Synthesis of Area-Efficient and High-Throughput Rate Data Format Converters[1]

Jongwoo Bae and Viktor K. Prasanna
Department of Electrical Engineering-Systems,
EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
{jongwoo, prasanna}@halcyon.usc.edu
URL: http://ceng.usc.edu/~prasanna

Abstract

We propose two design methodologies for synthesis of area-efficient Data Format Converters (DFCs) with high throughput rate. DFCs are grouped into various classes according to the specification of design parameters. The first design methodology is suitable for design of many representative classes of DFCs. The designs using this methodology are based on a two-dimensional *architecture. They have maximum throughput rate and are area-efficient. Various design examples are shown to demonstrate improved performance, flexibility and usefulness of this design methodology. For several representative problems, the area requirements of our designs are compared against those obtained by earlier design methodologies. For all the problems considered, this methodology leads to compact designs. The second design methodology employs an architecture using dual buffers. The simple and regular architecture using dual buffers leads to area-efficient DFCs. The design procedure using this methodology is simple and can reduce the design effort in many applications.*

1: Introduction

Data format converter (DFC) is a hardware interface module employed to reorganize the format of the transferred data between various processing modules with different I/O format requirements [1, 2, 3]. Data format conversion is required in many signal/image processing applications such as two-dimensional Discrete Fourier Transform [4], low-level vision applications, and video coding/decoding (codec) [5, 6], video pre-/post-processing, image scaling etc.. For example, consider the design of a VLSI system for an application based on a video compression standard such as H.263 [7, 8, 9]. The system consists of two parts, the video processing part and the video compression part. In the video processing part, the format of the image grabbed from a video camera and the format of the image required in the video compression system are different. For pre-processing and post-processing of video, data format conversion is required during image scaling. Format conversion to and from the standard image formats such as CIF and QCIF is essentially a data format conversion problem. In the video compression part, there are various processing modules, and data format conversion is required within the modules as well as between successive modules. Specifically, data format conversion is required during the computation of two-dimensional Discrete Cosine Transform, and between the Quantizer and the Variable Length Coding module. Data format conversion is also required to perform different modes of the motion estimation module such as half-pixel computation and multi-resolution search (when a fast search algorithm is used) [7, 8].

The design of data format converters is an important problem in the overall system design. In High Level Synthesis, it is an essential design step, and it is desirable to automate this step. As far as we know, no formal framework has been proposed for the design of data format converters in High Level Synthesis. A methodology is needed for systematic design of various data format converters.

There are two possible solutions for performing data format conversion: (1) conventional software method employing memory buffers and (2) use of dedicated hardware (such as DFC). The software approach employs a buffer which is shared by the processing modules. This is appropriate for designing a general purpose system, and also permits independent operation of processing modules. However, in designing very large special-purpose real-time systems, shared read/write buffers result in low throughput and also require more memory compared with using a dedicated hardware. In many applications, dedicated hardware is preferred.

Various algorithms for synthesis of DFCs have been studied [2, 3, 10]. In [2], the number of buffers needed for DFCs interfacing systolic arrays is studied. In that paper, a framework for the synthesis of DFCs to generate outputs with

---

[1] This research was supported in part by NSF under grant CCR-9317301 and in part by (Defense) Advanced Research Projects Agency under contract F-49620-89-C-0126.

the desired clock skews between systolic arrays is shown. However, the DFC problems defined in [2] are different from ours. Recently, a forward-backward register allocation scheme has been proposed in [3]. The DFC problems are defined and classified according to the I/O patterns. Various techniques and notations are also introduced including the register allocation scheme and the register allocation table. This design methodology employs a pipeline-like one-dimensional architecture. The feedback connections between the registers enable the registers to be reused when they are idle. The number of registers employed is shown to be minimum among all one-dimensional designs. However, one-dimensional DFCs are limited to applications with small input size and they result in slow execution speed because the amount of data movement is limited by the bandwidth of serial connections.

In this paper, two design methodologies for synthesis of DFCs are shown. First, the format conversion problems are grouped into various classes according to the specification of the design parameters. To evaluate the performance of the designs, area and time are considered. The factors that contribute to the area of DFCs are the number of registers, the number of multiplexors and the number of connection wires, and the size of the control circuit. The time performance is measured by throughput.

Our design scheme is based on a two-dimensional architecture, in which the registers are placed along multiple data paths to form a two-dimensional array of registers. This architecture leads to area-efficiency and higher throughput compared with earlier design schemes. Area-efficiency is obtained by fewer number of registers employed in this scheme, and by fewer control states and control lines. Higher throughput is achieved by parallel data movement on the two-dimensional architecture. Also, our proposed design scheme can be used to develop interface modules to perform certain useful format conversion operations not supported by earlier dedicated hardware designs.

