PAPER Special Section on Parallel and Distributed Computing and Networking

A Graph Rewriting Approach for Converting Asynchronous ROMs into Synchronous Ones

Md. Nazrul Islam MONDAL[†], *Nonmember*, **Koji NAKANO**[†a)], *and* **Yasuaki ITO**[†], *Members*

SUMMARY Most of FPGAs have Configurable Logic Blocks (CLBs) to implement combinational and sequential circuits and block RAMs to implement Random Access Memories (RAMs) and Read Only Memories (ROMs). Circuit design that minimizes the number of clock cycles is easy if we use asynchronous read operations. However, most of FPGAs support synchronous read operations, but do not support asynchronous read operations. The main contribution of this paper is to provide one of the potent approaches to resolve this problem. We assume that a circuit using asynchronous ROMs designed by a non-expert or quickly designed by an expert is given. Our goal is to convert this circuit with asynchronous ROMs into an equivalent circuit with synchronous ones. The resulting circuit with synchronous ROMs can be embedded into FPGAs. We also discuss several techniques to decrease the latency and increase the clock frequency of the resulting circuits.

key words: *FPGA, block RAMs, asynchronous read operations, rewriting algorithm*

1. Introduction

An FPGA is a programmable VLSI (Very Large Scale Integration) in which a hardware designed by the users can be embedded quickly. Typical FPGAs consist of an array of programmable logic blocks (slices), memory blocks, and programmable interconnects between them. The logic block contains four-input logic functions implemented by a LUT and/or several registers. Using four-input logic functions, registers, and their interconnections, any combinational circuit and sequential logic can be implemented. The memory block is a dual-port RAM which can perform read and/or write operations for a word of data to two distinct or same addresses in the same time. Usually, the dual-port RAM supports synchronous read and synchronous write operations. The read and write operations are performed at the rising clock edges. The dual-port RAM outputs data of a specified address after the rising edge. Similarly data is written to a specified address at the rising edge of clock if write enable is high. Design tools are available to the users to embed their hardware logic into the FPGAs. Some circuit implementations are described [1]–[4] to accelerate computation.

In this paper, we focus on the asynchronous and synchronous read operations of memory blocks as follows:

Asynchronous read operation The memory block outputs the data specified by the address given to the address port. When the address value is changed, the output data is updated immediately within some delay time. In other words, the output data port always outputs $M[d]$, which is the data stored in the input address value d.

Synchronous read operation Even if the address value is changed, the output data is not updated. The output data is updated based on the address value at the rising edge of clock. More specifically, the output data port outputs $M[d]$, where d is the address data at the previous point of rising clock edge.

Let *AROMs* and *SROMs* denote ROMs with asynchronous and synchronous read operations, respectively. In general, the circuit design is simpler and easier to the designers, in particular to the non-expert circuit designers if AROMs are available. In asynchronous read operation, the value of a specified address can be obtained immediately. However, in synchronous read operation, one clock cycle is required to obtain it. Nevertheless, block RAMs embedded in most of the current FPGAs do not support asynchronous read operation for increasing its operating clock frequency.

The main contribution of this paper is to present a circuit rewriting approach that converts *an asynchronous circuit* consisting

combinational circuits (CCs), registers (Rs), and ROMs with asynchronous read operations (AROMs)

into *an equivalent synchronous circuit* consisting

combinational circuits (CCs), registers (Rs), and ROMs with synchronous read operations (SROMs).

Note that, most of the current FPGAs support synchronous read operation, but do not support asynchronous one. We are thinking the following scenario to use our circuit rewriting algorithm:

- An asynchronous circuit designed by a non-expert, or quickly designed by an expert is given.
- Our circuit rewriting algorithm convert it into an equivalent synchronous circuit.
- The resulting synchronous circuit can be implemented in FPGAs.

In other words, designers can design a circuit for FPGAs under the assumption of asynchronous read operation, which

Manuscript received January 6, 2011.
Manuscript revised June 6, 2011.

[†]The authors are with the Department of Information Engineering, Hiroshima University, Higashihiroshima-shi, 739–8527 Japan.
a) E-mail: nakano@cs.hiroshima-u.ac.jp

DOI: 10.1587/transinf.E94.D.2378

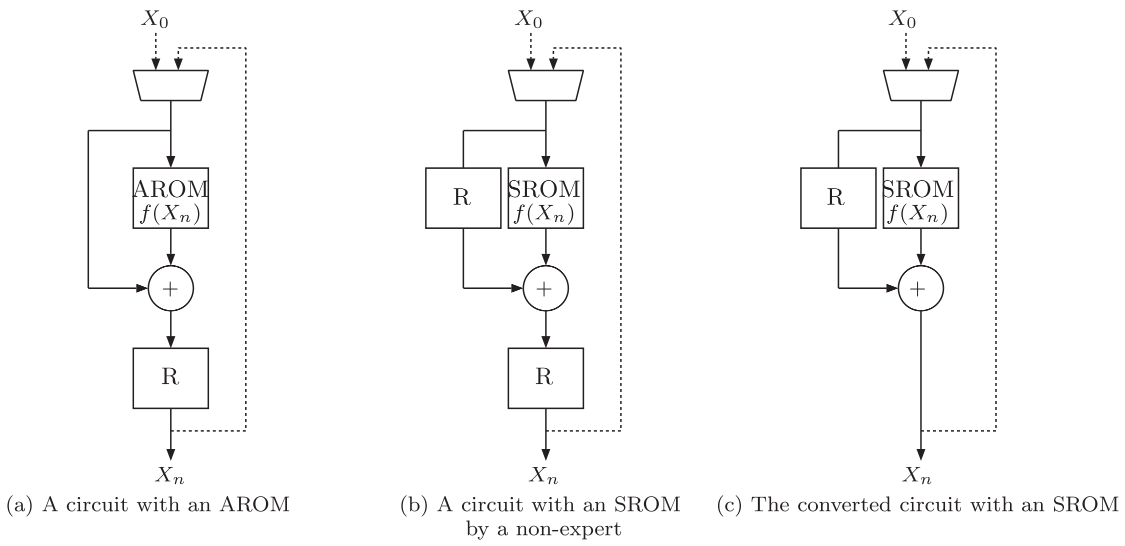
X_0
AROM
$f(X_n)$
+
R
X_n
(a) A circuit with an AROM
X_0
R
SROM
$f(X_n)$
+
R
X_n
(b) A circuit with an SROM by a non-expert
X_0
R
SROM
$f(X_n)$
+
X_n
(c) The converted circuit with an SROM

Fig. 1 An example of circuits using an AROM and an SROM.

is simpler and easier than one with synchronous read operation.

We will show a simple example illustrating that the circuit design is simpler if AROMs are available. Suppose that for an input X_0, we need to compute $X_n = X_{n-1} + f(X_{n-1})$ for every $n \geq 1$. We assume that the function f is computed using a ROM. More specifically, we use a ROM such that address i is storing a value of $f(i)$. Figure 1 (a) illustrates a circuit with an AROM to compute $X_1, X_2, \ldots$ for input X_0. An AROM is used to compute the value of $f(X_n)$ for a given X_n. It should be clear that this circuit outputs $X_1, X_2, \ldots$ in every clock cycle. Figure 1 (b) shows a circuit with an SROM. Since one clock cycle is necessary to read the value of $f(X_n)$ for input X_n, we need to insert a register to synchronize two inputs X_n and $f(X_n)$ of the adder as illustrated in the figure. This circuit outputs $X_1, X_2, \ldots$ in every two clock cycles. Hence, the circuit in Fig. 1 (b) needs double clock cycles over the circuit in Fig. 1 (a). Using our algorithm to the sub-circuit with solid lines (wires) in Fig. 1 (a), we can obtain the circuit in Fig. 1 (c) automatically. In the circuit with an SROM in Fig. 1 (c), $X_1, X_2, \ldots$ is output in every clock cycle. Thus, the timings of the circuits in Fig. 1 (a) and (c) are identical.

