A New EDA Defined Interchange Format for Processing of Dependence Graphs

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

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

Abstract: A new file interchange format has been proposed for interacting with standard EDA (Electronic Design Automation) software, oriented towards Systolic Array Processors (SAPs) design. The Dependence Graphs (DGs), which are large graphs consisting of a set of arcs and Processing Elements (PEs), are transcribed by using specific declarations and lists, in order to be compatible with standard SAP based EDA software.

Key–Words: Dependence Graph, modeling, Systolic Array Processors, EDA software

1 Introduction

The Electronic Design Automation (EDA), spans over a broad category of CAD software packages, oriented towards the automatic, semi-automatic and/or interactive analog and digital electronics design [1] and [2].

Recently, there is an interest in developing EDA software tools for building digital hardware implementing a special class of complex iterative algorithms, which are modelled as Systolic Array Processors. This important class of algorithms includes DSP applications [3], image processing [4], digital filters [5], cryptography [6] and computer arithmetic units [7] and [8].

Digital VLSI hardware, designed under SAP modelling, realizes the computations thousands and million times faster than traditional, software controlled, computer processing. Thus, such a hardware is directly applicable to real-time solutions, described by extremely severe timing demands as, e.g., in automatic trajectory control of aircraft and or space devices, in high definition TV, and many other.

This paper defines a new command-line interchange format, being able to introduce Dependence Graphs into standard digital embedded EDA software for FPGA, SOC, and DSP oriented processor VLSI design. This format is called Graph Description Interchange Format (GDIF) and is defined specifically for introducing Dependence Graphs into the previously mentioned EDA software.

GDIF introduces declarations and lists for Inputs, Outputs, arcs and Processing Elements, all describing the constituent components of Dependence Graphs. An example is given with the DG of Fig. 2a, modeling a $4 \times 4$ matrix multiplication. It is important to note that each arc on this graph models bundle of lines, depending on the external accuracy of the numbers that are included in the multiplied matrices. That is, it can have for example 32,64,128 or 256 binary lines. Similarly, the PEs are in general complex arithmetic logic units, ALUs, being able to implement computations on these bundles of binary data.

2 Dependence Graph Modelling Necessity

The generation of a realizable systolic array, beginning from a Regular Iterative Algorithm (RIA) [9], is a complicated and time consuming process. Initially, a dependence graph (DG) has to be generated from the RIA and it’s processing elements (PEs) has to be defined as described in [10]. Next the dependence graph has to be transformed into a systolic array by applying mapping methods. Usually this methods project the dependence graph in a specific direction in order to allocate the DG nodes to hardware processors and then they schedule the the input data flow in order to synchronize the function of the derived processor array.

In order to automate this transformation process, it is necessary to model the DG in a textual format using a specific description language. Acquiring this text representation, the DG can be further processed so to become a systolic array circuit.
Dependence Graph in Geometric Form

Dependence Graph in Textual Form

Figure 1: Dependence graph modelling

3 Graph Description Interchange Format

The goal of the Graph Description Interchange Format is to describe a dependence graph circuit in textual form. A DG is a graph that shows the dependence of the computations that occur in an algorithm. A DG can be viewed as a directed acyclic graph of combinational or sequential logic nodes called processing elements. Each PE has a logic function associated with it.

3.1 Dependence Graph Model

A DG model is a hierarchical circuit that describes the structure of a DG. A DG model is declared as follows:

```
.name <decl-model-name>
dims <dimensionality-of-DG>
.pe <DG-PE-list>
.arc <DG-arc-list>
.grp <group-of-PEs/arcs>
.input <DG-input-list>
.output <DG-output-list>
.end
```

The parameters that constitute the DG model declaration are analyzed in the following.

