Energy-Aware Interconnect Resource Reduction Through Buffer Access Manipulation for Data-Centric Applications

Woohyung Chun, *Student Member, IEEE*, Sungroh Yoon, *Member, IEEE*, and Sangjin Hong, *Senior Member, IEEE*

***Abstract*—This paper presents a methodology for reducing interconnect resources in reconfigurable platforms such as field-programmable gate arrays (FPGAs). This methodology utilizes the techniques developed for the buffer-based dataflow, a new design representation suitable for implementing data-centric applications in a reconfigurable platform. In a buffer-based dataflow, nodes correspond to processing blocks and buffer controllers represent the interconnects between the processing blocks. Since we can isolate the functional execution and data transfer of each node by using buffer controllers, a buffer-based dataflow is helpful for reducing overall design time and for increasing reconfigurability. In this paper, we propose a sharing methodology that can reduce the buffer memory and the number of buses used in the realization of a buffer-based dataflow. By reducing the resources allocated to buffer controllers, we can achieve interconnect resource reduction. The proposed sharing methodology can increase the dynamic energy consumption due to the increased port-loading capacitance. By using the energy consumption model determined by the costs of buffers and buses, we investigate whether the sharing case with the minimum resources corresponds to the sharing case consuming the minimum energy or not. We evaluate the proposed sharing methodology with the dataflow graphs representing data-centric applications such as SIRF, IPv4, MC-CDMA transmitter and receiver.**

***Index Terms*—Buffer-based dataflow, interconnect resource reduction, field-programmable gate array (FPGA), reconfigurable architecture.**

I. Introduction

FOR complex designs, the greatest impact on performance, costs, and functionality can be made at the architectural level [1]. Top-down design methodologies proceeding from the architectural to lower levels, are thus becoming popular. This design trend relaxes redesign efforts at higher levels of system assembly [2], and most designers endeavor to simplify the complex system under design from an early stage.

One of the most popular simplifying methods is to represent a target application as a dataflow graph. Especially, modeling digital signal processing (DSP) applications through coarse-grained dataflow graphs is widespread and is adopted in many high-level design frameworks [3]. When a dataflow graph is synthesized in a fine-grained reconfigurable platform such as field-programmable gate arrays (FPGAs), the interconnects of the dataflow consume more resources than the functional units do [4]. In addition, the implementation of multiplexors for data routing is very expensive in typical FPGA design [5]. In [6], Cong *et al.* presented a method that can bind a dataflow graph to a parameterizable register-file microarchitecture in order to reduce the interconnect and multiplexor complexity. However, this approach is limited in terms of reducing the number of multiplexors because the microarchitecture still uses multiplexors for data routing.

In this paper, we use the techniques developed for buffer-based dataflow representations [7], in which buffers are inserted between the nodes of a dataflow graph. This technique is applicable to a subtype of synchronous dataflow (SDF) graphs [8] where the number of samples produced and consumed is always unity since the input and output data of the buffers have the same size. Due to the buffers, data transfers between nodes can be separated from the functional executions of nodes [9]. Thus, a buffer-based dataflow can increase the reusability and reconfigurability of the interconnects between nodes. Moreover, the data transfers between nodes can be *timely multiplexed with the buffer controller parameters* at the level of a dataflow. This allows the designer to use tristate buffers for efficiently routing data transfers; the tristate buffers are activated by the timely multiplexed signals represented with buffer controller parameters.

Since the buffers in a buffer-based dataflow represent the interconnects between nodes, we propose a methodology for sharing buffers in a buffer-based dataflow in order to reduce the number of the interconnects required. Buffer sharing is a well-known technique to reduce system memory in register transfer level (RTL) designs. The register allocation problem occurring in compiler design is also related. Existing buffer-sharing techniques merge those buffers that have non-overlapping lifetimes into one [10]–[13]. These approaches are based on buffer lifetime analysis such that they are limited to control the lifetimes of buffers provided by a target architecture. In this paper, buffers are inserted in order to parameterize data transfers through edges at the level of a dataflow graph. Sharing buffers and buses can thus be manipulated with the parameters of buffer controllers to reduce interconnect resources.

In the proposed approach, buffer lifetimes are fully controlled in a given time constraint in order to increase the possibility of

Manuscript received September 11, 2009; revised December 03, 2009. First published March 22, 2010; current version published April 27, 2011. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0000407 and No. 2010-0000631).

W. Chun and S. Hong are with the Department of Electrical and Computer Engineering, Stony Brook University—SUNY, Stony Brook, NY 11794-2350, USA (e-mail: (wchun, snjhong)@ece.sunysb.edu).

S. Yoon is with the School of Electrical Engineering, Korea University, Seoul 136-713, Republic of Korea (e-mail: sryoon@korea.ac.kr).

Digital Object Identifier 10.1109/TVLSI.2010.2042087

1063-8210/$26.00 © 2010 IEEE

buffer sharing. Our buffer sharing method moves (or translates) buffer lifetimes within the time constraint so that the buffer lifetimes become non-overlapped with each other. Furthermore, for a finer-grained control of data transfers, a buffer lifetime is separated into two: reading-activity time and writing-activity time. In case activity times are non-overlapped, they are allocated to the same interconnect (i.e., bus) to reduce the number of buses. Thus, our bus-sharing method also translates activity times within the time constraint so that the translated activity times are non-overlapped with each other as much as possible.

Even though the techniques to share buffers and buses can reduce the synthesizable resources, they may increase the total energy consumption when the runtime operation such as data transfers has the dominant effect on the total energy consumption. When the buffers having different sizes of memory are merged to one, it is possible that a large buffer memory is activated by a small buffer-memory access. Therefore, the energy consumption by the buffer memory activation becomes larger after buffer sharing is applied. In addition, since the bus sharing increases the number of ports in a bus, bus sharing increases the energy consumption of data transfers due to the increased port-loading capacitance. Thus, the sharing case consuming the minimum energy may be different from the sharing case demanding the minimum resources, depending on the buffer and bus costs of the target system used. In order to identify the fact that the sharing case consuming the minimum energy does not correspond to the sharing case demanding the minimum resource, by referring to [14]–[16], we construct an energy consumption model with the estimated buffer and bus costs.

The rest of this paper is organized as follows. Section II reviews the method to construct a buffer-based dataflow with buffer controller parameters. Section III provides the overview of the proposed approach to reduce interconnect resources by using a buffer-based dataflow. In Section IV, we propose a sharing methodology to reduce buffers and buses in a buffer-based dataflow. In the end of Section IV, we establish an energy consumption model with estimated buffer and bus costs. We also propose an algorithm to find the sharing case based on the energy consumption model. Section V evaluates the proposed method described in Section IV. In the evaluation of energy consumption, we observe that the sharing case consuming the minimum energy does not correspond to the sharing case having the minimum resources. Finally, Section VI summarizes our contribution.

II. Background

A. Buffer-Based Dataflow Representation

Most real-time signal processing algorithms consider sets of data as frames. Such systems possess two unique characteristics of execution. First, they can be represented as dataflow graphs which represent data dependency between processing blocks (nodes) [17]. Second, each node in the dataflow executes a set of data elements per frame (iteration). Fig. 1 shows a buffer-based dataflow that is constructed by inserting buffers between nodes.

In Fig. 1(a), a node can be regarded as the source and destination of data. As shown in Fig. 1(b), this source-destination relationship can be isolated by inserting buffers between nodes.

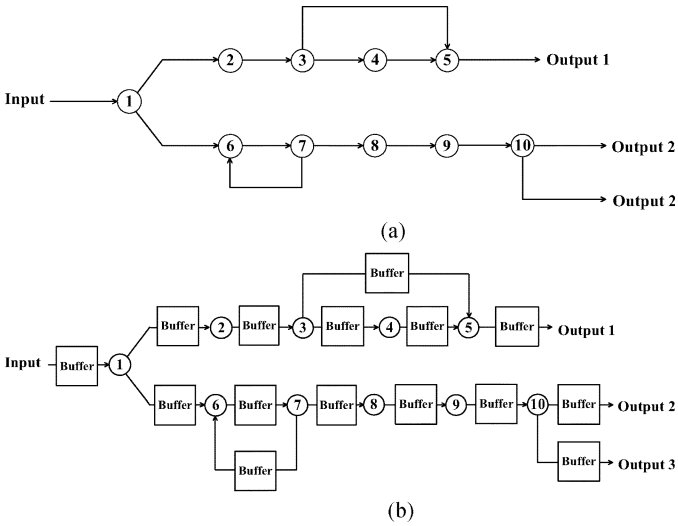


Fig. 1. Buffer-based dataflow derived from a dataflow by buffer insertion. (a) A dataflow graph. (b) Buffer insertion.

By separating the relationships between nodes, nodes only represent their functionality. The isolation also facilitates to reconfigure the overall system.

In a buffer-based dataflow graph, inserting a buffer to an edge represents delivering a data frame from its source to destination. Thus, the size of data frames appearing at the input port of a buffer is identical to the size of data frames at the output port of the buffer. Furthermore, while a source node is writing data to a buffer, the corresponding destination node is able to read data from the buffer. Therefore, buffers in a buffer-based dataflow can be realized as the dual-port memory which allows simultaneous writing and reading access.

Let $BC_{i,j}$ denote the buffer between the producer node i and the consumer node j. The primary parameters which determine the buffer controller structure and overall physical realization are the following [7]: logic latency (L_i), write offset $(nw_{i,j})$, read offset $(nr_{i,j})$, block size $(M_{i,j})$, and delay factor $(D_{i,j})$. The logic latency L_i is the latency of node i. The write offset $nw_{i,j}$ represents the difference between reading data from the previous buffer and writing data to the current buffer without considering L_i. The read offset $nr_{i,j}$ is the offset from the start of writing data to $BC_{i,j}$ to the start of reading data from $BC_{i,j}$ when the writing speed of node i and the reading speed of node j are matched. However, if the writing speed of node i is slower than the reading speed of node j, node j does not read valid data from $BC_{i,j}$. The delay factor $D_{i,j}$ is to represent this rate mismatch between nodes i and j. The block size $M_{i,j}$ characterizes the data size generated by node i. It also determines the maximum storage requirement of $BC_{i,j}$. The unit of all the parameters described above is the unit cycle of the target platform used.

B. Buffer Controller Parameter Characteristics

In order to construct a buffer-based dataflow, a table called the *buffer controller parameter table* should be extracted from the operational dependency of a dataflow and the offsets of fan-ins and fan-outs of processing elements.

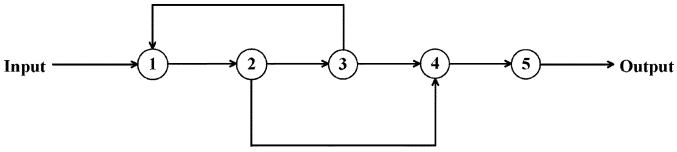


Fig. 2. Example dataflow.