It is not trivial for the non-expert designers to minimize the number of clock cycles to obtain circuit as illustrated in Fig. 1 (c). However, our algorithm can do it automatically. Although, clock performance may degrade in the converted circuit; however, designers can make a trade-off between the maximum delay and number of clock cycles for their designs. The readers should refer to Fig. 2 for an illustration. In Fig. 2, the number of clock cycles is increased as well as the maximum delay is decreased by inserting the pipelined registers. In general, the insertion of the pipelined registers is not difficult. On the other hand, our algorithm may decrease the number of clock cycles by removing redundant registers. Although it may increase the maximum delay, sometimes the resulting circuit takes smaller total computing time.

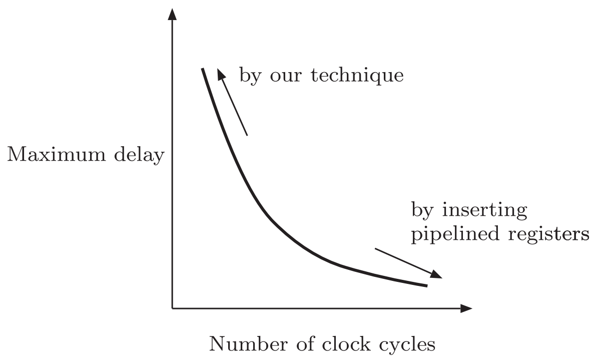

Fig. 2 A relation between the maximum delay and the number of clock cycles.

The outlines of our new idea are as follows:

1. We introduce *a negative register* (NR), which is an imaginary register latching a future input data.
2. We define simple *five rules* that rewrite a circuit.
3. The rewriting algorithm that we propose just repeats applying these rules until no more rules can be applied. When the rewriting algorithm terminates, we have an equivalent AROM-free circuit to the original circuit.

The key and innovative idea is to introduce a negative register. In our rewriting algorithm, a circuit with AROMs is first converted into an AROM-free circuit with negative registers. After that, our algorithm continues to rewrite circuit such that all NRs are removed. When the algorithm terminates, all negative registers will be removed if possible, and the resulting circuit becomes an equivalent to the original circuit.

A circuit implementation with AROMs is better than SROMs implementation, because of less power consumption, easy to design etc. But it has some problems like small in size so that it does not support the designer's demand, more expensive, and less speedy [5]–[7]. To cut the clock distribution power, an asynchronous circuit design in FPGAs is very much suitable, described in [8]–[10]. However, it is not supported by the current FPGAs.

On the other hand, a circuit implementation with SROMs is dominating the modern digital circuit design industry, because it supports the modern FPGA architecture although it has some drawbacks to design like clock distribution, more power consumption etc [5], [7]. So we should use SROMs when we need a function of ROMs.

One of the research works described the implementation of asynchronous circuit in FPGA [11]. In this paper, they described the problems like hazards, timing constraints, state holding elements, analog components and decomposition of the asynchronous circuit implementation in FPGA. Another research work described a novel FPGA architecture for implementing various styles of asynchronous logic [12]. They implemented a full-adder circuit in two different logic styles. While in synchronous circuits a clock globally controls the activity where as asynchronous circuit activity is locally controlled using communication channels to detect the presence of data at their inputs and outputs. An asynchronous module communicates with each other using requests and acknowledges [13]. Some dedicated FPGAs have also been developed to test asynchronous designs. Unfortunately, these FPGAs are closely associated to a style of design. For instance, MONTAGE [11] and PGA-STC [14] are based on an asynchronous design, GALSA [15] and STACC [16] are globally asynchronous FPGAs but locally synchronous and PAPA [17] is a fully asynchronous FPGA dedicated to optimize pipeline circuits.

To the best of our knowledge, there is no previous research work on our topic. It is well known that the architecture of the current FPGAs is the best suited for digital synchronous circuit designs. Unfortunately, they do not have block RAMs supporting asynchronous read operations. It is also known that AROM is implemented in LUTs which is easy to use because of the immediate output of data. However it is small in size and costly. Therefore, our target is to generate an AROM-free fully synchronous sequential circuit from a sequential circuit with AROMs which is an equivalent to the original circuit so that it can support the modern FPGA architecture.

We summarize several significant points of our results as follows:

- Negative registers (NRs) are newly introduced. Further, the correctness of our algorithm is proved in a rigorous manner.
- Our circuit rewriting algorithm moves all redundant registers toward the output ports. They can be removed to decrease the latency of the circuit. Therefore, the circuit that obtained has minimum latency in the sense that all redundant registers are deleted.
- We can also improve the clock frequency by inserting registers appropriately. These performance improvement technique for the resulting circuit will be discussed in Sect. 5.
- FPGA vendors may think that they will support asynchronous read operation for next-generation FPGAs satisfying low latency circuits with forfeiting the high clock frequency. If this is the case, our rewriting approach is useless. However, our results suggest to the FPGA vendors that support of asynchronous read operation is not necessary, because it can be automatically converted into synchronous one using our algorithm.
- The readers may think that circuits dealt with this paper are too restricted where as circuits in real-world are more complicated. However, it may be possible to extract a sub-circuit from the complicated circuit. We can then apply our circuit rewriting algorithm to this subcircuit.
- Even if a user designs a circuit with pipeline structure, our algorithm moves pipeline registers toward the output ports and destroys the pipeline structure. However, it may be possible to perform AROM-free conversion locally without collapsing a global pipeline structure. For this purpose, we need to extract sub-circuits in the original circuit such that they contain no pipeline register. By using our algorithm for each sub-circuit, it can be converted into AROM-free circuit. Since the timing of each sub-circuit is not changed, the whole converted circuit is identical to the original circuit. In this way, our algorithm may be applicable to the pipelined circuits.

This paper is organized as follows: Section 2 briefly describes the circuits and their equivalency. In Sect. 3, we describe our rewriting algorithm, circuit graph and also explain the equivalency for our rewriting rules. Section 4 presents the proof of the correctness of our rewriting algorithm. In Sect. 5, we explain the performance improvement of the AROM-free resulting circuits. Finally Sect. 6 concludes this work and also describes the future works.

2. Circuits and Their Equivalence

Let us consider a synchronous sequential circuit that consists of input ports, output ports, combinational circuits (CCs), registers (Rs), Read Only Memories (ROMs), a global clock input (clock), and a global reset input (reset).

A combinational circuit (CC) is a network of fundamental logic gates with no feedback. So, it can compute Boolean functions represented by Boolean formulas, such as $F = A \cdot \overline{B} + B \cdot C$ and $G = \overline{B \cdot C}$ as illustrated in Fig. 3. Once inputs are given, the outputs are computed in small propagation delay.

A b-bit register has a clock input and a reset input. It can store a b-bit data as shown in Fig. 4. If reset is 1, then the b-bit data is initialized by 0. If reset is 0, the stored data is updated by the value given to the input port d at every rising clock edge. The data stored in the register is always output from port q.

