Asynchronous control network optimization using fast minimum cycle time analysis

Law, Chong Fatt; Gwee, Bah Hwee; Chang, Joseph Sylvester

2008

Law, C. F., Gwee, B. H., & Chang, J. S. (2008). Asynchronous control network optimization using fast minimum cycle time analysis. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems. 27(6), 985-998.

https://hdl.handle.net/10356/92668

https://doi.org/10.1109/TCAD.2008.923238

Asynchronous Control Network Optimization Using Fast Minimum-Cycle-Time Analysis

Chong-Fatt Law, Bah-Hwee Gwee, *Senior Member, IEEE*, and Joseph S. Chang, *Member, IEEE*

***Abstract*—This paper proposes two methods for optimizing the control networks of asynchronous pipelines. The first uses a branch-and-bound algorithm to search for the optimum mix of the handshake components of different degrees of concurrence that provides the best throughput while minimizing asynchronous control overheads. The second method is a clustering technique that iteratively fuses two handshake components that share input channel sources or output channel destinations into a single component while preserving the behavior and satisfying the performance constraint of the asynchronous pipeline. We also propose a fast algorithm for iterative minimum-cycle-time analysis. The novelty of the proposed algorithm is that it takes advantage of the fact that only small modifications are made to the control network during each optimization iteration. When applied to nontrivial designs, the proposed optimization methods provided significant reductions in transistor count and energy dissipation in the designs' asynchronous control networks while satisfying the throughput constraints.**

***Index Terms*—Asynchronous pipelines, control networks, handshake components, optimization.**

I. Introduction

ASYNCHRONOUS circuits dissipate dynamic power only when and where there are computations and, therefore, are very power efficient. The asynchronous pipeline [1] is a popular design method for asynchronous circuits due to its architectural resemblance to the prevalent synchronous design method. Most asynchronous pipelines are based on the distributed control architecture, i.e., the data paths are controlled by a network of small asynchronous controllers, which are called handshake components, that communicate with each other by using handshake signals. Such a network is called an asynchronous control network.

For data-path-dominated pipelines (such as finite-impulse-response filters), the asynchronous control networks are relatively small. However, there is strong evidence that the asynchronous control networks are relatively large for pipelines that are control dominated or synthesized from high-level programming languages [2]–[6]. For example, the control area overhead of an asynchronous Reed–Solomon error detector is 35% [6]. Furthermore, handshake signals in asynchronous control networks have a signal switching factor of at least one, whereas that in single-rail data paths is only 0.5. Thus, in asynchronous pipelines, the control power overhead is likely to be higher than the control area overhead. For example, the control network of an asynchronous pipelined processor occupies 18% of the total circuit area but dissipates 35% of the total power [3].

Such large asynchronous control overheads are undesirable because they might offset the advantages that asynchronous circuits enjoy over synchronous ones, particularly the advantage of better power efficiency. This paper proposes the optimization techniques that reduce the circuit area and power dissipation overheads of control networks in asynchronous pipelines.

One way to reduce the control network overheads is to keep the handshake components small [7], [8]. However, the circuits reported in [7] have two-sided timing assumptions that affect their robustness. To date, the smallest reported handshake component (a basic latch controller) consists of only a single XNOR gate [8].

Small and robust handshake components can also be designed by imposing tight couplings between the input and output channel events in their handshake-protocol specifications. Fig. 1(a) shows the specification of a maximally coupled (MAXC) latch controller that operates its latches in the normally closed mode (note that this latch controller is similar to that reported in [9]; the latter, however, operates in the normally opened mode). The specification is a signal transition graph (STG) [10], where a signal transition is enabled (to occur) when each of its input arcs is marked with a token (depicted as a black dot). When a signal transition occurs, the tokens on its input arcs are removed, and a token is placed on each of its output arcs. For example, in Fig. 1(a), ri+ is enabled, and its occurrence removes the token on $(\mathrm{ai}-, \mathrm{ri}+)$ and places a token on $(\mathrm{ri}+, \mathrm{en}+)$, which causes en+ to be enabled. Observe, in Fig. 1(a), the strong couplings between the events, particularly the path ri$-\rightarrow$ en$-\rightarrow$ ro$-\rightarrow$ ai$-$. Such strong couplings are not necessary for correctness but simplify the circuit, which comprises only one C-gate (AND element for events; see Fig. 2). However, using only MAXC handshake components leads to poor pipeline throughput [9]. For better pipeline throughput, one can use the handshake components of higher concurrence, such as the semicoupled and minimally coupled handshake components (the latter is similar to the fully decoupled (FULLYD) long hold circuit [9]), as shown in Fig. 1(b) and (c), respectively. These circuits are, however, larger and dissipate more power.

In this paper (in Section IV-A), we propose a technique, which is called optimal decoupling, that resolves the dilemma between small handshake components and good throughput performance. Given a control network, the technique uses a

Manuscript received March 6, 2007; revised August 22, 2007. This work was supported by the Singapore Millennium Foundation. This paper was recommended by Associate Editor S. Nowick.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: lawc0001@ntu.edu.sg).

Digital Object Identifier 10.1109/TCAD.2008.923238

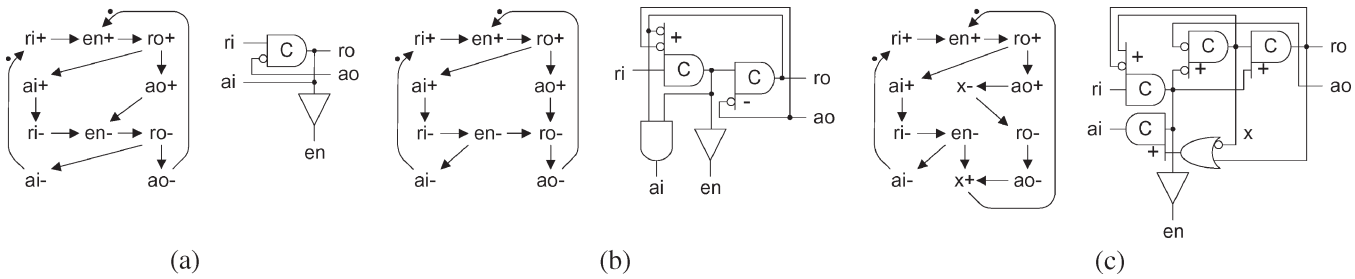


Fig. 1. Basic latch controller specifications and circuit implementations. (a) MAXC. (b) Semicoupled. (c) Minimally coupled. These specifications also serve as templates for the specifications of other functionally more complex handshake circuits.

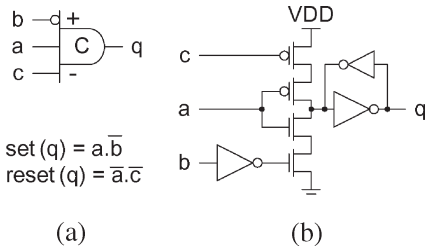


Fig. 2. (a) Symbol and (b) CMOS implementation of a C-gate.

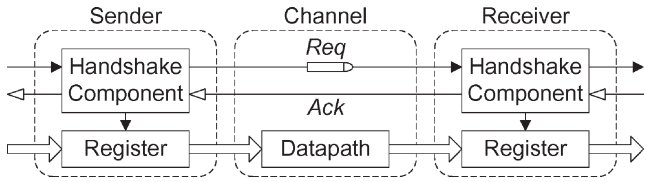


Fig. 3. Asynchronous pipeline with delay matching.

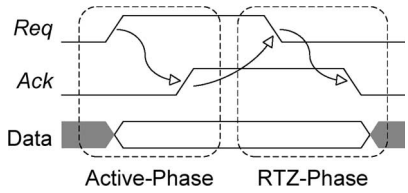


Fig. 4. Example of a handshake protocol.

branch-and-bound algorithm to search for the optimum mix of the handshake components of different degrees of concurrence that provides the highest pipeline throughput achievable by the given pipelining structure and computation delays. The idea is to use as many MAXC handshake components as possible without sacrificing throughput by selecting the handshake components of higher concurrence where necessary.

Another approach to optimize asynchronous control networks is clustering [3], [4]. This involves identifying and replacing, which are based on some rules, a group of interconnected handshake components with a single handshake component while preserving the behavior of the control network. In this paper (in Section IV-B), we propose a clustering-based technique, which is termed the handshake-component fusion, which iteratively fuses two handshake components of the same type that share input channel sources or output channel destinations into a single component. The component fusion is subject to the conditions that the restructured pipeline is flow equivalent [11] to the original one and has a minimum cycle time that satisfies the given timing constraint.

An important component of the proposed optimization techniques is a minimum-cycle-time analysis, i.e., the computation of the minimum cycle time, which is denoted as $\tau_{\min}$, of the pipeline (note that the reciprocal of $\tau_{\min}$ can be taken as the pipeline's throughput). The execution time of minimum-cycle-time analysis is critical to the feasibility of the proposed techniques because it is performed after each incremental improvement on the design. In this paper (in Section V), we propose a fast algorithm for the iterative minimum-cycle-time analysis that exploits the fact that only small changes are made to the control network at each optimization iteration.

This paper is organized as follows. Section II introduces some basic asynchronous pipeline concepts and graphical modeling tools for asynchronous circuits. Section III reviews the previous works and explains how this paper is different. In Section IV, we propose two optimization techniques for asynchronous control networks. In Section V, we propose an algorithm for fast iterative minimum-cycle-time analysis. Section VI describes the application of the optimization techniques to two design examples. We present our conclusions in Section VII.

II. Background

A. Asynchronous Pipelines

In an asynchronous pipeline, data registers are governed by asynchronous controllers, which are called handshake components (see Fig. 3). The data transfers between data registers are carried out in channels, which consist of handshake signals (request Req and acknowledge Ack) between the handshake components and data paths between the registers. The delay-matching technique is commonly used to ensure that Req does not arrive at the receiver before the data. It involves placing a delay element on the request wire whose latency matches the worst case delay of the corresponding data path.

A handshake protocol refers to the procedure of communication between handshake components. An example of a handshake protocol is shown in Fig. 4. In the active phase, Req is asserted by the sender to indicate to the receiver that new data are ready to be consumed. If the data being held by the receiver have already been consumed by their successor, it enables its data registers and asserts Ack. In the return-to-zero (RTZ) phase, the sender first deasserts Req. This is followed by the receiver disabling its registers and deasserting Ack.

Two channel-delay schemes are possible for the handshake protocol described earlier. The symmetric channel-delay

scheme uses inverter chains as request-wire delay elements and thus imposes the same request delay for the active and RTZ phases. The asymmetric channel-delay scheme exploits the redundancy of the RTZ phase by reducing the RTZ-phase delay to the latency of a single gate. In this scheme, AND-gate chains are used as request-wire delay elements.