TABLE I
OPERATIONAL DEPENDENCY DERIVED FROM FIG. 2

	Operational dependency
$e_{input,1}$	$start\_write_{input,1} < start\_read_{input,1}$
$e_{1,2}$	$start\_write_{1,2} < start\_read_{1,2}$
$e_{2,3}$	$start\_write_{2,3} < start\_read_{2,3}$
$e_{2,4}$	$start\_write_{2,4} < start\_read_{2,4}$
$e_{3,1}$	$start\_write_{3,1} < start\_read_{3,1}$
$e_{3,4}$	$start\_write_{3,4} < start\_read_{3,4}$
$e_{4,5}$	$start\_write_{4,5} < start\_read_{4,5}$
$e_{5,output}$	$start\_write_{5,output} < start\_read_{5,output}$
node 1	$\max\{start\_read_{input,1}, start\_read_{3,1}\} + L_1 < start\_write_{1,2}$
node 2	$start\_read_{1,2} + L_2 < \min\{start\_write_{2,3}, start\_write_{2,4}\}$
node 3	$start\_read_{2,3} + L_3 < start\_write_{3,4}$
node 4	$\max\{start\_read_{3,4}, start\_read_{2,4}\} + L_4 < start\_write_{4,5}$
node 5	$start\_read_{4,5} + L_5 < start\_write_{5,output}$

The fan-ins and fan-outs offsets represent the input and output characteristics of a node. When a node has multiple fan-ins, the timing differences between reading data from the multiple fan-ins determine the fan-ins offset. For example, suppose that node i has two fan-ins and that the start time of reading data from one fan-in port is ten cycles earlier than the start time of reading data from the other fan-out port. The fan-ins offset of node i is then represented as [0, 10]. If a node has multiple fan-outs, the timing differences between writing data to the multiple fan-outs determine the fan-outs offset. The fan-ins and fan-outs offsets and the operational dependency determine the functional characteristics of a dataflow. Therefore, when a dataflow is converted to a buffer-based dataflow by inserting buffer controllers, the fan-ins and fan-outs offsets and the operational dependency *must* be preserved.

The operational dependency of a dataflow is represented by edge activity times and node execution times. Edge activity times represent the timing relation of data transfers through edges, and node execution times represent the time to process data from fan-in and fan-out edges.

For the dataflow shown in Fig. 2, Table I represents the edge activity times and node execution times. In Table I, $e_{i,j}$ represents the edge from source node i to destination node j. start_write$_{i,j}$ and start_read$_{i,j}$ represent the start times of writing data and reading data through $e_{i,j}$, respectively. The operational dependency of each edge is described from $e_{\text{input},1}$ to $e_{5,\text{output}}$. This dependency simply represents start_write$_{i,j}$ < start_read$_{i,j}$ because writing data through an edge always precedes reading data through the edge. In addition, since the edge is connected to the fan-in/fan-out port of a node, the functional execution of the node determines the operational dependency between fan-in and fan-out edges. The operational dependency between fan-in and fan-out edges are listed from node 1 to node 5. In case of node 3 in Table I, $start\_write_{3,1}$ is removed to prevent the deadlock condition. That is, if node 3 generates data by referring to the data read from node 2 in the current iteration period, node 1 keeps waiting for the data from node 3 while node 2 waits for the data from node 1, and node 3 also keeps waiting for the data from node 2.

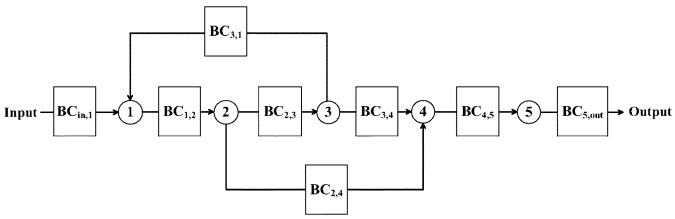


Fig. 3. Buffer-based dataflow derived from Fig. 2.

Fig. 3 shows the buffer-based dataflow converted from the dataflow of Fig. 2. Since $BC_{i,j}$ has the operational characteristics of $e_{i,j}$ in Table I, start_write$_{i,j}$ and start_read$_{i,j}$ of Table I are used to represent the operational dependency of this buffer-based dataflow. In the buffer controller $BC_{i,j}$, the start signals are realized with the primary parameters introduced in Section II-A as follows:

$$\text{start\_write}_{i,j} = L_i + nw_{i,j} + \text{start}_i \quad (1)$$

$$\text{start\_read}_{i,j} = \text{start\_write}_{i,j} + \max\{nr_{i,j}, D_{i,j}\} \quad (2)$$

$$\text{stop\_write}_{i,j} = \text{start\_write}_{i,j} + M_{i,j} \quad (3)$$

$$\text{stop\_read}_{i,j} = \text{start\_read}_{i,j} + M_{i,j}. \quad (4)$$

In (1), start$_i$ is the time value in which node i begins reading data from the previous buffer controller through its fan-in port. Equation (2) reflects the rate mismatch between source node i and destination node j. If the writing speed of node i is slower than the reading speed of node j, the reading of $BC_{i,j}$ may finish before the writing of $BC_{i,j}$. In this case, node j does not read the whole data generated from node i. In order to prevent wrong data transfers due to the mismatch of writing and reading speed, node j starts to read data from $BC_{i,j}$ when $\max\{nr_{i,j}, D_{i,j}\}$ passes from the start of writing. Equations (3) and (4) represent the end time of writing and reading data to/from $BC_{i,j}$, respectively. In case a node sends a portion of data and then sends the rest later (i.e., a data transfer is split), the split data transfer can be represented with the delay factor D. For example, in Fig. 3, when node 1 writes half of $M_{1,2}$ to $BC_{1,2}$ first and then writes the rest half after the delay of $D_{1,2}$, the following holds:

$$\text{start\_write}_{1,2}(1) = L_1 + nw_{1,2} + \text{start}_1 \quad (5)$$

$$\text{stop\_write}_{1,2}(1) = \text{start\_write}_{1,2}(1) + \frac{M_{1,2}}{2} \quad (6)$$

$$\text{start\_write}_{1,2}(2) = \text{stop\_write}_{1,2}(1) + D_{1,2} \quad (7)$$

$$\text{start\_write}_{1,2}(2) = \text{start\_write}_{1,2}(2) + \frac{M_{1,2}}{2} \quad (8)$$

where start_write$_{1,2}(1)$ and stop_write$_{1,2}(1)$ correspond to the start and stop signals for the first part of data written to $BC_{1,2}$,

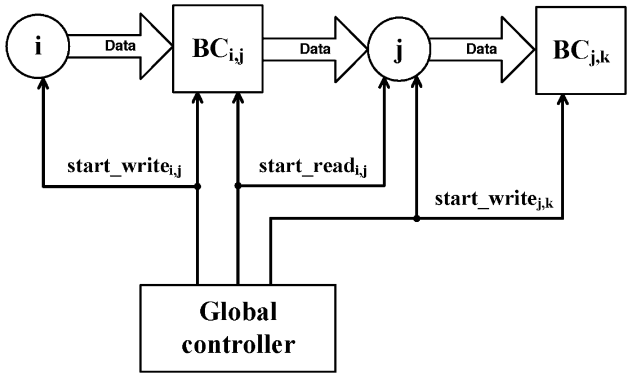


Fig. 4. Buffer controllers are synchronized by a global controller.

respectively, and $\mathrm{start\_write}_{1,2}(2)$ and $\mathrm{stop\_write}_{1,2}(2)$ are the start and stop signals for the second part of data written to $BC_{1,2}$, respectively. The delay between the first and second writings is represented as $D_{1,2}$ in (7).

To synchronize the data transfer between a node and a buffer controller, $\mathrm{start\_write}$ and $\mathrm{start\_read}$ are generated by a global controller, as shown in Fig. 4.

III. Approach and Objectives

Fig. 5 illustrates the realization of a dataflow in a reconfigurable platform. When a dataflow is synthesized in the reconfigurable target platform such as FPGA, the edges are realized through preassigned interconnects. If a dataflow is converted to a buffer-based dataflow, inserting buffer controllers doubles the number of interconnects as shown in Case (b). However, the buffer-based dataflow controls data transfers by using the buffer controller parameters introduced in Section II. Therefore, the data transfer control does not require any change of the functional characteristics of nodes. *If data transfers are controlled to use the same interconnects*, the number of interconnects may be reduced.

In Fig. 5, data transfers are controlled by applying the buffer and bus sharing technique to the buffer-based dataflow. As the result of sharing, compared with Case (a), the number of interconnects is reduced by two in Case (d). In Fig. 5, $BC_{(\mathrm{in},1)(2,3)(3,\mathrm{out})}$ is the buffer controller which $BC_{\mathrm{in},1}$, $BC_{2,3}$ and $BC_{3,\mathrm{out}}$ are merged into. Since the buffer controller corresponds to the edge of the original dataflow, buffer sharing reduces the number of interconnects (i.e., buses) in a target platform. Fig. 6 illustrates the control of data transfers for the buffer sharing shown in Fig. 5. The rectangles enclosed with solid lines correspond to the data transfer times of the buffer controllers in the y-axis. The rectangles with dotted lines and arrows represent delayed data transfers. The data transfers corresponding to the solid rectangles require four buffer controllers because the data transfer times are overlapped in the iteration period. However, if a time constraint is larger than the iteration period such that the data transfers corresponding to the dotted rectangles are delayed to be non-overlapped, the number of buffer controllers required for data transfers is reduced into two. Therefore, in Fig. 6, the data transfers of $BC_{\mathrm{in},1}$, $BC_{2,3}$, and $BC_{3,\mathrm{out}}$ are done through $BC_{(\mathrm{in},1)(2,3)(3,\mathrm{out})}$ of Fig. 5.

For a finer-grained control of data transfers, we further divide the data transfer of a buffer controller into two parts, namely writing and reading. Fig. 7 illustrates the case where the writing and reading parts of the buffer controllers in Fig. 6 are arranged to be non-overlapped within a time constraint. Note that non-overlapped writings and readings can be assigned to the same bus. Thus, the bus sharing of Fig. 5 is achieved by assigning the reading of $BC_{\mathrm{in},1}$ to one bus and all others to the same bus. Compared with the case where the original dataflow is directly realized, the number of buses is reduced into two.

