Graph Rewriting Approaches to Convert Asynchronous Read Operation into Synchronous One for Embedded Memory in FPGAs

by

Md. Nazrul Islam Mondal

A dissertation submitted
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Information Engineering

Under Supervision of
Professor Koji Nakano

Department of Information Engineering,
Graduate School of Engineering,
Hiroshima University

September, 2012

SUMMARY

Recently, FPGAs (Field Programmable Gate Arrays) are widely used to implement algorithms with circuits for accelerating computation. Circuit design that minimizes the number of clock cycles is easy if we use asynchronous read operation. However, embedded blocks of memories in the most modern FPGAs support synchronous read and synchronous write operations, but do not support asynchronous read operation. Hence, we can not implement circuits with memories supporting asynchronous read operation in FPGAs. Because of the above background, this dissertation shows circuit rewriting algorithms to convert circuits with memories supporting asynchronous read operation into the equivalent circuits with memories supporting synchronous read operation for implementing in FPGAs, as follows:

In the domain of digital circuit design, one of the most important tasks is to make circuit design easy to the designers. We say that circuit design that minimizes the number of clock cycles is easy if we use asynchronous read operation. However, embedded block RAMs (Random Access Memories) of the most FPGAs support synchronous read and synchronous write operations, but do not support asynchronous read operation. To resolve this problem, the first main contribution of this dissertation is to present a *circuit rewriting algorithm* which is used to convert a designed circuit with AROMs (Asynchronous Read Only Memories) into an equivalent circuit with SROMs (Synchronous Read Only Memories). More specifically, a circuit using AROMs supporting asynchronous read operation designed by a non-expert or quickly designed by an expert is given. Our *circuit rewriting algorithm* converts this circuit with AROMs into an equivalent circuit with SROMs supporting synchronous read operation automatically. Finally, the resulting circuit with SROMs can be embedded into FPGAs.

Circuits designed by users may have ARAMs (Asynchronous RAMs) supporting *both* asynchronous read and synchronous write operations instead of AROMs supporting asynchronous read operation *only*. However, embedded block RAMs of the most modern FPGAs support synchronous read and synchronous write operations but do not support asynchronous read operation. The second main contribution is to present a *circuit rewriting algorithm* to resolve this problem. Therefore, presented *circuit rewriting algorithm* which is devoted to convert automatically a circuit using ARAMs into an equivalent circuit with SRAMs (synchronous RAMs) supporting *both* synchronous read and synchronous write operations in order to embed the resulting circuit in FPGAs.

Many practical circuits, designed by users may have *cycles*. However, *circuit rewriting algorithms*, presented as the first and second main contributions are unable to process those circuits with *cycles*. Our third main contribution of this dissertation is to propose a *modified circuit rewriting algorithm* which is able to process circuits with *cycles*. More specifically, a circuit with *cycles* using AROMs designed by users is given. Our *modified circuit rewriting algorithm* converts it into an equivalent circuit with *cycles* using SROMs. Finally, the resulting circuit with *cycles* using SROMs can be embedded into FPGAs.

By our *circuit rewriting algorithms*, most of the registers move towards the output ports, whenever possible. Hence, in general, the resulting circuits may have the longest paths from input ports to registers/SROMs/SRAMs or from registers/SROMs/SRAMs to registers/SROMs/SRAMs or from registers/SROMs/SRAMs to output ports. Performance of the resulting circuits therefore may be degraded in terms of *latency* and *clock frequency*. However, it is easier to improve the performance of the resulting circuits than minimizing the number of clock cycles by the designers.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

In the context of accelerating computation, many parallel and distributed computing methods are proposed by the researchers. One of the widely used parallel computing methods with a multi-core and/or multi-processor computer is presented in [6] and other one [5], is presenting the distributed computing method using a computer cluster. On the other hand, FPGAs (Field Programmable Gate Arrays) are a programmable VLSI which can be used for implementing parallel and hardware algorithms. As a device for low cost pseudo-specialized LSI, FPGAs are attracting attention to the researchers. With the development of LSI device fabrication, FPGAs are chosen by the researchers for implementing their applications. For the technical improvement of LSI production, their size, capabilities, and speed have been increased. Compared with parallel and distributed computing, the computation granularity of computation using FPGAs would be the finest. Considering programmability, FPGAs can be considered as hardware that has an ability of software. Recently, FPGAs are widely used to implement algorithms

with circuits for accelerating computation. Circuit design that minimizes the number of clock cycles is easy if we use asynchronous read operation. However, embedded blocks of memories in the most modern FPGAs support synchronous read and synchronous write operations, but do not support asynchronous read operation. Hence, we can not implement circuits with memories supporting asynchronous read operation in FPGAs. Because of the above background, we are inspired to present circuit rewriting algorithms to convert circuits with memories supporting asynchronous read operation into the equivalent circuits with memories supporting synchronous read operation for implementing in FPGAs. For this purpose, this dissertation shows circuit rewriting algorithms which are as follows:

A Circuit Rewriting Algorithm for Converting Asynchronous ROMs into Synchronous Ones for FPGAs

A *circuit rewriting algorithm* that is used to rewrite a given circuit with AROMs (Asynchronous Read Only Memories) until an equivalent circuit with SROMs (Synchronous Read Only Memories) is generated for implementing in FPGAs. More specifically, a circuit, X with AROMs is given. Our *circuit rewriting algorithm* generates a circuit, Y with SROMs which is an equivalent to X with AROMs for implementing in the current FPGAs. 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 operation. However, embedded memories of the most FPGAs support synchronous read and synchronous write operations, but do not support asynchronous read operation. Hence, the main contribution of this chapter

is to present a potent circuit rewriting approach to resolve this problem. We assume that a circuit using asynchronous ROMs (AROMs) designed by a non-expert or quickly designed by an expert is given. Our goal is to convert this circuit with asynchronous ROMs into an equivalent circuit with synchronous ones (SROMs) automatically. Finally, the resulting circuit with synchronous ROMs can be embedded into FPGAs. We briefly discuss the techniques to improve performance of the AROM-free resulting circuit and also describe a technique for applying our rewriting algorithm even if a user designs a circuit with pipeline structure.

A Circuit Rewriting Algorithm to Obtain Circuits with Synchronous RAMs for FPGAs

A *circuit rewriting algorithm*, presented in this chapter is devoted for converting a circuit with RAMs supporting asynchronous read and synchronous write operations (ARAMs) into an equivalent circuit with RAMs supporting synchronous read and synchronous write operations (SRAMs); more specifically, a circuit using asynchronous RAMs (ARAMs) designed by a non-expert or quickly designed by an expert is given. This rewriting algorithm converts it into an equivalent circuit using synchronous RAMs (SRAMs) for implementing in FPGAs. In our previous work, mentioned earlier (followed by the Chapter 3), we considered *only read operation* of the memory blocks (ROMs). Particularly, presented circuit rewriting algorithm was used to convert a circuit with AROMs into an equivalent circuit with SROMs. The resulting circuit can be embedded into FPGAs. However, this circuit rewriting algorithm, presented in this chapter considers *both read and write operations* of the memory blocks (RAMs). In fact, we improved our previous research work, where RAMs can be used as the additional circuit

elements to the given input circuits. The conversion of a sequential circuit with ARAMs into an equivalent fully synchronous circuit with no ARAMs for supporting the modern FPGA architecture is not trivial. However, our algorithm can do it automatically. We also briefly discuss the techniques to improve performance of the ARAM-free resulting circuit.

A Modified Circuit Rewriting Algorithm for the Circuits with Cycles

A *modified circuit rewriting algorithm* is used to convert a circuit *with cycles* using AROMs into an equivalent circuit using SROMs for implementing in FPGAs. The main contribution of this chapter is to consider a given circuit *with cycles* using AROMs. Particularly, our new circuit rewriting algorithm can be used to convert circuits which have cycles. In our previous works, mentioned above (followed by the Chapter 3 and Chapter 4), we have presented circuit rewriting algorithms to convert a circuit with asynchronous ROMs or asynchronous RAMs into an equivalent circuit with synchronous ones. The resulting circuit with synchronous ROMs or synchronous RAMs can be embedded into FPGAs. However, these circuit rewriting algorithms can handle circuits represented by a directed acyclic graph (DAG) and do not work for those with cycles. By the work in this chapter, 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. We also briefly discuss the techniques to improve performance of the AROM-free resulting circuit.

Performance Improvement of the Resulting Circuits

In this chapter, we mainly discuss about the performance improvement of the AROM-free and ARAM-free resulting circuits. Basically, our rewriting algorithms move registers towards the output ports, whenever possible. Hence, in general, the resulting circuits may have the longest paths from input ports to registers/SROMs/SRAMs or from registers/SROMs/SRAMs to registers/SROMs/SRAMs or from registers/SROMs/SRAMs to output ports. Therefore, the resulting AROM-free or ARAM-free circuit has large propagation delay and low clock frequency. Hence, we say that performance of the resulting AROM-free or ARAM-free circuit may be degraded in terms of latency and clock frequency. However, it is easier to improve the performance of the resulting circuits than minimizing the number of clock cycles. For the reader's benefit, performance improvement techniques in terms of latency and clock frequency of the resulting circuits are described in Chapter 6, although these are *beyond of this dissertation*.

1.2 Dissertation Organization

This doctoral dissertation is organized as follows: The background with motivation and the introduction of this dissertation are presented in Chapter 1. In Chapter 2, we briefly introduce an FPGA. Chapter 3 describes a *circuit rewriting approach* to convert a circuit with AROMs into an equivalent circuit with SROMs for implementing in FPGAs. Chapter 4 describes a *circuit rewriting approach* to convert a circuit with ARAMs supporting asynchronous read and synchronous write operations into an equivalent circuit with SRAMs supporting synchronous read and synchronous write operations. In Chapter 5, a *modified circuit rewriting approach* is described to convert a circuit with cycles

using AROMs into an equivalent circuit with cycles using SROMs for implementing in FPGAs. Techniques to improve performance of the resulting circuits are described in Chapter 6. Finally, this dissertation is concluded in Chapter 7.

Chapter 2

FPGA

An FPGA (Field Programmable Gate Array) is a programmable VLSI in which a hardware designed by users can be embedded instantly. In mid-1980's, the first commercially viable FPGA has been invented. At first, since the size of a hardware that can be embedded on an FPGA was small and the speed was slow. Hence, applications of FPGAs were limited to a specific area, such as digital signal processing, prototyping ASIC, and computer hardware emulation. With technical improvement of LSI production, however, their size, capabilities, and speed have been increased. Recently, because of their flexibility, they are widely used for various applications.

2.1 FPGA Architecture Basics

Typical FPGAs consist of an array of programmable logic elements, distributed memory blocks, embedded multipliers, Input/Output Block and programmable interconnections between them. Figure 2.1 shows an example of FPGA hardware fabric. The logic block usually contains either a logic function or a multiplexer and several flip-flops. An exam-

ple of a logic block is illustrated in Figure 2.2. The distributed memory block is usually a dual-port RAM (Random Access Memory) on which a word of data for possibly distinct addresses can be read/written at the same time. Note that, a RAM can be treated as ROM by selecting the write enable input of a RAM as low. The embedded multipliers can compute multiplication much faster than multipliers that consists of logic elements. The user's hardware logic design can be embedded into the FPGAs using the design tools supplied by the FPGA vendors. Using design tools supplied by FPGA vendors, a hardware logic designed by users can be embedded into the FPGAs. However, designed circuits by users with AROMs (Asynchronous ROMs) supporting asynchronous read operation or ARAM (Asynchronous RAMs) supporting asynchronous read and synchronous write operations can not be implemented in FPGAs, because, embedded blocks of memories in FPGAs support synchronous read and synchronous write operations. Hence, our goal is to provide circuit rewriting algorithms that can convert circuits with AROMs or ARAMs into the equivalent circuits with synchronous ones for implementing in FPGAs. In particular, circuits consist of registers (Rs), combinational circuits (CCs) and AROMs or ARAMs are converted automatically by our circuit rewriting algorithms into the equivalent circuits having registers (Rs), combinational circuits (CCs) and SROMs or SRAMs for implementing in FPGAs.

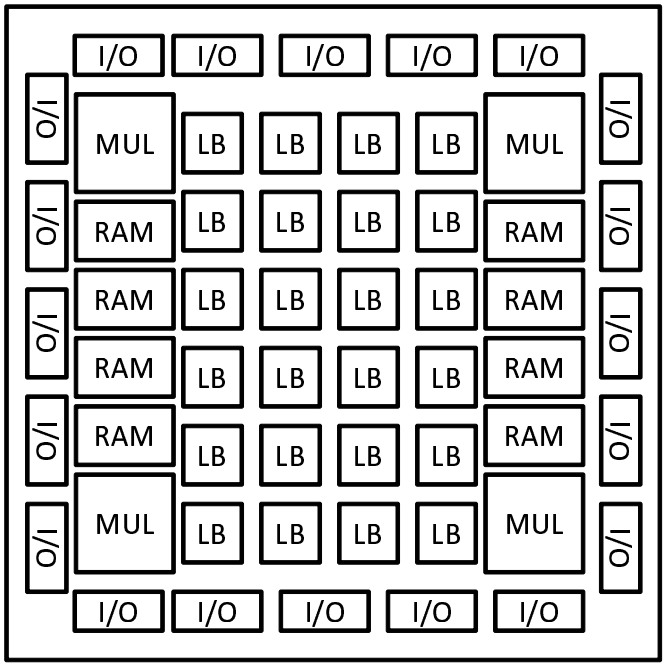


Figure 2.1: An example of FPGA hardware fabric.

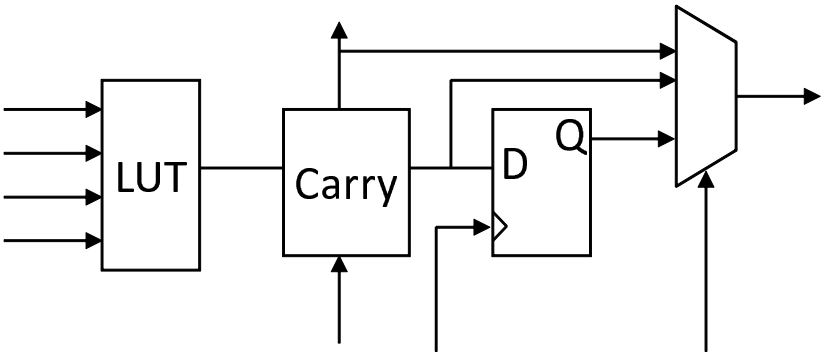


Figure 2.2: An example of a logic block.

Chapter 3

A Circuit Rewriting Algorithm for Converting Asynchronous ROMs into Synchronous Ones for FPGAs

The main contribution of this chapter is to present a circuit rewriting algorithm to convert a circuit with AROMs supporting asynchronous read operation into an equivalent circuit with SROMs supporting synchronous read operation for implementing in FPGAs. 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). We say that circuit design that minimizes the number of clock cycles is easy if we use asynchronous read operations. However, embedded memories of the most FPGAs support synchronous read and synchronous write operations, but do not support asynchronous read operations. This chapter provides one of the potent approaches to resolve this problem. We assume that a circuit using asynchronous ROMs designed by a non-expert or quickly designed by an expert

is given. Our goal is to convert this circuit with asynchronous ROMs (AROMs) into an equivalent circuit with synchronous ones (SROMs) automatically. The resulting circuit with synchronous ROMs can be embedded into FPGAs. We briefly discuss the techniques to improve performance of the AROM-free resulting circuit and also describe a technique for applying our rewriting algorithm even if a user designs a circuit with pipeline structure.

3.1 Introduction

Our approach, presented in this chapter is devoted to convert a circuits with AROMs into an equivalent circuit with SROMs for implementing in FPGAs. An FPGA is a programmable VLSI (Very Large Scale Integration) in which a hardware designed by users can be embedded quickly. FPGAs may implement hundreds of circuits that work in parallel. Therefore, they are used to accelerate useful computations. Some circuit implementations in FPGAs are described [1, 2, 9, 14] to accelerate computation. For example, a parallel implementation [9] for the exhaustive verification of the Collatz conjecture has been presented. In this implementation, 24 co-processors embedded in a Xilinx Virtex-2 Family FPGA perform the exhaustive verification in parallel.

In this chapter, we mainly focus the following asynchronous and synchronous read operations of memory blocks.

3.1.1 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.

