An Algorithm to Obtain Circuits with Synchronous RAMs

Md. Nazrul Islam Mondal, Koji Nakano, Yasuaki Ito

Department of Information Engineering, Hiroshima University, 1-4-1 Kagamiyama, Higashi-Hiroshima 739-8527, Japan

Abstract: Field Programmable Gate Arrays (FPGAs) are used to embed a circuit designed by users instantly. 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. It is one of the main difficulties for users to implement parallel and hardware algorithms in FPGAs. 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 RAMs designed by a non-expert or quickly designed by an expert is given. Our goal is to convert this circuit with asynchronous RAMs into an equivalent circuit with synchronous ones. The resulting circuit with synchronous RAMs can be embedded into the FPGAs.

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 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. In our work, we consider single-port RAM which can be embedded into a dual-port block RAM of the current FPGA. Usually, the dual-port RAM supports synchronous read and synchronous write operations. The read and write operations are performed at the rising clock edges. Design tools are available to the users to embed their hardware logic into the FPGAs. Some circuit implementations are described in the papers [1-4] to accelerate computation.

In this paper, we focus on the asynchronous read, synchronous read and synchronous write 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[a], which is the data stored in the input address value a.

Corresponding author: Md. Nazrul Islam Mondal (2009-), Ph.D student, research fields: algorithm for FPGA-based re-configurable architectures, image processing, email: nimbd@yahoo.com.

Koji Nakano (2003-), Ph.D, professor, research fields: hardware algorithms, FPGA-based reconfigurable computing, parallel and mobile computing, algorithms and architectures, e-mail: nakano@cs.hiroshima-u.ac.jp

Yasuaki Ito (2007-), Ph.D, assistant professor, research fields: reconfigurable architecture, parallel computing, computational complexity and image processing, e-mail: yasuaki@cs.hiroshima-u.ac.jp

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 the clock. More specifically, the output data port outputs M[*a*] on the rising edge of the clock, where *a* is the address data at the previous point of the rising clock edge.

Synchronous write operation The memory block stores the input data, given to the data port on the rising edge of the clock only when write enable *we* is high. Even though, the input data value is changed and the rising edge of the clock is available, nevertheless write enable *we* is low, the input data value is not written into the memory block. Particularly, the input data value *d* is only written into the memory of M[*a*] on the rising edge of the clock when write enable *we* is high, where *a, d* and *we* represent address data value, input data value and write enable respectively at the previous point of the rising clock edge.

In this work, we consider asynchronous RAMs (ARAMs) and synchronous RAMs (SRAMs). They have a data input port *D*, an address input port *A*, clock input port *clock*, write enable input port *we* and data output port *Q* , shown in Fig 1. ARAMs and SRAMs support asynchronous and synchronous read operations respectively. Also they both support synchronous write operations. In general, the circuit design is simpler and easier to the designers, more specifically to the non-expert circuit designers if ARAMs 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 clock frequency.

In our previous paper [5], we have presented a circuit rewriting approach for Directed Acyclic Graph (DAG) circuits considering only read operations of the memory blocks (ROMs). However, this paper considers both read and write operations of the memory blocks (RAMs). Note that, a RAM can be treated as a ROM when *we* is low. It is not trivial to convert an asynchronous circuit with ARAMs into an equivalent synchronous one. However, it is solved by our technique.

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 RAMs with asynchronous read and synchronous write operations (ARAMs)

into an *equivalent synchronous circuit* consisting

> Combinational Circuits (CCs), Registers (Rs), and RAMs with synchronous read and synchronous write operations (SRAMs).

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 the circuit rewriting algorithm.

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

The outlines of our work are as follows:

1. We use *a Negative Register* (NR) which is originally introduced in our previous paper [5]. The NR is an imaginary register that is used for latching a future input data.
2. We define simple *five rules* that rewrite a circuit.
3. The rewriting algorithm just repeats applying these rules until no more rules can be applied. When the rewriting algorithm terminates, we have an equivalent ARAM-free circuit to the original circuit.

2. Random Access Memory (RAM)

A RAM is an array of memory where information can be stored until power is switched off. It has b-bit data input *D*, e-bit address input *A* and c-bit data output *Q*, it can store 2^e words such as M[0], M[1], …., M[2^e-1] with c bits each, shown in Fig. 1. A RAM can support the following operations.

Asynchronous read operation:

A RAM continuously outputs the data specified by the address given to the address port *A*. When the address value *a* is changed, the output data is updated immediately within some delay time. In other words, the output data port always outputs M[*a*], which is the data stored in the input address value *a*.

Synchronous read operation:

Even if the input address value *a* is changed, the output data specified by the address given to the address port *A* is not updated. The output data is updated based on the address value *a* at the rising edge of the clock. More specifically, the output data port *Q* outputs M[*a*] on the rising edge of the clock, where *a* is the address data at the previous point of the rising clock edge.

Synchronous write operation:

A RAM stores the input data value *d* which is given to the data port *D* on the rising edge of the clock only when write enable *we* is high. Even though, the input data value *d* is changed and the rising edge of the clock is available, nevertheless write enable *we* is low, the input data value *d* is not written into the memory of M[*a*]. Particularly, the input data value *d* is only written into the memory of M[*a*] on the rising clock edge when write enable *we* is high, where *a*, *d* and *we* represent address data value, input data value and write enable respectively at the previous point of the rising clock edge.

