設 計 自 動 化 89－7
(1998. 9. 21)

Self-Timed Implementation of Boolean Functions

Märt Saarepera　　米田 友洋

東京工業大学 計算工学専攻
〒152-8552 東京都目黒区大岡山 2-12-1

あらまし　非同期式回路はクロックを持たないため，演算結果の終了を特別な方法を用いて示す必要がある．あるクラスの非同期式回路は，そのような演算完了をその演算回路自身が出力する．本稿では，このようなクラスの非同期式組み合せ回路の実現方法について検討する．従来，このような回路を実現するためには，内部の安定性を検査する付加回路が必要であった．このような付加回路は，回路を複雑にし，その結果，動作速度の低下を招く．本稿では，演算完了を正しく判定できるようなプロトコルを新しく提案し，そのプロトコルを満足する回路が付加回路なしに実現できることを示す．

キーワード　　非同期式組み合わせ回路，delay-insensitivity，プロトコル

Self-Timed Implementation of Boolean Functions

Märt Saarepera　　Tomohiro Yoneda

Department of Computer Science
Tokyo Institute of Technology
2-12-1 Ookayama Meguro-ku Tokyo 152-8552, Japan

Abstract　Since asynchronous circuits have no clock, the completion of the computation must be notified by using some special ways. There exists a class of asynchronous circuits which generate such completion signals by themselves. These circuits function under a signaling scheme called protocol. In this paper, we discuss how to implement such a class of asynchronous circuits. In the previous works, the implementation of such circuits needs asynchronous automata which make the circuits complicated and slow. In this paper, we propose a new protocol which guarantees that environment can correctly know the completion of the computation, and show that circuits which satisfy the protocol can be implemented without involving asynchronous automata.

key words　　Asynchronous combinational circuits, delay-insensitivity, protocol

1 Introduction

Implementation of Boolean functions is the basis for computation. If computation is synchronized by clock then the Boolean circuits implement the Boolean functions. Implementation of Boolean functions in asynchronous circuits is not so straightforward. Since asynchronous circuits have no clock, the completion of the computation must be notified by using some special ways. One way of doing this is to use coded inputs and outputs and to organize the work of the circuit so that the computation starts in a neutral state, then the coded inputs are given to the circuit, and after the coded outputs appear in the circuit, the same coded inputs are given to the circuit again which take the circuit back to the neutral state. This signaling scheme is called $2-phase$ protocol [7]. It consists of a computation phase followed by a reset phase. [3], [5], [1], and [2] have suggested asynchronous implementations of Boolean functions using 2-rail code which satisfy this $2-phase$ protocol. However, all the solutions involve asynchronous automata, which make the circuits complicated and slow. The existence of solutions not involving asynchronous automata has been an open question. In this paper we suggest two such solutions.

We begin this paper with the definition of a protocol and the conditions for a protocol to be delay-insensitive. The delay-insensitivity of a protocol guarantees that environment can correctly know the completion of the computation of a circuit which satisfies the protocol, independently of the delays associated to the circuit. We also show several conditions for a circuit to satisfy a delay-insensitive protocol.

Then, we suggest a new delay-insensitive protocol and show that circuits which satisfy the protocol can be implemented without involving asynchronous automata. The protocol is obtained by extending the $2-phase$ protocol, and consists of a computation phase followed by several reset phases which are necessary to take circuits back to the initial state.

Finally, we suggest that the above protocol and circuits can be improved by overlapping a computation phase and a reset phase. Consequently, the number of the reset phases in the protocol is reduced.

2 Protocol

We consider a circuit and an environment that communicate with one another by sending and receiving signals. There are two types of signals: from the environment to the circuit, which we call *input signals*, and from the circuit to the environment, which we call *output signals*. It is assumed that signals are conveyed via wires.

An underlying model for specifying circuit-environment communications in [6] is *trace structure*. There are several ways of representing trace structures. For example, I-nets, which is a class of Petri-nets, interface state graphs, and regular expressions are representations of trace structures. Below we define a model we call *protocol*, which is an abstraction of interface state graphs. We prefer it to other representations, because it allows more compact specifications.