3.1.2 Synchronous read operation

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

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

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

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

into *an equivalent synchronous circuit* consisting

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

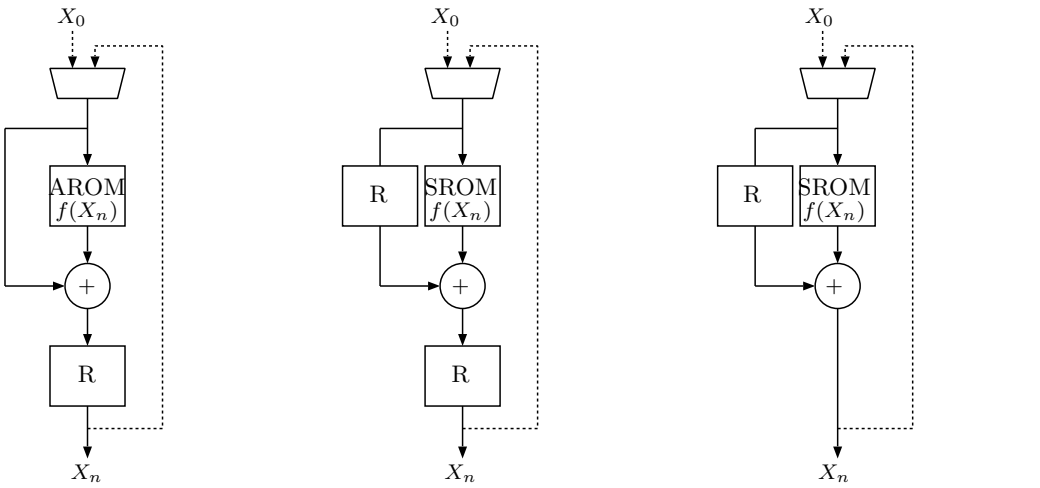
Note that, most of the current FPGAs support synchronous read operation, but do not

support asynchronous one. We are thinking the following scenario to use our circuit rewriting algorithm:

- An asynchronous circuit designed by a non-expert, or quickly designed by an expert is given.
- Our circuit rewriting algorithm 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 $n \geq 1$. 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 3.1(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 3.1(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 3.1(b) needs double clock cycles over the circuit in Figure 3.1(a). Using our algorithm to the sub-circuit with solid lines (wires) in Figure 3.1(a), we can obtain the circuit in Figure 3.1(c) automatically. In the circuit with an SROM in Figure 3.1(c), $X_1, X_2, \ldots$ is output in every clock cycle. Thus, the timings of the circuits in Figure 3.1(a) and (c)



(a) A circuit with an AROM (b) A circuit with an SROM by a non-expert (c) The converted circuit with an SROM

Figure 3.1: An example of circuits using an AROM and an SROM.

are identical.

It is not trivial for the non-expert designers to minimize the number of clock cycles to obtain circuit as illustrated in Figure 3.1(c). However, our algorithm can do it automatically. Although, clock performance may degrade in the converted circuit due to the moving of registers (Rs) towards the output ports by our algorithm; however, designers can make a trade-off between the maximum delay and number of clock cycles for their designs. The readers should refer to Figure 3.2 for an illustration. In Figure 3.2, the number of clock cycles is increased as well as the maximum delay is decreased by inserting the pipelined registers. In general, the insertion of the pipelined registers is not difficult. On the other hand, our algorithm may decrease the number of clock cycles by removing redundant registers. Although it may increase the maximum delay, sometimes the resulting circuit takes smaller total computing time. Readers may refer to Chapter 6 for details about performance improvement techniques.

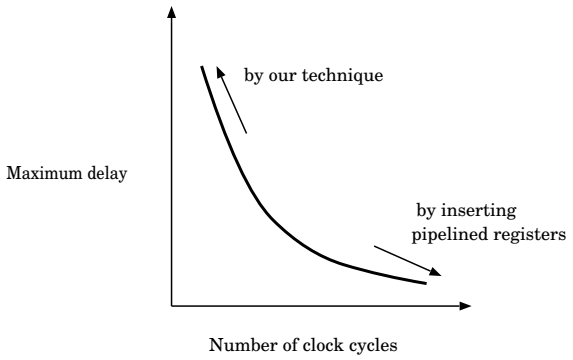


Figure 3.2: A relation between the maximum delay and the number of clock cycles.

The outlines of our new idea are as follows:

1. We introduce *a negative register* (NR), which is an imaginary register latching a future input data.

2. We define simple *five rules* that rewrite a circuit.

3. The rewriting algorithm that we propose just repeats applying these rules until no more rules can be applied. When the rewriting algorithm terminates, we have an equivalent AROM-free circuit to the original circuit.

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

A circuit implementation with AROMs is better than SROMs implementation, because of less power consumption, easy to design etc. But it has some problems like small in size so that it does not support the designer's demand, more expensive, and less

speedy [3, 10, 11]. To cut the clock distribution power, an asynchronous circuit design in FPGAs is very much suitable, described in [13, 8, 19]. 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 [3, 11]. Therefore, we should use SROMs when we need a function of ROMs.

One of the research works described the implementation of asynchronous circuit in FPGA [16]. In this research work, they described the problems like hazards, timing constraints, state holding elements, analog components and decomposition of the asynchronous circuit implementation in FPGA. Another research work described a novel FPGA architecture for implementing various styles of asynchronous logic [7]. They implemented a full-adder circuit in two different logic styles. While in synchronous circuits a clock globally controls the activity where as asynchronous circuit activity is locally controlled using communication channels to detect the presence of data at their inputs and outputs. An asynchronous module communicates with each other using requests and acknowledges [17]. Some dedicated FPGAs have also been developed to test asynchronous designs. Unfortunately, these FPGAs are closely associated to a style of design. For instance, PGA-STC [12] and MONTAGE [16] are based on an asynchronous design, GALSA [4] and STACC [15] are globally asynchronous FPGAs but locally synchronous and PAPA [18] is a fully asynchronous FPGA dedicated to optimize pipeline circuits.

To the best of our knowledge, there is no previous research work on our topic. It is well known that the architecture of the current FPGAs is the best suited for digital

synchronous circuit designs. Unfortunately, they do not have block RAMs supporting asynchronous read operations. It is also known that AROM is implemented in LUTs which is easy to use because of the immediate output of data. However, it is small in size and costly. Therefore, our target is to generate an AROM-free fully synchronous sequential circuit from a sequential circuit with AROMs which is an equivalent to the original circuit so that it can support the modern FPGA architecture.

We summarize several significant points of our results as follows:

- Negative registers (NRs) are newly introduced as a key and innovative idea. Further, the correctness of our algorithm is proved in a rigorous manner.

- 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 will discuss a technique to minimize latency of the resulting circuits in Chapter 6.

- Clock performance may degrade in the resulting circuit by our rewriting algorithm. However, we can improve the clock performance by inserting registers appropriately, although this is beyond of this dissertation. For the benefit of readers, a technique to improve clock performance of the resulting circuit will be discussed in Chapter 6.

- 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 opera-

tion is not necessary, because it can be automatically converted into synchronous one using our algorithm.

- The readers may think that circuits dealt with this chapter are too restricted where as circuits in real-world are more complicated. However, it may be possible to extract a sub-circuit from the complicated circuit. We can then apply our circuit rewriting algorithm to this sub-circuit.

- Even if a user designs a circuit with pipeline structure, our algorithm moves pipeline registers towards the output ports and destroys the pipeline structure. However, it may be possible to perform AROM-free conversion locally without collapsing a global pipeline structure. For this purpose, we need to extract sub-circuits in the original circuit such that it contains no pipeline register. By using our algorithm for each sub-circuit, it can be converted into an AROM-free circuit. Since the timing of each sub-circuit is not changed, the whole converted circuit is identical to the original circuit. In this way, our algorithm may be applicable to the pipelined circuits.

This chapter is organized as follows: Section 3.2 briefly describes the circuits and their equivalence. In Section 3.3, we describe our rewriting algorithm, circuit graph and also explain the equivalence for our rewriting rules. Section 3.4 presents the proof of the correctness of our rewriting algorithm. Finally Section 3.5 concludes this chapter.

3.2 Circuits and Their Equivalence

This section briefly describes the circuit elements such as combinational circuit (CC), registers (R), read only memory (ROM) and their equivalence. Let us consider a se-

quential 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). The following subsections describe the circuit elements such as combinational circuit (CC), register (R), read only memories (ROMs).

3.2.1 Combinational Circuit (CC)

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

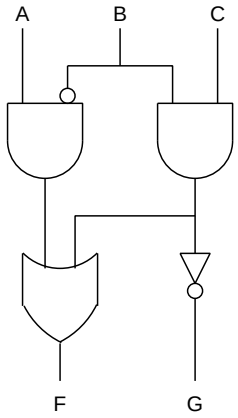


Figure 3.3: An example of a combinational circuit (CC).

3.2.2 Register (R)

Register is a memory element that can store data or information. A b-bit register has a clock input and a reset input. It can store a b-bit data as shown in Figure 3.4. If reset is 1, then the b-bit data is initialized by 0. If reset is 0, the stored data is updated by the

value given to the input port *d* at every rising clock edge. The data stored in the register is always output from port *q*.

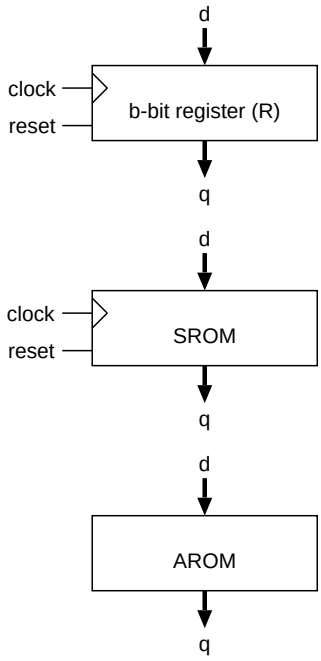


Figure 3.4: A register (R), a synchronous ROM (SROM) and an asynchronous ROM (AROM).

3.2.3 ROM (Read Only Memory)

ROM is also known as a memory element where data or information can be stored permanently. A ROM (Read Only Memory) has a *b*-bit input *d* and a *c*-bit data output *q*. It is storing 2^b words such as $M[0]$, $M[1]$, ..., $M[2^b - 1]$ with *c* bits each. We deal with two types of ROMs in terms of read operations as follows:

- **Synchronous ROM (SROM)** An SROM has a clock input and a reset input. If reset is 1 then the stored value is initialized by 0. The read operation is performed

at 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 3.5 shows a timing diagram of reading operations of the R, SROM, AROM and NR. In the figure, time 0, 1, 2, ... correspond to rising edges of the periodic clock input. Initially global reset is 1 and it drops to 0 just before time 0. Data d_0, d_1, d_2, ... are given to the input port d. As shown in the figure, the value of output, q of R and SROM is 0 at time 0. Also, at time 1, 2, ... the values of output, q of R and SROM are d_0, d_1, d_2, ... and $M[d_0]$, $M[d_1]$, $M[d_2]$, ..., respectively. For the AROM, the data $M[d_0]$, $M[d_1]$, $M[d_1]$, ... are taken from the output port, q immediately at time 0, 1, 2, ..., respectively.

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

We will describe a behavior of a circuit element using a sequence of output at every rising clock edge for 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 3.5. The behavior of each circuit element is described by the output sequences as follows:

- **Combinational Circuit (CC)** For simplicity, we assume 3-input 2-output combinational circuit which is shown in Fig. 3.3. There is no difficulty to extend the definition for general m-input n-output combinational circuit. We assume that, at

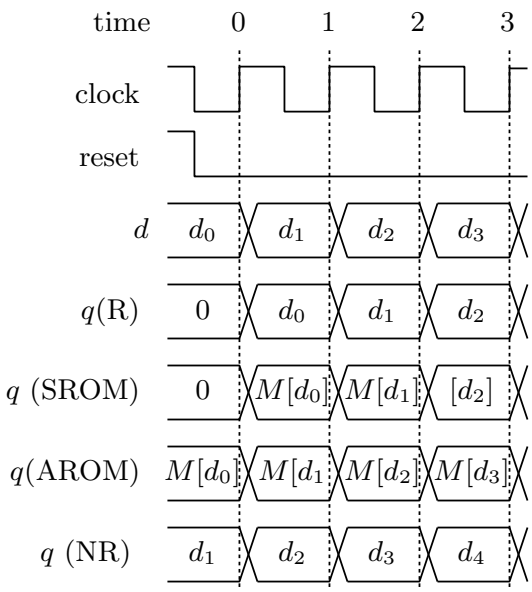


Figure 3.5: A timing chart of a register (R), an SROM, an AROM and a negative register (NR).

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 chapter, we assume that a fully synchronous circuit has data inputs, data outputs, a global clock input, a global reset input, combinational circuits (CCs), registers (Rs), SROMs, AROMs, and their interconnects. The readers should refer to the Figure 3.6 for illustrating an example of a fully synchronous circuit. The global clock and the global reset are directly connected to the clock input ports and the reset input ports of all Rs and SROMs. Also, we assume that a circuit has no loop.

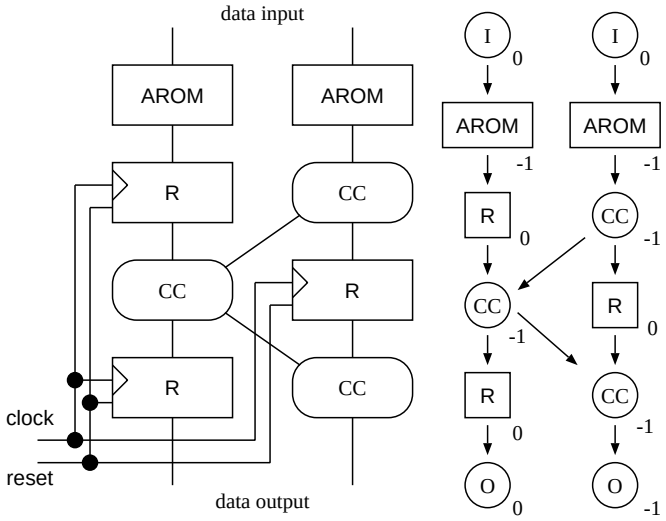


Figure 3.6: An example of a fully synchronous circuit and the corresponding circuit graph with potentiality.

Let us define *equivalence* of two fully synchronous circuits for the periodic clock and initial reset. 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

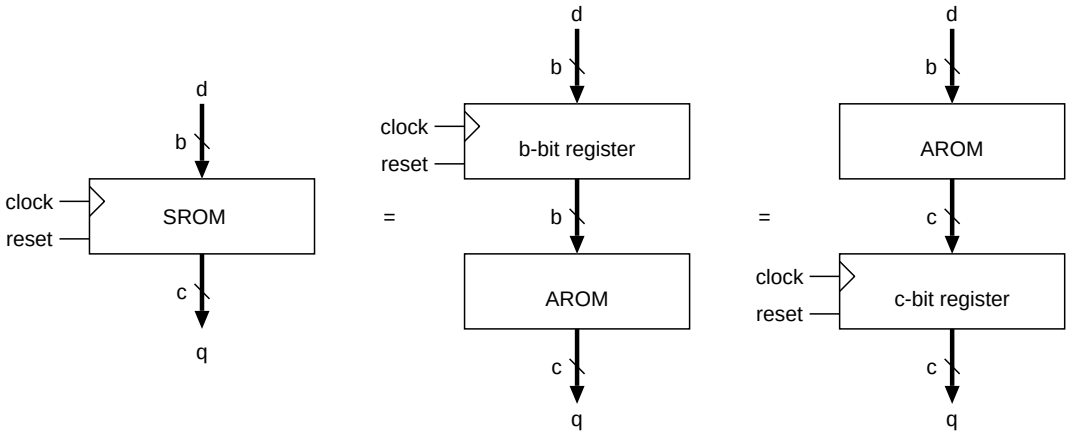


Figure 3.7: An example of three circuits such as SROM, R+AROM, and AROM+R for showing an equivalence.

reader's benefit, we will show an example of the equivalence.

Let us consider a circuit R+AROM, that is, the output of R is connected to the input of AROM as illustrated in Figure 3.7. We also consider a circuit AROM+R, in which the output of AROM and the input of R are connected. For the periodic clock with initial reset, the output sequences of SROM, R+AROM, and AROM+R are as follows:

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

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

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

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

Suppose that a circuit X with AROMs is given. The main contribution of this chapter 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.