We also propose another design methodology using dual buffers. The dual buffer DFCs require more registers than the two-dimensional DFCs. However, the dual buffer DFCs employ a simple and regular architecture to avoid complex wire connections between registers. This leads to designs with area-efficiency and high throughput rate. Another advantage of dual buffer DFCs is simplicity of the design procedure. They are also easy to modify and expand as the design parameters change. It is particularly useful in rapid prototyping.

The rest of this paper is organized as follows. In Section 2, the structure and function of DFCs and a classification of DFCs are discussed. Also, various performance measures are discussed. Our methodology for synthesizing two-dimensional DFCs is shown in Section 3. A lower-bound on the minimum number of registers needed in this approach is also shown in Section 3. Five representative design examples are shown in Section 4. In addition, the circuit layouts of some DFCs designed by our scheme are compared with those produced by earlier techniques with respect to area, the number of registers needed and the number of control states. In Section 5, a design methodology for dual buffer DFCs is proposed. Also, the circuit layouts of the resulting dual buffer DFCs are compared with those of two-dimensional DFCs. Concluding remarks are made in Section 6.

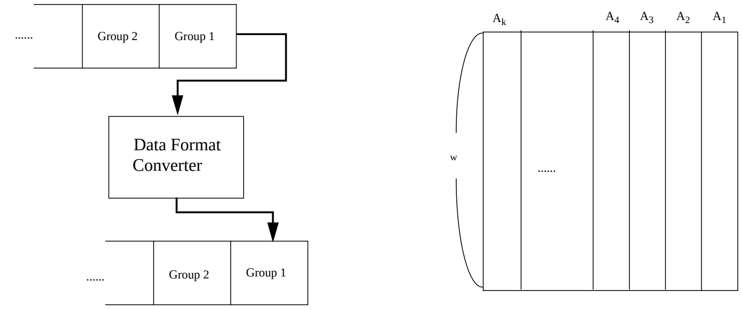
2: Data Format Converters

DFC is a special kind of permutation architecture used in signal/image processing applications in order to reorganize the transfered data between Processing Modules (PMs) to suit the input and output format requirements. DFC consists of the following: registers, connection wires, multiplexors, and a control circuit. Input data is stored in registers, relocated through connection wires and multiplexors; and then output at the desired time. The hardware including the number of registers to be used, the placement of registers and multiplexors, the connections and the controller are specified by the design methodology.

The DFC problems and the related terminology are introduced in [3]. For the sake of completeness, we formally define the problem and introduce important design parameters that were not included in the earlier classification of DFCs. We begin with a classification of the problems.

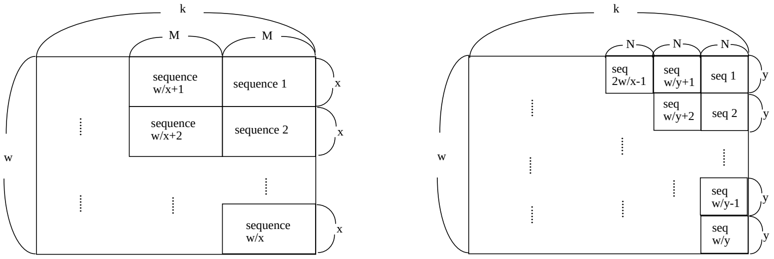
2.1: A Class of DFCs

Input/output streams to the DFC are partitioned into *Groups*. A group is a minimum size of input data, such that after processing an input group of data, the DFC goes back to its initial state to process the next group. Thus, the same operations are repeated on consecutive groups as shown in Figure 1-(a). A group consists of arrays



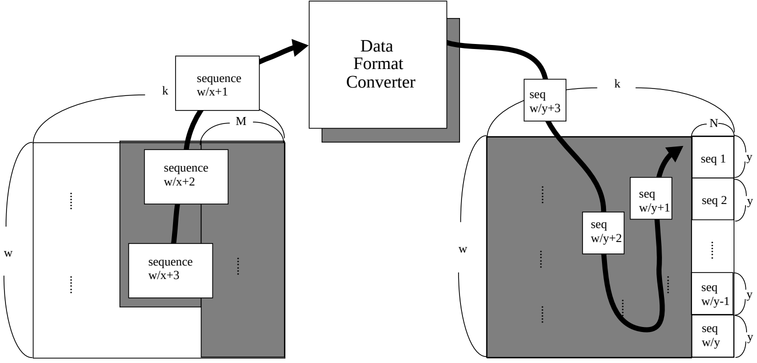
(a) Data Format Conversion over Groups of Input Streams

(b) A Group with k arrays



(c) Input Group partitioned into input sequences

(d) Output Group partitioned into output sequences



(e) Overall Operation between Input/Output Groups

Figure 1. Operation of Data Format Converter

as shown in Figure 1-(b). Assume that each array is w bits long. According to the input/output specification of the DFC, a group is partitioned into input and output sequences.

