A Rewriting Algorithm to Generate AROM-free Fully Synchronous Circuits

Md. Nazrul Islam Mondal, Koji Nakano, and Yasuaki Ito
Department of Information Engineering, Hiroshima University
1-4-1 Kagamiyama, Higashi-Hiroshima, 739-8527, Japan

***Abstract*—A Field Programmable Gate Array (FPGA) is used to embed a circuit designed by users instantly. FPGAs can be used for implementing hardware algorithms. 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 RAMs and ROMs in modern FPGAs support synchronous read operations, but do not support asynchronous read operations. It is one of the main difficulties for users to implement hardware algorithms using RAMs and ROMs with synchronous read operations. The main contribution of this paper is to provide one of the potent methods to resolve this problem. We assume that a circuit using asynchronous ROMs designed by a user is given. Our goal is to convert this circuit into an equivalent circuit with synchronous ROMs. We first clarify the condition that a given circuit with asynchronous ROMs can be converted into a circuit without asynchronous ROMs. For this purpose, we will show an algorithm that can generate a circuit with synchronous ROMs, whenever the original circuit with asynchronous ROMs satisfies this condition. Using our conversion algorithm, users can assume that FPGAs support asynchronous ROMs when they design their circuits. Finally, we will show that we can generate an almost equivalent circuit with synchronous ROMs by modifying the circuit even if it does not satisfy this condition.**

***Index Terms*—FPGA, Block RAMs, Asynchronous read operations, Rewriting algorithm.**

I. 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 (CC) 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 the synchronous read and synchronous write operations. The read and/or 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], [2], [11] to accelerate the computation in FPGAs. In particular, FPGAs can implement hundreds of circuits that work in parallel, they are used to accelerate useful computations. For example, a parallel implementation [6] for the exhaustive verification of the Collatz conjecture has been presented. In this implementation, 24 co-processors embedded in a Xilinx Virtex-2 Family FPGA perform the exhaustive verification in parallel.

The circuit design is easier if asynchronous read operation is possible. Let us show a practical example. Suppose that two input sequences $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ are given to the two input ports. We need to design a circuit to compute $(f(x_i) + g(y_i)) \cdot (f(x_i) - g(y_i))$ for every i $(i \geq 1)$. Figure 1 illustrates an example of the circuit for this task. Two ROMs supporting asynchronous read operation (AROMs) are used to compute $f(k)$ and $g(k)$. More specifically, they are storing the values of $f(k)$ and $g(k)$ in address k. Since the read operation of these ROMs are asynchronous, the values of $f(k)$ and $g(k)$ are output immediately for input k. Using an adder, a subtractor, and a multiplier, $(f(x_i)+g(y_i)) \cdot (f(x_i) - g(y_i))$ is computed for the input x_i and y_i. The resulting value is stored in a register. In this way, the circuit can be designed using AROMs easily. However, it is not always easy to design circuits using ROMs supporting synchronous read operations (SROMs). The main contribution of this paper is to present an algorithm to convert a circuit with AROMs into an equivalent circuit with SROMs. (i.e. an AROM-free circuit) automatically.

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.
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 continue 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

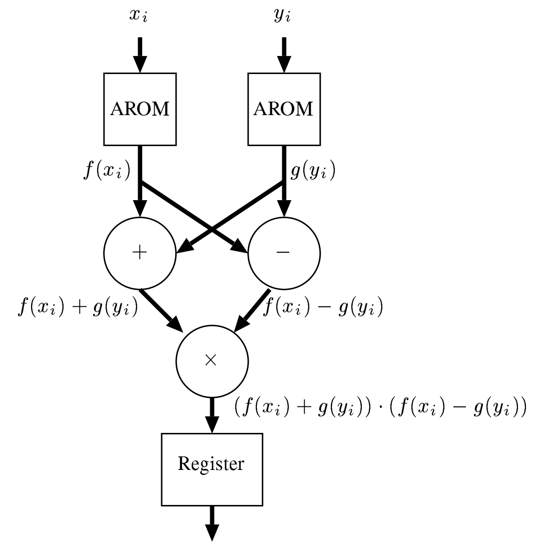

Fig. 1. A circuit to compute $(f(x_i) + g(y_i)) \cdot (f(x_i) - g(y_i))$ using Asynchronous ROMs (AROMs).