For later reference, we will introduce *a negative register* (NR), which is a nonexistent device used only for showing our algorithm to derive Y and related proofs. Recall that, a regular register latches the input at the rising clock edge. *A negative register* latches a future input. The Figure 3.5 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 3.5. Thus, the NR has the following output sequence for a periodic clock with an initial reset:

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

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

3.3 Circuit Graph and Rewriting Rules

This section shows a circuit with its underlying directed circuit graph and also describes five rewriting rules in details. We simply use a directed graph to denote the interconnections of a fully synchronous circuit. We call such graph *as a circuit graph*. A circuit graph consists of a set of nodes and a set of directed edges connecting two nodes. Each node is labeled by either I (Input port), O (Output port), CC (Combinational Circuit), R (Register), NR (Negative Register), AROM, or SROM. A node with label I is connected with one or more outgoing edges. A node with label O is connected with exactly one incoming edge. A node with label CC has one or more incoming edges and one or more

outgoing edges. A node with label R, NR, AROM, or SROM has one incoming and one outgoing edge. We also assume that a circuit graph is a directed acyclic graph (DAG), that is, it has no directed cycles. The Figure 3.6 illustrates an example of a directed graph. Note that nodes with label I, R, NR, AROM, or SROM has only one outgoing edge. The readers may think that one outgoing edge is a too stringent restriction because it does not allow two or more fan-outs. However, we can implement multiple fan-outs by attaching a simple combinational circuit (CC) that just duplicates the input. For example, a CC with one input port A and two output ports F and G such that $F = A$ and $G = A$ is used to implement fan-out 2 as illustrated in Figure 3.8.

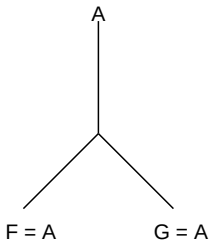


Figure 3.8: A combinational circuit to implement fan-out 2 circuit.

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 3.9 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

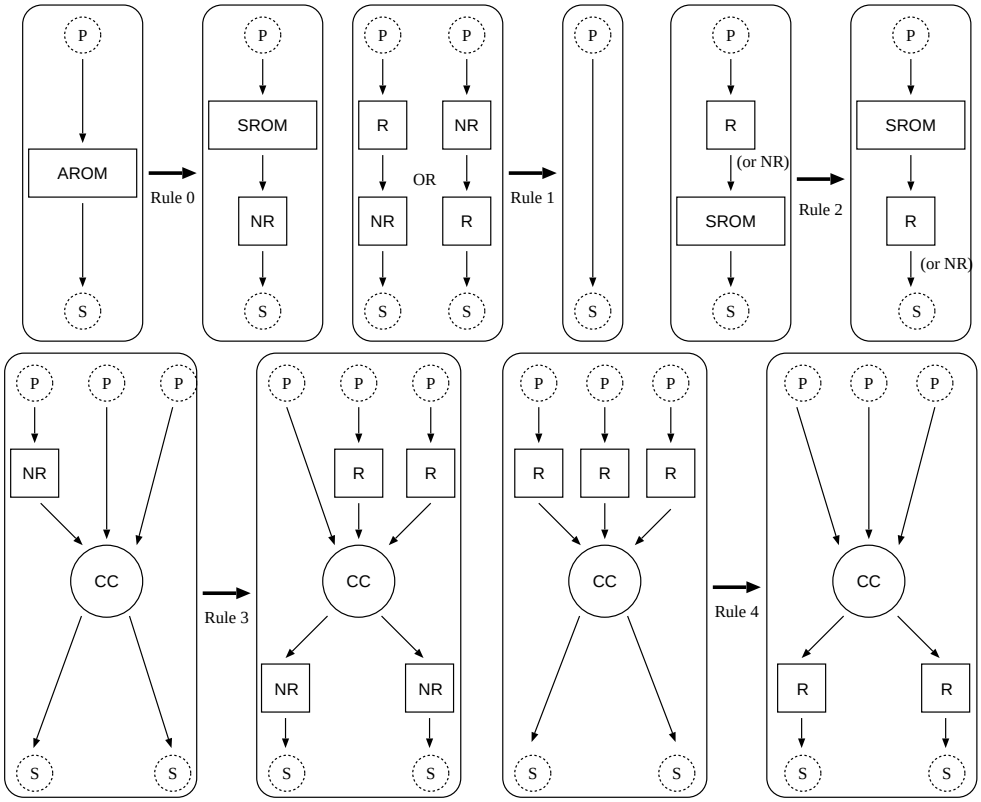


Figure 3.9: Rules to rewrite a circuit graph.

removed.

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

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

Rule 4 If all the incoming edges of a CC node are connected to an R node, then all the Rs are moved 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, 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. On the other hand, NR+SROM and SROM+NR have the output sequences $\langle 0,\ M[d_1],\ M[d_2],\ M[d_3],\ \ldots\rangle$ and $\langle M[d_0], M[d_1], M[d_2], M[d_3]\ldots\rangle$, respectively and thus they are an equivalent.

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

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

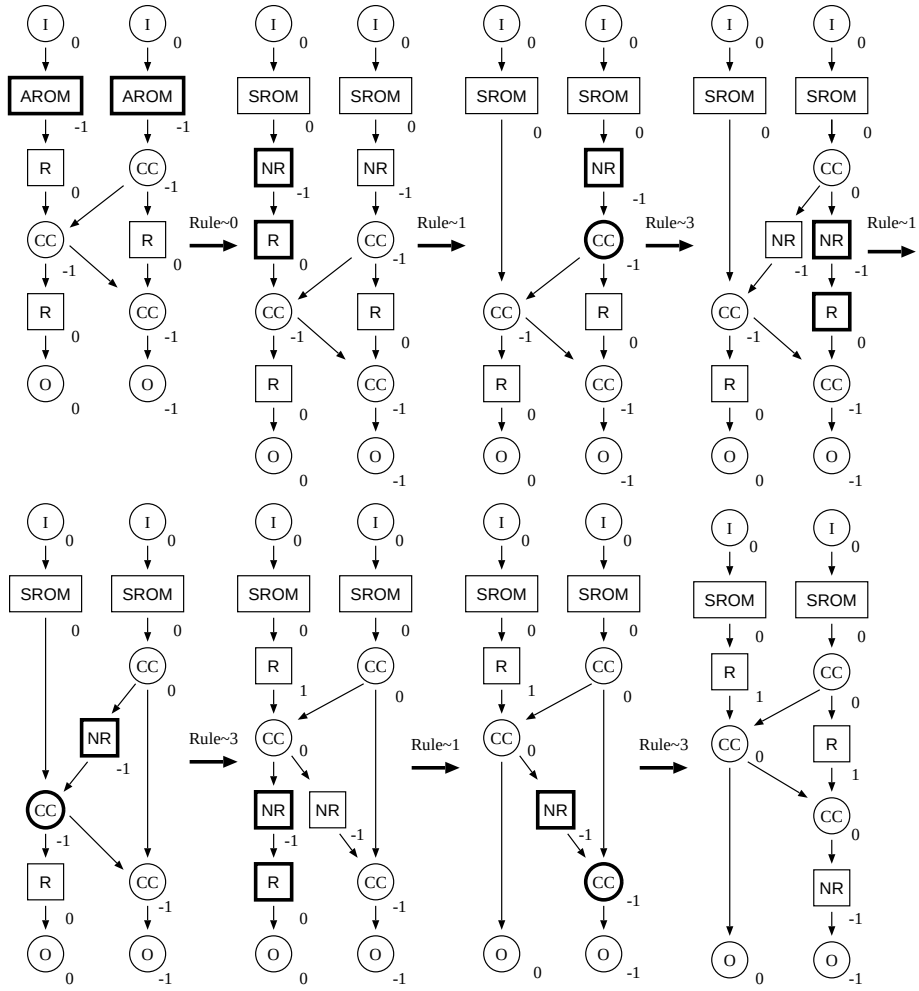


Figure 3.10: Interim and resulting circuit graphs obtained by our rewriting algorithm for a circuit graph.

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

The readers should refer to Figure 3.10 for illustrating interim and resulting circuit graphs obtained using our rewriting algorithm. In this figure, nodes applied rules are highlighted.

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

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

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

The Figure 3.6 also shows the potentiality of each node.

We have the following theorem.

Theorem 3.3.1 *All O nodes of a circuit graph have non-negative potentiality, if and only if our rewriting algorithm generates an AROM-free and NR-free circuit graph, equivalent to the original circuit graph.*

In other words, we can determine a fully synchronous circuit that can be converted into an AROM-free circuit by evaluating the potentiality of all O nodes of the corresponding circuit graph. Also, the potentiality of all O nodes are non-negative, our rewriting algorithm generates an AROM-free and NR-free circuit graph, and the corresponding fully synchronous circuit is an AROM-free and an equivalent to the original fully synchronous circuit. For example, in Figure 3.10, the potentiality of the right O node is negative. Hence, the resulting circuit graph has an NR node and our rewriting algorithm fails to remove all NRs.

3.4 Proof of Theorem 3.3.1

The main purpose of this section is to show a proof of Theorem 3.3.1. We will show several lemmas for a proof of Theorem 3.3.1.

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

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

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

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

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

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

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

Lemma 3.4.1 *The potentiality of every O node of the resulting circuit graph is the same as that of the corresponding O node of the original circuit graph.*

In Figure 3.10, the potentialities of the left and the right O nodes are 0 and -1, respectively, and these values are never changed.

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

Lemma 3.4.2 *Let u be an NR node and* (u, v) *be its outgoing edge in the resulting circuit graph. Node v must be either NR or O node. Also, all NR nodes must be in segments ending at O node.*

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

From Lemma 3.4.2, we will prove that all SROM and CC nodes in the resulting circuit graph have zero potentiality.

Lemma 3.4.3 *All SROM and CC nodes in the resulting circuit graph have non-negative potentiality.*

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

Similarly, we have the following lemma.

Lemma 3.4.4 *All SROM and CC nodes in the resulting circuit graph have non-positive potentiality.*

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

Case 1 v is an SROM node

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

Case 2 v is a CC node

Let $(u_1, v), (u_2, v), \ldots, (u_k, v)$ $(k \geq 1)$ denote the incoming edges. From Lemma 3.4.2, none of $u_1, u_2, \ldots, u_k$ is an NR node. If all of them are R nodes, then Rule 4 can be applied. Thus, at least one of them is not an R node. It follows that at least one of them is either I, SROM, or CC node. From the assumption, the potentiality of such node is non-positive, Hence, the potentiality of v is non-positive, a contradiction.

We are now in position to show the proof of Theorem 3.3.1. From Lemma 3.4.3 and 3.4.4, all SROM and CC nodes in the resulting circuit graph have zero potentiality. Hence, if the potentiality of one of the O nodes in the resulting circuit graph is negative, a segment ending at O node in the resulting graph should have NR from Lemma 3.4.2. 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 3.4.1, 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 3.3.1.

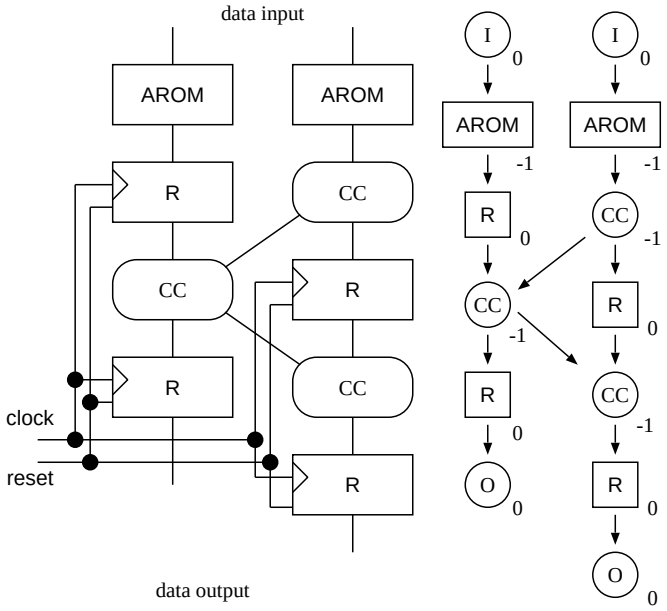


Figure 3.11: A circuit an almost equivalent to that of Figure 3.6 that can be converted into an AROM-free circuit.

From Theorem 3.3.1, it is not always possible to have an equivalent AROM-free circuit to the original one. 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 poten-

tiality of all O nodes in the corresponding circuit graph. After that, we insert registers just before O nodes with negative potentiality so that the potentiality of the corresponding O nodes turns into a zero. Since the potentiality of the corresponding O nodes now is 0, it can be converted into an equivalent AROM-free circuit by our Theorem 3.3.1. The readers should refer to the Figure 3.11 for illustrating an example. Note that, the resulting circuit is not an equivalent to the original circuit. However, the difference is the latency of the output. Thus, we can say that the resulting circuit is an almost equivalent to the original circuit.

3.5 Concluding Remarks

The main contribution of this chapter was to present a rewriting algorithm and five rewriting rules to convert a circuit with AROMs into an equivalent circuit with no AROMs for the current FPGA. Using our rewriting algorithm, any sequential circuit with AROMs can be converted into an equivalent fully synchronous sequential circuit with no AROMs to support the modern FPGA architecture. Although this conversion is not trivial. However, our approach, presented in this chapter did it automatically. We also described a technique for applying our rewriting algorithm even if a user designs a circuit with pipeline structure.

Chapter 4

A Circuit Rewriting Algorithm to Obtain Circuits with Synchronous RAMs for FPGAs

We present a circuit rewriting algorithm to generate an equivalent circuit with Synchronous Random Access Memories (*SRAMs for short*) for a circuit with Aynchronous Random Access Memories (*ARAMs for short*) such that generated circuit with SRAMs can be embedded into FPGAs. The main contribution of this chapter is to consider *both the read and write operations* of the memories (RAMs) for the algorithm. The contribution, described in Chapter 3, was to consider *only read operation* of the memry blocks (ROMs). More specifically, presented algorithm was to obtain an equivalent circuit with SROMs supporting synchronous read operation for the circuit with AROMs supporting aynchronous read operation. However, this circuit rewriting algorithm, presented in this chapter, is used to obtain an equivalent circuit with SRAMs supporting both synchronous read and synchronous write operations for the circuit with ARAMs

supporting aynchronous read and synchronous write operations. We say that that circuit design that minimizes the number of clock cycles is easy if we use asynchronous read operations. However, embedded memories in FPGAs support synchronous read and synchronous write operations, but do not support asynchronous read operations. To resolve this problem, we provide one of the potent approaches in this chapter. 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. We also briefly discuss the techniques to improve performance of the ARAM-free resulting circuit.

4.1 Introduction

Recall the contribution, presented in Chapter 3 which was concerned with *only read operation* of the memory blocks such ROMs in FPGAs. More specifically; presented algorithm in Chapter 3, was used to convert a circuit with AROMs supporting asynchronous read operation into an equivalent circuit with SROMs supporting synchronous read operation. However, algorithm, presented in this chapter mainly concerns both *read and write operations* of the memory blocks. Particularly, presented algorithm is used to convert a circuit using asynchronous RAMs (ARAMs) designed by a non-expert or quickly designed by an expert into an equivalent circuit using synchronous RAMs (SRAMs) for implementing in FPGAs. For the benefit of readers, we recall a brief description of a memory block. 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 this chapter, we consider single-port RAM which can be embedded into a dual-port

block RAM of the current FPGAs. Usually, the dual-port RAM supports synchronous read and synchronous write operations. The read and write operations are performed at the rising clock edges. FPGAs can be used to implement designed circuits by users. For the benefit of readers, we recall some examples of circuit implementations in FPGAs, described in [1, 2, 9, 14] to accelerate computation.

In this chapter, we mainly 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.

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 chapter, 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* as shown in Figure 4.1. ARAMs and SRAMs support asynchronous and synchronous read operations respectively. Also they both support synchronous write operation. 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 work in Chapter 3, we have presented a circuit rewriting approach for Directed Acyclic Graph (DAG) circuits considering only read operations of the memory blocks (ROMs). However, this chapter considers both read and write operations of the memory blocks (RAMs). Note that, a RAM can be treated as a ROM when write enable *we* is low. It is not trivial to convert a circuit with ARAMs into an equivalent circuit with synchronous ones. However, it is automatically done by our algorithm.

