INDIANA UNIVERSITY

COMPUTER SCIENCE DEPARTMENT

TECHNICAL REPORT NO. 373

Automatic Synthesis of Sequential Synchronizations

Zheng Zhu and Steven D. Johnson

FEBRUARY 1993

To appear in the procedings of the *1993 IFIP Conference on Hardware Description Languages and their Applications* (*CHDL '93*), Ottawa, Canada, April, 1993.

Also published as University of British Columbia Computer Science Department Technical Report TR 93-3.

Automatic Synthesis of Sequential Synchronizations*

Zheng Zhu[†‡] Steven D. Johnson[§]

Abstract

To compose sequential systems, designers usually have to devise a synchronization mechanism which coordinates constituents of the composition in order to achieve certain goals of computation. In this paper, we present a simple language for specifying sequential behaviors. An advantage of the language is that a specification of synchronization, when composition is required, can be easily obtained from specifications of subsystems. We also briefly describe an algorithm which converts a specification of synchronization to a description of synchronization in our language.

Our approach illustrates that, with a proper sequential descriptions of subsystems, necessary synchronization can be obtained automatically. This frees designers from control design, thus leaving more time and energy to consider architectural improvement and timing efficiency.

1 Introduction

Composing simple subsystems to form a more complicated system has become a common practice in digital hardware designs. In order to achieve certain goals of computation, designers usually have to devise a synchronization mechanism which coordinates constituents of the composition, if the constituents exhibit sequential behaviors. Therefore, providing formalisms and practical tools to facilitate system composition has become not only necessary but also an urgent task to CAD researchers and design practitioners. In our opinion, a design system which facilitates design compositions should, at least, have three capabilities:

- A specification language for sequential behaviors of systems.

*This work was supported, in part, by the National Science Foundation under the grants numbered MIP87-07067, DCR85-21947, and MIP89-21842.

[†]Dept of CS, The University of British Columbia, Vancouver, B.C. Canada V6T 1Z2, zhu@cs.ubc.ca.

[‡]Currently supported, in part, by operating grants OGPO 109688 and OGPO O46196 from the Natural Sciences Research Council of Canada, fellowships from the Province of British Columbia Advanced Systems Institute, by research contract 92-DJ-295 from the Semiconductor Research Corporation, and by the Department of Computer Science, University of British Columbia, Vancouver, B.C. Canada.

[§]Dept of CS, Indiana University, Bloomington, IN, USA 47405, sjohnson@cs.indiana.edu.

- A method of extracting an appropriate synchronization requirements from (1) sequential descriptions of constituents to be composed and (2) the behavioral description of the result of the composition.
- An algorithm which converts an extracted synchronization requirement into a description of synchronization, in the language in which sequential behaviors of constituents are specified.

In this paper, we present a simple language for specifying sequential behaviors of digital circuits. An advantage of the language is that a specification of synchronization, when composition is required, can be easily obtained from specifications of subsystems. We also briefly describe an algorithm which converts a specification of synchronization to a description in our language.

Various languages for describing, verifying and synthesizing sequential behaviors of hardwares have been proposed. Well known formalisms for concurrent systems include CSP [3] by Hoare and CCS [8] by Milner. In [7], Milner extended CCS to model synchrony. Hardware design oriented formalisms, often based on formalisms such CSP and CCS, have been developed in the last decade to address hardware design related issues. Those include CIRCAL [6] by Milne, HOP [2] by Gopalakrishnan and that reported in [9] by Subrahmanyam. In recent years, more and more researchers have focused on hardware timing related issues and have achieved substantial progress. Borriello [1] proposed a method of specifying and automatically synthesizing hardware interfaces. Milne [5] investigated the issues related to hardware timing description and verification. Wolf and Takach [11, 10] used finite state machines to specify hardware and used control transformations to explore design spaces.

Majority of the work in this area focus on verification aspects of sequential system designs. There are only limited works (*e.g.* [1]) which addressed formal and practical aspects in derivation of sequential systems. The work reported in this paper aims at providing a language for sequential behaviors and an automatic generation of synchronizations when sequential compositions are performed. It is our view that derivation approach of designs is a complement to verification approach of designs. Both should be pursued in parallel.

The paper is organized as follows: we first present an example to illustrate the problem to be addressed. Section 2 presents a simple language for specifying sequential behaviors of digital operations. Section 3 demonstrates, through examples, how the language is used to specify sequential operations. Section 4 discusses the methods of specifying sequential synchronizations and derivation of synchronizations from their specifications. Section 5 briefly introduces an algorithm used to derive a synchronization from its specification.

Example 1 In a behavioral specification of hardware, compositions of operations are represented by terms such as $mul(rd(m,a),r)$ where m,a,r denote abstract registers. mul and rd are primitive function symbols, denoting multiplication and memory-read operation respectively. The structure of the term illustrates clearly how the data channels between hardware components $\boxed{mul}$ and $\boxed{rd}$ should be established. However, an actual realization of the term requires knowledge of the sequential characterization of two components. If either components is sequential, then the composition requires a synchronization

mechanism to coordinate the data communication between them. In this particular example, $\boxed{mul}$ and $\boxed{rd}$ should be synchronized in such a way that $\boxed{rd}$ will provide the content of the memory cell to $\boxed{mul}$ as long as it is needed.

2 The Specification Language

Our language for the sequential behavior specification is called the *language of timing expressions*. Timing expressions serve two purposes: describing behaviors of sequential systems and specifying sequential constraints a sequential system has to satisfy. In this section, we briefly present its syntax and the semantics of the language. Instead of presenting every detail of the language, we shall convey concepts through examples. More detailed and formal description of the language can be found in [12].

2.1 Timing Expressions and Timing Functions

Let T a set of terms[1], and $\#$, $\perp$ be two special symbols. Elements of T, $\#$ and $\perp$ are partially ordered by the "information content" to form the following partial order: for every $t \in T$, $\# \sqsubseteq t \sqsubseteq \perp$, and for $t_1, t_2 \in T$, if t_1 and t_2 are distinct, then they are not ordered.

In a digital circuit, terms $t_0, \cdots, t_n, \cdots \in T$ represent known values carried by a signal. $\#$ represents an unknown but fixed value while $\perp$ is the symbol for invalid values. One of the situations where an invalid value occurs on a signal is when two or more signals carrying different values are "hard-wired" together, such as short-circuit of source and ground in a circuit. Situations where a signal carries an invalid value is generally undesirable thus should be avoided.

Definition 2.1. $TE(T)$ is the smallest set which satisfies:

1. $\epsilon \in TE(T)$ where ϵ is an empty string, $\# \in TE(T)$ and $T \subseteq TE(T)$;
2. If $t \in T$, p is a signal and $b \in \{0, 1\}$, then $[p = b \rightarrow t] \in TE(T)$.
3. If $\tau_1, \tau_2 \in TE(T)$ and p is a signal, then (τ_1), $\tau_1 \wedge \tau_2$, $\tau_1 - \tau_2$, $\tau_1\, a$ and $\tau_1\, [p = b \rightarrow a]$ are elements of $TE(T)$.

Every member of $TE(T)$ is called a *timing expression*. Signal p in $[p = b \rightarrow a]$ is called a control signal. We use $C(\tau)$ to denote the set of all control signals appearing in τ.

For example,

(τ_1) $[p = 1 \rightarrow 0]$ $\qquad$ (τ_2) $[p = 1 \rightarrow x \times y]\, x \times y$

(τ_3) $[p = 1 \rightarrow 0] \wedge [q = 1 \rightarrow 0]$ $\qquad$ (τ_4) $([p = 1 \rightarrow 0] \wedge [q = 1 \rightarrow 0])\ 1$

[1] We are not concerned by how this set of terms is constructed. They may be generated from some signature Σ, or obtained in some way.

are valid timing expressions. But

$$([p_0 = 1 \rightarrow 0] \wedge [p_1 = 1 \rightarrow 0])\ ([p_2 = 0 \rightarrow a] \wedge a)$$
$$([p_0 = 1 \rightarrow 0] - [p_1 = 1 \rightarrow 0])\ ([p_2 = 0 \rightarrow a] \wedge a)$$

are not.

A timing expression $[p = b \rightarrow a]$ is called a (p, b)-*iteration*. Sometimes, when b is known, it is also called a p-*iteration*. As explained later, $[p = b \rightarrow a]$ behaves similar to

$$\textbf{while } p = b \textbf{ do } a$$

in conventional programming language.

Before we discuss the meaning of timing expressions, we need to introduce a binary operator $\sqcup$ on the set of all terms. It defines the consequence of joining two signals together: Let t_1 and t_2 be either terms, or #,

$$t_1 \sqcup t_2 = \begin{cases} t_2 & t_1 = \# \\ t_1 & t_2 = \# \\ t_1 & t_1 \text{ and } t_2 \text{ are identical} \\ \text{invalid} & \text{Otherwise} \end{cases}$$

Intuitively, a timing expression denotes a sequence of values. Timing expression 0 denotes stream $< 0 >$. $[p = 0 \rightarrow 1]$ denotes a stream of zero or finite number of 1s. The length of the stream depends on the stream assigned to p. For example, if $< 0\ 0\ 0\ 1\ 0\ 0 \cdots >$ is the stream for signal p, then $[p = 0 \rightarrow 1]$ is $< 1\ 1\ 1 >$. $\tau_1\ \tau_2$ represents concatenation of streams denoted by τ_1 and τ_2 respectively. $\tau_1 \wedge \tau_2$ denotes the stream resulted from pair-wisely applying $\sqcup$ operator to the streams denoted by τ_1 and τ_2. For instance, $\tau_1 = 0\ \#\ 1$, $\tau_2 = \#\ [p = 1 \rightarrow 1]$ and $p =< 1\ 1\ 1\ 0 >$, then $\tau_1 \wedge \tau_2$ denotes the stream $< 0\ 1\ 1\ 1 >$. If α, β are streams denoted by τ_1 and τ_2, and $len(\alpha)$ is the length of α, then $\tau_1 - \tau_2$ denotes the stream $\alpha\ \beta^{+len(\alpha)}$ where β^{+n} denotes the suffix of β whose first element is the $n+1$th element of β. If n is greater or equal to the length of β, then $\beta^{+n} = \epsilon$. For instance, let $\alpha =< 0\ \#\ 1\ 3 >$ and $\beta =< 4\ 4\ 4\ 4\ 4\ 4 >$, the resulting stream is $< 0\ \#\ 1\ 3\ 4\ 4 >$. If $\beta =< 4\ 4 >$, then the steam denoted by $\tau_1 - \tau_2$ is $< 0\ \#\ 1\ 3 >$.

Functions which map signals to timing expressions are called *timing functions* in this paper. We use the following notation to display a timing function ρ whose domain is $\{p_1, \cdots, p_n\}$, and $\rho(p_i) = \tau_i$ for $i = 1, \cdots, n$:

$$\begin{aligned} \rho: \quad & p_1 \Leftarrow \tau_1 \\ & p_2 \Leftarrow \tau_2 \\ & \vdots \\ & p_n \Leftarrow \tau_n \end{aligned}$$

Later in this paper, timing functions are used to specify behaviors of digital circuits. When doing so, signals in the domain of a timing function denote either input ports or output port of the specified digital component. All the control signals, except those in the domain of the timing function, constitute the environment in which the specified component operates. This leads to the following definition:

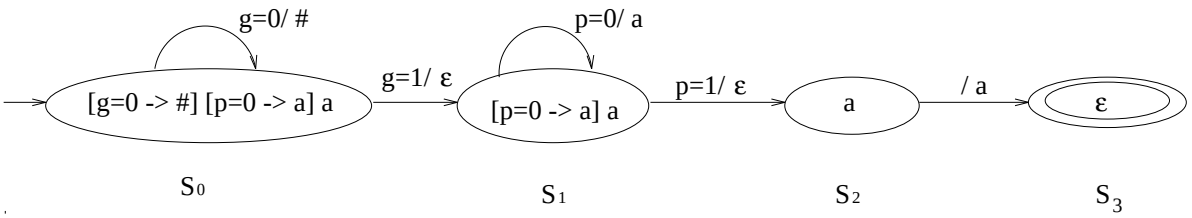

Figure 1: Automaton for $[g = 0 \rightarrow \#]\ [p = 0 \rightarrow a]\ a$

Definition 2.2. Let τ be a timing expression and ρ be a timing function. The environments of τ, denoted by $Env(\tau)$, is the set of all functions $C(\tau) \rightarrow \{0,1\}^\infty$ where $\{0,1\}^\infty$ is the set of all infinite sequences of 0s and 1s. The environments of ρ, denoted by $Env(\rho)$, is the set of all functions $(C(\rho) - dom(\rho)) \rightarrow \{0,1\}^\infty$ where

$$C(\rho) = \bigcup_{p \in dom(\rho)} C(\rho(p))$$

In other words, an environment provides actual sequences for each of the control signals.

Example 2 Let ρ is timing function on $\{p_1, p_2, p_3, p_4\}$:

$$\begin{aligned}
\rho: \quad & p_1 \Leftarrow [g = 0 \rightarrow \#]\ [p_3 = 0 \rightarrow a]\ a \\
& p_2 \Leftarrow [g = 0 \rightarrow \#]\ [p_3 = 0 \rightarrow b]\ b \\
& p_3 \Leftarrow [g = 0 \rightarrow \#]\ [p = 0 \rightarrow 0]\ 0\ 1 \\
& p_4 \Leftarrow [g = 0 \rightarrow \#]\ [p_3 = 0 \rightarrow \#]\ [rls = 0 \rightarrow a \times b]\ a \times b
\end{aligned}$$

and $C(\rho) = \{g, p_3, p, rls\}$. An environment of ρ is a function $\{g, p, rls\} \rightarrow \{0,1\}^\infty$. Although p_3 is used as a control signal, it is internally generated rather than imported. Therefore, p_3 is not a constituent of the environment.

2.2 Finite Interpretation of Timing Functions

The *finite interpretation* of timing function ρ simulates "one pass computation" of ρ. Firstly, we can simulate a timing expression by a finite state automata (FSA). For example, a timing expression

$$[g = 0 \rightarrow \#]\ [p = 0 \rightarrow a]\ a$$

can be regarded as a four-state Mealy machine. The initial state is the timing expression itself. Every suffix of the timing expression, *e.g.* $[p = 0 \rightarrow a]\ a$ and a, is a state. Finally, the empty string ϵ is the terminal state (Figure 1).

Using the diagram and given sequences of g and p, we can obtain the following correspondence between input sequences and state-transition, and output values:

$$\begin{array}{rccccccc}
g: & < & 0 & 0 & 1 & 0 & \cdots & > \\
p: & < & 0 & 0 & 1 & 0 & \cdots & > \\
State: & < & S_0 & S_0 & S_2 & S_3 & > & \\
Output: & < & \# & \# & a & > & &
\end{array}$$

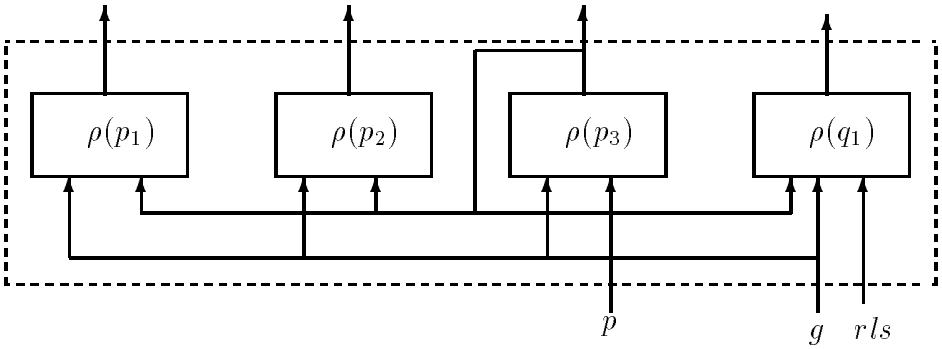

Figure 2: Timing Function ρ as a System of Parallel Automata

The sequence of output is what we expect intuitively from the timing expression when signals q and p are assigned the given sequences respectively. The sequence of states tells us the "residue" of the timing expression in the given environment. This analogy between timing expressions provides the basis of meanings of timing functions which are regarded as systems of automata running in parallel in a *lock-step* fashion. For example, the timing function in Example 2 can be viewed as four parallel automata shown in Figure 2. Arrows into each box represent control inputs to the automaton. There are only three control signals going into the system from outside of the dashed box.

Formally, let ρ be a timing function and e be an environment of ρ. $\mathcal{E}_0(\rho, e)$ is a function on $dom(e) \cup dom(\rho)$:

Definition 2.3. For every $p \in dom(\rho) \cup dom(e)$,

1. If $p \in dom(e)$, then $\mathcal{E}_0(\rho, e)(p) = e(p)$. If $\rho(p) = \epsilon$, then $\mathcal{E}_0(\rho, e)(p) = \epsilon$. And if $\rho(p) = a$, then $\mathcal{E}_0(\rho, e)(p) = a$;

2. If $\rho(p) = [q = b \rightarrow a]$ then

$$\mathcal{E}_0(\rho, e)(p) = \begin{cases} a & \mathcal{E}_0(\rho, e)(q) = b \\ \epsilon & \mathcal{E}_0(\rho, e)(q) = \neg b \end{cases}$$

3. If $\rho(p) = \tau_1 \wedge \tau_2$, then $\mathcal{E}_0(\rho, e)(p) = \mathcal{E}_0(\rho[p/\tau_1], e)(p) \sqcup \mathcal{E}_0(\rho[p/\tau_2], e)(p)$

4. If $\rho(p) = \tau_1 - \tau_2$ or $\rho(p) = \tau_1\, \tau_2$ then

$$\mathcal{E}_0(\rho, e)(p) = \begin{cases} \mathcal{E}_0(\rho[p/\tau_1], e)(p) & \mathcal{E}_0(\rho[p/\tau_1], e)(p) \neq \epsilon \\ \mathcal{E}_0(\rho[p/\tau_2], e)(p) & \text{Otherwise} \end{cases}$$

2 of the definition needs some clarification. Apparently, when $p = q$, the definition becomes:

$$\mathcal{E}_0(\rho, e)(p) = \begin{cases} a & \mathcal{E}_0(\rho, e)(p) = b \\ \epsilon & \mathcal{E}_0(\rho, e)(p) = \neg b \end{cases}$$

which causes self-reference of signal p thus undefined. Self-reference is analogous to the race condition in circuit designs, or *interlock* in theories of concurrent systems. An algorithm for detecting self-references is described in [12]. A timing function that does not

have self-reference during its evaluation in every given environment is called *self-reference free*. Throughout this paper, we assume that every timing function is self-reference free. The next definition simulates state transitions of an automaton.

Definition 2.4. Let τ be a timing expression and $e \in Env(\tau)$. The continuation of τ under e, denoted by $Cont(\tau, e)$ is defined by:

1.
$$Cont(\epsilon, e) = \epsilon, \quad \text{and} \quad Cont(a, e) = \epsilon$$

2.
$$Cont([p = b \to a], e) = \begin{cases} \epsilon & \text{if } e(p) = \neg b \\ [p = b \to a] & \text{if } e(p) = b \end{cases}$$

3.
$$Cont(\tau_1\,\tau_2, e) = \begin{cases} Cont(\tau_1, e)\ \tau_2 & \text{if } Cont(\tau_1, e) \neq \epsilon \\ Cont(\tau_2, e) & \text{if } Cont(\tau_1, e) = \epsilon \end{cases}$$

4. Let $\bullet \in \{\wedge, -\}$, then
$$Cont(\tau_1 \bullet \tau_2, e) = \begin{cases} Cont(\tau_1, e) & \text{if } Cont(\tau_2, e) = \epsilon \\ Cont(\tau_2, e) & \text{if } Cont(\tau_1, e) = \epsilon \\ Cont(\tau_1, e) \bullet Cont(\tau_2, e) & \text{Otherwise} \end{cases}$$

To extend the above definition to timing functions, let ρ be a timing function and $e \in Env(\rho)$. $Cont(\rho, e)$ is defined as a timing function such that for every $p \in dom(\rho)$, $Cont(\rho, e)(p) = Cont(\rho(p), \mathcal{E}_0(\rho, e))$.

Finally, we can define the "finite" response of a timing function ρ to an environment e, denoted by $\ll\rho\gg(e)$:

Definition 2.5. Let ρ be a timing function. $\ll\rho\gg : Env(\rho) \to (dom(\rho) \to \mathcal{T}^*(\rho))$ is defined by

$$\ll\rho\gg(e) = \begin{cases} \rho & \forall p \in dom(\rho).\ \rho(p) = \epsilon \\ \mathcal{E}_0(\rho, e(1)) \downarrow dom(\rho)\ \ll Cont(\rho, e(1))\gg(e^+) & \text{Otherwise} \end{cases}$$

Example 3. Let e be an environment of the timing function in the Example 2:

$$\begin{array}{rcc ccccc c} & & \overbrace{}^{e(1)} & \multicolumn{6}{c}{\overbrace{\qquad\qquad\qquad\qquad}^{e^+}} \\ e(g) & \Leftarrow & 0 & 1 & 1 & 0 & 0 & 0 & \cdots \\ e(p) & \Leftarrow & 0 & 0 & 1 & 1 & 0 & 1 & \cdots \\ e(rls) & \Leftarrow & 1 & 1 & 0 & 0 & 0 & 1 & \cdots \end{array}$$

We now compute $\ll\rho\gg(e)$ as follows:

$$\begin{array}{rcc ccccc} \ll\rho\gg(e)(p_1) & \Leftarrow & \# & a & a & a & & \\ \ll\rho\gg(e)(p_2) & \Leftarrow & \# & b & b & b & & \\ \ll\rho\gg(e)(p_3) & \Leftarrow & \# & 0 & 0 & 1 & & \\ \ll\rho\gg(e)(p_4) & \Leftarrow & \underbrace{\#}_{\mathcal{E}_0(\rho, e(1)) \downarrow dom(\rho)} & \# & \# & a \times b & a \times b & a \times b \end{array}$$

(the last five columns together: $\ll Cont(\rho, e(1))\gg(e^+)$)

Finally, $Cont(\rho, e(1)) = \rho$,

$$\begin{array}{ll} Cont(Cont(\rho, e(1)), e(2)): & p_1 \Leftarrow [p_3 = 0 \rightarrow a]\, a \\ & p_2 \Leftarrow [p_3 = 0 \rightarrow b]\, b \\ & p_3 \Leftarrow [p = 0 \rightarrow 0]\, 0\, 1 \\ & p_4 \Leftarrow [p_3 = 0 \rightarrow \#][rls = 0 \rightarrow a \times b]\, a \times b \end{array}$$

Consequently,

$$\begin{array}{ll} Cont(Cont(Cont(\rho, e(1)), e(2)), e(e)): & p_1 \Leftarrow [p_3 = 0 \rightarrow a]\, a \\ & p_2 \Leftarrow [p_3 = 0 \rightarrow b]\, b \\ & p_3 \Leftarrow 1 \\ & p_4 \Leftarrow [p_3 = 0 \rightarrow \#][rls = 0 \rightarrow a \times b]\, a \times b \end{array}$$

$\ll\gg$ can be extended to evaluating a timing expression in a given environment as follows: Let τ be a timing expression and e be an environment of τ, and ρ be a timing function such that $dom(\rho) = \{p\} \not\subseteq C(\tau)$ and $\rho(p) = \tau$. Define $\ll\tau\gg(e) = \ll\rho\gg(e)(p)$.

The (infinite) semantics of timing functions is an extension of the finite interpretation. We only give an informal description here. A detailed account can be found in [12]. The (infinite) semantics of a timing function is a function which, given an environment and a set of input sequences, returns sequences of data outputs. Under a given control sequence, the requirements for input data, including when data is available on a particular port, how long it lasts, *etc.*, are represented by variables in the timing function's finite interpretation. For example, assume that the timing function in Example 2 is evaluated to a set of sequences:

$$\begin{array}{lllllll} p_1 & \Leftarrow & \# & a & a & a & \\ p_2 & \Leftarrow & \# & b & b & b & \\ p_3 & \Leftarrow & \# & 0 & 0 & 1 & \\ p_4 & \Leftarrow & \# & \# & \# & a \times b & a \times b \quad a \times b \end{array}$$

The set of the timing function imposes a restriction on data inputs: signals p_1 and p_2 need to be stable for three clock cycles before the output produces $a \times b$ for 3 consecutive cycles. If the sequences of p_1 and p_2 are:

$$\begin{array}{lllllll} p_1 & \Leftarrow & 4 & 3 & 3 & 3 & \cdots \\ p_2 & \Leftarrow & 9 & 7 & 7 & 7 & \cdots \end{array}$$

Then, the operation should respond to the input sequence and the given control environment by generating the following sequence on signal p_4:

$$p_4 \quad \Leftarrow \quad \# \quad \# \quad \# \quad 21 \quad 21 \quad 21$$

However, if the input sequence cannot provide a stable input, say

$$p_1 \quad \Leftarrow \quad 4 \quad 3 \quad 3 \quad 9 \quad \cdots$$

Then the sequence for p_4 is unpredictable because the sequence for p_1 does not meet the requirement.

Figure 3: Digital Circuits

We use $[\![\rho]\!](e, \alpha)$ to denote the meaning (infinite semantics) of the timing function ρ in an environment e and input (data) sequence α. Sometimes, ρ may not require any data input[2]. In these situations, we simply use $[\![\rho]\!](e)$ to denote the infinite interpretation of ρ in the environment e.

3 Specification of Digital Components

Specifications of digital components are concerned with different facets of their operations. Among those facets are the algebraic properties of the operations, temporal characterizations, constraints to the environment in which the components are used, electrical properties such as fan-in, fan-out, power, input/output waveforms, and physical requirements such as layout, and so on. Sequential specifications introduced in this section address two facets of operations: input/output (behavioral) relations among signals and temporal properties of a specified operation.

3.1 Examples of Sequential Specification

We now present specifications of a combinational adder and a sequential multiplier (Figure 3). As a convention, we draw data signals vertically and control signals by vertical and horizontal lines (*e.g.* *start* and *done* in Figure 3.) Arrows on signals indicate signal flow direction.

[2] *e.g.* ρ is a timing function which contains no variables but $0, 1$ and $\#$. Timing functions of this type will be seen in later sections of this paper.

3.1.1 Example: A Combinational Adder

This combinational adder has two input signals i_1 and i_2, and one output signal o (Figure 3). Its timing function ρ_a is:

$$\begin{array}{rl} \rho_a: & i_1 \Leftarrow x_1 \\ & i_2 \Leftarrow x_2 \\ & o \Leftarrow x_1 + x_2 \end{array}$$

Since ρ_a does not have any control signal, its evaluation depends on data inputs only. Assume α is:

$$\begin{array}{rlllllll} \alpha: & i_1 & \Leftarrow & 1 & 3 & 5 & \cdots \\ & i_2 & \Leftarrow & 8 & 9 & 3 & \cdots \end{array}$$

ρ_a is evaluated to a sequence whose first 3 elements (a prefix of $[\![\rho_a]\!](\alpha)$) are

$$(1{+}8)\ (3{+}9)\ (5{+}3)$$

3.1.2 Example: A Sequential Multiplier

Assume the multiplier in Figure 3 behaves as follows

> "If 1, x_1, and x_2 are values on signals $start$, i_1 and i_2, respectively, at time t_1. x_1, x_2 stay on i_1, i_2 until time t_2 which is the first time when or after t_1 that signal $done$ holds value 1, then the value on signal o at time t_2 is $x_1 \times x_2$.

The timing function of this multiplier is:

$$\begin{array}{rl} \rho_m: & i_1 \Leftarrow [start = 0 \rightarrow \#\,]\ [done = 0 \rightarrow x_1] \\ & i_2 \Leftarrow [start = 0 \rightarrow \#\,]\ [done = 0 \rightarrow x_2] \\ & o \Leftarrow [start = 0 \rightarrow \#\,]\ [done = 0 \rightarrow \#\,]\ (x_1 \times x_2) \end{array} \quad (1)$$

Let us evaluate ρ_m in an environment e and α where

$$\begin{array}{rlllllll} e: & start \Leftarrow 0 & 0 & 1 & 0 & 1 & \cdots \\ & done \Leftarrow 0 & 1 & 0 & 0 & 1 & \cdots \end{array} \qquad \begin{array}{rlllllll} \alpha: & i_1 \Leftarrow 4 & 3 & 3 & 3 & 3 & \cdots \\ & i_2 \Leftarrow 2 & 9 & 5 & 5 & 5 & \cdots \end{array}$$

We can compute a prefix of $[\![\rho_m]\!](e, \alpha)$ in the following steps:

1. First, the finite interpretation of ρ_m:

$$\begin{array}{rlllll} \ll\rho_m\gg(e): & i_1 \Leftarrow \# & \# & x_1 & x_1 & x_1 \\ & i_2 \Leftarrow \# & \# & x_2 & x_2 & x_2 \\ & o \Leftarrow \# & \# & \# & \# & (x_1 \times x_2) \end{array}$$

2. We then match the first 3 elements of $\alpha(i_1)$ against those of $\ll\rho_a\gg(e, \alpha)(i_1)$ to find an assignment to the variable $x_1 = 3$. Similarly, matching the first 3 elements of $\alpha(i_2)$ against $\ll\rho_a\gg(e, \alpha)(i_2)$ yields an assignment to variable $x_2 = 5$.

