# A New Register Allocation Scheme for Low Power Data Format Converters

**Kala Srivatsan, Chaitali Chakrabarti**

Department of Electrical Engineering
Arizona State University
Tempe, AZ 85287-5706
chaitali@asu.edu

**Lori E. Lucke**

Minnetronix, Inc.
2610 University Ave, Suite 400
St. Paul, MN 55114
lelucke@minnetronix.com

**Abstract**

In many applications, such as digital signal processing, data format converters are used to reformat the data transferred between processing modules. Various methods have been proposed to synthesize data format converter architectures while optimizing the number of registers used to store the data. In this paper, we present a new register allocation scheme which not only minimizes the number of registers, but also minimizes the power consumption in the data format converter. Low power data format converters are synthesized by minimizing the transitions and interconnections between the registers used to store the data. We present both a heuristic and an integer linear programming formulation to solve the allocation problem. Our method shows significant improvement over previous techniques.

# 1 Introduction

Data format converters (DFCs) are used to permute the data from one format to another in signal processing and image processing applications. A DFC is used to transpose the data within an algorithm or to reorder the data transferred between heterogeneous modules within an implementation. (The modules within a heterogeneous implementation are assumed to operate on different block lengths and different wordlengths.) Examples of DFCs include matrix transposers, data sequencers, serial to parallel converters, and digit-serial to bit-parallel converters. In this paper we concentrate on the design of low power data format converters. Low power design of DFC architectures is of particular importance since DFCs represent a sizeable portion (20% to 40%) of a VLSI chip, especially for two-dimensional DSP systems. Current industry trends towards low power VLSI circuits mandates a DFC design that minimizes power consumption.

DFCs consist of data registers, interconnect, and control. The registers read in data from the input bus and place data on the output bus. The registers communicate with each other via dedicated interconnections. The width of a register depends on the data wordlength. In this paper we concentrate primarily on minimizing the power consumed in the registers and their interconnections.

The power consumption of a CMOS VLSI circuit can be modeled as $P = 0.5\alpha f_c C_l V_{dd}^2$, where $\alpha$ is the number of transitions, $f_c$ is the clock frequency of the circuit, $C_l$ is the effective load capacitance, and $V_{dd}$ is the power supply voltage [5]. Thus an effective way of reducing the power consumption in the DFC registers is by reducing the number of register transitions. This is equivalent to reducing the number of variables that move from one register to another.

Recently, techniques have been proposed to synthesize data format converters using the minimum number of registers [1, 2, 3, 6]. The forward-backward allocation scheme in [1] results in a serial interconnection of registers, thereby increasing the interconnection area. A 2-D extension of this scheme is proposed in [2], where multiple data are input and output at the same time, resulting in reduced interconnect area. The design methodology presented in [3] for implementing DFCs in a 2D architecture also results in a small area. All these schemes require large number of register transitions making them unsuitable for low power applications. The sequencer based data path synthesis scheme in [6] is the only other scheme that tries to reduce the number of memory/register access operations by exploiting the regularity of patterns.

We recently proposed a new register allocation scheme to design low power DFCs [4]. Our register allocation scheme uses the minimum number of registers and minimizes the power consumption by first minimizing the number of register transitions. We further refine the allocation to minimize register interconnects and to reduce the control circuit complexity as these secondary concerns also affect the power consumption. We propose a new register allocation scheme called *semi-static allocation*, where each variable is allocated to as few registers as possible. It can be shown that this scheme runs to completion and also sustains interframe pipelining rate as in [1]. We also present an integer linear programming (ILP) model for optimal allocation of variables to registers. In this paper we concentrate on proving the correctness of this approach by implementing and experimenting with several examples. Implementations using Mentor Graphics CAD Tools show that our designs consume significantly less power compared to [1],[2]. The semi-static allocation scheme results in larger area since more control signals are required for the gated clocks (used to hold the data in the registers), and larger number of multiplexers are required to gate the outputs to the output bus. However, the reduction in register switching activity more than outweighs the extra interconnection complexity yielding a lower power converter.

The rest of the paper is organized as follows. The proposed greedy heuristic and the ILP formulation are discussed in Section 2. Several data format converters are compared with respect to switching activity, area and power consumption in Section 3.

## 2 Low Power Register Allocation Scheme

In this section, we propose two methods, one based on heuristics and the other based on ILP formulation, for designing low power data format converters. Both methods achieve low power design by minimizing two factors: number of register transitions and number of split variables. Minimizing the transitions reduces the activity factor while minimizing the split variables reduces the register interconnect and the control complexity.

### 2.1 Proposed Heuristic

The proposed heuristic tries to minimize both the number of transitions of any particular variable as well as the number of variables undergoing transitions. Let $P$ be the period (defined as the number of time steps necessary to input all the variables for one data conversion). Let $L_i$ and $D_i$ be the birth time and death time of variable $i$.

