Controller Implementation by Communicating Asynchronous Sequential Circuits Generated from a Petri Net Specification of Required Behavior

J. Beister and R. Wollowski

Department of Electrical Engineering,
University of Kaiserslautern, Germany

Abstract

This paper presents a completely systematic design procedure for asynchronous controllers. The initial step is the construction of a signal transition graph (STG, an interpreted Petri net) of the dialog between data path and controller: a formal representation without reference to time or internal states. To implement concurrently operating control structures, and also to reduce design effort and circuit cost, this STG can be decomposed into overlapping subnets. A universal initial solution is then obtained by algorithmically constructing a primitive flow table from each component net. This step links the procedure to classical asynchronous design, in particular to its proven optimization methods, without restricting the set of solutions. In contrast to other approaches, there is no need to extend the original STG intuitively.

Keyword Codes: B.1.1; B.6.1
Keywords: Control Design Styles; Logic Design, Design Styles

1. INTRODUCTION

Asynchronous sequential circuits (ASCs) used as controllers perceive changes of decision variables without delay, can react immediately if so required, and remain at rest between controlling actions. Hence, asynchronous controllers often are faster and more power-saving than their clocked counterparts, and usually more natural because they do not force concurrent processes into a rigid timing scheme.

The present paper presents the summary of an entirely systematic procedure for designing ASCs from the Petri net representation of the required interaction across the interface to their immediate environment (see Fig.1). This design approach covers elementary sequential circuits such as flip-flops as well as communicating ASCs for controlling concurrent processes. For more details see [1].

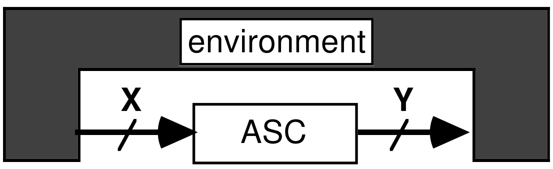


Figure 1. Interface modelled by an STG

2. FORMAL SPECIFICATION OF THE REQUIRED BEHAVIOR BY STGs

We follow [2], [3], [4], [5], [6], [7], and [8] in considering a special type of Petri net, the signal transition graph or STG ([3], [7]), to be the adequate formal representation of behavior from which asynchronous circuits should be designed. The STG represents the causal dependencies of the circuit´s output changes (observable at Y in Fig.1) upon the input changes (observable at X), and also the constraints imposed on the X- by the Y-changes. It should not contain references to time or changes of internal states, these being matters of design, and not of specification. The style of drawing chosen is that of Wendt [3], where transitions are labelled with the associated binary signal and a symbol ↑ for leading or ↓ for trailing edges. Output transitions are drawn as hollow rectangles, while full rectangles stand for input transitions. The firing of a transition represents the occurrence of the signal edge with which it is labelled.

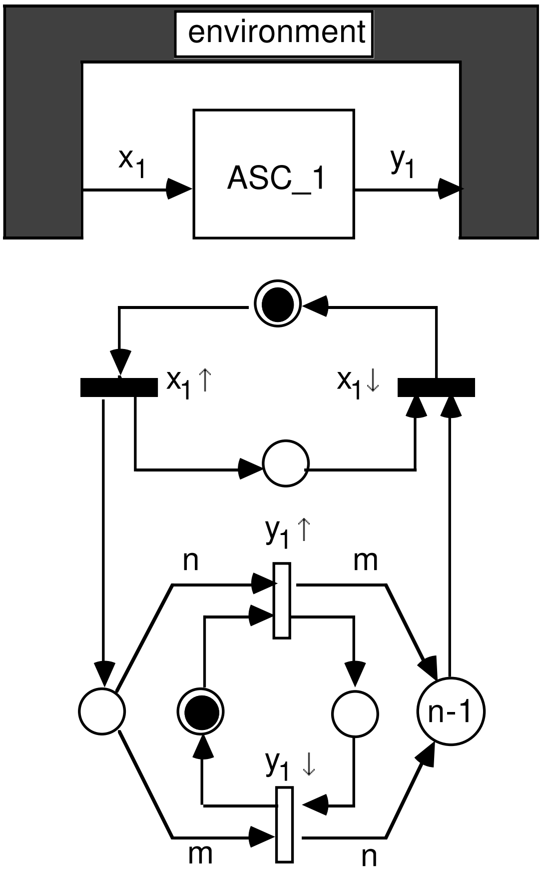


Figure 2. STG_1 (n, m ∈ N)

