Asynchronous Modular Arbiter

J. CALVO, J. I. ACHA, AND M. VALENCIA

Abstract — **A practical *N*-user arbiter and its implementation are presented in this correspondence. Because of the asynchronous character of its input variables (request signals), the design proposed is asynchronous and keeps in mind the possibility of metastable operations. The structure of the arbiter is very simple and modular.**

Index Terms — **Asynchronous arbiter, asynchronous logic design, conflict resolution, metastable operation, modular control logic.**

I. INTRODUCTION

When in a digital system there are multiple processors operating under the control of independent clocks, and they share the use of a common resource, a circuit is necessary to arbitrate its use and to resolve the conflicts that may occur. A valid solution to this problem would be to design a control mechanism (*N*-user arbiter) that satisfies the following general conditions.

1) Only one processor may use the resource shared at any one time.

2) Any request for using the common resource must be serviced in a finite time.

An arbiter is not a sequential circuit operating in fundamental mode. Because of the asynchronous character of its input variables (request signals), they may change at arbitrary times, independent of one another, and then it is necessary to define the proper operation of the circuit under the unrestricted input change assumption [5]. Therefore, with reference to arbiters, the traditional design approaches are inadequate [1]. However, we think that these methods can be used for designing some parts of the arbiter; normally, they will make the design easier. On the other hand, it is necessary to bear in mind the possibility of metastable operations in the arbiter's memory elements. Although it is an inevitable problem [8], [9], the designs must tend to diminish the probability of any anomalous behavior, and if possible, such situations should not affect the output signals of the arbiter.

The purpose of this correspondence is to present a design of a simple and modular *N*-user asynchronous arbiter that takes into account the metastable operation. The design is based on the experience of other earlier work [1]–[4]. However, we dedicate special attention to design simplicity taking advantage of the power of traditional design methods whenever possible. In Sections II-B and II-C the modularity equations of the arbiter are given and a detailed study of the control circuit timing is carried out, relating the complexity of the control circuit and the number of processors that must share the resource. Finally, in Section III we discuss the failure probability of the arbiter due to metastable operations and estimate this probability in a concrete realization of the arbiter.

II. THE ARBITER

Communication between each processor P_i, $i = 1, \cdots, N$, and the arbiter takes place by means of two wires: the request wire r_i and the acknowledge wire a_i. The processor P_i communicates that it wants to use the common resource and sends a request signal by switching its wire r_i to 1. The arbiter, according to its priority scheme, will give permission to use the common resource to processor P_i with an acknowledge signal by switching its wire a_i to 1. When the P_i has finished using the common resource, it resets its request line ($r_i = 0$), which causes the arbiter to reset the corresponding acknowledge line ($a_i = 0$). That is, we adopt the signaling convention shown in Fig. 1 [1], and in agreement with condition 1),

$$a_i a_j = 0, \quad \forall i, j, \quad \text{and } i \neq j. \tag{1}$$

Manuscript received June 11, 1984; revised November 27, 1984.
The authors are with the Departamento de Electricidad y Electrónica, F. Fisica, Universidad de Sevilla, Sevilla, Spain.
IEEE Log Number 8405249.

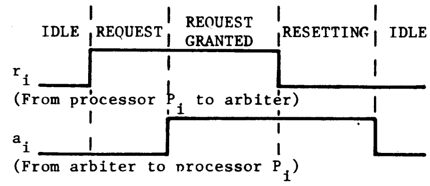

Fig. 1. Signaling conventions for arbiters (not to scale).

We propose a design scheme which is based on the following rules.

a) The arbiter must have one module M_i for each processor P_i that has access to the shared resource. Each module M_i will be made up of a memory element to record the request and additional gates for determining the processor P_i priority and for generating the acknowledge signal.

b) The *N*-user arbiter will be realized by a linear array of *N* such modules. The priority is determined by the order of the modules in the array.

c) A signal C_o controls the intervals of time where the request signals must be recorded in the arbiter.

d) The signal C_o will order a new recording ($C_o = 1$) of request signals only when all requests recorded in the last iteration are serviced and the common resource can be used again.

e) The design must be asynchronous, simple, and must keep in mind the possibility of metastable operations.

A. Memory Element

In agreement with the above rules, the state table, the transition table, and the implementation of the memory element are shown in Fig. 2. Notice that the memory element can record request $r_i = 1$ to use the common resource only if $C_o = 1$. When $C_o = 0$, if the request was recorded ($Q_i = 1$), the memory element will change state when the request signal r_i switches to 0. After this, where C_o is 0, the memory element will remain inhibited for any change of r_i.

B. Priority Scheme

After each process of request storage, the arbiter initiates a new iteration of acknowledgments to use the common resource from among the processors P_i that had made the requests. The arbiter will grant the use of the resource to the processor P_i iff the following hold.

1) Its request was recorded ($Q_i = 1$).

2) All processor P_j, with $j < i$, requests were serviced.

That is, $\forall i$ with $1 \leqq i \leqq N$:

$$a_i = Q_i b_{i-1} \tag{2}$$

where

$$b_i = \overline{Q}_i b_{i-1}. \tag{3}$$

If we include the necessary gates for realizing (2) and (3), the module design M_i is shown in Fig. 3. Of course, for M_1, b_o must always be 1.

We must complete the arbiter design with the control circuit for generating the signals C_o and I_a. The function of the latter signal will be shown later.

C. Control Circuit

In the *N*-user arbiter that we propose, the control signals C_o and I_a are generated from the output b_N of module M_N. In agreement with (3), the switch of signal b_N to 1 indicates that all requests have been serviced. From this moment the arbiter must initiate a new iteration and record the new requests to use the common resource. For the sake of discussion, the design of the control circuit is represented in Fig. 4, and its significant signals are shown in Fig. 5.

Supposing that all gates used in the design have the same delay Δt_g, the pulse b_N duration Δt_b has a minimum value equivalent to the delay of five gates ($5\Delta t_g$). This minimum value would corre-

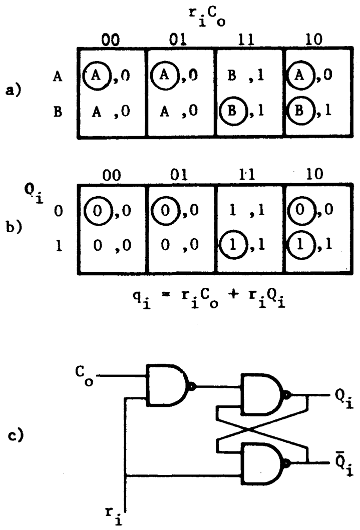

Fig. 2. Memory element of the module M_i. (a) State table. (b) Transition table. (c) Circuit.

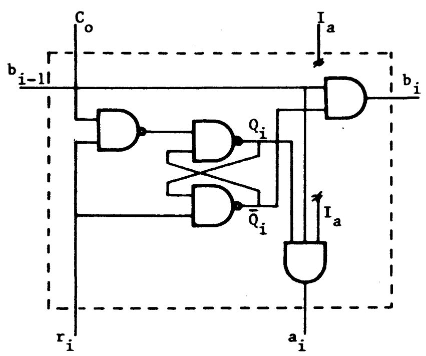

Fig. 3. Module M_i of the arbiter.

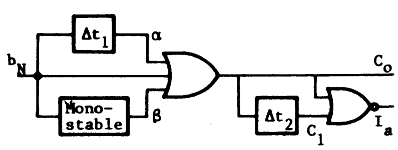

Fig. 4. The arbiter's control circuit.

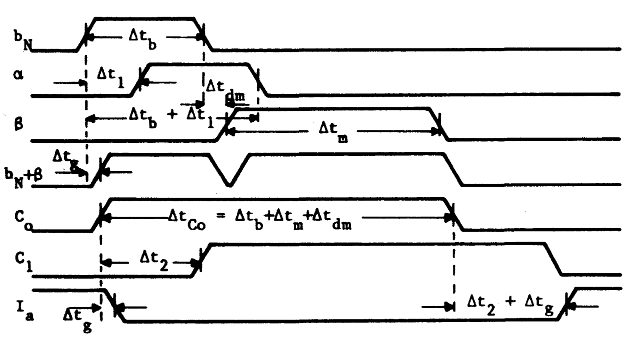

Fig. 5. Signals in the control circuit.

spond to the situation in which processor P_N had requested the use of the resource. The greatest value of Δt_b is not bounded because in a situation with all $r_i = 0$, all $a_i = 0$ and $b_N = 1$, the arbiter would be waiting for some processor to request the use of the resource.