3. Therefore, the first four elements of $[\![\rho_m]\!](e, \alpha)$ is $\# \ \# \ \# \ (3 \times 5)$. The rest of the sequence is $[\![\rho_m]\!](e^{+3}, \alpha^{+3})$ where

$$e^{+3}: \begin{array}{lllll} start \Leftarrow 0 & 1 & \cdots \\ done \Leftarrow 0 & 1 & \cdots \end{array} \qquad \alpha^{+3}: \begin{array}{lllll} i_1 \Leftarrow 9 & 1 & \cdots \\ i_2 \Leftarrow 7 & 8 & \cdots \end{array}$$

4 Synchronization Specification and Derivation

4.1 Example: Sequential Composition

We now study how to compose the multiplier with a memory-read operator to perform the operation $mul(rd(m, a), r)$. We were given the timing function of a sequential multiplier ρ_m earlier. The memory-read's timing function is ρ_r:

$$\begin{array}{rrcl} \rho_r: & mem & \Leftarrow & [s_r = 0 \rightarrow \#\,]\ m \\ & addr & \Leftarrow & [s_r = 0 \rightarrow \#\,]\ a \\ & r_o & \Leftarrow & [s_r = 0 \rightarrow \#\,]\ \#\ [rls = 0 \rightarrow rd(m, a)]\ rd(m, a) \end{array}$$

In ρ_r, mem and $addr$ are input signals and r_o is an output signal. s_r, rls are input control signals. ρ_r can be described narratively as:

> The environment first raises *start-read* signal s_r and asserts the address of a memory cell a to be accessed on signal $addr$. After one clock cycle, the content of the memory cell becomes available on port r_o and stays stable until one clock cycle after the signal rls became 0.

It is not hard to see that connecting appropriate signals of two components alone does not do the job. For example, assume that both operations start from their initial states. Proper action should be taken to guarantee that when s_m signal rises, the correct value from memory should be available on r_o so that it can appear on one of multiplier's input signals. In ρ_m, the timing expression $[s_m = 0 \rightarrow \#]$'s termination signifies the rise of s_m signal. In ρ_r, availability of memory cell's content is signified by the termination of the timing expression $[s_r = 0 \rightarrow \#\,]\ \#$. If both timing expressions are regarded as measurements of durations, we can say that a necessary condition to synchronize is that $[s_m = 0 \rightarrow \#]$ consumes at least as much time as $[s_r = 0 \rightarrow \#\,]\ \#$ does. We use the following notation to express this condition:

$$[s_r = 0 \rightarrow \#]\ \# \quad \preceq \quad [s_m = 0 \rightarrow \#\,] \tag{2}$$

On the other hand, we also need a condition that "$[s_m = 0 \rightarrow \#\,]$ does not consume more time than $[s_r = 0 \rightarrow \#]\ \#\ [rls = 0 \rightarrow rd(m, a)]\ rd(m, a)$ does", which is expressed by

$$[s_m = 0 \rightarrow \#\,] \quad \preceq \quad [s_r = 0 \rightarrow \#]\ \#\ [rls = 0 \rightarrow rd(m, a)]\ rd(m, a) \tag{3}$$

Both (2) and (3) are called *inequalities (of timing expressions)*. These two inequalities impose constraints as shown in Figure 4, which consists of three groups of timing diagrams.

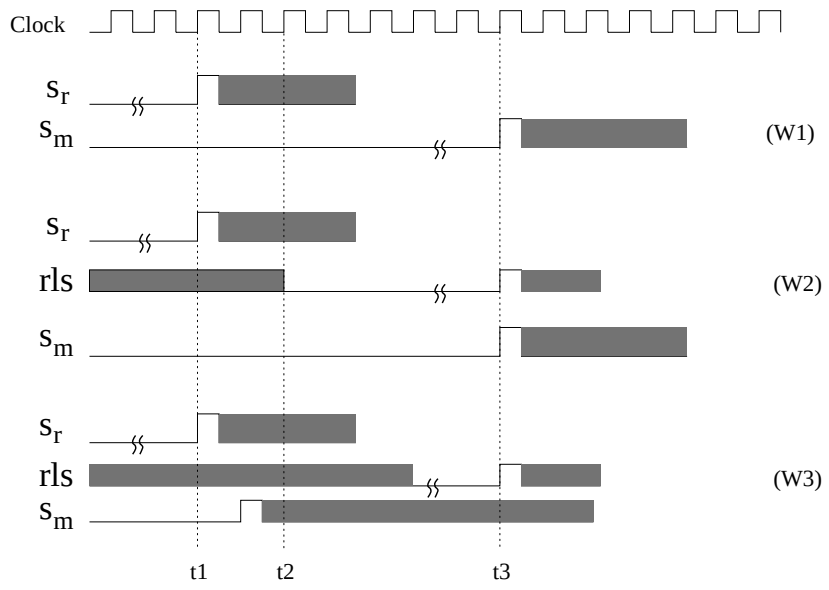

Figure 4: Timing Diagrams: Inequality of Timing Expressions

$W1$ shows the relationship among signals s_r, and s_m. The shaded areas mean "don't care the values during the period". By the inequality (2), the rise of s_m (at time t_3) can be moved back to as early as t_2 and still satisfy the inequality. However, if $t_3 < t_2$, then $W1$ does not satisfy the inequality any more. $W2$ shows waveforms of signals which satisfy inequality (3): the rise of s_m can be moved back to anywhere but not later than the rise of rls at t_3. The issue we are facing is to find an arrangement of signal waveforms so that both inequalities can be satisfied. If we compare $W1$ and $W2$, we find that waveforms in $W2$ satisfy both inequalities (2) and (3). On the other hand, the waveforms in $W3$ also satisfy inequality (3), but it does not satisfy (2), which requires the rise of s_m after t_2.

