International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 2, Number 2, pages 269–290, July 2012

A Rewriting Approach to Replace Asynchronous ROMs with Synchronous Ones for the Circuits with Cycles

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

Received: February 13, 2012
Revised: May 21, 2012
Accepted: June 19, 2012
Communicated by Sayaka Kamei

Abstract

Field Programmable Gate Arrays (FPGAs) are a dominant implementation medium for digital circuits which are used to embed a circuit designed by users instantly. FPGAs can be used for implementing parallel and hardware algorithms. 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 is given. In our previous work, we have presented a circuit rewriting algorithm to convert a circuit with asynchronous ROMs into an equivalent circuit with synchronous ones. The resulting circuit with synchronous ROMs can be embedded into FPGAs. However, this circuit rewriting algorithm can handle circuits represented by a directed acyclic graph and does not work for those with cycles. In this paper, we succeeded in relaxing the cycle-free condition of circuits. More specifically, we present an algorithm that automatically converts a circuit with cycles using asynchronous ROMs into an equivalent circuit using synchronous ROMs.

Keywords: FPGA, Read Only Memories, Asynchronous read operations, Circuit 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. 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 clock 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, 3, 5, 6, 7, 8, 13, 14] to accelerate computation. In particular, the FPGAs can implement hundreds of circuits that work in parallel to accelerate useful computations. For example, in paper [6], parallel implementation for the exhaustive verification of the Collatz conjecture is presented. In this implementation, 24 co-processors embedded in a Xilinx Virtex-2 Family FPGA perform the exhaustive verification in parallel.

We mainly focus the asynchronous and synchronous read operations of memory blocks in this paper.

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.

In other words, we say that asynchronous ROMs (AROMs) and synchronous ROMs (SROMs) support asynchronous and synchronous read operations respectively. 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. Hence, latency of asynchronous read operation is 0, while synchronous read operation is 1. To understand clearly, readers may refer to Figure 5 that shows the timing chart of AROM and SROM supporting asynchronous and synchronous read operations respectively. Embedded block memories in most modern FPGAs support synchronous read operation, but do not support asynchronous one. Hence, users who design circuits embedded into FPGAs can not use asynchronous read operation. However, circuit design using asynchronous one is easier, because it has 0 latency.

The main contribution of this paper is to provide one of the potent approaches to resolve this problem. Suppose that user design a circuit with ROMs supporting asynchronous read operation (*AROMs* for short). We present an algorithm that automatically converts the circuit into an equivalent circuit with ROMs supporting synchronous read operation (*SROMs* for short). The resulting circuit can be implemented into FPGAs.

Our circuit rewriting approach, presented in this paper is used to convert *an asynchronous circuit* consisting

> *Combinational Circuits (CCs), Registers (Rs), and ROMs with asynchronous read operations (AROMs)*

into *an equivalent synchronous circuit* consisting

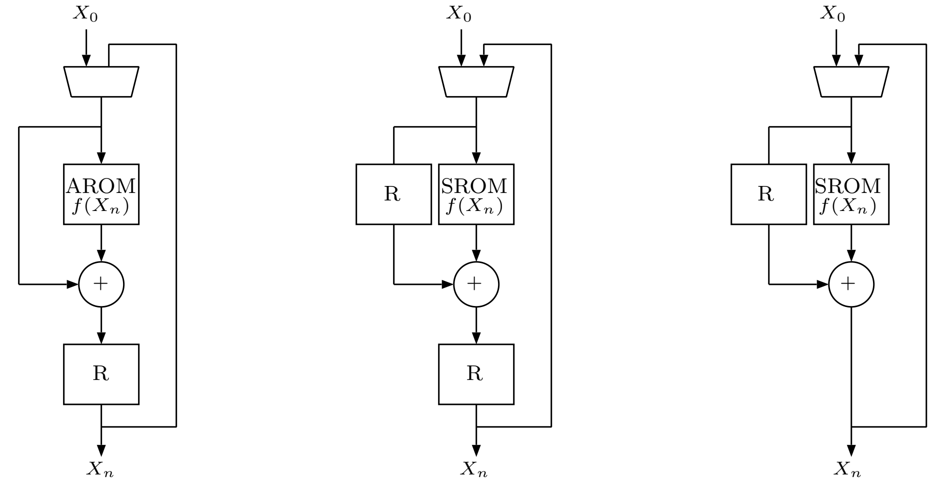
> *Combinational circuits (CCs), Registers (Rs), and ROMs with synchronous read operations (SROMs).*

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 converts 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 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



(a) A circuit with an AROM

(b) A circuit with an SROM by a non-expert

(c) The converted circuit with an SROM

Figure 1: An example of circuits using an AROM and an SROM

$n \geq 1$. 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 an 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 Figure 1 (b) needs double clock cycles over the circuit in Figure 1 (a). Using our algorithm to the circuit in Figure 1 (a), we can obtain the circuit in Figure 1 (c) automatically. In the circuit with an SROM in Figure 1 (c), $X_1, X_2, \ldots$ is output in every clock cycle. Thus, the timings of the circuits in Figure 1 (a) and (c) are identical.

By our rewriting algorithm, obviously we can minimize the number of clock cycles in the AROM-free resulting circuits as illustrated in Figure 1 (c), but it is not trivial for the non-expert or quickly designed by an expert to minimize the number of clock cycles to obtain circuit in Figure 1 (c). However our algorithm can do it automatically.

On the other hand, the readers may think that the resulting AROM-free circuit has large propagation delay and low clock frequency, because our rewriting algorithm moves registers towards 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 circuit performance degrades. If this is the case, then it is possible to improve circuit performance of the AROM-free resulting circuit. The ideas to improve the performance of the AROM-free resulting circuit in terms of the latency and delay are the same as described details in our paper [12] by the same authors, although performance improvement of the AROM-free resulting circuits is beyond of this paper. However, we will briefly describe the circuit performance improvement techniques of the AROM-free resulting circuit. The techniques are as follows:

- In order to minimize latency in the AROM-free resulting circuit, we first need to define redundant registers. The redundant registers are the registers which are connected to output ports of the AROM-free resulting circuit. For minimizing latency, we may remove all the redundant registers, if they do not create the timing problems for a circuit connected to the output ports.

- Clock performance of the AROM-free resulting circuit degrades due to the longest path between input ports to registers/SROMs or registers/SROMs to registers/SROMs or registers/SROMs to output ports. For this case, we can add registers by dividing the AROM-free resulting circuit into several layers so that the longest path becomes small. Hence clock performance is increased in the AROM-free resulting circuit.

The outlines of our idea are described as follows:

- We use *a Negative Register* (NR) which is originally introduced in our previous paper [12] by the same authors. The NR is an imaginary register latching a future input data.
- We define simple *six rules* that rewrite a circuit.
- 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.

We use the key and innovative idea of introducing Negative Register (NR). For the reader's benefit, we briefly describe the behavior of our rewriting algorithm. 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. The readers may refer to the Section 5 for the details about the behavior of our rewriting algorithm.

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 [4, 9, 10]. To cut the clock distribution power, an asynchronous circuit design in FPGAs is very much suitable, described in [11, 16, 18]. 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 [4, 10]. Therefore, we should use SROMs when we need a function of ROMs. There are some dedicated FPGAs to test asynchronous circuits. However, these FPGAs are closely associated to a style of design. For example, MONTAGE [15] is based on an asynchronous design and PAPA [17] is a fully asynchronous FPGA dedicated to optimize pipeline circuits.