An input (output) sequence is a set of data to be input (output) during a clock cycle. Assume that the input sequence consists of M arrays with x elements from each array and that the output sequence consists of N arrays with y elements each (See Figure 1-(c) and (d)). The first M arrays are input during the first w/x clock cycles, and then the next M arrays are input during the next w/x clock cycles. This is repeated until the entire group is processed.

The DFC stores the input sequences in its registers, and constructs the output sequence from them. If all the data for an output sequence has not arrived, the rest of the data should wait in the registers. In Figure 1, it is assumed that $M > N$ and $x > y$ for the sake of illustration. Figure 1-(e) denotes the relationship between the I/O sequences. The order of I/O sequences are only shown here. The delays between the I/O sequences are not specified. We will classify the DFCs by the way we specify the delays between the I/O sequences.

The *total delay* d_t is defined as the elapsed time between the first input sequence of a group and the first input sequence of the next group. We define *input throughput* (or *input throughput* *rate)* T_i as the ratio of the number of data in a group ($= kw$ in Figure 1) to d_t. Similarly, the *output throughput rate* T_o is defined. Note that, to obtain a feasible design, $T_i = T_o$.

We consider various input parameters in designing DFCs. These parameters affect the resulting designs with respect to the number of registers needed, the delay between the I/O sequences, the number of control states, the number of control signals and the throughput. We consider three parameters; the format of I/O sequences (s), the delay between consecutive sequences in a group (d), and the delay between the last sequence of a group and the first sequence of the next group (g). g is also called *group delay*. In general, the DFC is denoted *DFC(s,d,g)*. The parameters can have the following values:

- **s :** $S_i/S_o/w$ is used when the format of I/O sequences is specified. S_i can be denoted by M_x and S_o can be denoted by N_y, where the input (output) sequence consists of M (N) arrays with x (y) elements for each array. w denotes the length of an array in a group.
- **d :** D_i/D_o is used when the delay between consecutive input sequences and delay between output sequences in a group are specified.
- **g :** GD_i/GD_o is used when the group delays are specified. $\hat{GD} = \hat{D}$ is used when $GD_i = D_i$ and $GD_o = D_o$.

We can classify DFCs into various classes according to the specification of the parameters. In the following, we have chosen those that are useful in practice.

1. $DFC(S_i/S_o/w, \_, \_)$ **:** In this problem space, only s is specified. The delays d and g are determined during the design procedure.
2. $DFC(S_i/S_o/w, D_i/D_o, \_)$ **:** In this problem space, s and d are specified. We can vary g such that $T_i = T_o$.
3. $DFC(S_i/S_o/w, \_, \hat{GD} = \hat{D})$ **:** In this problem space, delays d and g are not specified; however, $GD_i = D_i$ and $GD_o = D_o$.
4. $DFC(S_i/S_o/w, D_i/D_o, GD_i/GD_o)$ **:** In this problem space, all parameters are specified. If $T_i = T_o$, we can design this class of DFCs by the methodology discussed in this paper. If $T_i \neq T_o$, the design is not feasible.

Clearly, many variations are possible and we have introduced some representative classes of DFCs. In this paper, we mainly focus on the design of the DFCs enumerated above. In general, the design of DFCs becomes harder as the number of specified parameters increases.

2.2: Performance Measures

Most earlier design methodologies considered designs with minimum number of registers [3]. However, we believe that many other factors must be considered to evaluate the designs. Area and time are generally considered to be the most important factors in evaluating VLSI designs. The area of a DFC is determined by the following factors; the number of registers, the number of multiplexors, and the interconnection structure among the registers, multiplexors and the controller. Even though the number of registers, the number of multiplexors and the number of connection wires affect the area of the design, the total area of the circuit layout depends on the interconnection structure.

The controller also affects the overall area of the design. Controllers having small number of control states and few control signals in each state are desirable. It is also desirable to design a controller such that its area is not influenced by the size of M, x, N, y or w.

The time performance can be measured in terms of throughput. The throughput of a DFC is a significant factor since the throughput of the PMs is limited by the throughput of the DFC connecting them.

2.3: Previous Work

Various design schemes have been developed for the synthesis of DFCs. In [2], a framework for synthesis of DFCs performing skew operations between systolic arrays is shown. The skew distributions of the I/O groups are defined using two directional vectors. A sequence of transformations is obtained from these directional vectors of I/O groups. Then, the DFC is obtained by mapping those transformations into hardware. In [2], the number of registers needed for DFCs interfacing systolic arrays is also studied. However, the design methodology does not consider the minimization of number of registers needed as well as maximizing the throughput.