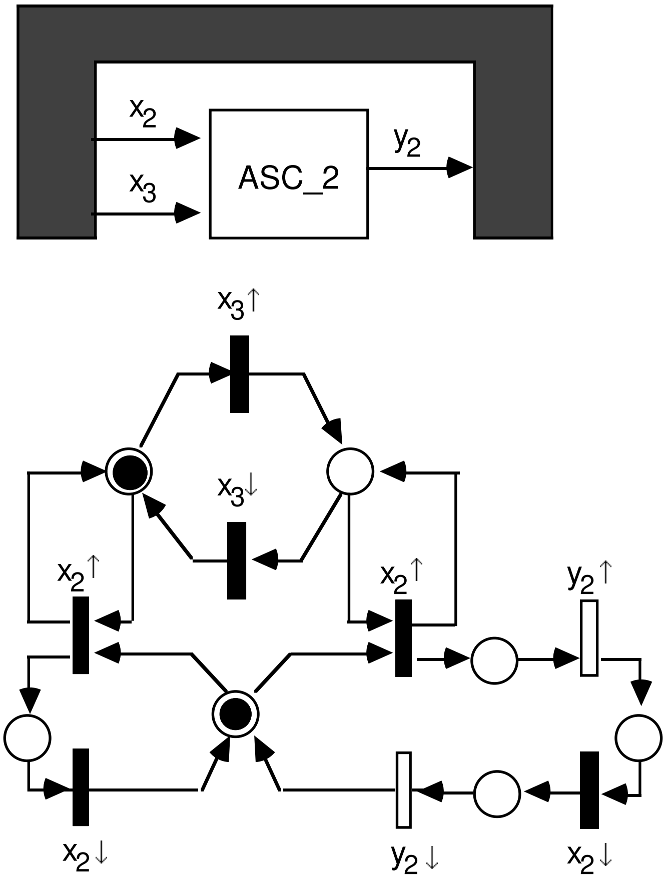


Figure 3. STG_2

Figs. 2 and 3 each show the STG of a typical elementary ASC. ASC_1 is an asynchronous counter: y_1↑ occurs after n leading edges of x_1, and y_1↓ after m further occurrences of x_1↑. ASC_2 gates clock pulses: y_2 follows x_2 whenever x_3 is 1 when x_2↑ occurs, and remains 0 otherwise. STG_1 and STG_2 represent this verbally described interface behavior formally, precisely, and adequately.

3. DESIGN FOR NON-FUNDAMENTAL MODE OPERATION

Certain design procedures, e.g. self-timed design ([9] and others), restrict themselves to ASCs where the completion of *every* change of internal state is signalized by the transition of either an original output signal or of one especially introduced for this purpose. *If* the evironment is susceptible to such signals, fundamental mode (FM) operation can easily be assured.

But among the ASC designer´s everyday problems there is the design of ASCs that do not signalize every internal state transition at the output. In this case,

(A) either measures to ensure safe non-fundamental mode (NFM) operation must be taken, or

(B) FM operation must be ensured by imposing timing conventions.

The procedure to be presented here is especially suitable for handling STGs of asynchronous circuits such as these.

Two relevant input changes in too rapid succession are at the core of all difficulties encountered in NFM operation: the state transition triggered by the first input change is “struck” by the second while still in progress, resulting in the well-known failures. Table 1 presents a classification of successive input changes and appropriate countermeasures.

Table 1: Design for NFM operation - detection of critical situations in the STG and appropriate countermeasures.

		Two successive input transitions (changes) are			
		causally dependent and stem from		causally independent ; recognition of their temporal order is	
		the same signal	different signals	not necessary	necessary
Type		1	2	3	4
Example		STG_1,2,3	no example	STG_2, 3, 4	no example
Measures	A		look-ahead	- decomposition - NFM state assignment	analog circuitry (arbiters)
	B	circuit faster than the fastest succession of input changes			

In STG_1 and 2, most of the x_1, x_2, and x_3 input changes are type 1 and must obviously be answered by state transitions, but only a few of these are made visible through output changes.

To introduce additional output signals for this purpose would be of no avail, because in general the signal generators for x_1, x_2, and x_3 could not be slowed down by them. Correct operation can only be guaranteed if timing conventions (B) are adhered to: either the environment must keep the pulse rate below a value that takes into account worst-case delays in the ASC, or the ASC designer must adapt circuit speed to a given maximum pulse rate. In cases such as the latter, which are quite common, the environment generates the input without regard for the ASC´s reactions. Then, the dotted arcs in STG_1 and 2 represent *timing* conventions instead of real influence: the STGs specify the *required* behavior.

In STG_2, the trailing edges of x_2 can be allowed to occur at any time relative to changes of x_3 (type 3), provided a suitable state assignment has been chosen for the ASC; the usual state assignments are insufficient and must be modified.

For reasons of space, types 2 and 4 will not be discussed. For measures to handle type-4 successive input changes, see [10], [4], [9]. NFM state assignment will only be shown by an example (Fig. 5g).

The presented shortened version of the design algorithm operates under the assumption that successive input changes of types 1 and 2 conform to FM operation by obeying timing conventions (easily done in the case of elementary ASCs operating in a clocked environment), and that no type-4 changes appear. Section 4 will describe the basic design step, the construction of a state machine with a particular type of next-state behavior - the primitive flow table (PFT) - from an STG. Section 5 will then discuss STG decomposition, a step that generally precedes PFT construction.

4. CONSTRUCTION OF THE PRIMITIVE FLOW TABLE FROM AN STG

A universal initial solution would be a state machine that is capable of generating the set of interface processes specified by the STG, *and only this set,* in the form of linearly quasi-ordered sets of events, accompanied by evolutions of the internal state. The primitive flow table (PFT) describes such a machine: one in which every input change triggers a state transition. Every state machine that solves the problem, when converted to standard form, will cover the PFT. Hence the set of solutions is not prematurely restricted by arbitrary state mergers or state assignments. Instead, finding the best solution is left to the classical design procedures, such as state reduction and secondary state assignment.

