# Transforming Dependence Graphs into Signal Flow Graphs During Systolic Array Processors Design

ARGIRIS MOKIOS
Aristotle University of Thessaloniki
Department of Electrical and Computer Engineering
54124, Thessaloniki
GREECE

STAVROS DOKOUZYANNIS
Aristotle University of Thessaloniki
Department of Electrical and Computer Engineering
54124, Thessaloniki
GREECE

*Abstract:* The algorithm for the transformation of Dependence Graphs, DGs, into Signal Flow Graphs, SFGs, during the design of Systolic Array Processors, SAPs, is presented. The transformation is a complex process, employing task scheduling, projection and the formation of recursive surfaces, i.e., hyperplanes corresponding to simultaneously executed processes during a common clock phase. A matrix method is developed, in order to overcome the limitation of more than 3-$Dimensional$ euclidean space graphical modeling, used up to now, and based on the designer imagination. The proposed method is intended to be used, in the near future, by Design Automation CAD tools.

*Key–Words:* Dependence Graph, modeling, Systolic Array Processors, task scheduling, recursive surfaces, projection vectors

## 1 Introduction

The notion of Systolic Array Processors was introduced in the decade of 60's, before 35 years, by excellent mathematicians involved in Computer Science computation problems. It refers to the graphical modelling of complex iterative computations like DSP [1], image processing [2], digital filtering [3] and cryptography algorithms [4], which could be implemented by mutually interconnected digitally operative hardware units, the so called processing elements, PEs.

The scope of the mentioned SAP was at that time, the hope that computations implemented in hardware could be much faster than there respective ones, implemented by software driven computations. The target was focused on real time systems, which were under development and needed constantly to grow faster and faster.

Moreover, in the decade of 60's, the computers were large, heavy and not movable and were installed only in large enterprises and government institutions. Thus, for at least 2 decades, the digital design through SAPs was considered in special cases of full and semi-custom physically designed VLSI chips.

Nowadays, with the advent of programmable devices, e.g., FPGAs, SOC, and fast DSP processors, the implementation of complex algorithms through SAPs becomes extremely actual and the need to develop CAD tools for their management has a very high priority.

This work is focused on one of a number of problems, necessary to be resolved during the SAP design. Namely, the transformation of DGs into SFGs, which provides a minimized and fully applicable for hardware implementation SFG, as shown in Fig. 1.

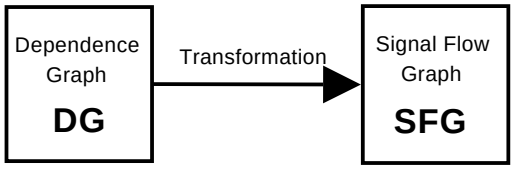


Figure 1: Dependence Graph to Signal Flow Graph transformation process

## 2 Dependence Graph to Systolic Array Transformation Process

The generation of a realizable systolic array, beginning from a Regular Iterative Algorithm (RIA) [5], is a complicated and time consuming process. Initially, a Dependence Graph (DG), i.e. a graph that describes the data dependencies of the algorithm, has to be generated from the RIA. Then the functionality of the DG's nodes has to be defined as described in [6]. It is also necessary to model the DG in a textual format using a specific description language, so that the DG can be further processed. Next, the dependence graph

has to be transformed into a Signal Flow Graph (SFG) by applying mapping methods.

An SFG is the graphical model, that allows the design of the final hardware, implementing a given dependence graph. It uses the same PEs with these of a DG, but it is modelled in a way that the lowest number of these PEs have to be able to implement the computations imposed by the DG.

The method for determining this lowest number of PEs is performed through Task Scheduling and Projection Vectors, described in detail in the next sections.

# 3 Task Scheduling and Projection Vectors

The derivation of a DG from a RIA leads to a *shift-invariant* graph, i.e., a DG where the dependence arcs that correspond to all it's nodes are independent of their positions. Properties like *shift invariance* (*homogeneity*) emphasize the regularity of the DG. These characteristics are exploited, in this section, in order to create intermediate structures like an SFG and therefore achieve the generation of regular processor arrays. In order to transform a DG into an SFG, PEs have to be assigned to the nodes of the DG. In the following, it is necessary to construct the scheduling diagram for these PEs. This will be achieved by using a projection and a scheduling vector respectively. The role of a linear projection vector is to assign DG's nodes, that belong to the same straight line segment, on only one SFG's PE. As a result, the projection maps the whole DG into an SFG. The role of a scheduling vector is to define the distinct time intervals where each DG's node is to be executed. Thus, it is ensured that only one DG's node will be executed by any SFG's PE in any time interval.