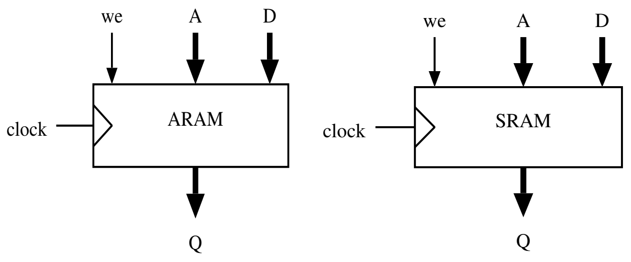

Fig. 1 An asynchronous RAM (ARAM) and a synchronous RAM (SRAM).

For the reader's benefit, we will describe two types of RAM (shown in Fig. 1) as follows:

- **Asynchronous RAM (ARAM):** An ARAM supports asynchronous read and synchronous write operations. It has a clock input *clock* and a write enable input *we*. The clock input *clock* is only needed for write operations. The data values of M[*a*] are continuously output from port *Q*. They do not depend on clock input *clock*. Only when write enable *we* is high, initial stored values of M[*a*] are updated by input data value *d,* given to the data input port *D* at the latest rising clock edge.
- **Synchronous RAM (SRAM):** An SRAM supports synchronous read and synchronous write operations. It has also a clock input *clock* and a write enable input *we*. The read operation of the SRAM is performed on every rising clock edge. The output *Q* is the value of M[*a*] at the latest rising clock edge. The write operations for an SRAM are the same as an ARAM. The readers may refer to the Fig. 2 for read and write operations of an ARAM and an SRAM.

In this work, we consider Write After Read (WAR) mode for data handling of the memory blocks. Now, we will discuss the WAR and RAW mode only for the SRAM, because the SRAM supports synchronous read and synchronous write operations. The WAR and RAW modes of an SRAM are described as follows:

- **Write After Read (WAR) Mode:** First, currently stored data, specified by the address given to the address port *A* outputs from the output port *Q* at the latest rising clock edge. Then input data value *d,* given to the data port *D* is written into the memory of M[*a*] at the latest rising clock edge only when write enable *we* is high. More specifically, the output data port *Q* outputs currently stored data of M[*a*] on the latest rising clock edge first. Then the input data value *d* is written into the memory of M[*a*] on the latest rising clock edge only when write enable *we* is high.

- **Read After Write (RAW) Mode:** First, input data value *d,* given to the data port *D* is written into the memory of M[*a*] at the latest rising clock edge only when write enable *we* is high. Then currently stored data, specified by the address given to the address port *A* outputs from the output port *Q* at the latest rising clock edge. Particularly, input data value *d* is written first into the memory of M[*a*] on the latest rising clock edge only when write enable *we* is high. Then stored data value of M[*a*] outputs from the output port *Q* at the latest rising clock edge.

The readers should refer to the Fig. 2 for the illustrations of the WAR and RAW. The Fig. 2 shows a timing diagram of the ARAM, SRAM, Register (R) and Negative Register (NR). Initially global reset is 1 and it drops to 0 just before time 0. We assume that write enable *we* is high for the first several clock cycles from the beginning (initially *we* is 1 and drops to 0 before the fourth rising clock edge). Data 11, 12, 13, -, -, - and 1, 2, 3, 1, 2, 3 are given to the input data port *D* and address port *A* respectively. The dash (-) line represents any data which is not necessary in our case. For simplicity, we have used the written data values of M[1], M[2], M[3] for an SRAM to read again at the latest rising clock edge at time 3, 4, 5 respectively. We have also used the written data values of M[1], M[2], M[3] for an ARAM to read immediately at time 3, 4, 5 respectively. The edges at time 0, 1, 2 of the clock represent the latest rising edges for the stored data values of M[1], M[2], M[3] respectively of an SRAM and ARAM. On the other hand, the edges at time 0, 1, 2, 3, 4 of the clock represent the latest rising edges for the output data sequence of 0, 0, 0, 11, 12 respectively of an SRAM when it follows WAR. We assume that the stored values of M[*a*] are initialized by 0 of an SRAM and an ARAM. For the case of an ARAM, the data values of M[1], M[2], M[3], M[1], M[2], M[3] correspond to 0, 0, 0, 11, 12, 13 are taken respectively at time 0, 1, 2, 3, 4, 5 from the output port *Q* immediately due to the asynchronous read operations. Therefore, the output sequence of an ARAM [Q (ARAM)] is: 0, 0, 0, 11, 12, 13. According to the WAR, the output sequence of an SRAM [Q (SRAM, WAR)] is: 0, 0, 0, 0, 11, 12 at time 0, 1, 2, 3, 4, 5 respectively. On the other hand, according to the RAW, the output sequence of an SRAM [Q (SRAM, RAW)] is: 0, 11, 12, 13, 11, 12 at time 0, 1, 2, 3, 4, 5 respectively. Note that the output value of an SRAM at time 0 is initialized by 0. The stored data of M[1], M[2] and M[3] of the time 0, 1, 2, 3, 4, 5 are the same for an ARAM and an SRAM which are M[1]: 0, 11, 11, 11, 11, 11; M[2]: 0, 0, 12, 12, 12, 12 and M[3]: 0, 0, 0, 13, 13, 13. The output of R is 0 at time 0. Also at time 1, 2, 3, 4, 5; the value of output R is 11, 12, 13, -, - respectively. The value of output NR is 12, 13, - ,-, -, - of the time 0, 1, 2, 3, 4, 5 respectively.