5 Solving Systems of Inequalities

Solving a system of inequalities, such as (2) and (3), is a key for deriving a synchronization required by the composition. In this section, we briefly outline an algorithm and theorems concerning the algorithm. It should be emphasized that the algorithm is more a "constructive proof" of the existence of such an algorithm than a practical solution to the problem.

There are three steps in the algorithm:

Step 1 Converting a system of inequalities to a set of timing relations;

Step 2 Eliminating self-references;

Step 3 Converting timing relations to timing functions.

5.1 Step 1

The first step is to convert a system of inequalities (I) to a set of timing relations ($\Re(I)$). Every relation $r \in \Re(I)$ is a finite set of pairs (p, τ) where p is an input control signal and τ is a timing expression. For example, the inequalities

$$\begin{array}{rcl} [s_r = 0 \rightarrow \#]\,\# & \preceq & [s_m = 0 \rightarrow \#] \\ [s_m = 0 \rightarrow \#] & \preceq & [s_r = 0 \rightarrow \#]\,\#\,[rls = 0 \rightarrow rd(m,a)]\,rd(m,a) \end{array} \tag{4}$$

can be translated to two timing relations:

$$\left\{ \begin{array}{l} (s_m,\ [s_r = 0 \rightarrow 0]\,0), \\ (rls,\ [s_r = 0 \rightarrow \#]\,\# - [s_m = 0 \rightarrow 0]) \end{array} \right\} \tag{5}$$

and

$$\left\{ \begin{array}{l} (s_m,\ [s_r = 0 \rightarrow 0]\,0), \\ (s_r,\ [s_m = 0 \rightarrow 0]) \end{array} \right\} \tag{6}$$

In (5), the pair

$$(s_m,\ [s_r = 0 \rightarrow 0]\,0)$$

means that, in order to satisfy the first inequality, the input signal s_m should carry value 0 until one clock cycle after the signal s_r changes from 0 to 1. The pair

$$(rls,\ [s_r = 0 \rightarrow \#]\,\# - [s_m = 0 \rightarrow 0])$$

means that, in order to satisfy the second inequality, values on signal rls should meet the following conditions: Assume that the signal s_m receives n 0s followed by a 1, and the signal s_r receives m 0s followed by a 1 ($m, n \geq 0$).

1. If $m \geq n-1$, that is, $[s_r = 0 \rightarrow \#]\,\#$ consumes more time than $[s_m = 0 \rightarrow \#]$ does, then the values on signal rls is a sequence of #s, *i.e. don't-care*s.
2. If $m < n-1$, that is, $[s_m = 0 \rightarrow \#]$ consumes more time than $[s_r = 0 \rightarrow \#]\,\#$ does, then rls will receive a sequence whose first $m+1$ values are #s and then followed by $n-(m+1)$ 0s.

5.2 Step 2

The second step of the algorithm is to eliminate self-references in every timing relation in $\Re(I)$. $r \in \Re(I)$ contains self-references if there exist

$$(p_0, \tau_0), \cdots, (p_{m-1}, \tau_{m-1}) \in r$$

such that

$$p_i \in C(\tau_{i+1})\ \ 0 \leq i \leq m-2 \qquad \text{and} \qquad p_{m-1} \in C(\tau_0)$$

This means that dependencies among signals are cyclic, which is undesirable. In general, there are two possible outcomes to a relation which contains self-references. The

first outcome is that the relation can be "repaired" by eliminating the self-reference. The following timing relation is an example of such a "repairable" self-reference relation.

$$\left\{ \begin{array}{l} (p_2, [p_1 = 0 \rightarrow 0]\,[p_3 = 0 \rightarrow 0]), \\ (p_1, [p_2 = 0 \rightarrow 0]\,[p_4 = 0 \rightarrow 0]) \end{array} \right\} \tag{7}$$

If the inequalities

$$\begin{array}{lcl} [p_1 = 0 \rightarrow 1]\,[p_3 = 0 \rightarrow 0] & \preceq & [p_1 = 0 \rightarrow 1] \\ [p_2 = 0 \rightarrow 0]\,[p_4 = 0 \rightarrow 0] & \preceq & [p_2 = 0 \rightarrow 0] \end{array}$$

are satisfied, then we can prove that (7) is equivalent to the following timing relation:

$$\left\{ \begin{array}{l} (p_1, [p_2 = 0 \rightarrow 0]) \\ (p_2, [p_1 = 0 \rightarrow 0]) \\ (p_3, [p_1 = 0 \rightarrow \#]\,1) \\ (p_4, [p_2 = 0 \rightarrow \#]\,1) \end{array} \right\} \tag{8}$$

Furthermore, we can prove that the relation (8) is equivalent to the following [12]:

$$\left\{ \begin{array}{l} (p_1, [p_2 = 0 \rightarrow 0]) \\ (p_3, [p_1 = 0 \rightarrow \#]\,1) \\ (p_4, [p_2 = 0 \rightarrow \#]\,1) \end{array} \right\}$$

It is obvious that this timing relation does not contain any self-reference.