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

Combinational Circuits (CCs), Registers (Rs), and RAMs with asyn-

chronous 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 our 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 with SRAMs.
- The resulting synchronous circuit can be implemented in FPGAs.

The outlines of our work are as follows:

- We use a Negative Register (NR) which is originally introduced in our previous Chapter 3. The NR is an imaginary register that is used for latching a future input data.
- We define simple five rules that rewrite a circuit.
- 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.

Our results have several significant points as follows:

- The correctness of our algorithm is proved in a rigorous manner.
- Our algorithm works for the circuits with RAMs. In particular, our circuit rewriting algorithm is used to generate an equivalent circuit with Synchronous Random Access Memories (*SRAMs for short*) for a circuit with Aynchronous Random Access Memories (*ARAMs for short*) such that generated circuit with SRAMs can be embedded into FPGAs.
- Our circuit rewriting algorithm moves all redundant registers toward the output ports. They can be removed to decrease the latency of the circuit. Therefore, the circuit that obtained has minimum latency in the sense that all redundant registers are deleted. Readers may refer to Chapter 6 for an example.
- Since, our rewriting algorithm moves registers towards the output ports, whenever possible. Hence, in general, the resulting circuit may have the longest path from input ports to registers/SRAMs or from registers/SRAMs to registers/SRAMs or from registers/SRAMs to output ports. Hence, clock performance of the resulting circuit may degrade. However, it is easier to improve clock performance of the resulting circuit than minimizing number of clock cycles. Clock performance of the resulting circuit can be improved by inserting registers (Rs) appropriately. In this regard, we refer the readers to Chapter 6, where we have shown an example for improving clock performance of the resulting ARAM-free circuit.

This chapter is organized as follows: Section 4.2 briefly describes random access memory (RAM). We briefly review the circuits with RAMs and also show their equivalence in Section 4.3. In Section 4.4, we describe our rewriting algorithm,

circuit graph with RAMs and also explain the equivalence for our rewriting rules. For the reader's benefits, Section 4.5 shows how our circuit rewriting algorithm works for circuit graphs with RAMs. Section 4.6 presents the proof of the correctness of our rewriting algorithm. Finally Section 4.7 concludes this chapter.

4.2 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 b bits each, shown in Figure 4.1. A RAM can support asynchronous read, synchronous read and synchronous write operations. These are described by the following subsections.

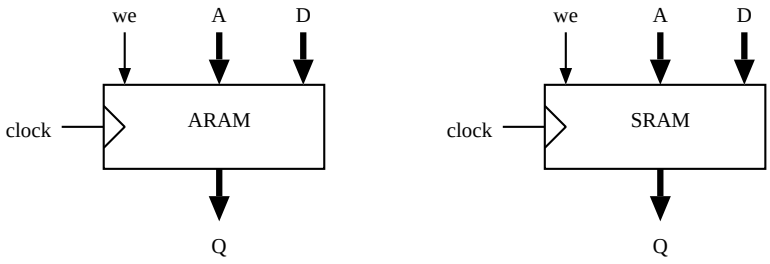


Figure 4.1: An asynchronous RAM (ARAM) and a synchronous RAM (SRAM).

4.2.1 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.

4.2.2 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.

4.2.3 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.

For the reader's benefit, we will describe two types of RAM (shown in Figure 4.1) 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 operation. 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 operation for an SRAM is the same as an ARAM. The readers may refer to the Figure 4.2 for read and write operations of an ARAM and an SRAM.

In this chapter, 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.

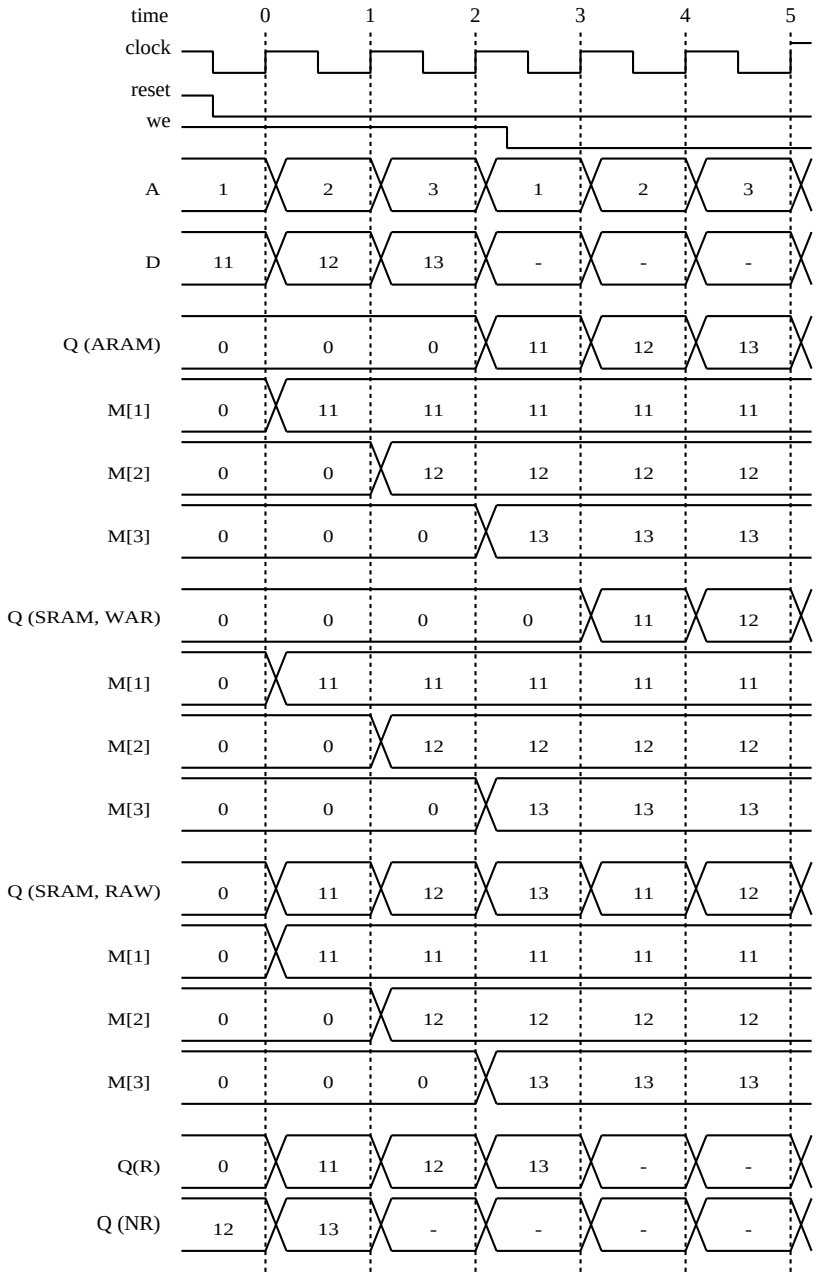


Figure 4.2: A timing chart of an ARAM, an SRAM, a register (R), and a negative register (NR).

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 when only 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 Figure 4.2 for the illustrations of the WAR and RAW. Figure 4.2 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 at the memory content 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 at the memory content 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 at the memory content 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 at the memory content 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 at the memory content of $M[1]$, $M[2]$, $M[3]$ 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.

4.3 Circuits with RAMs and Their Equivalence

In this section, we mainly describe circuits with RAMs including the behavior of the circuit elements 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*. For the benefit of read-

ers, we recall a brief overview from the previous Chapter 3 about combinational circuits (CCs), registers (Rs) as follows. However, RAM as a circuit element is newly described.

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

A b-bit register has a clock input and a reset input. It can store a b-bit data. 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, as shown in the previous Chapter 3. About RAMs, we refer readers to the Section 4.2 of this chapter.

4.3.1 Behavior of the Circuit Element

We will describe a behavior of the circuit elements 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 Figure 4.2. For the benefit of readers, we recall the output sequences for combinational circuits (CCs) and registers (Rs). However, output sequences and stored data of RAMs will be described newly. The behavior of each circuit element is described by the output sequences and stored data as follows:

Combinational Circuit (CC):

For the benefit of readers, we recall the output sequences of a combinational circuit (CC) from the previous Chapter 3. For simplicity, we assume 3-input 2-output combinational circuit, as illustrated in the previous Chapter 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 ($i \geq 0$). As shown in Figure 4.2, the output sequence of the register (R) is described as follows:

R: $\langle 0, 11, 12, 13, -, -, \ldots\rangle$

Synchronous and Asynchronous RAMs (SRAMs and ARAMs):

Let $M[j]$ denotes the value stored in address j ($j \geq 0$) 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 Figure 4.2, 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): $\langle 0, 0, 0, 0, 11, 12\rangle$

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

$M[1]$: $\langle 0, 11, 11, 11, 11, 11\rangle$. It means that memory content of the

address value 1 is updated by 11 at time 1 and remains the same until further updating.

$M[2]$: $\langle 0, 0, 12, 12, 12, 12\rangle$. 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

$M[3]$: $\langle 0, 0, 0, 13, 13, 13\rangle$. 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): $\langle 0, 0, 0, 11, 12, 13\rangle$

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

$M[1]$: $\langle 0, 11, 11, 11, 11, 11\rangle$. $M[2]$: $\langle 0, 0, 12, 12, 12, 12\rangle$ and $M[3]$: $\langle 0, 0, 0, 13, 13, 13\rangle$. These have the same aforesaid explanations.

In this chapter, 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 Figure 4.3 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.

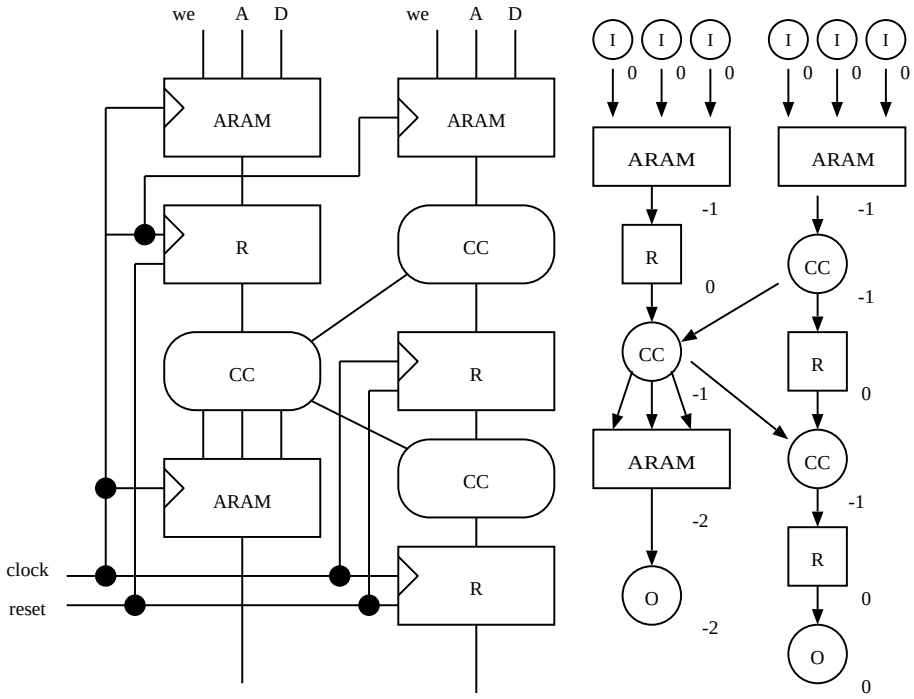


Figure 4.3: An example of a fully synchronous circuit with ARAMs and the corresponding circuit graph with potentiality.

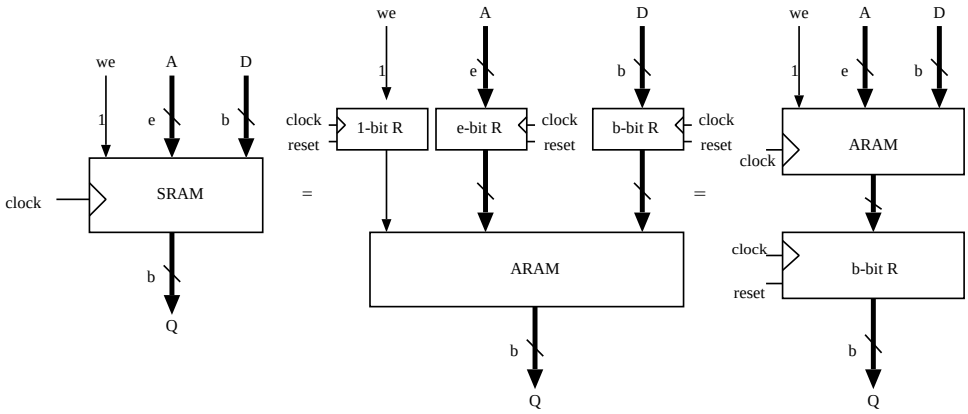


Figure 4.4: An example of three circuits such as SRAM, R + ARAM and ARAM + R for showing an equivalence.

4.3.2 An Example of Circuits with RAMs to Show an Equivalence

We will describe *an equivalence* of the circuits with RAMs in terms of output sequences and stored data. Let us define *an 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 in case of three circuits such as SRAM, R+ARAM and ARAM+R as illustrated in Figure 4.4 are as follows. The readers may refer to Figure 4.5 for better understanding. Figure 4.5 shows timing diagram of the earlier mentioned three circuits for illustrating *an equivalence*.

SRAM (output sequence): $\langle 0, 0, 0, 0, 11, 12\rangle$

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

$M[1]$: $\langle 0, 11, 11, 11, 11, 11\rangle$, $M[2]$: $\langle 0, 0, 12, 12, 12, 12\rangle$ and $M[3]$: $\langle 0, 0, 0, 13, 13, 13.\rangle$

R+ARAM (output sequence): $\langle 0, 0, 0, 0, 11, 12\rangle$

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

$M[1]$: $\langle 0, 0, 11, 11, 11, 11\rangle$, $M[2]$: $\langle 0, 0, 0, 12, 12, 12\rangle$ and $M[3]$: $\langle 0, 0, 0, 0, 13, 13.\rangle$

ARAM+R (output sequence): $\langle 0, 0, 0, 0, 11, 12\rangle$

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

$M[1]$: $\langle 0, 11, 11, 11, 11, 11\rangle$, $M[2]$: $\langle 0, 0, 12, 12, 12, 12\rangle$ and $M[3]$: $\langle 0, 0, 0, 13, 13, 13.\rangle$

These have also the same explanations mentioned above.

Since, these three circuits 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 *an equivalent*. In this chapter, we ignore first several clock cycles when we determine *an equivalence* of the circuits.

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

- a necessary condition such that an ARAM-free circuit, Y can be generated, which is an 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 Figure 4.2. Thus, an NR has the following output sequence for a periodic clock with an initial reset.

NR: $\langle 12, 13, -, -, -, -, \rangle$.

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

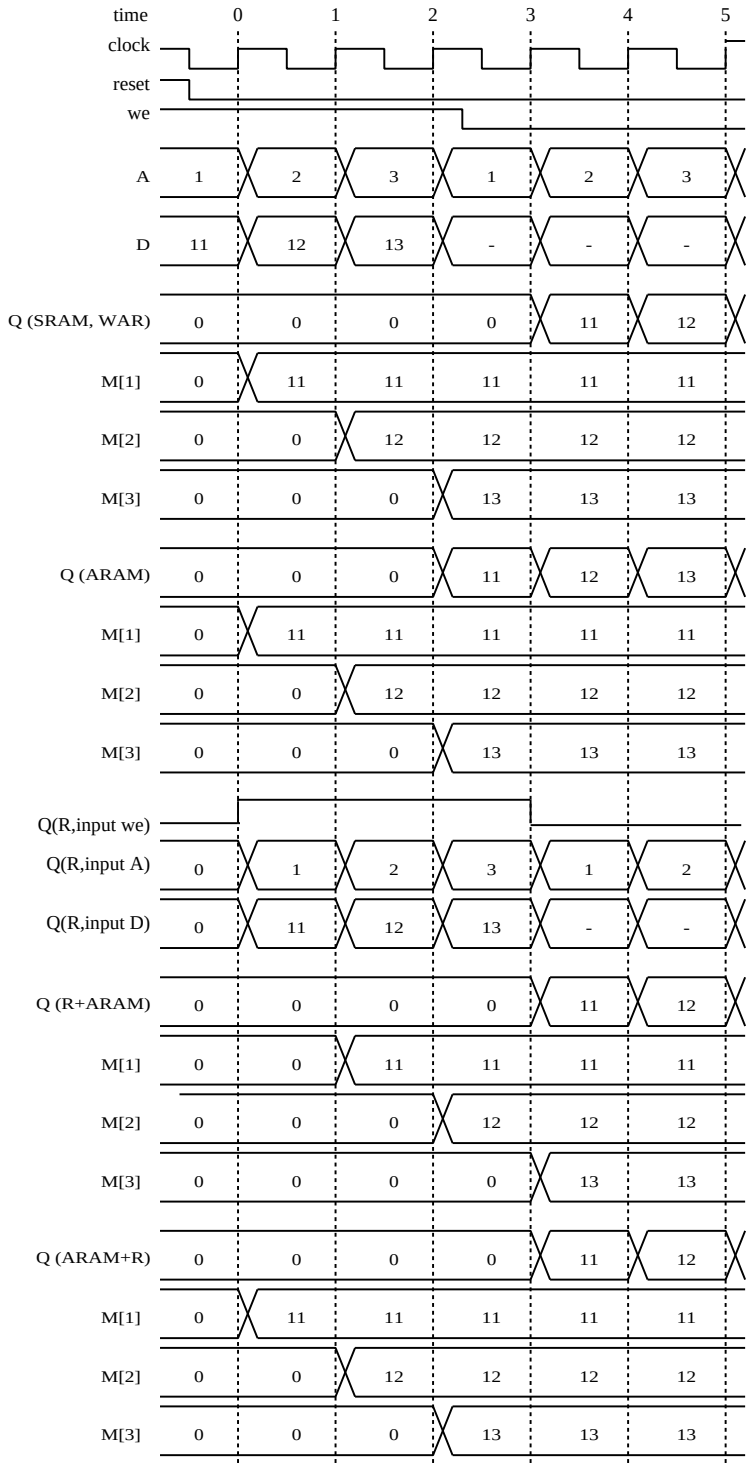


Figure 4.5: A timing chart for showing an equivalence of the three circuits such as SRAM, R + ARAM and ARAM + R.

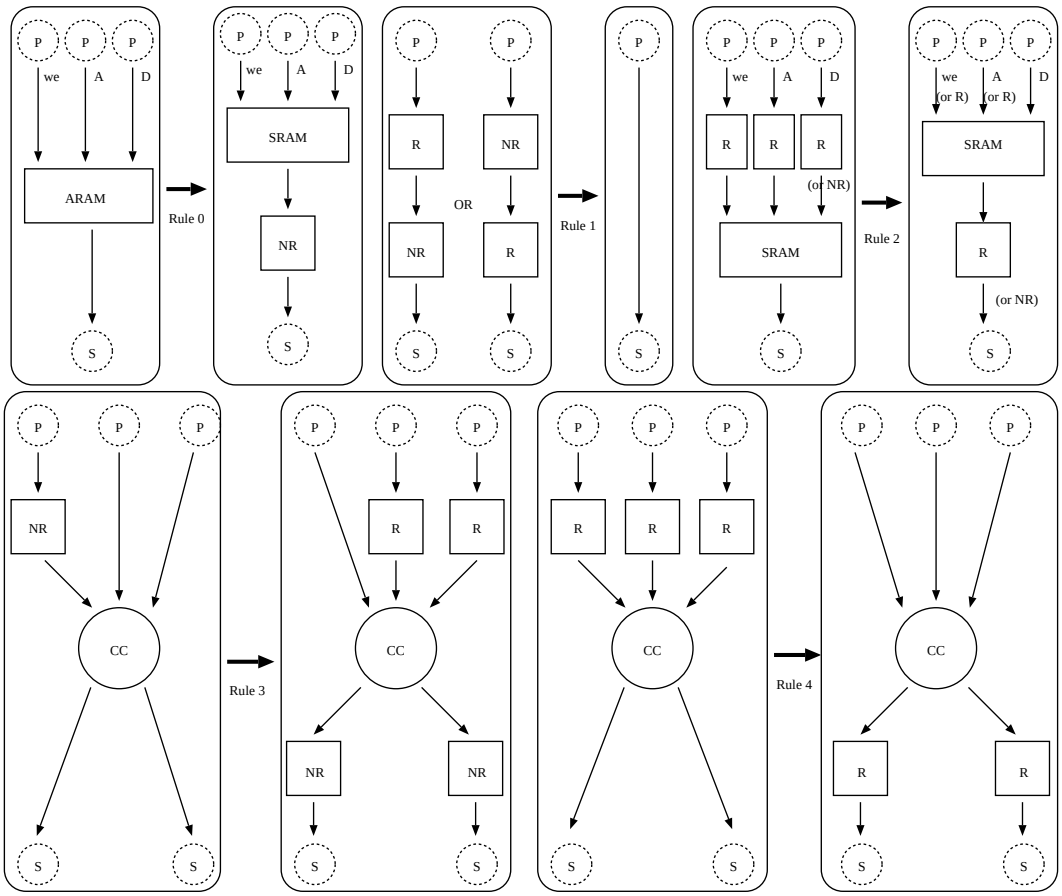


Figure 4.6: Rules to rewrite a circuit graph with RAMs (ARAMs and SRAMs).

4.4 Circuit Graph with RAMs and Rewriting Rules

This section will describe a circuit with RAMs whose underlying graph is directed acyclic graph (DAG) and rules for generating an equivalent ARAM-free and NR-free circuit graph to the given input circuit graph with ARAMs. 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. Note that, we assume a circuit graph which is a directed acyclic graph (DAG), that is, it has no directed cycles. The Figure 4.3 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 as illustrated in Chapter 3. 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 Figure 4.6 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).

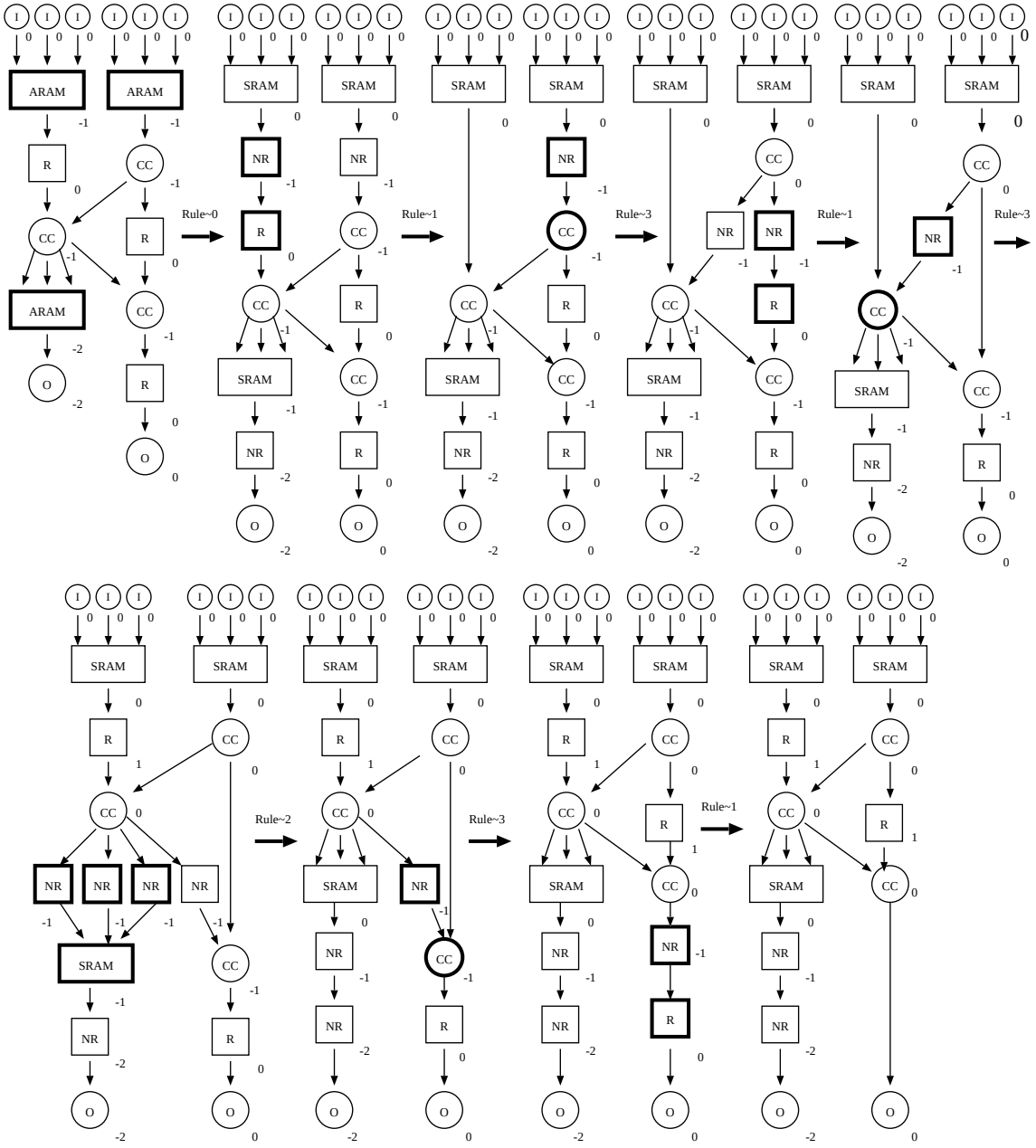


Figure 4.7: Interim and resulting circuit graphs obtained by our rewriting algorithm for a circuit graph with RAMs (ARAMs and SRAMs).

More specifically, if each incoming edge of an SRAM node is connected to a R node, then all the Rs are moved 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 the 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, an R node is added to all the other incoming edges, and the NR node is moved to all the outgoing edges of the CC node.

Rule 4 If all the incoming edges of a CC node are connected to a R node, then all the Rs are moved 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* ($i \geq 0$) 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): $\langle 0, 0, 0, 11, 12, 13 \rangle$.

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

$M[1]$: $\langle 0, 11, 11, 11, 11, 11 \rangle$, $M[2]$: $\langle 0, 0, 12, 12, 12, 12 \rangle$ and $M[3]$: $\langle 0, 0, 0, 13, 13, 13 \rangle$.

Thus they are an equivalent.

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

R+NR : $\langle 11, 12, 13, -, -, - \rangle$ and NR+R: $\langle 0, 12, 13, -, -, - \rangle$. Also, NULL

circuit has the following output sequence. NULL: $\langle 11, 12, 13, -, -, - \rangle$. Thus, they are an equivalent.

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

R+SRAM (output sequence): $\langle 0, 0, 0, 0, 0, 11 \rangle$

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

$M[1]$: $\langle 0, 0, 11, 11, 11, 11 \rangle$, $M[2]$: $\langle 0, 0, 0, 12, 12, 12 \rangle$ and $M[3]$: $\langle 0, 0, 0, 0, 13, 13 \rangle$

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

SRAM+R (output sequence): $\langle 0, 0, 0, 0, 0, 11 \rangle$

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

$M[1]$: $\langle 0, 11, 11, 11, 11, 11 \rangle$ $M[2]$: $\langle 0, 0, 12, 12, 12, 12 \rangle$ and $M[3]$: $\langle 0, 0, 0, 13, 13, 13 \rangle$

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

NR+SRAM (output sequence): $\langle 0, 0, 0, 0, 12, 13 \rangle$

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

$M[1]$: $\langle 0, 12, 12, 12, 12, 12 \rangle$, $M[2]$: $\langle 0, 0, 13, 13, 13, 13 \rangle$ and $M[3]$: $\langle 0, 0, 0, -, -, - \rangle$

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

SRAM+NR (output sequence): $\langle 0, 0, 0, 0, 12, 13 \rangle$

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

$M[1]$: $\langle 0, 0, 12, 12, 12, 12 \rangle$, $M[2]$: $\langle 0, 0, 0, 13, 13, 13 \rangle$ and $M[3]$: $\langle 0, 0, 0, 0, -, - \rangle$

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 equivalence* of the Rule 2 can be shown in similar way.

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

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

4.5 An Example to Show the Behavior of Our Algorithm for the Circuits with RAMs

In this section, we will describe, how our rewriting algorithm works for the circuits with RAMs. 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 nodes is moved towards the output nodes using Rule 2, 3 and 4 whenever possible.

Figure 4.7 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 $p(v) = \min(p(u_1), p(u_2), \ldots, p(u_k))$.

Figure 4.3 shows the potentiality of each node.

We have the following theorem.

Theorem 4.5.1 *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.*

4.6 Proof of Theorem 4.5.1

The main purpose of this section is to show a proof of Theorem 4.5.1. 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 4.6.1 *The potentiality of every O node of the resulting circuit graph is the same as that of the corresponding O node of the original circuit graph.*

In Figure 4.7, 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 $u_1, u_2, \ldots, u_k$ such that, u_1 and u_k are either I, O, SRAM, or CC, and $u_2, \ldots, u_{k-1}$ are either R or NR. Note that, if $k = 2$ then it represents a null segment with u1 and u2. We also have the following lemma.

Lemma 4.6.2 *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 4.6.2, we will prove that all SRAM and CC nodes in the resulting circuit graph have zero potentiality.

Lemma 4.6.3 *All SRAM and CC nodes in the resulting circuit graph have non-negative potentiality.*

Proof Since the resulting graph is an ARAM-free, nodes follow the 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 4.6.4 *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 4.6.2, 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, $p(u_1) = 0$ or $p(u_2) = 0$ or $p(u_3) = 0$ and thus, $p(v) = 0$, a contradiction.

Case 2 v is a CC node

Let (u_1, v), (u_2, v), ..., (u_k, v) ($k \geq 1$) denote the incoming edges. From Lemma 4.6.2, 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 4.6.3 and 4.6.4, 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 4.6.2. 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 4.6.1, 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 4.5.1.

By the Theorem 4.5.1, 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 4.5.1.

4.7 Concluding Remarks

The main contribution of this chapter was to convert a circuit with ARAMs into an equivalent circuit with no ARAMs for the current FPGA considering both read and write operations of the memory blocks (RAMs), however, in our previous work, described in Chapter 3, we consider only read operation of the memory blocks (ROMs). For the purpose of converting into ARAM-free circuits, we have presented a rewriting algorithm and five rewriting rules in this chapter. In fact, we improved our previous research work, described in Chapter 3, where RAMs can be used as the additional circuit elements to the given input circuits. Although, it is not trivial to convert a sequential circuit with ARAMs into an equivalent fully synchronous circuit with no ARAMs for supporting the modern FPGA architecture, however, our algorithm did it automatically.

Chapter 5

A Modified Circuit Rewriting Algorithm for the Circuits with Cycles

The main contribution of this chapter is to show a modified circuit rewriting algorithm to convert a circuit with cycles using AROMs supporting asynchronous read operation into an equivalent circuit with cycles using SROMs supporting synchronous read operation for implementing in FPGAs. In Chapter 3 and Chapter 4, we have presented circuit rewriting algorithms to convert a circuit with asynchronous ROMs or asynchronous RAMs into an equivalent circuit with synchronous ones. The resulting circuit with synchronous ROMs or synchronous RAMs can be embedded into FPGAs. However, these circuit rewriting algorithms can handle circuits represented by a directed acyclic graph and do not work for those with cycles. In this chapter, we succeeded in relaxing the cycle-free con-

dition 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. We also briefly discuss the techniques to improve performance of the AROM-free resulting circuit.

5.1 Introduction

We present a modified circuit rewriting algorithm to consider *cycles* in the given input circuits with AROMs. The rewriting approaches, presented in Chapter 3 and Chapter 4 have a strict limitation in terms of input circuits that only work for the circuits whose underlying graphs have *no cycles*. However, practical circuits have cycles. We are inspired to convert the real world practical circuits. Hence, the main contribution of this chapter is to overcome this strict limitation in terms of input circuits. More specifically, we present a modified circuit rewriting approach in this chapter which is able to convert a circuit with *cycles* using AROMs into an equivalent circuit with *cycles* using SROMs for implementing in FPGAs.

In this chapter, we also focus the asynchronous and synchronous read operations of memory blocks, as illustrated in Chapter 3. For the benefit of readers, we recall these operations as follows:

Asynchronous read operation

The memory block outputs the data specified by the address given to the address port. When the address value is changed, the output data is updated immediately within some delay time. In other words, the output data port

always outputs $M[d]$, which is the data stored in the input address value d.

Synchronous read operation

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

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

Our circuit rewriting approach, presented in this chapter is devoted 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).*

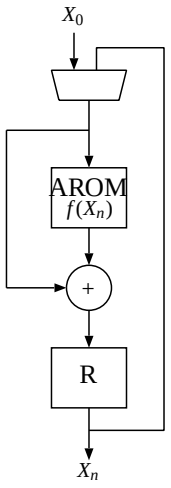
Note that, most of the current FPGAs support synchronous read operation, but do not support asynchronous one. We are thinking the following scenario to use our circuit rewriting algorithm:

- 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.

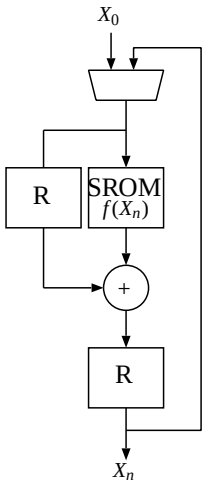
In other words, designers can design a circuit for FPGAs under the assumption of asynchronous read operation, which is simpler and easier than one with synchronous read operation.

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 $n \geq 1$. 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

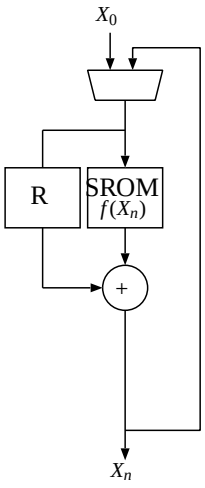
$f(i)$. Figure 5.1 (a) illustrates a circuit with an AROM to compute $X_1, X_2, \ldots$ for an input X_0. An AROM is used to compute the value of $f(X_n)$ for a given X_n. It should be clear that this circuit outputs $X_1, X_2, \ldots$ in every clock cycle. Figure 5.1 (b) shows a circuit with an SROM. 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 5.1 (b) needs double clock cycles over the circuit in Figure 5.1 (a). Using our algorithm to the circuit in Figure 5.1 (a), we can obtain the circuit in Figure 5.1 (c) automatically. In the circuit with an SROM in Figure 5.1 (c), $X_1, X_2, \ldots$ is output in every clock cycle. Thus, the timings of the circuits in Figure 5.1 (a) and (c) are identical.



(a) A circuit with an AROM



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



(c) The converted circuit with an SROM

Figure 5.1: An example of circuits with cycles using an AROM and an SROM

Obviously, we can minimize the number of clock cycles in the AROM-free resulting circuits by our rewriting algorithm, as illustrated in Figure 5.1 (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 5.1 (c). However our algorithm can do it automatically.

Conversely, 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, whenever possible. Hence, in general, the resulting circuit may have the longest path from input ports to registers/SROMs or from registers/SROMs to registers/SROMs or from registers/SROMs to output ports. Therefore, the circuit performance degrades. If this is the case, then it is possible to improve circuit performance of the AROM-free resulting circuit. We will briefly describe the techniques to improve the performance of the AROM-free resulting circuit in terms of the latency and clock frequency, although performance improvement of the AROM-free resulting circuit is beyond of this dissertation. 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 port.
- Clock performance of the AROM-free resulting circuit degrades due to the longest path from input ports to registers/SROMs or from registers/SROMs to registers/SROMs or from registers/SROMs to output ports. For this case, we can add registers appropriately in the AROM-free resulting circuit so that the longest path becomes shorter. 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 Chapter 3. 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 5.5 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. However, it is small in size so that it can not support the designer's demand, more expensive, and less speedy. However, it is not supported by the current FPGAs.

On the other hand, a circuit implementation with SROMs is dominating, although it has some drawbacks to design like clock distribution, more power consumption etc. As a result, we should use SROMs when we need a function of ROMs.

The main contribution of this chapter is to modify the circuit rewriting algorithm, presented in Chapter 3 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 5.2 (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 5.2 (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 toward the output ports. They can be removed to decrease the latency of the circuit. Therefore, the circuit that obtained has minimum latency in the sense that

all redundant registers are deleted. Readers may refer to Chapter 6 for an example.

- We also briefly discuss a technique to improve the clock frequency by inserting registers in the AROM-free resulting circuit appropriately.
- We will 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 5.7 of this chapter.
- 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 chapter is organized as follows: Section 5.2 briefly describes the related work so far. We briefly review the circuits and their equivalence in Section 5.3. In Section 5.4, we describe our rewriting algorithm, circuit graph and also explain the equivalence for our rewriting rules. For the reader's benefits, Section 5.5 shows how our circuit rewriting algorithm works for circuit graphs. Section 5.6 presents the proof of the correctness of our rewriting algorithm. Section 5.7 shows how we handle nodes that are not in the path from an input node. Finally Section 5.8 concludes this chapter.

5.2 Related Work

In this section, briefly we will describe about the related work. However, there is no related work except our previous one, described in Chapter 3. Hence, we will briefly summarize our previous work in Chapter 3 as a related one such that readers may compare our contribution in the current work, described in Section 5.1 with the previous one in Chapter 3. Note that, we are providing an innovative approach 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 work in Chapter 3, we have presented a circuit rewriting approach for circuits represented by a directed acyclic graph (DAG), illustrated in Figure 5.2 (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 5.2 (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 Chapter 3 has a strict restriction in terms of input circuits. It works only for a circuit whose underlying

graph is a DAG, illustrated in Figure 5.2 (1). Although most of practical circuits have cycles, it can not handle such circuits as illustrated in Figure 5.2 (2). To overcome this problem, a modified circuit rewriting algorithm is presented in this chapter. More specifically, our new circuit rewriting algorithm can convert any circuit with AROMs, represented by directed reachable graph (DRG) as illustrated in Figure 5.2 (2) into an equivalent circuit with SROMs for implementing in current FPGAs.

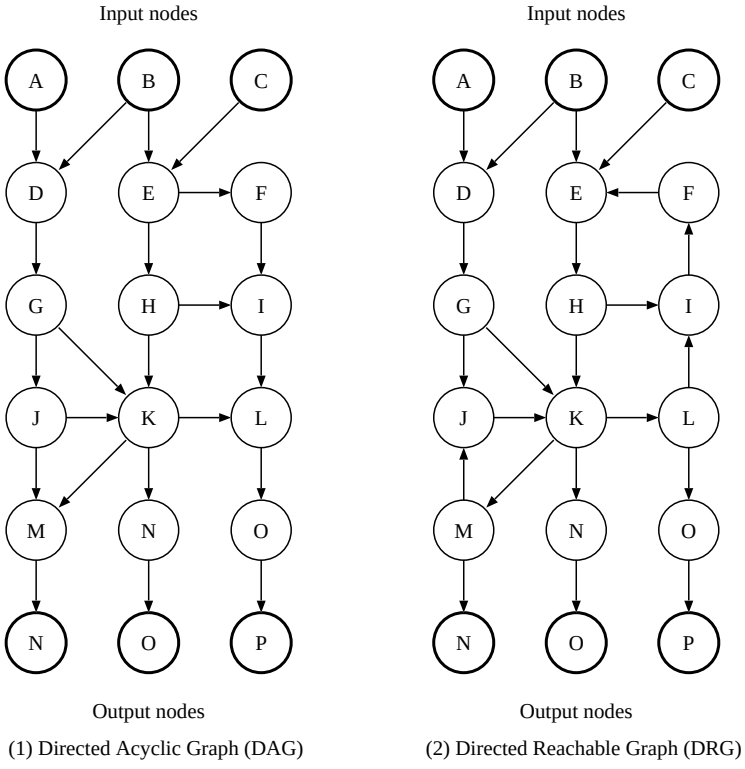


Figure 5.2: A directed acyclic graph (DAG) and a directed reachable graph (DRG).

5.3 Review of the Circuits and Their Equivalence

In this section, we shortly review the circuits and their equivalence for the benefit of readers. The readers may refer to Chapter 3 for details.

Let us consider a 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 Chapter 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 Chapter 3. 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 Chapter 3. 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.

For the reader's benefit, we also recall the Figure 5.3 from the Chapter 3, specifically to understand a new example of an equivalence clearly (we will show later). The Figure 5.3 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 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.

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 5.3. 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 Chapter 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:

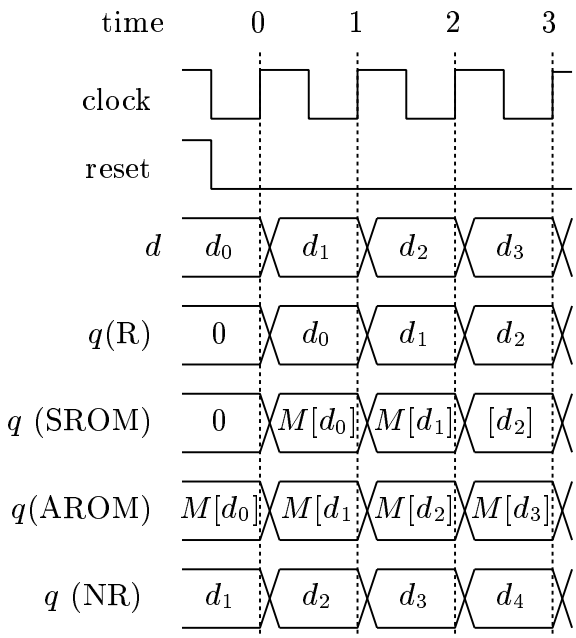


Figure 5.3: Recall a timing chart of a register (R), an SROM, an AROM and a negative register (NR) from Chapter 3.

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 chapter, we assume that a fully synchronous circuit has data inputs, data outputs, a global clock input, a global reset input, combinational circuits (CCs), registers (Rs), SROMs, AROMs, and their interconnects. The readers should refer to Figure 5.4 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.

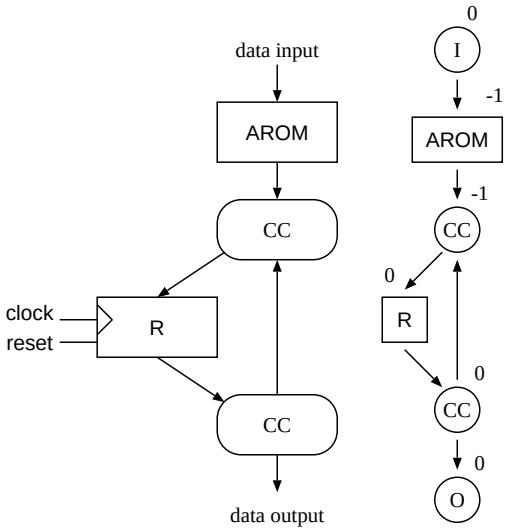


Figure 5.4: An example of a fully synchronous circuit with cycle and the corresponding circuit graph with potentiality.

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 5.5. 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 two registers (two 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 5.5. For the periodic clock with initial reset, the output sequences of SROM+R, R+SROM, and R+AROM+R are as follows (The readers may refer to Figure 5.3 for better understanding):

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 chapter, we ignore first several clock cycles when we determine an equivalence of the circuits.

Suppose that a circuit X with AROMs is given. The main contribution of this chapter 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 Chapter 3. Recall that, a regular register latches the input at the rising clock edge. *A negative register* latches a future input. The Figure 5.3 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 5.3.

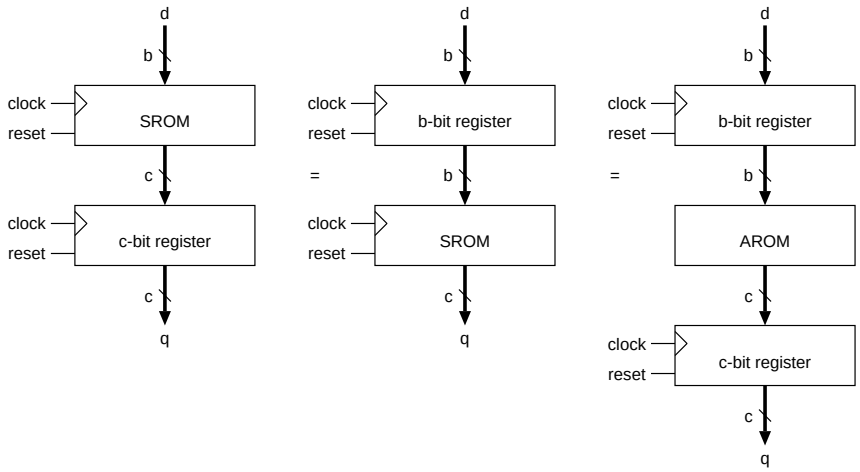


Figure 5.5: A new example of three circuits such as SROM+R, R+SROM, and R+AROM+R for showing an equivalence.

Thus, the NR has the following output sequence for a periodic clock with an initial reset:

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.

5.4 Circuit Graph with Cycles and Rewriting Rules

In this section, we will describe circuits, represented by Directed Reachable Graphs (DRGs) and the necessary circuit 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 5.2 (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 Chapter 3.

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 5.6 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 AROM node is rewritten into SROM+NR.

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

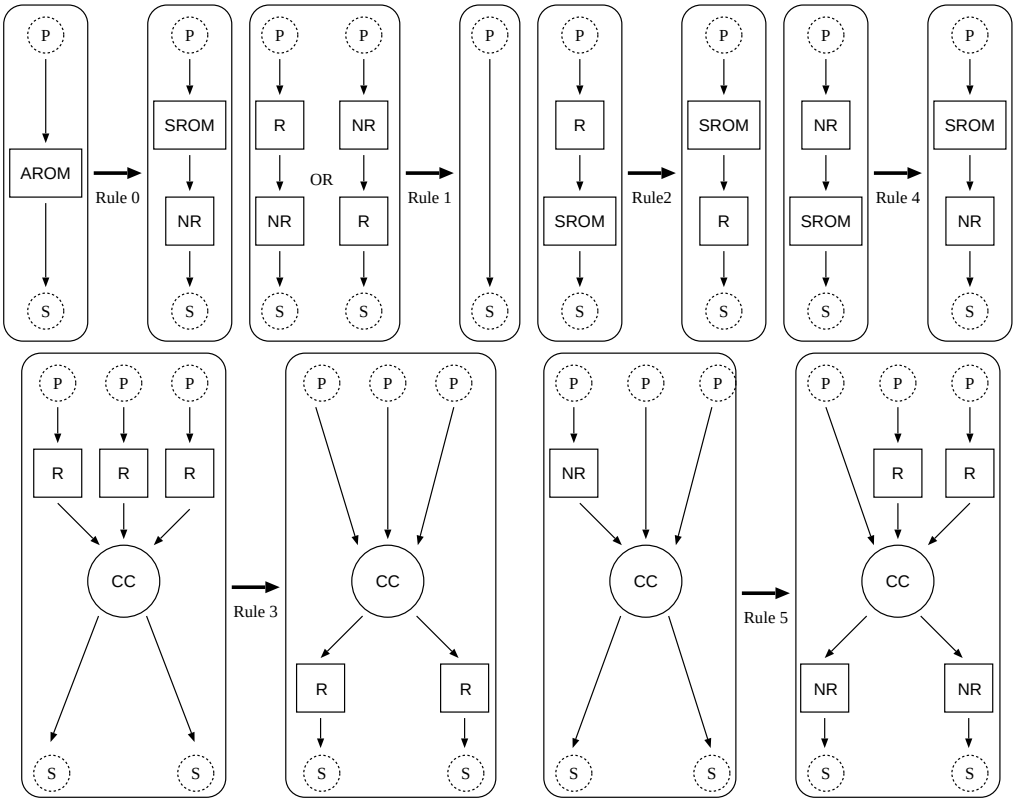


Figure 5.6: Rules to rewrite a circuit graph with cycles.

are removed.

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.

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.

5.5 Behavior of Our Circuit Rewriting Algorithm

This section mainly describes the behavior of our 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 5.7, 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.