Recently, an algorithm for synthesis of DFCs has been proposed using forward backward register allocation scheme [3]. In this scheme, the registers are connected serially like in a linear array with some additional feedback connections. Clearly, the throughput of the DFC is limited by the bandwidth of the serial data movements over these connections. In [3], a life time analysis is performed to calculate the minimum number of registers needed. The life time is defined as the time the data stays in the DFC before it is sent out [3]. This analysis assumes that the life times of data are available. However, the life time is not solely determined by the I/O specification of the DFC and can change depending on the design methodology. This methodology has been shown to lead to optimal designs in terms of throughput rate and number of registers employed under *one-dimensional* environment. However, new design methodologies are required to design DFCs having high throughput and area-efficiency for applications with parallel I/O. In this paper, we show two design methodologies to obtain area-efficient DFCs with high throughput.

3: Design of Data Format Converters

The methodology for the design of DFCs specifies as output an architecture and an algorithm describing the behavior of the architecture. This algorithm has been called register allocation scheme [3]. The main steps in our methodology are: calculation of the minimum number of registers needed, specification of the register allocation scheme, specification of the hardware connections between the registers and the design of the controller. In the following, we illustrate the two-dimensional DFC by designing an example DFC. Figure 2 shows the design for $DFC(2_2/4_1/4, 1/1, 1/1)$.

Given the I/O buffers and the registers, the register allocation scheme specifies the allocation of data to the I/O buffers and registers as a function of time such that the operations of the DFC are accomplished. In our design, parallel shifting and compaction are performed to make room for input data. We specify a heuristic for performing compaction. The register allocation table is used to represent the register allocation scheme. The register allocation table shows the snapshots of the contents of the input buffer, the registers, and the output buffer as a function of time [1, 11]. Table segment i is defined as the segment of the register allocation table corresponding to clock period i. The register allocation table for the example design is shown in Figure 2-(d). The architecture and the control logic are obtained directly from the register allocation table. The above example illustrates a two-dimensional architecture for the synthesis of DFC.

A *two-dimensional* architecture for DFC has multiple data paths. A data path is a vertical pipeline of registers. Registers are connected along the data path row-by-row for parallel shifting (See Figure 2-(b)). The output pins of the input buffer are connected to the input pins of the multiplexors placed under the first row of registers. The input pins of the multiplexors placed under the output buffer are connected to the output lines of the registers having data to be output. Some additional connection links between registers on different data paths are employed for compaction. Compaction refers to gathering of idle registers into the same rows so as to make more room for parallel shifting. Idle registers are defined as registers having no data or registers available for reuse after their data has been moved to other registers or the output buffer. Idle buffers or idle rows of registers can be defined similarly. Such an architecture is also suitable to perform parallel I/O operations.

3.1: A Lower-Bound on the Number of Registers

The number of registers needed can be minimized for the design of two-dimensional DFCs. The minimum number of registers needed is calculated in the following. First, a special property of DFCs is introduced in Theorem 1 before the calculation of the lower-bound. The lower-bound on the number of array-level registers for $DFC(M_w/N_w/w,\_,\_)$ is given in Theorem 2. The lower-bound on the number of registers for $DFC(M_x/N_y/w,\_,\_)$ is shown in Theorem 3. The number of registers in the I/O buffers are not considered in these analyses.

Theorem 1 *The minimum number of registers needed for $DFC(M_x/N_y/w,\_,\_)$ is same as that needed by $DFC(N_y/M_x/w,\_,\_)$.*

Proof: Assume that a design for $DFC(M_x/N_y/w,\_,\_)$ is given. A design for $DFC(N_y/M_x/w,\_,\_)$ can be obtained by reversing the design process of $DFC(M_x/N_y/w,\_,\_)$ as follows. Figure 2-(d) is used to illustrate this procedure. (1) Assume that the input (output) buffer of $DFC(M_x/N_y/w,\_,\_)$ as an output (input) buffer of $DFC(N_y/M_x/w,\_,\_)$. (2) Assume that the direction of the parallel shifting and compaction are reversed. (3) The new input starts from the last table segment at

Figure 2. An Example Design of $DFC(2_2/4_1/4, 1/1, 1/1)$

$t = 6$ in Figure 2-(d) and the design process goes back to the previous table segments until the segment at $t = 1$ is reached. (4) The desired output is obtained from the new output buffer of $DFC(N_y/M_x/w, \_, \_)$. Now, we have showed that the register allocation table of $DFC(M_x/N_y/w, \_, \_)$ can be reversed to obtain a design for $DFC(N_y/M_x/w, \_, \_)$. Therefore, the numbers of registers used in the two DFCs are the same. □

In the following theorems, we will show a lower-bound on the number of registers for $DFC(M_x/N_y/w, \_, \_)$ only for $M \geq N$. Theorem 1 can be used to obtain the bound for the case of $M < N$. Consider the lower-bound on the number of registers for array-level $DFC(M_w/N_w/w, \_, \_)$. In array-level design, the basic unit of data is an array of length w. Therefore, the size of each register and the buffer, and the width of the bus are assumed to be w.