The first step in the construction of the PFT is the generation of the step graph SG_FM - the reachability graph extended by edges that represent the simultaneous firing of concurrently enabled transitions - from the STG (Fig. 4). Concurrent interface processes are now represented as linearly quasi-ordered events in time. SG_FM is generated assuming fundamental mode operation. I.e. if both input and output transitions are enabled in the STG, then the output transitions always have priority (FM firing rule). SG_FM is covered by the usual step graph, which is used only when measures to protect type 2 successive input changes are to be implemented.

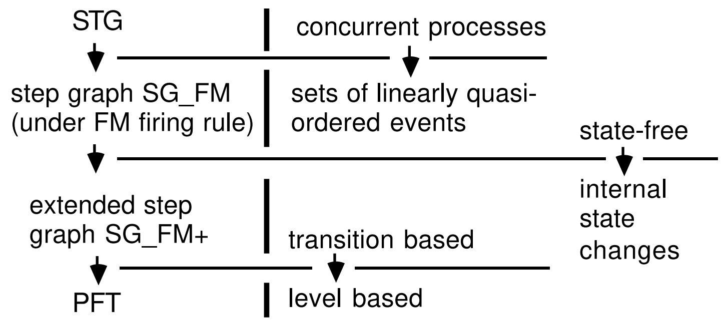


Figure 4. Construction of the primitive flow table (PFT) from an STG.

In principle, the PFT can now be constructed by putting the SG_FM through its evolutions: after each input step in the SG_FM, a change of internal state is specified in the PFT. When different input steps lead to the same marking, then the PFT returns to the same next state; otherwise, a new internal state is introduced. Steps containing more than one input transition correspond to multiple input changes. Output steps in the SG_FM are mapped into the PFT as output changes that have no further effect.

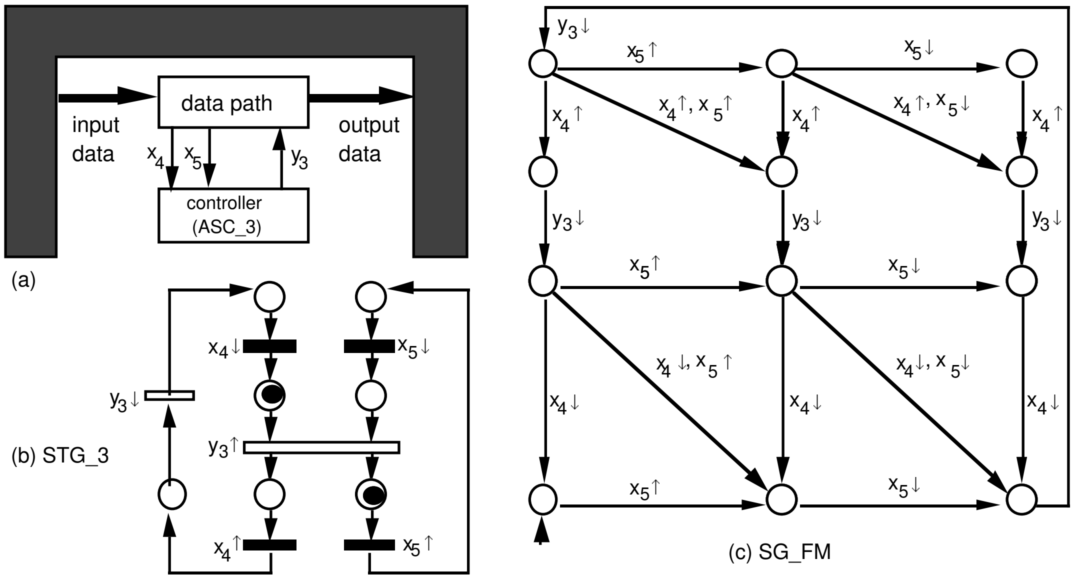
This principle can be systematically converted into an extension of the SG_FM by state transition steps and additional markings (as illustrated in Design Example 1). The result is the extended step graph SG_FM+, characterized by a 1:1 correspondence between its markings and the total states of the PFT.

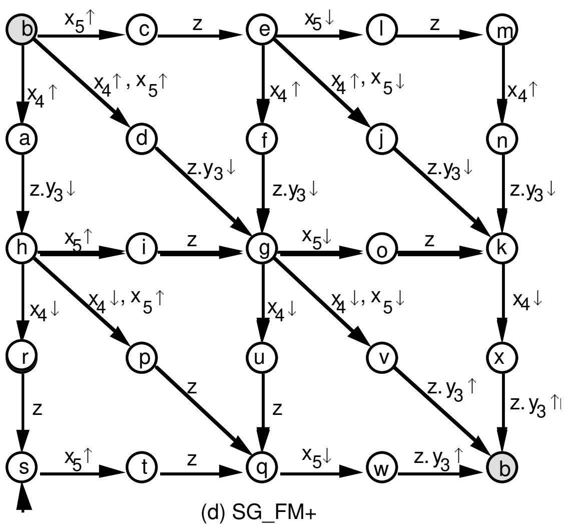
The PFT then undergoes state reduction. Although the PFT was constructed assuming FM conditions, choosing a suitable state assignment leads to an ASC that operates safely even if completely concurrent type-3 input changes occur. Classical state assignment schemes have to be modified to achieve this goal (see Example 1, Fig. 5g).