A linear scheduling vector is normal to a set of parallel uniformly spaced hyperplanes. These hyperplanes are important for the DG's nodes separation and their allocation in distinct time intervals, as all the nodes on the same hyperplane must be processed at the same time.

During the DG to SFG transformation, the DG is considered as being spanned into $N-1$, normal to the projection direction, planes. These parallel planes will be called *recursive surfaces*, because all the nodes included in each of these planes have the same recursion index. These surfaces are used as an intermediate step in the proposed transformation methodology. The $N-1$ recursive surfaces, as a consequence of the projection, will collapse in one level. Thus, the initial $N$–1-$Dimensional$ DG will be mapped onto a $N$–2-$Dimensional$ SFG.

As an example, the projection vector $[1\ \ 0\ \ 0]^T$ and the scheduling vector $[1\ \ 1\ \ 1]^T$ are applied to a DG that multiplies matrices $A$ and $B$. The DG is being spanned into four recursive surfaces, as shown in Fig. 2, where the dashed lines represent the parallel hyperplanes. A number is assigned to each hyperplane corresponding to the time interval that the included DG nodes are going to be executed. As a result of the collapse of the four surfaces into one the 2-$Dimensional$ mapping in Fig. 3 is derived. The enforcement of the linear scheduling vector, is necessary in order to specify the sequence of the operations in all the PEs of the SFG. The correct scheduling is verified by examining the insertion of elements $a_{ik}$ and $b_{kj}$ of matrices $A$ and $B$ in the SFG (see Fig. 4).

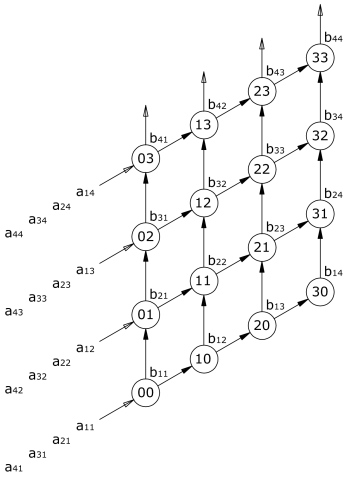


Figure 3: Post DG structure derived after the application of projection vector $[1\ 0\ 0]^T$.

Next, the behavior of the DG's nodes, during the projection process, has to be examined. Thus, Fig. 5 is considered where the PE in position $(0,\ 0,\ 0)$ and its adjacent PEs (positioned in $(1,\ 0,\ 0)$, $(0,\ 1,\ 0)$, and $(0,\ 0,\ 1)$), are illustrated in the 3-$Dimensional$ space. Projecting these nodes in the directions $i$, $j$ and, $k$, (i.e. using vectors $[1\ 0\ 0]^T$, $[0\ 1\ 0]^T$, and $[0\ 0\ 1]^T$) Fig. 6, 7, and, 8 are derived, where the nodes to PEs assignments are illustrated for each direction respectively. Each row of nodes parallel to the projection vector is minimized to a single PE. The coordinates of this PE are equal to its initial coordinates reduced by one dimension, i.e. the coordinate index that is parallel to the projection vector is deleted.

Whether an arc is going to be transformed in a *loop* or remain *simple*, depends on the non-trasmittent or transmittent status of the arc. In the processing element of Fig. 6 exist transmittent data $a_{ik}$ illustrated with the white arrows and non-transmittent data $b_{kj}, d_{ik}$ and $p_{ik}$ illustrated with the black arrows. If the recursive surfaces collapse in the same direction with a set of arcs and if an arc that belongs to this set is non-transmittal and obtain as source and sink the