The definition of protocol is parametric to input and output indexes. It is assumed that inputs and outputs are unidirectional. Let i (o) denote a set of *input wires* (*output wires*) of a circuit-environment model. We call input wires inputs and output wires outputs. Each input and output wire takes *values* from the same set D. We call an input valuation in the set $I = [i \to D]$, an *input state*, and an output valuation in the set $O = [o \to D]$, an *output state*.

A protocol is a tuple $P = (D, I_{CD}, O_{CD}, R)$, where D is a set of *values*, I_{CD} is a set of *input codeword classes*, O_{CD} is a set of *output codeword classes*, and R a *protocol relation*.

The set of input codeword classes is a set of nonempty sets of input states, i.e. $I_{CD} \subseteq \mathcal{P}(I)$, where $\mathcal{P}$ is the power set operator. So, each input codeword class is a set of input states. We assume that the input codeword classes are independent, i.e. $\forall x, y \in I_{CD}.x \subseteq y \vee y \subseteq x \Rightarrow x = y$. The set of output codeword classes is defined similarly. We call $I \times O$ the set of *input-output pairs*. The protocol relation or protocol in short, is a set of tuples of input-output pairs $R \subseteq (I \times O) \times (I \times O)$.

In this paper, we assume that the following conditions for a protocol hold:

- The set of input codeword classes is the same as the set of output codeword classes. We call $B = I_{CD} = O_{CD}$ the set of codeword classes.
- The set D of values of a protocol has an ordering structure which can be extended to the states, to the set $(I \times O)$, and to the codeword classes.

Now, we define delay-insensitivity of a protocol. Delay-insensitivity is defined in [6] according to the Foam Rubber Wrapper Postulate from [8]. A mechanism-environment model is delay insensitive if there are no computation and transmission interferences. In [6], the sufficient and necessary requirements for trace structures to be delay-insensitive are given.

Protocol is a representation of a subclass of trace structures. Here, we give the sufficient conditions for a protocol to be delay insensitive.

For codeword classes $\mathbf{X}, \mathbf{Y} \in B$, we call a set $(\mathbf{X} \triangleright \mathbf{Y}) \subseteq (I \times O) \times (I \times O)$ a *phase* if the following conditions hold:

- $\mathbf{X} < \mathbf{Y}$ or $\mathbf{X} > \mathbf{Y}$,
- $\forall a, c \in I. \forall b, d \in O. (ab, cd) \in (\mathbf{X} \triangleright \mathbf{Y}), \mathbf{X} < \mathbf{Y} \Rightarrow \exists k, l \in \mathbf{X}. \exists h, j \in \mathbf{Y}. kl \leq ab < cd \leq hj, b \leq a, d \leq c$,
- $\forall a, c \in I. \forall b, d \in O. (ab, cd) \in (\mathbf{X} \triangleright \mathbf{Y}), \mathbf{X} > \mathbf{Y} \Rightarrow \exists k, l \in \mathbf{X}. \exists h, j \in \mathbf{Y}. kl \geq ab > cd \geq hj, b \geq a, d \geq c$,
- $\forall a, c, e \in I. \forall b, d, h \in O. (ab, cd) \in (\mathbf{X} \triangleright \mathbf{Y}), (cd, eh) \in (\mathbf{X} \triangleright \mathbf{Y}) \Rightarrow (ab, eh) \in (\mathbf{X} \triangleright \mathbf{Y})$,

A protocol $P = (D, B, R)$ is delay-insensitive if the protocol relation R can be uniquely partitioned into phases, and for an injective function $p : B \rightarrow B$ on the codeword classes such that $p(\mathbf{X}) = \mathbf{Y} \Leftrightarrow (\mathbf{X} \triangleright \mathbf{Y})$, $\forall \mathbf{X} \in B. p^{|B|}(\mathbf{X}) = \mathbf{X}$ holds.