Design Example 1 (Fig. 5)

STG_3 specifies the signal transition dialog between an ASC used as a controller and a data path (Figs. 5a, b). Figs. 5c, d, and e, respectively, show the SG_FM, the SG_FM+, and the PFT. The 1:1 correspondence between the markings of SG_FM and the total states of the PFT is indicated by lower-case letters. Steps labeled “z”, “z.y↑”, or “z.y↓” in the SG_FM+ correspond to state transitions without or with accompanying output changes.
Fig. 5f shows the reduced automaton, Fig. 5g a state assignment that prevents too rapid successive input changes of type 3 (indicated by flashes) from doing any harm. The idea behind the state assignment is to have independent changes of state variables follow independent input changes. Therefore, timing conventions to guarantee safe operation of the ASC are necessary only for the sequence $x_5\uparrow$, $x_5\downarrow$ (type 1), provided a Moore-ASC with equal delays on all signal paths is implemented. In this case the ASC indicates completion, i.e. all internal changes are completed when the output changes - the environment may change dependent input signals directly after “seeing” the output change □

The design procedure presented here proceeds systematically and optimizes by classical methods. In contrast, other approaches lack systematical means of introducing and optimizing states and their transitions: The procedures proposed in [2], [4], and [5] require the STG to be made state-machine decomposable and to be extended intuitively by state transitions.The approaches of Chu [7] and Yakovlev et al. [8] leave the introduction of “internal signals” and “auxiliary variables” - state variables in the state-machine model - to the skill of the designer.





S	x_4x_5 = 00	10	11	01	y_3
1	1_s	*	*	2_t	0
2	3_w	*	*	2_q	0
3	3_b	5_a	6_d	4_c	1
4	7_l	8_j	6_f	4_e	1
5	1_r	5_h	6_i	2_p	0
6	3_v	8_o	6_g	2_u	0
7	7_m	8_n	*	*	1
8	3_x	8_k	*	*	0

(e) Primitive flow table

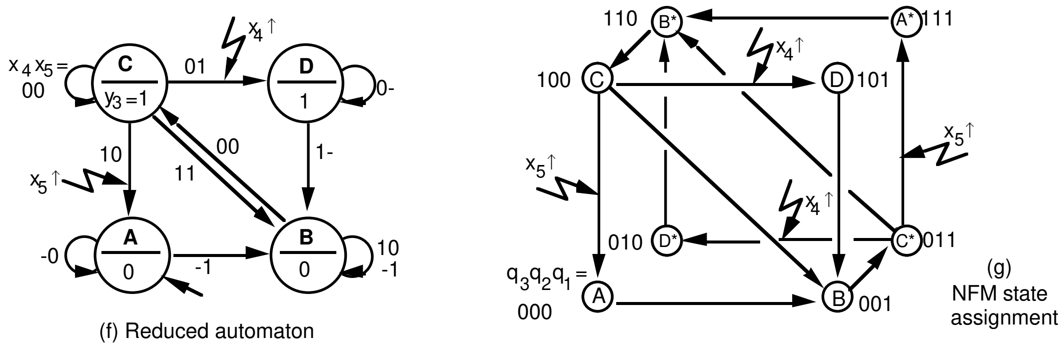


Figure 5. Design Example 1

Since the procedure proposed here generates the PFT from the STG *indirectly* by way of the step graph SG_FM, it does not depend on using Petri nets of specific types or with specific properties. For instance, the STG has to be neither state-machine decomposable nor safe. P/T systems and colored nets are equally admissible. However, timed nets of any kind are inadmissible.

5. DECOMPOSITION OF THE STG

The basic idea behind STG decomposition is to extract that part of the global signal transition dialog - in the form of a component STG - that is needed to generate correctly a subset of the output signals. In this way, the overall STG can be decomposed into overlapping component STGs. This supports the implementation of distributed, concurrently operating control structures. It also serves to cope with complexity by reducing synthesis effort and hardware cost as compared to the construction - which is always possible - of a global PFT from the overall STG.

The decomposition technique is similar to what is called net contraction in [7] and projection in [8]. Improvements and extensions are possible and even necessary in order to deal with transitions that have the same label. Decomposition is applied directly to the *original* STG, i.e. to the specification of the interface processes only. By this feature, the method differs from [4], [5], [7], and [8], where the overall STG is first enhanced intuitively by transitions of internal state variables. Just to determine whether the state transitions were correctly introduced requires a large amount of computation.

An STG is decomposed in three steps.

1. The set of output variables is suitably partitioned. No block of the partition should contain a signal of which the changes are at least locally concurrent to those of another signal in the same block. Total decomposition, where every block contains only a single variable, always fulfils this condition. Each block Bi contains the output variables of a future ASCi, and a component net, STGi, will be extracted for Bi from the overall STG.

2. For each block Bi, one must determine those global input and output signals that are necessary for correctly generating the output variables contained in Bi. This
set K_Bi is the set of input variables of PFTi and ASCi. Primarily, K_Bi contains the signals that have transitions which can enable or disable at least one transition belonging to a signal in Bi. Therefore the transitions to be inspected are the pre and post transitions of the pre places of all transitions labeled with variables from Bi.

