MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 30, NO. 2, 2023
journal homepage: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Signal Transition Graphs for Asynchronous Data Path Circuits

A. Kushnerov[1], S. Bystrov[2]

DOI: 10.18255/1818-1015-2023-2-170-186

[1]Independent researcher, Beer-Sheva, Israel.
[2]Independent researcher, Sochi, Russia.

MSC2020: 68W35
Research article
Full text in English

Received May 5, 2023
After revision May 29, 2023
Accepted May 31, 2023

The paper proposes a method for constructing signal transition graphs (STGs), which are directly mapped into asynchronous circuits for data processing. The advantage of the proposed method is that the resulting circuits are not only output-persistent, but also conformant to the environment. In other approaches, the environment is specified implicitly and/or inexactly and therefore they guarantee only output persistence. The conformation can be verified if both the circuit and its environment are specified by STGs. As an example, we consider a module realizing the function AND2. This module can either wait for both 1s or evaluate the function as soon as at least one 0 arrives. For each case, we draw up a separate STG (scenario) and map it into NCL gates. To provide such a mapping, we specify the behaviors of NCL gates by STG protocols. For data path, such an STG always contains alternative branches with the so-called garbage transitions at the gate inputs. The garbage transitions on a certain wire mean that the circuit is sensitive to the delay in this wire. Ignoring the garbage may lead to a violation of conformation or/and output persistence. For example, in the combinational part of the NCL circuits, the garbage appears on the inputs of NCL gates, and therefore these circuits are not delay insensitive.

Keywords: arithmetic; conformation; decomposition; delay in wires; handshake; pipeline; verification; weak causality

INFORMATION ABOUT THE AUTHORS

Alex Kushnerov | orcid.org/0000-0003-3953-1995. E-mail: kushnero@gmail.com
Independent researcher.

Sergey Bystrov
corresponding author | orcid.org/0009-0008-6525-0517. E-mail: bsa1969@yandex.ru
Independent researcher.

For citation: A. Kushnerov and S. Bystrov, “Signal transition graphs for asynchronous data path circuits”, *Modeling and analysis of information systems*, vol. 30, no. 2, pp. 170-186, 2023.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 30, № 2, 2023
сайт журнала: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Графы сигнальных переходов для схем асинхронного тракта данных

А. Кушнеров[1], С. Быстров[2]

DOI: 10.18255/1818-1015-2023-2-170-186

[1]Независимый исследователь, Беэр-Шева, Израиль.
[2]Независимый исследователь, Сочи, Россия.

УДК 004.312.44
Научная статья
Полный текст на английском языке

Получена 5 мая 2023 г.
После доработки 29 мая 2023 г.
Принята к публикации 31 мая 2023 г.

В статье предлагается метод построения графов сигнальных переходов (STG), которые напрямую отображаются в схемы асинхронной обработки данных. Преимуществом предлагаемого метода является то, что полученные схемы не только неизменны по выходу (output-persistent), но и конформны внешней среде. В других подходах среда задаётся неявно и/или неточно, и поэтому они гарантируют только неизменность по выходу. Конформность можно проверить, если как схема, так и её внешняя среда заданы STG. В качестве примера мы рассматриваем модуль, реализующий функцию 2И. Этот модуль может либо ожидать лог. 1 на обоих входах, либо вычислить функцию, как только придёт хотя бы один 0. Для каждого случая мы составляем отдельный STG (сценарий) и отображаем его в элементы NCL. Чтобы обеспечить такое отображение, мы задаём поведение NCL элементов STG протоколами. Для тракта данных такой STG всегда содержит альтернативные ветви с так называемыми мусорными переключениями на входах элементов. Мусорные переключения на определенном проводе означают, что схема чувствительна к задержке в этом проводе. Игнорирование мусора может привести к нарушению конформности и/или неизменности по выходу. Например, в комбинационной части NCL схем мусор появляется на входах NCL элементов, поэтому эти схемы чувствительны к задержкам.

Ключевые слова: арифметика; верификация; декомпозиция; задержка в проводах; конформность; пайплайн; слабая причинность; хэндшейк

ИНФОРМАЦИЯ ОБ АВТОРАХ

Александр Кушнеров | orcid.org/0000-0003-3953-1995. E-mail: kushnero@gmail.com
независимый исследователь.

Сергей Быстров (автор для корреспонденции) | orcid.org/0009-0008-6525-0517. E-mail: bsa1969@yandex.ru
независимый исследователь.

Для цитирования: A. Kushnerov and S. Bystrov, “Signal transition graphs for asynchronous data path circuits”, *Modeling and analysis of information systems*, vol. 30, no. 2, pp. 170-186, 2023.

Introduction

Asynchronous circuits do not use clock to ensure the validity of signals and operate in the mode of request-acknowledge. Like any digital circuits, they are built from logic gates. An output of a logic gate can be either in a stable or in an excited state. A stable state corresponds to the value of the Boolean function of the gate. An excited state is opposite to the value of this function. From an excited state, the gate can either switch to a new stable state or return to the previous one. The effect of returning to the previous stable state is called a hazard[1]. The goal of asynchronous design is hazard-free circuits. Input signals can also be in an excited state, but they produced by the environment, which is not realized by a circuit. Thus, to model the environment, it is more natural to use not Boolean functions, but something else. In this paper we use the event-based model called *Signal Transition Graphs (STGs)*.

In this model, the signals can be input, internal and output. The signal is excited if all the conditions for its switching to a new state are met. Any signal can stay excited for an arbitrary, but finite time. For non-input signals, i. e. for the gates obtained from STG, this means that their delay can be unbounded, but finite. The most important property of STG is output persistence. This property means that non-input signals must switch from an excited to a new stable state. The excitation can be removed only from input signals and only by switching of other input signals. Output persistence guarantees that an STG is mapped into a hazard-free circuit with arbitrary gate delays. In terms of data path design, such circuits are called delay-insensitive circuits free from gate orphans [1]. If we have one STG for the circuit and another one for the environment, we can check if the circuit does exactly what the environment expects from it to do. In other words, the circuit interface must be conformant to the environment and vice versa. The concepts of output-persistence and conformation are considered in more detail in Section 1.

Traditional approaches to designing an asynchronous data path are algebraic. Often they convert the initial combinational logic into a hazard-free one using dual-rail encoding (Table 1). Thus, each initial variable and its inversion are considered as two new signals. These signals may switch independently, but must reset into a spacer (all 0s or all 1s). Such a discipline is called a 4-phase protocol and shown in Fig. 1.

Table 1. Dual-rail encoding

a	b	a1	a0	b1	b0
0	0	0	1	0	1
0	1	0	1	1	0
1	0	1	0	0	1
1	1	1	0	1	0
spacer		0	0	0	0

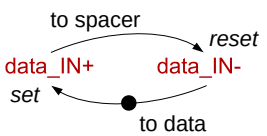


Fig. 1. A 4-phase protocol for input data

STGs are convenient for designing a control path, where the variables are single-rail. This encoding can be viewed as follows. The only value of a variable (e. g. the command “execute”) is encoded by switching a signal from 0 to 1. Switching the same signal from 1 to 0 can be interpreted as a spacer. Thus, the control is just a realization of functions in a unary (1-of-1) encoding. Let this encoding be the first in the sequence, then the second one is 1-of-2 (dual-rail), and the N-th is 1-of-N. We can use STGs for any of these encodings and thereby embed control into data. This means that we do not need to split the circuit into the control and data

[1]Traditionally, hazards are considered in terms of input changes and divided into functional and logic ones.

