAL language. The main operations that can be performed on flows and processes are briefly described and an important lemma synthesising the properties of these operations is stated in conclusion. Next, the main aim is to define an equivalence relation on the space of affine relations on flows. With this in mind, the first question to be answered is whether the space is closed under the operations that can be performed on affine relations. These are either union, intersection or difference induced by the homonym operations on clocks, or operations specific to affine relations like inverse and composition. The properties can be directly transposed to affine relations on processes and are further used to derive a new set of synchronisability rules on clocks.

3.3 Properties on Affine Relations & Synchronisability Rules

SIGNAL flows and processes are part of a general semantical model of traces (flows) the main principles of which are thoroughly described in [17]. Due to lack of space, in this paper we present the main elements of the model indispensable in the sequel.

The semantics of traces. Consider a set A of variables which represent finite or infinite sequences of typed values. Variables are usually denoted as *ports* and they are equivalent to SIGNAL variables (signals). The set of all possible configurations or flows defined on A is denoted by $\mathcal{F}_A$. Subsets of flows from $\mathcal{F}_A$ can be grouped in processes which are members of the set $\mathcal{P}_A$ of all processes that can be defined on A. A SIGNAL program manipulating the set A of variables defines a process $P \in \mathcal{P}_A$; each flow $F \in P$ is a member of $\mathcal{F}_A$ ($F \in \mathcal{F}_A$) and satisfies additional constraints imposed by the SIGNAL operators on the clocks and values of the variables in A.

SIGNAL disposes of four basic operators (SIGNAL kernel) which are sufficient for the construction of each other SIGNAL program regardless its complexity. Kernel operators are combined through composition and restriction in order to build programs. The composition and restriction of programs have induced similar operations on processes and flows. Given a set of variables $A' \subseteq A$, the restriction of $P \in \mathcal{P}_A$ to A' (the projection of P on A') contains the flows $F \in P$ manipulating exclusively variables of A':

$$\Pi_{A'}(P) = \{F' \in \mathcal{F}_{A'} \mid F' = \Pi_{A'}(F), \forall F \in P\} \tag{10}$$

Similarly can be defined the composition of processes $P_1 \in \mathcal{P}_{A_1}$ and $P_2 \in \mathcal{P}_{A_2}$, with A_1, A_2 arbitrary set of variables:

$$P_1 \mid P_2 = \{F \in \mathcal{F}_{A_1 \cup A_2} \mid \Pi_{A_1}(F) \in P_1, \Pi_{A_2}(F) \in P_2\} \tag{11}$$

The restriction of a flow $F \in \mathcal{P}_A$ to a subset of variables $A' \subseteq A$, noted $\Pi_{A'}(F)$, is not detailed here. Intuitively, the flow $\Pi_{A'}(F)$ is obtained from F as following: at each instant of F, only the actions involving variables from A' are considered; in case there is no such an action, the instant is eliminated from $\Pi_{A'}(F)$ and the next instant is considered.

Additional operations my be defined on flows and processes; however, we have restricted our study to the operations of composition and restriction given their relevance in the context of the SIGNAL language. An extensive study of their properties can be found in [17], together with a lemma of particular relevance for the integration of the affine and boolean clock calculus:

Lemma 1. $\forall P \in \mathcal{P}_{A_1}$, $\forall Q \in \mathcal{P}_{A_2}$, $A_2 \subseteq A_1$,

$$\Pi_{A_2}(P) \subseteq Q \Leftrightarrow P \mid Q = P \tag{12}$$

Intuitively, in Lemma 1 process Q expresses a property valid in P. In other words, every property valid in Q is valid also in P.