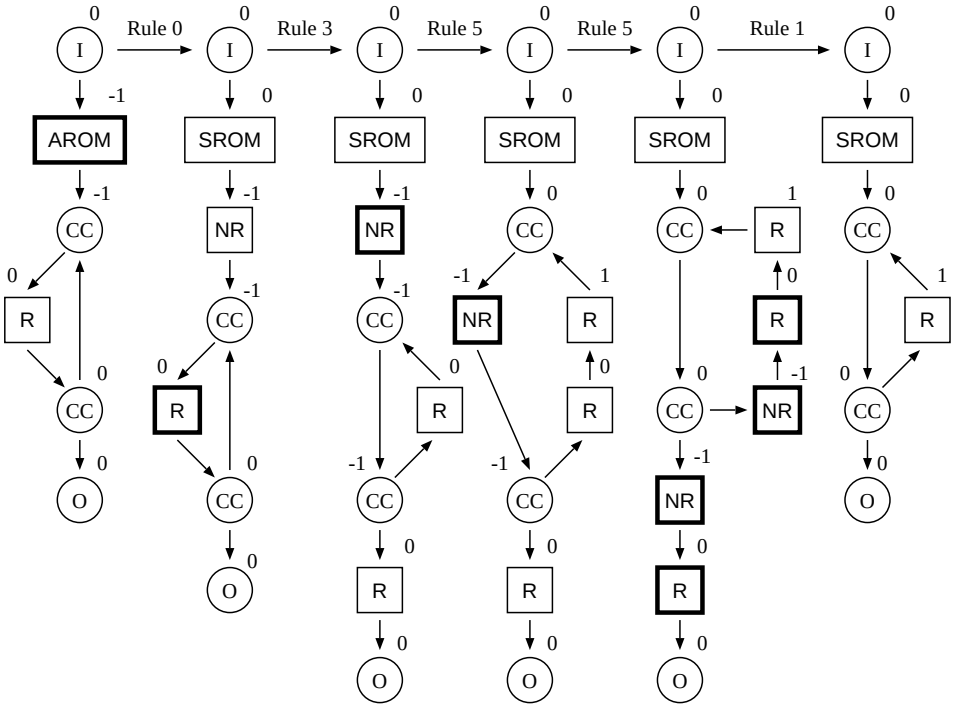


Figure 5.7: Interim and resulting circuit graphs obtained by our rewriting algorithm for a circuit graph with cycles.

Our circuit rewriting algorithm may not terminate for a circuit graph that has no way to convert an equivalent AROM-free circuit. Figure 5.8 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 5.8. 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.

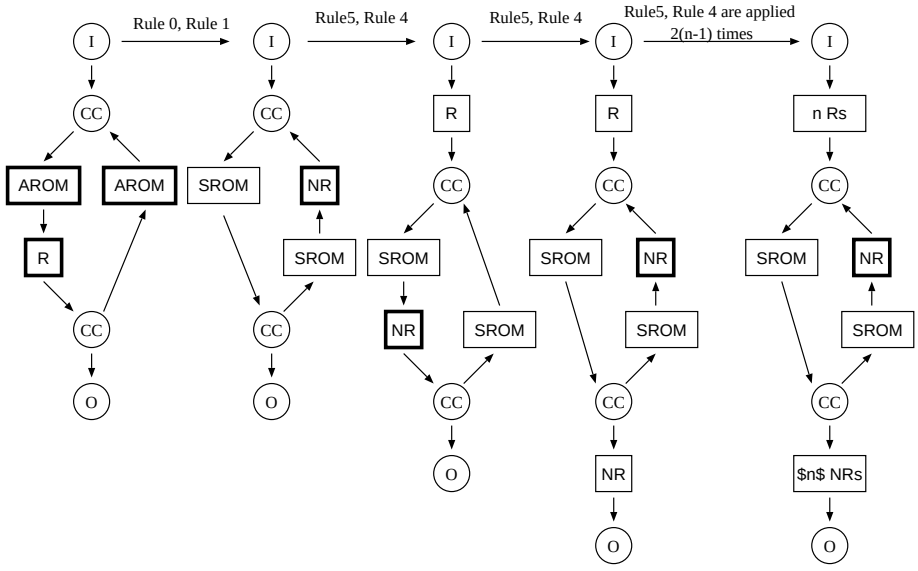


Figure 5.8: An example of a circuit graph with cycles for which our rewriting algorithm does not terminate.

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 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 5.9. 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 5.9 (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 5.9 (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 5.9 (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.

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:

- $p'(v_0) = 0$.

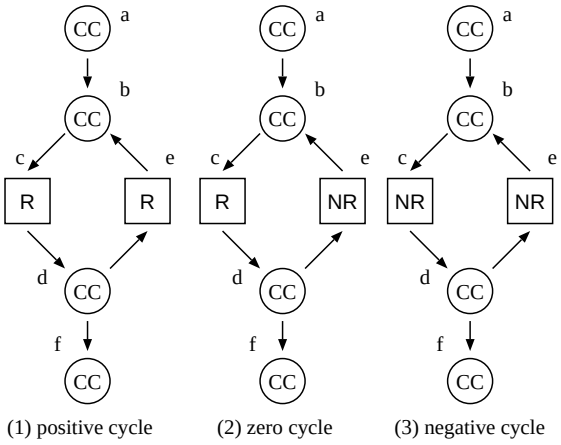


Figure 5.9: The potentiality for circuits with a cycle

- 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 5.9 (1), (2), and (3) are 2, 0, and -2, respectively.

We have the following theorem.

Theorem 5.5.1 *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 5.7 and Figure 5.10. In Figure 5.7, 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 5.10, 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 5.7 with a slight modification as illustrated in Figure 5.10 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 5.10. It is observed in Figure 5.10 that the resulting circuit graph has an NR node and hence we say, our rewriting algorithm fails to remove all NRs.

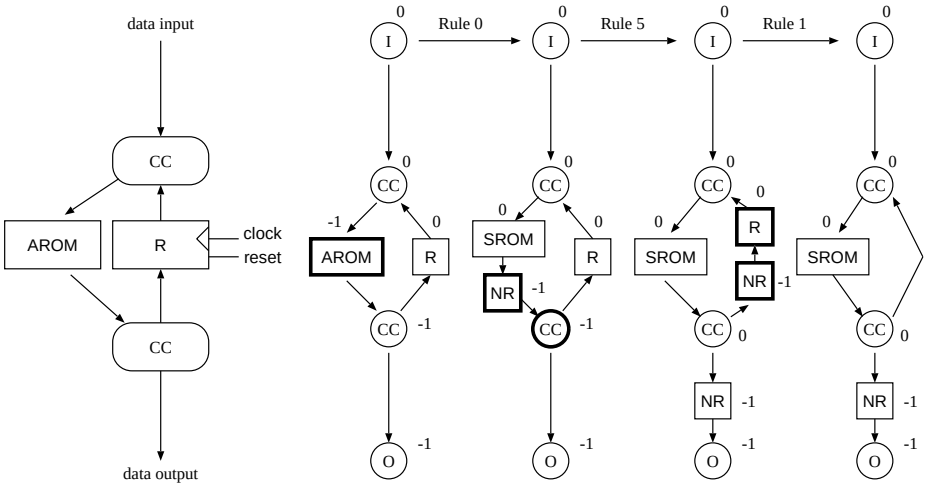


Figure 5.10: A circuit with a cycle and its corresponding circuit graph with a slight modification of the Figure 5.7 that can not be converted into an AROM-free circuit.

5.6 Proof of Theorem 5.5.1

The main purpose of this section is to show a proof of Theorem 5.5.1. We will show several lemmas for a proof of Theorem 5.5.1.

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 5.6.1 *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 5.7. 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 5.6.2 *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 5.7, we see that the cycle increases the potentiality by +1 and this value is also never changed. Readers may refer to the Figure 5.9 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 5.6.3 *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 5.11). To apply Rule 2 and Rule 3 later, the non-R node in Figure 5.11 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.

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

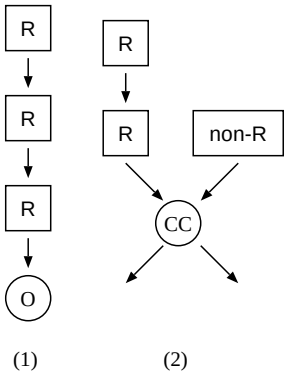


Figure 5.11: Illustration for the proof of Lemma 5.6.3.

Lemma 5.6.4 *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 5.6.3, 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 5.8, 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 5.6.5 *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 5.10 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 5.2 (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 5.6.5, we will prove that all regular SROM and CC nodes in the resulting circuit graph have zero potentiality.

Lemma 5.6.6 *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 5.6.7 *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 5.6.5, 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 5.5.1. From Lemma 5.6.6 and 5.6.7, 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 5.6.5. 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 5.6.1, the potentiality of O nodes does not change by our rewriting algorithm. Thus, from Lemma 5.6.4, 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 5.5.1.

From Theorem 5.5.1, 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 5.5.1. The readers should refer to Figure 5.12 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.

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 cy-

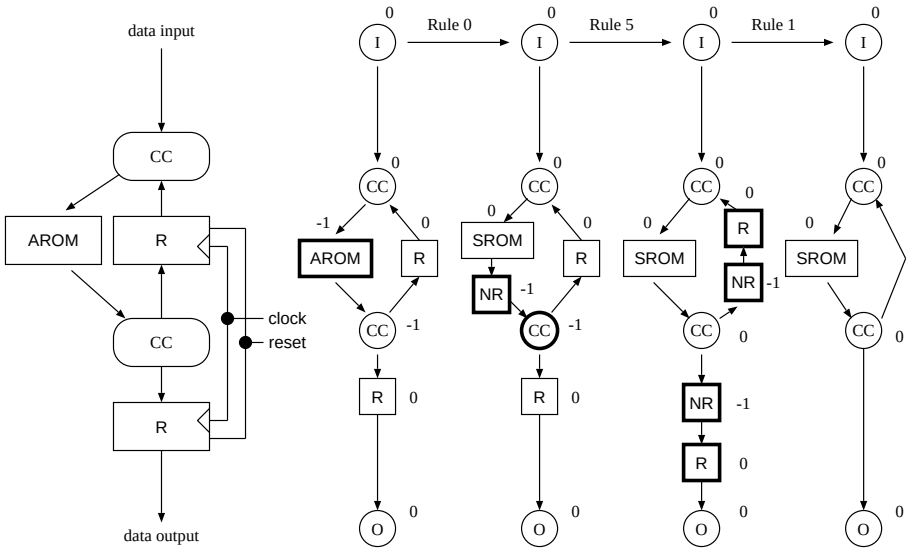


Figure 5.12: An almost equivalent circuit with cycle and it corresponding circuit graph to that of Figure 5.10 that can be converted into an AROM-free circuit.

cles. 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.

5.7 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 5.13 (a). The circuit in Figure 5.13 (a) has one Register (R) and one adder. Register (R) has a reset input and a clock input as illustrated in Section 5.3. Readers may also refer to the Section 5.3 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 5.13 (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 5.13 (a) is deterministic which is similar to other inputs of the DRG circuit. Hence, we treat this citcuit as illustrated in Figure 5.13 (a) as a dummy input to the DRG circuit, as shown in Figure 5.13 (b). Readers may refer to Figure 5.13 (a), where dotted circle is indicating the dummy input for the DRG circuit as shown in Figure 5.13 (b) in which the dummy input is connected to the adder of the DRG circuit. Note that, in Figure 5.13 (b), DRG circuit is shown by enclosed dotted line. If this is the case, then we consider whole circuit in Figure 5.13 (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 5.13 (b) by enclosed dotted line. For the benefit of readers, we have

shown an application of our rewriting algorithm in Figure 5.13 (c). Figure 5.13 (c) represents a converted circuit (by our rewriting algorithm) with no AROMs for the circuit, shown in Figure 5.13 (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 5.13 (c). Obviously, we can conclude from the converted circuit in Figure 5.13 (c) for an input circuit in Figure 5.13 (b) that users can design their input circuit in wider range instead of only considering DRG circuit, shown in Figure 5.13 (b) by enclosed dotted line.

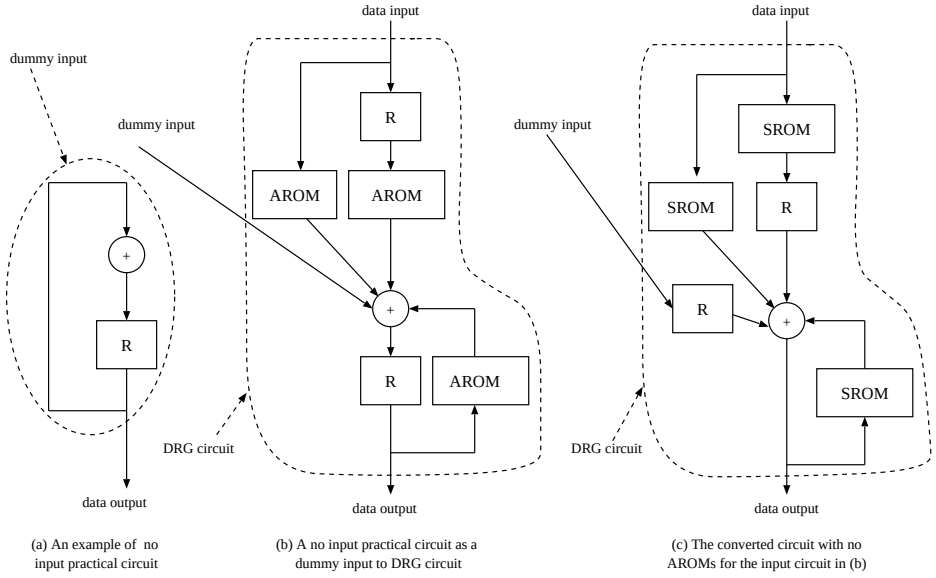


Figure 5.13: An example to extend the input circuit with cycle for our algorithm.

5.8 Concluding Remarks

In this chapter, we have presented a rewriting algorithm and six rewriting rules to obtain the equivalent circuits with Synchronous ROMs (SROMs) for the *practi-*

cal 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. 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 did it automatically.

Chapter 6

Performance Improvement of the Resulting Circuits

In this chapter, we will discuss about the performance improvement of the resulting circuits. First, we will recall our main contribution of this dissertation. The main contribution of this dissertation is to minimize the number of clock cycles in the designed circuits by users. We say that circuit design that minimize the number of clock cycles is easy if we use asynchronous read operation. However, embedded memories in the most modern FPGAs support synchronous read and synchronous write operations but do not support asynchronous read operation. To resolve this problem, we provide circuit rewriting approaches to convert a circuit using AROMs (Asynchronous Read Only Memories) or ARAMs (Asynchronous Random Access Memories) into an equivalent circuit using SROMs (Synchronous Read Only Memories) or SRAMs (Synchronous Random Access Memories) for implementing in FPGAs. More specifically, a circuit designed by users consists

of registers (Rs), combinational circuits (CCs), AROMs supporting asynchronous read operation or ARAMs supporting asynchronous read and synchronous write operations is given. Our circuit rewriting approaches automatically convert the given circuit with AROMs supporting asynchronous read operation or ARAMs supporting asynchronous read and synchronous write operations into an equivalent circuit with SROMs supporting synchronous read operation or SRAMs supporting synchronous read and synchronous write operations. Therefore, the resulting circuit can be embedded into the most modern FPGAs.

However, performance of the resulting circuits may degrade, because, by our rewriting algorithms, registers in the circuits are moved towards the output ports, whenever possible. Although, Performance improvement of the resulting circuits is the beyond of this dissertation. However, we will discuss several techniques to improve the performance of the resulting circuits as follows:

6.1 Circuit Performance

Circuit performance can be measured in terms of latency and clock frequency. Basically, our rewriting algorithms move registers towards the output ports, whenever possible. Hence, in general, the resulting circuits may have the longest paths from input ports to registers/SROMs/SRAMs or from registers/SROMs/SRAMs to registers/SROMs/SRAMs or from registers/SROMs/SRAMs to output ports. Therefore, the resulting AROM-free or ARAM-free circuit has large propagation delay and low clock frequency. Hence, we say that performance of the resulting AROM-free or ARAM-free circuit may be degraded. However, it is easier

to improve the performance of the resulting circuit than minimizing the number of clock cycles by the designers. For the reader's benefit, performance improvement techniques in terms of latency and clock frequency of the resulting circuit are described as follows:

6.1.1 Minimizing latency by eliminating redundant registers

We will describe here how to minimize latency of the resulting circuit. For this purpose, we first define *redundant registers* in the resulting AROM-free or ARAM-free circuit. The registers which are connected to the edges ending at O nodes are called redundant registers. For the benefit of readers, we have shown an example of a resulting circuit, as illustrated in Figure 6.1, where the redundant registers are highlighted. Essentially, the redundant registers only work just as buffers for the output nodes. Thus, the latency of each output port can be decreased by eliminating redundant registers if they do not cause a timing problem for a circuit connected to the output port. Also, we can say that, after removing all redundant registers, the latency is minimized, because no other registers can be deleted.