IV. Proposed Methodology

A. Buffer Lifetime and Buffer Sharing

The buffer lifetime of $BC_{i,j}$ is defined as the time period from the start time when node i writes the first data to $BC_{i,j}$ to the end time when node j reads the last data from $BC_{i,j}$. Thus, the buffer lifetime of $BC_{i,j}$, $T(BC_{i,j})$ is given by

$$T(BC_{i,j}) = \mathrm{stop\_read}_{i,j} - \mathrm{start\_write}_{i,j}. \tag{9}$$

Fig. 8 illustrates the parameterized buffer-based dataflow corresponding to the dataflow in Fig. 1(a). The buffer-based dataflow shown in Fig. 8 has the minimum iteration period possible. In Fig. 8, the difference between $\mathrm{start\_read}$ and $\mathrm{start\_write}$ of each buffer is minimal. For example, according to the operational dependency in $BC_{1,2}$ (i.e., $\mathrm{start\_read}_{1,2} > \mathrm{start\_write}_{1,2}$), $\mathrm{start\_read}_{1,2} = \mathrm{start\_write}_{1,2} + 1$. Therefore, the buffer lifetime $T(BC_{1,2})$) has its minimal value. For the same reason, other buffer controllers have their minimal buffer lifetimes. If a dataflow has multiple outputs, the end of an iteration period is the maximum value among $\mathrm{stop\_reads}$ of outputs. Thus, the iteration period of Fig. 8 is from $\mathrm{start\_write}_{0,1}$ to $\mathrm{stop\_read}_{10,\mathrm{out}2}$. Since buffer controllers are shared when buffer lifetimes are non-overlapped, buffer lifetimes are translated in order to create non-overlapped areas between lifetimes. The range of translated lifetimes is defined as $\mathrm{iteration\_period\_slack}$

$$\begin{aligned}\mathrm{iteration\_period\_slack} = {} & \mathrm{given\_constraint} \\ & - \min\{\mathrm{iteration\_period}\}\end{aligned} \tag{10}$$

where $\mathrm{given\_constraint}$ comes from the application specification and $\min\{\mathrm{iteration\_period}\}$ is the minimum iteration period of the buffer-based dataflow corresponding to the application. Thus, when $\mathrm{iteration\_period\_slack} > 0$, buffer lifetimes can be translated.

Buffer sharing is done through merging buffer controllers. In order to merge $BC_{i,j}$ and $BC_{k,l}$ into the merged buffer controller $BC_{(i,j)(k,l)}$, the lifetimes of $BC_{i,j}$ and $BC_{k,l}$ must satisfy the following condition:

$$\begin{aligned}&\max\{\mathrm{stop\_read}_{i,j}, \mathrm{stop\_read}_{k,l}\} \\ &\quad - \min\{\mathrm{start\_write}_{i,j}, \mathrm{start\_write}_{k,l}\} \\ &\quad > T(BC_{i,j}) + T(BC_{k,l})\end{aligned} \tag{11}$$

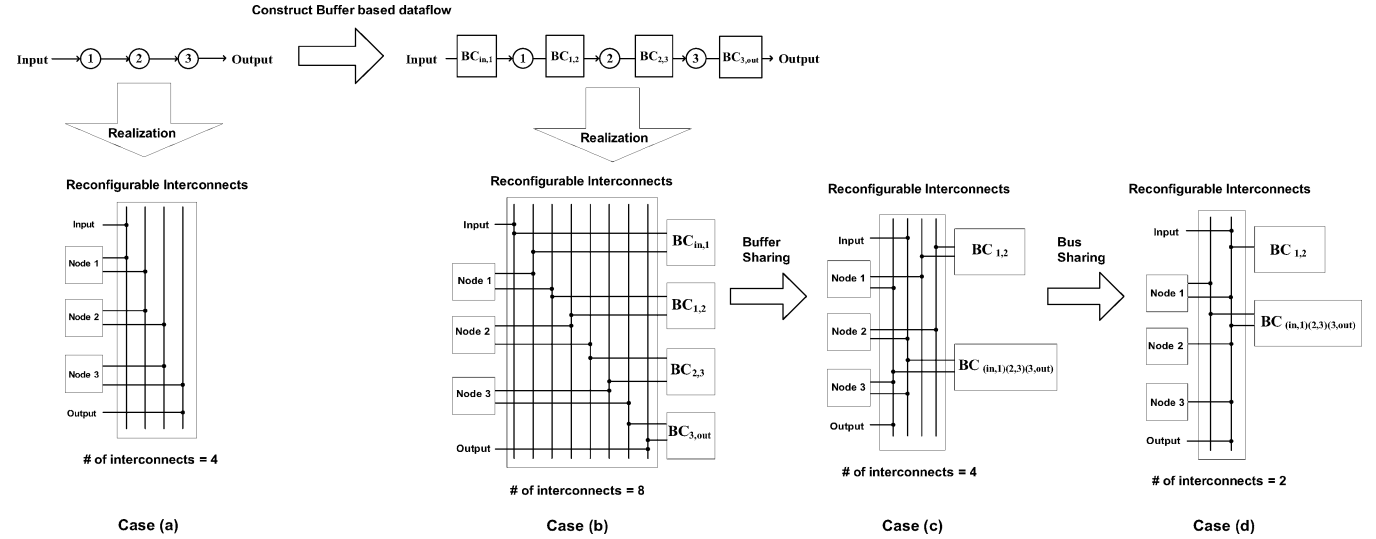


Fig. 5. Realization of a dataflow in a reconfigurable platform.

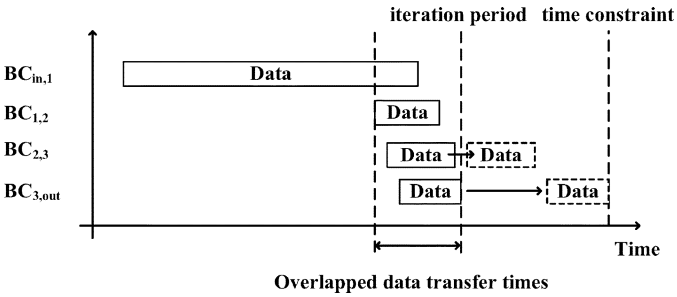


Fig. 6. Control of data transfers for buffer sharing in Fig. 5.

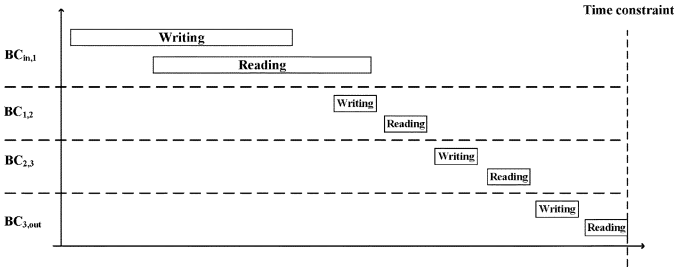


Fig. 7. Finer-grained view of the data transfers shown in Fig. 6.

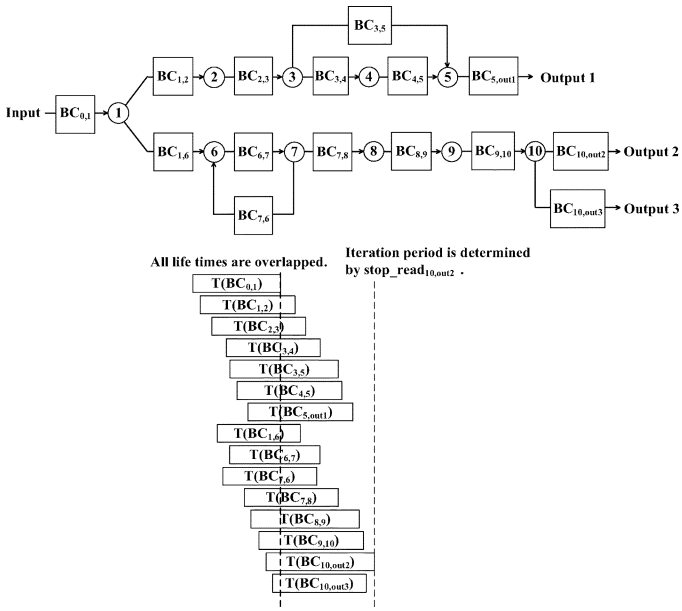


Fig. 8. Parameterized buffer-based dataflow derived from Fig. 1(a).

where $i \neq k, j \neq l$. Equation (11) represents that two lifetimes are non-overlapped if the difference between the maximum $\mathrm{stop\_read}$ and the minimum $\mathrm{start\_write}$ of two lifetimes is larger than the summation of the two lifetimes.

The first step of the proposed buffer sharing is to separate buffer lifetimes in a coarse-grained manner. For the separation, buffer lifetimes are moved only to the right. The next step is to translate buffer lifetimes in a finer-grained manner. In order to decrease the number of overlapped lifetimes, buffer lifetimes are moved to either left or right. When all separation and translation steps are completed, buffer controllers are merged according to (11).

Algorithm 1 Maximal separation scheme

1: **MAX_SEPARATION**$(\forall BC_{i,j}, T(BC_{i,j}))$
2:
3: **while** (true) **do**
4: **if** (node j is an output) and (node i has a single fan-out) **then**
5: $nw_{i,j} \leftarrow nw_{i,j} + \mathrm{given\_constraint} - \mathrm{stop\_read}_{i,j}$;
6: **else**
7: Count the number of overlapped lifetimes;
8: Record the counting number to $\mathrm{cnt\_bf\_tr}$;
9: $nw_{i,j} \leftarrow nw_{i,j} + \mathrm{iteration\_period\_slack}$;
10: Count the number of overlapped lifetimes;
11: Record the counting number to $\mathrm{cnt\_af\_tr}$;
12: **if** $(\mathrm{cnt\_bf\_tr} \leq \mathrm{cnt\_af\_tr})$ **then**
13: $nw_{i,j} \leftarrow nw_{i,j} - \mathrm{iteration\_period\_slack}$;
14: break;
15: **end if**
16: **end if**
17: **end while**

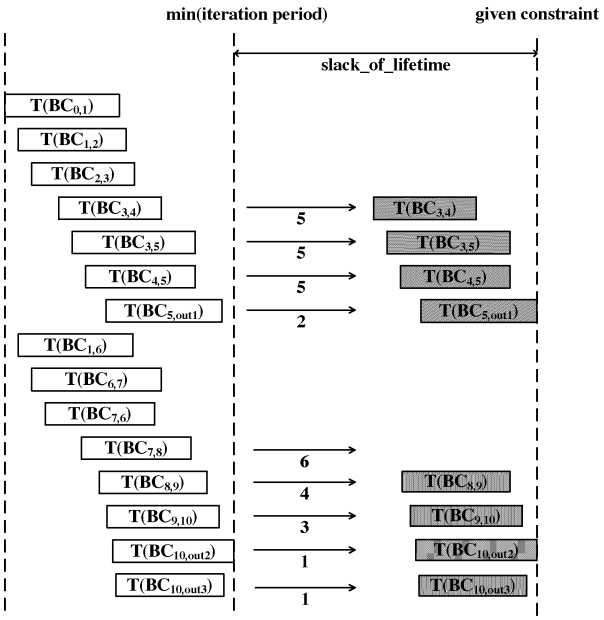


Fig. 9. Maximal separation of the lifetimes shown in Fig. 8.

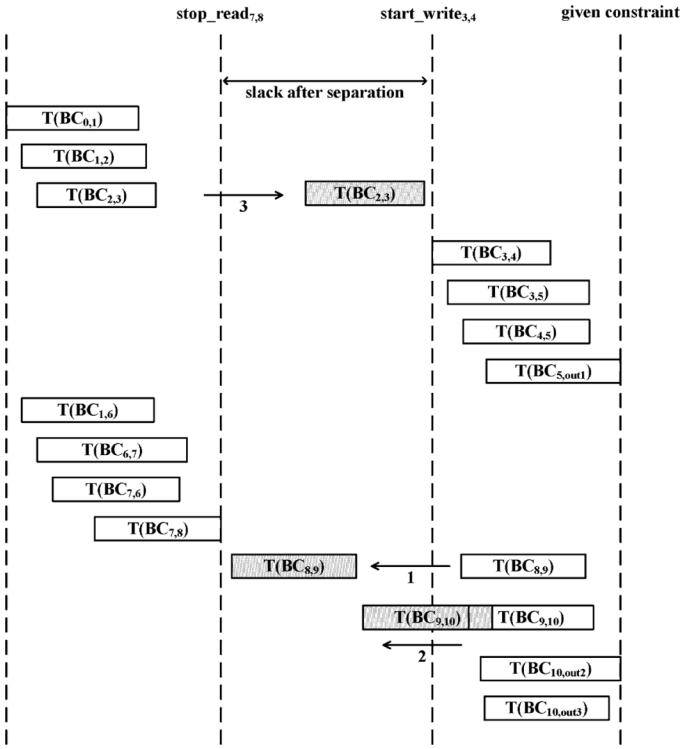


Fig. 10. Lifetime translation when the separation in Fig. 9 is completed.

For the separation of lifetimes, we propose a procedure called the *maximal separation scheme*, which is detailed in Algorithm 1. The input of this algorithm consists of all buffer lifetimes in a given buffer based dataflow; the output is the translated (separated) lifetimes up to a given constraint. When a constraint is given for $\mathrm{iteration\_period\_slack} > 0$, the scheme begins moving lifetimes toward the output (i.e., lifetimes are moved to the right) in order to decrease the number of overlapped lifetimes. If two lifetimes, $T(BC_{i,j})$ and $T(BC_{k,l})$ does not satisfy (11), then $T(BC_{i,j})$ and $T(BC_{k,l})$ are overlapped. The maximal separation scheme stops when the number of overlapped lifetimes is not decreased by the translation of lifetimes. When a buffer lifetime is translated, only nw of the corresponding buffer controller is changed because the amount of varying nw reflects to all start and stop signals as described in (1)–(4). If a buffer controller translates its lifetime to the right, its nw is increased. When a buffer controller translates its lifetime to the left, its nw is decreased.

Fig. 9 shows how the maximal separation scheme is applied to the lifetimes of Fig. 8. The shaded boxes represent the translated lifetimes and the numbers below arrows indicate the order of translations. For the maximal separation between lifetimes of the input and output sides, the amount of each translation except $BC_{5,\mathrm{out1}}$ is equal to $\mathrm{iteration\_period\_slack}$. In case of $BC_{5,\mathrm{out1}}$, its amount of translation is larger than $\mathrm{iteration\_period\_slack}$. Even though original $\mathrm{stop\_read}_{5,\mathrm{out1}} <$ original $\mathrm{stop\_read}_{10,\mathrm{out2}}$, the translated $\mathrm{stop\_read}_{5,\mathrm{out1}}$ does not have to be less than the translated $\mathrm{stop\_read}_{10,\mathrm{out2}}$ because there is neither operational dependency nor the fan-ins/fan-outs offset between $BC_{5,\mathrm{out1}}$ and $BC_{10,\mathrm{out2}}$. Thus, the original $\mathrm{stop\_read}_{5,\mathrm{out1}}$ is moved up to the given constraint. This translation case corresponds to line 4 of Algorithm 1.

The translation starts from the buffer controllers connected to outputs. The next translation begins from the buffer controller having the largest $\mathrm{start\_write}$ because the largest $\mathrm{start\_write}$ represents that the buffer controller has no operational dependency on the other buffer controllers which do not yet translate their lifetimes. This rule determines the order of translations. In the fifth translation of Fig. 9, $BC_{3,4}$, $BC_{3,5}$, and $BC_{4,5}$ simultaneously translate their lifetimes because $BC_{4,5}$ has the fan-in offset with $BC_{3,5}$, and $BC_{3,5}$ has the fan-out offset with $BC_{3,4}$. The separation stops at the sixth translation because the lifetime translation of $BC_{7,8}$ increases the number of overlapped lifetimes.

When the separation process is completed, a new slack is created in the middle of the input and output sides. The range of slack is from the maximum $\mathrm{stop\_read}$ of the input side to the minimum $\mathrm{start\_write}$ of the output side. In order to further decrease the number of overlapped lifetimes, lifetimes are translated within the slack range. The buffer controllers in the input side translate their lifetimes to the right until (upper bound of $\mathrm{slack} - 1$). The right translation updates the upper bound of slack as $\mathrm{start\_write}$ of the translated lifetime. The buffer controllers in the output side translate their lifetimes to left until (lower bound of $\mathrm{slack}+1$). The left translation updates the lower bound of slack as $\mathrm{stop\_read}$ of the translated lifetime.

Fig. 10 illustrates the lifetime translations when the separation process depicted in Fig. 9 is completed. The shaded boxes represent the translated lifetimes and the numbers below arrows denote the ordering of translations. The upper bound of slack is $\mathrm{start\_write}_{3,4}$ of $T(BC_{3,4})$ and the lower bound of slack is $\mathrm{stop\_read}_{7,8}$ of $T(BC_{7,8})$. However, when $T(BC_{7,8})$ is translated, all lifetimes in the input side are also translated in the same amount as $T(BC_{7,8})$ to preserve fan-ins/fan-outs offsets. Thus, the translation does not reduce the number of overlapped lifetimes. In order to translate one lifetime at a time, the right

translation starts from $T(BC_{2,3})$ and the left translation begins from $T(BC_{8,9})$.

B. Activity Time and Bus Sharing

Since a buffer controller, $BC_{i,j}$ has one writing and one reading ports, there are two activities; writing activity, $W_{i,j}$ and reading activity, $R_{i,j}$. In $T(BC_{i,j})$, the writing activity time, $T(W_{i,j})$ is defined as the time from $start\_write_{i,j}$ to $stop\_write_{i,j}$. The reading activity time, $T(R_{i,j})$ is defined as the time from $start_read_{i,j}$ to $stop\_read_{i,j}$. Each writing or reading activity uses a bus for the data transfer between node and a buffer controller.

Bus sharing is done by assigning non-overlapped activity times to the same bus. In order to assign activity times of $BC_{i,j}$ and $BC_{k,l}$ to the same bus, the activity times must satisfy the following condition:

$$\begin{aligned}&\max\{\text{stop}(\text{Act}_{i,j}),\text{stop}(\text{Act}_{k,l})\}\\&\quad-\min\{\text{start}(\text{Act}_{i,j}),\text{start}(\text{Act}_{k,l})\}\\&\qquad> M_{i,j}+M_{k,l},\text{Act}_{i,j}\in\{W_{i,j},R_{i,j}\},Act_{k,l}\\&\qquad\in\{W_{k,l},R_{k,l}\}\end{aligned}\tag{12}$$

where $i\neq k, j\neq l, \text{stop}(\text{Act})$ represents stop of activity time and $\text{start}(\text{Act})$ denotes start of activity time. For example, if $\text{Act}_{i,j}=W_{i,j}, \text{stop}(\text{Act}_{i,j})$ is $\text{stop\_write}_{i,j}$ and $\text{start}(\text{Act}_{i,j})$ is $\text{start\_write}_{i,j}$. Equation (12) represents that two activity times are non-overlapped if the difference between the maximum stop and the minimum start of two activity times is larger than the summation of transferred data sizes of the two activities.

The first step of the proposed bus sharing technique separates all activity times without considering a time constraint. Then, if the iteration period exceeds a given time constraint, activity times are translated to satisfy the condition that the iteration period should be less or equal to the time constraint. When all separation and translation steps end, according to (12), non-overlapped activity times are assigned to the same bus.

We consider two strategies to use buses: one is to separate writing and reading activities, and the other is not to distinguish writing and reading activities. Based on these bus strategies, we propose the method of separating activity times for bus sharing. Basically, the method separates activity times to be non-overlapped with each other. The separation has two steps: one is separating activity times within a single buffer controller, and the other is separating activity times among buffer controllers.

Algorithm 2 Separation of activity times

```
1: //Act_{i,j} ∈ {W_{i,j}, R_{i,j}}, Act_{k,l} ∈ {W_{k,l}, R_{k,l}}
2: SEPARATION(Act_{i,j}, Act_{k,l})
3:
4: if (i == k) and (j == l) then
5:   //Separation of activity times within a single buffer controller.
6:   if (Act_{i,j} ≠ Act_{k,l}) then
7:     nr_{i,j} ← nr_{i,j} + (stop_write_{i,j} − start_read_{i,j}) + 1;
8:   end if
9: else
10:  //Separation of activity times between buffer controllers.
11:  if (max{stop(Act_{i,j}), stop(Act_{k,l})} − min{start(Act_{i,j}), start(Act_{k,l})}) ≤ (M_{i,j} + M_{k,l}) then
12:    if max{stop(Act_{i,j}), stop(Act_{k,l})} == stop(Act_{i,j}) then
13:      if Act_{i,j} == W_{i,j} then
14:      nw_{i,j} ← nw_{i,j} + (stop(Act_{k,l}) − start_write_{i,j}) + 1;
15:      else if Act_{i,j} == R_{i,j} then
16:      nr_{i,j} ← nr_{i,j} + (stop(Act_{k,l}) − start_read_{i,j}) + 1;
17:      end if
18:    else if max{stop(Act_{i,j}), stop(Act_{k,l})} == stop(Act_{k,l}) then
19:      if Act_{k,l} == W_{k,l}
20:      nw_{k,l} ← nw_{k,l} + (stop(Act_{i,j}) − start_write_{k,l}) + 1;
21:      else if Act_{k,l} == R_{k,l} then
22:      nr_{k,l} ← nr_{k,l} + (stop(Act_{i,j}) − start_read_{k,l}) + 1;
23:      end if
24:    end if
25:  end if
26: end if
```

Algorithm 2 provides the details of the procedure to separate activity times. Line 11 represents that two activity times are overlapped if the difference between the maximum stop and the minimum start of two activity times is smaller than the summation of transferred data sizes of the two activities. According to the sequence of two separation steps, the following two separation approaches are possible: 1) the separation within a single buffer controller precedes the separation among buffer controllers and 2) the activity times among buffer controllers are separated prior to the separation of activity times within a buffer controller.

Fig. 11 illustrates these two separation approaches applied to the buffer controllers depicted in Fig. 8 under the assumption that the target platform uses separate buses for writing and reading activities. The shaded boxes represent the overlapped region of writing and reading activity times within buffer lifetimes. The circled numbers denote the order of separation. The arrows represent the internal separation of writing and reading activity times within a buffer lifetime. Case a) represents the case that the separation of activity times among buffer controllers precedes the separation of activity times within a buffer controller. Case b) corresponds to the case that the separation of activity times within a buffer controller is prior to the separation of activity times among buffer controllers.