size so that it does not support the designer's demand, more expensive, and less speedy [3], [7], [8]. To cut the clock distribution power, an asynchronous circuit design in FPGAs is very much suitable, described in [10], [14], [17]. But 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 [3], [8]. 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 [13]. 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 [5]. 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 [15]. 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 [13] and PGA-STC [9] are based on an asynchronous design, GALSA [4] and STACC [12] are globally asynchronous FPGAs but locally synchronous and PAPA [16] 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 current FPGA architecture is the best suited for digital synchronous circuit design. 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. So, our target is to generate an AROM-free fully synchronous sequential circuit from a sequential circuit with AROM which is an equivalent to the original circuit so that it can support the modern FPGA architecture.

This paper is organized as follows: Section II briefly describes the circuits and their equivalency. In Section III, we describe our rewriting algorithm, circuit graph and also explain the equivalency for our rewriting rules. Section IV presents the proof of the correctness of our rewriting algorithm. Finally Section V concludes this work and also describes the future works.

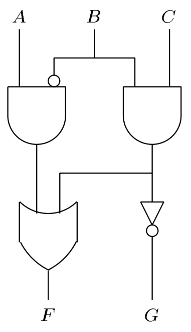

Fig. 2. An example of a combinational circuit (CC).

II. 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 Figure 2. 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 Figure 3. 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 b-bit input d and c-bit data output q. It is storing 2^b words such as $M[0]$, $M[1]$, ..., $M[2^b - 1]$ 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

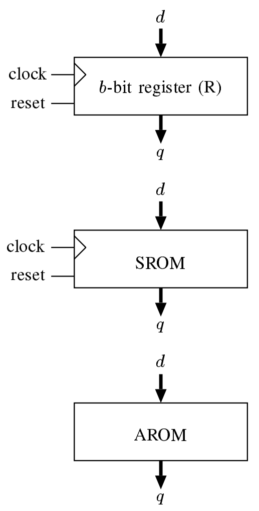

Fig. 3. A register (R), a synchronous ROM (SROM) and an asynchronous ROM (AROM).

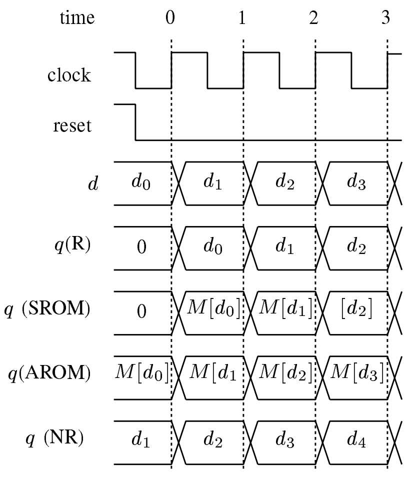

Fig. 4. A timing chart of a register(R), an SROM, an AROM and a negative register (NR).

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 Figure 4 shows a timing diagram of reading operations of the R, SROM, AROM and NR. In the figure, time 0, 1, 2, ... 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, ..., 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, ..., the values of output,q of R and SROM are d_0, d_1, d_2, ..., and $M[d_0]$, $M[d_1]$, $M[d_2]$, ..., respectively. For the AROM, the data $M[d_0]$, $M[d_1]$, $M[d_1]$, ... are taken from the output port, q immediately at time 0, 1, 2, ... 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 *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 Figure 4. 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 Figure 2. It should not have 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:

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

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

- **Synchronous and Asynchronous ROMs (SROMs 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 input, data output, a global clock input, a global reset input, combinational circuits (CCs), registers (Rs), SROMs, AROMs, and their interconnects. The readers should refer to the Figure 5 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 *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 Figure 6. 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