- `decl-model-name` is a string giving the name of the DG.
- `dimensionality-of-DG` is a white-space-separated list of two numbers that determines whether the DG is 2 or 3 dimensional and gives the value of it’s dimension (for the time being, the DGs are considered to be of the form \(N \times N\) or \(N \times N \times N\)).
- `DG-PE-list` is a list of PE’s that constitute the nodes of the DG. A description of its syntax is given in Section 3.2.
- `DG-arc-list` is a list of arcs that describes the interconnection of the DG’s nodes. A description of its syntax is given in Section 3.3.
- `group-of-PEs/arcs` is a group of PEs or arcs that can be used, instead of multiple .pe and .arc statements, to describe the DG’s structure. A description of its syntax is given in Section 3.4.
- `DG-input-list` is a list of input defining the primary input for the DG model being declared. A description of its syntax is given in Section 3.5.
- `DG-output-list` is a list of output defining the primary output for the DG model being declared. A description of its syntax is given in Section 3.6.

The GDIF parser demands the .name statement to be declared. If .name is not specified, then the parser will produce an error message and exit.

3.2 Processing Elements

A DG-PE-list declares the type of PEs that are used in order to construct the DG. It contains information about the name, the position and the associated logic function of each PE. A DG-PE-list declaration has following syntax:

```
.pe <PE-name> <PE-network> <PE-coordinates>
```

The .pe keyword informs the GDIF parser that a PE is declared. The PE-name is the name of the PE, the PE-network defines the logic function that is associated with the PE. The logic function is retrieved from the Berkeley Logic Interface Format (BLIF) ([11] appendix A) processing elements blocks library. BLIF describes a logic-level hierarchical circuit in textual form. The PE-coordinates designates the position of the PE in the DG.

```
Example 1
.pe pe11 "4bitmac.blif" 0 0
.pe pe21 "4bitmac.blif" 1 0
.pe pe31 "4bitmac.blif" 2 0
.pe pe41 "4bitmac.blif" 3 0
```

In this example, the processing elements that constitute the first row of a \(4 \times 4\) DG are declared.

It is important to note that the BLIF file describing the PE’s logic function must follow two naming conventions for the primary input and output names.
Naming convention 1
In general, the names of the primary I/O must be of the form:

\texttt{name '(' <bit-number> ')'}

The I/O name starts with a \texttt{name} that can be any alphanumeric string, but cannot have as its final character an \texttt{i}' or an \texttt{o}' character. Next, follows a \texttt{('} character, followed by a \texttt{bit-number} defining the number of bits that represent the signal. Finally the name is completed by a \texttt{')'} character.

Naming convention 2
In case of signals that are transmitted to neighboring PEs, the primary I/O names must be of the form:

\texttt{name 'i' '(' <bit-number> ')'}
\texttt{name 'o' '(' <bit-number> ')'}

The I/O name starts with a \texttt{name} that can be any alphanumeric string and has as its final character an \texttt{i}' or an \texttt{o}' character, depending on whether a input or output signal is declared. Next, follows a \texttt{('} character, followed by a \texttt{bit-number} defining the number of bits that represent the signal. Finally the name is completed by a \texttt{')'} character.

Example 3
\begin{verbatim}
.model 4bitmac
.inputs a(0) a(1) a(2) a(3) bi(0) bi(1)\bi(2) bi(3) d(0) d(1) d(2) d(3)...
.outputs bo(0) bo(1) bo(2) bo(3) bo(4)\bo(5) bo(6) bo(7) p(0) p(1)...
\end{verbatim}

In this example, a 4 bit multiply-accumulate (MAC) PE is declared. The MAC has three simple signals named \texttt{a}'s, \texttt{d}'s, and \texttt{p}'s and a transmitted signal named \texttt{b}'. The model’s \texttt{name} is declared in the first line. In the next line input \texttt{a}'s and \texttt{bi}'s are defined. Signals \texttt{a}'s and \texttt{d}'s are simple input signals and are named according to the naming convention 1. Signal \texttt{bi}'s is a transmitted signal and is named according to the naming convention 2. Signals \texttt{a}'s and \texttt{bi}'s are the multiplicands of the MAC unit and each of them is represented by 4 bits. Signal \texttt{d}' carries the 8-bit product of another MAC unit and is accumulated with the the partial product \texttt{a} \cdot \texttt{bi}. Finally, the primary output \texttt{bo}'s and \texttt{p}'s are defined, consisting of 4 and 8 bits respectively.