B. Petri Nets and STGs

A Petri net [12] (PN) N is a 5-tuple $N = (P, T, F, W, M_0)$, where $P = \{p_1, p_2, \ldots, p_m\}$ is a set of m places, $T = \{t_1, t_2, \ldots, t_n\}$ is a set of n transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, $W : F \rightarrow \{1, 2, 3, \ldots\}$ is a weight function, and $M_0 : P \rightarrow \{0, 1, 2, \ldots\}$ is the initial marking.

For transition t_j (place p_i), its input places (transitions) are denoted as $\bullet t_j (\bullet p_i)$ and its output places (transitions) $t_j \bullet (p_i \bullet)$.

The incidence matrix $A = [a_{ij}]$ of a PN is an $m \times n$ integer matrix such that $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$, where a_{ij}^{+} is the weight of the arc from transition t_j to place p_i and a_{ij}^{-} is the weight of the arc from p_i to transition t_j.

An S-invariant, which is denoted as y, is an m-vector of integers such that $A^{\mathrm{T}} y = 0$. Intuitively, an S-invariant (or, more accurately, its positive elements) corresponds to a set of places in the net whose weighted sum of tokens is constant for all possible transition firing sequences (note that a place can hold any nonnegative integer number of tokens). Such a set of places is termed as the support of an S-invariant and denoted by $\|y\|$. A support is minimal if no proper nonempty subset of the support is also a support (note that algorithms for computing minimal-support S-invariants are well reported [13]).

A T-invariant, which is denoted as x, is an n-vector of integers such that $Ax = 0$. Intuitively, a T-invariant indicates the number of times each transition in the net needs to fire in a transition firing sequence that leads back to the initial state of the net. The set of transitions corresponding to the positive elements of a T-invariant is termed as the support of a T-invariant and denoted by $\|x\|$.

An STG is an interpreted and restricted form of PNs, where each transition is labeled as a signal transition, the capacity of all places is restricted to one, and the arc weights are all ones.

A timed PN is the one whose transitions or places are associated with a delay. In a periodically operated timed PN, the minimum cycle time $\tau_{\min}$ is the time to complete a firing sequence in the net, leading back to the starting marking after firing each transition at least once. If the delays are associated with the places and can be predetermined, then $\tau_{\min}$ is given by [12]

$$\tau_{\min} = \max_{k} \left\{ y_k^{\mathrm{T}} D(A^{+}) x / y_k^{\mathrm{T}} M_0 \right\} \tag{1}$$

where the maximum has taken over all minimal-support S-invariants $y_k \geq 0$, $x > 0$ is a T-invariant, D is the diagonal matrix of place delay d_i, $i = 1, 2, \ldots, m$, and $A^{+} = [a_{ij}^{+}]$.

III. Previous Works

The proposed optimal decoupling technique is different from other handshake-protocol decoupling techniques [9], [11], [14] in two ways. First, the reported techniques only consider the control networks comprising the handshake components of the same degree of concurrence. In contrast, the proposed technique mixes the handshake components of different degree of concurrence in one control network. Second, the reported techniques only target throughput improvement. Our technique, in contrast, tries to minimize the circuit area and power dissipation penalties that are incurred when it is optimizing the throughput.

The proposed handshake-component fusion technique is different from other clustering methods [3], [4] in that it incorporates a minimum-cycle-time analysis, which is beneficial for two reasons.

First, the selection of optimization targets (i.e., the components to be fused) is not restricted by the need to preserve performance because minimum-cycle-time analysis is used to predict the effect of the fusion on performance. If the fusion of the optimization targets is predicted to violate the pipeline throughput constraint, then the fusion is abandoned. Thus, a greater degree of freedom in optimization target selection is allowed, which potentially leads to a better quality of result.

Second, a tradeoff is possible between the pipeline's throughput, power dissipation, and circuit area during the optimization process. When the optimization targets are fused, the asynchronous control network is modified, which might be detrimental to the throughput of the design. This decrease in performance, however, does not pose a problem, as long as the specified throughput constraint is not violated. Thus, by performing a minimum-cycle-time analysis at each optimization iteration, the proposed handshake-component fusion method (FusionE) is able to reduce the power dissipation and circuit area of the asynchronous control network while allowing (within the specified constraint) the throughput of the design to decrease.

Other optimization techniques for asynchronous pipelines include minimal pipelining for a given timing constraint [16] and slack matching [17]–[19]. These methods, however, do not specifically address the problem of asynchronous control overheads. Furthermore, the optimization algorithms in [16]–[18] are based on marked graphs [12], which are subclasses of PNs that do not model choices. This implies that these algorithms support only deterministic pipelines which do not make real-time decisions that govern the flow of data. In contrast, the optimization techniques proposed in this paper are based on STGs, which do support the modeling of choices. Thus, the proposed techniques can support nondeterministic pipelines.

IV. Optimization

This section describes in detail the proposed optimization methods of optimal decoupling and handshake-component fusion.

A. Optimal Decoupling

In this section, we propose a technique that searches for the optimum mix of the handshake components of different degrees of concurrence in a given asynchronous control network with the objective of providing the highest pipeline throughput that is achievable by the given pipelining structure and computation

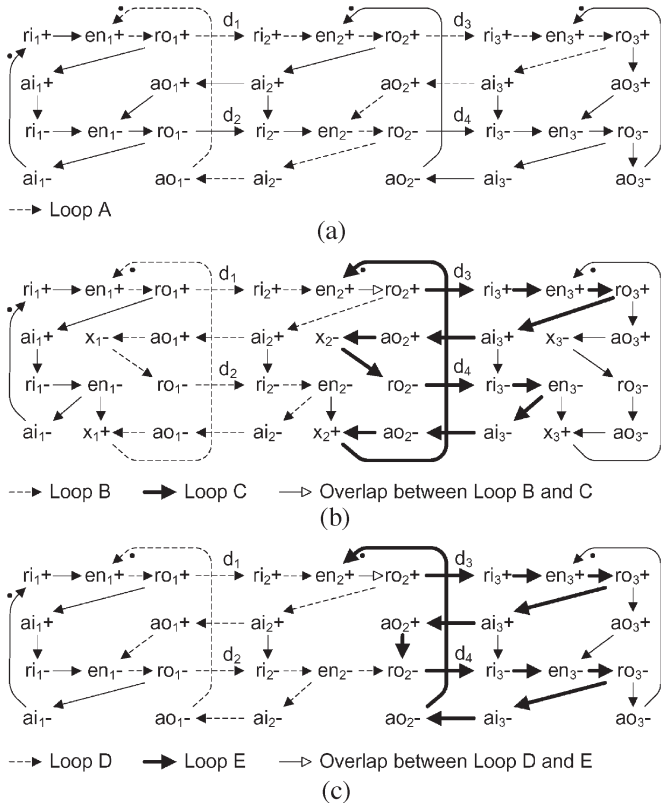


Fig. 5. STGs of three-stage linear pipelines. (a) MAXC. (b) Minimally coupled. (c) Optimally decoupled.

blocks while incurring minimal handshake-component area and power dissipation. The technique is summarized as follows.

First, a PN model of the given asynchronous control network is composed by abutting the STGs of the handshake components according to the network's structure (note that in the context of this paper, a PN model is simply a composition of STG models). Initially, only MAXC handshake protocols are used.

Second, each place in the model is associated with a time delay. The places representing channels are assigned either the active-phase channel delay (i.e., the critical data-path delay across a channel) or the RTZ-phase channel delay; the others are assigned as internal handshake-component delays (note that associating delays with places of PNs is a general practice that is common in the literature). These delays are obtained through the static timing analyses of the handshake components and data paths. Note that the effect of latch-enable signal buffering on minimum cycle time is taken into consideration by including the delays of latch-enable signals in channel delays.

Third, guided by system-level minimum-cycle-time analyses, a branch-and-bound algorithm is used to selectively decouple the handshake protocols so as to improve the pipeline's throughput. The increase in the internal delays of decoupled handshake components is taken into consideration, because minimum-cycle-time analysis is performed every time a handshake component is decoupled.

1) Demonstration: Fig. 5(a) shows the PN model of a three-stage linear pipeline with asymmetric channel delays based on MAXC latch controllers. To compute the minimum cycle time τ_1 of the pipeline, we first extract all minimal-support S-invariants of the model, using an existing algorithm [13]. Note that, in the current context, the minimal-support S-invariants are simply directed loops. For example, the dotted arcs in Fig. 5(a) correspond to a directed loop, which is labeled as A. Next, we compute the "cycle time" of each directed loop by using (1). It can be shown that A has the longest "cycle time" among all directed loops in Fig. 5(a), i.e., $\tau_1 = \tau_A = d_1 + d_3 + d_A$, where d_1 and d_3 are active-phase channel delays and d_A is the total internal circuit delay associated with A.

To improve the pipeline's throughput, one can replace all the MAXC latch controllers with the minimally coupled ones [see Fig. 5(b)]. Now, the minimum cycle time is $\tau_2 = \max\{\tau_B, \tau_C\}$, where $\tau_B = d_1 + d_2 + d_B$ and $\tau_C = d_3 + d_4 + d_C$ are the cycle times of the directed loops B and C, respectively, and d_2 and d_4 are the RTZ-phase channel delays. Because d_2 and d_4 are less than d_1 and d_3 (due to channel-delay asymmetry), we know that $\tau_2 < \tau_1$.

Alternatively, we can eliminate the directed loop A in Fig. 5(a) to improve the minimum cycle time by the following: 1) replacing the arc $\mathrm{ro}_2- \rightarrow \mathrm{ai}_2-$ with $\mathrm{en}_2- \rightarrow \mathrm{ai}_2-$ to allow the new input active phase of stage 2 to start before the end of its current output active phase and 2) replacing the arc $\mathrm{ao}_2+ \rightarrow \mathrm{en}_2-$ with $\mathrm{ao}_2+ \rightarrow \mathrm{ro}_2-$ to allow the output handshake cycle of stage 2 to complete. The result [see Fig. 5(c)] is a semicoupled latch controller at stage 2, whereas the latch controllers at stages 1 and 3 remain MAXC. Thus, we now have a pipeline whose minimum cycle time is $\tau_3 = \max\{\tau_D, \tau_E\}$, where $\tau_D = d_1 + d_2 + d_D$ and $\tau_E = d_3 + d_4 + d_E$ (corresponding to the directed loops D and E, respectively). Because the MAXC and semicoupled circuits are less complex than the minimally coupled ones, d_D and d_E are likely to be less than d_B and d_C, and therefore, we expect τ_3 to be less than both τ_1 and τ_2. Thus, we have achieved a maximum pipeline throughput by decoupling only the stage 2 of the pipeline.

2) Algorithm: We now describe the algorithm that implements optimal decoupling. The algorithm is based on the branch-and-bound approach [20].