Another possibility is that the self-reference is a result of contradiction in original inequalities or an over-constraint during solving inequalities. An example of contradicting inequalities is:

$$[p = 0 \rightarrow 0]\,0 \preceq [p = 0 \rightarrow 0]$$

It is translated to a singleton relation:

$$\{\, (p, [p = 0 \rightarrow 0]\,0)\,\} \tag{9}$$

which contains a self-reference on signal p. This relation is not repairable because the inequality is contradictory: in every environment of $[p = 0 \rightarrow 0]$, if the evaluation of $[p = 0 \rightarrow 0]$ results in a sequence of length n, then the evaluation of $[p = 0 \rightarrow 0]\,0$ results in a sequence of length $n + 1$. Since $n \leq n + 1$, (9) can not be true in any environment.

Timing relation (6) is an example of "over-constrained" partial solution. Although

$$(s_r, [s_m = 0 \rightarrow 0])$$

is a solution to the second inequality in (4), it contradicts to the solution to the first inequality in (4). Therefore, (6) will not lead to any solution to inequalities in (4).

Our algorithm can detect such contradictions thus eliminate contradictory timing relations. Otherwise, the algorithm translates a timing relation to an equivalent timing relation which does not contain any self-reference.

5.3 Step 3

The last step of the algorithm is to translate relations to functions. This step is necessary because, in general, Step 1 of the algorithm results in relations rather than functions. This translation is done by merging all timing expressions related to the same signal. For example, a timing relation, among others, contains the following two pairs:

$$\begin{array}{l}(p,\ [s_m = 0 \rightarrow 0]\,[d = 0 \rightarrow 0]) \\ (p,\ [s_r = 0 \rightarrow 0]\,0\,[r = 0 \rightarrow 0]\,1)\end{array} \tag{10}$$

Since both pairs denote values carried on signal p, it is necessary to find a synchronization among signals involved in two expressions so that two expression represent a consistent value sequence for signal p. Since the expression

$$[s_r = 0 \rightarrow 0]\,0\,[r = 0 \rightarrow 0]\,1$$

has a trailing value 1,

$$[s_m = 0 \rightarrow 0]\,[d = 0 \rightarrow 0] \preceq [s_r = 0 \rightarrow 0]\,0\,[r = 0 \rightarrow 0]$$

is a sufficient condition under which two pairs can be "merged" into a single one:

$$(p,\ [s_r = 0 \rightarrow 0]\,0\,[r = 0 \rightarrow 0]\,1)$$

It can be proven that (10) is equivalent to the function

$$\left\{ \begin{array}{l}(p,\ [s_r = 0 \rightarrow 0]\,0\,[r = 0 \rightarrow 0]\,1), \\ (r,\ [s_r = 0 \rightarrow \#]\,\# - [s_m = 0 \rightarrow 0]\,[d = 0 \rightarrow 0])\end{array} \right\}$$

In [12], we proved the following fact of the algorithm: given a system of inequalities I, the algorithm terminates on I and returns three possible values:

1. $\{\emptyset\}$. $\emptyset$ is an empty function which is regarded as an "unconstrained" timing function. This means that every inequality in I holds on itself. An example of such an I is

$$\left\{ \begin{array}{l}\# \preceq \#\ 1 \\ [p = 0 \rightarrow 0] \preceq [p = 0 \rightarrow 0]\,1\end{array} \right\}$$

   which has a solution $\emptyset$.

2. $\{\perp\}$. This means that I does not have any solution.

3. $\{f_1, \cdots, f_n\}$ where every f_i is a self-reference free timing function and every f_i is a solution of I.

6 Conclusion

This paper presented a language for sequential behavior specification. This language is also used to specify sequential interfaces (synchronizations) for sequential compositions and sequential control integrations. We also briefly described the algorithm which derives a synchronization description in terms of timing functions from a synchronization specification in terms of system of inequalities. The algorithm should be regarded as a "constructive proof" of the existence of an algorithm rather than a practical solution to the problem, due to its computational complexity. Currently, we are studying the possibility of simplifying the algorithm and making it computationally tractable.

References

[1] Borriello, G. Specification and synthesis of interface logic. In *High-Level VLSI Synthesis*, R. Camposano and W. Wolf, Eds. Kluwer Acdemic Publishers, 1991, ch. 7, pp. 153–176.

[2] Gopalakrishnan, G. C., Fujimoto, F. M., Akella, V., and Mani, N. S. HOP: A process model for synchronous hardware; semantics and experiments in process composition. *Integration, the VLSI journal 8* (1989), 209–247.

[3] Hoare, C. A. R. *Communicating Sequential Processes.* Prentice Hall, 1985.

[4] Manna, Z. *Mathematical Theory of Computation.* McGraw-Hill, New York, 1974.

[5] Milne, G. Timing constraints, formalizing their description and verification. In *Proceedings of the IFIP WG 10.2 9th International Symposium on Computer Hardware Description Languages and Their Applications* (1989), J. Darringer and F. Ramming, Eds., North Holland.

[6] Milne, G. J. CIRCAL and the representation of communication concurrency and time. *ACM Transactions on Programming Languages and Systems 7*, 2 (1985).

[7] Milner, R. Calculi for synchrony and asynchrony. *Theoretical Computer Science 25* (1983), 267–310.

[8] Milner, R. *Communication and Concurrency.* Prentice Hall, 1989.

[9] Subrahmanyam, P. A. What's in a timing discipline?: Considerations in the specification and synthesis of systems with interacting asynchronous and synchronous components. In *Lecture Notes in Computer Science* (1989), M. Leeser and G. Brown, Eds., Cornell University, Springer-Verlag.

[10] Takach, A., and Wolf, W. Behavior FSMs for high-level synthesis and verification. Tech. Rep. CE-W91-13, Dept of EE, Priceton University, 1991.

[11] Wolf, W., and Takach, A. Architectural optimization methods for control-dominated machines. In *High-Level VLSI Synthesis*, R. Camposano and W. Wolf, Eds. Kluwer Acdemic Publishers, 1991, pp. 231–254.

[12] Zhu, Z. *Structured Hardware Design Transformations*. PhD thesis, Computer Science Department, Indiana University, USA, 1992.