There are two cases in which the determination of K_Bi is not yet complete.

a) Some signals may have been selected only because of a redundant place. They are redundant and can easily be found and eliminated in the component step graph. Not removing them will not falsify the decomposition.

b) Decomposition may transform (legal) conflicts in the STG into illegal conflicts between transitions labeled with the same variable in a component STGi. In this case, signals that resolve the conflict must be added to K_Bi. They, too, can be found by local inspection.

3. Each STGi is now extracted from the overall STG by removing every transition of every signal not contained in K_Bi in such a way that the causal relationship between the remaining transitions is preserved. The validity of these transformations have been proved by means of the step graph [1]. Only examples can be given here. Fig. 6a shows the usual way of removing a transition t_x which has no side places and the pre and post arcs of which are all weighted with one.

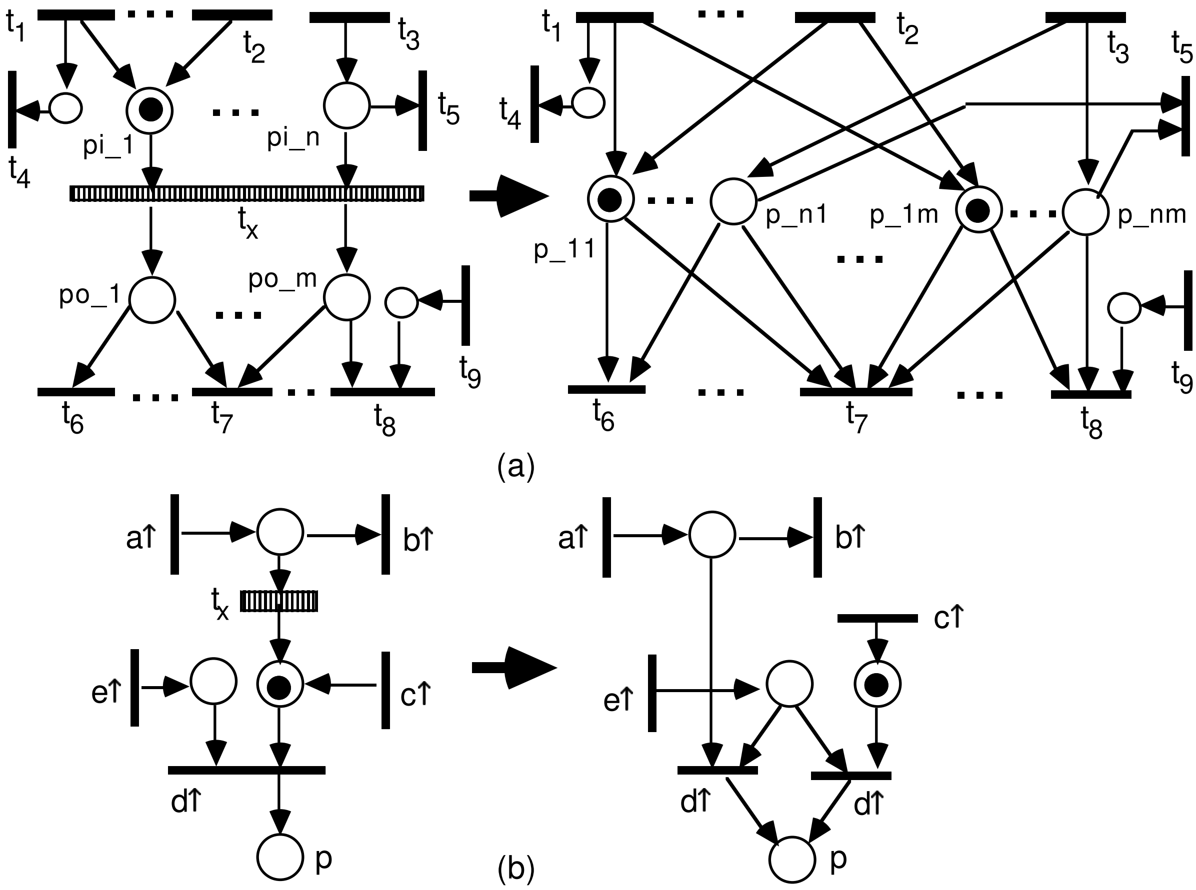


Figure 6. Removal of transitions t_x

Only its immediate environment must be considered: The pre places pi_1, ..., pi_n of t_x its post places po_1, ..., po_m are replaced by new places p_11, ..., p_n1, ..., p_rs, ..., p_nm, where p_rs replaces a pair consisting of a pre place pi_r and a post place po_s; furthermore, ∑p_rs = ∑pi_r and p_rs∑ = po_s∑.Special consideration must be given to cases such as the one shown in Fig. 6b. In order to preserve the causal structure when removing t_x, transition d↑ must be doubled □

Thus, in its essential features, decomposition is guided by net topology only. The laborious generation of the overall reachability or step graph, for instance, is quite unnecessary.