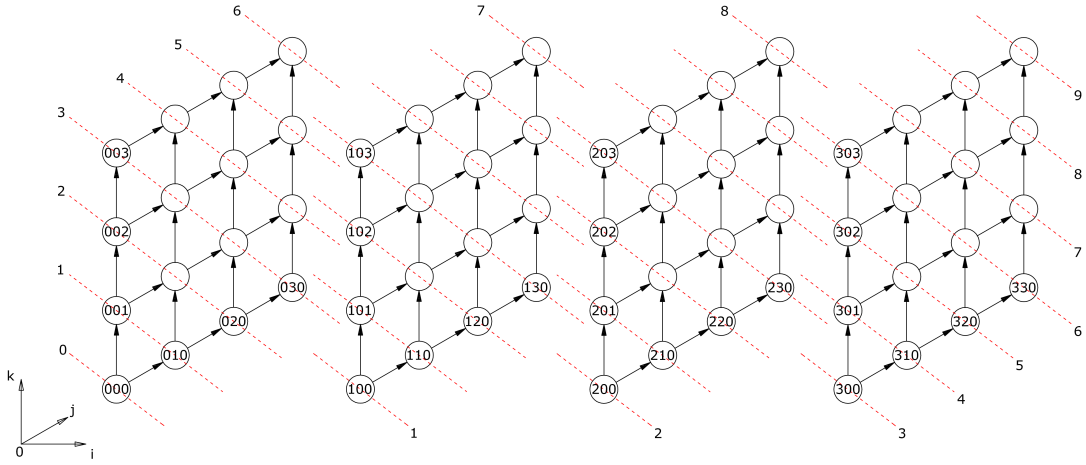


Figure 2: DG span onto recursive surfaces for projection vector $[100]^T$. The dashed lines represent the hyperplanes. A number is assigned to each hyperplane corresponding to the time interval that the included DG nodes are going to be executed. Each $4 \times 4$ plane (left to right) is an recursive surface for a value of $i$.

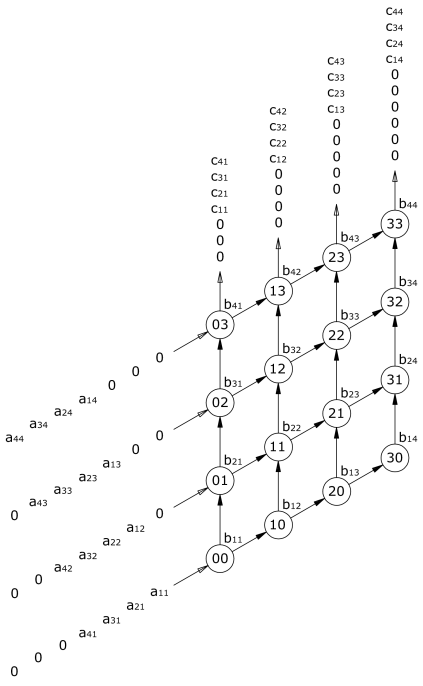

Figure 4: SFG structure derived after the application of scheduling vector $[1\ 1\ 1]^T$.

same PE, then it will be transformed into a *loop*, i.e., $\forall\, arc \in DG$

$$arc = \begin{cases} loop & \text{if } arc == non\text{–}transmittal \\ & \text{and } pe_{src} == pe_{snk} \\ simple & \text{otherwise} \end{cases}$$

For example, the PE in Fig. 8 obtains a loop that feeds the $p_{ijk}$ back to the PE.

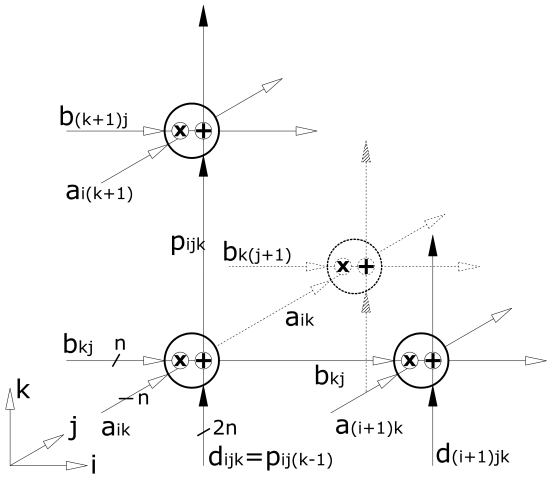


Figure 5: Illustration of the PE in position $(0,\ 0,\ 0)$ and its adjacent PEs in positions $(1,\ 0,\ 0)$, $(0,\ 1,\ 0)$, and $(0,\ 0,\ 1)$. The values of variables $a, b, d$, and $p$ denote the circulation of the input values (elements of the matrices) and the derived products throughout the DG.

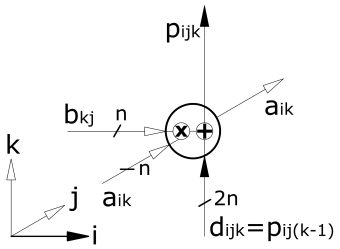


Figure 6: PE after $[100]^T$ projection. Input and output are determined only by j and k coordinates, as index i is eliminated during collapse.