Properties on affine relations. Operations specific to relations in general, like inverse $()^{-1}$ and composition $*$, can be applied to affine relations [19]. As an example, consider the process $P \in \mathcal{P}_{\{c,c_1,c_2,c_3\}}$ with clocks c, c_1, c_2 and c_3 satisfying $c\ \mathcal{R}^P_{(n_1,\varphi_1,d_1)}\ c_1$, $c_1\ \mathcal{R}^P_{(n_2,\varphi_2,d_2)}\ c_2$ and $c\ \mathcal{R}^P_{(n_3,\varphi_3,d_3)}\ c_3$. Obviously, it results $c\ \mathcal{R}^P_{(n_1,\varphi_1,d_1)} * \mathcal{R}^P_{(n_2,\varphi_2,d_2)}\ c_2$ and the synchronisability of c_2 and c_3 depends on properties specific to relation composition. In case the space of affine relations is closed with respect to composition, the test of clock synchronisability reduces itself to the verification of the equivalence of affine relations.

Affine relations can be further combined through union $\cup_r$, intersection $\cap_r$ and difference $\setminus_r$ induced by the homonym operations on clocks ($\vee$, $\wedge$, $\setminus$). A similar argument as before conducts to the necessity of studying closing properties of these operators with respect to the space of affine relations.

Here is a brief presentation of the main steps and results obtained in the study of affine relations.

Equivalence of Affine Relations. Identity of affine relations is an equivalence which depends exclusively on the triplets (n, φ, d) associated with each relation [20]. Moreover, it has been demonstrated that the relationship between these triplets, noted $\sim$, is an equivalence.

Proposition 1.

$$\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')},\ \ \forall\ F \in \mathcal{F}_A \Leftrightarrow (n, \varphi, d)\ \sim\ (n', \varphi', d')$$

Testing the equivalence $\sim$ is based on a integer calculus which may become quickly fastidious. Therefore, we have defined a canonical form $(n_{FC}, \varphi_{FC}, d_{FC})$ for a triplet (n, φ, d) which reduces the complexity of the above verification [21].

Canonical Form. The canonical form of a triplet (n, φ, d) is based on the greatest common divisor of n and d, noted $gcd(n, d)$, and is defined as follows:

Proposition 2.

$$\begin{array}{ll} a) & G \mid \varphi \Rightarrow (n_{FC}, \varphi_{FC}, d_{FC}) = (\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G}) \\ b) & G \not| \varphi \Rightarrow (n_{FC}, \varphi_{FC}, d_{FC}) = (2\frac{n}{G}, (2\left[\frac{\varphi}{G}\right] + 1), 2\frac{d}{G}) \end{array}$$

with $G = gcd(n, d)$ *and* $[x]$ *the integer part of* $x \in \mathbb{N}$.

Consequently, the canonical form of $\mathcal{R}^F_{(n,\varphi,d)}$ is $\mathcal{R}^F_{(n_{FC},\varphi_{FC},d_{FC})}$ and the verification of the identity of two affine relations is reduced to a simple verification if two triplets of integers are identical:

Proposition 3.

$$\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')} \Leftrightarrow (n_{FC}, \varphi_{FC}, d_{FC}) = (n'_{FC}, \varphi'_{FC}, d'_{FC})$$

Operations on affine relations. In case any expression on affine relations reduces itself to an affine relation, the verification of clock synchronisability would consists only in a test of equivalence applied to affine relations as above. Or, it has been observed that this was not the case in general. The property remains true for the inverse of an affine relation. Also, the affine relation $\mathcal{R}^F_{(1,0,1)}$ is element neutral with respect to composition. However, the property is lost when dealing with composition. The composition of two general affine relations $\mathcal{R}^F_{(n,\varphi,d)}$ and $\mathcal{R}^F_{(n',\varphi',d')}$ does not generally produce an affine relation. Nevertheless, it has been possible to identify in the space of the affine relations $\mathcal{R}^F_{(n,\varphi,d)}$ a subspace consisting of relations of the form $\mathcal{R}^F_{(1,\varphi,d)}$, with $\varphi \geq 0$, in which the property remains true. Consequently, our study followed two disjoint directions conducting to two different sets of synchronisability rules.