6. SUMMARY OF THE DESIGN PROCEDURE

Once the system has been partitioned into data path and controller, we have an informal specification of the control task on the data processing level (Fig. 7).

Step A. Development of a formal specification by means of an STG.

Step B. Analysis of the STG and testing for formal properties (e.g. liveness).

Step C. Decomposition of the STG into component nets STGi.

Step D. Construction of a primitive flow table PFTi for each STGi.

Step E. Synthesis of an ASCi from each PFTi by classical methods and NFM state assignment procedures; output transitions have to indicate completion □

The resulting controller is an assemblage of communicating ASCs that operate concurrently and synchronize with one another by their global output signals only - as prescribed by the overall STG.

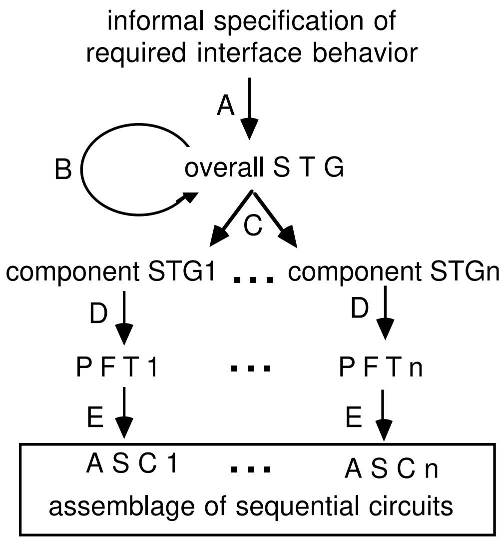


Figure 7. Design procedure

7. DESIGN EXAMPLE 2: FIFO CONTROLLER

The controller of a first-in-first-out memory (FIFO) according to Fig. 8a is to be designed. Incoming data are to be distributed alternately to the two D register pipelines VD1, VD2, VD3 and RD1, RD2, RD3 under control of the signals Ri (i=1, 2, 3), and to be transported to the output as quickly as possible. The multiplexer MUX serves to reestablish the original order of the data sequence. The VDi are triggered by the leading edges of the Ri, the RDi by the trailing edges.

Fig. 8b shows an overall STG capable of generating the signal transition dialog (between the controller and its environment) as necessary for correct control. The meanings associated with the signals are summarized in the following table.

Request signals:

RIN ↑ , ↓	data input request
ROUT ↑ , ↓	data output request
Ri ↑ (↓)	load VDi (RDi)
RY ↑ (↓)	select MUX input 1 (0).

Completion signals:

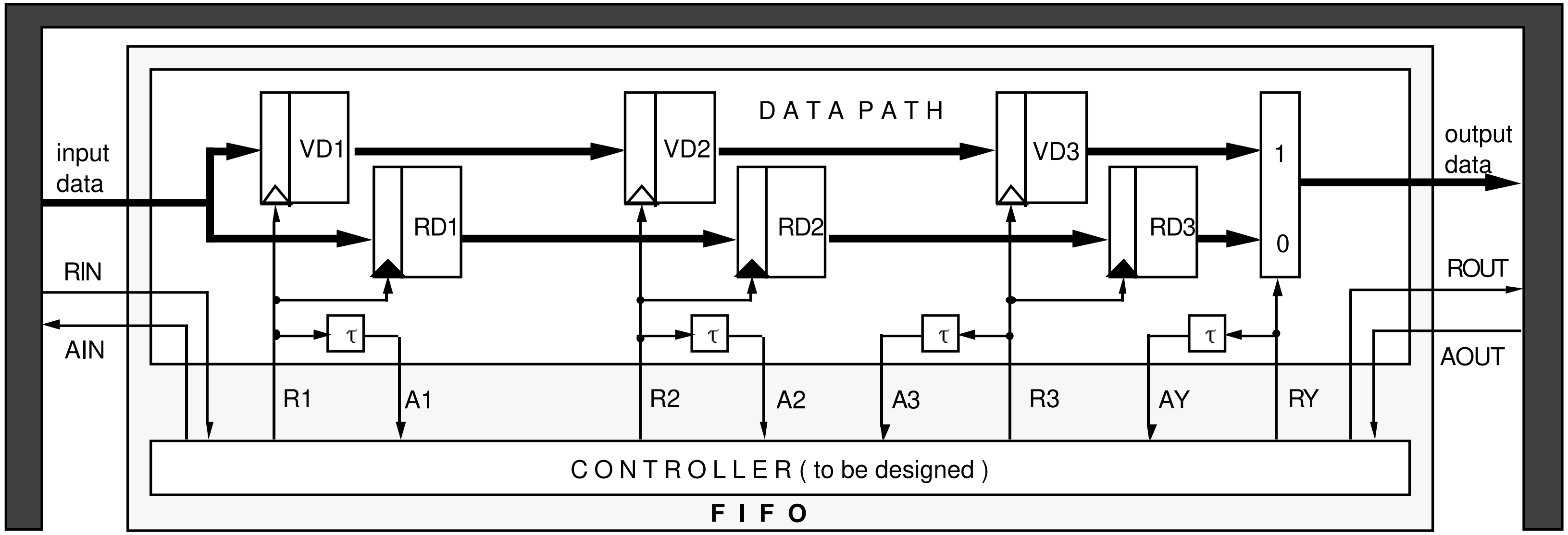
AIN ↑ , ↓	input data stored
AOUT ↑ , ↓	output data received
Ai ↑ (↓)	VDi (RDi) loaded
AY ↑ (↓)	MUX input switchover completed.