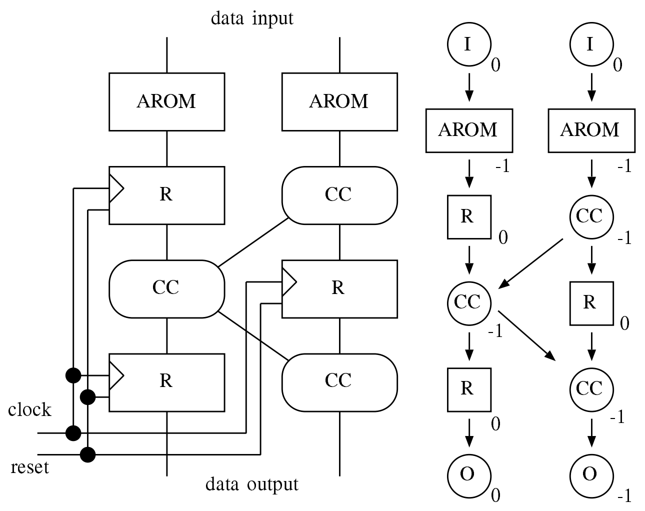

Fig. 5. An example of a fully synchronous circuit and the corresponding circuit graph with potentiality.

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, ..., 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 generates, 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 Figure 4 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 Figure 4. 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 NR will be used as interim results.

III. 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

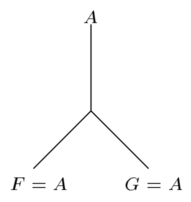

Fig. 7. A combinational circuit to implement fan-out 2 circuit.

(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 Figure 5 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 strong 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 duplicate 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 Figure 7.

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 Figure 8 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 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, a 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 a 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.

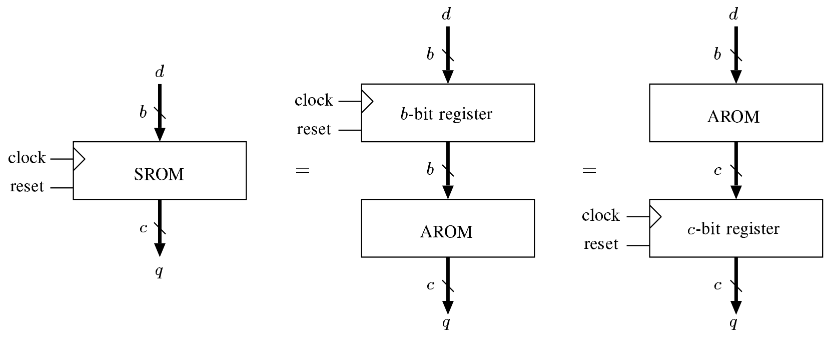

Fig. 6. SROM, R+AROM, and AROM+R.