A ROM (Read Only Memory) has a b-bit input d

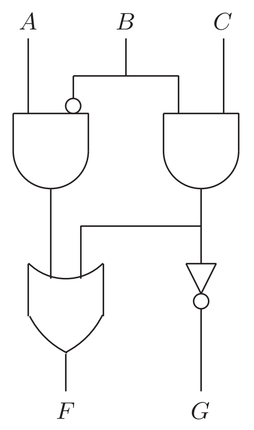

Fig. 3 An example of a combinational circuit (CC).

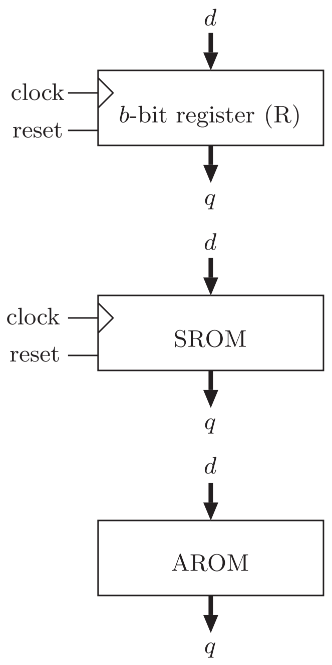

Fig. 4 A register (R), a synchronous ROM (SROM) and an asynchronous ROM (AROM).

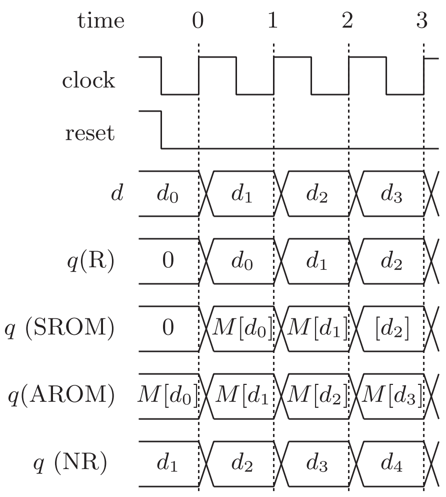

Fig. 5 A timing chart of a register (R), an SROM, an AROM and a negative register (NR).

and a c-bit data output q. It is storing 2^b words such as $M[0], M[1], \ldots, M[2^b - 1]$ with c bits each. We deal with two types of ROMs in terms of read operations as follows:

- **Synchronous ROM (SROM)** An SROM has a clock input and a reset input. If reset is 1 then the stored value is initialized by 0. The read operation is performed at every rising clock edge when reset is 0. The output q is the value of $M[d]$ at the latest rising clock edge.
- **Asynchronous ROM (AROM)** An AROM has no clock input and no reset input. The value of $M[d]$ is continuously output from port q.

The Fig. 5 shows a timing diagram of reading operations of the R, SROM, AROM and NR. In the figure, time $0, 1, 2, \ldots$ correspond to rising edges of the periodic clock input. Initially global reset is 1 and it drops to 0 just before time 0. Data $d_0, d_1, d_2, \ldots$ are given to the input port d. As shown in the figure, the value of output, q of R and SROM is 0 at time 0. Also, at time $1, 2, \ldots$ the values of output, q of R and SROM are $d_0, d_1, d_2, \ldots$ and $M[d_0], M[d_1], M[d_2], \ldots$, respectively. For the AROM, the data $M[d_0], M[d_1], M[d_1], \ldots$ are taken from the output port, q immediately at time $0, 1, 2, \ldots$, respectively.

In current FPGAs, an SROM can be implemented in embedded block RAMs. However, an AROM is implemented in LUTs, which are very costly. Hence, we should use SROMs when we need a function of ROMs. On the other hand, AROM is easy to use, because we can get output data from the AROM immediately.

We will describe a behavior of a circuit element using a sequence of output at every rising clock edge for the *periodic clock* (clock is inverted into a fixed frequency), and *initial reset* (initially, reset is 1 and drops to 0 before the first rising clock edge) as illustrated in Fig. 5. The behavior of each circuit element is described by the output sequences as follows:

- **Combinational Circuit (CC)** For simplicity, we assume 3-input 2-output combinational circuit which is shown in Fig. 3. There is no difficulty to extend the definition for general m-input n-output combinational circuit. We assume that, at time i ($i \geq 0$), a_i, b_i, and c_i are given to the 3 input ports A, B, and C. Let f and g be the two functions with three arguments that determine the value of output ports F and G. The output sequences of F and G are as follows:

$$\text{CC(F)}: \langle f(a_0, b_0, c_0), f(a_1, b_1, c_1), f(a_2, b_2, c_2), \ldots \rangle$$
$$\text{CC(G)}: \langle g(a_0, b_0, c_0), g(a_1, b_1, c_1), g(a_2, b_2, c_2), \ldots \rangle$$

- **Register (R)** Let d_i denote an input value given to an input port d at time i ($i \geq 0$). The output sequence is described as follows:

$$\text{R}: \langle 0, d_0, d_1, d_2, \ldots \rangle$$

- **Synchronous and Asynchronous ROMs (SROMs**

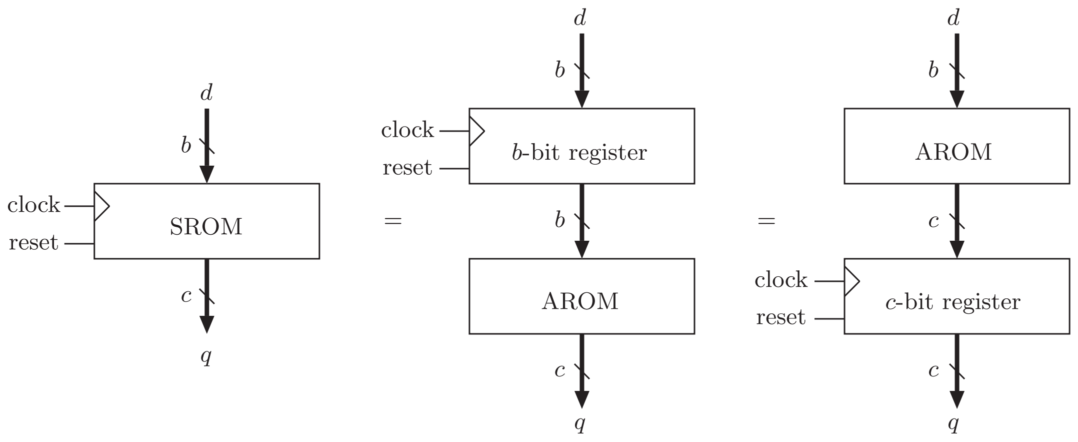

Fig. 7 SROM, R+AROM, and AROM+R.