Properties of affine relations $\mathcal{R}^F_{(1,\varphi,d)}$*, with* $\varphi \geq 0$. It has been demonstrated [17] that the space of affine relations $\mathcal{R}^F_{(1,\varphi,d)}$, closed under composition, does not have the same property with respect to union $\cup_r$ and difference $\backslash_r$ of affine relations. As a consequence, the space of study has been extended to arbitrary expressions constructed with affine relations of the form $\mathcal{R}^F_{(1,\varphi,d)}$ using composition, union, intersection and difference. Obviously, this space is closed under these operations. Given its complexity and the necessity for efficient verification algorithms, the question of the existence of a canonical form in this space appears. The idea is to provide a canonical form based exclusively on the $\cup_r$ operator, as it can be demonstrated that any expression in this space can be rewritten as an union of affine relations $\mathcal{R}^F_{(1,\varphi,d)}$. Or, the infinite number of possibilities in which an affine relation $\mathcal{R}^F_{(1,\varphi,d)}$ can be rewritten as a union of affine relations of the same type excluded the possibility of defining a canonical form. A *normal form relative* to a given subspace has been defined [17], but there is still a large degree of freedom in the choice of the subspace to make possible the definition of a canonical form.

Properties of general affine relations $\mathcal{R}^F_{(n,\varphi,d)}$. The identification of the set of triplets (n,φ,d) and (n',φ',d') such that $\mathcal{R}^F_{(n,\varphi,d)}\ op_r\ \mathcal{R}^F_{(n',\varphi',d')}$, with $op_r \in \{*, \cup_r, \cap_r, \backslash_r\}$, is affine, is a complex problem. In [17] we propose a subset of triplets (n,φ,d) and (n',φ',d') for which the property is true for composition. Computing the elements of the subset starting from its present characterisation is an NP-complete problem. Reducing the complexity of the above computation remains an open problem. An alternative resides in the use of heuristic search methods.

Synchronisability rules. Based on the properties of affine relations of the form $\mathcal{R}^F_{(1,\varphi,d)}$, with $\varphi \geq 0$, a set of synchronism rules has been induced. The main operations on $\mathcal{R}^F_{(1,\varphi,d)}$ have been completely defined and these results have been directly transposed to affine relations on processes. As an example, consider the process $P \in \mathcal{P}_{\{c,c_1,c_2,c_3\}}$ with clocks c, c_1, c_2 and c_3 satisfying $c\ \mathcal{R}^P_{(1,\varphi_1,d_1)}\ c_1$, $c_1\ \mathcal{R}^P_{(1,\varphi_2,d_2)}\ c_2$ and $c\ \mathcal{R}^P_{(1,\varphi_3,d_3)}\ c_3$. It has been demonstrated that $\mathcal{R}^P_{(1,\varphi_1,d_1)} * \mathcal{R}^P_{(1,\varphi_2,d_2)} = \mathcal{R}^P_{(1,\varphi_1+d_1\varphi_2,d_1d_2)}$. From this definition it results that c_2 and c_3 are synchronisable iff $\mathcal{R}^P_{(1,\varphi_1+d_1\varphi_2,d_1d_2)} = \mathcal{R}^P_{(1,\varphi_3,d_3)}$. With Propositions 2 and 3, $\mathcal{R}^P_{(1,\varphi_1+d_1\varphi_2,d_1d_2)}$ and $\mathcal{R}^P_{(1,\varphi_3,d_3)}$ are equivalent iff $(1,\varphi_1+d_1\varphi_2,d_1d_2)$ and $(1,\varphi_3,d_3)$ are identical, that is, $\varphi_1+d_1\varphi_2=\varphi_3$ and $d_1d_2=d_3$. This result is further resumed by the synchronism rule below:

Proposition 4. *$\forall P \in \mathcal{P}_{\{c,c_1,c_2,c_3\}}$ for which c, c_1, c_2 and c_3 satisfy $c\ \mathcal{R}^P_{(1,\varphi_1,d_1)}\ c_1$, $c_1\ \mathcal{R}^P_{(1,\varphi_2,d_2)}\ c_2$ and $c\ \mathcal{R}^P_{(1,\varphi_3,d_3)}\ c_3$, the following equivalences are verified:*

$$c_2 \overset{P}{\odot} c_3 \Leftrightarrow \left\{ \begin{array}{l} \varphi_1 + d_1\varphi_2 = \varphi_3 \\ d_1d_2 = d_3 \end{array} \right\} \Leftrightarrow\ c_2 \overset{P}{=} c_3$$