For example the $2-phase$ protocol for ternary logic is a tuple $2-phase =< D, B, R >$, where $D = \{0, 1, S\}$ with the ordering $<= \{(S, 1), (S, 0)\}$, $B = \{\mathbf{S}, \mathbf{C}\}$, with $\mathbf{S} = \{\lambda x.S\}$ and $\mathbf{C} = \{\lambda d | d = 1 \vee d = 0\}$, and the protocol relation is $R = (\mathbf{S} \triangleright \mathbf{C}) \cup (\mathbf{C} \triangleright \mathbf{S})$. $2-phase$ protocol is delay-insensitive.

3 Combinational Circuits

Let Σ be a set of elementary circuits. We call elementary circuits *gates*. With each gate a number of inputs is associated.

Let i be a set of inputs. We define terms T by induction:

- each $x \in i$ is a term in T.
- if A is a n input gate and $t_0, ..., t_{n-1}$ are terms, then $A(t_0, ..., t_{n-1})$ is a term in T.

A set o of outputs is a nonempty subset of terms T. A set of outputs is also called a circuit. So, $\mathcal{C} = \mathcal{P}_f(T)$ is the set of circuits, where $\mathcal{P}_f$ is the nonempty power set operator.

For example, in case we have one gate $\Sigma = \{NAND\}$ and inputs $i = \{a, b, c\}$, then $\{a, NAND(a, a)\}$ is a circuit with two outputs and $\{NAND(a, NAND(a, a))\}$ is a circuit with one output.

Let a domain of values the inputs and outputs can take be D.

For any circuit $C \in \mathcal{C}$ there is a unique function $(\_)^{\#}_C : I \rightarrow O$ which maps an input state $x \in I$ to an output state $(x)^{\#}_C \in O$. We call this function *computation*. (It is usually called truth table).

We also say that the *circuit C is implemented in D*.

For each circuit $C \in \mathcal{C}$ implemented in D, there is a corresponding circuit input-output state relation $R_C \subseteq (I \times O) \times (I \times O)$. An environment assumed for this relation has no restrictions for input changes.

We say that a circuit C *satisfies a protocol* P if $\forall a, c \in I, \forall b, d \in O. (ab, cd) \in P \Rightarrow \forall k, h \in O. (ak, ch) \in R_C \Rightarrow (ak, ch) \in P$.

Sub-circuits of a circuit $C \in \mathcal{C}$ can be made observable. We call such a circuit C_T which is sub-circuit closed a *transparent circuit*.

A circuit C is called monotonic relative to an ordering $\leq$ if $\forall a, b \in I. a \leq b \Rightarrow (a)^{\#}_C \leq (b)^{\#}_C$.

Theorem 1: A circuit C satisfies a protocol P when $C_T = C$, C is monotonic relative to the ordering relation of P, and $(\forall \mathbf{X} \in B, \forall a \in \mathbf{X}. (a)^{\#}_C \in \mathbf{X})$

According to Theorem 1 the class of transparent 2-rail circuits satisfies $2-phase$ protocol.

Theorem 2: A circuit C satisfies a protocol P iff C is monotonic relative to the ordering of P, $(\forall \mathbf{X} \in B, \forall a \in \mathbf{X}. (a)^{\#}_C \in \mathbf{X})$ and $(\forall \mathbf{X}, \mathbf{Y}. (\mathbf{X} \triangleright \mathbf{Y}) \subseteq P \Rightarrow (\forall a, c \in I, \forall b, d \in O. ((ab, cd) \in (\mathbf{X} \triangleright \mathbf{Y})$ and $c \notin \mathbf{Y}) \Rightarrow (c)^{\#}_C \notin \mathbf{Y}))$.

Theorem 3: If a binary gate $\sigma \in \Sigma$ satisfy a protocol P and circuits $C_1, C_2 \in \mathcal{C}$, satisfy the protocol P, then $\sigma(C_1, C_2)$ satisfies the protocol P.

4 Implementation of Boolean Functions Under 5-phase Protocol

We consider a set of values $X_9 = \{0, 1, A, D, M, N, B, \overline{A}, \overline{B}\}$ and an embedding of X_9 into binary tuples $B^4 = B \times B \times B \times B$ as defined in the Figure 1, so that the ordering on B^4 is preserved.