In Case a) of Fig. 11, the separation process starts from the reading activity of a buffer controller connected to an output because the reading activity does not have the operational dependency on other activities. If there are multiple outputs, the output reading activity having the largest stop_read value is selected for the first separation. In the figure, for the first separation, $T(R_{10,out2})$ is thus translated to the right to be non-overlapped with $T(R_{10,out3})$. As the result, $nr_{10,out2}$ is increased

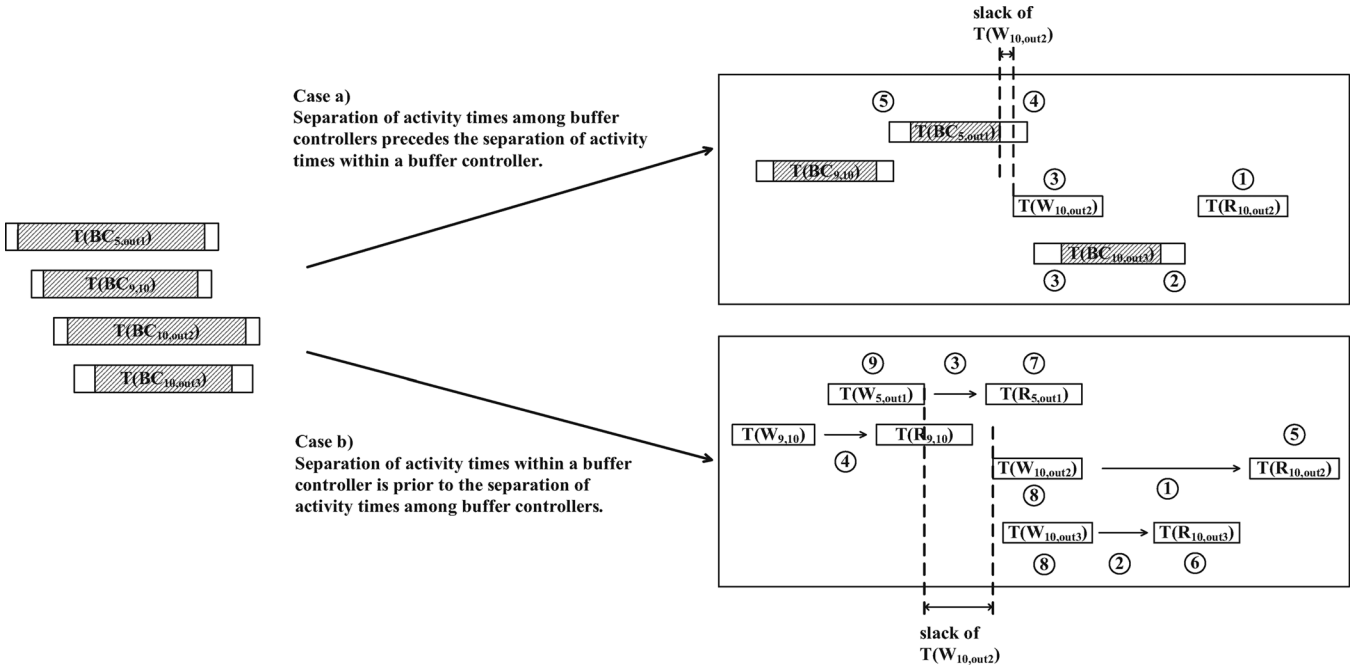


Fig. 11. Two separation approaches applied to the buffer controllers in Fig. 8 when buses are separated for writing and reading activities.

by $stop\_read_{10,out3} - start\_read_{10,out2} + 1$. The next separation step is determined by the *stop* signals of activity times. Since $stop\_read_{10,out3} > stop\_write_{10,out2}$, the sequence of separation is $T(R_{10,out3} \rightarrow T(W_{10,out2}$. When $T(R_{10,out3}$ is translated, $T(R_{10,out2}$ is also translated by the same amount to preserve the previous separation between $T(R_{10,out2})$ and $T(R_{10,out3})$.

When buses separate writing and reading activities, the separation of activity times within a buffer controller does not reduce the number of buses because writing and reading activities within a buffer controller are not assigned to the same bus. Therefore, in Fig. 11, Case b) requires more separation steps than Case a).

According to the separation approaches, the slack size of an activity time varies. The slack of an activity time represents the range within which translating an activity time does not increase the number of overlapped activity times. In Case a) of Fig. 11, the slack size of an activity time is 1 because the separation distance between adjacent non-overlapped activity times is 1. For example, when the separation of $T(W_{10,\mathrm{out2}})$ is completed, $\mathrm{start\_write}_{10,\mathrm{out2}} = \mathrm{stop\_write}_{5,\mathrm{out1}} + 1$. However, in Case b) of Fig. 11, the slack size is affected by the internal overlap between writing and reading activity times within a lifetime. Thus, $\mathrm{start\_write}_{10,\mathrm{out2}} = \mathrm{stop\_write}_{5,\mathrm{out1}} + \mathrm{stop\_write}_{9,10} - \mathrm{start\_read}_{9,10} + 1$ (i.e., the translation amount to separate $T(R_{9,10})$ from $T(W_{9,10})$ is reflected). As the result, Case b) has larger slack of $T(W_{10,\mathrm{out2}})$ than Case a) has as illustrated in Fig. 11.

If $\mathrm{iteration\_period} > \mathrm{given\_constraint}$ when the separation of activity times is completed, a *coming-back* scheme begins to satisfy $\mathrm{iteration\_period} \leq \mathrm{given\_constraint}$ from the activity time having the largest slack size in a descending order. By reducing each slack sizes to 1, $\mathrm{iteration\_period}$ approaches to $\mathrm{given\_constraint}$. If $\mathrm{iteration\_period}$ is still larger than $\mathrm{given\_constraint}$ even after all slack sizes are minimized, the left translation of activity times starts from the output reading activity. Each left translation overlaps activity times as much as possible to reduce $\mathrm{iteration\_period}$. When $\mathrm{iteration\_period} \leq \mathrm{given\_constraint}$, the left translation stops.

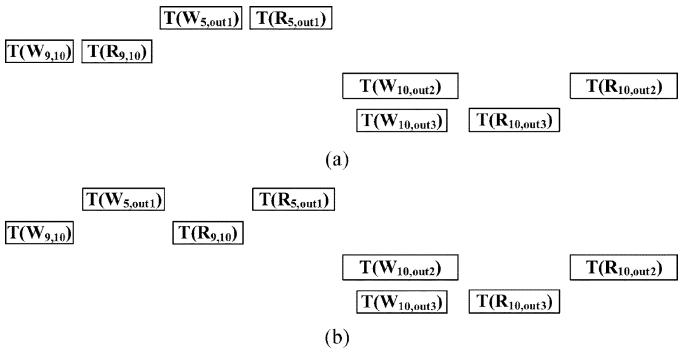


Fig. 12. Two separation approaches when buses are not distinguished for writing and reading activity. (a) Separating activity times among buffer controllers is prior to the separation of activity times within a buffer controller. (b) Separating activity times within a buffer controller precedes the separation of activity times among buffer controllers.

In case buses do not distinguish writing and reading activities, the two separation approaches shown in Fig. 11 further isolate overlapped regions between writing and reading activities. Fig. 12 illustrates such cases. Fig. 12(a) and 12(b) correspond to Case a) and Case b) of Fig. 11, respectively. In both cases, activity times are translated to be non-overlapped with other activity times except the case where activity times have fan-out offsets. Thus, the numbers of non-overlapped activity times are the same in both cases. However, the numbers of *non-overlapped lifetimes* of these two approaches are different due to the different separation sequences used. Compared to

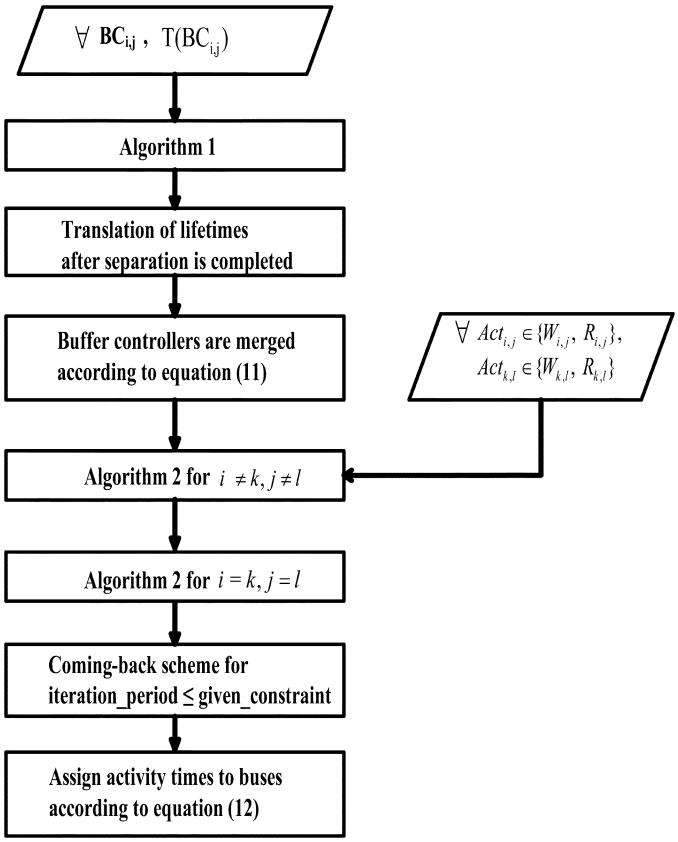


Fig. 13. Overall flow of the proposed sharing methodology.

Fig. 12(a), lifetimes of Fig. 12(b) are more stretched such that the number of overlapped lifetimes of Fig. 12(b) is larger than that of Fig. 12(a). In Fig. 12(a), the number of overlapped lifetimes is 1 (i.e., $BC_{10,\mathrm{out2}}$ and $BC_{10,\mathrm{out3}}$). On the other hand, in Fig. 12(b), the number of overlapped lifetimes is 2 (i.e., $BC_{10,\mathrm{out2}}$ and $BC_{10,\mathrm{out3}}, BC_{5,\mathrm{out1}}$ and $BC_{9,10}$). To this end, our proposed bus sharing separates activity times among buffer controllers prior to the separation of activity times within a buffer controller.

C. Relation Between Activity Time and Lifetime

Since the buffer and bus sharing starts from the minimal lifetimes, lifetimes are stretched only when activity times are separated. When the lifetime of $BC_{i,j}$ is stretched, $\mathrm{MEM}(BC_{i,j})$, (i.e., the memory size of $BC_{i,j}$) is increased because of the following:

$$\mathrm{MEM}(BC_{i,j}) = \min\{M_{i,j}, (\mathrm{start\_read}_{i,j} - \mathrm{start\_write}_{i,j})\}. \tag{13}$$

In (13), when writing and reading activity times are overlapped within the lifetime of $BC_{i,j}$, the buffer memory, $\mathrm{MEM}(BC_{i,j})$ is determined by the difference between $\mathrm{start\_read}_{i,j}$ and $\mathrm{start\_write}_{i,j}$. Otherwise, $BC_{i,j}$ needs the buffer memory to store $M_{i,j}$, the total amount of transferred data.