On the other hand, if all edges ending at O nodes have equal number of redundant registers (Rs), we can eliminate those redundant registers (Rs) in a sense that there is no timing problems at O nodes of the resulting circuits. In this regard, readers may refer to the Figure 6.1. In this figure, it is shown that every edge ending to O node has one equal number of redundant registers (Rs) such that these redundant registers (Rs) can be eliminated for minimizing latency. If this is the case, the latency must be minimized as well as performance of the resulting

circuit is improved definitely.

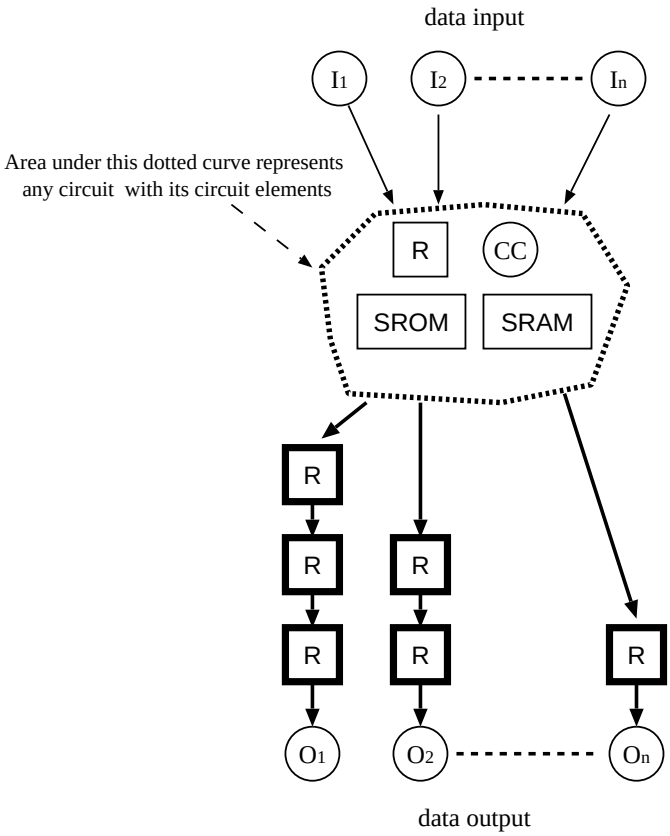


Figure 6.1: An example to minimize latency in the AROM-free or ARAM-free circuit by eliminating redundant registers.

6.1.2 Increasing clock frequency by adding registers

Here, we will describe about the clock performance improvement of the AROM-free or ARAM-free resulting circuit. Recall that, by our rewriting algorithms, registers in the circuits are moved towards the output ports, whenever possible. Hence, resulting circuit may have the longest path. The maximum clock frequency depends on the longest path from input ports to registers/SROMs/SRAMs

or from registers/SROMs/SRAMs to registers/SROMs/SRAMs or

from registers/SROMs/SRAMs to output ports. For the benefit of readers, we have shown the longest path in Figure 6.2(a) and in Figure 6.3(a) by highlighting the arrows. Figure 6.2(a) represents an AROM-free resulting circuit and Figure 6.3(a) represents an ARAM-free resulting circuit. Due to the longest path in the clock dependent circuits (i.e. circuits with Rs, CCs and SROMs or SRAMs in our case), clock performance of those circuits must be degraded. To overcome of this problem, we need to divide the AROM-free or ARAM-free resulting circuit (when no rule is applicable) into several layers so that the longest path becomes shorter. Designers can select the layers properly in order to make the longest path into shorter for getting optimum clock performance. In fact, proper selection of the layers in the AROM-free or ARAM-free resulting circuit for getting optimum clock performance may be another research work. This topic is beyond of this dissertation. For the benefit of readers, we have shown two examples in Figure 6.2(a) and in Figure 6.3(a). Figure 6.2(a) and Figure 6.3(a) show the examples of two layers. The cutting points (created by layers and edges) are highlighted by the bullet circles in these figures. After that, registers are added at every cutting point in the AROM-free or ARAM-free resulting circuit such that longest path of the AROM-free or ARAM-free resulting circuit becomes shorter, as illustrated in Figure 6.2(b) and Fig. 6.3(b). Due to shorter path instead of the longest path, the clock performance of the AROM-free or ARAM-free resulting circuit must be improved definitely. In this case, we can ignore the latency of the added registers in the AROM-free or ARAM-free resulting circuit.

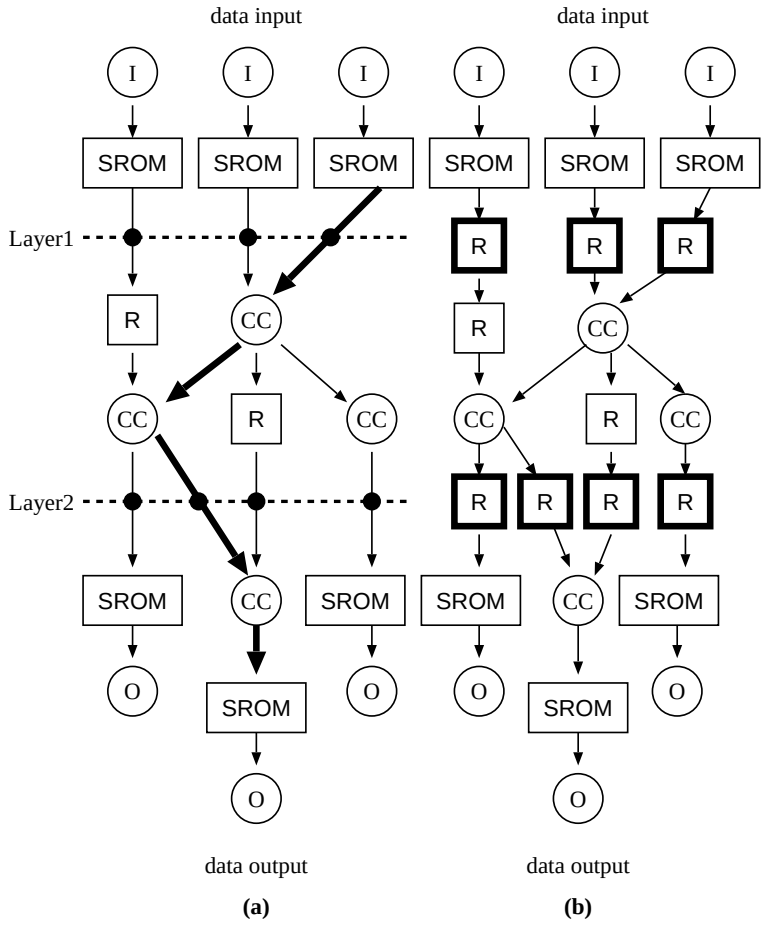


Figure 6.2: An example for improving the clock performance in the AROM-free circuit.

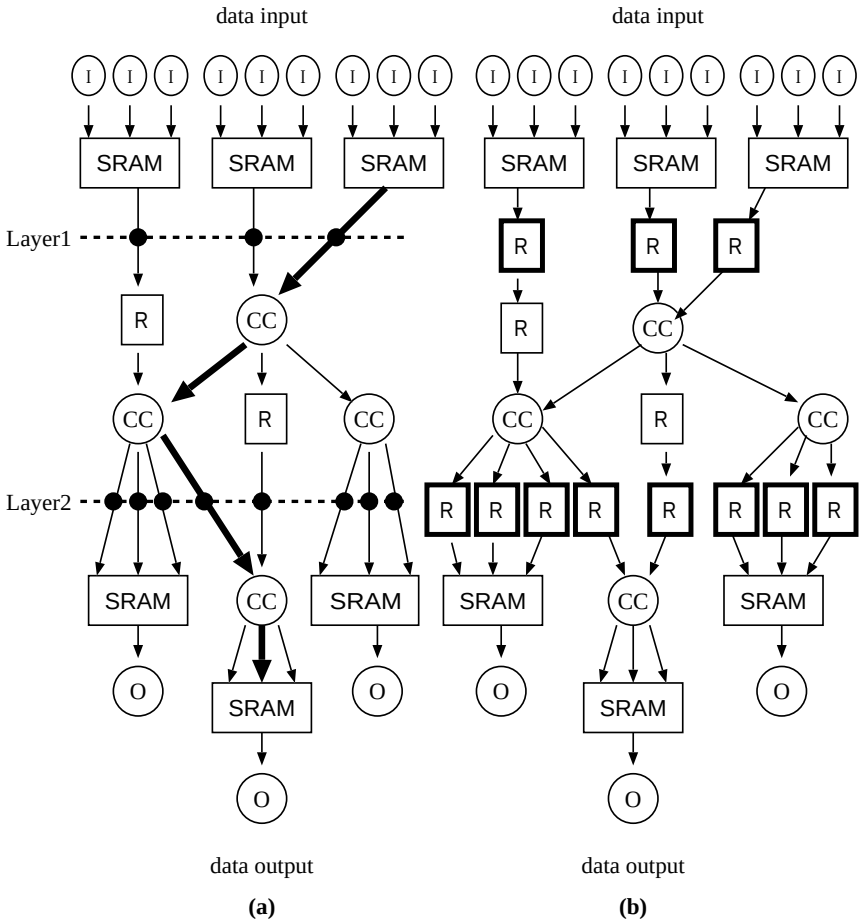


Figure 6.3: An example for improving the clock performance in the ARAM-free circuit.

Chapter 7

Conclusions

7.1 Summary

In this dissertation, we have presented circuit rewriting algorithms to convert circuits with memories supporting asynchronous read operation or circuits with memories supporting asynchronous read and synchronous write operations into the equivalent circuits with memories supporting synchronous read operation or equivalent circuits with memories supporting synchronous read and synchronous write operations for implementing in FPGAs.

First, we have shown a *circuit rewriting approach* for converting circuit with asynchronous ROMs into an equivalent circuit with synchronous ROMs for implementing in FPGAs. For the purpose of circuit conversion, we have presented a *circuit rewriting algorithm* and *five rewriting rules*. In fact, our algorithm invokes the rules and applied repeatedly in the current circuit until no applying is possible such that we have an AROM-free circuit to implement in the current FPGAs. Us-

ing our rewriting algorithm, any sequential circuit with AROMs can be converted into an equivalent fully synchronous sequential circuit with no AROMs to support the modern FPGA architecture. Although, this circuit conversion by our approach is not trivial. However, our circuit rewriting algorithm can do it automatically. We briefly discuss the techniques to improve performance of the AROM-free resulting circuit and also describe a technique for applying our rewriting algorithm even if a user designs a circuit with pipeline structure.

Next, we have presented a *circuit rewriting algorithm* and *five rewriting rules* to convert a circuit with ARAMs into an equivalent circuit with no ARAMs for the current FPGA considering *both read and write operations* of the memory blocks (RAMs). However, in our previous work mentioned earlier (followed by Chapter 3), we considered *only read operation* of the memory blocks (ROMs). In fact, we improved our previous work, described in Chapter 3, where RAMs can be used as the additional circuit elements to the given input circuits. It is not trivial to convert a sequential circuit with ARAMs into an equivalent fully synchronous circuit with no ARAMs for supporting the modern FPGA architecture. However, our algorithm is able to do it automatically. We also briefly discuss the techniques to improve performance of the ARAM-free resulting circuit.

Next, a *modified circuit rewriting algorithm* is presented to convert a circuit *with cycles* using AROMs into an equivalent circuit *with cycles* using SROMs for implementing in FPGAs. In fact, our modified circuit rewriting algorithm is able to convert practical circuits which have *cycles* using *six rewriting rules*. In our previous works, mentioned above (followed by the Chapter 3 and Chapter 4), we have presented circuit rewriting algorithms to convert a circuit with asynchronous

ROMs or asynchronous RAMs into an equivalent circuit with synchronous ones. The resulting circuit with synchronous ROMs or synchronous RAMs can be embedded into FPGAs. However, these circuit rewriting algorithms can handle circuits represented by a directed acyclic graph (DAG) and do not work for those with cycles. The work in this chapter , 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. We briefly discuss the techniques to improve performance of the AROM-free resulting circuit and also describe a technique to generate AROM-free circuit even if the input circuit is beyond of the directed reachable graph (DRG) circuit.

Finally, we have discussed several techniques to improve the performance of the resulting circuits. By our rewriting algorithms, performance may degrade of the resulting circuits, because, our rewriting algorithms move registers towards the output ports, whenever possible. Hence, in general, the resulting circuits may have the longest paths from input ports to registers/SROMs/SRAMs or from registers/SROMs/SRAMs to registers/SROMs/SRAMs or from registers/SROMs/SRAMs to output ports. As a result, performance of the resulting circuits may be degraded in terms of latency and clock frequency. However, it is possible to improve circuit performance of the AROM-free or ARAM-free resulting circuit as illustrated in Chapter 6, which is easier than minimizing the number of clock cycles, although performance improvement of the AROM-free or ARAM-free resulting circuit is beyond of this dissertation.

References

[1] 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.

[2] 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.

[3] S. S. C. Design of an FPGA logic element for implementing asynchronous NULL convention logic circuits. *IEEE Transactions on very large scale integration (VLSI) system*, 15(6):672–683, June 2007.

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

[5] T. Gerard. *Introduction to Distributed Algorithms*. Cambridge University Press, 2nd edition, 2001.

[6] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison Wesley, 2nd edition, 2003.

[7] N. Huot, H. Dubreuil, L. Fesquet, and M. Renaudin. FPGA architecture for multi-style asynchronous logic. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 32–33, 2005.

[8] S. Ishira, Y. Komatsu, M. Hariyama, and M. Kameyama. An asynchronous field-programmable VLSI using LEDR/4-phase-dual-rail protocol converters. In *The In-*

ternational Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), Monte Carlo Resort, Las Vegas, Nevada, USA, July 2009.

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

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

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

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

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

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

[16] 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.

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

[18] 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.

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

Acknowledgment

First and foremost, I would like to express my sincere gratitude to my adviser, Professor Koji Nakano for his continuous encouragement, advice and support. His knowledge and research experience are in value through the whole period of my Ph.D. study. He is acknowledged, in particular, for his kindness, generous guidance and illuminating discussions. As an advisor, he taught me practices and skills that will benefit my future academic career.

I wish to express sincere appreciation to my thesis committee members, Professor Satoshi Fujita and Associate Professor Shuichi Ohno for taking time to review my dissertation and giving helpful comments.

I also wish to express my heartful thanks to Assistant Professor Yasuaki Ito for his invaluable help throughout the study. My heartiest thanks go to all members of computer system laboratory. They were always kind and very keen to help.

I am highly obligated to the Japanese Government for financial support to pursue my study in Japan. I am grateful to all the faculty members and staffs of the Department of Information Engineering, Hiroshima University.

Last but not least, I wish to express my thanks to my family who has always supported me. My special thanks go to my wife Shema and my son Shoumo.

List of publications

Journals

[J-1] Md. Nazrul Islam Mondal, Koji Nakano, Yasuaki Ito, A Rewriting Approach to Replace Asynchronous ROMs with Synchronous Ones for the Circuits with Cycles, International Journal of Networking and Computing, Vol. 2, No.2, pp.269-290, July 2012.

[J-2] Md. Nazrul Islam Mondal, Koji Nakano, Yasuaki Ito, An Algorithm to Obtain Circuits with Synchronous RAMs, Journal of Communication and Computer, Vol. 9, No.5, pp.547-559, May 2012.

[J-3] Md. Nazrul Islam Mondal, Koji Nakano, Yasuaki Ito, A Graph Rewriting Approach for Converting Asynchronous ROMs into Synchronous Ones, IEICE Transactions on Information and Systems, Vol. E94-D, No.12, pp.2378–2388, December 2011.

International Conferences

[I-1] Md. Nazrul Islam Mondal, Koji Nakano, Yasuaki Ito, An Algorithm to Remove Asynchronous ROMs in Circuits with Cycles, Proc. of International Conference on Networking and Computing, pp. 77–86, December 2011.

[I-2] Md. Nazrul Islam Mondal, Koji Nakano, Yasuaki Ito, A Rewriting Algorithm to Generate AROM-free Fully Synchronous Circuits, Proc. of International Conference on Networking and Computing, pp. 148–157, November 2010.