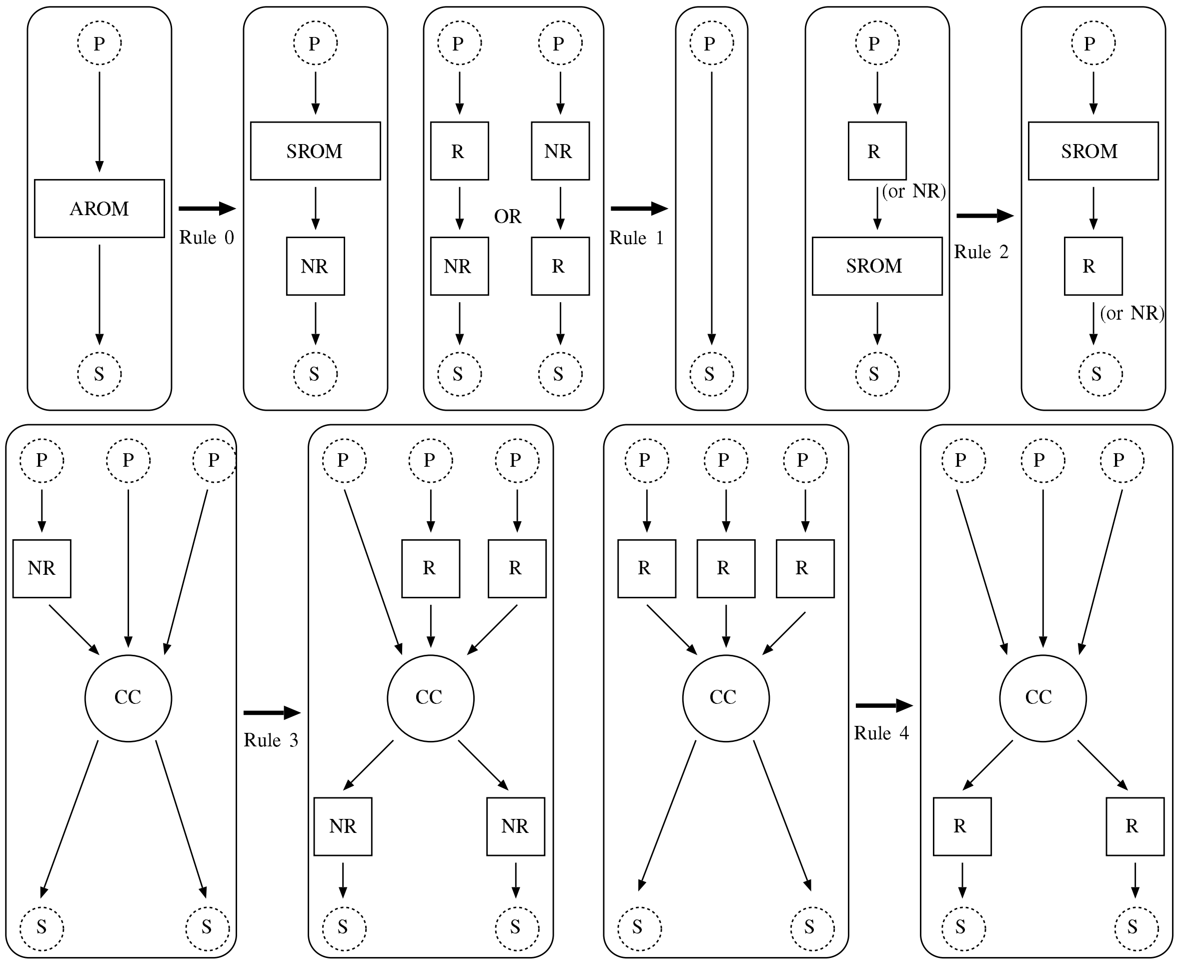

Fig. 8. Rules to write a circuit graph.

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 Figure 9 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 ($\geq$ 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 Figure 5 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 Figure 9, 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.

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

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 Figure 9, 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, *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 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 rule can be applied to the resulting circuit graph, v must be either NR or O nodes. Since the successor of NR nodes must be NR or O nodes, 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.