3. Circuits and Their Equivalence

Let us consider a synchronous sequential circuit that consists of input ports, output ports, combinational circuits (CCs), registers (Rs), Random Access Memories (RAMs), a global clock input (clock), a global reset input (reset) and a write enable input (*we*).

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 our previous paper [5]. 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. 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. The readers should refer to our previous paper [5] and Section 2 for the details about combinational circuit, register and RAM.

We will describe a behavior of the circuit elements

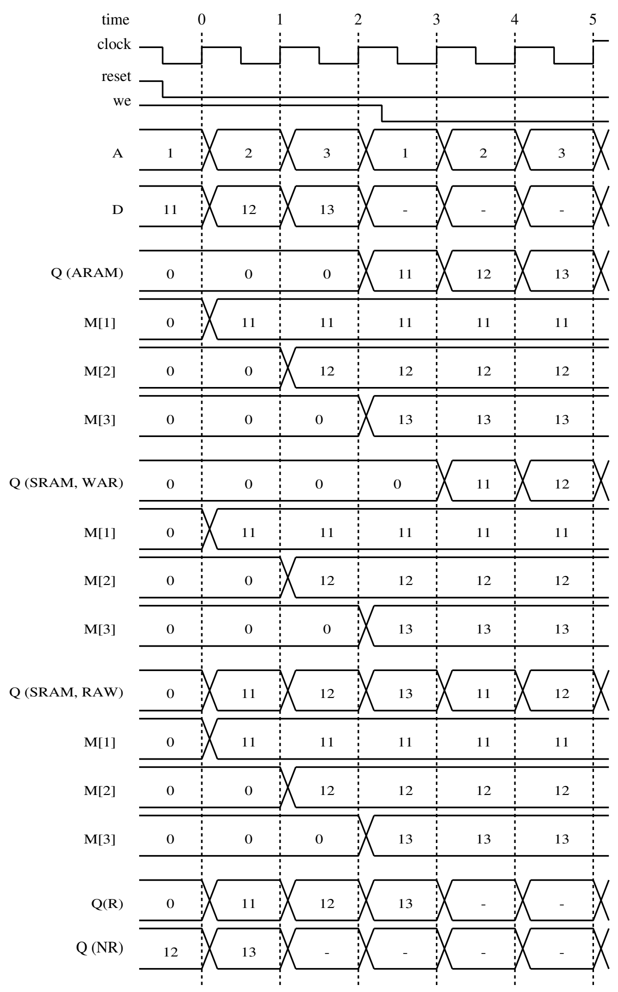

Fig. 2 A timing chart of an ARAM, an SRAM, a register (R) and a negative register (NR).

using a sequence of output as well as stored data at every rising clock edge for *periodic clock* (clock is inverted into a fixed frequency), *initial reset* (initially, reset is 1 and drops to 0 before the first rising clock edge) and write enable, *we* (initially, *we* is 1 and drops to 0 before the fourth rising clock edge) as illustrated in Fig. 2. The behavior of each circuit element is described by the output sequences and stored data as follows:

- **Combinational Circuit (CC)** For simplicity, we assume 3-input 2-output combinational circuit, shown in our previous paper [5]. There is no difficulty to extend the definition for general m-input, n-output combinational circuit. We assume that, at time i (i>=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 described as follows:

  CC(F): <f(a_0,b_0,c_0),f(a_1,b_1,c_1), f(a_2,b_2,c_2),…………..>.

  CC(G): <g(a_0,b_0,c_0), g(a_1,b_1,c_1), g(a_2,b_2,c_2),…………...>.

- **Register (R)** Let d_i denote an input value given to an input port *D* at time i (i>= 0). As shown in Fig. 2, the output sequence of the register is described as follows:

  R: < 0, 11, 12, 13, -, -,………>.

- **Synchronous and Asynchronous RAMs (SRAMs and ARAMs)** Let M[j] denotes the value stored in address j (j>= 0) of the RAM. Recall that the initial data values of M[j] are 0 and the WAR mode of the SRAM is considered. As shown in Fig. 2, the output sequences and stored data of an SRAM and ARAM of the time 0, 1, 2, 3, 4, 5 are as follows:

SRAM (output sequence): < 0, 0, 0, 0, 11, 12 >.

SRAM (stored data of M[1], M[2] and M[3]):

M[1]: < 0, 11, 11, 11, 11, 11 >. It means that memory content of the address value 1 is updated by 11 at time 1 and remains the same until further updating.

M[2]: < 0, 0, 12, 12, 12, 12 >. It means that memory content of the address value 2 is updated by 12 at time 2 and remains the same until further updating and

M[3]: < 0, 0, 0, 13, 13, 13 >. It means that memory content of the address value 3 is updated by 13 at time 3 and remains the same until further updating.

ARAM (output sequence): < 0, 0, 0, 11, 12, 13>.