The main contribution of this paper is to modify the circuit rewriting algorithm, presented in [12] to process practical circuits with cycles. More specifically, our new circuit rewriting algorithm can convert any circuit represented by a directed reachable graph (DRG), illustrated in Figure 2 (2). A directed reachable graph is a directed graph such that, for every internal node, there exists a directed path from an input node to an output node which includes it. Note that, one node and/or one directed path may appear twice or more in a directed path. For example, $(B, E, H, I, F, E, H, K, N, O)$ is a directed path. It should not have any difficulty to confirm that, every internal node in Figure 2 (2) is included. Clearly, a class of the DRG includes that of the DAG. Also, almost all practical circuits can be represented by a DRG. If there exists a node that is not in the directed path from an input node to an output node, the directed graph is not a DRG. Clearly, circuit elements corresponding nodes that are not in the directed path to an output node make no sense because such circuit elements do not affect the outputs. However, practical circuits may have circuit elements corresponding nodes that are not in the directed path from an input node. We will show that, even if a circuit graph has such nodes, we can convert it to an equivalent AROM-free circuit graph.

Our results have several significant points as follows:

- The correctness of our algorithm is proved in a rigorous manner.
- Our algorithm works for the practical circuits. In particular, we handle practical circuits which have cycles.

- Our circuit rewriting algorithm moves all redundant registers towards 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. We briefly discuss these performance improvement techniques for the resulting circuit which are the same as described details in our paper [12].

- We additionally describe a technique to generate AROM-free circuit even if the input circuit is beyond the DRG circuit. Particularly, if the input circuits have such elements which are not in the path of DRG circuits, we can also convert those circuits into the equivalent AROM-free circuits as illustrated in Section 7.

- 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.

This paper is organized as follows: Section 2 briefly describes the related work so far. We briefly review the circuits and their equivalency in Section 3. In Section 4, we describe our rewriting algorithm, circuit graph and also explain the equivalency for our rewriting rules. For the reader's benefits, Section 5 shows how our circuit rewriting algorithm works for circuit graphs. Section 6 presents the proof of the correctness of our rewriting algorithm. Section 7 shows how we handle nodes that are not in the path from an input node. Finally Section 8 concludes this work.

2 Related Work

In this section, briefly we will describe about the related work so far. However, there is no related work except our previous one [12]. Hence, we will briefly summarize our previous work [12] as a related one such that readers may compare our contribution in the current work, described in Section 1 with the previous one [12]. Note that, we are providing an innovative approach which is devoted for implementing asynchronous read operation in the current FPGAs. We assume that the input circuit with AROMs supporting asynchronous read operation, designed by users is given. However, we can not implement this circuit into the current FPGAs, because current FPGAs have SROMs supporting synchronous read operation. For this purpose, we provide one of the potent circuit rewriting approaches to implement circuits with AROMs supporting asynchronous read operation in the current FPGAs. In our previous paper [12], we have presented a circuit rewriting approach for circuits represented by a directed acyclic graph (DAG), illustrated in Figure 2 (1) which has no directed cycle. This graph has 3 input nodes and 3 output nodes, each of which corresponds to input ports and output ports of the circuit, respectively. The other internal nodes correspond to circuit elements such as combinational circuits, registers, and ROMs. The presented circuit rewriting approach converts a circuit with combinational circuits, registers and AROMs represented by a DAG, illustrated in Figure 2 (1) into an equivalent AROM-free circuit with combinational circuits, registers and SROMs for implementing in the current FPGAs.

However, the circuit rewriting approach presented in [12] has a strict restriction in terms of input circuits. It works only for a circuit whose underlying graph is a DAG, illustrated in Figure 2 (1). Although most of practical circuits have cycles, it can not handle such circuits as illustrated in Figure 2 (2). To overcome this problem, a modified circuit rewriting approach is presented in this paper. More specifically, our new circuit rewriting algorithm can convert any circuit with AROMs, represented by Directed Reachable Graph (DRG), illustrated in Figure 2 (2) into an equivalent circuit with SROMs for implementing in the current FPGAs.

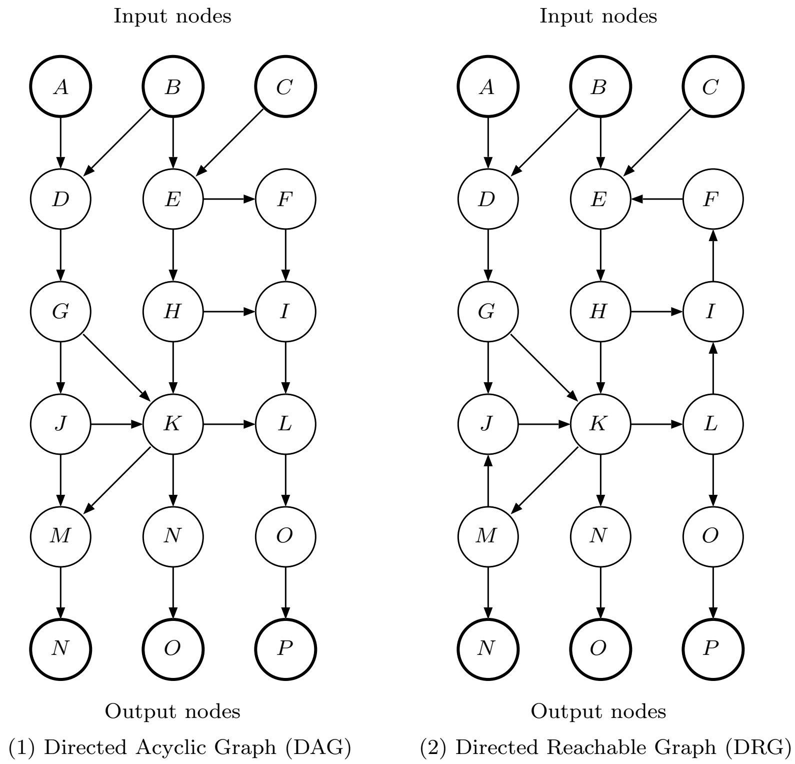


Figure 2: A directed acyclic graph(DAG) and a directed reachable graph(DRG)

3 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 3. Once inputs are given, the outputs are computed in small delay.

A register has a clock input and a reset input as illustrated in Figure 4. It can store fixed bits of 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.

A ROM (Read Only Memory) has a (address) input d and a data output q as illustrated in Figure 4. It is storing 2^b words such as $M[0]$, $M[1]$, …, $M[2^b - 1]$, where b is the number of address bits. 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 Figure 5 shows a timing diagram of reading operations of the R, SROM, AROM and NR (Negative Register). In the figure, time 0, 1, 2, … correspond to rising edges of the periodic clock

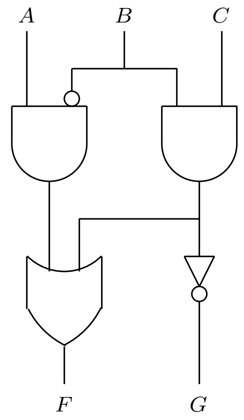


Figure 3: An example of a combinational circuit (CC).