At each node in the branch-and-bound tree, each handshake component in the control network is in one of four states—unconstrained, MAXC, semicoupled, or minimally coupled. An unconstrained handshake component is identical to a MAXC one except that it can be selected as a branching variable during a branching operation, whereas the latter cannot. The other three states are constrained states representing different degrees of concurrence for a handshake component. Collectively, the states of all handshake components at a node form a configuration, which is denoted as c and represents a potential solution to the optimization problem. At the root node, all handshake components are unconstrained.

During a branching operation, an unconstrained handshake component is selected from the configuration of the parent node as the branching variable, and a constrained state is assigned to it. The constrained state assignment to the branching variable is called the branching constraint. The configuration of the parent node is altered by the branching constraint and passed to the child node. Because there are three constrained states, there are three possible assignments to the branching

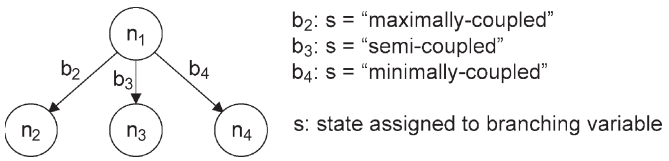


Fig. 6. Branching operation in the branch-and-bound algorithm for the proposed optimal decoupling technique.

variable, leading to three child nodes (see Fig. 6). For the first branch, the branching variable is assigned to be MAXC ($s =$ maximally coupled, where s denotes the constrained state that is assigned to the branching variable), and therefore, there is no modification to the configuration, i.e., the nodes n_1 and n_2 in Fig. 6 have the same configuration. For the second and third branch (leading to the nodes n_3 and n_4 in Fig. 6, respectively), the branching variable is assigned to be semicoupled ($s =$ semicoupled) and minimally coupled ($s =$ minimally coupled), respectively. These branching constraints alter the configurations and necessitate the recomputations of the associated PN model's invariants and cycle times. Because the modification to the models are small, the algorithm that is proposed in Section V can be used to efficiently compute the new or modified S-invariants in a relatively short time without resorting to complete recomputation. The minimum cycle time of the new configuration is then compared against the overall upper bound, which is denoted as u, i.e., the best minimum cycle time among the configurations explored thus far by the algorithm. If the new minimum cycle time is less than u, then it replaces u as the upper bound.

A node is pruned if the lower bound, which is denoted as l, for the minimum cycle time of its configuration c is not less than u. To compute l, we need to predict how far the minimum cycle time of the node can be improved through further handshake-component decoupling. First, the S-invariants of the PN model of c are grouped into sets based on their respective cycle times. Second, the sets are ranked in decreasing order of cycle time, i.e., the first set has the longest cycle time, the second set has the second longest cycle time, and so forth. Third, the decoupling expression, which is denoted as E, of a set S_i (starting with the highest ranking set) is composed as follows. The support of each invariant y_k in S_i is searched to detect the presence of certain predetermined ordered sets of places. These sets of places identify which handshake components should be decoupled to eliminate y_k (and therefore potentially improve the minimum cycle time).

The handshake components identified for decoupling in y_k are represented as literals, which are joined together by using an OR operator to form a clause. The OR operator is used because only one of the handshake components needs to be decoupled in order to eliminate y_k. Because all invariants in S_i must be eliminated to improve the minimum cycle time, the clauses extracted from all invariants in S_i are joined together by using the AND operator, resulting in E. In the event that no circuit has been identified in y_k, then y_k cannot be eliminated through decoupling, and the corresponding 'clause' is set to FALSE. Furthermore, to reflect the fact that the state of a constrained handshake component cannot be changed, the constraints of the current configuration are imposed on E by setting the literals of E that represent constrained handshake components to FALSE.

At the end of the aforementioned procedure, if E is FALSE, then it implies that the invariants in S_i cannot all be eliminated through further decoupling. In this case, l is given by the cycle time of S_i. On the other hand, if E has at least one literal, then it implies that all invariants in S_i can be eliminated through further decoupling. In this case, the aforementioned procedure is repeated for the next highest ranking set.

A node in the branch-and-bound tree is terminated if the minimum cycle time of its associated configuration can no longer be improved through further decoupling, i.e., if the minimum cycle time is equal to l.

If a node is neither pruned nor terminated, then it is branched into three child nodes; the procedure for which has already been described. The branching variable for the branching operation is arbitrarily selected from the literals of the decoupling expression of the invariant set with the longest cycle time.

At the end of the search, the configuration associated with u is the optimum mix of the handshake components that offer the best throughput with minimal circuit complexity.

B. Handshake-Component Fusion

In this section, we propose a clustering technique that locally restructures the control network of an asynchronous pipeline to reduce the pipeline's control overheads. Similar to the optimal decoupling technique described in Section IV-A, a timed PN model of the control network is first composed. A proposed algorithm then iteratively selects two handshake components of the same type, which are called optimization targets, that share at least one (fork) input channel source or (join) output channel destination. An extended version of the algorithm is also proposed, which considers, as optimization targets, all handshake-component pairs of the same type (instead of just forks and joins). The optimization targets are replaced by a single component of the same type (in short, we say that the optimization targets are fused) such that the replacement component has the following: 1) all those of the optimization targets as its input channel sources; 2) all those of the optimization targets as its output channel destinations; and 3) control on all the latches of the optimization targets. For each pair of channels that are fused, the delay of the fused channel is given by the larger channel delay of the pair.

1) Demonstration: Consider the control network in Fig. 7(a), where handshake components A and B share a common channel source. Fusing A and B leads to the network shown in Fig. 7(b), where A' is the replacement component. Note that the reduction in power dissipation and circuit area is not limited to the elimination of one handshake component but also potentially applies to the delay elements. This is because the activation of the output channels of A and B is now synchronized at A', thus allowing the corresponding delay elements to be shared, i.e., $d_6 = d_4$ and $d_7 = d_3 - d_4$ (assuming that $d_3 \geq d_4$), where d_i is the latency of D_i. Furthermore, note that the number of C-gates remains unchanged and, to preserve correct data transfer, it must be ensured that $d_5 = d_1 + d_2$.

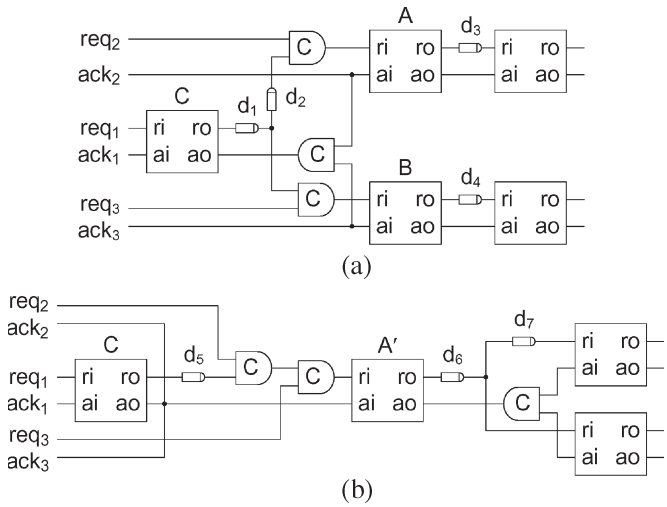


Fig. 7. Handshake-circuit fusion. (a) Original control network. (b) Control network after the fusion of circuits A and B.

2) Preconditions: The fusion of the optimization targets is subject to two conditions. First, the restructured pipeline satisfies the given minimum-cycle-time constraint. This condition is checked by the iterative minimum-cycle-time-analysis method proposed in Section V.

It was mentioned at the beginning of this section that, when the optimization targets are replaced with a single component, the latches that were originally controlled by the optimization targets are collectively controlled by the replacement component. This means that the delay of the latch-enable signal, which is denoted as en_r, generated by the replacement component is likely to be longer (due to a higher capacitive load) than those generated by the optimization targets. Consequently, the speed performance of the pipeline might be affected.

The longer delay of en_r is taken into consideration by appropriately increasing the active-phase request delays associated with the output channels of the replacement component. The procedure for doing so is as follows.

1) The optimization process computes the capacitive load of en_r by summing the respective capacitive loads of the latch-enable signals generated by the optimization targets. This is facilitated by a lookup table that is initialized at the beginning of the optimization with the capacitive load of each latch-enable signal in the original pipeline. During the optimization, when two handshake components are fused, a new entry that corresponds to en_r is created in the table to record the capacitive load of en_r.
2) The optimization process consults another lookup table to determine the delay of en_r (note that the table is prepared by the designer prior to the optimization). More specifically, each entry in the table corresponds to a particular range of capacitive load that a latch-enable signal is driving and a delay is associated with each range. The delays are obtained through the transistor-level simulations of distribution networks (typically consisting of signal buffers arranged in a treelike structure) for the latch-enable signals. In general, the size, and thus the delay, of a distribution network is proportional to the total capacitive load that it has been designed to drive (note that the design of distribution networks for signals with large capacitive loads, such as clock signals, is a well-researched topic in a very large scale integration design (see, for example, [21]).
3) For each output channel C of the replacement component, the associated active-phase request delay t_{req} is updated by using $t'_{\mathrm{req}} = t_{\mathrm{req}} + t_{\mathrm{enr}} - t_{\mathrm{eno}}$, where t'_{req} denotes the new active-phase request delay, t_{enr} denotes the delay of en_r, and t_{eno} denotes the delay of the latch-enable signal that is generated by the optimization target, which has C as its output channel, before the fusion of the optimization targets.

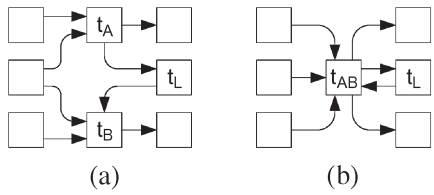


Fig. 8. (a) Directed path t_L from t_A to t_B forms a loop after the (b) fusion of t_A to t_B.

Second, there is flow equivalence [11] between the original and restructured pipelines. The original and restructured pipelines are said to be flow equivalent if the following are satisfied: 1) They have the same set of data-path latches and 2) for each latch L, the projections of the traces onto L (i.e., the sequence of values stored by L) are the same in both pipelines. The following theorem is used when determining if the flow equivalence is observed by the restructuring.

Theorem 1: The original and restructured pipelines are flow equivalent if the following are satisfied: 1) The optimization targets A and B, and the channel link from A to B and vice versa, have deterministic behavior (i.e., there are no free choices in their STG representations) and 2) at least one stage, but not all stages, of each channel link is active upon the initialization of the original pipeline.

Proof: Because the data path of the original pipeline is not modified by the optimization process, the first condition of flow equivalence, that data-path latches are preserved, is immediately satisfied. The trace projection onto a latch depends on the enabling conditions for the latch and the data that are present at the input of the latch when it is enabled. The latter is a function of the combinational logic block driving the latch, as well as the inputs that are fed into the logic block. Because combinational logic blocks are not affected by the optimization, and their inputs are values held by latches, it is sufficient, when testing the second condition of flow equivalence, to analyze the effect of the restructuring on the enabling conditions for latches.

Consider the PN model of a fork as shown in Fig. 8(a), where t_A and t_B represent the optimization targets A and B, respectively, t_L represents a channel link from A to B, the source and sink transitions represent channel sources and sinks for A and B, and each place (depicted as an arc) represents a channel (the representation of A, B, and the channel link by transitions is legitimate, as they have deterministic behavior). The capacity of the places is one, and the enabling of the transitions is governed by the strict transition rule,

i.e., a transition is enabled to fire only when its firing does not cause the number of tokens in its postplaces to exceed one. A place that is marked with a token indicates that the corresponding channel is active, i.e., a handshake cycle is in progress. Each firing of t_A or t_B represents an operation of the corresponding handshake component, which resets the corresponding input channels, activates the corresponding output channels, and enables its latches to register a new set of data. Each firing of t_L represents the entry of a token into the channel link and the exit of a token from the channel link.

The fusion of A and B is modeled by the fusion of t_A and t_B to form t_{AB}. Fig. 8(b) shows the transformed model, where t_{AB} represents the handshake component that has replaced A and B. Fig. 8(b) shows that the enabling conditions of t_{AB} are the conjunction of those of t_A and t_B. This means that the enabling conditions for A are added to those for B and vice versa, resulting in both A and B having the same enabling conditions. The effect on A and B of the additional enabling conditions depends on whether at least one of the additional enabling conditions is causally related to at least one of the original conditions (i.e., whether there exists a channel link t_L between A and B). If such a causal relation does not exist (i.e., t_L does not exist), then the additional enabling conditions will only have the effect of potentially delaying the firing of A and B and will not affect the sequence of values that are stored by the latches controlled by A and B, i.e., the original and restructured pipelines are flow equivalent. On the other hand, if such a causal relation exists (i.e., t_L exists), then a channel loop is formed by t_{AB} and t_L, and the liveness of the loop must be investigated. It is known that a channel loop is live if the number of tokens in it is greater than zero and less than the number of stages in the loop. Thus, if at least one stage, but not all stages, of the channel link from A to B, and vice versa, is active upon the initialization of the original pipeline, then the channel loop is live, and flow equivalence is observed between the original and restructured pipelines. To complete the proof, the aforementioned argument is also applied to a join. ■

Using Theorem 1, a simple algorithm that analyzes the directed graph representation of a pipeline is devised, where vertices represent the handshake components and arcs represent the channels, to determine whether a channel link exists between the optimization targets. If so, the channel link is checked for liveness to determine if the fusion of the optimization targets should be abandoned or not.

3) Optimization Target Selection: This section describes the proposed heuristic algorithm for the selection of the optimization targets. The optimization targets selected at each optimization iteration are those whose fusion is judged by the algorithm to be least likely to cause a change in the minimum cycle time of the pipeline.

In the algorithm, the S-invariants of the control network's PN model N are divided into groups of the same cycle time. The groups are then ranked by basing on their cycle times, i.e., the group with the maximum cycle time is ranked first, that with the next highest cycle time is ranked second, and so forth. Starting from the highest ranking group, the algorithm iteratively selects an S-invariant y_k and traces its path through N by examining $\|y_k\|$. Each handshake component whose STG is traversed by the path of y_k (for convenience, we say that the handshake component is associated with y_k) is assigned a score that is equal to the ranking of the group to which y_k belongs, if it does not have a score yet. The score of a handshake component is therefore "reversely" proportional to the cycle time of its associated S-invariant, i.e., the shorter the cycle time, the higher the score. Because the fusion of the optimization targets will modify only the associated invariants, it can be rationalized that if the fork or join with the highest combined handshake-component score is selected as optimization target, then the fusion of the targets is less likely to have an effect on the minimum cycle time of the pipeline. The algorithm is described in detail in Fig. 9.

1) Initialize Y as the set of all minimal-support S-invariants of N.
2) Initialize the current score δ to the value of 1.
3) While there exist handshake components that have not been assigned a score, do the following:
   3.1) Select from Y the S-invariant y_k that has the maximum cycle time.
   3.2) If the maximum cycle time is less than that of the previous iteration, then increment δ by 1.
   3.3) For each internal place p_i of $\|y_k\|$, assign the current value of δ to the handshake component whose STG contains p_i if it has not already been assigned a score.
   3.4) Remove y_k from Y.
4) Compute the score for each fork and join in the pipeline by summing the scores of its components.
5) Select as optimization targets the fork or join that has the highest score.

Fig. 9. Proposed algorithm for optimization target selection.

An extended version of the algorithm is also proposed, which considers, as optimization targets, all handshake-component pairs of the same type in the control network instead of just the pairs in fork or join configurations. Specifically, the extension involves modifying steps 4 and 5 of the original algorithm such that the score for each handshake-component pair of the same type is computed and the handshake-component pair with the highest score is selected as the optimization targets. If there are two or more candidate pairs with the same highest score, then priority is given to those that form forks or joins. The proposed extension increases the search space for optimization targets, and it can potentially lead to better results.

V. Fast Minimum-Cycle-Time Analysis

The optimization techniques proposed in this paper make a small incremental improvement to the asynchronous control network in each iteration of the optimization flow. For every optimization iteration (during which, either a handshake component is decoupled or two handshake components are fused), the minimum cycle time of the PN model needs to be updated. As explained in Section II-B, the computation of the minimum cycle time of a timed PN model requires finding all the minimal-support S-invariants of the model. It is known that the number of minimal-support S-invariants in a PN is upper bounded by the combinatory number $\binom{n}{\lceil n/2 \rceil}$ [13], where n is the number of places in the net and $\lceil * \rceil$ denotes rounding up to an integer. Thus, in the worst case, the number of minimal-support S-invariants in a net is exponential in the number of places in the net.

In this section, we propose an algorithm for iterative S-invariant computation that exploits the fact that only a small incremental change is made to the control network at each optimization iteration. The novelty of the proposed algorithm is that, instead of recomputing all S-invariants of the control network's PN model at each optimization iteration, it computes only those that are generated or modified due to the incremental change made to the model. As a result, the minimum-cycle-time analysis of the model at each optimization iteration is greatly expedited.

The algorithm consists of the macroplace extraction phase (Phase 1) and the invariant generation phase (Phase 2), which are executed before and after each incremental improvement of the control network, respectively.

A. Macroplace Extraction (Phase 1)

In the following, let N denote the PN model of a given control network at the beginning of each optimization iteration, let A denote the incidence matrix of N, and let m and n denote the number of places and transitions in N, respectively. Let G denote the subnet of N in which modifications are made, let B denote the incidence matrix of G, and let χ and δ denote the number of places and transitions in G, respectively.

In Phase 1 of the proposed algorithm, the macroplaces of N that form S-invariants with one or more places of G are extracted from the minimal-support S-invariants of N. Intuitively, a macroplace mp_k of N is created by the series fusion of two or more places of N. Mathematically, a macroplace of N corresponds to a row of A that is created by the linear combination of two or more rows of A (recall from Section II-B that each row of A corresponds to a place of N).

The method to extract a macroplace mp_k from a minimal-support S-invariant y_k of N, where mp_k forms an S-invariant with one or more places of G, is as follows. First, we rearrange A such that the first χ rows of A correspond to the places of G, and the first δ columns of A correspond to the transitions of G, i.e.,

$$A = \begin{matrix} \delta & n-\delta \\ \leftrightarrow & \leftrightarrow \end{matrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} \updownarrow \chi \\ \updownarrow m-\chi \end{matrix} \tag{2}$$

where $A_{11} = B$. Let $S_1 = \|y_k\| \cap P(G)$, where $P(G)$ denotes the places of G and $S_2 = \|y_k\| \backslash S_1$. We can see in (2) that the expression $\sum_{i=1}^{\chi} y_k[i] \cdot a_{ij}$ gives the number of tokens that are gained or lost by S_1 when transition t_j of G fires. Because the weighted sum of tokens held by the places of $\|y_k\|$ is, by definition, constant (i.e., $A^{\mathrm{T}} y_k = 0$), we know that the number of tokens that are gained or lost by S_2 when transition t_j fires is given by $-\sum_{i=1}^{\chi} y_k[i] \cdot a_{ij}$. If we let a macroplace mp_k of N be an abstraction of S_2, i.e., mp_k is formed by the series fusion of the places of S_2, then it can be deduced from the earlier statement that the weights of the arcs that connect mp_k to G are given by

$$u_k[j] = -\sum_{i=1}^{\chi} y_k[i] \cdot b_{ij}, \qquad j = 1, 2, \ldots, \delta \tag{3}$$

where $u_k[j]$ denotes the weight of the arc connecting mp_k to the jth transition of G (the directions of the arcs are given by the polarities of the weights; a positive weight means the arc is from a place of G to mp_k and vice versa). Thus, if at least one element of u_k is positive, then we say that y_k yields the macroplace mp_k, and u_k is appended as a new row to B. Furthermore, an entry is created for mp_k in the lookup table U such that $U(\mathrm{mp}_k) = q_k$, where $q_k = [y_k[\chi+1] y_k[\chi+2] \ldots y_k[m]]$ denotes the invariance vector of mp_k. A formal description of the macroplace extraction method is shown in Fig. 10.

/* Macro place extraction (Phase 1) */

1) Initialization: $C := B$, where B denotes the incidence matrix of G.
2) For each minimal-support S-invariant y_k of N, do the following:
   2.1) Extract the corresponding macro place mp_k by computing the weights of the arcs connecting mp_k to G:
   $$u_k[j] = -\sum_{i=1}^{\chi} y_k[i] \cdot b_{ij}\ ,\ j = 1, 2, \ldots, \delta$$
   where χ and δ denote the number of places and transitions, respectively, in G.
   2.2) If u_k is not a zero vector, then append u_k as a new row to C, and create an entry for mp_k in the lookup table U such that $U(mp_k) = q_k$, where $q_k = [y_k[\chi+1]\, y_k[\chi+2] \ldots y_k[m]]$ denotes the invariance vector of mp_k.
   2.3) If u_k is a zero vector and if $\|y_k\| \not\subset P(G)$, where $P(G)$ denotes the places of G, then y_k is a minimal-support S-invariant of R.

/* Invariant generation (Phase 2) */

3) Modify C to obtain V according to the changes made in G.
4) Compute all minimal-support S-invariants of V using the Fourier-Motzkin algorithm [13].
5) For each minimal-support S-invariant z_k of V, do the following:
   5.1) Create a vector y_k such that $y_k[i] = z_k[i]$ for $i = 1, 2, \ldots, \gamma$, where γ denotes the number of places in X less the macro places, and $y_k[i] = 0$ for $i = \gamma+1, \gamma+2, \ldots, \varepsilon$, where ε denotes the number of places in R.
   5.2) For each macro place $mp_j \in \|z_k\|$, do the following:
   $$y_k[i] = y_k[i] + (w_j \cdot U(mp_j)[i-\gamma]),\ i = \gamma+1, \gamma+2, \ldots, \varepsilon$$
   where w_j denotes the weight of mp_j in z_k.
   5.3) The resulting y_k is an S-invariant of R.

Fig. 10. Proposed algorithm for iterative S-invariant computation.

This method of macroplace extraction does not extract macroplaces that do not form invariants with the places of G. However, as we shall show, this exclusion does not affect the correctness of the method because such macroplaces also do not form invariants with the places of the modified G, provided that the modifications in G are restricted to those that are carried out by the proposed optimization techniques.

Let R denote the modified N and H the modified G (thus, H is a subnet of R). We show that the earlier assertion is true by proving its contra positive, i.e., if a macroplace forms an invariant with the places of H, then it also forms an invariant with the places of G. Thus, in the following proofs, we are considering the "reverse" transformation of H to G (and, equivalently, R to N).

In optimal decoupling, the steps to "reverse" transform a semicoupled or minimally coupled STG specification into a MAXC one can be classified into two types—place fusion and elimination.

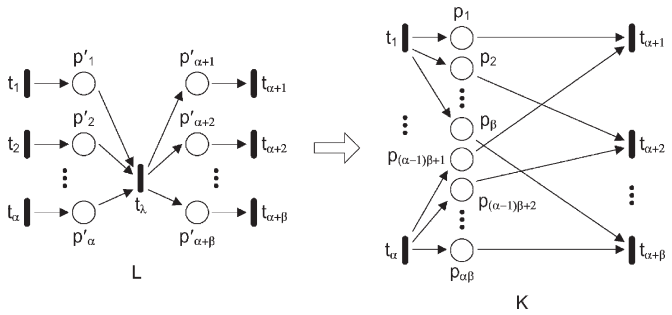


Fig. 11. Place fusion—a step in the "reverse" transformation of a decoupled STG specification into a MAXC specification (with reference to optimal decoupling).

Place fusion, shown in Fig. 11, refers to the positive linear combinations of each input place of a transition t_λ with each output place of t_λ, eliminating t_λ in the process. The following lemma states that the macroplace extraction method is correct for place fusion.

Lemma 1: Let K and L (see Fig. 11) be subnets of N and R, respectively. If a macroplace of R forms an S-invariant with the places of L, then it also forms an S-invariant with the places of K.

Proof: The proof is first summarized as follows. Let the macroplace mp_k form an S-invariant z_k with the places of L. To show that mp_k also forms an S-invariant with the places in K, we need to show that there exists a nonnegative m-integer vector y_k such that $A^{\mathrm{T}}y_k = 0$, where $A = [a_{ij}]$ is the incidence matrix of N, and the elements of y_k that correspond to mp_k are the same as those of z_k. Using $E^{\mathrm{T}}z_k = 0$, where $E = [e_{ij}]$ is the incidence matrix of R, and $A^{\mathrm{T}}y_k = 0$, a linear system of equations is set up. We then show that a nonnegative integer solution (not all zeros) exists for the linear system by reformulating the linear system as an integer programming [22] problem and proving that the linear programming relaxation of the problem has an integer solution.

The details of the proof are as follows. Using $E^{\mathrm{T}}z_k = 0$ and considering the transitions $t_1, t_2, \ldots, t_{\alpha+\beta}$ of R, we have

$$z_k[j] = \begin{cases} -\left(\sum\limits_{i=\alpha+\beta+1}^{\varepsilon} e_{ij} z_k[i]\right), & j = 1, 2, \ldots, \alpha \\ \sum\limits_{i=\alpha+\beta+1}^{\varepsilon} e_{ij} z_k[i], & j = \alpha+1, \alpha+2, \ldots, \alpha+\beta \end{cases} \tag{4}$$

where ε denotes the number of places in R. Likewise, using $E^{\mathrm{T}}z_k = 0$ and considering t_λ of R, we have

$$\sum_{i=1}^{\alpha} z_k[i] = \sum_{i=\alpha+1}^{\alpha+\beta} z_k[i]. \tag{5}$$

To show that mp_k also forms an S-invariant with the places in K, we need to show that there exists a nonnegative m-integer vector y_k such that $A^{\mathrm{T}}y_k = 0$ and $y_k[\alpha\beta + i] = z_k[\alpha + \beta + i]$ for $i = 1, 2, \ldots, m - \alpha\beta$, where α and β are the number of input and output transitions of t_λ, respectively. From $A^{\mathrm{T}}y_k = 0$, we get

$$\sum_{i=1}^{\beta} y_k[i+(j-1)\beta] = -\left(\sum_{i=\alpha\beta+1}^{m} a_{ij} y_k[i]\right), \qquad j = 1, 2, \ldots, \alpha$$

$$\sum_{i=1}^{\alpha} y_k[(i-1)\beta+(j-\alpha] = \sum_{i=\alpha\beta+1}^{m} a_{ij} y_k[i],$$

$$j = \alpha+1, \alpha+2, \ldots, \alpha+\beta. \tag{6}$$

Because $a_{(\alpha\beta+i)j} = e_{(\alpha+\beta+i)j}$ and $z_k[\alpha\beta + i] = y_k[\alpha + \beta + i]$ for $i = 1, 2, \ldots, m - \alpha\beta$ and $j = 1, 2, \ldots, \alpha + \beta$, we have

$$\sum_{i=\alpha\beta+1}^{m} a_{ij} y_k[i] = \sum_{i=\alpha+\beta+1}^{\varepsilon} e_{ij} z_k[i], \qquad j = 1, 2, \ldots, \alpha + \beta. \tag{7}$$

It can be derived from (4), (6), and (7) that

$$z_k[j] = \sum_{i=1}^{\beta} y_k\left[i+(j-1)\beta\right], \qquad j = 1, 2, \ldots, \alpha$$

$$z_k[j] = \sum_{i=1}^{\alpha} y_k\left[(i-1)\beta+(j-\alpha)\right], \qquad j = \alpha+1, \alpha+2, \ldots, \alpha+\beta \tag{8}$$

or equivalently

$$C\nu = d \tag{9}$$

where ν^{T}=$[y_k[1]y_k[2]\ldots y_k[\alpha\beta]]$, d^{T}=$[z_k[1]z_k[2]\ldots z_k[\alpha+\beta]]$, and C^{T} is the same as the incidence matrix of K except that all "-1" elements of K are changed to "$+1$" (note that the first α rows of C represent $\{t_1, t_2, \ldots, t_\alpha\}$ and the last β rows of C represent $\{t_{\alpha+1}, t_{\alpha+2}, \ldots, t_{\alpha+\beta}\}$).

Thus, to prove that y_k exists, it must be shown that there exists a nonnegative integer solution to ν (not all zeros). To that end, we reformulate the linear system in (9) as an integer programming problem by changing its equality relations into inequality constraints

$$\max\left\{\sum_{i=1}^{\alpha\beta} \nu[i] : C\nu \le d, \nu \in \mathbf{Z}_+^{\alpha\beta}\right\}. \tag{10}$$

We know that the linear programming relaxation in (10), i.e., $\max\{\sum_{i=1}^{\alpha\beta} \nu[i] : C\nu \le d, \nu \in \mathrm{R}_+^{\alpha\beta}\}$, has an integer solution because C is totally unimodular [22], i.e., C satisfies the following conditions: 1) $c_{ij} \in \{+1, -1, 0\}$ for all i, j (unity arc weights in K); 2) each column contains at most two nonzero elements ($|\bullet p| = |p \bullet| = 1$ for every $p \in P(K)$); and 3) there exists a partition (S_1, S_2) of the set S of rows such that each column j satisfies $\sum_{i\in S_1} c_{ij} = \sum_{i\in S_2} c_{ij}$. The satisfaction of condition 3) becomes obvious when (9) is partitioned into

$$C_1\nu = d1 \tag{11}$$

$$C_2\nu = d2 \tag{12}$$

where $C_1(=S_1)$ and $C_2(=S_2)$ comprise the first α and last β rows of C, respectively, $d_1^{\mathrm{T}}=[z_k[1]z_k[2]\dots z_k[\alpha]]$, and $d_2^{\mathrm{T}}=[z_k[\alpha+1]z_k[\alpha+2]\dots z_k[\alpha+\beta]]$. A useful observation from (11) and (12) is that, because each place of K has exactly one input transition from $\{t_1,t_2,\dots,t_\alpha\}$ and exactly one output transition from $\{t_{\alpha+1},t_{\alpha+2},\dots,t_{\alpha+\beta}\}$, we have

$$\sum_{i=1}^{\alpha}\mathrm{Row}_i(C_1)\nu=\sum_{i=1}^{\beta}\mathrm{Row}_i(C_2)\nu=\sum_{i=1}^{\alpha\beta}\nu[i] \tag{13}$$

where $\mathrm{Row}_i(C)$ denotes the ith row of C. Finally, we need to show that the (maximum) solution to ν satisfies $C\nu=d$ instead of $C\nu\le d$. For the sake of argument, suppose there exists a row i of C_1 such that $\mathrm{Row}_i(C_1)\nu<d_1$, then it follows that $\sum_{k=1}^{\alpha}\mathrm{Row}_k(C_1)\nu<\sum_{k=1}^{\alpha}d_1[k]$. Therefore, according to (5) and (13), $\sum_{k=1}^{\beta}\mathrm{Row}_k(C_2)\nu<\sum_{k=1}^{\beta}d_2[k]$, i.e., there is at least one row (let it be row j) in C_2 such that $\mathrm{Row}_j(C_2)\nu<d_2[j]$. As a result, there is a solution element $\nu[(i-1)\beta+k]$, where $k\in\{1,2,\dots,\beta\}$, that can be increased by one without violating $C\nu\le d$. This, however, contradicts the fact that $\max\{\sum_{i=1}^{\alpha\beta}\nu[i]\}$ has already been found. Thus, the solution to ν satisfies $C\nu=d$. ■

The second type of steps to "reverse" transform a decoupled STG specification into a MAXC one is place elimination. Let p be a place in H such that: 1) $|\bullet p|=|p\bullet|=1$ and 2) there exists a directed path Z in H from the input transition to the output transition of p, where $p\notin P(Z)$. The following lemma states that the macroplace extraction method is correct for place elimination.