The monostable is triggered by the negative-going edge of the pulse in the signal b_N. This means that at least one request $r_i = 1$ has been recorded and

$$b_j = 0, \quad \forall j \geqq i. \tag{4}$$

The purpose of the delay Δt_1 is to avoid runt pulses (see $\beta + b_N$ in Fig. 5), in the signals C_o and I_a, which could cause metastable operations in some memory element. Logically, its value must be smaller than the minimum value of Δt_b and larger than the propagation delay of monostable Δt_{dm}, i.e.,

$$\Delta t_{dm} < \Delta t_1 < 5\Delta t_g. \tag{5}$$

For typical values of Δt_g and Δt_{dm} of gates and monostables with the same technology [6], a simple solution for Δt_1 could be two or four inverters in cascade connection.

The signal I_a is obtained from the NOR operation of signals C_o and C_1 where C_1 is C_o delayed an interval Δt_2. The function of I_a is to inhibit the outputs a_i where I_a is 0. The purpose of the delay Δt_2 is dual:

1) to inhibit the outputs a_i of possible metastable operations caused in some memory element because of the simultaneity between the negative-going edge of the pulse in the signal C_o and the switching to 1 of some signal r_j;

2) to guarantee that all signals in the modules are stable before the arbiter grants the initial use of the resource to a processor, in accordance with the priority scheme.

Let us assume the situation indicated in (4) and $C_o = 1$, when the processor P_k requests ($r_k = 1$) the use of the common resource at time t_o. If we designate t_s as the time in which all signals in the modules M_k through M_{i-1} are stable again, the value of $t_s - t_o$ would be

$$t_s - t_o = 3\Delta t_g + \begin{cases} (i-k)\Delta t_g & \text{if } k < i \\ 0 & \text{if } k > i \end{cases} \tag{6}$$

where the first term on the right side of (6) represents the time interval for the transition from stable state ($Q_k = 0$, $r_k C_o = 01$) to stable state ($Q_k = 1, r_k C_o = 11$) of the memory element in the module M_k, and the second term represents the time spent on the transmission of logical value 0 from b_k to b_{i-1}. According to (4), when $k > i$, this second term is zero.

To determine Δt_2, we must select the worst case, which occurs when $i = N$ and $k = 1$ in (6) and, because of the request of P_1, the module M_1 is in the situation indicated in point 1) above. Then, Δt_2 must be

$$\Delta t_2 > 3\Delta t_g + (N-1)\Delta t_g = (N+2)\Delta t_g. \tag{7}$$

That is,

$$\Delta t_2 = (N+2)\Delta t_g + \Delta t \tag{8}$$

where the term Δt must cover the duration of the metastable operation. Logically, as we will discuss in Section III, the larger Δt is, the smaller the probability that some output a_i will show a possible metastable operation.

On the other hand, in agreement with the function of I_a, it is obvious (see Fig. 5) that the delay Δt_2 must be smaller than the minimum pulse duration in the signal C_o (Δt_{Co}),

$$\Delta t_2 < \Delta t_{Co}|_{\min}. \tag{9}$$

The last expression suggests to us some simplifications (which reduce the circuit cost) of the control circuit that we have proposed. The utility of these simplifications will depend on the number N of processors and the particular characteristics of the resource shared. We discuss three solutions.

Solution 1: We eliminate the monostable and the delay Δt_1. Notice that, in this case, the gate OR in Fig. 4 is not necessary. Then,

$$C_o \equiv b_N, \quad \Delta t_{Co}|_{\min} = \Delta t_b|_{\min} = 4\Delta t_g.$$

From (8) and (9),

$$4\Delta t_g > \Delta t_2 = (N+2)\Delta t_g + \Delta t. \tag{10}$$

Obviously, N must be smaller than 2, and then this solution is not valid.

Solution 2: We eliminate only the monostable.
Supposing that $\Delta t_1 = 4\Delta t_g$,

$$\Delta t_{Co}|_{\min} = (\Delta t_b + \Delta t_1)|_{\min} = 9\Delta t_g.$$

From (8) and (9),

$$9\Delta t_g > \Delta t_2 = (N + 2)\Delta t_g + \Delta t. \quad (11)$$

This solution is valid, but the number N of processors is determined by the estimated value for Δt. In any case, $N < 7$.