ARAM (stored data of M[1], M[2] and M[3]):

M[1]: < 0, 11, 11, 11, 11, 11 >, **M[2]:** < 0, 0, 12, 12, 12, 12 > and **M[3]:** < 0, 0, 0, 13, 13, 13 >. These have the same aforesaid explanations.

In this paper, we assume that a fully synchronous circuit has data input, data output, a global clock input, a global reset input, a write enable input, combinational circuits (CCs), registers (Rs), SRAMs, ARAMs, and their interconnects. The readers should refer to the Fig. 3 for illustrating an example of a fully synchronous circuit. The global clock is directly connected to the clock input ports of all Rs and SRAMs, ARAMs and the global reset is connected to the reset input ports of all Rs. Also the write enable is directly connected to the write enable input ports of all SRAMs and ARAMs. We assume that a circuit has no loop.

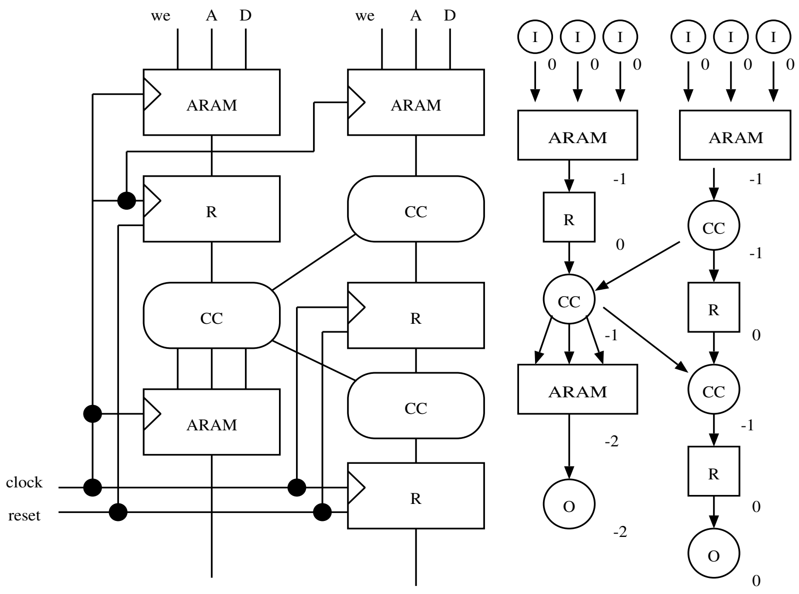

Fig. 3 An example of a fully synchronous circuit and the corresponding circuit graph with potentiality.

Let us define *equivalence* of two fully synchronous circuits for the periodic clock and initial reset. Note that, an *equivalence* of the circuits with RAMs is determined by the output sequences and stored data. We say that two circuits X and Y are an equivalent if, for any input sequence, the output sequences and stored data at the same memory locations are the same except for first several outputs and stored data.

For the periodic clock with initial reset and write enable, the output sequences and stored data of SRAM, R + ARAM and ARAM + R are described as follows (As shown in Fig. 5):

SRAM (output sequence): < 0, 0, 0, 0, 11, 12 >.
SRAM (stored data of M[1], M[2] and M[3]):
M[1]: < 0, 11, 11, 11, 11, 11 >, **M[2]:** < 0, 0, 12, 12, 12, 12 > and **M[3]:** < 0, 0, 0, 13, 13, 13 >.
R + ARAM (output sequence): < 0, 0, 0, 0, 11, 12 >.
R + ARAM (stored data of M[1], M[2] and M[3]):
M[1]: < 0, 0, 11, 11, 11, 11 >, **M[2]:** <0, 0, 0, 12, 12, 12 > and **M[3]:** < 0, 0, 0, 0, 13, 13 >.
ARAM +R (output sequence): < 0, 0, 0, 0, 11, 12 >.
ARAM + R (stored data of M[1], M[2] and M[3]):
M[1]: < 0, 11, 11, 11, 11, 11 >, **M[2]:** < 0, 0, 12, 12, 12, 12 > and **M[3]:** < 0, 0, 0, 13, 13, 13 >.

These also have the same explanations mentioned above.

Since, these three circuits, shown in Fig. 4 have the same output in time 0, 1, 2, 3, 4 and 5. Also they have the same stored data at the memory location 1 (M[1]) of the time 2, 3, 4 and 5, at memory location 2 (M[2]) of the time 3, 4 and 5 and at the memory location 3 (M[3]) of time 4 and 5. Thus, these three circuits are *equivalent*. In this paper, we ignore first several clock cycles when we determine the *equivalency* of the circuits.