In Fig. 7 the particular case $\varphi_1 = 6$, $d_1 = 2$, $\varphi_2 = 1$, $d_2 = 2$, and $\varphi_3 = 8$, $d_3 = 4$ is illustrated. It can be observed that clock c_1 is an *affine sampling* of phase φ_1 and period d_1 on clock c. Clock c_2 is defined similarly by an affine sampling of parameters φ_2 and d_2 on c_1. Or, the same clock c_2 can be obtained by an affine sampling of φ_3 and d_3 on c; the clock c_3 constructed in this manner is synchronous, and therefore synchronisable, with c_2.

The results concerning the equivalence of general affine relations $\mathcal{R}^F_{(n,\varphi,d)}$, resumed by Propositions 1, 2 and 3, and the composition of these relations, permitted the derivation of a set of synchronisability rules which have been sufficient for the validation of the Signal-Alpha interface introduced in Section 2. Further work should be dedicated to the study of the union $\cup_r$, intersection $\cap_r$ and difference $\backslash_r$ of general affine relations.

3.4 Implementation of the Affine Clock Calculus

A prototype implementing the synchronisability rules introduced in Section 3.3 has been integrated with the existing clock calculus and used for the validation

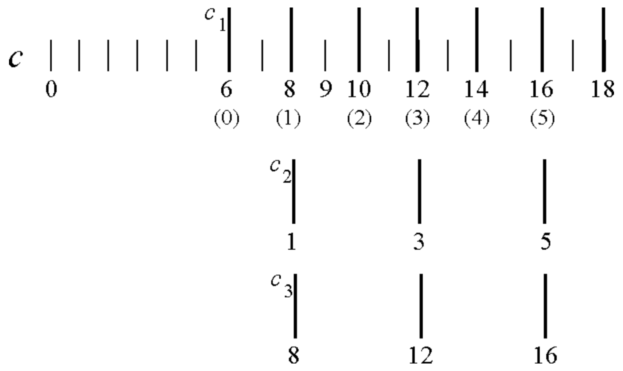

Fig. 7. Illustration of Proposition 4.

of the SIGNAL-ALPHA interface on a video image coding application described in Section 4. In Section 3.1 we have demonstrated that the existing (boolean) clock calculus relies, on one hand, on the properties of the lattice $\preceq$ existing on the space of clocks and, on another hand, is equivalent to a system of synchronisability rules. The reverse strategy has been applied for the implementation of the affine clock calculus. By choosing an appropriate implementation of a general affine relation $\mathcal{R}_{(n,\varphi,d)}$ as detailed in [17], clock equations are expressed exclusively in terms of *affine clocks*, i.e. affine samplings of other clocks. Thus, the order $\preceq_{aff}$ defined by

$$\preceq_{aff} = \{(c_1, c_2) |\ \exists \varphi_i \geq 0, d_i > 1, \mathcal{R}_t = \mathcal{EXP}(\ldots, \mathcal{R}_{(1,\varphi_i,d_i)}, \ldots), c_1\ \mathcal{R}_t\ c_2\} \tag{13}$$

induces on the space of *affine clocks* a lattice structure. A system of equations on affine clocks associated with a SIGNAL program by compilation is solved by a triangularisation method as defined in Section 3.1. The only difference appears when the equivalence of two clock expressions has to be demonstrated: in this case, synchronisability rules deduced in Section 3.3 are applied. Finally, the integration of the affine and boolean clock calculus is mathematically founded by Lemma 1: each synchronisability rule is deduced in a process $Q \in \mathcal{P}_{A_2}$ and has to be used in a larger context $P \in \mathcal{P}_{A_1}$, with $A_2 \subseteq A_1$, satisfying $\Pi_{A_2}(P) \subseteq Q$.