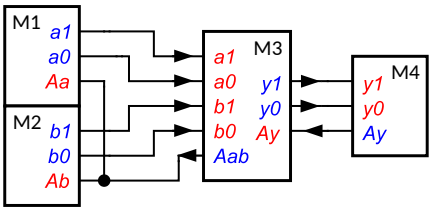


Fig. 2. Interface of the module M3 insensitive to delays in the wires

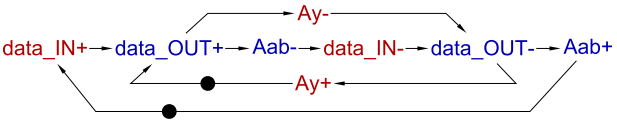


Fig. 3. Interface protocol for the module M3 in Fig. 2, which consists of input (the signals *data_IN*, *Aab*) and output (*data_OUT*, *Ay*) handshakes

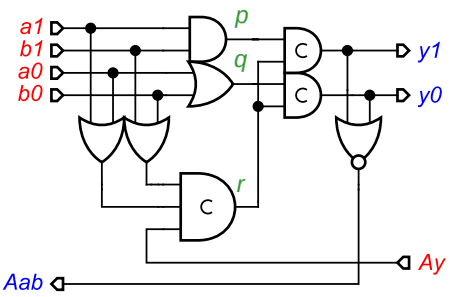


Fig. 4. Dual-rail AND circuit with embedded handshake

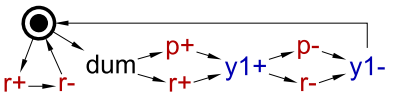


Fig. 5. Behavior of the C-element *y1* in Fig. 4. The transitions *r+*, *r-* in the short alternative branch are garbage that complicates decomposition

path. Moreover, the embedded control will allow us to build circuits according to the modular-hierarchical principle. In some sense, this principle is the opposite of RTL design. In particular, to synthesize an N-bit adder it is sufficient to construct an STG for a *single bit only*.

The 4-phase protocol in Fig. 1 is used only to exchange data. Supplementing it with a control signal, we obtain the handshake protocol, which allows us to organize the interaction between modules. Each handshake is realized by two variables as follows. The sender module sends a dual-rail variable (data) to the receiver module, which sends a unary control variable back to the sender. Let us consider the connection of modules shown in Fig. 2[2]. The module M3 receives the dual-rail variables a and b from the modules M1 and M2, and sends back an acknowledgement *Aab*. Theoretically, M1 and M2 may send data sequentially. In this case, delays in the wires may cause the sequential transitions to occur concurrently. However, the interface of each module is concurrent i. e. the module waits for input data to arrive in any order. Fig. 3 shows the interface protocol for the module M3. This is a generalized 4-phase protocol with two handshakes. The signal *data_IN* stands for a bus of input signals, which are switching concurrently. The signal *Ay* is also switching concurrently with any signal of *data_IN*. Thus, we have the handshake protocol and the concurrent interface[3]. This is what guarantees that the circuit will operate correctly with arbitrary delays of intermodule wires.

A module can realize algorithms to compute multiple functions. The modules can be connected arbitrary. The only restriction is that any ring must contain at least three modules [5]. Such a system may have some degree of concurrency and in a particular case, is a one-dimensional pipeline. The control in this system is entirely local (intermodule). No global control required.

An example of the realization of a simple module is shown in Fig. 4. This is a dual-rail AND circuit with embedded handshake. Note that the inputs of the 3-input C-element switch only once before the element fires. However, for the 2-input C-elements, this is not true. Fig. 5 shows the behavior of the C-element *y1*, which contain two alternative branches. The transition *r+* in the short branch does not cause *y1+*. In the circuit, this *r+* causes *y0+*, which initiates two concurrent processes. One is *Aab-* and reset of *a* and *b* into spacer, and the other is *Ay-*. Both of these processes are synchronized by *r-*. Thus, *r+* is eventually reset to *r-*. We will refer to such alternative branches and transitions as garbage.

[2]At the circuit level, such a structure was proposed first in [2] and formalized at the STG level in [3].

[3]The method proposed in [4] removes immediate relations between input transitions in an STG and makes them concurrent.

a) b)

Fig. 6. Decompositions of C-element verified under the environment in Fig. 5. Operating correctly (a) and incorrectly (b)

Fig. 6 shows two decompositions of the C-element [6], which have been verified under the environment in Fig. 5. The circuit in Fig. 6a is output-persistent and conformant. However, the circuit in Fig. 6b violates the conformation being still output-persistent[4].

Thus, the environment must be specified formally, otherwise we can get a wrong decomposition. In this paper, we propose a method that allows one to specify the behavior of an arbitrary module using STG. The obtained STG can be used for both verification and synthesis. To verify any previously designed circuit, this STG is used as an environment.

The synthesis is a mapping of STG into a library of logic gates. As shown in [5], the library of NCL gates is optimal. This was revealed in experiments with STGs, where the best results are obtained by "mirroring" the data phase into the spacer phase. In this case, for each alternative branch, considered separately, each signal is realized by a C-element. However, such signals can take place in different alternative branches. Hence, NCL gates are the simplest and natural mean to realize them.

1. Theoretical Background

a) Signal Transition Graphs (STGs) [8]. These graphs are used to specify the behavior of asynchronous circuits. An STG is a type of a labeled Petri net, where transitions are associated with the changes in the values of binary signals. For example, $x+$ means the switching of a signal x from 0 to 1, and $x-$ means a 1 to 0 switching. Input, internal and output signals are denoted and processed differently. The arcs in an STG capture the causal relations between the transitions. An STG may contain places with multiple incoming and outgoing arcs. If all arcs outgoing from a place, enter input transitions, such a place models a free (non-deterministic) choice made by the environment.

A signal transition is called excited if all entering it arcs, have tokens. An STG is called *output-persistent* [9] if the excitation is removed in a strictly defined way. For every non-input signal, the excitation is removed only by its firing. For every input signal, the excitation is removed either by its firing or by firing other input signal. The circuit can be converted to the so-called circuit Petri net [10], which itself is a type of STG. For verification, the circuit Petri net is combined (by parallel composition) with an STG that specifies the environment. This gives an STG of the closed system: the outputs of the circuit are the inputs for the environment STG and vice versa.

The circuit is called *conformant* if two conditions are met [10]. On one hand, the environment STG must provide only such transitions of the output signals, which the corresponding circuit Petri net can receive and still remain hazard-free. On the other hand, the circuit Petri net must provide only such transitions of the output signals, which the environment STG expects.

In this paper, we construct STGs by hand, so we need to make them readable. To this end, we use dummy transitions (*dum*) and proxy places. A dummy does not represent any real signal and is just a placeholder. A proxy place is a label for a regular place, from which an arc goes out and/or where it comes in. To verify and map STGs into circuits, as well as for verifying the obtained circuits, we use the Workcraft tool (http://workcraft.org). If an STG with dummies is used as an environment, Workcraft can verify the circuit for conformation. However, to verify the circuit or STG for output persistence, one needs to contract dummies.