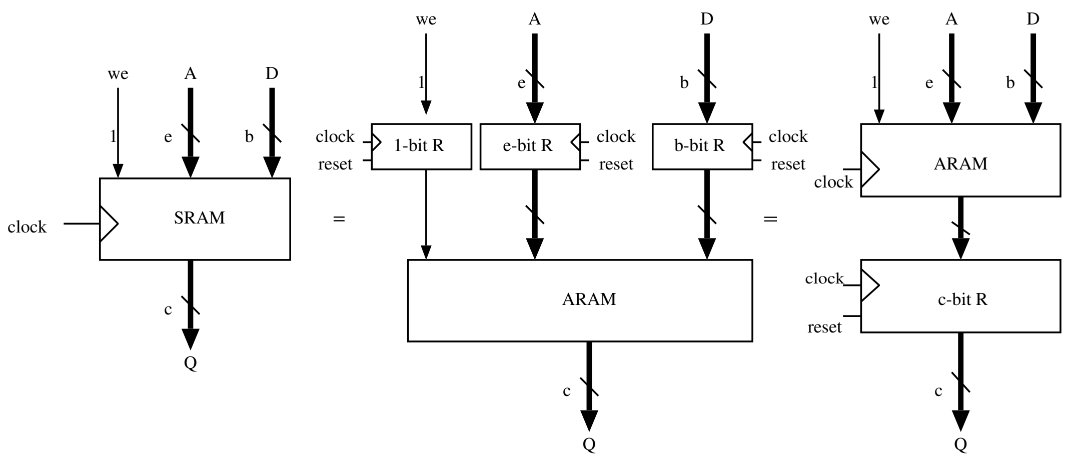

Fig. 4 SRAM, R + ARAM and ARAM + R.

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

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

Recall *a Negative Register* (NR), which is a nonexistent device used only for showing our algorithm to derive Y and related proofs. Recall again that, a regular register latches the input at the rising clock edge whereas a *negative register* latches a future input. An NR latches the value which is given to input data port *D* at the rising edge of two clock cycles later as illustrated in Fig. 2. Thus, the NR has the following output sequence for a periodic clock with an initial reset.

NR: < 12, 13, -, -, -, - >.

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

4. 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), ARAM, or SRAM. 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 and NR has one incoming and one outgoing edge. A node with label ARAM or SRAM has three incoming edges 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. 3 illustrates an example of a directed graph. Note that nodes with label I, R, NR, ARAM, or SRAM has 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.

For a given circuit X with ARAMs, we will show an algorithm to derive an ARAM-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

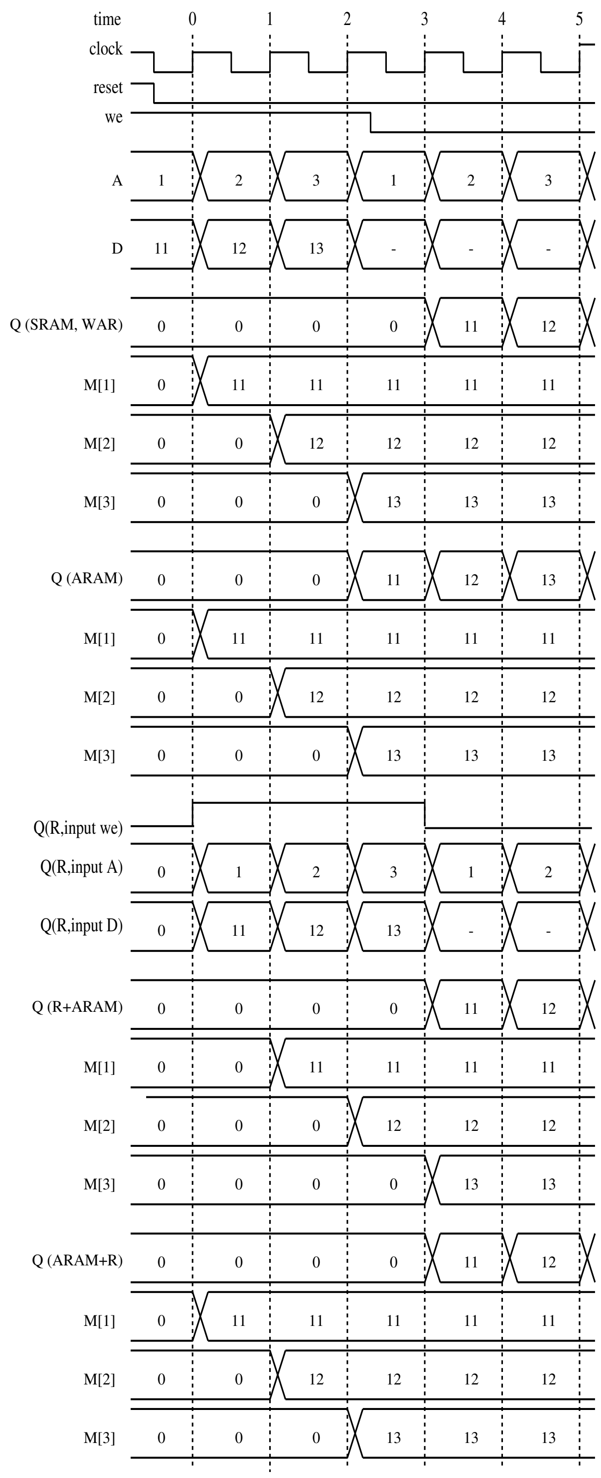

Fig. 5 A timing chart for showing the equivalency of SRAM, R + ARAM and ARAM + R.

Fig. 6 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** ARAM node is rewritten into SRAM + NR.
- **Rule 1** Adjacent R and NR nodes are rewritten into NULL circuit, that is, they are removed.