Fig. 13 shows the overall flow of the proposed sharing methodology. Here the procedure "Algorithm 2 for $i \neq k, j \neq l$" represents the separation of activity times among buffer controllers and the procedure "Algorithm 2 for $i = k, j = l$" means the separation of activity times within a buffer controller. Our sharing methodology puts additional buffers to the original dataflow for parameterizing data transfers. Thus, our approach is to minimize the numbers of buffers and buffer memory as much as possible. When buffer lifetimes are translated, write offset (nw) is translated and does not affect the buffer memory size. However, if activity times are separated within a buffer controller, the buffer memory size is increased by the increment of read offset (nr). The increment of read offset stretches the corresponding lifetime such that the number of overlapped lifetimes can be increased as shown in Fig. 12. Thus, in order to minimize the numbers of buffers and buffer memory, the proposed sharing methodology translates (separates) lifetimes prior to activity times, and the separation of activity times among buffer controllers precedes the separation of activity times within a buffer controller.

Fig. 14 illustrates the relationship between buffer sharing and bus sharing when buses are not separated for writing and reading activities. $BC_{(1,2)(4,5)(9,10)}$ represents the shared buffer controller merging $BC_{1,2}, BC_{4,5}$, and $BC_{9,10}$. BUS1-BUS6 are pre-assigned buses in the target platform used. The lifetimes of $BC_{1,2}, BC_{4,5}$, and $BC_{9,10}$ are non-overlapped, and there are enough slacks for the translation of all activity times to be non-overlapped. In Fig. 14(a), neither buffer sharing nor bus sharing is applied. In this case, each bus has only one activity. When only buffer sharing is applied as shown in Fig. 14(b), the number of buses is two for the writing and reading activities of $BC_{(1,2)(4,5)(9,10)}$. In addition, since only one activity transfers data through a bus at a time, there is *no additional hardware required* in order to multiplex accessing a bus/a shared buffer controller. On the other hand, due to the increased number of activities assigned to each bus, the load capacitance per bus increases. Furthermore, the activated buffer memory size for each activity is determined by the maximum buffer memory size among $BC_{1,2}, BC_{4,5}$, and $BC_{9,10}$. In Fig. 14(c), the activated buffer memory size is the same as in Fig. 14(b), but the load capacitance per bus has increased more than that in Fig. 14(b) because all activities are assigned to one bus. The dynamic energy-consumption amounts of Fig. 14(a)–(c) are

$$\begin{aligned}
&2 \times \{2C_{\mathrm{load}}V^2 f M_{1,2} + 2C_{\mathrm{load}}V^2 f M_{4,5} \\
&\quad + 2C_{\mathrm{load}}V^2 f M_{9,10}\} \\
&\quad + 2 \times \{\mathrm{MEM}(BC_{1,2}) \times T(BC_{1,2}) \\
&\quad + \mathrm{MEM}(BC_{4,5}) \times T(BC_{4,5}) \\
&\quad + \mathrm{MEM}(BC_{9,10}) \times T(BC_{9,10})\} \times \mathrm{mem_cost} \qquad (14) \\
&2 \times \{4C_{\mathrm{load}}V^2 f M_{1,2} + 4C_{\mathrm{load}}V^2 f M_{4,5} \\
&\quad + 4C_{\mathrm{load}}V^2 f M_{9,10}\} \\
&\quad + 2 \times \{\mathrm{MEM}(BC_{(1,2)(4,5)(9,10)}) \times T(BC_{1,2}) \\
&\quad + \mathrm{MEM}(BC_{(1,2)(4,5)(9,10)}) \times T(BC_{4,5}) \\
&\quad + \mathrm{MEM}(BC_{(1,2)(4,5)(9,10)}) \\
&\quad \times T(BC_{9,10})\} \times \mathrm{mem_cost} \qquad (15) \\
&2 \times \{7C_{\mathrm{load}}V^2 f M_{1,2} + 7C_{\mathrm{load}}V^2 f M_{4,5} \\
&\quad + 7C_{\mathrm{load}}V^2 f M_{9,10}\} \\
&\quad + 2 \times \{\mathrm{MEM}(BC_{(1,2)(4,5)(9,10)}) \times T(BC_{1,2}) \\
&\quad + \mathrm{MEM}(BC_{(1,2)(4,5)(9,10)}) \times T(BC_{4,5})
\end{aligned}$$

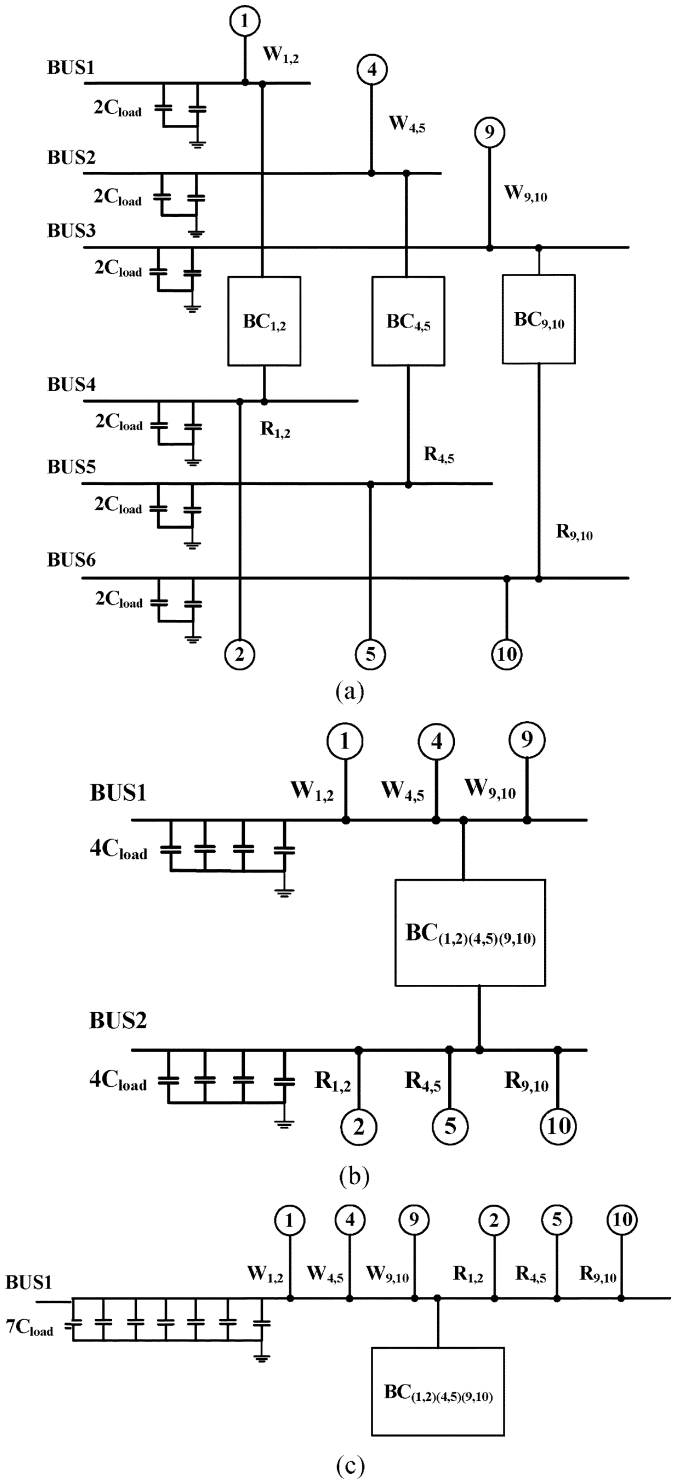


Fig. 14. Buffer and bus sharing when buses are not separated for writing and reading activities. (a) Neither buffer nor bus sharing is applied. (b) Only buffer sharing is applied. (c) Both buffer and bus sharing is applied.

$$+\,\mathrm{MEM}(BC_{(1,2)(4,5)(9,10)}) \times T(BC_{9,10})\} \times \mathrm{mem\_cost} \quad (16)$$

respectively, where C_{load} is the load capacitance of the buses in Fig. 14, f is the operating frequency of the buses, V is the supply voltage, and $\mathrm{mem\_cost}$ is the power consumption per unit memory access. Compared with (14), the load capacitance for each activity is doubled in (15) and is 3.5× higher in (16). In addition, the maximum buffer memory size, $\mathrm{MEM}(BC_{(1,2)(4,5)(9,10)})(=\max\{\mathrm{MEM}(BC_{1,2}), \mathrm{MEM}(BC_{4,5}), \mathrm{MEM}(BC_{9,10})\})$ is activated for accessing each buffer memory in (15) and (16). Therefore, the dynamic energy consumption in (14) $<$ the dynamic energy consumption in (15) $<$ the dynamic energy consumption in (16).

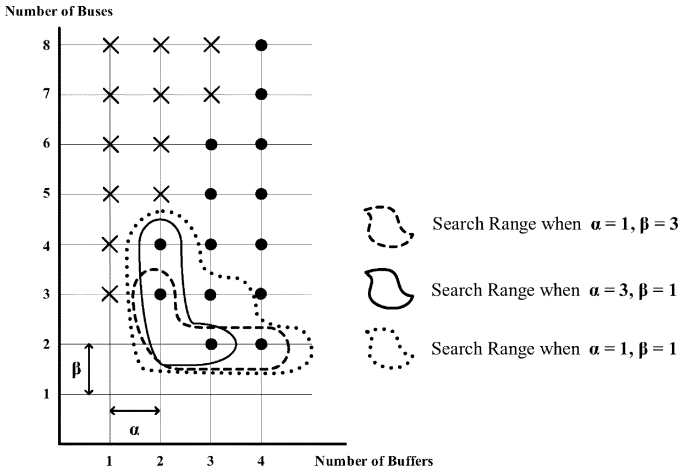


Fig. 15. All possible sharing cases for the buffer-based dataflow that has five buffers and ten buses.

D. Energy Consumption

In order to select the sharing case consuming the minimum energy, all sharing cases must be examined with the energy consumption model that incorporates all the factors such as buffer costs, bus costs, the number of buffers, the number of buses, the number of ports in buses, activity times, buffer lifetime, and the size of activated buffer memory. However, the exhaustive search is a very time-consuming process if there are a great amount of sharing cases. In order to reduce the computational complexity, we need to limit the search range by reducing the number of sharing cases examined.