Lemma 2: Let p be a place of H such that: 1) $|\bullet p|=|p\bullet|=1$ and 2) there exists a directed path Z in H from the input transition to the output transition of p, where $p\notin P(Z)$, i.e., there exists an ε-integer vector ν such that $\nu[k]=-1$ and $\sum_{i=1}^{\varepsilon}e_{ij}\nu[i]=0$ for $j=1,2,\dots,\phi$, where k is the index of p in R, e_{ij} is an element of the incidence matrix E of R, and ε and ϕ are the number of places and transitions in R, respectively. If a macroplace mp_k of R forms an S-invariant y_1 with the places of H such that $p\in\|y_1\|$, then mp_k also forms an S-invariant y_2 with the places of H such that $P(Z)\subseteq\|y_2\|$.

Proof: Let $k=1$ (there is no loss of generality because the index of a place in a PN is arbitrary). Because $A^{\mathrm{T}}y_1=0$, we have

$$a_{1j}w+\sum_{i=2}^{\varepsilon}a_{ij}y_1[i]=0,\qquad j=1,2,\dots,\phi \tag{14}$$

where w is the weight of p in y_1, i.e., $y_1[1]=w$. Substituting $a_{1j}=\sum_{i=2}^{\varepsilon}a_{ij}\nu[i]$ for $j=1,2,\dots,\phi$ into (14) and using $y_1[1]+w\nu[1]=0$, we get

$$\sum_{i=1}^{\varepsilon}a_{ij}\left(y_1[i]+w\nu[i]\right)=0,\qquad j=1,2,\dots,\phi. \tag{15}$$

That is, there exists an invariant $y_2=y_1+w\nu$ of N. It is easy to see that $P(Z)\subseteq\|y_2\|$. ■

In handshake-component fusion (the second proposed optimization technique), the "reverse" transformation of H into G refers to the transformation of the replacement circuit's STG specification into the two identical STG specifications of the optimization targets.

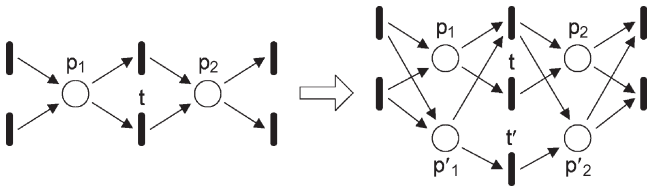


Fig. 12. Transition splitting—a step in the "reverse" transformation of a replacement circuit's STG specification into the two identical specifications of the optimization targets (with reference to handshake-component fusion).

This transformation involves splitting each transition t (see Fig. 12) of H by using the following steps. First, create a twin transition t'. Second, for each place $p_i\in\bullet t\backslash S$, where S contains the twin places created since the beginning of the transformation (S is initially empty), create a twin place p_i', if p_i' does not already exist (i.e., if $p_i'\notin S$), such that $\bullet p_i'=\bullet p_i$ and $p_i'\bullet=(p_i\bullet\backslash\{t\})\cup\{t'\}$ and add p_i' to S. On the other hand, if p_i' already exists (i.e., $p_i'\in S$), then modify the postset of p_i' such that $p_i'\bullet=(V\backslash\{t\})\cup\{t'\}$, where V denotes the original $p_i'\bullet$. Third, for each place $p_j\in t\bullet\backslash S$, create a twin place p_j', if p_j' does not already exist, such that $p_j'\bullet=p_j\bullet$ and $\bullet p_j'=(\bullet p_j\backslash\{t\})\cup\{t'\}$ and add p_j' to S. On the other hand, if p_j' already exists, then modify the preset of p' such that $\bullet p_j'=(Y\backslash\{t\})\cup\{t'\}$, where Y denotes the original $\bullet p_j'$. The following lemma states that the macroplace extraction method is correct for transition splitting.

Lemma 3: Let H be transformed to G by the splitting of some of the transitions of H. Let H be a subnet of R and G be a subnet of N. If a macroplace mp_k of R forms an S-invariant z_k with the places of H, then mp_k also forms an S-invariant y_k with the places of G.

Proof: To show that mp_k forms an S-invariant with the places of G, we need to show that there exists a nonnegative m-integer vector y_k such that $y_k[\gamma+i]=q_k[i]$ for $i=1,2,\dots,\varepsilon$, where $(q_k)^{\mathrm{T}}=[z_k[\gamma+1]z_k[\gamma+2]\dots z_k[\varepsilon]]$ is the invariance vector mp_k, and that $A^{\mathrm{T}}y_k=0$, where $A=[a_{ij}]$ is the incidence matrix of A.

Let $D=[d_{ij}]$ be the incidence matrix of H, and let γ and η be the number of places and transitions in H, respectively. Let $E=[e_{ij}]$ be the incidence matrix of R, and let ε and ϕ be the number of places and transitions in R, respectively. Let m and n be the number of places and transitions in N, respectively. Because H is a subnet of R, we can arrange the elements of E such that

$$d_{ij}=e_{ij},\qquad i=1,2,\dots,\gamma;\ j=1,2,\dots,\eta. \tag{16}$$

We can obtain A from E as follows. First, for each twin transition that is created during the splitting of the transitions of H, E is extended with a zero column (the zero columns reflect the fact that the twin transitions are not connected to the places of E). Thus

$$a_{ij}=0,\qquad i=1,2,\dots,\varepsilon;\ j=\phi+1,\phi+2,\dots,n. \tag{17}$$

Second, for each twin place p that is created during the splitting of the transitions of H, a new row vector of length n that

corresponds to p is appended to the extended E. Thus, the resulting A is such that

$$a_{ij} = e_{ij}, \qquad i = 1, 2, \ldots, \varepsilon;\ j = 1, 2, \ldots \phi. \tag{18}$$

Because mp_k of R forms an S-invariant z_k with the places of H, we have $E^{\mathrm{T}} z_k = 0$, i.e.,

$$\sum_{i=1}^{\varepsilon} e_{ij} z_k[i] = 0, \qquad j = 1, 2, \ldots, \phi. \tag{19}$$

Note that mp_k is defined by the invariance vector $(q_k)^{\mathrm{T}} = [z_k[\gamma+1] z_k[\gamma+2] \ldots z_k[\varepsilon]]$. In order to show that mp_k also forms an S-invariant with the places of G, we need to show that there exists a nonnegative m-integer vector y_k such that $y_k[\gamma+i] = q_k[i]$ for $i = 1, 2, \ldots, \varepsilon$ and $A^{\mathrm{T}} y_k = 0$. It is easy to see that, by letting $y_k[i] = z_k[i]$ for $i = 1, 2, \ldots, \gamma$ and letting $y_k[i] = 0$ for $i = \varepsilon+1, \varepsilon+2, \ldots, m$, and from (18) and (19), we have

$$\sum_{i=1}^{m} a_{ij} y_k[i] = \sum_{i=1}^{\varepsilon} e_{ij} z_k[i] = 0, \qquad j = 1, 2, \ldots, \phi. \tag{20}$$

Moreover, using (17), we have

$$\sum_{i=1}^{m} a_{ij} y_k[i] = 0 \qquad \text{for } j = \phi+1, \phi+2, \ldots, n. \tag{21}$$

Thus, mp_k forms the S-invariant $(y_k)^{\mathrm{T}} = [z_k[1] z_k[2] \ldots z_k[\varepsilon] 0\ 0 \ldots 0]$ with the places of G. ■

Lemmas 1, 2, and 3 lead to the following theorem.

Theorem 2: Let the PN R be transformed to N through place fusion, place elimination, and/or transition splitting. Let H denote the subnet of R in which the transformations are performed. Let G denote the transformed H (thus, G is a subnet of N). If a macroplace mp_k of R forms an S-invariant with the places of H, then mp_k also forms an S-invariant with the places of G.

Proof: See Lemmas 1, 2, and 3. ■

B. Invariant Generation (Phase 2)

In the following, Q denotes the net that contains G and the extracted macroplaces, and C denotes the incidence matrix of Q [thus, C is given by B (the incidence matrix of G) appended with rows that represent the extracted macroplaces; see (3)].

Phase 2 of the proposed algorithm begins with the modification of Q caused by STG decoupling or fusion. Let X denote the modified Q, and let V denote the incidence matrix of X. The minimal-support S-invariants of X are then computed by using the well-known Fourier–Motzkin (FM) algorithm [13].

For each minimal-support S-invariant z_k of X that is computed by the FM algorithm, the proposed algorithm searches for the presence of macroplaces in $\|z_k\|$. All macroplaces found in $\|z_k\|$ are then refined into their constituents (as recorded by the lookup table U) to generate the corresponding S-invariants of R, where R denotes the PN model of the control network after the network has been incrementally improved.

More precisely, for each minimal-support S-invariant z_k of X, an ε vector, which is denoted as y_k, where ε denotes the number of places in R, is created such that $y_k[i] = z_k[i]$ for $i = 1, 2, \ldots, \gamma$, where γ denotes the number of places in X less the macroplaces and $y_k[i] = 0$ for $i = \gamma+1, \gamma+2, \ldots, \varepsilon$. Then, for each macroplace $\mathrm{mp}_j \in \|z_k\|$, the following is executed:

$$y_k[i] = y_k[i] + (w_j \cdot U(mp_j)[i-\gamma]), \qquad i = \gamma+1, \gamma+2, \ldots, \varepsilon \tag{22}$$

where w_j is the weight of mp_j in z_k. The resulting y_k is an S-invariant of R. A formal description of the invariant generation phase is provided in Fig. 10.

The aforementioned procedure of generating an S-invariant of R from an S-invariant of X through macroplace refinement is correct according to the following theorem.

Theorem 3: Let mp be a macroplace of net N, i.e., there exists a vector ν such that $\nu[k] = -1$ and $\sum_{i=1}^{m} a_{ij} \nu[i] = 0$ for $j = 1, 2, \ldots, n$, where k is the index of mp in N, m and n are the number of places and transitions in N, respectively, and $A = [a_{ij}]$ is the incidence matrix of N. If N has an invariant y_1 such that $y_1[k] = w$, then there exists another invariant $y_2 = y_1 + w\nu$ of N.