3.3 Arcs
A \texttt{DG-arc-list} declares the arcs that are used in order to interconnect the previously defined PEs of the DG. It contains information about the two interconnected PEs, as well as information considering their associated I/O. The syntax of a \texttt{DG-arc-list} declaration has the following form:

\begin{verbatim}
.arc <source-PE> <sink-PE> <PE-IO-associates>
\end{verbatim}

The \texttt{arc} keyword informs the GDIF parser that an arc is going to be declared. The \texttt{source-PE} is the name of the PE that the arc is going to begin from and the \texttt{sink-PE} is the name of PE that the arc is going to end to. The \texttt{PE-IO-associates} is a white space separated list of PE’s input and output. The selected I/O declare the association(s), in a communication level, of the logic functions between the \texttt{source-PE} and the \texttt{sink-PE}. In case the DG is symmetric and the \texttt{PE-IO-associates} are identical in the whole of the DG or in distinct arc groups of the DG, then the list must only be declared in the first arc of the DG (or group), and can be omitted in the following arc declarations.

Example 4
\begin{verbatim}
.arc pe11 pe21 bo bi
.arc pe21 pe31
.arc pe31 pe41
\end{verbatim}

Notice that the \texttt{PE-IO-associates} are declared only in the first line, since they are the same for the following arcs.

3.4 Groups
In sections 3.2 and 3.3 the syntax of \texttt{DG-PE-list} and \texttt{DG-arc-list} were presented. The disadvantage of that analytical syntax is that in case of large symmetric DGs the user has to write long .dg files in order to describe the structure of the DG. To avoid this problem, the \texttt{group-of-PEs/arcs} declaration was developed, which provides the ability to define groups of PEs and arcs in a single line. The syntax of the \texttt{group-of-PEs/arcs} declaration has the following form:

\begin{verbatim}
.grp <coordinates> <type> <PE-network>
.grp <coordinates> <type> <arc-direction> <PE-IO-associates>
\end{verbatim}

The \texttt{grp} keyword informs the GDIF parser that a declaration of a group is about to follow. The \texttt{coordinates} keyword defines the position of the group’s elements in the dependence graph. The \texttt{type} of the group’s elements is defined by \texttt{type} and can be one of ‘pes’ or ‘arcs’. If the \texttt{type} is ‘pes’ then the \texttt{PE-network} defines the logic function that is associated with the PE. If the \texttt{type} is set to ‘arcs’ then the \texttt{arc-direction} defines the direction of the arcs. The \texttt{PE-IO-associates} is a white space separated list of PE’s input and output. The selected I/O declare the association(s), in a communication level, of the logic functions between the \texttt{source-PE} and the \texttt{sink-PE}.

Example 5
\begin{verbatim}
.grp 00 30 pes 4bitmac.blif
.grp 00 30 arcs 10 bo bi
\end{verbatim}

The first line is equivalent to the declaration presented in Example 2 and the second line is equivalent to Example 4.
3.5 Input
A DG-input-list declares and associates the input signals with the primary input of the DG. A DG-input-list declaration has the following syntax:

.input <name> <destination-PE> <PE-associate>

The .input keyword informs the GDIF parser that a declaration of an input is about to follow. The name keyword defines the name of the input, destination-PE defines the PE where the input is to be inserted and the PE-associate defines the signal of the PE that is associated with the given input. If the input declarations that follow have the same PE-associate, e.g., due to DG symmetry, that field maybe omitted.

**Example 6**

```
.input A11 pe11 a
.input A21 pe21
.input A31 pe31
... 
.input B1 pe11 bi
.input B2 pe12
```

Notice that the PE-associate needs to be declared once for a set of inputs of the same type.