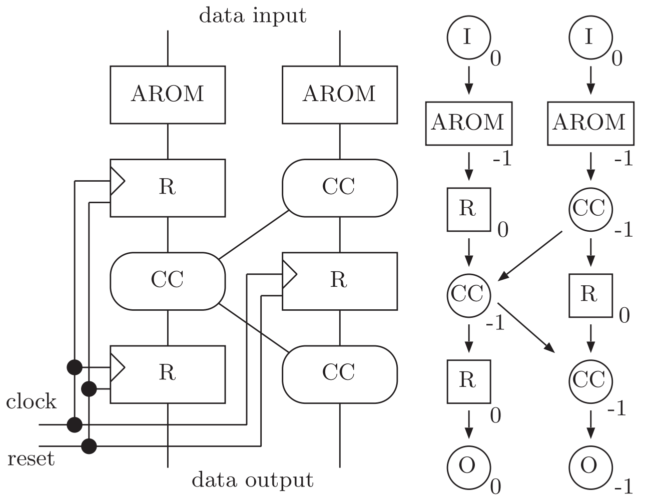

Fig. 6 An example of a fully synchronous circuit and the corresponding circuit graph with potentiality.

and AROMs) Let $M[j]$ denote the value stored in address j ($j \geq 0$) of the ROM. The output sequences of SROM and AROM are as follows:

SROM: $\langle 0, M[d_0], M[d_1], M[d_2], \ldots\rangle$
AROM: $\langle M[d_0], M[d_1], M[d_2], M[d_3], \ldots\rangle$

In this paper, we assume that a fully synchronous circuit has data inputs, data outputs, a global clock input, a global reset input, combinational circuits (CCs), registers (Rs), SROMs, AROMs, and their interconnects. The readers should refer to the Fig. 6 for illustrating an example of a fully synchronous circuit. The global clock and the global reset are directly connected to the clock input ports and the reset input ports of all Rs and SROMs. Also, we assume that a circuit has no loop.

Let us define *equivalence* of two fully synchronous circuits for the periodic clock and initial reset. We say that two circuits X and Y are an *equivalent* if, for any input sequence, the output sequences are the same except for first several outputs. For the reader's benefit, we will show an example of the equivalence.

Let us consider a circuit R+AROM, that is, the output of R is connected to the input of AROM as illustrated in Fig. 7. We also consider a circuit AROM+R, in which the output of AROM and the input of R are connected. For the periodic clock with initial reset, the output sequences of SROM, R+AROM, and AROM+R are as follows:

SROM: $\langle 0, M[d_0], M[d_1], M[d_2], \ldots\rangle$
R+AROM: $\langle M[0], M[d_0], M[d_1], M[d_2], \ldots\rangle$
AROM+R: $\langle 0, M[d_0], M[d_1], M[d_2], \ldots\rangle$

Since these three circuits have the same output in time $1, 2, \ldots$, they are equivalent. Note that the outputs in time 0 are not equal. In this paper, we ignore first several clock cycles when we determine the equivalency of the circuits.

Suppose that a circuit X with AROMs is given. The main contribution of this paper is to show

- a necessary condition such that an AROM-free circuit, Y can be generated, which is equivalent to X, and
- an algorithm to derive Y if the necessary condition is satisfied.

For later reference, we will introduce *a negative register* (NR), which is a nonexistent device used only for showing our algorithm to derive Y and related proofs. Recall that, a regular register latches the input at the rising clock edge. *A negative register* latches a future input. The Fig. 5 also shows a timing diagram of a negative register (NR). An NR latches the value of input d at the rising edge of two clock cycles later as illustrated in Fig. 5. Thus, the NR has the following output sequence for a periodic clock with an initial reset is as follows:

NR: $\langle d_1, d_2, d_3, \ldots\rangle$.

In our algorithm to derive an AROM-free circuit Y, circuits with NRs will be used as interim results.

3. Circuit Graph and Rewriting Rules

We simply use a directed graph to denote the interconnections of a fully synchronous circuit. We call such graph *as a* *circuit graph*. A circuit graph consists of a set of nodes and a set of directed edges connecting two nodes. Each node is labeled by either I (Input port), O (Output port), CC (Combinational Circuit), R (Register), NR (Negative Register), AROM, or SROM. A node with label I is connected with one or more outgoing edges. A node with label O is connected with exactly one incoming edge. A node with label CC has one or more incoming edges and one or more outgoing edges. A node with label R, NR, AROM, or SROM has one incoming and one outgoing edge. We also assume that a circuit graph is a directed acyclic graph (DAG), that is, it has no directed cycles. The Fig. 6 illustrates an example of a directed graph. Note that nodes with label I, R, NR, AROM, or SROM has only one outgoing edge. The readers may think that one outgoing edge is a too stringent restriction because it does not allow two or more fan-outs. However, we can implement multiple fan-outs by attaching a simple combinational circuit (CC) that just duplicates the input. For example, a CC with one input port A and two output ports F and G such that $F = A$ and $G = A$ is used to implement fan-out 2 as illustrated in Fig. 8.

For a given circuit X with AROMs, we will show an algorithm to derive an AROM-free and NR-free circuit, Y by rewriting circuits. We assume that X is given as a circuit graph. We will define rules to rewrite a circuit graph. The readers should refer to Fig. 9 for illustrating the rules, where P and S denote predecessor and successor nodes respectively. The nodes between predecessor and successor nodes are rewritten as follows:

Rule 0 AROM node is rewritten into SROM+NR.

Rule 1 Adjacent R and NR nodes are rewritten into NULL circuit, that is, they are removed.

Rule 2 R+SROM (or NR+SROM) is rewritten into

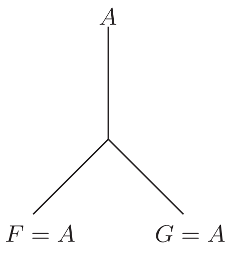

Fig. 8 A combinational circuit to implement fan-out 2 circuit.

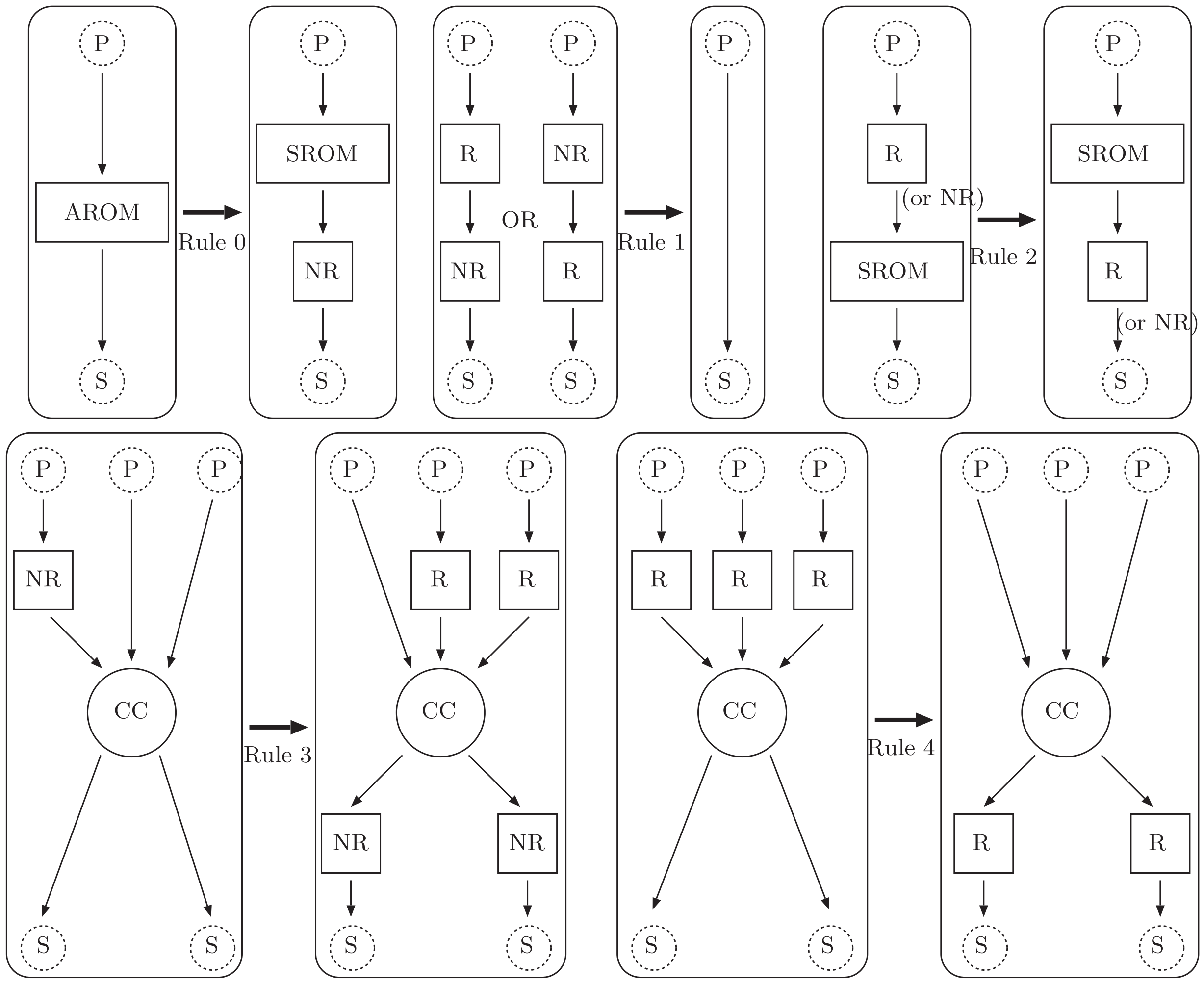

Fig. 9 Rules to rewrite a circuit graph.

SROM+R (or SROM+NR).

Rule 3 If one of the incoming edges of a CC node is connected to an NR node, then the NR node is removed, an R node is added to all the other incoming edges, and the NR node is moved to all the outgoing edges of the CC node.

Rule 4 If all the incoming edges of a CC node are connected to an R node, then all the Rs are removed to all the outgoing edges of the CC node.

Let us confirm that, after applying one of the rewriting rules, an original circuit and the resulting circuit are equivalent. Let a_i, b_i, c_i, and d_i ($i \geq 0$) denote inputs given from the predecessor node at time i.

Rule 0 Both AROM and SROM+NR have the output sequence $\langle M[d_0], M[d_1], M[d_2], M[d_3], \ldots\rangle$, and thus they are equivalent.

Rule 1 R+NR and NR+R have the output sequences $\langle d_0, d_1, d_2, d_3, \ldots\rangle$ and $\langle 0, d_1, d_2, d_3, \ldots\rangle$, respectively. Also, NULL circuit has the output sequence $\langle d_0, d_1, d_2, d_3, \ldots\rangle$. Thus, they are equivalent.

Rule 2 R+SROM and SROM+R have the output sequences $\langle 0, M[0], M[d_0], M[d_1], \ldots\rangle$ and $\langle 0, 0, M[d_0], M[d_1], \ldots\rangle$, respectively and thus they are equivalent. On the other hand, NR+SROM and SROM+NR have the output sequences $\langle 0, M[d_1], M[d_2], M[d_3], \ldots\rangle$ and $\langle M[d_0], M[d_1], M[d_2], M[d_3] \ldots\rangle$, respectively and thus they are equivalent.

Rule 3 The output sequences of the left-hand side of the rule are $\langle f(a_1, b_0, c_0), f(a_2, b_1, c_1), f(a_3, b_2, c_2), \ldots\rangle$ and $\langle g(a_1, b_0, c_0), g(a_2, b_1, c_1), g(a_3, b_2, c_2), \ldots\rangle$. Those of the right-hand side are $\langle f(a_1, b_0, c_0), f(a_2, b_1, c_1), f(a_3, b_2, c_2), \ldots\rangle$ and $\langle g(a_1, b_0, c_0), g(a_2, b_1, c_1), g(a_3, b_2, c_2), \ldots\rangle$. Thus, they are equivalent.

Rule 4 The output sequences of the left-hand side of the rule are $\langle f(0,0,0), f(a_0, b_0, c_0), f(a_1, b_1, c_1), \ldots\rangle$ and $\langle g(0,0,0), g(a_0, b_0, c_0), g(a_1, b_1, c_1), \ldots\rangle$. Those of the right-hand side are $\langle 0, f(a_0, b_0, c_0), f(a_1, b_1, c_1), \ldots\rangle$ and $\langle 0, g(a_0, b_0, c_0), g(a_1, b_1, c_1), \ldots\rangle$. Thus, they are equivalent.

We are now in position to describe the rewriting algorithm. Suppose that an input circuit graph has nodes with labels *I*, *O*, *R*, *AROM*, *SROM*, and *CC*. The following rewriting algorithm generates a circuit graph equivalent to the original circuit graph.

Find a minimum i such that Rule i can be applied to the current circuit graph. Rewrite the circuit graph using such Rule i. This rewriting procedure is repeated until no more rewriting is possible.

The readers should refer to Fig. 10 for illustrating interim and resulting circuit graphs obtained using our rewriting algorithm. In this figure, nodes applied rules are highlighted.

Let us observe the behavior of our rewriting algorithm. First, our rewriting algorithm repeats the applying Rule 0 to all AROM nodes until all AROM nodes are rewritten into SROM+NR. After that, NR nodes are moved toward the output nodes using Rules 2 and 3. Similarly, R nodes are moved toward the output nodes using Rules 2 and 4 whenever possible. Also, adjacent pairs of R and NR are removed by Rule 1. Thus, intuitively, all NR nodes in the resulting circuit graph are moved and placed just before the output nodes.

For the purpose of clarifying the condition such that our rewriting algorithm can generate NR-free circuit graph, we define *the potentiality of the nodes* in a circuit graph. Suppose that a node v of a circuit graph has k (≥ 0) incoming edges such as $(u_1, v), (u_2, v), \ldots, (u_k, v)$. Let us define *the potentiality* $p(v)$ of a node v as follows:

- If v is I, then $p(v) = 0$.
- If v is O or SROM, then $p(v) = p(u_1)$.
- If v is AROM or NR then $p(v) = p(u_1) - 1$.
- If v is R then $p(v) = p(u_1) + 1$.
- If v is CC, then $p(v) = \min(p(u_1), p(u_2), \ldots, p(u_k))$.

The Fig. 6 also shows the potentiality of each node.

We have the following theorem.

Theorem 1: All O nodes of a circuit graph have non-negative potentiality, if and only if our rewriting algorithm generates an AROM-free and NR-free circuit graph, equivalent to the original circuit graph.

In other words, we can determine a fully synchronous circuit that can be converted into an AROM-free circuit by evaluating the potentiality of all O nodes of the corresponding circuit graph. Also, the potentiality of all O nodes are non-negative, our rewriting algorithm generates an AROM-free and NR-free circuit graph, and the corresponding fully synchronous circuit is AROM-free and equivalent to the original fully synchronous circuit. For example, in Fig. 10, the potentiality of the right O node is negative. Hence, the resulting circuit graph has an NR node and our rewriting algorithm fails to remove all NRs.

4. Proof of Theorem 1

The main purpose of this section is to show a proof of Theorem 1. We will show several lemmas for a proof of Theorem 1.

Let us observe how the potentiality of nodes is changed by our rewriting algorithm. We focus the potentiality of successor nodes. Let P and S denote the predecessor and successor nodes for Rules 0, 1, and 2. Also, let P_1, P_2, P_3, and S_1, S_2 be the three predecessor and two successor nodes in Rules 3 and 4. We compute the potentiality of each successor node both before and after applying the rules as follows.