Solution 3: Design of Fig. 4.
For each value of Δt_m,

$$\Delta t_{Co}|_{\min} = \Delta t_b|_{\min} + \Delta t_m + \Delta t_{dm} = 5\Delta t_g + \Delta t_m + \Delta t_{dm}$$

with

$$\Delta t_m + \Delta t_{dm} > 4\Delta t_g. \quad (12)$$

From (8) and (9),

$$5\Delta t_g + \Delta t_m + \Delta t_{dm} > \Delta t_2 = (N + 2)\Delta t_g + \Delta t. \quad (13)$$

It is evident from the last two expressions that the monostable output pulse is necessary whenever $N \geqq 7$. Taking into account the fixed values of N and Δt, the value of Δt_m must be chosen in agreement with (13). Then, this solution is valid in all cases.

III. Discussion of the Metastable Operation

In each module M_i the core of the memory element is a simple R–S flip-flop constructed by cross-tying two NAND gates. With relation to metastable operation, numerous studies and experimentation have been carried out on this device [10]–[14]. For our purpose, we wish emphasize the works of Chaney and collaborators [15], [17]; they have provided abundant experimental data for a good number of bistable devices, relating the temporal characteristics of input signals and some parameters of the devices with probabilistic measures of metastable operation.

For the sake of discussion, we will designate event A as being the negative-going edge of the pulse in signal C_o and event B as being the switching to 1 of the signal r_i. In the arbiter that we propose, the metastable operation in some memory element can only be caused by the simultaneity between events A and B, that is, by the transition r_iC_o: 01 → 10 in the state table of Fig. 2. This situation is shown schematically in Fig. 6.

We define the resolution time of the memory element as the time it takes to produce logically defined and stable outputs, after the time of ocurrence of its excitation. In good behavior conditions this resolution time will be the normal propagation delay time (ζ), which we have estimated as $3\Delta t_g$. When the device is in a metastable state, its resolution time is nondeterministic.

In agreement with [15], [16], a characteristic of the memory element is a time interval $[t_1, t_2]$, termed glitch window, whose location and width ($\delta = t_2 - t_1$) are defined with respect to event A. If the relative delay t_d between events A and B is within this interval ($t_d \in [t_1, t_2]$), the memory element will be set in a metastable state where t_1 is small enough so that the memory always records the request to use the common resource within its specified propagation delay time, and t_2 is sufficiently large so that event B is never recognized.

Assuming that event A occurs at time $t = 0$ and that the probability density function $f(t_d)$ is uniform over the interval δ, that is,

$$f(t_d) = \begin{cases} 0 & \text{if } t_1 > t_d \text{ or } t_d > t_2 \\ \dfrac{1}{\delta} & \text{if } t_1 \leqq t_d \leqq t_2 \end{cases} \quad (14)$$

we can define $F(\Delta t)$ as the probability that the resolution time exceeds Δt. Hurtado [7] has shown that for $\Delta t > \zeta$ this probability can be approximated by

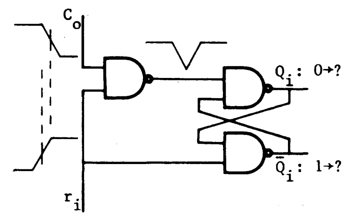

Fig. 6. The simultaneity between events A and B can cause metastable operations.

TABLE I

Δt(ns)	F(Δt)	FI One failure every
30	$2.8\ 10^{-6}$	34 seconds
40	$1.1\ 10^{-8}$	2.5 hours
50	$4.3\ 10^{-11}$	27 days
60	$1.6\ 10^{-13}$	19 years
70	$6.4\ 10^{-16}$	4982 years

$$F(\Delta t) = \frac{T_o}{\delta} e^{-\Delta t/\tau}, \qquad \text{with } \Delta t > \zeta \quad (15)$$

where the parameters T_o, δ, and τ are characteristic to each bistable device.

As an example, we wish to estimate $F(\Delta t)$ and its variation for a particular realization of the R–S flip-flop constructed by cross-tying two NAND gates of chips 74S00 [TTL(S)]. From the data supplied by Chaney [17], for this realization $t = 1.8$ ns, $T_o = 1$ s, and $\zeta = 17$ ns.