A	0000
1	0010
0	0001
D	0011
M	0110
N	0101
B	0111
$\overline{A}$	1111
$\overline{B}$	1000

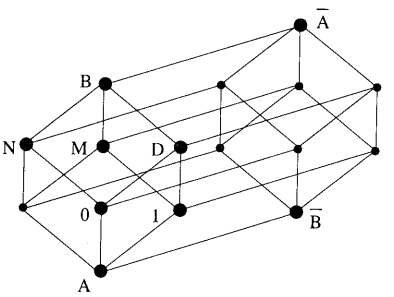

図 1: Embedding of X_9 into B^4.

Values 0 and 1 of X_9 denote the embedded binary values of $B = \{0, 1\}$ into X_9.

We define a $5-phase$ protocol as a tuple $5-phase =< X_9, B, R >$. The ordering on X_9 is the one defined by the embedding of X_9 into B^4. The defined codeword classes are

$$\begin{aligned} B \quad = \{ \quad & \mathbf{C} = \{\lambda x.d | d = 1 \vee d = 0\}, \\ & \mathbf{A} = \{\lambda x.A\}, \\ & \mathbf{B} = \{\lambda x.B\}, \\ & \overline{\mathbf{A}} = \{\lambda x.\overline{A}\}, \\ & \overline{\mathbf{B}} = \{\lambda x.\overline{B}\}\}. \end{aligned}$$

It is clear from the definition that the codeword classes are independent. The elements of the codeword class $\mathbf{C}$ are all binary valued state. Codeword classes $\mathbf{A}$, $\mathbf{B}$, $\overline{\mathbf{A}}$, $\overline{\mathbf{B}}$ are one element sets. We call the elements of these codeword classes the neutral states.

The protocol relation is defined as $R = (\mathbf{A} \triangleright \mathbf{C}) \cup (\mathbf{C} \triangleright \mathbf{B}) \cup (\mathbf{B} \triangleright \overline{\mathbf{A}}) \cup (\overline{\mathbf{A}} \triangleright \overline{\mathbf{B}}) \cup (\overline{\mathbf{B}} \triangleright \mathbf{A})$.

$5-phase$ protocol is delay-insensitive by definition.

We consider a set of gates $\Sigma = \{\vee, \wedge, \neg\}$ and the set X_9 as the domain of values the inputs and outputs of the circuits can take. It is not important that we choose these gates, because we are going to give interpretations for a general binary gate and an inverter gate.

The definitions of the general binary gate and the unary gate $\neg_{X_9}$ are given in the Figure 2.

The general binary gate is not completely defined. The input values of the undefined places in the definition of the general binary gate can never be given simultaneously to a binary circuit under the $5-phase$ protocol. In case we talk of some particular implementations we must define them completely. Our aim is to demonstrate the possibility of self-timed implementation of Boolean functions, and for this purpose the partial definition of the gates is sufficient.

	A	0	1	D	M	N	B	$\overline{A}$	$\overline{B}$
A	A	A	A	—	—	—	—	—	$\overline{B}$
0	A			D	D	D	D	—	—
1	A		BG	D	D	D	D	—	—
D	—	D	D	D	D	D	D	—	—
M	—	D	D	D	D	D	D	—	—
N	—	D	D	D	D	D	D	—	—
B	—	D	D	D	D	D	B	B	—
$\overline{A}$	—	—	—	—	—	—	B	$\overline{A}$	$\overline{A}$
$\overline{B}$	$\overline{B}$	—	—	—	—	—	—	$\overline{A}$	$\overline{B}$

	$\neg$
A	A
0	1
1	0
D	D
M	N
N	M
B	B
$\overline{A}$	$\overline{A}$
$\overline{B}$	$\overline{B}$

図 2: Definitions of a binary gate of X_9 and $\neg_{X_9}$.