[4]The problem of circuits not conformant to the environment was first considered by Izumi Kimura in [7].

At some point, we sacrifice formalities for readability and use dummies to specify weak (OR) causality. Namely, we specify a contradictory STG that violates *output determinacy* [11]. Such an STG cannot be mapped into a circuit, and if it is used as an environment, even the correct circuit will violate conformation. We propose a way around this obstacle.

b) NCL gates [12]. These gates are a special case of a generalized C (gC)-element. It is defined by the self-dependent expression $y(x, y) = S(x) \vee y\overline{R}(x)$, where $S(x)$ and $R(x)$ are set and reset functions. These functions must be orthogonal [9], that is, meet the condition $S(x)R(x) = 0$. Under this condition, the regular Boolean function $f(x, y) = S(x) \vee y\overline{R}(x)$ is monotone. A function $f(z_1, \dots, z_m)$ is called positive unate in variable z_i if $f(z_i = 1) \geqslant f(z_i = 0)$. A function positive unate in *all* variables, is called monotone. For NCL gates, the number of variables in $x = (x_1, \dots, x_n)$ is limited to $n \leqslant 4$. The NCL gates are used under the 4-phase protocol with zero spacer, hence their reset function $R(x) = \overline{(x_1 + \dots + x_n)}$, and the set functions $S(x)$ are different monotone functions.

2. Protocols for NCL gates

To guarantee output persistence, any logic gate must operate under a certain protocol. For NCL gates, we consider two types of protocols: with full and incomplete indication. The protocol with full indication realizes only *strong* (AND) causality. In this case, each transition is enabled to fire only after all of its immediate causes have fired. As shown in [5] (and outlined in Introduction), the protocols with full indication are realized best by NCL gates. In the case of *weak* (joint OR) causality [13] a transition is enabled to fire if at least one of its immediate causes has fired. Thus, there can be different variants of weak causality. The protocol with incomplete indication can realize both strong causality and all variants of weak causality.

a) Full indication. In protocols of this type each product term (conjunction) in the set function is given by an alternative branch, whose signals are synchronized in the set and reset phases. Fig. 7 shows the protocol with full indication and garbage branches for the NCL gate TH23w2 (the set function is *A+BC*). To verify this STG for output persistence, we need to contract the dummy. An equivalent STG without dummies can be obtained by translating the ProFlo expression [14]:

{B+;B-#C+;C-#A+;F+;A-;F-#(B+|C+);F+;(B-|C-);F-}

where the operators ';', '#'and '|'denote sequential composition, choice and concurrency respectively[5].

b) Incomplete indication. We can extend the protocol with full indication in different ways, each of which gives its own protocol with incomplete indication. For example, in the STG in Fig. 7 each alternative branch on the right may contain all the three input signals as shown in Fig. 8. This protocol realizes the only variant of weak causality possible for *A+BC*. In general case, each variant corresponds to a subset (from at least two conjunctions) of the set of conjunctions of the set function. For example, the set function *A+B+CD* contains four subsets: *A+B*, *A+CD*, *B+CD*, *A+B+CD*. Each of them represents a variant of weak causality.

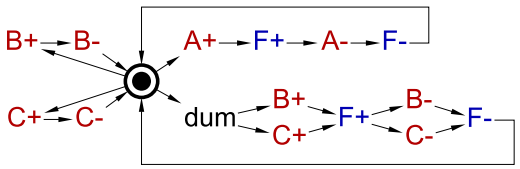


Fig. 7. Protocol with full indication and garbage branches for TH23w2 (A+BC)

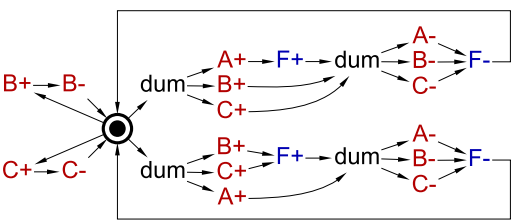


Fig. 8. Protocol with incomplete indication and garbage branches for TH23w2. Without timing constraints, the output determinacy is violated

[5]Currently, all signals in ProFlo are internal, i. e. we need to assign inputs and outputs. Otherwise, both output persistence and output determinacy are violated.

Let us consider the STG in Fig. 8 in more detail. The branches on the right contain the data phase and the spacer phase. In the data phase there are signals that are not indicated. In the upper branch this is *B+* and *C+*, and in the bottom one this is *A+*. In the spacer phase *A-*, *B-*, *C-* are indicated in both branches. On the other hand, in the upper branch *F+* is excited by *A+*, and in the bottom one — by both *B+* and *C+*. So, we have a contradiction that cannot be realized by the circuit and is a particular case of violation of output determinacy [11].

However, we presume that in the upper branch *A+* occurs earlier than either *B+* or *C+*, and in the bottom one *A+* occurs later than both *B+* and *C+*. These timing constraints are orthogonal and applied only to the input transitions. Hence, we can model them by interleaving [15]. Thus, the STG in Fig. 8 is converted into the output-determinate form. An equivalent interleaved STG without dummies is obtained by translating the ProFlo expression:

{B+;B-#C+;C-#(A+;(F+|(B+;C+))#A+;(F+|(C+;B+))#B+;A+;(F+|C+)#
B+;C+;(F+|A+)#C+;A+;(F+|B+)#C+;B+;(F+|A+));(A-|B-|C-)F-}

To guarantee output determinacy for both types of the protocols, we need to make sure that:

1) No set of immediate causes of the output signal in the data phase can be a strict subset of another set of immediate causes of the same signal in another branch (absorbed conjunction). For example, *A+BC+AB=A+BC* and therefore in Fig. 7 and Fig. 8 the branch, where *A+* and *B+* are the only immediate causes of *F+* is prohibited.

2) No garbage branch contains any set of immediate causes of the output signal in the data phase. In Fig. 7 and Fig. 8 the sets prohibited for the garbage branches are not only {A+}, {B+|C+}, but also all possible supplements: {A+|B+}, {A+|C+}, {A+|B+|C+}.

3. Proposed Method

The initial specification for the method is the truth tables of Boolean (or multiple-valued) functions[6]. Based on the truth tables, we construct an STG for the module and map this STG into a circuit. The protocols with full and incomplete indication are used as templates for the mapping. Let us demonstrate the method on the example of the AND function *y=ab*. The module realizing this function and its interface are shown in Fig. 2. According to the truth table, we represent the dual-rail variables taking the value "1", as shown in Table 2. Abbreviation "Sc." in this table means scenarios. We will introduce them later.

Table 2. Truth table of the AND function (a) and its dual-rail representation (b)

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

a)

Sc.	*as*	*bs*	*ys*
	a0	*b0*	*y0*
1.1	*a0*	*b1*	*y0*
	a1	*b0*	*y0*
1.2	*a1*	*b1*	*y1*

b)

The STG construction process consists of 6 steps (no iterations):

1. Analyze the truth table and elaborate a way to get functions. Since data is encoded by "+" transitions, it is sufficient to draw up an STG for the data phase only. This step largely determines the complexity of the resulting circuit.
2. Insert ("+") transitions of the input acknowledgement signals. According to the handshake protocol, they must precede the ("+") transitions of the functions. Since in the used NCL protocols the spacer phase is completely determined by the data phase, we can find the set functions. The number of variables in these functions must not exceed 4, otherwise decomposition is necessary.