Rule 0 $p(S) = p(P) - 1$.

Rule 1 $p(S) = p(P)$.

Rule 2 $p(S) = p(P) + 1$ if R and $p(S) = p(P) - 1$ if NR.

Rule 3 $p(S_1) = p(S_2) = \min(p(P_1) - 1, p(P_2), p(P_3)) = \min(p(P_1), p(P_2) + 1, p(P_3) + 1) - 1$.

Rule 4 $p(S_1) = p(S_2) = \min(p(P_1) + 1, p(P_2) + 1, p(P_3) + 1) = \min(p(P_1), p(P_2), p(P_3)) + 1$.

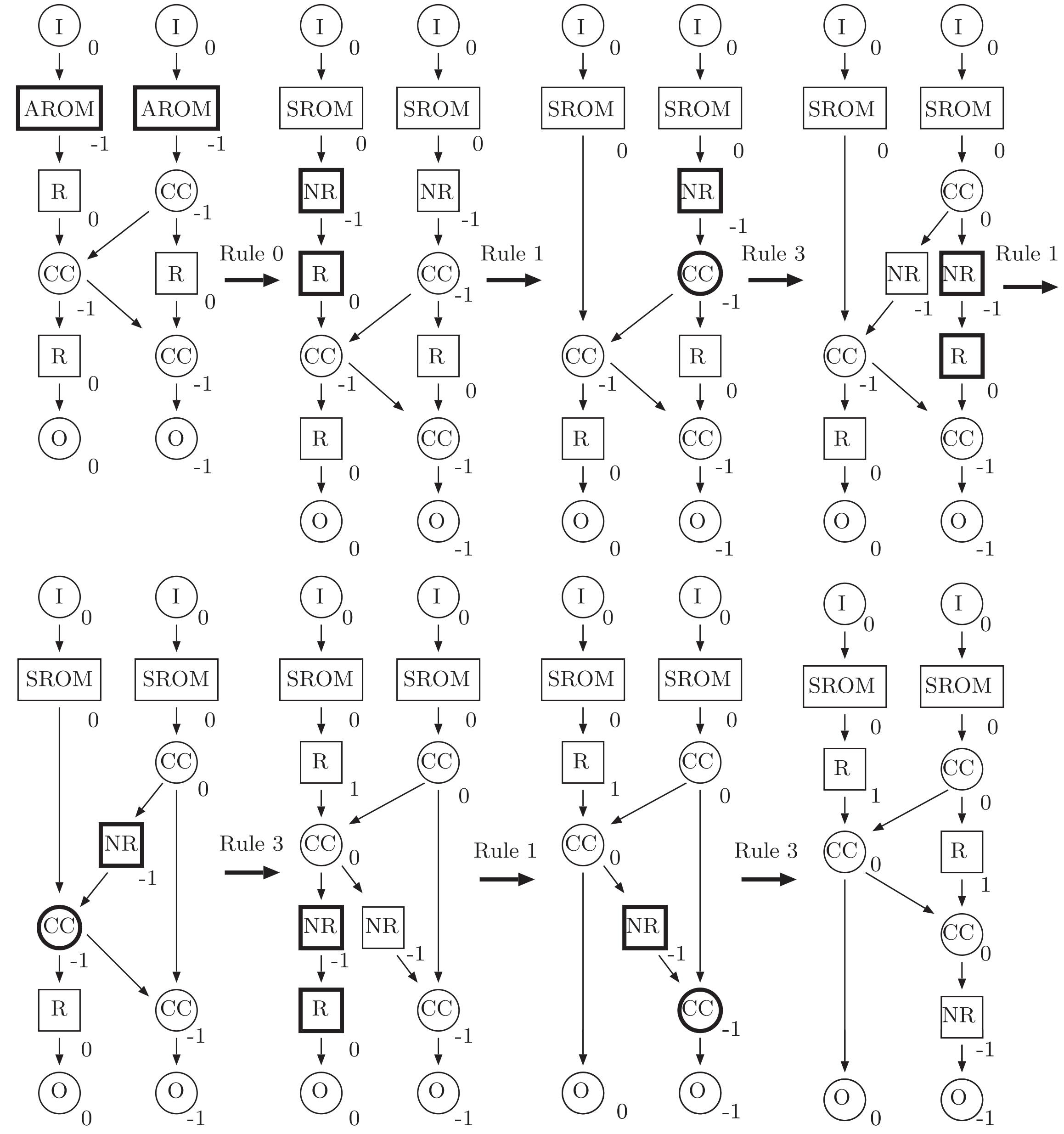

Fig. 10 Interim and resulting circuit graphs obtained by our rewriting algorithm for a circuit graph.

Thus, the potentiality of every successor node is never changed by applying the rules. In every rule, O nodes can only be successor nodes. Thus, we have,

Lemma 2: The potentiality of every O node of the resulting circuit graph is the same as that of the corresponding O node of the original circuit graph.

In Fig. 10, the potentialities of the left and the right O nodes are 0 and -1, respectively, and these values are never changed.

In a circuit graph, let *a segment* be a directed path $u_1, u_2, \ldots, u_k$ such that, u_1 and u_k are either I, O, SROM, or CC, and $u_2, \ldots, u_{k-1}$ are either R or NR. Note that, if $k = 2$ then it represents a null segment with u_1, u_2. We also have the following lemma.

Lemma 3: Let u be an NR node and (u, v) be its outgoing edge in the resulting circuit graph. Node v must be either NR or O node. Also, all NR nodes must be in segments ending at O node.

Proof If v is an R, SROM, or CC node then Rules 1, 2, or 3 can be applied. Since no more rules can be applied to the resulting circuit graph, v must be either NR or O node. Since the successor of NR nodes must be NR or O node, all NR nodes must be in segments ending at O node.

From Lemma 3, we will prove that all SROM and CC nodes in the resulting circuit graph have zero potentiality.

Lemma 4: All SROM and CC nodes in the resulting circuit graph have non-negative potentiality.

Proof Since the resulting graph is AROM-free, nodes follow NR nodes can have negative potentiality. Since no segment ending at SROM or CC has NR nodes, their potentiality must be non-negative.

Similarly, we have the following lemma.

Lemma 5: All SROM and CC nodes in the resulting circuit graph have non-positive potentiality.

Proof We assume that the resulting circuit graph has a SROM or CC node with positive potentiality, and show a contradiction. Let v be a first SROM or CC node with negative potentiality, that is, all SROM and CC nodes in all directed paths incoming to v have non-positive potentiality and SROM or CC node v has positive potentiality.

Case 1 v is an SROM node
Let (u, v) denote the incoming edge. If u is either R or NR, then Rule 2 can be applied. Since no more rules can be applied to the resulting circuit graph, it must be either I, SROM, or CC. If this is the case, $p(u) = 0$ and thus, $p(v) = 0$, a contradiction.

Case 2 v is a CC node
Let $(u_1, v), (u_2, v), \ldots, (u_k, v)$ $(k \geq 1)$ denote the incoming edges. From Lemma 3, none of $u_1, u_2, \ldots, u_k$ is an NR node. If all of them are R nodes, then Rule 4 can be applied. Thus, at least one of them is not an R node. It follows that at least one of them is either I, SROM, or CC node. From the assumption, the potentiality of such node is non-positive, Hence, the potentiality of v is non-positive, a contradiction.