Although the δ value is not available to us, we can use 20 ns as a pessimistic estimation for it. If we assume further that the average number of arbiter iterations (the average times that event A occurs) is 10^4 times per second, we can calculate the expected failure interval FI. Table I takes in these results.

We wish to point out that the development carried out in this Section is based on one memory element only. Therefore, the estimated values for $F(\Delta t)$ do not correspond to the failure probability, due to metastable situations in an arbiter with N modules. However, they represent a very good approximation. Take into account that during intervals of requests recording, the modules are statistically independent. The worst case occurs when the first module in the array goes into metastable operation. Otherwise, in agreement with (8), the memory element will have a time interval greater than the fixed value of Δt, in order to resolve its metastable situation. Moreover, if the module in the anomalous situation does not have the highest priority in the iteration, the above time interval will be increased by the time the processors, with higher priority, take to use the resource.

IV. Conclusions

The arbiter described in this correspondence uses a request–acknowledge signaling convention presented by Plummer. The arbiter operates by iterations, each of which begins with a process of request storage. After that, in accordance with a priority scheme for a linear selection, the arbiter initiates a process of acknowledgments to use the common resource from among the processors P_i that have made the requests. Only when all requests have been serviced will a new iteration begin. In this asynchronous and modular design, two characteristics have been pursued: simplicity of design and the intent to reduce the problems related to possible metastable situations.

References

[1] W. W. Plummer, "Asynchronous arbiters," *IEEE Trans. Comput.*, vol. C-21, pp. 37–42, Jan. 1972.
[2] R. C. Pearce, J. A. Field, and W. D. Little, "Asynchronous arbiter module," *IEEE Trans. Comput.*, vol. C-24, pp. 931–932, Sept. 1975.

[3] P. Corsini, "Self-synchronizing asynchronous arbiter," *Digital Processes,* vol. 1, pp. 67–73, 1975.
[4] ——, "Speed-independent asynchronous arbiter," *Comput. Digital Techniques,* vol. 2, no. 5, pp. 221–222, Oct. 1979.
[5] S.H. Unger, "Asynchronous sequential switching circuits with unrestricted input changes," *IEEE Trans. Comput.,* vol. C-20, pp. 1437–1444, Dec. 1971.
[6] Texas Instruments, *The TTL Data Book for Design Engineers,* Texas Instruments Incorporated, Dallas, TX, 1976.
[7] M. Hurtado, "Dynamic structure and performance of asymptotically bistable systems," D. Sc. dissertation, Dep. Elec. Eng., Washington Univ., St. Louis, MO, May 1975.
[8] M. Hurtado and D.L. Elliott, "Ambiguous behavior of logic bistable systems," in *Proc. 13th Annu. Allerton Conf. Circuit Syst. Theory,* Oct. 1975, pp. 605–611.
[9] D.F. Wann, C.E. Molnar, T.J. Chaney, and M. Hurtado, "A fundamental problem associated with the physical realization of certain classes of Petri nets," presented at the Conf. Petri-Nets and Related Methods, Mass. Inst. Technol., Cambridge, July 1975.
[10] T.J. Chaney and C.E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Trans. Comput.,* vol. C-22, pp. 421–422, Apr. 1973.
[11] G.R. Couranz and D.F. Wann, "Theoretical and experimental behavior of synchronizers operating in the metastable region," *IEEE Trans. Comput.,* vol. C-24, pp. 604–616, June 1975.
[12] M. Pěchouček, "Anomalous response times of input synchronizers," *IEEE Trans. Comput.,* vol. C-25, pp. 133–139, Feb. 1976.
[13] G. Lacroix, P. Marchegay, and P. Nouel, "Critical triggering of integrated flip-flops in synchronizer circuits," *Int. J. Electron.,* vol. 49, no. 3, pp. 179–184, Sept. 1980.
[14] L.R. Marino, "General theory of metastable operation," *IEEE Trans. Comput.,* vol. C-30, pp. 107–115, Feb. 1981.
[15] F. Rosenberger and T.J. Chaney, "Flip-flop resolving time test circuit," *IEEE J. Solid-State Circuits,* vol. SC-17, pp. 731–738, Aug. 1982.
[16] W. Lim and J.R. Cox, "Clocks and the performance of synchronizers," *Proc. IEE,* pt. E, vol. 130, no. 2, pp. 57–64, Mar. 1983.
[17] T.J. Chaney, "Measured flip-flop responses to marginal triggering," *IEEE Trans. Comput.,* vol. C-32, pp. 1207–1209, Dec. 1983.