[6]Multiple-valued (symbolic) STGs were introduced in [16]. To map such an STG into a circuit, one needs to take additional steps, which are not automated yet.

3. Decomposition. Insert ("+") transitions of internal signals before the transitions of those signals that are not realized by NCL gates. The goal is to reduce the amount of immediate causes for both the output and internal transitions.
4. Complement the STG with the spacer phase and then realize the output handshakes. The data phase is always mirrored into the spacer phase. In addition, in the protocols with incomplete indication, the signals non-indicated in the data phase, are indicated in the spacer phase. To realize the output handshakes, each "+" ("-") transition of the output signal must be an immediate cause of "-" ("+") transition of the corresponding input acknowledgement.
5. Provide the NCL protocols for all non-input signals. To this end, we need to establish new mediate relations between three sequential events: cause ["+" ("-") transition of non-input data], midterm event ["-" ("+") transition of output acknowledgement] and effect ["-" ("+") transition of input data]. In general case, these relations can be realized in several different ways. Finding the minimal realization is the classical set cover problem [5, 17]. However, there is a universal (but not always optimal) solution: to combine all the individual output acknowledgements into one, whose transitions are immediate effects of all output functions.
6. If the number of variables in the set function of the output acknowledgement exceeds 4, decompose it (as at Step 3).

From the obtained STG we can find the protocol under which any gate x operates. To this end, we need to convert all signals, except x and its immediate causes, to dummies. Contracting these dummies, we get the protocol with garbage transitions.

a) Variant 1. Using only strong causality.

Step 1. In the dual-rail representation in Table 2b *ys* depends on two variables *as* and *bs* that arrive concurrently. Although one of them can arrive earlier, we will not analyze this situation. Thus, *ys* waits for both *as* and *bs* to arrive, as shown in Fig. 9.

Step 2. The only option to insert the input acknowledgement *Ay+* is shown in Fig. 10. From this figure we find the set function *ys=as·bs·Ay*. Substituting here the variables from Table 2b, we split *ys* into *y0=Ay·(a0·b0+a0·b1+a1·b0)* and *y1=Ay·a1·b1*. Since *y0* depends on 5 variables, a decomposition is needed.

Step 3. Let us introduce Scenario 1.1 for *y0* and Scenario 1.2 for *y1*, as shown in Table 2b. Each scenario describes the behavior of the module for a certain set of input combinations. For different scenarios these sets do not overlap. To decompose *y0*, we insert *x+* into Scenario 1.1 as shown in Fig. 11. Since the NCL protocols are symmetric, any decomposition is output persistent. Now, the set functions are: *x=as·bs=a0·b0+a0·b1+a1·b0* and *y0=x·Ay*. For Scenario 1.2 the set function is the same *y1=Ay·a1·b1*.

Step 4. For the protocols with full indication, the spacer phase is a mirror reflection of the data phase. Thus, to realize the output handshakes, we establish immediate relations between *y0+*, *y1+ (y0-, y1-)* and *Ay- (Ay+)*, as shown in Fig. 12.

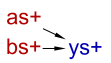


Fig. 9. STG obtained at Step 1

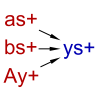


Fig. 10. STG obtained at Step 2

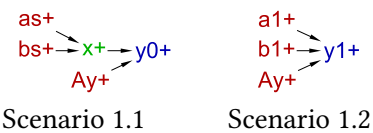


Fig. 11. STG obtained at Step 3

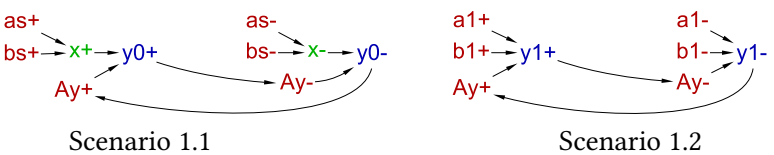


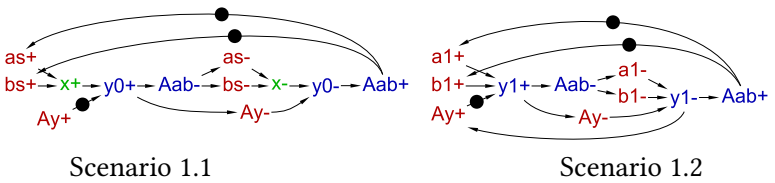
Fig. 12. STG obtained at Step 4

Table 3. All necessary mediate relations for Variant 1

Sc.	*cause*	*effect*
1.1	x+ or y0+	as-
		bs-
	y0+	as- or bs-
1.2	y1+	a1-
		b1-

Table 4. Minimal set of relations covering Table 3

Sc.	*cause*	*effect*
1.1	y0+	as-
		bs-
1.2	y1+	a1-
		b1-



Scenario 1.1 Scenario 1.2

Fig. 13. STG obtained at Step 5

Step 5. As at the previous step, to provide the NCL protocols for *x*, *y0*, *y1*, it is sufficient to establish relations between the data phase and the spacer phase. All the necessary mediate relations are shown Table 3. Minimizing the relations in Scenario 1.1, we obtain Table 4. Now, in Scenario 1.1 (1.2) the cause of *as- (a1-)* and *bs- (b1-)* is the same *y0+ (y1+)*. Therefore, all the relations for *a* and *b* must be realized via the same midterm transition *Aab-*.

Fig. 13 shows the obtained STG with mirrored new relations, which allow us to write *Aab=!(y0+y1)*.

Substituting into this STG the variables from Table 2b, we get the specification of strongly indicating AND circuit shown in Fig. 14.

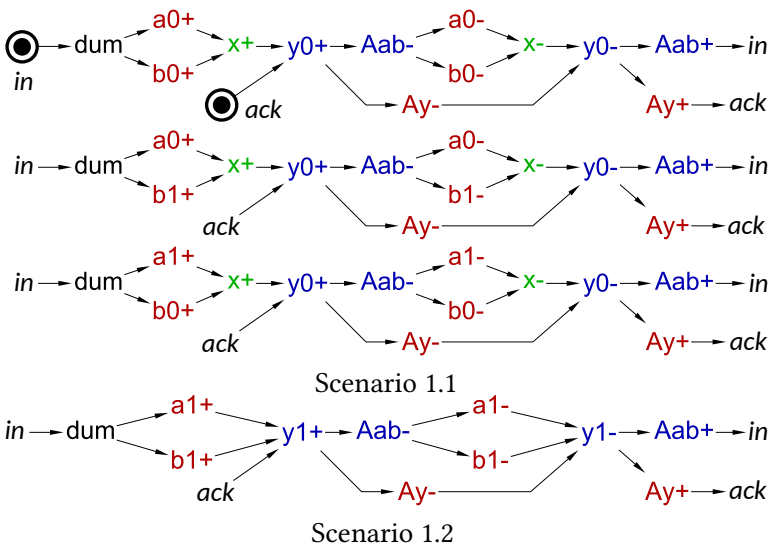


Fig. 14. STG specification of strongly indicating AND circuit with handshake

It is evident from Fig. 14 that the protocols for *x*, *y0*, *y1* are similar to the one shown in Fig. 7, i. e. are the protocols with full indication and garbage branches. We have already obtained all the equations of the gates, so the STG in Fig. 14 is mapped into the circuit shown in Fig. 15. To realize this circuit in the static CMOS, at least 46 transistors are required [15].