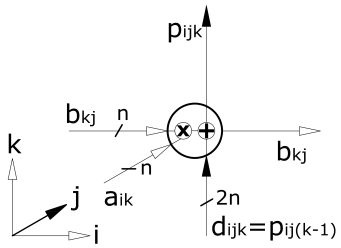


Figure 7: PE after $[010]^T$ projection. Input and output are determined only by i and k coordinates, as index j is eliminated during collapse.

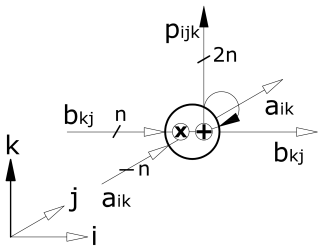


Figure 8: PE after $[001]^T$ projection. Input and output are determined only by i and j coordinates, as index k is eliminated during collapse.

# 4 Matrix Methods for Designing Signal Flow Graphs.

In order to derive a systematic method for designing SFGs, a nomenclature has to be defined. Thus, *pvec* is declared as the projection vector, *svec* as the scheduling vector, *dvec* as the dependence vector, i.e., the vector that defines the direction of the dependencies, *I/O* as the primary input and output of the DG, *arc* and *pe* as the DG's arcs and PEs respectively. I/O and arcs are considered to be complex elements that incorporate delays, which are denoted as *arc(dl)* and *I/O(dl)* respectively. PEs are also considered to be complex elements that incorporate coordinates and are denoted as *pe(crds)*.

## 4.1 Processing Element's Orthogonal Basis.

The processor basis $P_b$, introduced in [7], is required in order to determine the new PE's coordinates. $P_b$ is denoted by an $n \times (n-1)$ matrix and is orthogonal to the *dvec*, i.e.:

$$P_b{}^T \cdot dvec = \mathbf{0} \tag{1}$$

Given $dvec = [1\,0\,0]^T$ then $P_b$ would be an $3 \times 2$ matrix. Substituting these values to (1) and solving with respect to $p_{ij}$, the following equations are derived:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$p_{11} \cdot 1 + p_{12} \cdot 0 + p_{13} \cdot 0 = 0$$

$$p_{21} \cdot 1 + p_{22} \cdot 0 + p_{23} \cdot 0 = 0$$

Thus, $p_{11}$ and $p_{21}$ are equal to zero, while $p_{12}, p_{13}, p_{22}$, and $p_{23} \in \mathbb{N}$. The selection of the appropriate $p_{ij} \in \mathbb{N}$ in order to generate the $P_b$, will be assisted by the definition of restraining rules in the following subsections. In the case where $dvec = [010]^T$, $p_{12}$ and $p_{22}$ are equal to zero while $p_{11}, p_{13}, p_{21}$ and $p_{23} \in \mathbb{N}$. In a similar fashion when $dvec = [0\,0\,1]^T$, $p_{13}$ and $p_{23}$ are equal to zero while $p_{11}, p_{12}, p_{21}$ and $p_{22} \in \mathbb{N}$.

## 4.2 Determining the Coordinates of Processing Elements.

The new coordinates for every mapped PE are derived by the multiplication of the selected $P_b$ and the coordinates' vector of the PE, i.e.:

$$pe(crds') = P_b{}^T \cdot pe(crds) \tag{2}$$

The calculation of the new PEs coordinates for the $2 \times 2 \times 2$ DG shown in Fig. 9, using $P_b$ matrices $\left[\begin{smallmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \end{smallmatrix}\right]$, and $\left[\begin{smallmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right]$, is considered as an example. It was aforementioned that the selection of the $P_b$ heavily depends on $dvec$. In the case where $dvec = [1\,0\,0]^T$, thus $p_{11} = p_{21} = 0$, coordinate $i$ of each PE has no interference in the computation of the new coordinates. In the $P_b$ matrices given above, if the column that includes elements $p_{11}$ and $p_{12}$ is removed, then a set of *reduced* $P_b$ matrices ($\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, and $\left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]$,) is obtained. Considering the case where $P_b = \left[\begin{smallmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right]$, then the new coordinates of the DG's PEs shown in Fig. 9 would have the following values:

$$\begin{array}{rcl} pe_1(crds') = pe_2(crds') & = & [0\,0]^T, \\ pe_3(crds') = pe_4(crds') & = & [1\,0]^T, \\ pe_5(crds') = pe_6(crds') & = & [0\,1]^T, \\ pe_7(crds') = pe_8(crds') & = & [1\,1]^T. \end{array}$$