4 Application

The affine clock calculus has been used for the validation of the codesign principles exposed in Section 2. We have considered an application for the coding of sequences of images at 34 Mbits/s [22]. A system of this type consists of a set of numerical treatments applied iteratively on sequences of images of the same dimension. Images are divided into luminance and chrominance blocs and treatments are subsequently applied to each bloc. In Fig. 8 we have represented

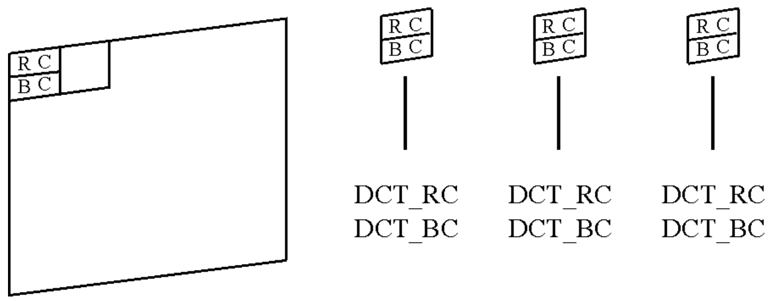

Fig. 8. The structure of a generic video image coding application.

a chrominance bloc which is further divided corresponding to the red and blue chrominance. Numerical treatments consist mainly of algorithms for inter and intra-image coding which require operations like discrete cosine transformation (DCT). This application contains an important control part, which has been programmed in SIGNAL, and operations like the DCT, which have been expressed via a system of affine recurrence equations in ALPHA. The application has been specified and simulated at both functional and architectural levels as described in Section 2.

Most of the operations involved in image coding applications are critical from the point of view of time of execution or resources. In this conditions a codesign approach could be considered. The affine clock calculus represents an important element in defining a complete codesign methodology based on the languages SIGNAL and ALPHA. Besides the cospecification and cosimulation of an application, using SIGNAL and ALPHA in a codesign framework is interesting since it offers solutions to other codesign problems like the automatic synthesis of specialised circuits for regular algorithms, or the generation of optimal code for the software implementation of both calculi and control. Concerning the latter, one might consider the hardware/software partitioning of an application corresponding to the partitioning into SIGNAL and ALPHA subsystems. Therefore, ALPHA processes would be implemented in hardware with the support of the ALPHA transformational environment for automatic synthesis, while SIGNAL processes describing control would be translated into C code for general purpose architectures. However, the proposed partitioning is not unique and automatic hardware/software partitioning remains an open problem, as it is the implementation of the hardware/software interface.

5 Conclusion

The joint use of the languages SIGNAL and ALPHA in hardware/software codesign has introduced the problem of the validation of mixed SIGNAL-ALPHA specifications both at the functional and architectural level. The refinement of SIGNAL-ALPHA specifications towards the architectural level and their subsequent validation necessitates the extension of the formal clock calculus implemented in the SIGNAL compiler. The paper presents the new affine clock calculus based on the properties of affine relations induced between clocks by the refinement of SIGNAL-ALPHA specifications. The properties of affine relations are studied in the semantical model of traces of the SIGNAL language, but can be extended to any general model with similar characteristics. Based on this study, a new set of synchronisability rules is defined and integrated with the set already implemented by the existing formal clock calculus.

The affine clock calculus is relevant for the definition and implementation of a codesign methodology using the languages SIGNAL and ALPHA. Techniques for real-time system validation (formal verification, simulation) available in the SIGNAL and ALPHA environments can be used for cospecification and cosimulation. Also, both environments dispose of tools for automatic generation of optimal implementations which can be used in a complementary manner for hardware synthesis and/or implementation on general architectures. Further work should be centred on the complete integration of the languages SIGNAL and ALPHA making thus possible the use of the most adapted formalism and environment for a given application.