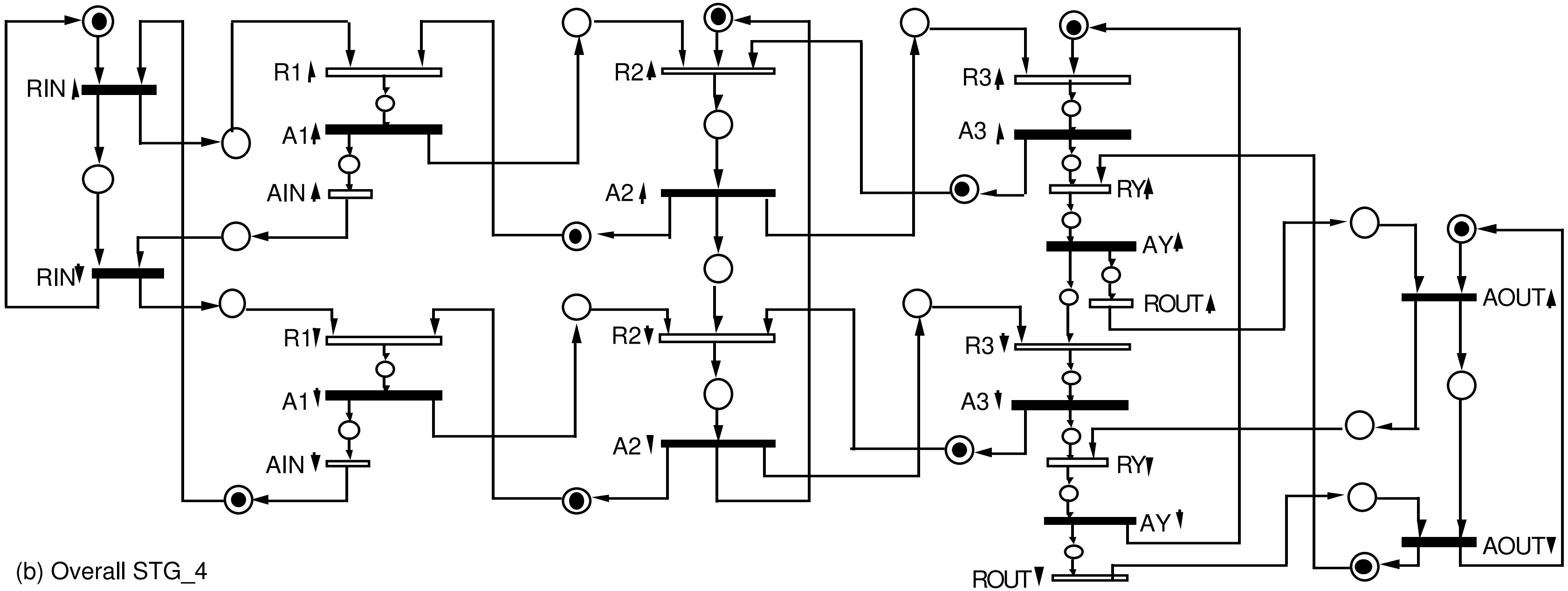
To simplify the drawing (but not in actual implementation), the completion signals A1, A2, A3, and AY are shown to be generated by delay elements (delay t).

The component STG for R1 (Fig. 8c) is obtained by decomposition of the overall STG (Fig. 8b). Fig. 8d shows the R1 circuit embedded in its environment. Only RIN and A2 are needed for generating R1: every R1 transition obtains concession to fire exclusively from RIN and A2 transitions.

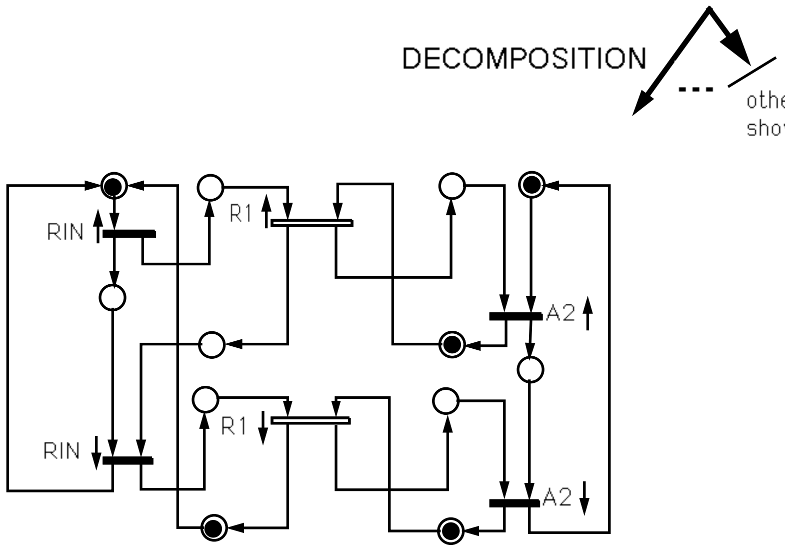
If the overall STG is decomposed totally, then the assemblage of controllers shown in Fig. 8f is the result.