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. 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 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 Figure 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 Figure 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:

  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 denotes 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 $M[j]$ denotes 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,

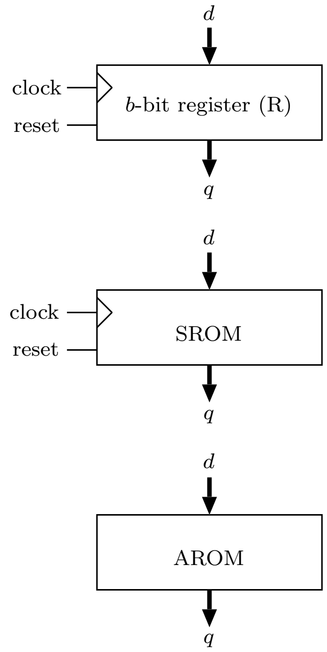


Figure 4: A register (R), a synchronous ROM (SROM) and an asynchronous ROM (AROM).

and their interconnects. The readers should refer to Figure 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 cycles.

Let us define *an 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 SROM+R, that is, the output of the SROM is connected to the input of the R as illustrated in Figure 7. We also consider a circuit R+SROM, in which the output of the R and the input of the SROM are connected. In this regard, we consider another circuit which consists 2 registers (2 Rs) and an AROM. The output of the R is connected to the input of the AROM whereas the output of the AROM is connected to the input of the other R, as illustrated in Figure 7. For the periodic clock with initial reset, the output sequences of SROM+R, R+SROM, and R+AROM+R are as follows:

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

Since these three circuits have the same output in time 2, 3, ..., they are an equivalent. Note that, the outputs in time 0 and 1 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 an equivalent to X, and
- an algorithm to derive Y if the necessary condition is satisfied.

We will introduce *a negative register* (NR), which is a nonexistent device used only for showing our algorithm to derive Y and related proofs. This is originally introduced in our previous paper [12].

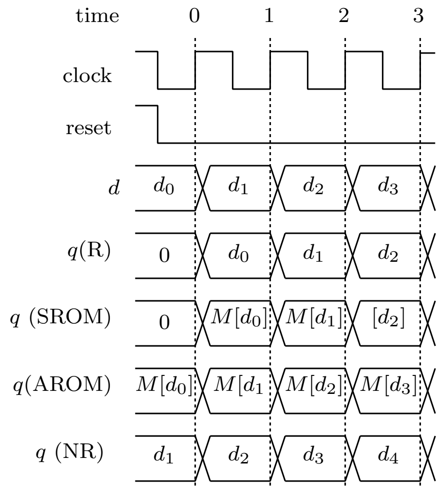


Figure 5: A timing chart of a register (R), an SROM, an AROM and a negative register (NR).

Recall that, a regular register latches the input at the rising clock edge. *A negative register* latches a future input. The Figure 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 Figure 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.

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 for 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 reachable graph (DRG), such that for every internal node, there exists a directed path from an input node to an output node which includes it. Figure 2 (2) illustrates an example of a DRG.

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 Figure 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 Figure 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.

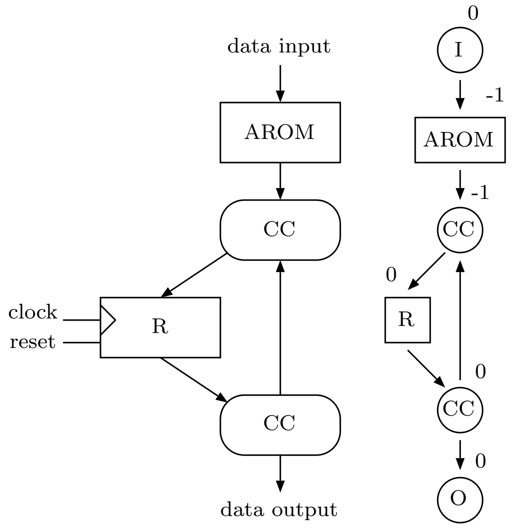


Figure 6: An example of a fully synchronous circuit with cycle and the corresponding circuit graph with potentiality.

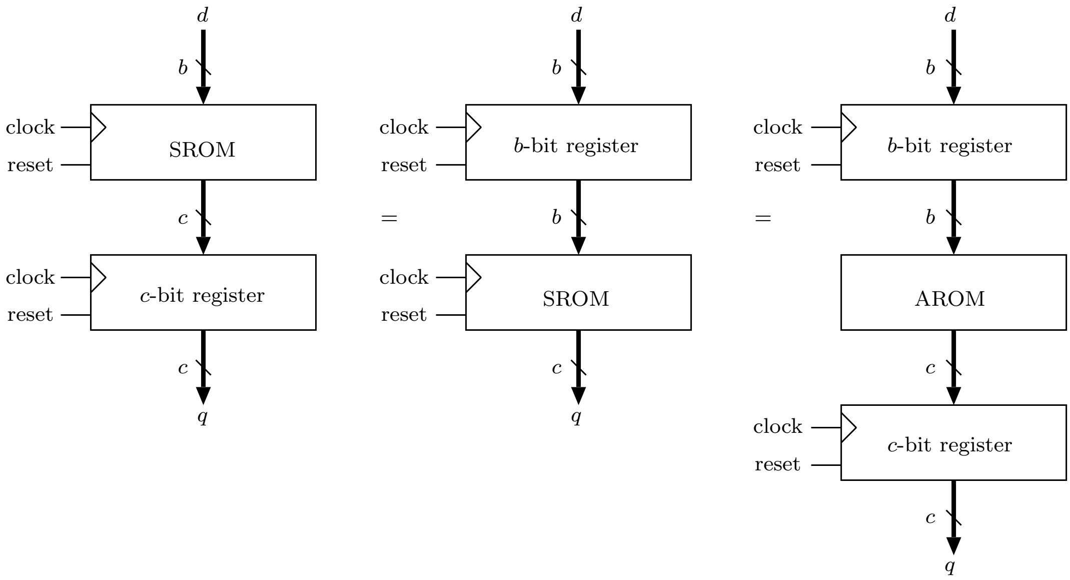


Figure 7: SROM+R, R+SROM, and R+AROM+R.

Rule 2 R+SROM is rewritten into SROM+R.

Rule 3 If all the incoming edges of a CC node are connected to an R node, then Rs are moved to all the outgoing edges of the CC node.

Rule 4 NR+SROM is rewritten into SROM+NR.

Rule 5 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.

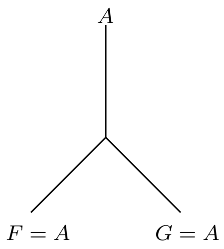


Figure 8: A combinational circuit to implement fan-out 2 circuit.