Let us now return to the circuit in Fig. 4 and try to embed the gates *p* and *q* into the C-elements *y1* and *y0*. To this end, we modify the STG in Fig. 14 as shown in Fig. 16. Let the signal *x* in this figure be the inversion of some signal *w*. This signal can be realized by the NCL gate TH24comp followed by a C-element.

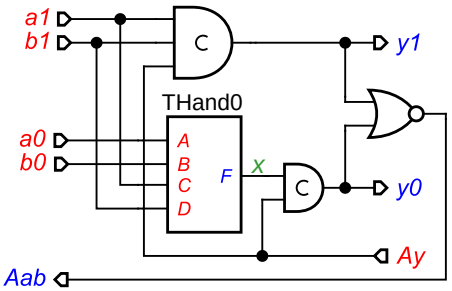


Fig. 15. AND circuit. Variant 1, obtained from the STG in Fig. 14

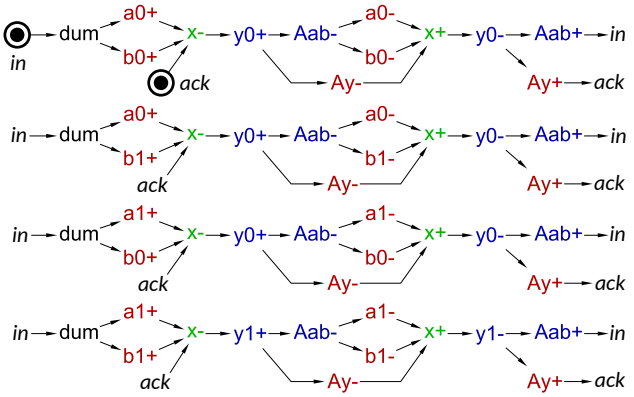


Fig. 16. Modification of the STG in Fig. 14

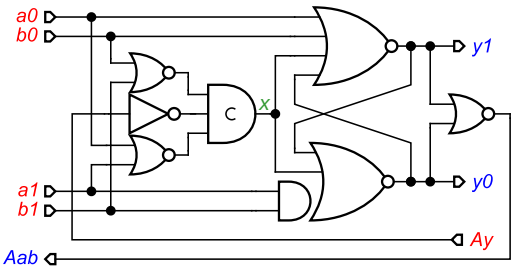


Fig. 17. SR-latch based variant of the circuit in Fig. 15

The set function of TH24comp is *S=a0·b0+a0·b1+a1·b0+a1·b1 or S=(a1+a0)·(b1+b0).* Hence, the signal *w* can be realized by the completion detector from the circuit in Fig. 4. The signals *y0*, *y1* in the STG in Fig. 16 are mapped into the SR-latch on complex gates, as shown in Fig. 17[7]. As compared to the circuit in Fig. 15, this circuit requires 42 transistors instead of 46.

b) Variant 2. Using both strong and weak causality (early propagation [19] or early evaluation [20]).

Step 1. From the dual-rail representation in Table 2b we can see that for *y0* it is sufficient to have either *a0* or *b0.* To describe this situation, we consider two cases. In the first case *a0* arrives earlier than *b0* and vice versa. This is Scenario 1.1. In the second case *a0 (b0)* switches concurrently with a don't-care term *b1 (a1).* This is Scenario 1.2. Let us denote in Scenario 1.1 the variable that arrives earlier by *N0*, and the variable that arrives later — by *Ms.* In Scenario 1.2 they arrive concurrently. For *y1* we need both *a1* and *b1.* This is Scenario 2. The variables encoded in each scenario are given in Table 5. The obtained STG is shown in Fig. 18.

Table 5. Scenarios and encoding for Variant 2

Sc.	*as*	*bs*	*N0*	*Ms*	*ys*
1.1	-	-	*a0*	*b0*	*y0*
	-	-	*b0*	*a0*	*y0*
1.2	-	-	*a0*	*b1*	*y0*
	-	-	*b0*	*a1*	*y0*
2	*a1*	*b1*	-	-	*y1*

Step 2. The only option to insert the transition *Ay+* of the input acknowledgement into both scenarios and follow the handshake protocol is shown in Fig. 19.

[7]Such circuits were first proposed in [18]. The principle of their operation is that data blocks that arm of the latch, which should remain in zero.

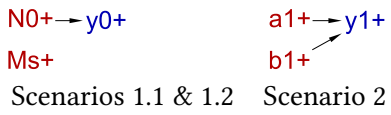


Fig. 18. STG obtained at Step 1

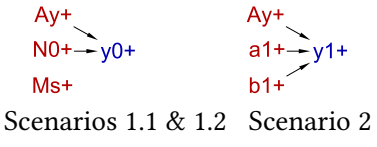


Fig. 19. STG obtained at Step 2

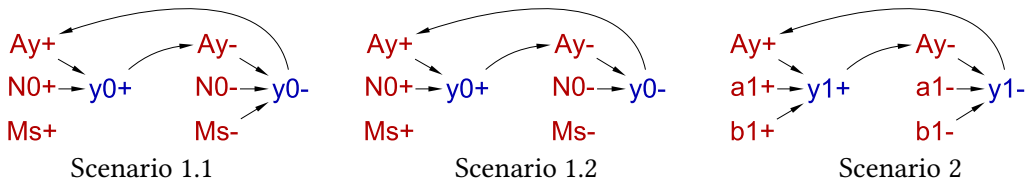


Fig. 20. STG obtained at Step 4

Substituting into Fig. 19 the variables from Table 5, we can write the set functions: *y0=Ay·N0=Ay·(a0+b0)* and *y1=Ay·a1·b1*. The number of variables in these functions does not exceed 4, therefore the decomposition (Step 3) is not needed.

Step 4. Scenario 1.1 realizes weak causality such that *Ms+* is not indicated by *y0+* in the data phase. In the spacer phase both of *N0-* and *Ms-* must be indicated by *y0-*. Scenarios 1.2 and 2 realize strong causality and therefore, the spacer phase is the mirror reflection of the data phase. To realize the output handshakes, we establish new immediate relations from *y0+ (y1+)* to *Ay-* and from *y0- (y1-)* to *Ay+* as shown in Fig. 20.

Step 5. To provide the NCL protocols for *y0* and *y1*, we establish the mediate relations given in Table 6.

Table 6. All necessary mediate relations for Variant 2

Sc.	*Com.*	*cause*	*effect*
1.1	*	y0+	N0- Ms-
	**	Ms+	N0- Ms-
1.2	*	y0+	N0-
	***	Ms+	Ms-
2	*	y1+	a1- b1-

* mirrored for spacer-data; ** only for data-spacer;
*** necessary only for sign alternation, mirrored for spacer-data.

The relations in this table cannot be minimized, so we establish them as is. Note that the effect of some relations is a nominal transition (*N0-* or *Ms-*) decoded as either *as-* or *bs-*. Hence, in such a relation, the midterm transition must be common for *a* and *b*. This is transition *Aab-*. Note also that in Fig. 20, *N0+* and *Ms+* can be decoded as the same real transition *a0+*. Therefore, we need to establish in Scenario 1.1 and 1.2 an additional relation between *N0+* and *Aab-*. The resulting STG is shown in Fig. 21.