Theorem 2

The minimum number of array-level registers (R_{min}) needed for $DFC(M_w/N_w/w, \_, \_)$ is

$$R_{min} = M - g, \text{ where } M \geq N \text{ and } g = gcd(M, N).$$

Proof: The $DFC(M_w/N_w/w, \_, \_)$ is an array-level DFC. Also, the I/O delays are not specified. An input sequence consists of M arrays of length w. An output sequence consists of N arrays of length w. To minimize the number of registers needed, after a new input sequence is input, data must be output until the number of remaining data in the DFC is less than N. If another input sequence is received before all the available data is output, the new input sequence is stored in the registers of length M and not processed until the number of remaining data from the previous sequence is less than N. Therefore, a design with minimum number of registers will not input a new input sequence before it is required for output.

When the DFC receives an input sequence consisting of M arrays during the first clock cycle, an output sequence consisting of N arrays can be output from this input sequence immediately, without storing the data. The remaining $M-N$ arrays need to be stored in the registers. If $M-N \geq N$, we can produce $k-1$ more output sequences until $0 \leq M-kN < N$, for some positive integer k.

Let R be the amount of data left in the registers just before a new input sequence is entered. Then, $R = M \bmod N$. Consider the maximum value taken by R. Note that, $R = M \bmod N = (M' \bmod N')g$, where $g = \gcd(M,N)$, $M = M'g$ and $N = N'g$. Let $R = R'g$. Note that, after inputting the a-th input sequence, $R = aM \bmod N = (aM' \bmod N')g$. After the N'-th input sequence, $R = N'M \bmod N = (N'M' \bmod N')g = 0$. The values taken by R repeats itself after the N'-th input sequence. Therefore, we only have to consider the values taken by R during the first N' inputs.

Note that, $0 \leq R' \leq N'-1$. We will show that R' takes N' distinct values during the first N' inputs. Then, R' reaches $N'-1$ as its maximum. Assume that R' does not have N' distinct values during the first N' inputs. Then, the values taken by R' for two distinct input sequences during the first N' inputs must be equal; i.e., for some a-th and b-th input sequences during the first N' inputs, $aM' \bmod N' = bM' \bmod N' = c$, where $a \neq b$, $1 \leq a,b \leq N'$, and $0 \leq c \leq N'-1$. In other words, $aM' = \alpha N' + c$ and $bM' = \beta N' + c$ for some positive integers α and β, where $\alpha \neq \beta$. Then, $(a-b)M' = (\alpha - \beta)N'$; i.e., $(a-b)M' \bmod N' = 0$. However, M' and N' are relatively prime and $0 \leq a-b < N'$. Therefore, a must be equal to b. This contradicts our assumption that $a \neq b$. Therefore, R' must have N' distinct values during the first N' inputs. Hence, R' reaches $N'-1$ as its maximum. The maximum of $R = R'g = (N'-1)g = N-g$.

Assume that R reaches the maximum value and a new input sequence is entered. Before the new input sequence of M arrays is stored, an output sequence of N arrays can be output. At this moment, the number of registers needed reaches its maximum, $R + M - N = (N-g) + M - N = M - g$. Thus, the lower-bound on the array-level registers (R_{min}) is $M-g$. The lower-bound on the number of registers is $(M-g)w$. □

Now consider the lower-bound on the number of registers needed for the bit-level design. For $DFC(M_x/N_y/w,\_,\_)$, each element of an array can be handled individually.

Theorem 3 *The minimum number of registers (R_{min}) needed for $DFC(M_x/N_y/w,\_,\_)$ is*

$$R_{min} = (M+N-g)w - Ny\lfloor \frac{w-x}{y} \rfloor - min[Mx, Ny],$$

where $M \geq N$ and $g = gcd(M,N)$.

Proof: The proof has been omitted due to space limitations. It can be found in [13].

3.2: A Design Methodology

Our design methodology is based on a two-dimensional architecture, in which the registers are placed along multiple data paths to form a two-dimensional array of registers. This architecture leads to area-efficiency and higher throughput rate compared with earlier design schemes. Area-efficiency is obtained by fewer number of registers employed in this scheme. Higher throughput is achieved by parallel data movement on a two-dimensional architecture.

We show a methodology for synthesis of area-efficient DFCs when the parameters s, d, and g are specified. The *total delay* d_t is defined as the elapsed time between the arrival of the first input sequence of a group and the departure of the first input sequence of the next group. The throughput is defined as the ratio of the number of data in a group to d_t (see Section 2.1). Our methodology results in designs having maximum throughput(i.e., d_t is minimized).

We explain the methodology for $DFC(S_i/S_o/w,\_,\_)$ before considering the design of $DFC(S_i/S_o/w, D_i/D_o,\_)$, $DFC(S_i/S_o/w, D_i/D_o, GD_i/GD_o)$ and $DFC(S_i/S_o,\_, \hat{GD} = \hat{D})$. The design methodology for $DFC(S_i/S_o/w,\_,\_)$ is as follows.