The box denoted BG in the Figure 2 must be filled with the interpretation of a corresponding binary Boolean gate. The gates of Σ satisfy $5-phase$ protocol.

The circuits generated by Σ function under $5-phase$ protocol as follows. Computation of a circuit C starts with the input and output states being neutral states $\lambda x.A$. In the first phase of the computation the input state of the circuit is changed to a codeword from the codeword class $\mathbf{C}$. The output state of the circuit is following the change to a codeword. The first phase will end with the input state $x \in \mathbf{C}$ and output state $(x)^{\#}_C \in \mathbf{C}$. The rest of the phases are devoted to changing the input and output states back to the neutral state A.

We can say that we have one computation phase per four reset phases. Can we do better?

Observing the lattice structure in the Figure 1, we see that the phase $(\overline{\mathbf{A}} \triangleright \overline{\mathbf{B}})$ of the $5-phase$ protocol can be replaced with the phases $(\overline{\mathbf{A}} \triangleright \overline{\mathbf{C}})$ and $(\overline{\mathbf{C}} \triangleright \overline{\mathbf{B}})$ assuming that we can have two types of codewords. In the next section we present a solution for this protocol.

5 Implementation of Boolean Functions Under $2 \times 3 - phase$ Protocol

We consider a set of values $X_{14} = \{0, 1, A, D, M, N, B, \overline{0}, \overline{1}, \overline{A}, \overline{D}, \overline{M}, \overline{N}, \overline{B}\}$. There are two types of values in the set X_{14}. Values with over-line and values without over-line. We define an embedding of X_{14} into B^4 similarly to the embedding X_9 into B^4. The definition is in the Figure 3.

We also assume an ordering on X_{14} defined by the embedding. There are over-lined and non-over-lined representations for the binary values in X_{14}.

We define a $2 \times 3 - phase$ protocol as a tuple $2 \times 3 -$

A	0000	1111
1	0010	1110
0	0001	1101
D	0011	1100
M	0110	1010
N	0101	1001
B	0111	1000

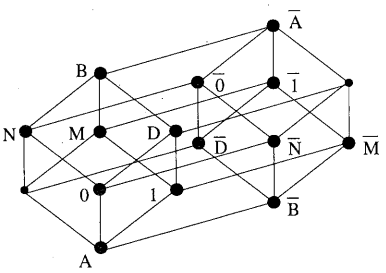

図 3: Embedding of X_{14} into B^4.

$phase =< X_{14}, B, R >$. The defined codeword classes are

$$\begin{aligned} B = \{ \quad & \mathbf{C} = \{\lambda x.d | d = 1 \vee d = 0\}, \\ & \mathbf{A} = \{\lambda x.A\}, \\ & \mathbf{B} = \{\lambda x.B\}, \\ & \overline{\mathbf{C}} = \{\lambda x.d | d = \overline{1} \vee d = \overline{0}\}, \\ & \overline{\mathbf{A}} = \{\lambda x.\overline{A}\}, \\ & \overline{\mathbf{B}} = \{\lambda x.\overline{B}\}\}. \end{aligned}$$

The codeword classes are also over-lined and non-over-lined as are the codewords. Similarly to the $5-phase$ protocol the codeword classes $\mathbf{A}, \mathbf{B}, \overline{\mathbf{A}}, \overline{\mathbf{B}}$ are one element sets and the elements of these classes are called the neutral states.

The protocol relation is defined as $R = (\mathbf{A} \triangleright \mathbf{C}) \cup (\mathbf{C} \triangleright \mathbf{B}) \cup (\mathbf{B} \triangleright \overline{\mathbf{A}}) \cup (\overline{\mathbf{A}} \triangleright \overline{\mathbf{C}}) \cup (\overline{\mathbf{C}} \triangleright \overline{\mathbf{B}}) \cup (\overline{\mathbf{B}} \triangleright \mathbf{A})$.

The circuits under this protocol function in two modes. In the over-lined mode and in the non-over-lined mode.

$2\times3-phase$ protocol is delay-insensitive by definition.