- **Rule 2** R + SRAM (or NR + SRAM) is rewritten into SRAM + R (or SRAM + NR). More specifically, if each incoming edge of an SRAM node is connected to a R node, then all the Rs are removed to the outgoing edge of the SRAM node. On the other hand, if one of the incoming edges of an SRAM node is connected to an NR node, then the NR node is removed, a R node is added to all other incoming edges, and the NR node is moved to the outgoing edge of the SRAM node.
- **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 an equivalent. Let a_i, b_i, c_i, d_i, a_i (address data) and *we* (i>=0) denote inputs given from the predecessor node at time i.

Rule 0 Both ARAM and SRAM + NR have the same output sequences and stored data.

ARAM/SRAM + NR (output sequence): < 0, 0, 0, 11, 12, 13 >.

ARAM/SRAM + NR (stored data of M[1], M[2] and M[3]):

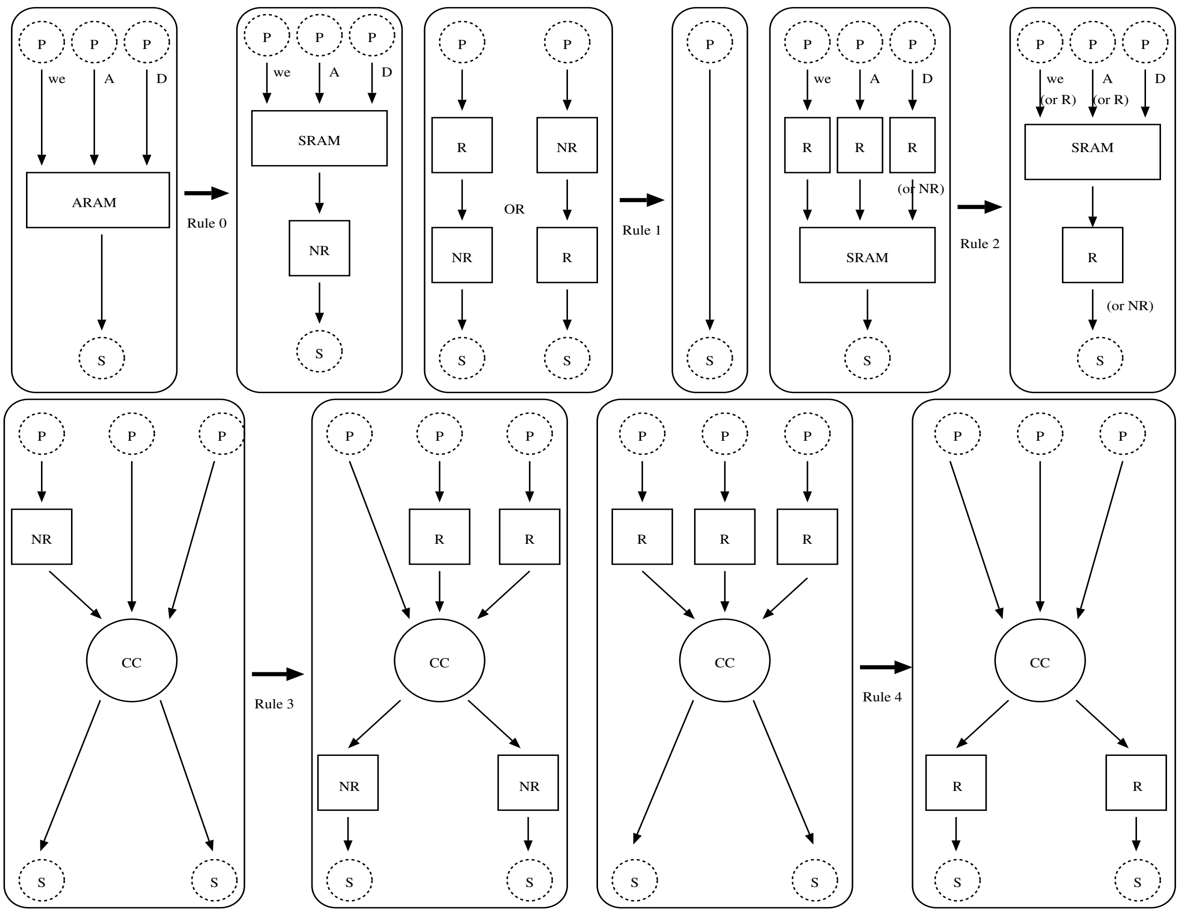

Fig. 6 Rules to rewrite a circuit graph.

M[1]: < 0, 11, 11, 11, 11, 11>; **M[2]:** < 0, 0, 12, 12, 12, 12 > and **M[3]:** < 0, 0, 0, 13, 13, 13 >. Thus they are an equivalent.

Rule 1 R + NR and NR + R have the following output sequences.

R + NR : < 11, 12, 13, -, - , - > and **NR + R**: < 0, 12,13,-, -, - >. Also, NULL circuit has the following output sequence.

NULL: < 11, 12, 13, -, -, - >. Thus, they are an equivalent.

Rule 2 R + SRAM has the following output sequence and stored data.

R + SRAM (output sequence): < 0, 0, 0, 0, 0, 11 >.

R + SRAM (stored data of M[1], M[2] and M[3]):

M[1]: < 0, 0, 11, 11, 11, 11>; **M[2]:** < 0, 0, 0, 12, 12, 12 > and **M[3]:** < 0, 0, 0, 0, 13, 13 >.

SRAM + R has the following output sequence and stored data.