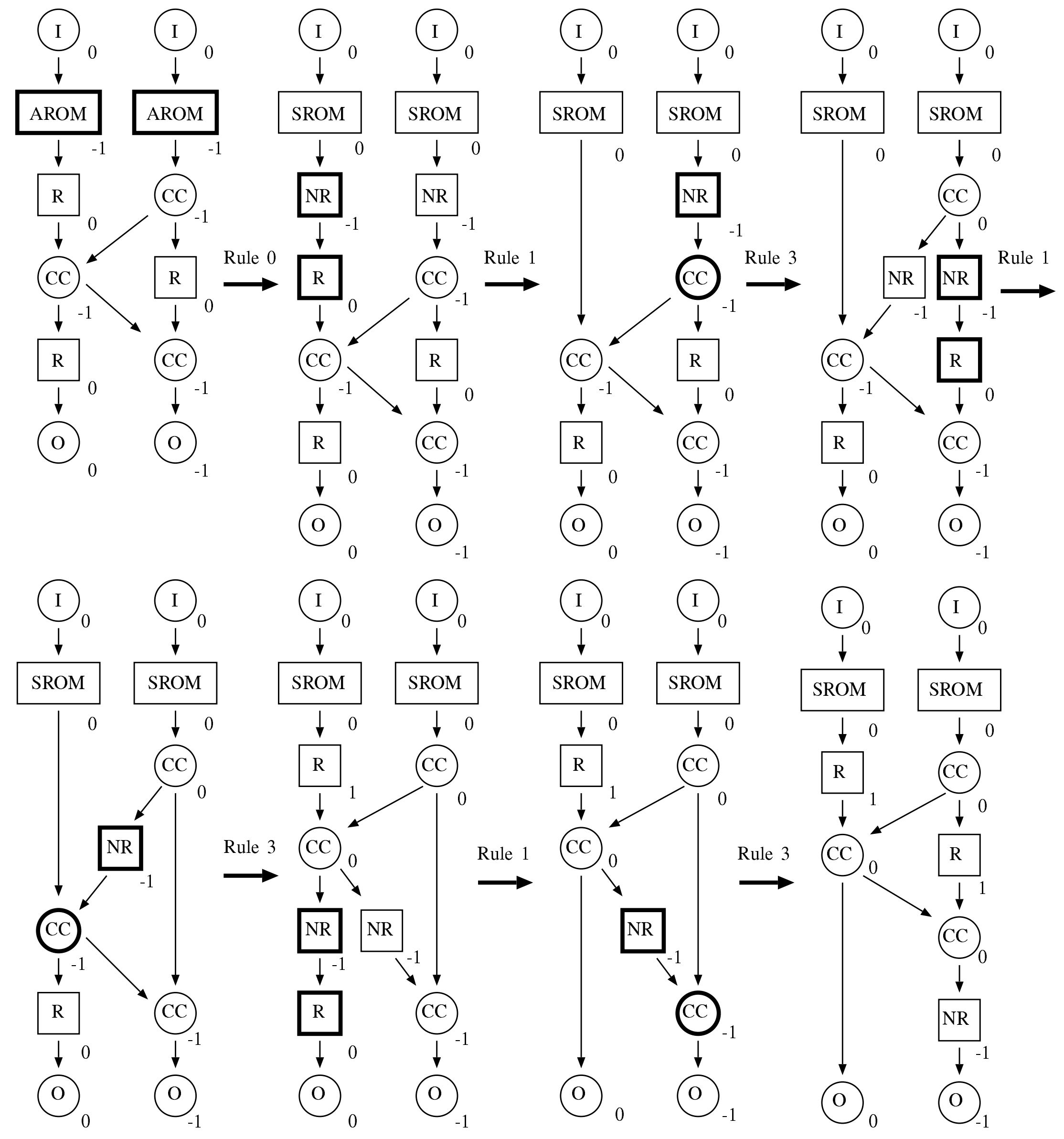

Fig. 9. Interim and resulting circuit graphs obtained by our rewriting algorithm for a circuit graph.

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 rule 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 CC

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.

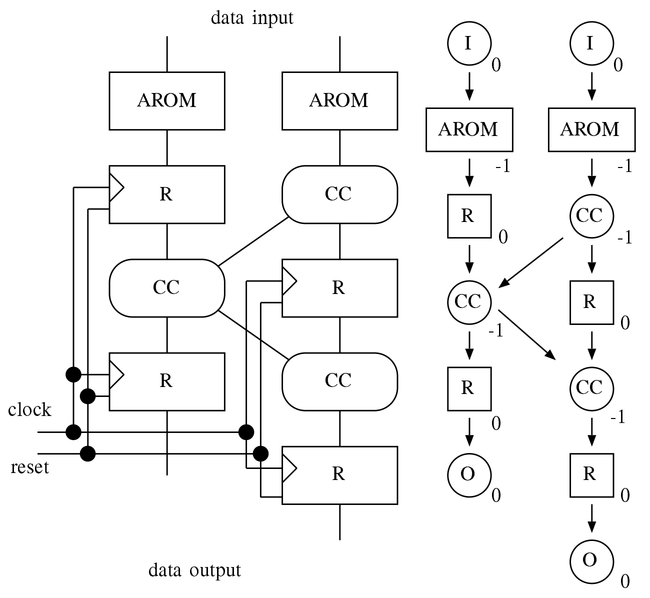

Fig. 10. A circuit almost equivalent to that of Figure 5 that can be converted into an AROM-free circuit.