We are now in position to show the proof of Theorem 1. From Lemma 4 and 5, all SROM and CC nodes in the resulting circuit graph have zero potentiality. Hence, if the potentiality of one of the O nodes in the resulting circuit graph is negative, a segment ending at O node in the resulting graph should have NR from Lemma 3. Similarly, if the potentiality of all the O nodes is non-negative, no segment ending at an output node has NR in the resulting circuit graph. From Lemma 2, the potentiality of O nodes does not change by our rewriting algorithm. Thus, all output nodes of a circuit graph have negative potentiality, if and only if our rewriting algorithm generates the resulting circuit graph with NR nodes. This completes the proof of Theorem 1.

From Theorem 1, it is not always possible to have an equivalent AROM-free circuit. However, we may modify a circuit such that it can be converted into an almost equivalent AROM-free circuit. For this purpose, we compute the potentiality of all O nodes in the corresponding circuit graph. After that, we insert registers just before O nodes with negative potentiality so that the potentiality of the corresponding O nodes turns into a zero. Since the potentiality of the corresponding O nodes now is 0, it can be converted into an equivalent AROM-free circuit according to our Theorem 1. The readers should refer to the Fig. 11 for illustrating an example. Note that, the resulting circuit is not equivalent to the original circuit. However, the difference is the latency

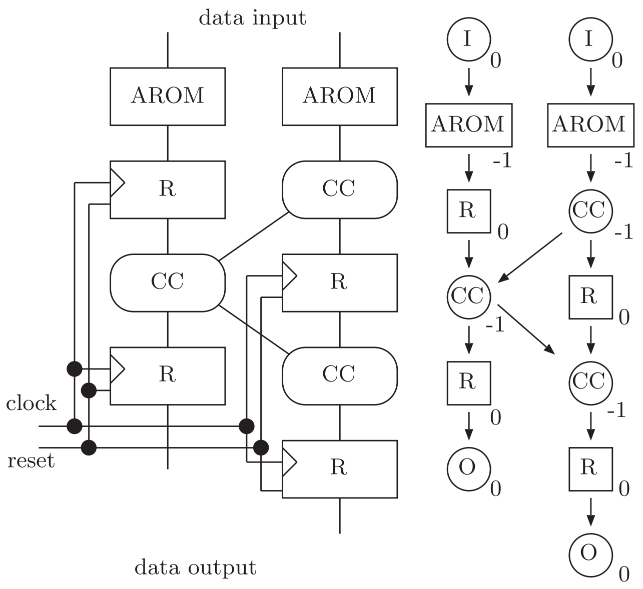

Fig. 11 A circuit almost equivalent to that of Fig. 6 that can be converted into an AROM-free circuit.

of the output. Thus, we can say that the resulting circuit is almost equivalent to the original circuit.

5. Circuit Performance Improvement

Basically, our rewriting algorithm moves registers toward the output ports. Hence, in general, the resulting circuits may have long paths from input ports to registers/SROMs and/or from registers/SROMs to registers/SROMs. Therefore, the resulting AROM-free circuit has large propagation delay and low clock frequency.

In this section, we will describe the ideas to improve the performance of the AROM-free circuit in terms of the latency and the delay, although performance improvement of the AROM-free resulting circuits is the beyond of this paper.

5.1 Minimizing Latency by Eliminating Redundant Registers

We first define *redundant registers* in the resulting AROM-free circuits. The registers which are connected to the edges ending at O nodes are called redundant registers. The redundant registers are highlighted in Fig. 12. Essentially, the redundant registers only work just as buffers for the output nodes. Thus, the latency of each output port can be decreased by eliminating redundant registers if it does not cause a timing problem for a circuit connected to the output port. Also, we can say that, after removing all redundant registers, the latency is minimized, because no other registers can be deleted.

5.2 Increasing Clock Frequency by Adding Registers

We will describe here about the clock performance improve-

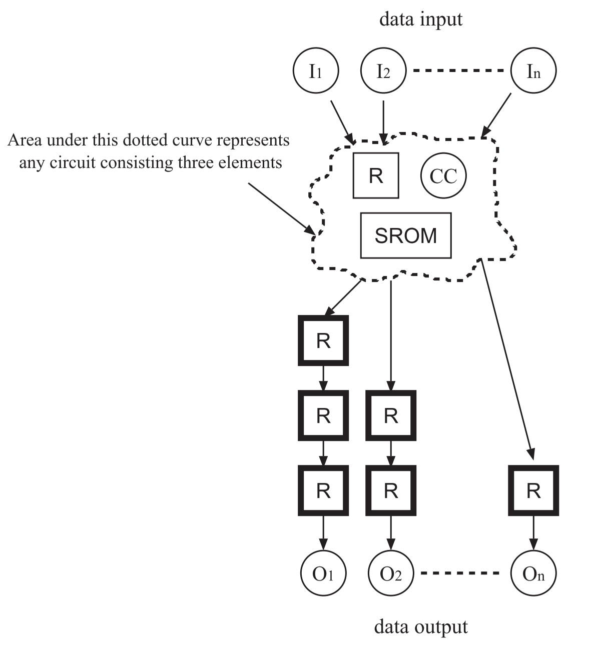
data input
I1
I2
In
Area under this dotted curve represents any circuit consisting three elements
R
CC
SROM
R
R
R
R
R
R
O1
O2
On
data output

Fig. 12 An example to improve the performance in the AROM-free circuit by eliminating redundant registers.

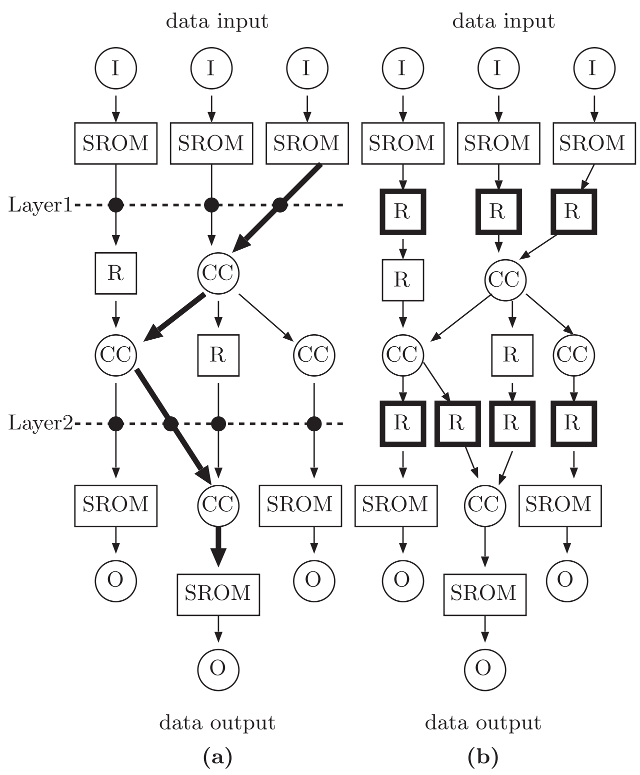

Fig. 13 An example for improving the clock performance in the AROM-free circuit.

ment of the AROM-free resulting circuits. The maximum clock frequency depends on the longest path between input ports to registers/SROMs, registers/SROMs to registers/SROMs, and registers/SROMs to output ports. The longest path is shown as an example in Fig. 13 (a) by highlighting the arrows. Due to the longest path in the clock dependent circuits (i.e. circuits with Rs, CCs and SROMs in our case), the clock performance of those circuits must be degraded. To overcome of this problem, we need to divide the AROM-free resulting circuits (when no rule is applicable) into several layers so that the longest path becomes small. Designers can select the layers properly in order to make the longest path into small for getting optimum clock performance. In fact, the proper selection of the layers in the AROM-free resulting circuits for getting optimum clock performance may be another research work. This topic is beyond of this paper.