References

1. Benveniste A., Berry G.: 'Special section on another look at real-time programming', *Proceeding of the IEEE*, September 1991, **79**, (9)
2. De Micheli G.: 'Computer-Aided Hardware-Software Codesign', *IEEE Micro*, August 1994, **14**, (4)
3. Gautier T., Le Guernic P., Quinton P., Rajopadhye S., Risset T., Smarandache I.: 'Le projet CAIRN : vers la conception d'architectures partir de SIGNAL et ALPHA' *Collection Technique et Scientifique des Tlcommunications*, 1997
4. Kalavade A., Lee E.A.: 'A Hardware-Software Codesign Methodology for DSP Applications' *IEEE Design & Test of Computers*, September 1993, **10**, (3), pp. 16-28
5. Thomas D.E., Adams J.K., Schmit H.: 'A Model and Methodology for Hardware-Software Codesign' *IEEE Design & Test of Computers*, September 1993, **10**, (3), pp. 6-15
6. Salinas M.H., Johnson B.W., Aylor J.H.: 'Implementation-Independent Model of an Instruction Set Architecture in VHDL' *IEEE Design & Test of Computers*, September 1993, **10**, (3), pp. 42-54
7. Gupta R.K., Coelho C.N., De Micheli G.: 'Program Implementation Schemes for Hardware-Software Systems' *Computer*, January 1994, pp. 48-55
8. Le Guernic P., Gautier T.: 'Data-Flow to von Neumann: the SIGNAL Approach', *Advanced Topics in Data-Flow Computing*, (Gaudiot J.-L. and Bic L., 1991), pp. 413-438
9. Le Guernic P., Gautier T., Le Borgne M., Le Maire C.: 'Programming Real-time Applications with SIGNAL', *Proceedings of the IEEE*, September 1991, **79**, (9), pp. 1321-1336
10. Wilde D.: 'The ALPHA Language'. Technical report 827, IRISA/INRIA - Rennes, France, 1994

11. Benveniste A., Beryy G.: 'The Synchronous Approach to Reactive and Real-time Systems', *Proceeding of the IEEE*, September 1991, **79**, (9)
12. Besnard L.: 'Compilation de SIGNAL : horloges, dpendances, environnement', PhD Dissertation, University of Rennes, France, September 1992
13. Amagbegnon T., Le Guernic P., Marchand H., Rutten E.: 'The SIGNAL dataflow methodology applied to a production cell'. Technical report 917, IRISA/INRIA - Rennes, France, 1995
14. Belhadj M.: 'Using VHDL for Link to Synthesis Tools'. Proceedings of the North Atlantic Test Workshop, June 1994, Nimes, France
15. Amagbegnon T., Besnard L., Le Guernic P.: 'Arborescent Canonical Form of Boolean Expressions'. Technical report 2290, IRISA/INRIA - Rennes, France, 1994
16. Wilde D., Sié O.: 'Regular array synthesis using ALPHA'. Technical report 829, IRISA/INRIA - Rennes, France, 1994
17. Smarandache I.: 'Transformations affines d'horloges : application au codesign de systèmes temps-réel en utilisant les langages SIGNAL et ALPHA', PhD Dissertation, University of Rennes, France, October 1998
18. Birkhoff G.: 'Lattice Theory', *AMS colloquium publications* 1973
19. Sanderson J.G.: 'A Relational Theory of Computing', Springer Verlag 1980, **80**, Goss G. and Hartmanis J.
20. Smarandache I., Le Guernic P.: 'Affine Transformations in SIGNAL and Their Applications in the Specification and Validation of Real-Time Systems' Transformation-Based Reactive Systems Development, Proceedings of the 4th International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software, Palma, Espagne, Springer Verlag 1997, **1231**
21. Smarandache I., Le Guernic P.: 'A Canonical Form for Affine Relations in SIGNAL'. Technical report 3097, IRISA/INRIA - Rennes, France, 1997
22. ETSI (European Telecommunication Standards Institute) 'Specification of Component TV codecs 32-45 Mbit/s'. December 1990