From Fig. 21 we find the set function *Aab=N0·Ms·y0+a1·b1·y1* and decode *N0* and *Ms* using Table 5. As a result, we obtain *Aab=(a0·b0+a0·b1+a1·b0)·y0+a1·b1·y1*, which is a function of 6 variables and therefore must be decomposed.

Step 6. To decompose *Aab*, we insert three new internal signals: *a*, *b*, *y*. In Scenario 1.1 and 1.2 the signals *a* and *b* are encoded by nominal signals *N* and *M* as shown in Table 7. Fig. 22 shows the STG obtained at Step 6.

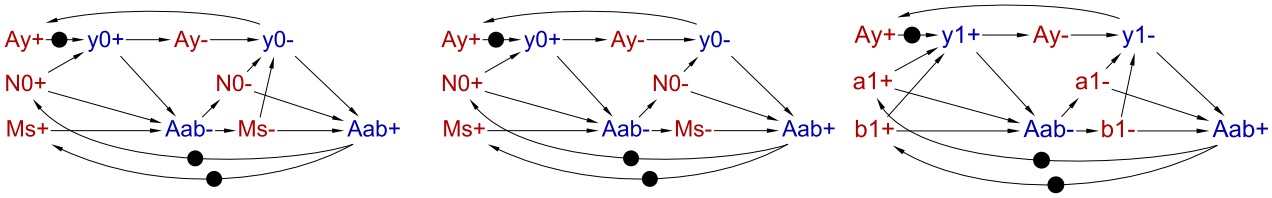


Fig. 21. STG obtained at Step 5

Table 7. Extending Table 5 by adding the columns N and M

Sc.	*as*	*bs*	*N0*	*Ms*	*N*	*M*	*ys*
1.1	-	-	*a0*	*b0*	*a*	*b*	*y0*
	-	-	*b0*	*a0*	*b*	*a*	*y0*
1.2	-	-	*a0*	*b1*	*a*	*b*	*y0*
	-	-	*b0*	*a1*	*b*	*a*	*y0*
2	*a1*	*b1*	-	-	-	-	*y1*

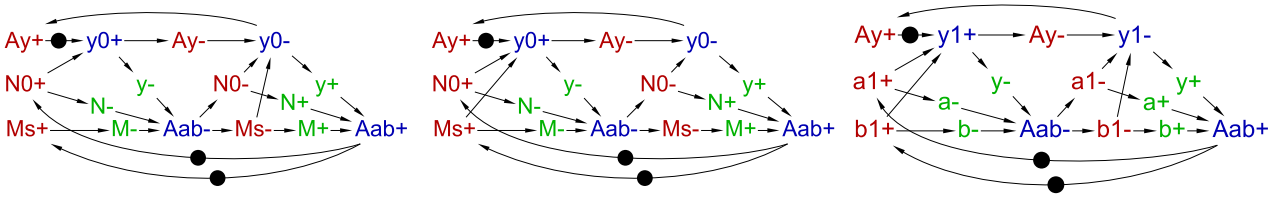


Fig. 22. STG obtained at Step 6

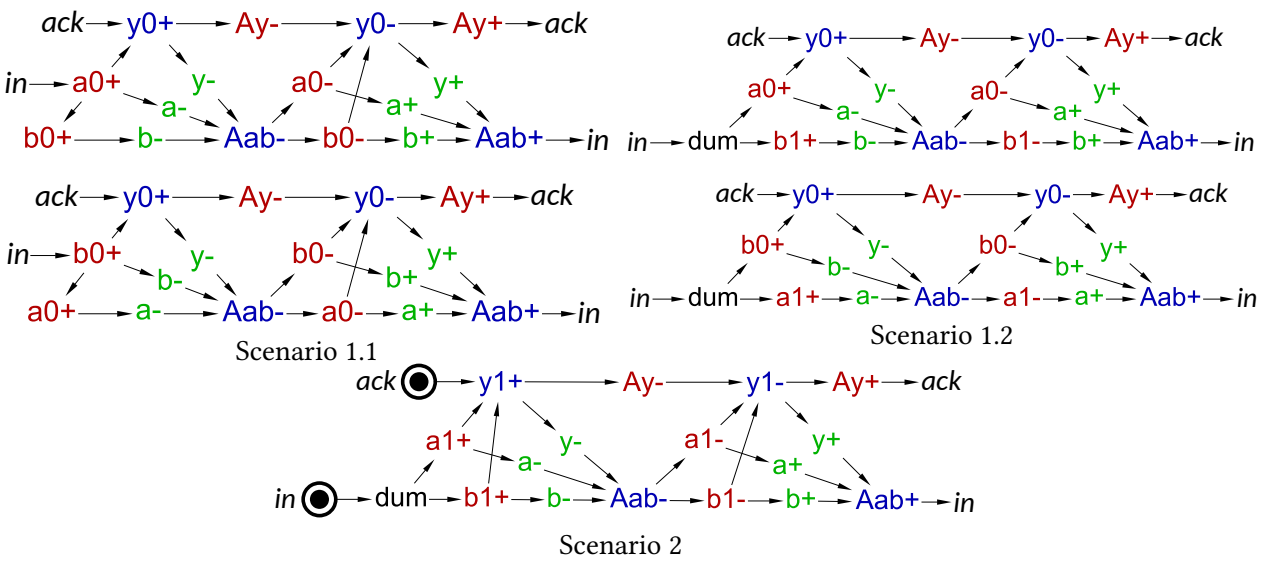


Fig. 23. STG specification of weakly indicating AND circuit with handshake

Substituting into this STG the variables from Table 7, we get the set function *Aab=a·b·y* and *a=!(a1+a0)*, *b=!(b1+b0)*, *y=!(y1+y0)*. In each equation, the number of variables does not exceed 4, so further decomposition is not needed. The above substitution gives the full STG of weakly indicating AND circuit shown in Fig. 23.

We have already obtained all the equations of the gates, so the STG in Fig. 23 is mapped into the circuit shown in Fig. 24. To realize this circuit in the static CMOS, at least 50 transistors are required [15].