We consider the same set of gates $\Sigma = \{\vee, \wedge, \neg\}$ as in the previous section. The set of the values the inputs and outputs of the circuits can take is X_{14}. The partial definition of a general binary gate is given in the left table in the Figure 4. The definition involves the non-over-lined values. The definition of the general binary gate for the over-lined values is symmetric to the definition of if for the non-over-lined values. The box denoted by BG in the Figure 4 must be filled with the interpretation of the corresponding binary gate. The inverter gate in the right part of the Figure 4, is similarly partially defined. The interpreted gates of Σ satisfy $2 \times 3-phase$ protocol.

The circuits generated by Σ function under $2 \times 3-phase$ protocol as follows. There are two opposite modes of computation. Computation of a circuit C starts with the input and output states being neutral states $\lambda x.A$.

	A	0	1	D	M	N	B	$\overline{A}$
A	A	A	A	—	—	—	—	—
0	A	BG		D	D	D	D	—
1	A			D	D	D	D	—
D	—	D	D	D	D	D	D	—
M	—	D	D	D	D	D	D	—
N	—	D	D	D	D	D	D	—
B	—	D	D	D	D	D	B	B
$\overline{A}$	—	—	—	—	—	—	B	$\overline{A}$

	¬
A	A
0	1
1	0
D	D
M	N
N	M
B	B

図 4: Definitions of a binary gate of X_{14} and $\neg_{X14}$.

The first phase of computation is the same as in the case of $5-phase$-protocol. The functioning of the circuit C under the phases $(\mathbf{C} \triangleright \mathbf{B}), (\mathbf{B} \triangleright \overline{\mathbf{A}})$ of the protocol result in the initial neutral state of the opposite mode of the computation, the input and output states being $\lambda x.\overline{A}$. The functioning of the two modes is identical.

Computation of the circuits under $2 \times 3-phase$ protocol us faster than computation of the circuits under $5-phase$ protocol, but the gates are bigger

6 Concluding remarks

In this paper we have demonstrated combinational circuit solutions for the implementation of delay-insensitive boolean functions. In the proposed computation completion signaling solutions there are more than one reset phase per computation phase. In the case of $5-phase$ protocol there are four reset phases per computation phase and in the case of $2 \times 3-phase$ protocol there are 2 reset phases per computation phase. In the case of $2 \times 3-phase$ protocol, faster computation is achieved due to the cost of more complex gates. No matter that the circuits in both solutions are big and the computation involves more than one reset phase when nothing 'useful' is done, these are the minimal solutions so far obtained.

参考文献

[1] T.Nanya, M.Kuwako, *"On Signal Transition Causality for Self-Timed Implementation of Boolean Functions"*, Workshop on Asynchronous and Self-timed Circuits and Systems at HICSS-26, pp. 356-368, January 1993.

[2] M.Kuwako, *"Seminar presentation"*, Nov.17.1994.

[3] D.Armstrong, A.D.Friedman, P.R.Menon, *"Design of Asynchronous Circuits Assuming Unbounded Gate Delays"*, IEEE Trans. on Computers, Vol. C-18, No 12, Dec. 1969.

[4] T.Verhoeff, *"Delay-insensitive codes - an overview "*, Cistributed Computing 3:1-8. 1988.

[5] I.David, R.Ginosar, M.Yoeli, *"An Efficient Implementation of Boolean Functions as Self-Timed Circuits"*, IEEE Trans. on Computers, Vol. 41, No 1, Jan. 1992.

[6] J.T.Udding, *"A Formal Model for Defining and Classifying Delay-insensitive Circuits and Systems"*. Distributed Computing 1:197-204. 1986.

[7] C.L.Seitz, *"System Timing"*. In: C.A.Mead, L.A.Conway, *"Introduction to VLSI systems"*. Addison-Wesley, Reading MA, pp 218-262. 1980.

[8] C.E.Molnar, T.Fang, F.U.Rosenberg, *"Synthesis of Delay-insensitive Modules"*. Chapel Hill Conference on VLSI, Chapel Hill, NC, pp 67-86. May 1985.