For the buffer-based dataflow having five buffers and ten buses, Fig. 15 illustrates all possible sharing cases. Each point of the grid represents a possible sharing case with some buffers and buses. A point does not distinguish different sharing cases that have an identical number of buffers and buses. A cross mark in the grid represents that there is no possible sharing case in the given numbers of buffers and buses. The cases having neither a solid circle nor a cross mark in the grid are not considered by buffer and bus sharing. α and β represent the cost of buffer energy consumption and the cost of bus energy consumption, respectively, and are dependent on the target platform used. The solid, dashed, and dotted lines represent the search ranges of finding the sharing case consuming the minimum energy according to α and β (i.e., buffer and bus costs). The boundary condition determining the search range is found by the following equations:

$$E_{\mathrm{est}}(\mathrm{number\ of\ buffers}, \mathrm{number\ of\ buses}) = \{\alpha(\mathrm{number\ of\ buffers}) + \beta(\mathrm{number\ of\ buses})\} \times \mathrm{itr\_period} \quad (17)$$

$$E_{\text{bound}} = \begin{cases} \{\alpha(\text{minimum number of buffers}+1) \\ +\beta(\text{minimum number of buses})\} \\ \times \text{itr\_period}, \quad \text{when } \alpha > \beta, \\ \{\alpha(\text{minimum number of buffers}) \\ +\beta(\text{minimum number of buses}+1)\} \\ \times \text{itr\_period}, \quad \text{when } \alpha < \beta, \\ \{\alpha(\text{minimum number of buffers}+1) \\ +\beta(\text{minimum number of buses}+1)\} \\ \times \text{itr\_period}, \quad \text{when } \alpha = \beta \end{cases} \tag{18}$$

where E_{est} (number of buffers, number of buses) is the estimated energy consumption value corresponding to the point (number of buffers, number of buses) in the grid of Fig. 15, itr_period is the iteration period of a given buffer-based dataflow and E_{bound} is the boundary condition to determine the search range for the minimum energy consumption.

If one resource (i.e., either buffer or bus) consumes more energy than the other, E_{bound} is set to minimize the number of the resource consuming more energy. Thus, the search range is determined between the minimum number and (the minimum number $+1$) of the resource, whichever is more expensive (i.e., the resource consuming more energy). In Fig. 15, when the cost of one resource is more expensive than the other, the search range is biased to the resource having the lower cost. If buffer and bus costs are the same, both buffers and buses have the same impact on determining the search range. In this case, the search range is evenly spread toward both buffer and bus axes in Fig. 15.

When the search range is determined, the sharing case for the minimum energy consumption is found within the search range based on the following energy consumption model:

$$\begin{aligned} E_{\text{total}} &= \alpha \times E_{\text{buffer}} + \beta \times E_{\text{BUS}} \\ &= \alpha \times \{(\#\text{of BCs} \times \text{itr\_period}) \\ &\quad + \sum_{n=1}^{\#\text{of BCs}} (T(BC_n) \times \text{MEM}(BC_n))\} \\ &\quad + \beta \times \left\{ \sum_{n=1}^{\#\text{of buses}} \left\{ (\#\text{of ports in} \text{BUS}_n) \right.\right. \\ &\quad \left.\left. \times \sum_{m=1}^{\#\text{of act in} \text{BUS}_n} T(\text{Act}_m) \right\}\right\} \end{aligned} \tag{19}$$

where E_{total} is the total energy consumption, E_{buffer} is the energy consumption by buffers, E_{BUS} is the energy consumption by buses, (# of act in BUS_n) denotes the number of activities in BUS_n and $T(\text{Act}_m)$ is the activity time assigned to BUS_n. (#of BCs $\times$ itr_period) represents the leakage energy consumption of buffer controllers. The other terms indicate the dynamic energy consumption by buffer memory access and data transfers through buses. Since the iteration period, the number of BCs and the size of activity time come from the buffer-based dataflow to which buffer and bus sharing is not applied, they are not determined by the sharing methodology. All sharing cases within a search range are evaluated with E_{total} to find the sharing case consuming the minimum energy.

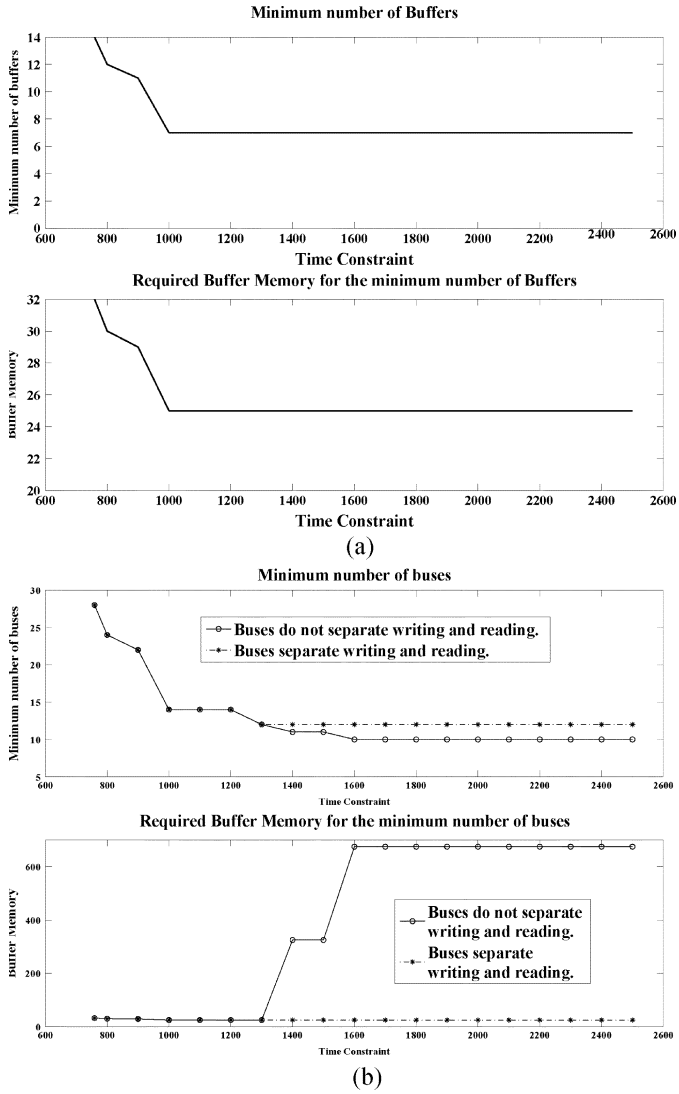


Fig. 16. Simulation results when the proposed sharing methodology is applied to the buffer-based dataflow in Fig. 8. (a) Minimum number of buffers and required buffer memory. (b) Minimum number of buses and required buffer memory.

V. Evaluation of the Proposed Methodology

In this section, we evaluate the methodology proposed in Section IV using the buffer-based dataflow shown in Fig. 8. The reason why we use the buffer-based dataflow in Fig. 8 is that it reflects various aspects of dataflow graphs such as a feed-forward path, a feed-forward loop, multiple fan-ins, multiple fan-outs, and a feedback loop. The proposed methodology was implemented in C.

A. Evaluation of Buffer Sharing and Bus Sharing

Fig. 16 shows the results of buffer and bus sharing. In all plots of Fig. 16, the x-axis represents the given time constraint. The top plots of Fig. 16(a) and (b) show the minimum number of buffers and buses in each time constraint, respectively. In Fig. 16(a) and (b), the bottom plots show the amount of buffer memory when the minimum number of buffers/buses is used in each time constraint.

In Fig. 16(a), for the time constraint greater than 1000, the minimum number of buffers becomes seven because at least

TABLE II
NUMBER OF BUSES WHEN THE PROPOSED SHARING METHODOLOGY IS APPLIED TO THE APPLICATIONS IN [9], [18], [19]

	# of buses (original)	# of buses (sharing)	Buffer memory	Constraint	Reduction ratio
SIRF [9]	9	5	308 bytes	4Mbps	44.4 %
IPv4 [18]	13	8	136 bytes	33M Packets/s	38.5 %
TX MC CDMA [19]	9	4	8 bytes	20.8μs /Frame	55.6 %
RX MC CDMA [19]	19	8	20 bytes	20.8μs /Frame	57.89 %

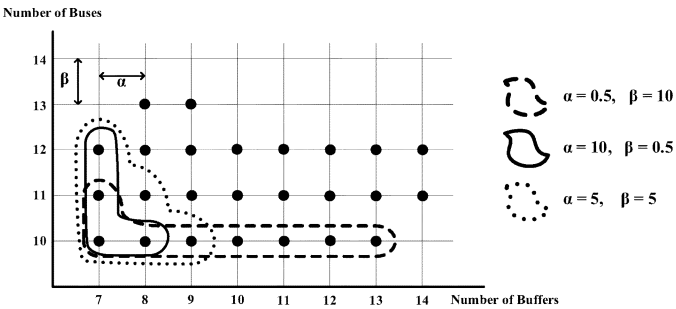


Fig. 17. Search ranges for the minimum energy consumption.

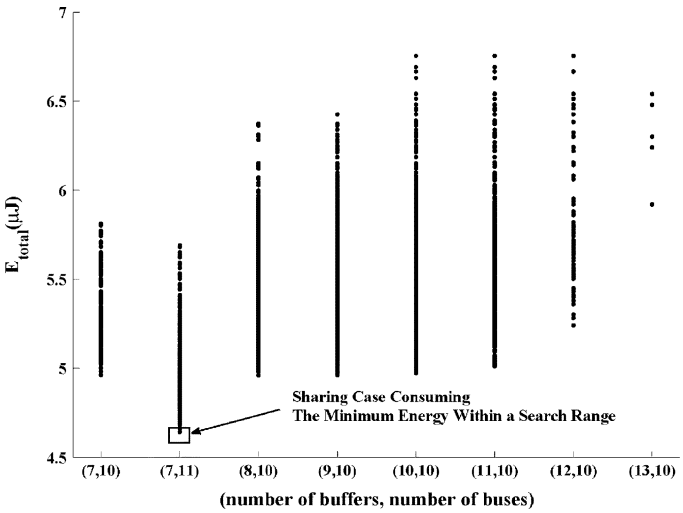
seven buffer controllers are required to preserve the fan-ins/fan-outs offsets and the feedback loop.

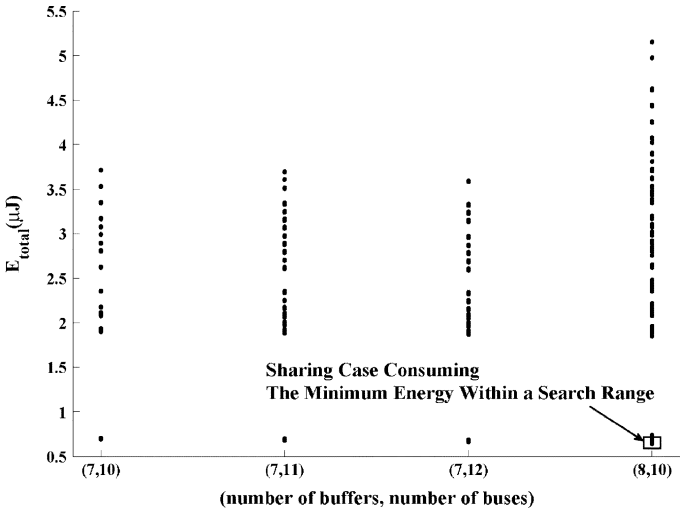
In Fig. 16(b), when a target platform does not separate buses for writing and reading, the buffer memory size increases for the time constraint greater than 1300 because the separation of activity times within a buffer controller starts there. However, when writing and reading buses are separated in the target platform used, the buffer memory size does not increase because there is no separation of activity times within a buffer controller.