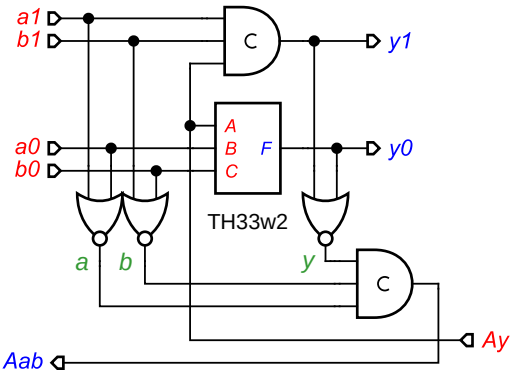


Fig. 24. AND circuit. Variant 2, obtained from the STG in Fig. 23

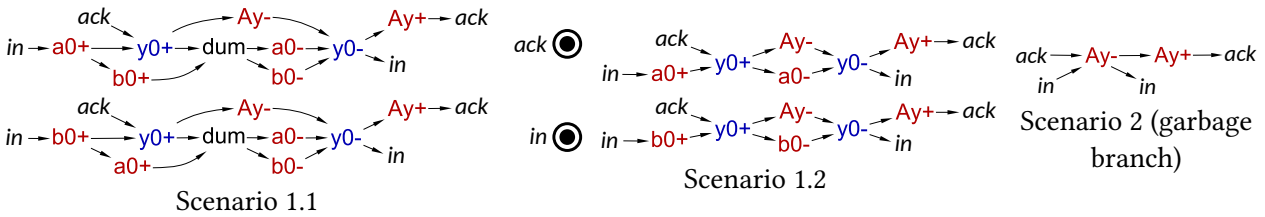


Fig. 25. Protocol with incomplete indication for *y0*

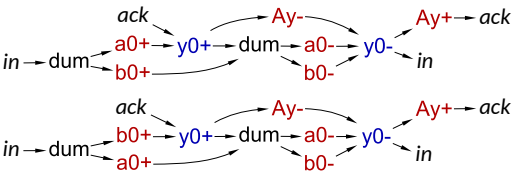


Fig. 26. Non-output-determine STG for Scenario 1.1 in Fig. 25

Let us now make sure that the behavior in Fig. 23 is mapped into the circuit in Fig. 24. To this end, we need to find a protocol under which every gate operates and compare it with the template for the corresponding NCL gate. As an example, we consider the signal *y0*, whose set function *y0=Ay·(a0+b0)*. To find the protocol for *y0*, we convert to dummies the transitions of those signals in Fig. 23, which are not causes for *y0*. Then, we contract extra dummies and obtain the STG shown in Fig. 25.

Let us consider Scenario 1.1 in Fig. 25. In the upper branch, *a0+* arrives earlier than *b0+*, and the bottom one — vice versa. Thus, we have timing constraints. Returning to the STG in Fig. 8, we specify Scenario 1.1 by non-output-determinate STG as shown in Fig. 26. The timing constraints make it output-determinate.

Thus, the obtained protocol for *y0* (Fig. 26 along with Scenarios 1.1 and 2 in Fig. 25) corresponds to the template in Fig. 8. In other words, the signal *y0* is indeed realized by the NCL gate TH33w2 as shown in the circuit in Fig. 24.

4. Related Works

Prior to this work, STGs had never been used to synthesize data path circuits at the level of gates. In terms of indication, early arithmetic circuits used only strong causality. The handshake in these circuits is realized on registers. We have shown that this is not necessary. In the so-called DIMS circuits [2, 21], dual-rail data signals are synchronized on multi-input C-elements, whose outputs are then collected by OR gates. Such circuits are two-level and therefore have large overhead in area. In contrast to DIMS, the so-called crossed implementation [18, 22] is a multilevel, purely combinational dual-rail circuit. To generate

a completion signal, all internal dual-rail signals in such a circuit are ORed and then collected by a tree of C-elements. Thus, the area occupied by the completion detector can be very large.

There is a variant of the crossed implementation [23] obtained by direct mapping of a Binary Decision Diagram (BDD). Note that transitions to spacer can be detected on the power rails, which in turn, can be used for attacks. On the other hand, logic layers in dual-rail circuits can have the opposite spacer. In particular, the corresponding variant of the cross-implementation [19] was designed especially for security purposes. To simplify the completion detection, one can use the so-called layer-wise optimization [24]. It should be taken into account that this optimization is based on relative timing, i. e. the circuit becomes sensitive to gate delays.

To optimize DIMS, it is necessary to find such C-elements that do not have garbage transitions at their inputs, as well as OR gates to which these C-elements are connected. The inputs without garbage, as well as all inputs of the corresponding OR gate, are inputs of the equivalent circuit. Its output is connected to the node where the output of the OR gate was connected. The equivalent circuit is similar to the completion detector with OR gates. Such an optimization is realized implicitly when a Multi-valued Decision Diagram (MDD) is mapped into a circuit [25].

Both DIMS and the crossed implementations can be optimized if the input signals of one gate are indicated at the output of some other gate. This principle defines the conditions of the so-called weak indication [26]. We used the same principle in the protocols with incomplete indication that realize both strong and weak causality. For these protocols, there are at least two different approaches to DIMS optimization [1, 27]. In the former approach, the optimization is deeper that allows one to convert DIMS into an NCL circuit. Then, the area or delay of this circuit is minimized. The crossed implementation for the above protocols can be optimized by a method based on solving Boolean equations [28].

A typical representative of circuits operating under the protocol with incomplete indication is the NCL full adder [29]. A simpler full adder circuit was realized on transistors [30]. However, in this circuit, the output carry rails do not return to zero. Instead, the previous data values are kept on the output capacitances. For these rails, we can either organize a 2-phase protocol or make the assumption that they are updated earlier than the sum. To design and optimize transistor circuits, similar to the one discussed above, there is a systematic approach [31].

Ad hoc algebraic techniques to embed the handshake into the NCL modules and thus to get rid of registers in the NCL pipelines are considered in [32–34]. The way to link these algebraic techniques with the presented STGs could be as follows. Let an STG is specified by the ProFlo expression and converted to a state transition diagram (using a finite state transducer). The operators of the ProFlo language on this diagram can represented by the operators of Tsirlin's algebra [22, 35].

Conclusion and Discussion

In this paper, the asynchronous data path is considered as a set of communicating dual-rail arithmetic circuits operating under the 4-phase protocol. We proposed a method for specifying such circuits by STGs. These STGs are correct by construction and are mapped into output-persistent circuits. To verify previously designed circuits, the obtained STGs can be used as an environment. The proposed method is based on the use of the protocols with full and incomplete indication. The latter is more complicated, but gives additional options for optimization. In both protocols, the data phase is mirrored into the spacer phase. If some signals are not indicated in the data phase, they are indicated in the spacer phase. This allows us to use the full potential of NCL gates.

The protocols with full indication use only strong causality. This is the 1st variant, which gives the circuits in Fig. 15 and in Fig. 17. The protocols with incomplete indication use both strong and weak causality. This is the 2nd variant, which gives the circuit in Fig. 24. This circuit requires 20 % more transistors (50 vs. 42) than the SR-latch based circuit (the 1st variant) in Fig. 17, but has a lower latency.

Note that the function AND2 is a special case of MAJ3 (carry in full adder). Moreover, the dual-rail MAJ3 is functionally complete, since we can invert the dual-rail signal by swapping the wires. A realization of MAJ3 on the SR-latch, whose arms are 5-input complex gates *!((a·b+c·(a+b))·d·e)* was proposed in [36].

Asynchronous circuits are sensitive to delays in some wires. To find such wires, we need to consider each gate separately and obtain the protocol under which it operates. In this protocol we distinguish between signals at the inputs and at the output of the gate. If all input transitions are indicated at the gate output, the corresponding input wires (from the gate to the fork) can have arbitrary delay. For each non-indicated transition there is a fork with unsafe wire. The same fork may correspond to a non-indicated transition in another protocol. We distinguish between two types of non-indicated transitions.

A non-indicated transition of the 1st type (for example, *x+*) precedes the opposite transition of the same signal (*x-*), which is followed by the output transition. A non-indicated transition of the 2nd type occurs concurrently with the output transition. For the considered protocols, this means that garbage transitions are of the 1st type, and weak causality generates transitions of the 2nd type. A circuit with unsafe wires retains all the properties (output-persistence, etc.), if the delays in these wires satisfy certain inequalities. Namely, the delay of each unsafe wire must be shorter than the delay of the corresponding adversary path [37]. Violation of this condition takes us beyond the STG model and leads to hazards. In terms of data path design, this situation is called wire orphan [1].