**Algorithm:**

- Step 1: Find the minimum number of registers using lifetime analysis [1].
- Step 2: Divide the variables in to three groups such that group (I) consists of variables with lifetimes equal to $P$, group (II) consists of variables with lifetimes less than $P$ and group (III) consists of variables with lifetimes greater than $P$.
- Step 3: Assign variables in group (I) directly to individual registers. Since all the variables in this group have time period equal to $P$, each variable is assigned to a different register. Update the available timeslot after this assignment.
- Step 4: Split each variable in group (III) into two variables: one with lifetime equal to or less than period, $P$, and the other with the remaining lifetime. Repeat Step 3 on the variables with lifetimes equal to $P$. Update the available timeslot. The unassigned variables in Step 4 are assigned in the next step.
- Step 5: Sort the group (II) variables and unassigned variables from Step 4 in decreasing order of their lifetimes. Using this sorted list, collect variables into subgroups such that no two variables in a subgroup have overlapping lifetimes and the sum of the lifetimes of all the variables within a subgroup is less than $P$. An ideal case for a subgroup would be if the death time of each variable is the birth time of some other variable and the combined lifetime of all variables equals $P$. Sort the subgroups in decreasing order of the sum of the lifetime values of all variables contained in those subgroups. Allocate the subgroups from the sorted list to registers. Update the available time slot. Variables that could not be sorted into subgroups are allocated in Step 6.
- Step 6: Assign the variables in the available time slots in decreasing order of their lifetimes. Update the available timeslot after each assignment. Repeat this step till all the variables are allocated to registers.
- Step 7: Regroup the variables in a different way and repeat Steps 4, 5 and 6 if more than one variable gets split. Repeat steps 4, 5 and 6 till the minimum number of variables are split.

We explain this procedure with the help of the $4 \times 4$ sequential matrix transposer example from [1]. In this example, the minimum number of registers determined from lifetime analysis is 9 and the period is 16 time units. There are no variables in group (I) and hence Step 3 is not applicable.

In Step 4, the variable $d$ can be split into two variables $d_1$ and $d_2$ by more than one method. If variable $d$ is split into $d_1$ with lifetime $(5-21)$ and $d_2$ with lifetime $(3-5)$, then the allocation requires 8 transitions and 2 additional variable splits. If, on the other hand, variable $d$ is split with variables $d_1$ and $d_2$ having lifetimes $(6-21)$ and $(3-6)$, respectively, then the allocation results in only 1 additional variable split and the same number of transitions. Fig. 1 shows the assignment of variables to registers by the proposed method. The total number of transitions required by this method is 24. Note that we have slightly improved the allocation of split variables compared to that reported in [4]. This leads to better results in most cases.

### 2.2 ILP Formulation

We next describe an ILP model for optimally allocating variables to registers. This ILP model finds a register allocation that minimizes total power consumption by modeling transition minimization, and variable split minimization. We define the following parameters for the ILP model.

$I$ and $J$ denote the set of variables and registers, respectively. $K$ denotes the total number of time steps and $P$ denotes the period. Note that $K$ is larger than $P$ since the schedule for the allocation of registers overlaps from one period to the next.

A variable $i \in I$ exists between $L_i$ and $D_i$ where $L_i$ is the birth time and $D_i$ is the death time.

$x_{i,j,k}$ denotes a binary variable that has a value of 1 if variable $i$ is assigned to register $j$ at time $k$ and has a value of 0 otherwise.

$y^1_{i,k}$ denotes a binary variable that takes on a value 1 if variable $i$ switches to a higher number register from time $k$ to time $k+1$ and takes on a value of 0 otherwise.

$y^2_{i,k}$ denotes a binary variable that takes on a value 1 if variable $i$ switches to a lower number register from time $k$ to time $k+1$ and takes on a value of 0 otherwise.

$S_i$ denotes a binary variable that takes on a value of 1 if variable $i$ splits during its lifetime and takes on a value of 0 otherwise.

The ILP model minimizes the power consumption while satisfying the following constraints.