SRAM + R (output sequence): < 0, 0, 0, 0, 0, 11 >.

SRAM + R (stored data of M[1], M[2] and M[3]:

M[1]: < 0, 11, 11, 11, 11, 11 >; **M[2]:** < 0, 0, 12, 12, 12, 12 > and **M[3]:** < 0, 0, 0, 13, 13, 13 >. Thus they are an equivalent.

On the other hand, NR + SRAM has the following output sequence and stored data.

NR + SRAM (output sequence): < 0, 0, 0, 0, 12, 13 >.

NR + SRAM (stored data of M[1], M[2] and M[3]):

M[1]: < 0, 12, 12, 12, 12, 12 >; M[2]: < 0, 0, 13, 13, 13, 13 > and **M[3]:** < 0, 0, 0, -, -, - >.

SRAM + NR has the following output sequence and stored data.

SRAM + NR (output sequence): < 0, 0, 0, 0, 12, 13 >.

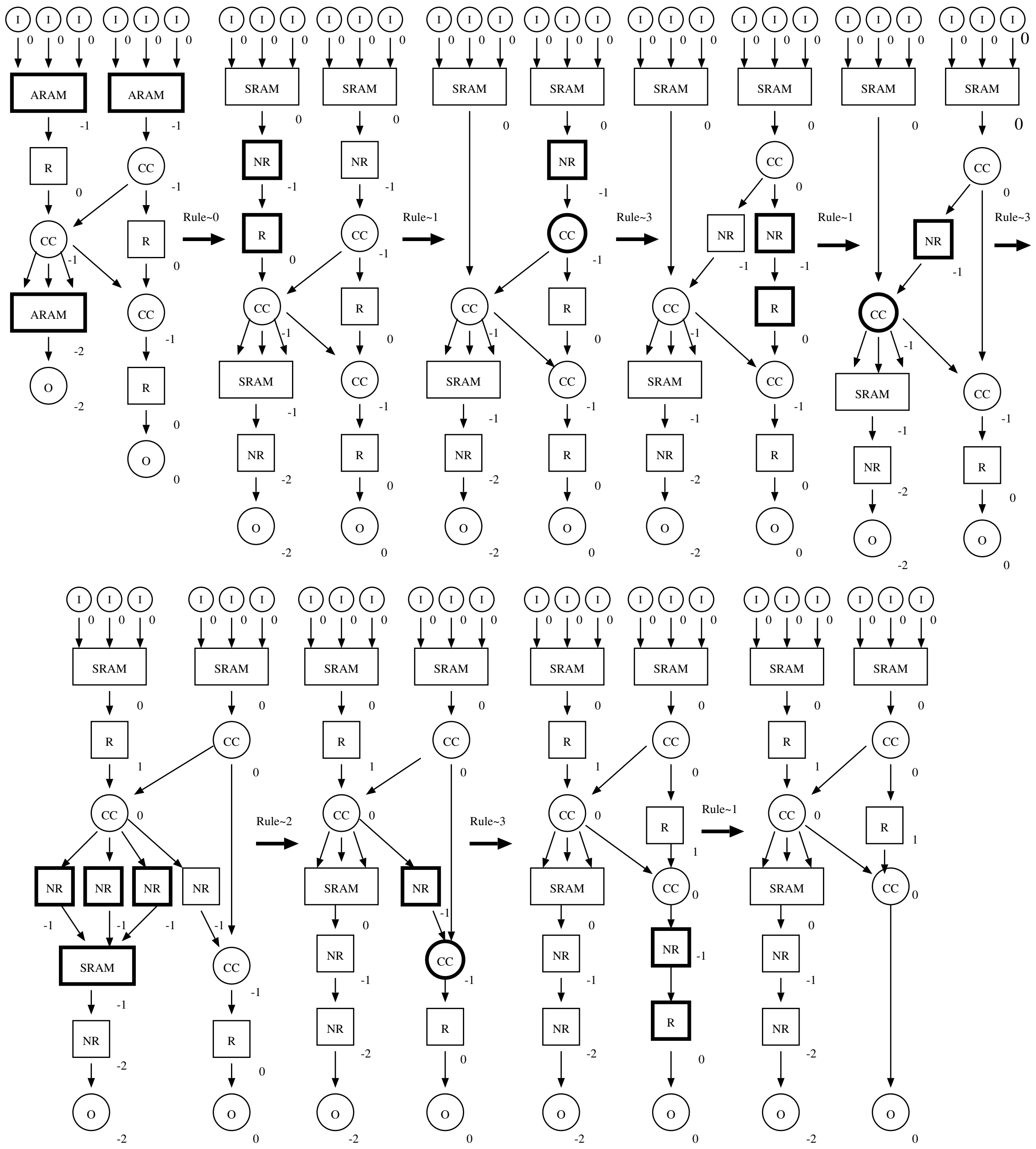

Fig. 7 Interim and resulting circuit graphs obtained by our rewriting algorithm for a circuit graph.

SRAM + NR (stored data of M[1], M[2] and M[3]): **M[1]:** < 0, 0, 12, 12, 12, 12 >; **M[2]:** < 0, 0, 0, 13, 13, 13 > and **M[3]:** < 0, 0, 0, 0, -, - >. Thus they are an equivalent.