1. **Calculation of minimum number of registers:**
   Calculate the minimum number of registers by using Theorem 3.
2. **Register allocation scheme:**
   The register allocation table shows the snapshots of the contents of the input buffer, the registers, and the output buffer as a function of time (See the second paragraph of Section 3). Compose the register allocation table for $DFC(S_i/S_o/w,\_,\_)$ according to the following rules.
   - The number of registers in a row is equal to the size of input sequence,

Figure 4. $DFC(4_2/2_3/6, \_, \_)$

Figure 6. One-Dimensional $DFC(4_2/2_3/6, \_, \_)$

4: Design Examples and Comparisons

Our design methodology can be applied to many real-time signal and image processing applications requiring fast data format conversion. In this section, the area requirements of our designs are compared with those designed by earlier methodologies. Five representative application examples related to video processing are also shown: word-level converter, matrix transposer, image-sampler, and DFCs for discrete wavelet transforms and for zig-zag scanning.

4.1: Comparisons with Earlier Designs

We have designed several DFCs using our approach and compared the designs with those designed by earlier techniques in the literature. An expert designer may obtain smaller circuit layouts than either a novice designer or that produced by an automatic design tool. For a fair comparison, the layouts for both the schemes are produced using well known Cadence Partitioning and Routing Tool in its automatic mode using standard MOSIS CMOSN cell libraries in 1.2 micron technology. Consider the entry corresponding to $DFC(8_4/4_6/12, 1/1, \_)$ in Table 1. We designed the one-dimensional DFCs shown in Table 1 using the forward-backward allocation scheme in [3]. As can be expected, the number of registers is not the main factor contributing to the overall area. Since fewer control states are needed in our design, the area of DFC decreases considerably. This example can be regarded as an extension of $DFC(4_2/2_3/6, \_, \_)$ shown in Section 3. Note that the delay g is specified in

Example	Performance	One-dimensional DFC [3]	Two-dimensional DFC	Reduction (%)
$DFC(8_4/4_6/12, 1/1, \_)$	Core size (mm^2)	1.389×1.324	0.895×0.925	54.98
	Chip size (mm^2)	4.344×3.916	3.732×3.917	14.07
	# of registers	64	48	25
	# of control states	96	5	94.8
$DFC(4_2/2_3/24, 1/1, \_)$	Core size (mm^2)	1.439×1.368	1.124×0.894	48.96
	Chip size (mm^2)	2.186×2.086	1.647×2.086	24.65
	# of registers	52	48	7.7
	# of Control states	96	18	81.25

Table 1. Comparison of Example Layouts - I

$DFC(8_4/4_6/12, 1/1, \_)$. The number of states required by the two-dimensional DFC does not change in spite of increased data size. However, the number of control states needed in the forward-backward allocation scheme [3] is four times as much as that of $DFC(4_2/2_3/6, \_, \_)$, in proportion to the increased data size.

The number of pins for the design of $DFC(8_4/4_6/12, 1/1, \_)$ is 59. The number of pins for $DFC(4_2/2_3/24, 1/1, \_)$ is reduced to 17 because the number of pins for I/O sequences decreases. In the designs for $DFC(4_2/2_3/24, 1/1, \_)$, the reduction in core size is 48.96%. This is smaller than that of the previous example (54.98%). However, the size of the complete chip is reduced by 24.65%. This reduction is larger than the reduction in the previous example (14.07%). Since the number of pins and pads are reduced considerably compared with the previous example, the core of the design occupies significant area of the complete chip.

In conclusion, our design methodology leads to reduced circuit area due to fewer number of registers and fewer control states, and results in increased throughput rate compared with one-dimensional DFCs designed by the technique in [3].

4.2: Illustrative Designs

Five representative application examples related to video processing are also shown in this subsection. These are word-level converter, matrix transposer, image-sampler, and DFCs for discrete wavelet transforms and for zig-zag scanning. See [13] for details.

5: A Simple Design Methodology employing a Dual Buffer

In the design of two-dimensional DFCs, we focussed on reducing the number of registers (as in the earlier design methodologies) to achieve area-efficiency. However, two-dimensional DFCs may lead to circuits with complex wire connections, which can be a dominant part of the entire circuit. Also, the two-dimensional DFC requires specification of the register allocation table during its design procedure. Heuristics are needed to schedule the register allocation. This also makes it difficult to automate the design procedure. To realize a simple and automated design procedure required for rapid-prototyping, it is important to simplify the wire connections, the overall structure, and the controller architecture. Minimizing the number of registers may not be the most important consideration.

In this section, we propose an alternate design methodology for the design of DFCs that employs a *Dual Buffer*. We use a simple and regular structure compared with the two-dimensional DFCs. Even though it requires more number of registers, the area occupied by the wire connections and the control circuit is reduced due to the simple and regular structure of the design. This architecture also leads to a simple design procedure; it is easy to modify and expand as the design parameters change.