Using the new PEs' coordinates the SFG structure (shown in Fig. 10), is generated. Fig. 11 and 12 illustrate the SFG structures that correspond to $P_b$ matrices $\left[\begin{smallmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \end{smallmatrix}\right]$, and $\left[\begin{smallmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right]$. In the case where $P_b = \left[\begin{smallmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right]$ the derived SFG has 3 PEs instead of 4 as the rest of the SFGs. This node assignment is illegal, because since a linear projection vector is used every DG's node belonging in the same line, is expected to be assigned to the same PE of the SFG. Thus, in order to select a good $P_b$ every row of the reduced processor basis

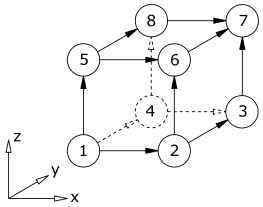


Figure 9: $2 \times 2 \times 2$ DG

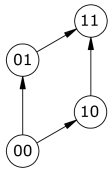


Figure 10: DG's nodes mapped SFG's PEs for processor basis consisting of $p_{12} = 1, p_{13} = 0, p_{22} = 0, p_{23} = 1$

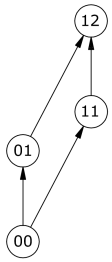


Figure 11: DG's nodes mapped SFG's PEs for processor basis consisting of $p_{12} = 0, p_{13} = 1, p_{22} = 1, p_{23} = 1$

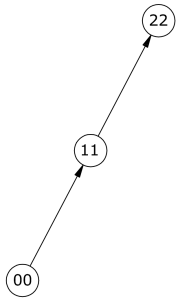


Figure 12: DG's nodes mapped SFG's PEs for processor basis consisting of $p_{12} = 1, p_{13} = 1, p_{22} = 1, p_{23} = 1$

matrix must be unique, i.e. its determinant must be non-zero:

$$\mid reduced\ P_b \mid \neq 0 \tag{3}$$

# 5 Signal Flow Graph Minimization.

During the mapping of the DG's nodes onto the SFG's PEs, duplicate arcs and collapsed nodes are generated. In order to obtain the final form of the SFG, all the redundant elements have to be removed. Starting from the PEs their new coordinates are examined and whenever a PE has the same coordinates with a previously examined one, then this PE is marked as *collapsed*, i.e.: $\forall\, pe \in DG$

$$pe = \begin{cases} collapsed & \text{if } pe'(crds) == pe(crds) \\ 0 & \text{otherwise} \end{cases}$$

All the DG's arcs are also examined, to determine if there are more than one arcs that have the same source and sink PEs. The second, third and so forth identical arcs encountered, are marked as *duplicate*, i.e.: $\forall\, arc \in DG$

$$arc = \begin{cases} duplicate & \text{if} \\ & arc'(pe'_{src}) == arc(pe_{src}) \\ & \text{and} \\ & arc'(pe'_{snk}) == arc(pe_{snk}) \\ unique & \text{otherwise} \end{cases}$$

In the example of Fig. 9, PEs $\{1, 4, 5, 8\}$ are collapsed to $\{2, 3, 6, 7\}$ respectively, as shown in Fig. 10. Thus, only one of these sets is going to be retained. The arcs $\{1 \rightarrow 4, 5 \rightarrow 8, 1 \rightarrow 5, 4 \rightarrow 8\}$ are duplicates of $\{2 \rightarrow 3, 6 \rightarrow 7, 2 \rightarrow 6, 3 \rightarrow 7\}$. One of these sets is going to be removed with respect to the previously retained set of PEs.

# 6 The Algorithm Transforming a Dependence Graph into a Signal Flow Graph.

In the previous sections the basic principles that have to be taken into account in order to transform a DG into an SFG, were presented. All the previously acquired knowledge is integrated in a complete algorithm that performs this transformation. As shown below, algorithm 1 initially calculates the delays that have to be incorporated into the I/O signals, in order to ensure the integrity of the SFG's data schedule. The number of delays for each I/O depends on the initial coordinates and the applied scheduling vector *svec*, and is derived by their vectors' product as shown in (4).