$$\text{Minimize}\ \ \text{COST} = C_1 \sum_i S_i + C_2 \sum_i \sum_k y^1_{i,k} + C_2 \sum_i \sum_k y^2_{i,k} \tag{1}$$

$$\sum_j x_{i,j,k} = 1 \text{ for } i \in I, j \in J, L_i \le k \le D_i \tag{2}$$

$$\sum_i x_{i,j,k} \leq 1 \text{ for } i \in I, j \in J, 0 \leq k \leq K \tag{3}$$

$$My^1_{i,k} + \left( j \sum_j x_{i,j,k} - j \sum_j x_{i,j,k+1} \right) \geq 0 \text{ for } i \in I, j \in J, L_i \leq k \leq D_i - 1 \tag{4}$$

$$My^2_{i,k} + \left( j \sum_j x_{i,j,k+1} - j \sum_j x_{i,j,k} \right) \geq 0 \text{ for } i \in I, j \in J, L_i \leq k \leq D_i - 1 \tag{5}$$

$$\sum_j x_{i,j,k} + \sum_j x_{i,j,k+P} \leq 1 \text{ for } i \in I, j \in J, k+P \leq D_i \tag{6}$$

$$MS_i - \left( \sum_k y^1_{i,k} + \sum_k y^2_{i,k} \right) \geq 0 \text{ for } i \in I, j \in J, L_i \leq k \leq D_i - 1 \tag{7}$$

The cost function in the ILP formulation is a function of the transition minimization and variable split minimization. Transition minimization is achieved by including the terms $y^1_{i,k}$ and $y^2_{i,k}$ in the COST function of the ILP. The number of variables that are split is minimized by the term $S_i$ in the COST function. The results presented at the end of this section are based on a simplistic assignment of $C_1 = C_2 = 1$. Needless to say that a better allocation would be obtained by calculating realistic values of $C_1$, $C_2$ from circuit simulations.

The significance of equations (1) through (6) are as follows. Constraint 2 ensures that each variable is assigned exactly to one register during each time step. Constraint 3 ensures that a register can have a value of 1 or 0 during each time period. Constraint 4(5) checks for transition of a variable from a lower(higher) number register to a higher(lower) number register at each time step. Here $M$ is a predefined large number. Constraint 6 is the period constraint which ensures that if a variable is allocated to a particular register at time $k$ then no other variable is allocated to the same register at a time $k+P$, where $P$ is the period. Constraint 7 reduces the total number of variables getting split. The ILP models were solved using the GAMS/OSL solver [7].

# 3 Comparisons and Conclusions

Table 1 compares the number of register transitions obtained for each of the existing methods [1], [6] with the proposed heuristic and ILP formulation. Table 2 compares the activity factors of the proposed methods with those of [1], [6]. Note that we have included the results of input and output transitions which were not counted in our original work [4]. The ILP model verifies the heuristic approach since it provides identical results. Other more generic methods for solving the allocation search problem, such as genetic algorithms may also be used. We expect the results would be similar.

In [1], every register makes a transition at every clock, resulting in an activity factor of one. The activity factors for the other techniques are calculated as the ratio of measured transitions divided by the number of transitions used in [1]. The activity factors indicate that the new method could lead to significantly less power consumption but do not account for circuit loading.

To get more accurate results, some of the DFCs were synthesized to the CMOSN 1.2 um standard cell library using the Mentor Graphics CAD tools. The synthesized designs were simulated in SPICE to generate accurate power consumption figures. Table 3 compares the area based on cell usage statistics, and the power consumption of the two methods. The large area increase in our method compared to [1] is due to the increased interconnect and muxes between registers. This increased interconnect does affect the circuit loading and thus lowers the power consumption results as compared to looking at the activity factors alone. However the reduced circuit activity more than offsets the increased loading and thus the power consumption is still significantly smaller. For instance, while the $(4 \times 4)$ seq-transposer requires almost twice the area (and thus approximately twice the load) than the design in [1], it requires only 42% of the power. If we multiply the increased load by the activity factor show in Table 2, the measured results agree well with the predicted results. Thus it is clear that the semi-static allocation scheme provides a significant amount of power savings.

The allocation schemes also work for 2D DFCs. We compared our designs with those in [2] by synthesizing using the Mentor Graphics CAD Tools as shown in Table 4. While our designs have 30-35% larger area, the number of register transitions is significantly lower. The increase in area is caused by the the control signals for the gated clocks and the large number of interconnects (connection between registers and between registers and MUXs) and MUXs that are required to route the data to the output bus. Our conclusion is that a good compromise between a low area DFC and a low energy DFC can be obtained by allowing only a restricted set of registers to be connected to the output bus.

**Acknowledgements** The authors gratefully acknowledge the support of the Center for Low Power Electronics. The authors would also like to thank Srikanth Adhiveeraraghavan of Arizona State University and Uong Chai of University of Minnesota for the help in synthesizing the DFC architectures.

## References


[1] K. K. Parhi, "Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation," *IEEE Trans. on Circuits and Sys.-II*, Vol. 39(7), pp. 423-440, 1992.

[2] M. Majumdar and K.K.Parhi, "Design of a Data Format Convereter using Two-Dimensional Register Allocation," *IEEE Trans on Circuits and Systems II*, vol. 45(4), pp. 504-508, 1998.