5.1: Design of Dual Buffer DFCs

The dual buffer DFC consists of an input module, an output module and three data busses connecting the I/O pins and these modules as shown in Figure 7. The input module receives

Figure 7. Structure of Dual-Buffer DFC

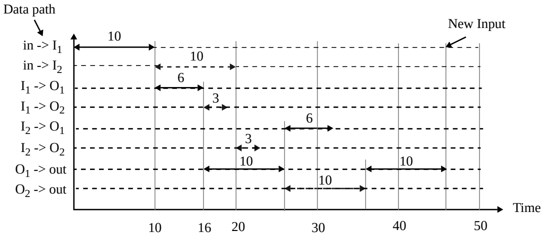


Figure 8. Data Schedule Graph for $DFC(9_1/6_2/10,\_,\_)$

and stores the input data, and transfers it to the output module. The output module outputs the received data. The input and output operations can be overlapped by separating the input and output modules.

Each module consists of a dual buffer. A dual buffer consists of two identical buffers sharing the input and output data busses of a module. Each buffer has connections for data movement in two directions: South-to-North and West-to-East. If the format of I/O sequences is $M_x/N_y/w$, the buffers in the input module, I_1 and I_2, are $M \times w$ meshes. Similarly, the buffers in the output module, O_1 and O_2, are $w \times N$ meshes. The basic element of each mesh is a register with a 2×1 multiplexor placed on its input. One input of the multiplexor is connected to the output of the register on the left. The other input of the multiplexor is connected to the output of the register below it.

Initially, the input sequences are stored in I_1 until it becomes full. The direction of the data movement is South-to-North in I_1. After I_1 is full, we begin to transfer the data in I_1 to O_1. The direction of data movements is West-to-East in I_1 and O_1. At the same time, we begin to store the new input sequences in I_2. If O_1 becomes full, we begin to send out the output sequences. The direction of data movement is South-to-North in O_1. If $M > N$, the remaining data in I_1 is transferred to O_2.

By using the dual buffers, while the data is transferred from the input module to the output module, the I/O operations can be performed on the other buffer of each module. The three busses are switched to the appropriate buffer of each module to maximize the use of each buffer.

Consider the design of $DFC(9_1/6_2/10,\_,\_)$. The buffers I_1 and I_2 are 9×10 meshes. The buffers O_1 and O_2 are 10×6 meshes. The schedule of data movements is specified by a *data schedule graph* (See Figure 8). For example, during time $t = 10$ and 16, the data movements for the data paths "*in* $\rightarrow I_2$" and "$I_1 \rightarrow O_1$" are shown active; the input data is stored in I_2, and the data in I_1 is transferred to O_1.

Note that in the two-dimensional DFC (as well as in the earlier designs), the connections between the registers depends on the specification of the DFC. However, the proposed dual buffer DFCs have the same architecture as shown in Figure 7. Hence, the number of control signals required for any dual buffer DFC to control the I/O buffers and the multiplexors is the same for any design based on this methodology. The control signals can be derived from the data schedule graph.

5.2: Comparison with Two-Dimensional DFCs

In general, the dual buffer DFCs require more number of registers than the two-dimensional DFCs. However, it has simpler wire connections and has simple control compared with the two-dimensional DFC. To illustrate the effects of these circuit components on the total circuit area, the layouts of two example DFCs are compared in Table 2.

In the designs for $DFC(9_1/6_2/10,\_,\_)$, the number of registers required for dual buffer DFC is 4.76 times that required in the two-dimensional DFC. However, the ratio of the core size is 2.96. We can see the effect of the dual buffer design on the area. The size of the I/O sequences and the arrays are increased in $DFC(10_3/6_4/12,\_,\_)$. The dual buffer DFCs in both examples require more number of registers than the two-dimensional DFCs. The ratio of the number of registers used in the designs is 4. However, the ratio of the core size is only 1.72. The chip size is

Example	Performance	Two-dimensional DFC	Dual-Buffer DFC	Ratio
$DFC(9_1/6_2/10,\_,\_)$	Core size (mm^2)	1.512×0.918	2.502×1.644	2.96
	Chip size (mm^2)	2.250×1.992	3.234×2.364	1.7
	# of registers	63	300	4.76
	Throughput	7.5	5.8	0.77
$DFC(10_3/6_4/12,\_,\_)$	Core size (mm^2)	1.743×1.813	2.262×2.399	1.72
	Chip size (mm^2)	3.728×3.715	3.726×3.718	1.0
	# of registers	96	384	4
	Throughput	22.5	7.5	0.3

Table 2. Comparison of Example Layouts - II

almost identical. In this larger example, the effect of the simple structure is magnified. In general, if compact design is not a major consideration and a simple design procedure is preferred, then the dual buffer DFCs are attractive.