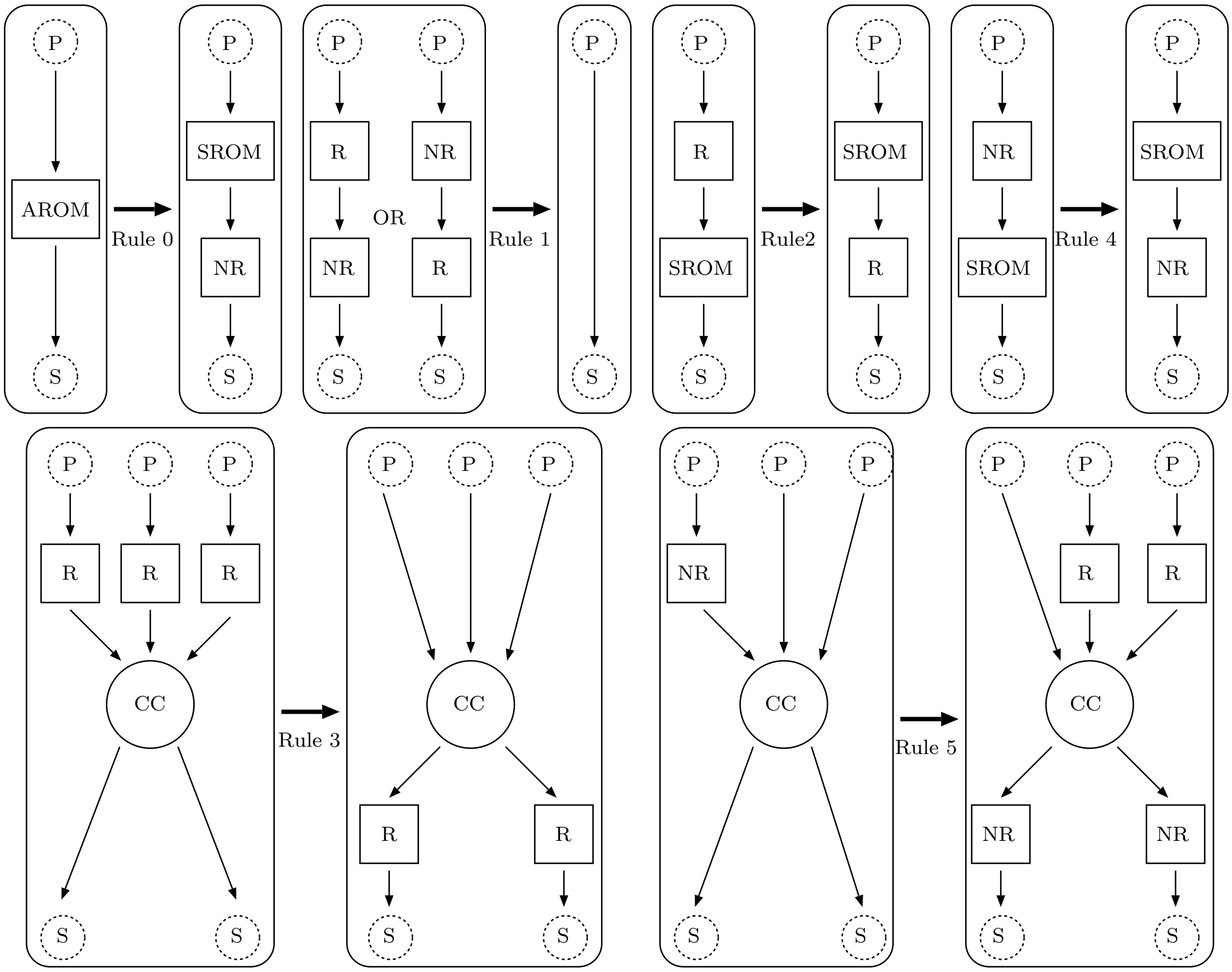


Figure 9: Rules to rewrite a circuit graph.

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 an 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 an 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 an equivalent.

Rule 3 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 an equivalent.

Rule 4 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 an equivalent.

Rule 5 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 an 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.*

In other words, our algorithm invokes the Rule i (i varies from 0 to 5) and applies (whenever applicable) as a priority basis to the current circuit graph until no more applying is possible. For example, Rule 0 has higher priority than Rule 1, Rule 1 has higher priority than Rule 2 and so on. When no rule is applicable to the current circuit graph, we have an equivalent AROM-free and NR-free resulting circuit graph to implement into the current FPGAs for the given input circuit graph with AROMs.

For the reader's benefit, we will show more concrete description of our rewriting algorithm. Our rewriting algorithm repeatedly changes a circuit graph. Let *#nodes* denote the number of nodes of the current circuit graph, and $v_0, v_1, \ldots, v_{\#nodes-1}$ denote all the nodes. Note that, the number of nodes may change by applying a rule. If this is the case, we assume that the value of *#nodes* is automatically updated. Our rewriting algorithm can be described as follows:

START:
 for $i \leftarrow 0$ to 5 do
  for $j \leftarrow 0$ to $\#nodes - 1$ do
   if Rule i can be applied for v_j or v_j with its neighbors
    begin
     Apply Rule i for v_j or v_j with its neighbors
     goto START
    end

It should be clear that, when a rule is applied, our rewriting algorithm starts over. Thus, our rewriting algorithm repeatedly applies Rule i, where i be the minimum possible number.

5 Behavior of Our Circuit Rewriting Algorithm

Let us observe the behavior of our circuit rewriting algorithm.

- First, Rule 0 is applied to all AROM nodes, and they are rewritten into SROM+NR. After that, Rule 0 is never applied.

- Rules 1 is applied and adjacent R and NR nodes are removed whenever possible.
- R nodes are moved towards the output nodes using Rules 2 and 3 whenever possible.
- NR nodes are moved towards the output nodes or are rotated in cycles using Rules 4 and 5.

Let us see how our circuit rewriting algorithm works using an example of a circuit in Figure 10, which shows the interim and resulting circuit graphs. First, Rule 0 is applied to the AROM, it is converted into SROM+NR. After that, Rule 3 is used to move the R, and two Rs are generated. Rule 5 is applied to move the NR and it is duplicated. Finally, adjacent R and NR are removed by Rule 1.

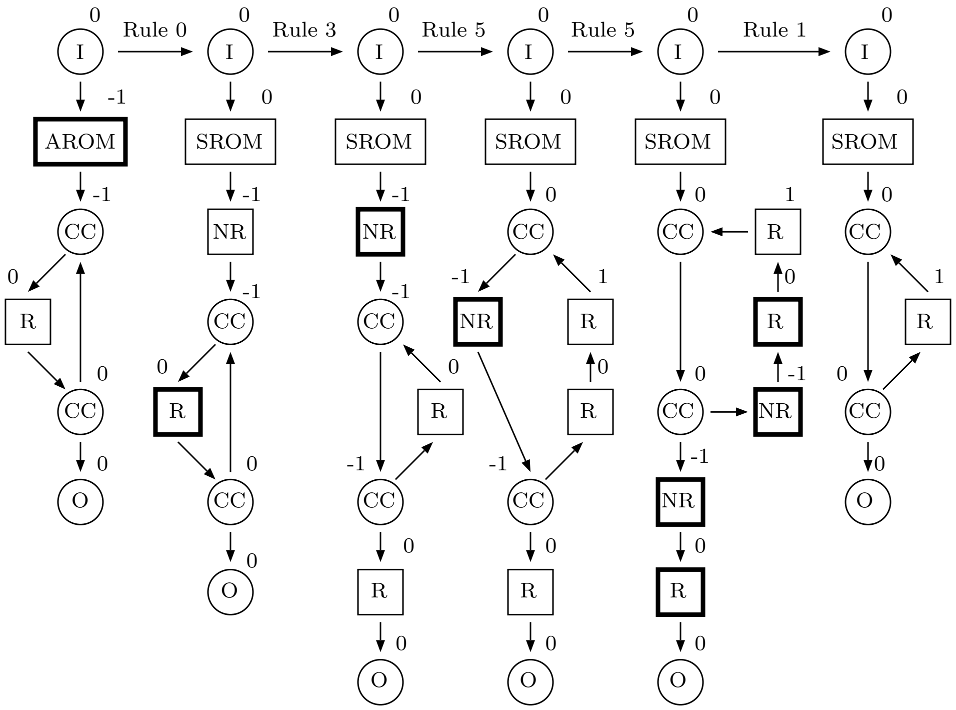


Figure 10: Interim and resulting circuit graphs obtained by our rewriting algorithm for a circuit graph.