[3] J. Bae, V.K. Prasanna and H. Park, "Synthesis of a Class of Data Format Converters with Specified Delays," *Proc. of the Int. Conf. on Application Specific Array Processors, 1994*, pp 283-294.

[4] K. Srivatsan, C. Chakrabarti and L. Lucke, "Low Power Data Format Converter Design using Semi-Static Allocation," *Proc. of ICCD, 1995.* 235-255,

[5] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low power CMOS digital design'," *IEEE Jour. of Solid State Circuits*, Vol. 27(4), 1992.

1994.

[6] M. Aloqeely and C. Y. Roger Chen, "Sequencer based data path synthesis of regular iterative algorithms," *31st DAC proceedings 1994(IEEE Cat. No. 94CH3408-2)*, pp. 155-160, 1994.

[7] A. Brooke, D. Kendrick, and A. Meeraus, *GAMS: A User's Guide*, San Francisco, CA: The Scientific Press, 1992.

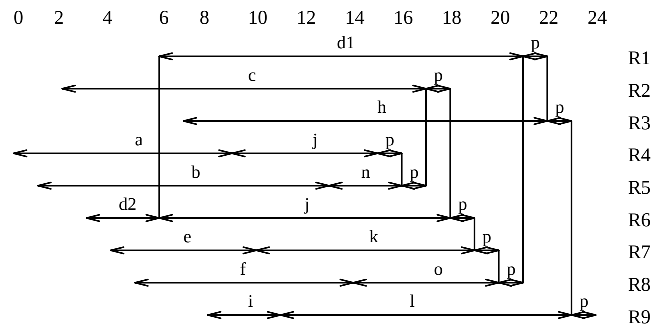


Figure 1: $(4 \times 4)$ sequential matrix transposer: assignment of variables by the proposed algorithm

| *DFC* | *[1]* | *[6]* | *heur.* | *ILP* |
|---|---|---|---|---|
| $3 \times 3$ seq-transposer | 36 | 20 | 11 | 11 |
| $4 \times 4$ seq-transposer | 144 | 48 | 24 | 24 |
| 1D-DWT (N=8,J=2) | 43 | 12 | 7 | 7 |
| $(2,1) \rightarrow (3,1)$[3]converter | 107 | 54 | 16 | 16 |
| $(3,1) \rightarrow (1,2)$[4]converter | 98 | 48 | 11 | 11 |
| $(4,1) \rightarrow (1,1)$[4] converter | 180 | - | 18 | 18 |
| $4 \times 4$ par-seq transposer | 180 | 57 | 18 | 18 |

Table 1: Comparison of the number of register transitions

| *DFC* | *[1]* | *[6]* | *heur.* | *ILP* |
|---|---|---|---|---|
| $3 \times 3$ seq-transposer | 1 | 0.55 | 0.31 | 0.31 |
| $4 \times 4$ seq-transposer | 1 | 0.33 | 0.17 | 0.17 |
| 1D-DWT (N=8,J=2) | 1 | 0.28 | 0.16 | 0.16 |
| $(2,1) \rightarrow (3,1)$[3]converter | 1 | 0.50 | 0.15 | 0.15 |
| $(3,1) \rightarrow (1,2)$[4]converter | 1 | 0.50 | 0.11 | 0.11 |
| $(4,1) \rightarrow (1,1)$[4] converter | 1 | - | 0.10 | 0.10 |
| $4 \times 4$ par-seq transposer | 1 | 0.32 | 0.10 | 0.10 |

Table 2: Comparison of activity factors

| | *Cell usage* | | *Power in mW* | | |
|---|---|---|---|---|---|
| *DFC* | [1] | *Ours* | [1] | *Ours* | *Reduction* |
| $3 \times 3$ seq-transposer | 62 | 108 | .06237 | .03976 | 36% |
| $4 \times 4$ seq-transposer | 148 | 272 | .12593 | .07214 | 42% |
| $(4,1) \rightarrow (1,1)$[4] converter | 109 | 290 | .16922 | .052018 | 69% |

Table 3: Comparison of cell usage statistics (where the cell complexity is that of a 2-input NAND gate), and power consumption of DFCs designed using [1] and the proposed method.

| | *2:1 MUXs* | | *Interconnect* | | *Area in sq.mm.* | | *Reg. transitions* | |
|---|---|---|---|---|---|---|---|---|
| | [2] | *Ours* | [2] | *Ours* | [2] | *Ours* | [2] | *Ours* |
| 1-D DWT | 6 | 14 | 22 | 30 | 4.56 | 5.91 | 28 | 16 |
| $4 \times 4$ par-transpose | 13 | 16 | 33 | 45 | 3.22 | 4.37 | 41 | 18 |

Table 4: Comparison of number of multiplexers, interconnects, layout area (2 $\mu$ CMOSN) and number of register transitions for DFCs designed using [2] and the proposed method.