Proof: This proof is the same as that for Lemma 2, except that the place p in Lemma 2 is replaced here with the macroplace mp. ■

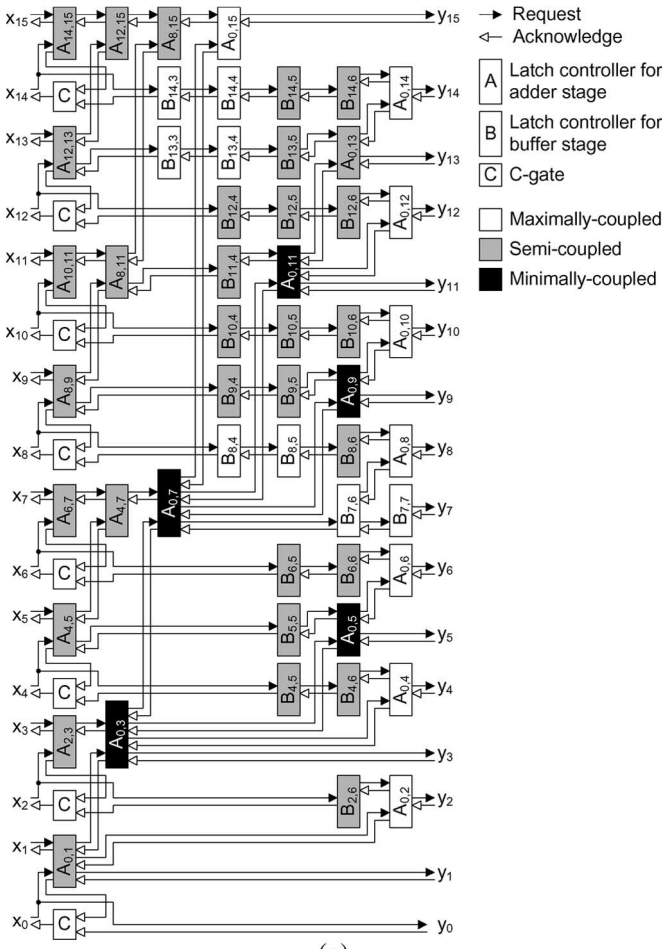
VI. Design Examples

In this section, we report the results of applying the proposed optimization methods on three designs: pipelined parallel prefix tree, cross-pipelined array multiplier, and Reed–Solomon error detector. The speed and energy dissipations of the circuits are obtained by using transistor-level Simulation Program with Integrated Circuit Emphasis simulations (Synopsys Nanosim) at a supply voltage of 3.3 V, using the AMS 0.35 μm CMOS process. The simulations do not take into account the interconnect parasitics.

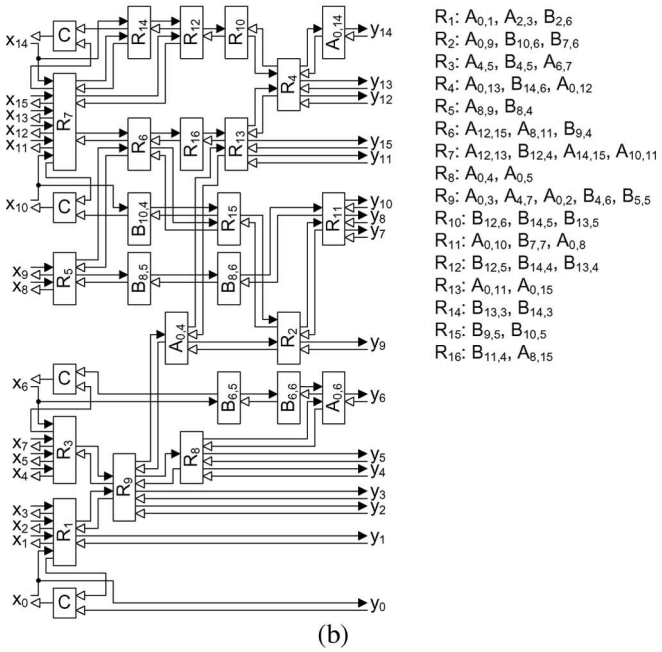
A. Pipelined Parallel Prefix Computation

The prefix problem is to compute, given $x_1, x_2, \ldots, x_n$, the results $y_1, y_2, \ldots, y_n$, where $y_k = x_1 \otimes x_2 \otimes \cdots \otimes x_k$ for $1 \leq k \leq n$ [23] and $\otimes$ is an associative operation (such as addition). A parallel prefix tree exploits the associativity of the operation and computes all the prefixes concurrently. The pipelined implementations of the parallel prefix tree using synchronous [24] and asynchronous [25] techniques have been reported.

Fig. 13(a) shows the control network (excluding the delay-matching elements) of our asynchronous implementation of the pipelined parallel prefix tree for the addition operation and for $n = 16$ after the optimal decoupling method has been applied on it. The optimally decoupled network (OPTD) consists of 28% MAXC, 60% semicoupled, and 12% minimally coupled handshake components. We also implemented other parallel prefix trees using the following handshake-component design styles: fall decoupled [11], desynchronization control [11], MAXC [9], semidecoupled [9], and FULLYD [9]. Compared with these networks (except the MAXC network), OPTD has at least 8% fewer transistors and dissipates at least 11% less energy per computation and yet is one of the fastest (see Table I).



(a)



(b)

Fig. 13. Control network of asynchronous pipelined parallel prefix tree for addition ($n = 16$). (a) Optimally decoupled. (b) After handshake-component fusion.

Compared with the MAXC circuit, the optimized circuit is 24% faster and 38% better in $E\tau^2$ (note that $E\tau^2$ is independent of the supply voltage at the first order of approximation [26]).

TABLE I
COMPARISONS OF PIPELINED PARALLEL PREFIX TREES BASED ON DIFFERENT HANDSHAKE-COMPONENT CONFIGURATIONS

Configuration	# of Trans. (k)		Energy (pJ)		$\tau_{\min}$	$E\tau^2$
	Control	Total	Control	Total	(ns)	(10^{-26}Js^2)
FALLD [11]	5.17	13.9	264	402	8.07	2.62
DESYNC [11]	5.86	14.6	353	497	7.59	2.86
MAXC	3.71	12.5	185	336	9.24	2.87
SEMID [9]	5.04	13.8	246	388	9.98	3.86
FULLYD [9]	5.83	14.6	375	515	7.83	4.03
OPTD	4.65	13.4	219	360	7.01	1.77

TABLE II
RESULTS OF PIPELINED PARALLEL PREFIX TREE AFTER HANDSHAKE-COMPONENT FUSION

	# of Trans. (k)		Energy (pJ)		$\tau_{\min}$	$E\tau^2$
	Control	Total	Control	Total	(ns)	(10^{-26}Js^2)
Fusion	2.23	11.0	106	268	9.86	2.61
FusionE	3.15	11.9	145	307	10.0	3..07

The FusionE was applied on the MAXC network, with the minimum-cycle-time constraint set at 10% higher than that of the full network. Fig. 13(b) shows the optimized network, where the replacement components are labeled as R. For example, the components $\{A_{0,1}, A_{2,3}, B_{2,6}\}$ are replaced with R_1. Compared with the MAXC network, the optimized network has 40% fewer transistors and dissipates 43% less energy (see Table II).

The extended version of the FusionE was also applied on the MAXC network. Despite the larger number of the considered optimization targets, the extended version was less effective than the standard version (the former provided 15% and 22% improvements in transistor count and energy dissipation, respectively). This is because, when handshake components that do not form forks or joins are fused, new channels are created between handshake components that are not directly related (channelwise), thus increasing the control network's complexity. As a result, many optimization targets were not successfully fused due to either the nonpreservation of flow equivalence or the violation of the minimum-cycle-time constraint.

B. Cross-Pipelined Multiplier

The cross-pipelined multiplier [27] has a highly regular structure consisting of interconnected cells, each containing a handshake component (a basic latch controller), a computation block, and latches. Fig. 14 shows the control network of a 16-b cross-pipelined multiplier (excluding the delay-matching elements).

We applied the optimal decoupling technique on the control network of the multiplier and obtained an optimized network which consists of MAXC (54%) and semicoupled (46%) handshake components but has no minimally coupled ones. Compared with the other control networks based on uniform handshake-component concurrence (with the exception of the MAXC network), the OPTD has at least 8% fewer transistors, dissipates at least 19% less energy, and yet is the fastest (see Table III). Compared with the MAXC circuit, it is 27% faster and 44% better in $E\tau^2$.

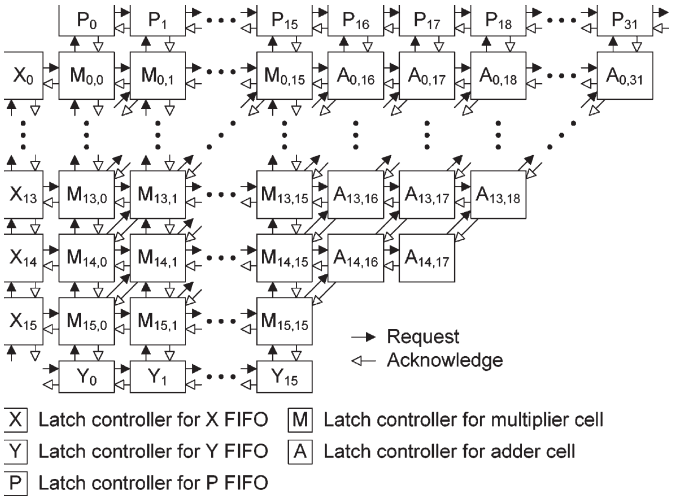


Fig. 14. Control network of asynchronous 16-b cross-pipelined array multiplier.

TABLE III
COMPARISONS OF 16-b CROSS-PIPELINED MULTIPLIERS BASED ON DIFFERENT HANDSHAKE-COMPONENT CONFIGURATIONS

Configuration	# of Trans. (k)		Energy (nJ)		τ_{min}	$E\tau^2$
	Control	Total	Control	Total	(ns)	(10^{-25}Js^2)
FALLD [11]	56.1	153	2.72	3.08	7.42	1.70
DESYNC [11]	63.7	161	3.29	3.65	6.93	1.75
MAXC	45.1	142	1.85	2.21	8.37	1.55
SEMID [9]	53.8	151	2.67	3.05	8.96	2.45
FULLYD [9]	61.5	158	3.25	3.62	7.14	1.85
OPTD	49.6	146	2.16	2.51	6.12	0.94

TABLE IV
RESULTS OF 16-b CROSS-PIPELINED MULTIPLIERS AFTER HANDSHAKE-COMPONENT FUSION

	# of Trans. (k)		Energy (nJ)		τ_{min}	$E\tau^2$
	Control	Total	Control	Total	(ns)	(10^{-25}Js^2)
Fusion	12.1	109	0.69	1.06	8.63	0.79
FusionE	28.9	125	1.51	1.87	8.94	1.49

The FusionE was applied on the MAXC control network. The result is a network that has 73% fewer transistors and dissipates 63% less energy than the full network (see Table IV).

C. Reed–Solomon Error Detector

The Reed–Solomon error detector for the compact disk player [28], [29] accepts codewords consisting of 28 or 32 8-b symbols. The error detector comprises two main functional blocks. The first block consists of a functional loop that inputs a codeword symbol by symbol and computes the four syndromes for the codeword. Once the syndromes are computed, they are transferred to the second block which determines the error values and locations.