Our circuit rewriting algorithm may not terminate for a circuit graph that has no way to convert an equivalent AROM-free circuit. Figure 11 shows an example of such circuit graph. It has a cycle with two AROMs and one R. Intuitively, one R is necessary to convert an AROM into an SROM. Thus, this circuit graph can not be converted into an equivalent AROM-free circuit. Let us see how our circuit rewriting algorithm works for the circuit graph in Figure 11. After applied Rule 0 and Rule 1, the interim circuit graph has an NR in the cycle. Rule 5 is applied to move the NR, and a new R is generated between the I node and the CC node. After that, the NR jumps over the SROM by Rule 4. Rule 5 is applied again, and a new NR is generated between the CC node and the O node. Again, the NR jumps over the SROM by Rule 4. The readers should have no difficulty to confirm that, while the NR is rotated in the cycle, one new R is generated between the I node and the CC node and one new NR is generated between the CC node and the O node. Rule 5 and Rule 4 can be repeated applied in the same way. In general, after Rule 5 and Rule 4 applied $2n$ times, new n R's and n NR's are generated, and our circuit rewriting algorithm never terminates.

For the purpose of clarifying the condition such that our rewriting algorithm can generate AROM-free and NR-free circuit graph, we define *the potentiality of the nodes* in a circuit graph. Suppose

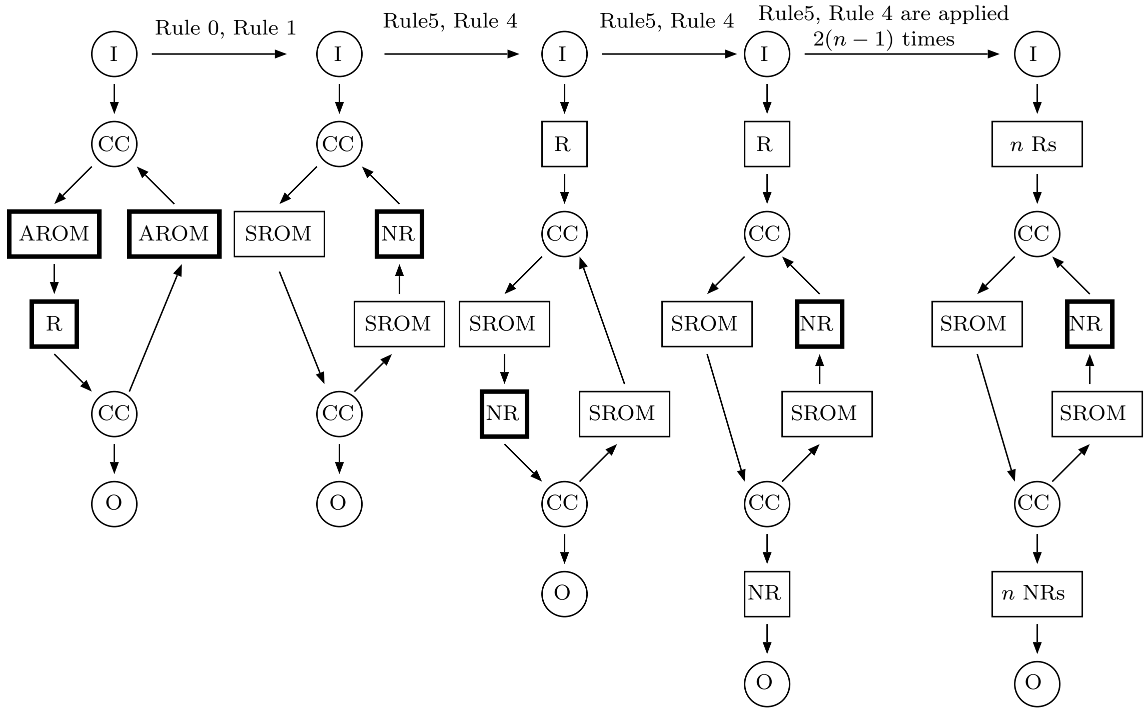


Figure 11: Example of a circuit graph for which our rewriting algorithm does not terminate.

that a node v of a circuit graph has m (≥ 0) incoming edges such as $(u_1, v), (u_2, v), \ldots, (u_m, v)$. 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 $p(v) = \min(p(u_1), p(u_2), \ldots, p(u_m))$.

From the definition, the potentiality of a node can be determined if the potentiality of all predecessor nodes are determined. Unfortunately, as we will show next, we may not determine the potentiality of every node by the above definition, if a circuit graph has a cycle.

Let us discuss the potentiality for a circuit graph with a cycle using three circuits in Figure 12. Let the potentiality $p(a)$ of the CC node a be k. From the definition of the potentiality, we can write the equations of potentiality for Figure 12 (1) as follows:

> $p(a) = k$, $p(b) = \min(p(a), p(e))$, $p(c) = p(b) + 1$, $p(d) = p(c)$, $p(e) = p(c) + 1$, and $p(f) = p(d)$.

From these equations, we have, $p(e) = p(c) + 1 = p(b) + 2$ and thus, $p(b) = \min(k, p(b) + 2)$. Hence, we can determine the value of $p(b)$ such that $p(b) = k$. Further, we can determine the potentiality of the other nodes as follows: $p(c) = p(d) = p(f) = k+1$, and $p(e) = k+2$. Intuitively, the equation $p(b) = \min(k, p(b)+2)$ means that the cycle is a *positive cycle* because the cycle $b-c-d-e$ increases the potentiality by $+2$.

We can do the same discussion for Figure 12 (2) as follows:

$p(a) = k$, $p(b) = \min(p(a), p(e))$, $p(c) = p(b) + 1$, $p(d) = p(c)$, $p(e) = p(c) - 1$, and $p(f) = p(d)$.

From these equations, we have, $p(b) = \min(k, p(b))$. Regardless the value of $p(b)$, this equation is satisfied. If this is the case, we assume that $p(b) = k$. We can then determine the potentiality of the other nodes as follows: $p(c) = p(d) = p(f) = k + 1$, and $p(e) = k$. Similarly, from the equation $p(b) = \min(k, p(b))$, we can think that the cycle is a *zero cycle*.

Figure 12 (3) shows an example of a *negative cycle*. We have the equations as follows:

$p(a) = k$, $p(b) = \min(p(a), p(e))$, $p(c) = p(b) - 1$, $p(d) = p(c)$, $p(e) = p(c) - 1$, and $p(f) = p(d)$.

From these equations, we have, $p(b) = \min(k, p(b) - 2)$. If $p(b) \neq k$ then $p(b) = p(b) - 2$. Hence $p(b) = k$ must be satisfied. If this is the case, $p(b) = \min(k, k-2) = k-2$, a contradiction. Therefore, $p(b) = \min(k, p(b) - 2)$ has no solution.

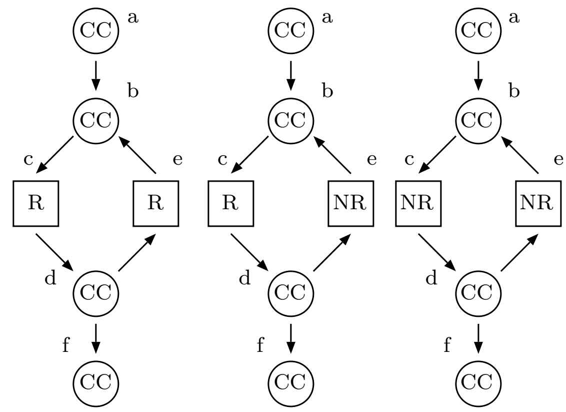


Figure 12: The potentiality for circuits with a cycle

From this observation, we define *the potentiality of a cycle* as follows: Let $v_0, v_1, \ldots, v_m (= v_0)$ be a cycle such that there is a directed edge (v_i, v_{i+1}) $(0 \leq i \leq m-1)$. We define the potentiality $p'(v_i)$ of node v_i $(1 \leq i \leq m)$ with respect to the cycle starting v_0 as follows:

- $p'(v_0) = 0$.
- If v_{i+1} is CC or SROM, then $p'(v_{i+1}) = p'(v_i)$ $(0 \leq i \leq m-1)$.
- If v_{i+1} is AROM or NR then $p'(v_{i+1}) = p'(v_i) - 1$ $(0 \leq i \leq m-1)$.
- If v_{i+1} is R then $p'(v_{i+1}) = p'(v_i) + 1$ $(0 \leq i \leq m-1)$.

We say that *the potentiality of the cycle* is $p'(v_m)$. For example, the potentialities of the cycles in Figure 12 (1), (2), and (3) are 2, 0, and -2, respectively.

We have the following theorem.

Theorem 1 *Our rewriting algorithm generates an AROM-free and NR-free circuit graph, equivalent to the original circuit graph, if all O nodes and all cycles of a circuit graph have non-negative potentiality.*

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 and all cycles of the corresponding circuit graph. Also, the potentiality of all O nodes and all cycles 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 an equivalent to the original fully synchronous circuit. For the reader's benefit, we will explain two examples as shown in Figure 10 and Figure 13. In Figure 10, the potentiality of the O node and cycle are non-negative. Hence, our rewriting algorithm generates an AROM-free and NR-free circuit graph. In Figure 13, the potentiality of the O node is negative, however the potentiality of the cycle is non-negative. Hence, our rewriting algorithm does not generate an AROM-free and NR-free circuit graph. In fact, we recall the Figure 10 with a slight modification as illustrated in Figure 13 to understand the failure case of our rewriting algorithm. A slight modification is that we just move the position of the AROM and R nodes in the designed input circuit graph as illustrated in Figure 13. It is observed in Figure 13 that the resulting circuit graph has an NR node and hence we say, our rewriting algorithm fails to remove all NRs.

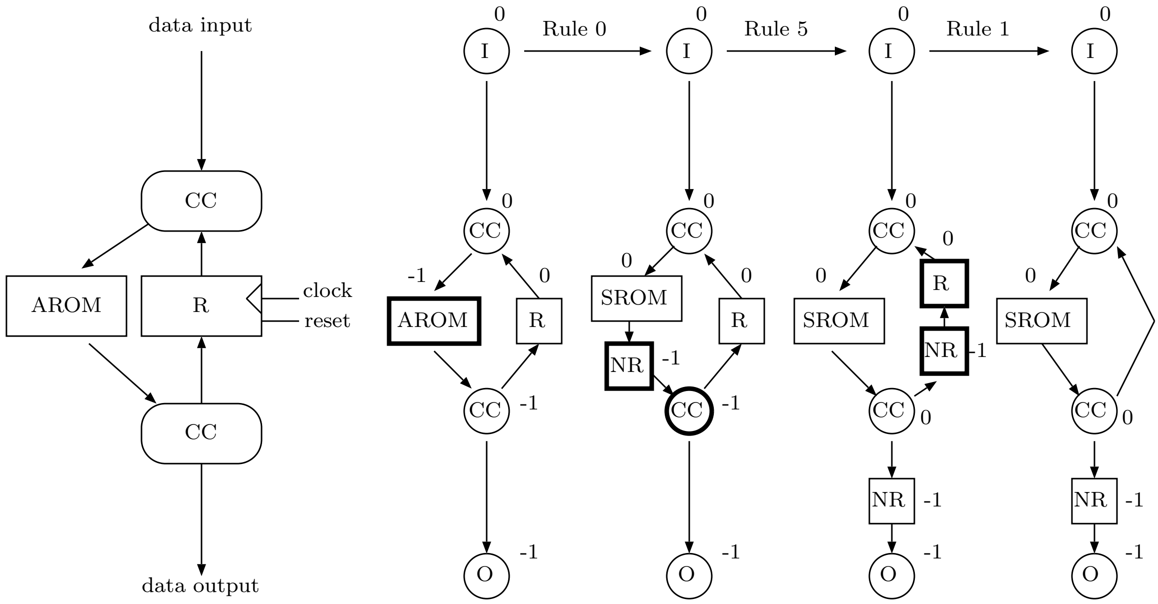


Figure 13: A circuit and its corresponding circuit graph with a slight modification of the Figure 10 that can not be converted into an AROM-free circuit

6 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.

First, 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, 2 and 4. 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 5. 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 $p(S) = p(P) + 1$.

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

Rule 4 $p(S) = p(P) - 1$.

Rule 5 $p(S_1) = p(S_2) = \min(p(P_1) - 1, p(P_2), p(P_3)) = \min(p(P_1), p(P_2) + 1, p(P_3) + 1) - 1$.

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.*

For this lemma, the readers may see the Figure 10. In this figure, the potentiality of the O node is 0 and this value is never changed. Similarly, we can prove the following lemma:

Lemma 3 *The potentiality of every cycle of the resulting circuit graph is the same as that of the corresponding cycle of the original circuit graph.*

In Figure 10, we see that the cycle increases the potentiality by +1 and this value is also never changed. Readers may refer to the Figure 12 for making clear about the potentiality of the cycles in circuits.

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

Lemma 4 *Once our circuit rewriting algorithm uses either Rule 4 or Rule 5 to move an NR node, it never applies Rule 2 and Rule 3 to move an R node.*

Proof If either Rule 4 or Rule 5 is applied an interim circuit, both Rule 2 and Rule 3 cannot be applied to it. If this is the case, all Rs are either (1) in the segment of Rs ending at an O node, or (2) in the segment of Rs ending at a CC node and another incoming edge of the CC node is not connected to R (Figure 14). To apply Rule 2 and Rule 3 later, the non-R node in Figure 14 must be an R node. However, to be an R node, Rule 2 and Rule 3 must be used. Thus, both Rule 2 and Rule 3 are never applied.

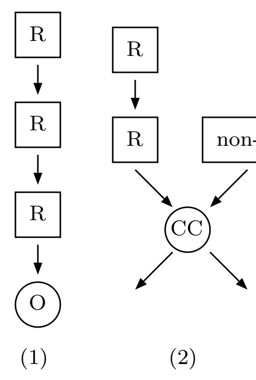


Figure 14: Illustration for the proof of Lemma 4

We will prove that all NRs in a cycle with non-negative potentiality will be removed by our rewriting algorithm.

Lemma 5 *Suppose that all cycles in a circuit graph have non-negative potentiality, and Rule 0 are repeatedly applied to remove all AROMs. If a cycle has m NRs, it also has at least m Rs. If either Rule 2 or Rule 3 is applied, the Rs are moved and adjacent R and NR may be removed by Rule 1. If either Rule 4 or Rule 5 is applied, the NRs are moved. Note that, from Lemma 4, the Rs are never moved, once either Rule 4 or Rule 5 is applied. In other words, the NRs are moved along the cycle, while Rs are never moved. Thus, at some point, all NRs in the cycle will be removed by Rule 1.*

Note that, if there exists a cycle with negative potentiality, our circuit rewriting algorithm does not terminate. As illustrated in Figure 11, an NR moves along the cycle and Rs and NRs are repeatedly generated. It should be clear that, there exists no way to generate an equivalent AROM-free circuit for such circuit.

When our rewriting algorithm terminates and the resulting circuit graph is obtained, we have the following lemma:

Lemma 6 *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, 4, or 5 can be applied. Since no more rules 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.

The reader may refer to Figure 13 for making clear about the proof of this lemma. In this figure, the resulting circuit graph (circuit graph in where no rule is applicable) has an NR which is in segment ending at O node.

A simple directed path is a directed path if it has no repeated nodes. For example, in Figure 2 (2), (B, E, H, K, N, O) is a simple directed path, but $(B, E, H, I, F, E, H, K, N, O)$ is not. We say that nodes are *regular* if it is on a simple directed path from an input node to an output node. Note that nodes on a cycle in a DRG can be a non-regular node. For example, nodes F and I are non-regular nodes.

From Lemma 6, we will prove that all regular SROM and CC nodes in the resulting circuit graph have zero potentiality.

Lemma 7 *All regular SROM and CC nodes in the resulting circuit graph have non-negative potentiality.*

Proof Since the resulting graph is AROM-free, nodes follows 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 8 *All regular SROM and CC nodes in a simple directed path from an input node to an output node in the resulting circuit graph have non-positive potentiality.*

Proof We assume that the resulting circuit graph has a positive potentiality SROM or CC node in a simple directed path from an input node to an output node, 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) denotes the incoming edge. If u is either R or NR, then Rule 2 or Rule 4 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 6, none of $u_1, u_2, \ldots, u_k$ is an NR node. If all of them are R nodes, then Rule 3 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 7 and 8, all SROM and CC nodes in a simple directed path from an input node to an output node of 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 6. 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, from Lemma 5, if all output nodes and all cycles of a circuit graph have negative potentiality 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 generate 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 and all cycles 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. In this case, we assume that all the cycles have non-negative potentiality. 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 Figure 15 for illustrating an example. Note that, the resulting circuit graph is not an equivalent to the original circuit graph. However, the difference is the latency of the output node. Thus, we can say that, the resulting AROM-free circuit is an almost equivalent to the original circuit.

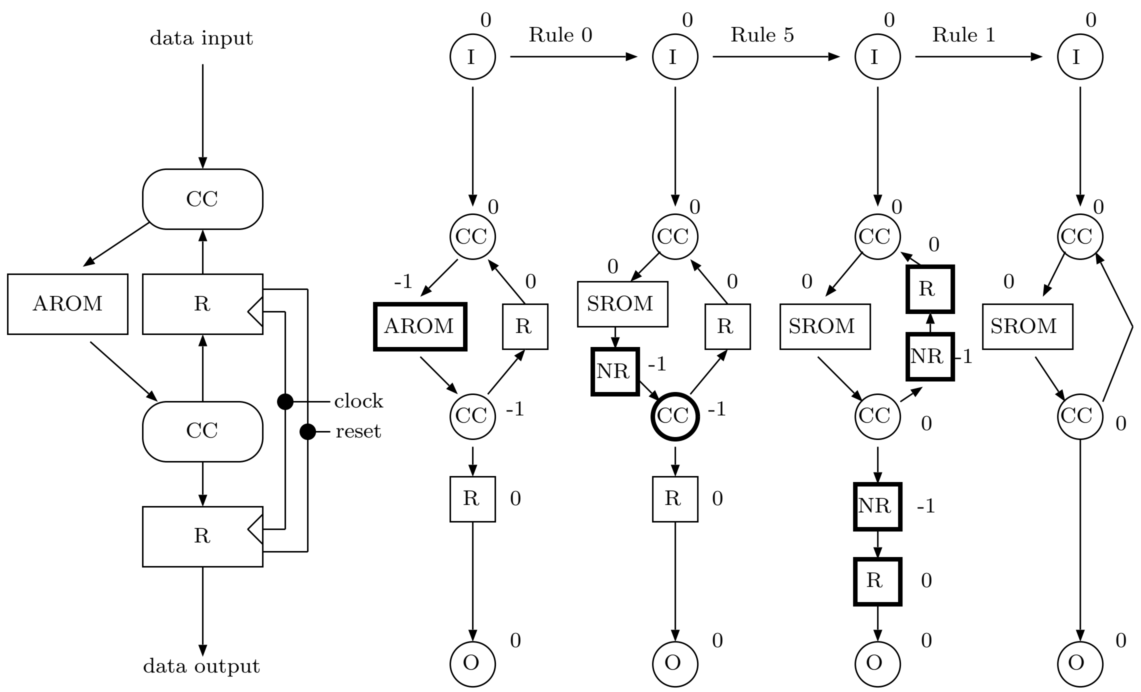


Figure 15: An almost equivalent circuit with corresponding circuit graph to that of Figure 13 that can be converted into an AROM-free circuit

As we have discussed, our circuit rewriting algorithm does not terminate for a circuit graph with a negative cycle. We can modify our circuit rewriting algorithm that always terminates as follows: First, we compute the potentiality of every cycles. If one of them is negative, we do not execute our circuit rewriting algorithm. Since it is impossible to generate an equivalent AROM-free circuit if this is the case, it is not reasonable to execute our circuit rewriting algorithm.

7 How to handle nodes that are not in a path from an input node

In this section, we will describe for understanding how to handle nodes corresponding circuit elements that are not in a path from an input of the circuits. For this purpose, we include a no input practical circuit such as counter in conjunction with DRG circuit as a designed input circuit instead of DRG circuit only. By this addition, in fact, we relax a restriction to the designed circuit by users in terms of input circuits. However, we assume that our no input practical circuit has no memory elements such as ROMs. It consists of Registers (Rs) and Combinational Circuits (CCs).