Deductive Fault Simulation of Internal Faults of Inverter-Free Circuits and Programmable Logic Arrays

FUSUN OZGUNER

Abstract — **A method for the deductive fault simulation of faults in inverter-free circuits is presented. It is shown that in an inverter-free circuit, fault lists on lines with complementary logic values are disjoint, and fault list calculations can be done by performing fewer set operations compared to conventional gate level deductive simulation. Applications of the method to programmable logic arrays (PLA's) and deductive fault simulation of PLA faults are discussed.**

Index Terms — **Deductive simulation, fault simulation, inverter-free circuits, programmable logic arrays.**

I. Introduction

Fault simulation is becoming an increasingly important part of fault diagnosis of digital systems as the circuits become larger and test generation becomes time consuming and expensive. Several effective algorithms for the simulation of stuck-type faults exist in the literature, i.e., parallel, deductive, and concurrent simulation [1], [4]–[6]. Fault simulation algorithms have been compared in several papers [2], [3]. It has been shown that the deductive method is faster than the parallel method for most circuits.

Manuscript received April 29, 1983; revised December 27, 1984.
The author is with the Department of Electrical Engineering, The Ohio State University, Columbus, OH 43210.
IEEE Log Number 8405698.

The deductive method, introduced by Armstrong [4], considers each gate in the circuit, and by analyzing the faults that cause incorrect signals at gate inputs, deduces a list of faults that would cause an incorrect signal at the gate output. By processing all the gates in this manner a list of faults causing incorrect signal values at the circuit outputs can be calculated in one simulation pass.

Deductive simulation involves set operations (union, intersection) on fault lists. It is shown in this paper that list operations on fault lists can be greatly simplified by taking into consideration the structural properties of the modules. A method for the deductive simulation of inverter-free circuits is presented in the following section. It is shown that fault list calculations are greatly simplified compared to the conventional deductive simulation. Results of Section II are applied to the deductive simulation of programmable logic array (PLA) faults in Section III. A two-valued simulation is considered. The single-fault assumption is made throughout the discussions.

II. Gate Level Deductive Simulation of Inverter-Free Circuits

Set operations performed for calculating the fault lists of each gate in deductive simulation take a considerable amount of computation time. The type of set operations performed on the fault lists of gate inputs depends on the gate input and output values. For example, for an n-input AND gate with output value 1 and input fault lists $L_1, L_2, \cdots, L_n$, the output fault list L_{out} is calculated as

$$L_{out} = \bigcup_{i=1}^{n} L_i. \tag{1}$$

If the output of the AND gate is 0, then L_{out} is calculated using the following equation:

$$L_{out} = \left(\bigcap_j L_j^0\right) \cap \left(\overline{\bigcup_k L_k^1}\right) \tag{2}$$

where

L_j^0 fault list of jth input with value 0,
L_k^1 fault list of kth input with value 1.

In each case, the faults of the gate which would affect its output are added to its output fault list.

If the circuit, however, is inverter free, then simpler expressions can be used to calculate fault lists of gates as shown by the following theorem.

Theorem 1: In an inverter-free circuit, internal stuck fault lists of lines with complementary logic values are disjoint.

Proof: The proof follows from the fact that an s-a-0 (s-a-1) fault can only cause erroneous 0's (1's) at gate inputs and outputs on paths from the fault site to the circuit outputs. The fault list of a line with value 1(0) consists of faults that would change it to a 0(1), and in this case these could only be s-a-0 (s-a-1) faults. Therefore, the same fault could not appear on the fault lists of lines with complementary logic values.

Q.E.D.

This means that in fault simulation of an inverter-free circuit by the deductive method, the same fault could not appear on the fault lists of inputs to the same gate with different logic values, and this leads to a simplification of operations performed for the calculations of the gate output fault lists.

In the conventional deductive simulation algorithm, the output fault list L_{out} of an AND gate with output value 0 in the fault-free circuit is calculated using (2), which can be written as

$$L_{out} = L_1^0 \cap L_2^0 \cap \cdots \cap L_j^0 \cap \cdots \cap \overline{L_1^1} \cap \cdots \cap \overline{L_k^1} \cap \cdots \tag{3}$$