■

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, 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 possible to have an equivalent AROM-free circuit. However, we can modify a circuit such that it can be converted into an 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 Figure 10 for illustrating an example.

V. Conclusions

In this paper, we have presented a rewriting algorithm and five rewriting rules to convert a sequential circuit with AROM into an equivalent fully synchronous circuit with no AROM for the current FPGA. Using our rewriting algorithm, any sequential circuit with AROM can be converted into an equivalent fully synchronous sequential circuit with no AROM to support the modern FPGA architecture. As a future work, we are now developing algorithm for circuits with feedback loops. We also have a plan to present algorithm for a circuit with ARAM, RAM with asynchronous read operations.

References

[1] J. Bordim, Y. Ito, and K. Nakano. Accelerating the CKY parsing using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):803–810, 2003.

[2] J. Bordim, Y. Ito, and K. Nakano. Instant-specific solutions to accelerate the CKY parsing for large context-free grammars. *Internation Journal on Foundations of Computer Science*, pages 403–416, 2004.

[3] S. S. C. Design of an fpga logic element for implementing asynchronous null convention logic circuits. *IEEE Transactions on very large scale integration (VLSI) system*, 15(6), June 2007.

[4] B. Gao. A globally asynchronous locally synchronous configurable array architecture for algorithm embeddings. PhD thesis, December 1996.

[5] N. Huot, H. Dubreuil, L. Fesquet, and M. Renaudin. Fpga architecture for multi-style asynchronous logic. In *Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 32–33, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[6] Y. Ito and K. Nakano. A hardware-software cooperative approach for the exhaustive verification of the collatz cojecture. In *Proc. of International Symposium on Parallel and Distributed Processing with Applications*, pages 63–70, 2009.

[7] S. Jens. Asynchronous circuit design, a tutorial. http://webee.technion.ac.il/courses/048878/book.pdf.

[8] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for synthesis and testing of asynchronous circuits*. Kluwer Academic, 1993.

[9] K. Maheswaran. Implementing self-timed circuits in field programmable gate arrays. Master's Thesis, 1995.

[10] R. Manohar. Reconfigurable asynchronous logic. In *Proceedings of IEEE Custom Intregated Circuits Conference*, pages 13–20, 2006.

[11] K. Nakano and Y. Yamagishi. Hardware n choose k counters with applications to the partial exhaustive search. *IEICE Transactions on Information and Systems*, 2005.

[12] R. Payne. Self-timed field programmable gate array architectures. PhD thesis, 1997.

[13] H. Scott, B. Steven, B. Gaetano, and E. Carl. An fpga for implementing asynchronous circuits. *IEEE Design and Test of Computers*, 11(3):60–69, 1994.

[14] I. Shota, K. Yoshiya, H. Masanori, and K. Michitaka. An asynchronous field-programmable vlsi using ledr/4-phase-dual-rail protocol converters. In *The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Monte Carlo Resort, Las Vegas, Nevada, USA, July 2009.

[15] J. Sparso and S. Furber. *Principles of Asynchronous Circuit Design*. Kluwer Academic Publishers, Boston, 2001.

[16] J. Teifel and R. Manohar. Programmable asynchronous pipeline arrays. In *Proc. of the 13th Int. Conf. on Field Programmable Logic and Applications*, pages 345–354, Lisbon, Portugal, September 2003.

[17] J. Teifel and R. Manohar. An asynchronous dataflow fpga architecture. *IEEE Transaction on Computers*, 53(11):1376–1392, 2004.