(a)

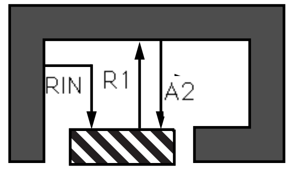




(b) Overall STG_4



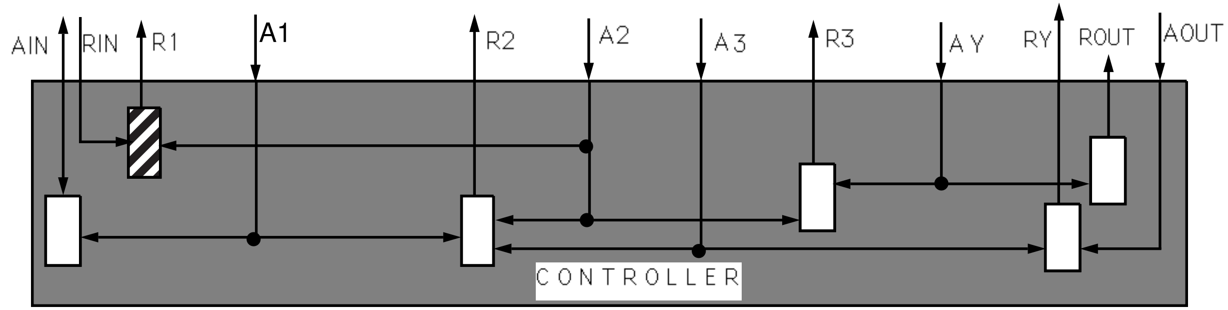
(c) Component STG for R1



(d)

state	RIN, A2: 00	10	11	01	R1
1	1	2	*	*	0
2	3	2	4	5	1
3	3	6	7	5	0
4	*	*	4	5	1
5	1	2	7	5	0
6	*	6	7	*	0
7	3	2	7	8	1
8	3	*	*	8	1

(e) PFT for the R1-STG



(f)

Figure 8. Design Example 2

Fig. 8e shows the Moore-type primitive flow table of the R1 circuit, algorithmically generated from the R1-STG (by our own design tool). The minimal-row flow table has four states. Only type-3 successive input changes occur, and can be handled by a modified state assignment procedure.

By varying the number of pipelines and their length, the capacity and performance of the FIFO can be adapted to actual requirements. The behavior of such a configurable FIFO can be described clearly and compactly by a colored Petri net. This net can also serve as the basis for the above design procedure.

8. CONCLUSION

We have briefly discussed a systematic procedure for designing an assemblage of communicating asynchronous circuits (ACSs). Such assemblages offer themselves for controlling complex, fast concurrent processes. Synthesis proceeds from a signal transition graph (STG) that solely specifies the required dialog between data path and controller, i.e. their communication across the interface, from a causal point of view, and without reference to time and internal states.

The two decisive steps from a state-free behavioral specification to an assemblage of finite-state machines are the decomposition of the STG and the introduction of internal states and their transitions. Our procedure solves the latter problem by constructing primitive flow tables directly from the component STGs. It thereby obtains access to the full range of classical design procedures that make it possible to optimize next-state behavior. Other approaches lack systematical means of introducing and optimizing states and their transitions.

9. REFERENCES

1 R. Wollowski, Systematischer Entwurf asynchroner Schaltwerksverbände aus der Petrinetz-Darstellung des Sollverhaltens, Dissertation in preparation, Universität Kaiserslautern, Fachbereich Elektrotechnik.

2 G.Ullrich, Der Entwurf von Steuerstrukturen für parallele Abläufe mit Hilfe von Petri-Netzen, Dissertation, Universität Hamburg, Institut für Informatik, 1976.

3 S. Wendt, Using Petri Nets in the Design Process for Interacting Asynchronous Sequential Circuits, Proceedings of the IFAC-Symposium on Discrete Systems Vol. 2 (1977), Dresden, 130.

4 W. Pohl, Petrinetz-Modelle der Dynamik diskreter, technischer Systeme, Dissertation, Universität Kaiserslautern, Fachbereich Elektrotechnik, 1980.

5 K. Steinbuch and W. Rupprecht, Nachrichtentechnik, Band III: Nachrichtenverarbeitung by S. Wendt, Springer, Berlin, 1982.

6 C.E. Molnar, T.-P. Fan and F.U. Rosenberger, Synthesis of Delay-Insensitive Modules, Proceedings of the 1985 Chapel Hill Conference on VLSI, Computer Science Press, Chapel Hill, 1985, 87.

7 T.A. Chu, Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications, Proceedings of the International Conference on Computer Design, 1987, 220.

8 A. Yakovlev and A. Petrov, Petri Nets and Parallel Bus Controller Design, Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris, 1990, 244.

9 V.I. Varshavski et al., Self-Timed Control of Concurrent Processes, Kluwer Academic Publishers, London, 1990.

10 S.H. Unger, Asynchronous Sequential Switching Circuits with Unrestricted Input Changes, IEEE Transacions on Computers Vol.C-20 (1971), No.12, 1437.