Acknowledgements

The authors would like to thank Dr. Victor Khomenko, who kindly answered our questions regarding the ProFlo language.

References

[1] C. Jeong and S. M. Nowick, “Optimization of robust asynchronous circuits by local input completeness relaxation”, in *IEEE Asia and South Pacific Design Automation Conference*, 2007, pp. 622–627.

[2] D. E. Muller, “Asynchronous logics and application to information processing”, *Switching Theory in Space Technology*, vol. 4, pp. 289–297, 1963.

[3] A. Yakovlev, “Designing self-timed systems”, *VLSI systems design*, no. 9, pp. 70–90, 1985.

[4] H. Saito, A. Kondratyev, J. Cortadella, L. Labagno, and A. Yakovlev, “What is the cost of delay insensitivity?”, in *IEEE/ACM International Conference on Computer-Aided Design*, 1999, pp. 316–323.

[5] S. Bystrov and A. Kushnerov, *Asynchronous data processing. Behavior analysis*, preprint, 2022. DOI: 10.13140/RG.2.2.14748.26248. [Online]. Available: https://www.researchgate.net/publication/362910934_Asynchronous_Data_Processing_Behavior_Analysis.

[6] A. Kushnerov, M. Medina, and A. Yakovlev, “Towards hazard-free multiplexer based implementation of self-timed circuits”, in *27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2021, pp. 17–24.

[7] I. Kimura, “Extensions of asynchronous circuits and the delay problem. Part II: Spike-free extensions and the delay problem of the second kind”, *Journal of Computer and System Sciences*, vol. 5, no. 2, pp. 129–162, 1971.

[8] L. Rosenblum and A. V. Yakovlev, “Signal graphs: From self-timed to timed ones”, in *International Workshop on Timed Petri Nets*, IEEE, 1985, pp. 199–206.

[9] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic Synthesis for Asynchronous Controllers and Interfaces*. Springer Science & Business Media, 2002, 273 pp.

[10] I. Poliakov, A. Mokhov, A. Rafiev, D. Sokolov, and A. Yakovlev, “Automated verification of asynchronous circuits using circuit Petri nets”, in *14th IEEE International Symposium on Asynchronous Circuits and Systems*, 2008, pp. 161–170.

[11] V. Khomenko, M. Schaefer, and W. Vogler, "Output-determinacy and asynchronous circuit synthesis", *Fundamenta Informaticae*, vol. 88, no. 4, pp. 541–579, 2008.

[12] K. M. Fant, *Logically determined design: clockless system design with NULL convention logic*. John Wiley & Sons, 2005, 310 pp.

[13] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny, "On the models for asynchronous circuit behaviour with OR causality", *Formal Methods in System Design*, vol. 9, pp. 189–233, 1996.

[14] V. Khomenko, M. Koutny, and A. Yakovlev, "Slimming down Petri boxes: Compact Petri net models of control flows", in *33rd International Conference on Concurrency Theory (CONCUR 2022)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, vol. 243, 2022, 8:1–8:16.

[15] A. Kushnerov and S. Bystrov, *On minimal realization and behavior of NCL gates*, preprint, 2022. DOI: 10.13140/RG.2.2.31525.47847. [Online]. Available: https://www.researchgate.net/publication/363918158_On_Minimal_Realization_and_Behavior_of_NCL_Gates.

[16] A. Yakovlev and A. I. Petrov, "Symbolic signal transition graphs and asynchronous circuit design", Technical Report Series 395. Department of Computing Science, Tech. Rep., 1992, 40 pp.

[17] O. Coudert, "Two-level logic minimization: An overview", *Integration*, vol. 17, no. 2, pp. 97–140, 1994.

[18] V. I. Varshavsky, Ed., *Aperiodic Automata*. Nauka, 1976, 424 pp., in Russian.

[19] D. Sokolov, "Automated synthesis of asynchronous circuits using direct mapping for control and data paths", Ph.D. dissertation, University of Newcastle upon Tyne, 2006, 203 pp.

[20] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin, "Elastic circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1437–1455, 2009.

[21] D. Hammel, "Ideas of asynchronous feedback networks", in *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, IEEE, 1964, pp. 4–11.

[22] V. I. Varshavsky, Ed., *Self-timed control of concurrent processes. The design of aperiodic logical circuits in computers and discrete systems*. Kluwer Academic Publishers, 1990, 408 pp.

[23] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, and A. Takamura, "TITAC: Design of a quasi-delay-insensitive microprocessor", *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 50–63, 1994.

[24] A. Mokhov, D. Sokolov, and A. Yakovlev, "Completion detection optimisation based on relative timing", in *Proceedings of the Eighteenth UK Asynchronous Forum*, 2006, pp. 73–76.

[25] B. Folco, V. Brégier, L. Fesquet, and M. Renaudin, "Technology mapping for area optimized quasi delay insensitive circuits", in *Vlsi-Soc: From Systems To Silicon. IFIP International Federation for Information Proc*, Springer, vol. 240, 2007, pp. 55–69.

[26] C. L. Seitz, "System timing", *Introduction to VLSI systems*, pp. 218–262, 1980.

[27] W. Toms and D. Edwards, "Prime indicants: A synthesis method for indicating combinational logic blocks", in *15th IEEE Symposium on Asynchronous Circuits and Systems*, 2009, pp. 139–150.

[28] L. P. Plekhanov, "Synthesis of self-timed combinational sections using the functional method", *Systems and Means of Informatics*, vol. 27, no. 2, pp. 85–97, 2017, in Russian.

[29] D. A. Duncan, G. E. Sobelman, and K. M. Fant, *Null convention adder*, US Patent 5,793,662, 1998.

[30] V. I. Varshavsky, A. Y. Kondratyev, V. A. Romanovsky, and B. S. Tsirlin, *Combinational adder*, USSR author's certificate SU1596321, 1988.

[31] Y. Zhou, "Automatic synthesis and optimisation of asynchronous data paths using partial acknowledgement", Ph.D. dissertation, University of Newcastle upon Tyne, 2008.

[32] S. C. Smith, "Design of an FPGA logic element for implementing asynchronous NULL convention logic circuits", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 6, pp. 672–683, 2007.

[33] M.-C. Chang, P.-H. Yang, and Z.-G. Pan, "Register-less NULL convention logic", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 3, pp. 314–318, 2016.

[34] M. Kim, "Null convention logic circuits for asynchronous computer architecture", Ph.D. dissertation, RMIT University, 2019, 197 pp.

[35] B. S. Tsirlin, "An algebra of asynchronous logic networks", *Cybernetics*, vol. 20, no. 1, pp. 23–29, 1984.

[36] A. I. Bukhshtab, V. I. Varshavsky, V. B. Marakhovsky, V. A. Peschansky, L. Y. Rosenblum, N. A. Starodubtsev, and B. S. Tsirlin, *Universal logic module*, USSR author's certificate SU561182, 1977.

[37] Y. Li, "Redressing timing issues for speed-independent circuits in deep sub-micron age", Ph.D. dissertation, University of Newcastle upon Tyne, 2012, 153 pp.