$$I/O(dl) = svec \cdot I/O(crds) \tag{4}$$

The number of the arc's delays have also to be calculated. These delays are necessary in order to identify if a generated SFG is capable of being transformed into a systolic array (SA), i.e.:

$$if\ \forall\ arc \in SFG \Rightarrow arc(dl) \geq 1$$

$$then\ SFG \Rightarrow SA$$

The number of delays for each arc depends on the associated dependence vector *dvec* and the applied scheduling vector *svec*, and is derived by their product as shown in (5).

$$arc(dl) = svec \cdot dvec \tag{5}$$

In the following a processor basis $P_b$ is selected, according to the constraints introduced in Section 4. The new PEs' coordinates are calculated and the arcs that interconnect them are identified as *loop* or *simple*. Afterwards, the I/O signals are mapped to the updated PEs. Finally, the SFG is minimized according to Section 5 and its final form is derived.

**Algorithm 1** DG2SFG Transformation Algorithm

1: **input:** DG data structure, $svec$, $pvec$
2: Calculate the number of delays for I/O signals.
3: **for** $I/O \in \mathcal{DG}$ **do**
4: $I/O(dl) = svec \cdot I/O(crds)$
5: **end for**
6: Calculate the number of delays for arcs.
7: **for** $arc \in \mathcal{DG}$ **do**
8: $arc(dl) = svec \cdot dvec$
9: **end for**
10: Select a processor basis $P_b : P_b^T \cdot pvec = \mathbf{0}$
11: Calculate the coordinates of the projected PEs.
12: **for** $pe \in \mathcal{DG}$ **do**
13: $pe(crds') = {P_b}^T \cdot pe(crds)$
14: **end for**
15: Identify *loop* arcs.
16: **if** $arc == non - transmittal$ **then**
17: **if** $pe_{src} == pe_{snk}$ **then**
18: $arc(type) = loop$
19: **end if**
20: **else**
21: $arc(type) = simple$
22: **end if**
23: Map $I/O$ signals to updated $PEs$.
24: Minimize the SFG by removing redundant elements.
25: **if** $pe'(crds) == pe(crds)$ **then**
26: remove $pe$
27: **end if**
28: **if** $arc'(pe'_{src}) == arc(pe_{src})$ **then**
29: **if** $arc'(pe'_{snk}) == arc(pe_{snk})$ **then**
30: remove $arc$
31: **end if**
32: **end if**
33: **output:** SFG data structure

# 7 Conclusion

A new algorithm has been presented for transforming a Dependence Graph into a minimized Signal Flow Graph.

The importance of the issued algorithm is expected in the development of a respective CAD tool, for transforming DGs in an automatic way, instead of a "by hand" method and by using human intuition that is used today.

The minimized form of an SFG is the first basis for the mapping of systolic algorithms SAPs onto programmable devices; in particular in FPGA and SOC devices. But in the near future it is also expected that fully implemented Computer Arithmetic Circuits, involving more complex computations, like convolution and DSP algorithms, will be embedded in DSP processors, which unfortunately today include only fast adders and multipliers.

*References:*

[1] M. Kortke, D. Fimmel, and R. Merker, "Parallelization of algorithms for a system of digital signal processors," in *Proc. 25th EUROMICRO Conference*, vol. 1, Sep.8-10 1999, pp. 46–50.

[2] I. Agi, P. Hurst, and A. Jain, "An expandable VLSI processor array approach to contour tracing," in *Proc. International Conference on Acoustics, Speech, and Signal Processing, ICASSP-88*, vol. 42, Apr.11-14 1988, pp. 1969–1972.

[3] S. Theodoridis, "Pipeline architecture for block adaptive LS FIR filtering and prediction," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 38, no. 1, pp. 81–90, Jan. 1990.

[4] M. Hani, T. S. Lin, and N. Shaikh-Husin, "FPGA implementation of RSA public-key cryptographic coprocessor," in *TENCON 2000 Proceedings*, vol. 3, Sep.24-27 2000, p. 611.

[5] S. Rao, "Regular iterative algorithms and their implementations on processor arrays," Ph.D. dissertation, Stanford University, Information Systems Laboratory, 1985.

[6] S. Dokouzyannis and A. Mokios, "Defining processing elements in dependence graphs from for-do programming constructs," in *6th WSEAS International Conference on Circuits, Systems, Electronics, Control and Signal Processing CSECS'07*, Dec.29-31 2007, pp. 15–20.

[7] S.-Y. Kung, *VLSI Arrays Processors*. Englewood Cliffs, NJ 07632: Prentice Hall, 1987.