The throughput of the dual-buffer DFC is lower than that of the two-dimensional DFC. The throughput is limited by the smallest bandwidth among the three busses connecting the I/O modules. However, we can change the wire connections of the buffers such that the data in multiple rows or columns can be transferred during a clock cycle. Then, we can increase the throughput of the dual buffer DFC. This can be done by a simple modification to the above design technique and it will not significantly increase the circuit area.

6: Conclusion

We have proposed two design methodologies for the synthesis of a class of DFCs. The first design methodology employed a two-dimensional structure. It resulted in area-efficient designs by reducing the number of circuit components such as registers and multiplexors compared with the previous techniques. The size of the control circuit was also reduced compared with the previous techniques by reducing the number of control states. The area efficiency of our design was shown by comparing the circuit layouts of DFCs designed by our scheme with the earlier ones. For a given number of registers and a specification of the I/O sequences, our designs achieve maximum throughput by minimizing the total delay. Improved performance was obtained by extending the dimension of the architecture from one to two; two-dimensional organization seems to be natural to process two-dimensional data.

Our second approach employing dual buffers has a simple and regular architecture. Any dual buffer DFC designed using our approach has the same number of control signals. In our approach, the area occupied by wire connections and control circuits was reduced. In addition, the simplicity of our design methodology allows easy modification and expansion as the design parameters change. This design procedure is well suited for rapid-prototyping.

To extend the functions of DFCs, we need to develop a new notation for I/O relationship. If we attempt to implement a DFC for a large class of permutations, the resulting DFC will be very complex. Therefore, careful consideration is required in extending the function of DFCs.

Acknowledgments

We thank Dong-Hyun Heo for his assistance in implementing the design layouts. Also, we would like to thank Karthik Kumar and Seonil Choi for their assistance in preparing the final version of the manuscript.

References

[1] J. Bae and V. K. Prasanna, "A General Framework for Synthesis of Data Format Converters," *IEEE International Conference on Parallel Processing,* 1994.

[2] B. W. Wah and M. Aboelaze, "Systematic design of buffers in macropipelines of systolic arrays," *J. Parallel Dist. Comp. vol. 5,* pp. 1-25, 1988.

[3] K. K. Parhi, "Systematic Synthesis of DSP Data Format Converters Using Life-Time Analysis and Forward-Backward Register Allocation," *Proc. 1992 IEEE Trans. Circuits and Syst. vol. 39,* pp. 423-440, July 1992.

[4] H. Park and V. K. Prasanna, "A Class of Optimal VLSI Architectures for Computing Discrete Fourier Transform," *International Conference on Parallel Processing,* 1992.

[5] C. Chakrabarti, M. Vishwanath, and R. M. Owens, "A Survey of Architectures for the Discrete and Continuous Wavelet Transforms," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing,* 1995.

[6] N. Ranganathan, "A Prototype VLSI chip for JPEG Image Compression Standard," *Proc. IEEE EDAC-EURO ASIC-95 Conference,* 1995.

[7] J. Bae, N. Park, S. Choi and V. K. Prasanna, "Design of an Architecture for Low Bit-rate Videoconferencing System," *Multimedia Technology and Applications Conference,* March 1996.

[8] J. Bae, N. Park, S. Choi and V. K. Prasanna, "Hardware/Software Codesign for a Low Bit-rate Videoconferencing System," *Eleventh International Conference on Systems Engineering,* July 1996.

[9] "ITU-T Recommendation H.263," *Draft ITU-T SG 15 Recommendation H.263,* Nov. 1995.

[10] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures,"*ACM/IEEE Design Automation Conf.,* pp. 155-169, 1971.

[11] J. Bae and V. K. Prasanna, "Synthesis of a Class of Data Format Converters with Specified Delays," *IEEE International Conference on Application Specific Array Processor,* 1994.

[12] J. Bae and V. K. Prasanna, "Synthesis of VLSI Architectures for Two-Dimensional Discrete Wavelet Transforms," *Int. Conf. on Application Specific Array Processors,* July 1995.

[13] J. Bae and V. K. Prasanna, "Synthesis of VLSI Architectures for Two-Dimensional Discrete Wavelet Transforms," *Technical Report CENG 98-10*, University of Southern California, April 1998.

[14] J. Bae and V. K. Prasanna, "A Fast and Area-Efficient VLSI Architecture for Embedded Image Coding," *International Conference on Image Processing,* Oct. 1995.

[15] H. M. Alnuweiri, "Routing BPC Permutations in VLSI", *Proc. 1992 IPPS,* March 1992.

[16] O. Rioul and M. Vetterli, "Wavelets and Signal Processing," *IEEE Signal Processing Magazine,* pp. 14-38, Oct. 1991.

[17] K. K. Parhi and Takao Nishitani, "VLSI Architectures for Discrete Wavelet Transforms," *IEEE Trans. VLSI Systems vol. 1, No. 2,* June 1993.