For the Rule 2, we consider that an NR is connected to data input port *D* only. If an NR is connected to the address input port *A* or write enable input *we*, an equivalency of the Rule 2 can be shown in the similar way.

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

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

We are now in position to apply the rewriting algorithm to the given input circuit. Suppose that an input circuit graph has nodes with labels **I, O, R, ARAM, SRAM**, 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.

5. Behavior of Our Circuit Rewriting Algorithm

Let us observe the behavior of the rewriting algorithm.

- First, the rewriting algorithm repeats the applying Rule 0 to all ARAM nodes until all ARAM nodes are rewritten into SRAM + NR. After that, Rule 0 is never be applied.
- Rules 1 is applied and adjacent R and NR nodes are removed whenever possible.
- R or NR node is moved toward the output nodes using Rules2, Rule 3 and Rule 4 whenever possible.

The Fig. 7 shows one of the applications of our rewriting algorithm. First, our rewriting algorithm repeats the applying Rule 0 to all ARAM nodes until all ARAM nodes are rewritten into SRAM + NR. After that, Rule 1 is used to remove adjacent R and NR. Then Rule 3, Rule 1, Rule 3, Rule 2, Rule 3, Rule 1 are applied one after another. 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 the 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, then $p(v) = p(u_1)$.
- If v is SRAM, then $p(v) = \min(p(u_1), p(u_2),p(u_3))$.
- If v is ARAM, then $p(v) = \min(p(u_1), p(u_2),p(u_3)) - 1$.
- If v is NR, then $p(v) = p(u_1) - 1$.
- If v is R, then $p(v) = p(u_1) + 1$.
- If v is CC, then $p(v) = \min(p(u_1), p(u_2),\ldots\ldots\ldots., p(u_k))$.

The Fig. 3 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 ARAM-free and NR-free circuit graph, equivalent to the original circuit graph.

6. Proof of Theorem 1

The main purpose of this section is to show 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_1, P_2, P_3 and S denote the predecessor and successor nodes for Rules 0 and 2. P and S denote the predecessor and successor nodes for Rules 1. Also let P_1, P_2, P_3 and S_1, S_2, denote the predecessor and successor nodes for Rules 3 and 4. We compute the potentiality of each successor node both before and after applying the rules as follows.

Rule 0 $p(S) = \min(p(P_1), p(P_2), p(P_3)) -1$.

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

Rule 2 $p(S) = \min(p(P_1)+1, p(P_2)+1, p(P_3)+1)) =$

min(p(P_1), p(P_2), p(P_3)) + 1 if all are R and p(S) = min(p(P_1), p(P_2), p(P_3) - 1)) = min(p(P_1) + 1, p(P_2) + 1, p(P_3)) - 1 if any of them is NR. In this case an NR is connected to data input port *D*.

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 Fig. 7, the potentialities of the left and the right O nodes are -2 and 0, respectively, and these values are never changed.

In a circuit graph, let *a segment* be a directed path u_1, u_2,…, u_k such that, u_1 and u_k are either I, O, SRAM, or CC, and u_2, ……, u_{k-1} are either R or NR. Note that, if k = 2 then it represents a null segment with u_1 and 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, SRAM, 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 SRAM and CC nodes in the resulting circuit graph have zero potentiality.

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

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

Similarly, we have the following lemma.

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

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

Case 1 v is an SRAM node

Let (u_1, v), (u_2, v), and (u_3, v), denote the incoming edges. From Lemma 3, none of u_1, u_2 and u_3 is an NR node. If u_1, u_2 and u_3, all are R, then Rule 2 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 or SRAM or CC node. If this is the case, p(u_1 or u_2 or u_3) = 0 and thus, p(v) = 0, a contradiction.

Case 2 v is a CC node

Let (u_1, v), (u_1, v),…, (u_k, v) (k >= 1) denote the incoming edges. From Lemma 3, none of u_1, u_2,…, 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 or SRAM or CC node. From the assumption, the potentiality of such node is non-positive, Hence, the potentiality of v is non-positive, a contradiction.

From Lemma 4 and 5, all SRAM 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.

By the Theorem 1, it is not always possible to have an equivalent ARAM-free circuit. However, we may modify a circuit such that it can be converted into an almost equivalent ARAM-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 ARAM-free circuit according to our Theorem 1.

7. Conclusions

In this paper, we have presented a rewriting algorithm and five rewriting rules to convert a sequential circuit with ARAMs into an equivalent fully synchronous circuit with no ARAMs for the current FPGA. We consider both read and write operations of the memory blocks (RAMs) whereas only read operation of the memory blocks (ROMs) is considered in our previous paper [5]. In fact, we have improved our previous research work where RAMs can be used as the additional circuit elements to the given input sequential circuits. Using the rewriting algorithm, any sequential circuit with ARAMs can be converted into an equivalent fully synchronous sequential circuit with no ARAMs to support the modern FPGA architecture. As a future work, we have a plan to present an algorithm for circuits with feedback loops.

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. *International Journal on Foundations of Computer Science,* pages 403-416, 2004

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

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

[5] M. N. I Mondal, K. Nakano and Y. Ito. A rewriting algorithm to generate AROM-free fully synchronous circuits. *In Proc. of the First International Conference on Networking and Computing (ICNC),* pages 148-155, November 2010.