Our error detector is based on the handshake components we developed for general asynchronous systems [30]. Fig. 15(a) shows the control network of the error detector (excluding the delay-matching elements). Syndrome computation is controlled by the ChannelMux (M_0, M_1, M_2, and M_3) and SyncPassiveOut (P_0, P_1, P_2, and P_3) components. A counter, which is controlled by a Sync component (S_0), counts the number of

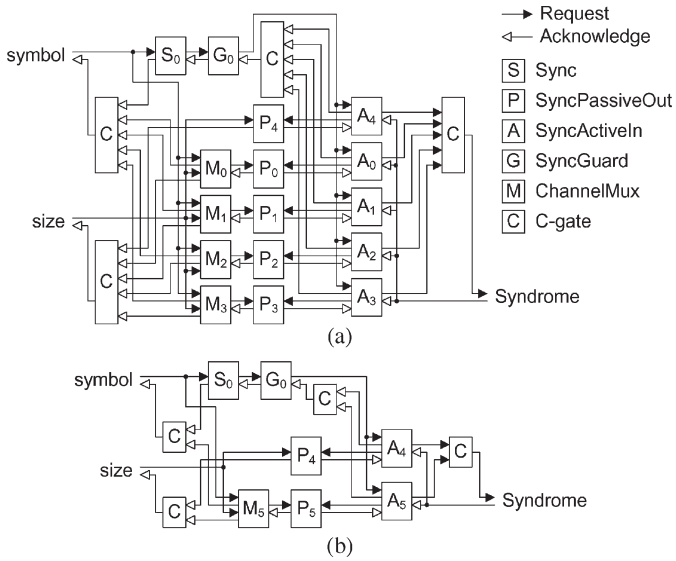


Fig. 15. Control network of asynchronous Reed–Solomon error detector (syndrome computation). (a) Original. (b) After handshake-component fusion.

TABLE V
COMPARISONS OF REED–SOLOMON ERROR DETECTORS BASED ON DIFFERENT HANDSHAKE-COMPONENT CONFIGURATIONS

Configuration	# of Trans. (k)		Energy (nJ)		τ_{min}	$E\tau^2$
	Control	Total	Control	Total	(ns)	(10^{-22}Js^2)
MAXC	1.22	6.98	1.63	5.04	269	3.65
SEMIC	1.45	7.22	2.08	5.53	276	4.21
MINC	1.67	7.43	2.57	6.06	278	4.68

TABLE VI
RESULTS OF REED–SOLOMON ERROR DETECTORS AFTER HANDSHAKE-COMPONENT FUSION

	# of Trans. (k)		Energy (nJ)		τ_{min}	$E\tau^2$
	Control	Total	Control	Total	(ns)	(10^{-22}Js^2)
Fusion	0.67	6.44	1.38	4.31	253	2.76

symbols received, and its value is monitored by a SyncGuard component (G_0). Upon reading an entire codeword, G_0 initiates a request to the SyncActiveIn components (A_0, A_1, A_2, and A_3), which control the transfer of the computed syndromes to the error detection block.

The speed of the error detector is mainly dependent on how fast the syndromes are computed. Because syndrome computation is accumulative, it is easy to see that the handshake-component concurrence is not likely to play an important role in determining the speed of the error detector. This intuition was confirmed by the optimal decoupling algorithm, which returned a control network consisting of only MAXC handshake components. As shown in Table V, the MAXC network has at least 16% fewer transistors and dissipates at least 22% less energy per codeword than the semicoupled and minimally coupled networks.

The simultaneity of syndrome computation suggests that it is possible to fuse the handshake components controlling the computation while preserving the throughput. We applied the FusionE on the control network and obtained an optimized network, as shown in Fig. 15(b). The optimization consists of fusing $\{M_0, M_1, M_2, M_3\}$, $\{P_1, P_2, P_3, P_4\}$, and

$\{A_0, A_1, A_2, \text{and} A_3\}$, replacing them with M_5, P_5, and A_5, respectively. The optimized control network has 45% fewer transistors and dissipates 15% less energy than the MAXC network (see Table VI).

VII. Conclusion

We have proposed two optimization methods for asynchronous control networks. We have also proposed a fast method for iterative minimum-cycle-time analysis that is incorporated into the optimization methods. The proposed methods have been developed formally and rigorously. Experimental results based on nontrivial design examples have indicated that the proposed optimization methods are able to provide significant improvements in transistor count and energy dissipation of asynchronous control networks.

References

[1] I. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.

[2] L. A. Plana, S. Taylor, and D. Edwards, "Attacking control overhead to improve synthesised asynchronous circuit performance," in *Proc. ICCD*, 2005, pp. 703–710.

[3] A. Davare, K. Lwin, Kondratyev, and A. L. Sangiovanni-Vincentelli, "The best of both worlds: The efficient asynchronous implementation of synchronous specifications," in *Proc. DAC*, 2004, pp. 588–591.

[4] T. Chelcea and S. M. Nowick, "Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems," in *Proc. DAC*, 2002, pp. 405–410.

[5] T. Kolks, S. Vercauteren, and B. Lin, "Control resynthesis for control-dominated asynchronous designs," in *Proc. ASYNC*, 1996, pp. 233–243.

[6] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, F. Schalij, and R. van de Wiel, "A single-rail re-implementation of a DCC error detector using a generic standard-cell library," in *Proc. Conf. Asynchronous Des. Methodologies*, 1995, pp. 72–79.

[7] J. Ebergen, J. Gainsley, J. Lexau, and I. Sutherland, "GasP control for domino circuits," in *Proc. ASYNC*, 2005, pp. 12–22.

[8] M. Singh and S. Nowick, "MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. ICCD*, 2001, pp. 9–17.

[9] S. B. Furber and P. Day, "Four-phase micropipeline latch control circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 2, pp. 264–272, Jun. 1996.

[10] T. A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, MIT, Cambridge, MA, 1987.

[11] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 1904–1921, Oct. 2006.

[12] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[13] M. Silva and J. M. Colom, *Convex Geometry and Semiflows in P/T Nets: A Comparative Study of Algorithms for Computation of Minimal P-Semiflows*, vol. 483, in *Lectures Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 1991, pp. 79–112.

[14] M. Lewis, J. Garside, and L. Brackenbury, "Reconfigurable latch controllers for low power asynchronous circuits," in *Proc. ASYNC*, 1999, pp. 27–35.

[15] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. Inf. Syst.*, vol. E80-D, no. 3, pp. 315–325, Mar. 1997.

[16] S. Kim and A. Beerel, "Pipeline optimization for asynchronous circuits: Complexity analysis and an efficient optimal algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 3, pp. 389–402, Mar. 2006.

[17] P. A. Beerel, A. Lines, M. Davies, and N. H. Kim, "Slack matching asynchronous designs," in *Proc. ASYNC*, 2006, pp. 184–194.

[18] P. Prakash and A. J. Martin, "Slack matching quasi delay-insensitive circuits," in *Proc. ASYNC*, 2006, pp. 195–204.

[19] J. Teifel, D. Fang, D. Biermann, C. Kelly, and R. Manohar, "Energy-efficient pipelines," in *Proc. ASYNC*, 2002, pp. 23–33.

[20] E. K. Burke, Ed., *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, 1st ed. New York: Springer-Verlag, 2005.

[21] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proc. IEEE*, vol. 89, no. 5, pp. 665–692, May 2001.

[22] L. A. Wolsey, *Integer Programming*. New York: Wiley, 1998.

[23] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[24] K. Papadantonakis, N. Kapre, S. Chan, and A. DeHon, "Pipelining saturated accumulation," in *Proc. IEEE Int. Conf. Field Programmable Technol.*, 2005, pp. 19–26.

[25] R. Manohar and J. A. Tierno, "Asynchronous parallel prefix computation," *IEEE Trans. Comput.*, vol. 47, no. 11, pp. 1244–1252, Nov. 1998.

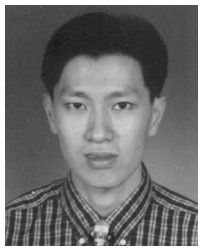
[26] A. J. Martin, "Towards an energy complexity of computation," *Inf. Process. Lett.*, vol. 77, no. 2, pp. 181–187, Feb. 2001.

[27] J. Butas, C. S. Choy, J. Povazanec, and C. F. Chan, "Asynchronous cross-pipelined multiplier," *IEEE J. Solid-State Circuits*, vol. 36, no. 8, pp. 1272–1275, Aug. 2001.

[28] H. Jacobson, E. Brunvand, G. Gopalakrishnan, and P. Kudva, "High-level asynchronous system design using the ACK framework," in *Proc. ASYNC*, 2000, pp. 93–103.

[29] J. Kessels, "VLSI programming of a low-power asynchronous Reed–Solomon decoder for the DCC player," in *Proc. Conf. Asynchronous Des. Methodologies*, 1995, pp. 44–52.

[30] C. F. Law, "Design methodologies for low-power asynchronous-logic digital systems," Nan. Tech. Univ., Aug. 2004. Tech. Rep..



Chong-Fatt Law received the B.Eng. (first-class honors) and M.Eng. degrees in electrical and electronic engineering from the Nanyang Technological University (NTU), Singapore, in 1997 and 1999, respectively.

He is a Research Associate with the School of Electrical and Electronic Engineering, NTU. His research interests include design and optimization methods for asynchronous circuits, computer-aided design algorithms and tools, and low-power very large scale integration design.

Bah-Hwee Gwee (S'93–M'97–SM'03) received the B.Eng. degree in electrical and electronic engineering from the University of Aberdeen, Aberdeen, U.K., in 1990, and the M.Eng. and Ph.D. degrees from the Nanyang Technological University (NTU), Singapore, in 1992 and 1998, respectively.

He is an Associate Professor with the School of Electrical and Electronic Engineering, NTU. His research interests include low-power asynchronous microprocessor and digital signal processor design, Class-D amplifiers, and soft computing.

Dr. Gwee was the Chair of IEEE Singapore—Circuits and Systems Chapter in 2005 and 2006. He was the Secretary of IEEE Biomedical Circuits and Systems in 2004, Publication Chair of IEEE Asia Pacific Conference on Circuits and Systems in 2006, and Technical Program Chair of International Symposium on Integrated Circuits in 2007.

Joseph S. Chang (M'04) received the B.Eng. degree in electrical and computer systems engineering from Monash University, Melbourne, Australia, in 1983, and the Ph.D. degree in the Department of Otolaryngology, University of Melbourne, Melbourne, in 1990.

He is an Associate Professor with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. His research interests include analog and digital signal processing, very large scale integration design, speech perception, biomedical engineering, and hearing instrument (hearing aid) research. He is the holder of several patents and has several pending patents in circuit design.

Dr. Chang is the recipient of the Best Presentation of a Paper Award at the Microelectronics Conference in 1989. He served as the Chairperson of the International Symposium on Intelligent Control in 2004.