Figure 13 (a) shows an example of two layers. The cutting points (created by layers and edges) are highlighted by the bullet circles in Fig. 13 (a). After that, we can add registers at every cutting point in the AROM-free resulting circuits such that longest path in the clock dependent AROM-free resulting circuits becomes small as illustrated in Fig. 13 (b). Due to the small path instead of the longest path, the clock performance of the AROM-free resulting circuits can be improved definitely. In this case, we can ignore the latency of the added registers in the AROM-free resulting circuits. The Fig. 13 represents the clock performance improvement of the AROM-free resulting circuit.

6. Conclusions

In this paper, we have presented a rewriting algorithm and five rewriting rules to convert a sequential circuit with AROMs into an equivalent fully synchronous circuit with no AROMs for the current FPGA. Using our rewriting algorithm, any sequential circuit with AROMs can be converted into an equivalent fully synchronous sequential circuit with no AROMs to support the modern FPGA architecture. It is not trivial to convert the sequential circuits with ARAMs into the equivalent fully synchronous circuits with no ARAMs for supporting the modern FPGA. We are now developing an algorithm for a circuit with ARAM, RAM with asynchronous read operations. As a future work, we have a plan to work on the clock performance improvement in the AROM-free resulting circuits.

References

[1] J. Bordim, Y. Ito, and K. Nakano, "Accelerating the CKY parsing using FPGAs," IEICE Trans. Inf. & Syst., vol.E86-D, no.5, pp.803–810, May 2003.

[2] J. Bordim, Y. Ito, and K. Nakano, "Instance-specific solutions to accelerate the CKY parsing for large context-free grammars," Int. J. Found. Comput. Sci., vol.15, no.2, pp.403–416, 2004.

[3] K. Nakano and Y. Yamagishi, "Hardware *n* choose *k* counters with applications to the partial exhaustive search," IEICE Trans. Inf. & Syst., vol.E88-D, no.7, pp.1350–1359, July 2005.

[4] Y. Ito and K. Nakano, "A hardware-software cooperative approach for the exhaustive verification of the Collatz cojecture," Proc. International Symposium on Parallel and Distributed Processing with Applications, pp.63–70, 2009.

[5] S.S. C, "Design of an FPGA logic element for implementing asynchronous NULL convention logic circuits," IEEE Trans. Very Large

Scale Integr. (VLSI) Syst., vol.15, no.6, pp.672–683, June 2007.
[6] S. Jens, "Asynchronous circuit design, a tutorial." http://webee.technion.ac.il/courses/048878/book.pdf
[7] L. Lavagno and A. Sangiovanni-Vincentelli, Algorithms for Synthesis and Testing of Asynchronous Circuits, Kluwer Academic, 1993.
[8] R. Manohar, "Reconfigurable asynchronous logic," Proc. IEEE Custom Integated Circuits Conference, pp.13–20, 2006.
[9] I. Shota, K. Yoshiya, H. Masanori, and K. Michitaka, "An asynchronous field-programmable VLSI using LEDR/4-phase-dual-rail protocol converters," The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), Monte Carlo Resort, Las Vegas, Nevada, USA, July 2009.
[10] J. Teifel and R. Manohar, "An asynchronous dataflow FPGA architecture," IEEE Trans. Comput., vol.53, no.11, pp.1376–1392, 2004.
[11] H. Scott, B. Steven, B. Gaetano, and E. Carl, "An FPGA for implementing asynchronous circuits," IEEE Des. Test Comput., vol.11, no.3, pp.60–69, 1994.
[12] N. Huot, H. Dubreuil, L. Fesquet, and M. Renaudin, "FPGA architecture for multi-style asynchronous logic," Design, Automation and Test in Europe Conference and Exhibition, pp.32–33, 2005.
[13] J. Sparso and S. Furber, Principles of Asynchronous Circuit Design, Kluwer Academic Publishers, Boston, 2001.
[14] K. Maheswaran, Implementing self-timed circuits in field programmable gate arrays, Master's Thesis, 1995.
[15] B. Gao, A globally asynchronous locally synchronous configurable array architecture for algorithm embeddings, PhD thesis, Dec. 1996.
[16] R. Payne, Self-timed field programmable gate array architectures, PhD thesis, 1997.
[17] J. Teifel and R. Manohar, "Programmable asynchronous pipeline arrays," Proc. 13th Int. Conf. on Field Programmable Logic and Applications, pp.345–354, Lisbon, Portugal, Sept. 2003.

Md. Nazrul Islam Mondal received the BE from the Department of Electrical & Electronic Engineering, RUET, Bangladesh in 2000 and the ME degree from the Department of Information & Communication Technologies, Asian Institute of Technology, Thailand in 2008. Presently, he is doing research as a PhD student under the Prof. Koji Nakano in the Department of Information Engineering, Hiroshima University, Japan. In 2000, he was a Research Assistant at Research and Development Department of Micro Institute of Technology, Dhaka, Bangladesh. In 2001, he moved to the Department of Computer Science and Engineering, RUET, Bangladesh as a lecturer. He has been worked as an assistant professor at the same Department, RUET, Bangladesh from 2004. In 2001–2006 and 2008–2009, he was a part-time lecture of a famous private University (Ahsanullah University of Science & Technology), Bangladesh. He has published his contributions in journal, conference proceedings. He is a member of Institution of Engineers, Bangladesh. He has organized some conferences and workshops. He performs research in the areas of algorithms for FPGA-based re-configurable architectures, image processing, Sensor Networks, Digital Signal Processing.

Koji Nakano received the BE, ME and Ph.D degrees from Department of Computer Science, Osaka University, Japan in 1987, 1989, and 1992 respectively. In 1992–1995, he was a Research Scientist at Advanced Research Laboratory. Hitachi Ltd. In 1995, he joined Department of Electrical and Computer Engineering, Nagoya Institute of Technology. In 2001, he moved to School of Information Science, Japan Advanced Institute of Science and Technology, where he was an associate professor. He has been a full professor at School of Engineering, Hiroshima University from 2003. He has published extensively in journals, conference proceedings, and book chapters. He served on the editorial board of journals including IEEE Transactions on Parallel and Distributed Systems, IEICE Transactions on Information and Systems, and International Journal of Foundations on Computer Science. He has also guest-edited several special issues including IEEE TPDS Special issue on Wireless Networks and Mobile Computing, IJFCS special issue on Graph Algorithms and Applications, and IEICE Transactions special issue on Foundations of Computer Science. He has organized conferences and workshops including International Conference on Parallel and Distributed Computing, Applications and Technologies, IPDPS Workshop on Advances in Parallel and Distributed Computational Models, and ICPP Workshop on Wireless Networks and Mobile Computing. His research interests includes image processing, hardware algorithms, FPGA-based reconfigurable computing, parallel computing, mobile computing, algorithms and architectures.

Yasuaki Ito received B.E. degree from Nagoya Institute of Technology (Japan), M.S. degree from Japan Advanced Institute of Science and Technology in 2003, and D.E. degree from Hiroshima University (Japan), in 2010. From 2004 to 2007 he was a Research Associate at Hiroshima University. Since 2007, Dr. Ito has been with the School of Engineering, at Hiroshima University, where he is working as an Assistant Professor. His research interests include reconfigurable architectures, parallel computing, computational complexity and image processing.