For the benefit of readers, we will show an example of a no input practical circuit as illustrated in Figure 16 (a). The circuit in Figure 16 (a) has one Register (R) and one adder. Register (R) has a reset input and a clock input as illustrated in Section 3. Readers may also refer to the Section 3 for details about Combinational Circuit (CC). Initially stored data value in R is 0 if reset is 1. When reset is 0, then stored data value is updated by the data value given to the input port at every rising clock edge.

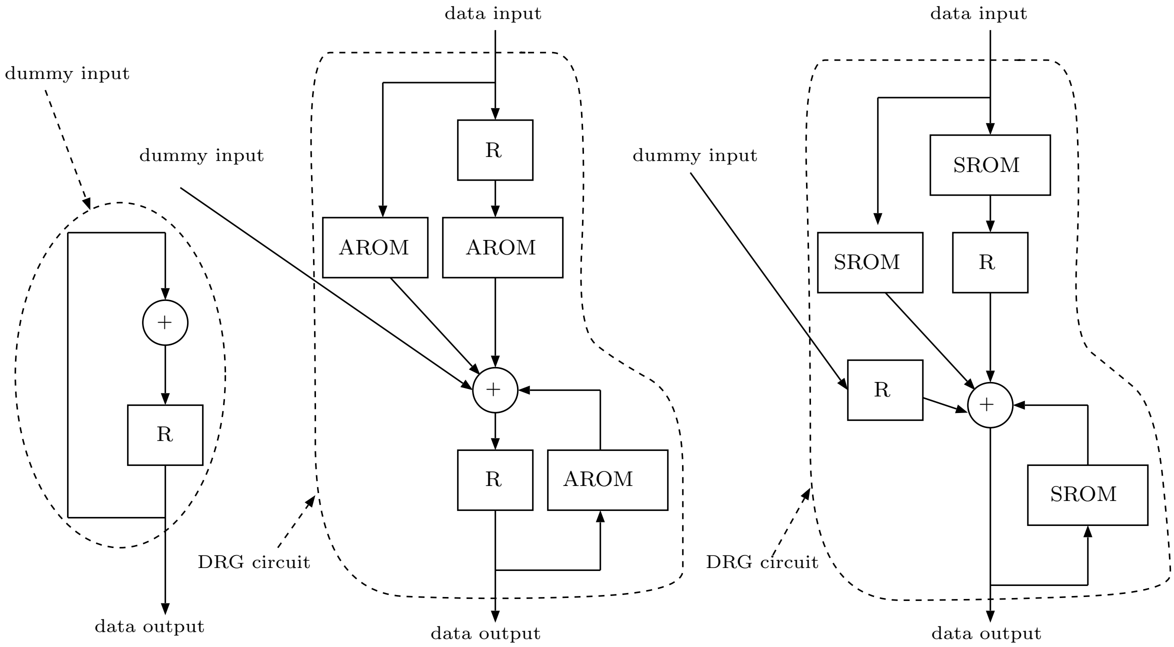
Let us recall the circuit, shown in Figure 16 (a). In this figure, we see that we may have output data sequence 0, 1, 2, ... of the time 0, 1, 2, ..., respectively. If this is the case, then we say that the output sequence of the circuit as shown in Figure 16 (a) is deterministic which is similar to other inputs of the DRG circuit. Hence, we treat this citcuit as illustrated in Figure 16 (a) as a dummy input to the DRG circuit, as shown in Figure 16 (b). Readers may refer to Figure 16 (a), where dotted circle is indicating the dummy input for the DRG circuit as shown in Figure 16 (b) in which the dummy input is connected to the adder of the DRG circuit. Note that, in Figure 16 (b), DRG circuit is shown by enclosed dotted line. If this is the case, then we consider whole circuit in Figure 16 (b) as an input circuit for our algorithm. Since, the dummy input can be treated as the same as other inputs to the DRG circuit, our rewriting algorithm is applied to the whole circuit, instead of only considering DRG circuit, as illustrated in Figure 16 (b) by enclosed dotted line. For the benefit of readers, we have shown an application of our rewriting algorithm in Figure 16 (c). Figure 16 (c) represents a converted circuit (by our rewriting algorithm) with no AROMs for the circuit, shown in Figure 16 (b). It is noted that one Register (R) is generated to the connecting edge from the dummy input to the adder (CC) of the DRG circuit by our algorithm, shown in Figure 16 (c). Obviously, we can conclude from the converted circuit in Figure 16 (c) for an input circuit in Figure 16 (b) that users can design their input circuit in wider range instead of only considering DRG circuit, shown in Figure 16 (b) by enclosed dotted line.

8 Conclusions

We presented a rewriting algorithm and six rewriting rules to obtain the equivalent circuits with Synchronous ROMs (SROMs) for the practical circuits with Asynchronous ROMs (AROMs). The practical sequential circuit with AROMs represented by a directed reachable graph (DRG) can be converted by our rewriting algorithm into an equivalent fully synchronous sequential circuit with no AROMs to support the architecture of the most FPGAs. In this paper, we also described a technique to extend the input designed circuits by users in wider range rather than DRG circuits. It is not trivial to convert the practical sequential circuits with AROMs into the equivalent fully synchronous circuits with no AROMs for supporting the modern FPGA architecture. However, our algorithm can do it automatically.

References

[1] Song Bo, Kensuke Kawakami, Koji Nakano, and Yasuaki Ito. An RSA encryption hardware algorithm using a single DSP block and a single block RAM on the FPGA. *International Journal of Networking and Computing*, pages 277–289, July 2011.



(a) An example of no input practical circuit

(b) A no input practical circuit as a dummy input to DRG circuit

(c) The converted circuit with no AROMs for the input circuit in (b)

Figure 16: An example to extend the input circuit for our algorithm

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

[3] J. L. Bordim, Y. Ito, and K. Nakano. Instance-specific solutions to accelerate the CKY parsing for large context-free grammars. *International Journal on Foundations of Computer Science*, 15(2):403–416, 2004.

[4] S. Scott 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):672–683, June 2007.

[5] Y. Ito and K. Nakano. FM screening by the local exhaustive search, with hardware acceleration. *International Journal of Foundations of Computer Science*, pages 89–104, February 2005.

[6] Yasuaki Ito and Koji 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] Yasuaki Ito and Koji Nakano. Low-latency connected component labeling using an FPGA. *International Journal of Foundations of Computer Science*, pages 405–426, June 2010.

[8] Yasuaki Ito and Koji Nakano. Efficient exhaustive verification of the Collatz conjecture using DSP blocks of Xilinx FPGAs. *International Journal of Networking and Computing*, pages 49–62, January 2011.

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

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

[11] R. Manohar. Reconfigurable asynchronous logic. In *Proc. of IEEE Custom Integated Circuits Conference*, pages 13–20, 2006.

[12] Md. Nazrul Islam Mondal, K. Nakano, and Y. Ito. A graph rewriting approach for converting asynchronous roms into synchronous ones. *IEICE Transactions on Information and Systems*, E94-D(12):2378–2388, December 2011.

[13] K. Nakano and E. Takamichi. An image retrieval system using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):811–818, May 2003.

[14] K. Nakano and Y. Yamagishi. Hardware n choose k counters with applications to the partial exhaustive search. *IEICE Transactions on Information and Systems*, E88-D(7):1350–1359, July 2005.

[15] 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.

[16] 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.

[17] 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.

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