Fig. 1(a) in Section II is the original dataflow (without buffer controllers) of Fig. 8. In order to realize the dataflow of Fig. 1(a), the target platform used utilizes 15 buses. When our proposed sharing methodology is applied to the buffer-based dataflow of Fig. 8, which is converted from the dataflow of Fig. 1(a), the total number of buses becomes smaller than 15 if the time constraint is greater than or equal to 1000.

We also apply the proposed sharing methodology to data-centric applications such as Sample Importance Resample Filter (SIRF) [9], IPv4 forwarding [18], multi-carrier code division multiple access (MC-CDMA) transmitter and receiver [19]. Table II summarizes the results we obtained.

In Table II, the second column shows the number of buses in the original dataflow. When each node in the original dataflow is implemented as an individual hardware, the number of edges is the same as the number of buses in the original dataflow. The third column is the number of buses when the proposed sharing methodology is applied to the original dataflow. Since our sharing methodology uses the buffer-based dataflow, which is constructed from its original dataflow, additional buffer memory is required as shown in the fourth column of the table. The fifth column is the constraint which comes from the application specification, and the sixth column indicates the





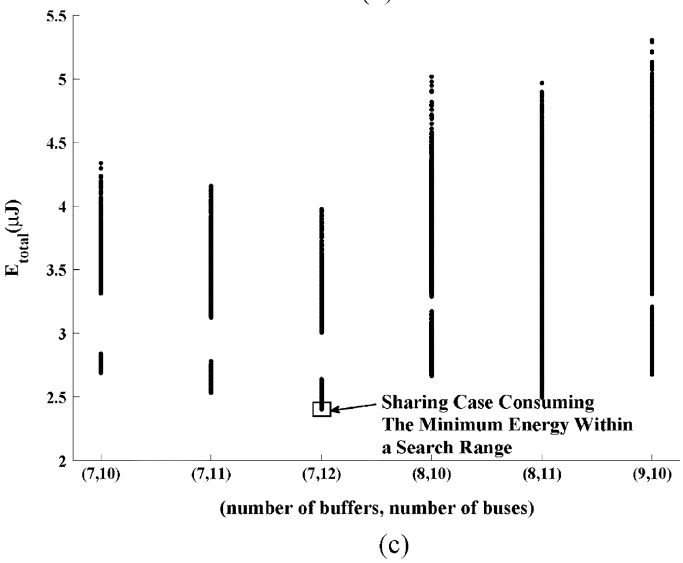


Fig. 18. Simulation results of finding the sharing case consuming the minimum energy within the search ranges determined by buffer and bus costs: (a) E_{total} when $\alpha = 0.5$ mW/buffer and $\beta = 10$ mW/bus; (b) E_{total} when $\alpha = 10$ mW/buffer and $\beta = 0.5$ mW/bus; (c) E_{total} when $\alpha = 5$ mW/buffer and $\beta = 5$ mW/bus.

bus reduction ratio. As shown in Table II, our proposed sharing methodology can reduce the number of buses with additional

buffer memory for controlling data transfers. Especially, in the TX and RX MC CDMA applications [19], since the input and output data transfer times of each node are separated (non-overlapped), the lifetimes of adjacent buffer controllers are non-overlapped. Hence, the proposed sharing methodology can achieve a higher reduction ratio by adding a relatively small amount of buffer memory.

B. Energy Consumption

Fig. 17 shows the search ranges according to buffer and bus costs when buffer and bus sharings are applied to the buffer-based dataflow of Fig. 8 with the time constraint of 1900 cycles (1 cycle = 10 ns). In the figure, α, the unit of buffer cost, is mW/buffer and β, the unit of bus cost, is mW/bus. The regions is mW/buffer and β, the unit of bus cost, is mW/bus. The regions enclosed with colored lines represent the search ranges according to α and β. The dashed line denotes the search range when $\alpha = 0.5$ and $\beta = 10$. The solid line represents the search range when $\alpha = 10$ and $\beta = 0.5$. The region within dotted line corresponds to the search range when $\alpha = 5$ and $\beta = 5$. Each point in the grid represents the sharing cases having the same numbers of buffers and buses.

Fig. 18 shows the simulation results of finding the sharing case consuming the minimum energy within each search range of Fig. 17. In Fig. 18, the pair values (number of buffers, number of buses) in the x-axis correspond to the points within each search range in Fig. 17. The pair values of Fig. 18(a)–(c) correspond to the points within the dashed, solid, and dotted lines of Fig. 18, respectively. Since one point in Fig. 18 represents the sharing cases having the same numbers of buffers and buses, the E_{total} values in one point vary because of the different sharing combination with the same numbers of buffers and buses. As shown in Fig. 18, depending on the values of α and β, the sharing case having the minimum E_{total} (i.e., the sharing case consuming the minimum energy) may appear at different points in Fig. 17. Furthermore, no minimum E_{total} appears at the sharing case having the minimum resources [i.e., the point (7, 10)].

VI. Conclusion

In this paper, we have presented a sharing methodology to reduce the interconnect resource in a data-centric applications represented as a buffer-based dataflow. The proposed sharing methodology rearranges the lifetimes and activity times of buffers to increase the possibility of buffer and bus sharing. However, since buffer and bus sharing increases the dynamic energy consumption, we do not guarantee that the sharing case with the minimum resource consumes the minimum energy. Thus, we establish an energy consumption model with the estimated buffer and bus costs. The proposed sharing methodology was evaluated with data-centric applications such as SIRF, IPv4, MC-CDMA transmitter and receiver. When the input and output data transfer times of each node are separated, we could achieve a higher interconnect resource reduction ratio. We also confirmed that the sharing case having the minimum resource may not correspond to the sharing case consuming the minimum energy.

References

[1] K. Kundert, "Design of mixed-signal systems on a chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1561–1571, Dec. 2000.

[2] McCorquodale, M. S. Gebara, F. H. Kraver, K. L. Marsman, E. D. Senger, and R. M. Brown, "A top-down microsystems design methodology and associated challenges," in *Proc. Des., Autom. Test Eur. Conf. Exhibition*, 2003, pp. 292–296.

[3] B. Bhattacharya and S. S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Trans. Signal Process.*, vol. 49, no. 10, pp. 2408–2421, Oct. 2001.

[4] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 7, no. 4, pp. 643–663, Oct. 2002.

[5] D. Chen, J. Cong, and Y. Fan, "Low-power high-level synthesis for FPGA architectures," in *Proc. Int. Symp. Low Power Electron. Des.*, Aug. 2003, pp. 134–139.

[6] J. Cong, Y. Fan, and J. Xu, "Simultaneous resource binding and interconnection optimization based on a distributed register-file microarchitecture," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 3, pp. 1–31, May 2009.

[7] J. Mun, S. Han Cho, and S. Hong, "Flexible controller design and its application for concurrent execution of buffer centric dataflows," *J. VLSI Signal Process.*, vol. 47, no. 3, pp. 233–257, Jun. 2007.

[8] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.

[9] S. Hong, J. Lee, A. Athalye, P. M. Djuric, and W. Cho, "Design methodology for domain specific parameterizable particle filter realizations," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 9, pp. 1987–2000, Sep. 2007.

[10] P. Briggs, K. Cooper, and L. Torczon, "Improvements to graph coloring register allocation," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 3, pp. 428–455, May 1994.

[11] P. K. Murthy and S. S. Bhattacharyya, "Buffer merging—A powerful technique for reducing memory requirements of synchronous dataflow specifications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 2, pp. 212–237, Apr. 2004.

[12] H. Jung, H. Yang, and S. Ha, "Optimized RTL code generation from coarse-grain dataflow specification for fast HW/SW cosynthesis," *J. Signal Process. Syst.*, vol. 52, no. 1, pp. 13–34, Jul. 2008.

[13] F. Berthelot, F. Nouvel, and D. Houzet, "A flexible system level design methodology targeting run-time reconfigurable FPGAs," *EURASIP J. Embed. Syst.*, vol. 2008, pp. 1–18, Jan. 2008.

[14] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect-power dissipation in a microprocessor," in *Proc. Int. Workshop Syst. Level Interconnect Prediction*, 2004, pp. 7–13.

[15] N. Chabini and W. Wolf, "An approach for reducing dynamic power consumption in synchronous sequential digital designs," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2004, pp. 198–204.

[16] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex™-II FPGA family," in *Proc. ACM/SIGDA 10th Int. Symp. Field-Program. Gate Arrays*, 2002, pp. 157–164.

[17] R. Lauwereins, M. Engels, M. Ade, and J. A. Peperstraete, "Grape-II: A system-level prototyping environment for DSP applications," *IEEE Computers*, vol. 28, no. 2, pp. 35–43, Feb. 1995.

[18] N. R. Satish, K. Ravindran, and K. Keutzer, "Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors," Presentation Slides on Embedded Systems Week, Oct. 2008.

[19] J. Delorme, "An automatic design flow for mapping application onto a 2D mesh NoC architecture," *Integr. Circuit Syst. Des. Power Timing Model., Opt. Simulation*, vol. 4644, pp. 31–42, 2007, LNCS.

Woohyung Chun received the B.S. and M.S. degrees in electronic and electrical engineering from Hongik University, Seoul, Korea, in 1997 and 1999, respectively. He is currently pursuing the Ph.D. degree from Stony Brook University-SUNY, Stony Brook, NY.

He was with System LSI Division, Samsung Electronics, Korea, from 2002 to 2004, where he developed embedded system software for Samsung SoCs. He was also a Software Engineer with Corecess and Comtec Systems. His research interests include reconfigurable architecture for DSP and wireless communications, hardware-software co-design, and low power VLSI design.

Sungroh Yoon (S'99-M'06) received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea, in 1996 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 2002 and 2006, respectively.

He is currently an Assistant Professor with the School of Electrical Engineering, Korea University, Seoul, Korea. From 2006 to 2007, he was with Intel Corporation, Santa Clara, CA, where he participated in developing Atom and Core i7 microprocessors. Previously he held research positions with Stanford University and Synopsys Inc., Mountain View, CA. His research interests include computer architecture, system-level low-power design, flash memory applications, and biocomputation.

Sangjin Hong (SM'05) received the B.S. and M.S. degrees from the University of California, Berkeley, and the Ph.D. degree from the University of Michigan, Ann Arbor, both in electrical engineering and computer science.

He is currently with the Department of Electrical and Computer Engineering, Stony Brook University—State University of New York, Stony Brook, NY. Before joining Stony Brook University, he was with the Computer Systems Division, Ford Aerospace Corporation as a Systems Engineer. He was also with Samsung Electronics, Korea as a Technical Consultant. His current research interests include the areas of low power VLSI design of multimedia wireless communications and digital signal processing systems, reconfigurable SoC design and optimization, VLSI signal processing, and low-complexity digital circuits.

Prof. Hong served on numerous Technical Program Committees for IEEE conferences.