3.6 Output
A DG-output-list declares and associates the output signals with the primary output of the DG. A DG-output-list is declared as follows:

.output <name> <source-PE> <PE-associate>

The .output keyword informs the GDIF parser that an output is about to be declared. The name keyword defines the name of the output, source-PE defines the PE where the output is to be derived from and the PE-associate defines the signal of the PE that is associated with the given output. If the output declarations that follow have the same PE-associate, e.g., due to DG symmetry, that field maybe omitted.

4 An Example of a $4 \times 4$ Matrix Multiplication Dependence Graph Modelling

This section presents the modeling of the dependence graph that performs the multiplication of two $4 \times 4$ matrices, namely $A$ and $B$. The corresponding DG is depicted in Fig. 2a. The coordinates for the DG are presented in the $XYZ$ axis and the first node is considered to be on point $(0, 0, 0)$. The data elements of the matrices $A$ and $B$ are presented using $ijk$ variables where $i, j, k > 1$. The relation between $XYZ$ and $ijk$ is $i = X + 1$, $j = Y + 1$ and $k = Z + 1$. The data elements of matrix $A$, are inserted through the $Y$ direction and the data elements of matrix $B$ are inserted through the $X$ direction. Considering the structure of the DG in Fig. 2a and the processing element illustrated on Fig. 2b, it is obvious that every node of the DG is calculating a partial product of the form $a_{ik} \cdot b_{kj}$ and accumulates it with the product of the node below it. The sum of products is the output of the processing element. The summation of all the products derived by nodes with the same $x, y$ coordinates generates the final $c_{ij}$ product, e.g. the nodes with coordinates $0, 0, z$ where $z = 0, 1, 2, 3$ produce partial products $a_{11} \cdot b_{11}, a_{12} \cdot b_{21}, a_{13} \cdot b_{31}$ and $a_{14} \cdot b_{41}$; their sum generates $c_{11}$.

According to Section 3 the geometry of the DG, it’s I/O communications and it’s logic functionality are described in a GDIF file. More specifically, as shown in A using .grp declarations suffices to analyze the DG node structure and their interconnec-

Figure 2: Dependence graph for $4 \times 4$ matrix multiplication with $n$ bits elements, where $n = 8, 12, 16, 24, 32, 64 \ldots$. The relation between $XYZ$ and $ijk$ is $i = X + 1$, $j = Y + 1$ and $k = Z + 1$. 

...
tions. In order to define the data input and output for the current implementation, 32 .input and 16 .output declarations are used.

5 Conclusion

A new file interchange format has been proposed, for interacting with standard EDA software during Systolic Array Processors design on programmable FPGA, SOC, and DSP embedded hardware, as well as semi-custom VLSI integrated systems.

It is expected that further investigation of methods and tools for SAP design automation, alongside with the continuing progress of VLSI technology, will have as a result the building of extremely powerful digital processors in the near future.

A 4 × 4 Matrix Product DG in GDIF

# DG name
.name mtx_product

# dimensionality of the DG
# res value dimension
# word value n
.dims 3 4

# processing elements information
# res coordi type pe net
# word nates
.grp 000 333 pes 16bitmac.blif

# arcs information
# res coordi type direc pe io
# word nates
.grp 000 333 arcs 100 bo bi
.grp 000 333 arcs 010 ao ai
.grp 000 333 arcs 001 pd

# input information
# res word name dest pe pe input associate
# .input A11 pe111 ai
.input A21 pe211
.input A31 pe311
.input A41 pe411
.input A12 pe112
.input A22 pe212
.input A32 pe312
.input A42 pe412
.input A13 pe113
.input A23 pe213
.input A33 pe313
.input A43 pe413
.input A14 pe114
.input A24 pe214
.input A34 pe314
.input A44 pe414

# output
# res word name source pe pe output associate
# .output C11 pe114 p
.output C12 pe124
.output C13 pe134
.output C14 pe144
.output C21 pe214
.output C22 pe224
.output C23 pe234
.output C24 pe244
.output C31 pe314
.output C32 pe324
.output C33 pe334
.output C34 pe344
.output C41 pe414
.output C42 pe424
.output C43 pe434
.output C44 pe444

# end DG definition

References:


