HIGH-LEVEL SYNTHESIS BASED UPON DEPENDENCE GRAPH FOR MULTI-FPGA

Mohamed Akil

Laboratoire A2SI, Groupe ESIEE, Cité Descartes, Noisy Le Grand cedex, France

INVITED PAPER
MIDEM 2003 CONFERENCE
01.10.2003 - 03.10.2003, Grad Ptuj

Abstract: The increasing complexity of signal, image and control processing algorithms in real-time embedded applications requires efficient system-level design methodology to help the designer to solve the specification, validation and synthesis problems. Indeed the real-time and embedded constraints may be so strong that the available high performant processors are not so enough. That leads to use, in complement of processor, the specific component like ASIC or FPGA. Several projects have developed high-level design flow that translates high-level algorithm specification to an efficient implementation for mapping onto multi-component architecture. In this paper, we present: 1. a unified model for hardware/software codesign, based on the AAA methodology (Algorithm-Architecture Adequation). In order to exhibit the potential parallelism of algorithm to be implemented, the AAA methodology is based on conditioned (conditional execution of computations) factorized (loop) data dependence graph, 2.Some simple rules that allow synthesizing both the data path and the control path of a circuit corresponding to an algorithm specified as a Conditioned and Factorized Data Dependence Graph (CFDDG), 3. the optimized implementation of CFDDG algorithm onto FPGA circuit and Multi-FPGA (partionning), by using simulated annealing approach, 4. the resources and time delay estimation method. This method allows us to have a performance analysis for the implementation. The obtained results: resource (gates, IO) and latency estimation are used by the optimization step to decide which implementation respects the constraints (real-time implementation which minimises the resource utilisation), 5. the results of the implementation of the matrix-vector product algorithm onto a Xilinx Multi FPGA and the software tool SynDEx which implements the AAA methodology.

Visokonivojska sinteza FPGA vezij na osnovi odvisnih grafov

Izvleček: Naraščajoča zapletenost algoritmov za obdelavo signalov, slike in opravljanje nadzora v vgrajenih sistemih v realnem času zahteva učinkovite metodologije načrtovanja na nivoju sistema z namenom pomagati načrtovalcu reševati probleme pri specifikaciji, validaciji in sintezi sistema. V resnici se lahko zgodi, da so omenjene zahteve tako zahtevne, da omejijo uporabo obstoječih zmogljivih procesorjev. To navede na uporabo dodatnih specifičnih komponent, kot so ASIC vezja ali FPGA vezja. Pri nekaj projektih se je že zgodilo, da smo uspeli zahteve algoritmov na visokem nivoju prevesti na implementacijo arhitekture zasnovane na sistemu z večimi komponentami. V tem prispevku predstavimo : 1. poenoten model za sočasno načrtovanje programske in strojne opreme zasnovane na metodologiji AAA (Algorithm-Architecture-Adequation), ki je zasnovana na pogojnem podatkovno odvisnem grafu s čimer lahko izkoristimo potencialno paralelno izvajanje algoritma. 2. nekaj osnovnih pravil, ki omogočajo sintezo podatkovnih in kontrolnih poti za vezje, ki odgovarja algoritmu definiranemu kot CFDDG graf. 3. optimizirano implementacijo CFDDG algoritma z enim ali več FPGA vezji z uporabo metode simuliranega ohlajanja. 4. sredstva in metodo za oceno časovnih zakasnitev. Ta metoda omogoča analizo delovanja za določeno implementacijo. Dobljeni rezultat : sredstva (vrata, IO) in oceno latentnosti uporabimo pri koraku optimizacije za odločanje, katera implementacija spoštuje omejitve (implementacija v realnem času, ki minimizira uporabo sredstev). 5. rezultate implementacije algoritma matričnega produkta na Xilinx Multi FPGA vezje in programsko orodje SynDex s katerim je izvedena AAA metodologija.

1. Introduction

As the size and complexity of high performance signal, image and control processing algorithms is increasing continuously, the implementations cost of such algorithms is becoming an important factor. This paper addresses this issue and presents an efficient rapid prototyping methodology to implement such complex algorithms using reconfigurable hardware. The proposed methodology is based on an unified model of conditioned factorized data dependence graphs where both data and control flow are represented as well to specify the application algorithm, than to deduce the possible implementations onto reconfigurable hardware, in terms of graphs transformations. This work is part of the AAA methodology and has been implemented in SynDEx (CAD software tool that support AAA, a system level CAD software tool.

To fulfill the ever increasing requirements of embedded real-time applications, system designers usually require mixed implementation that blends different types of programmable components (RISC or CISC processors, DSP,..)corresponding to software implementation, with specific non-programmable components (ASIC, FPGA,...) corresponding to hardware implementation.

This makes the implementation task a complicated and challenging problem, which implies a strong need for sophisticated CAD tools based on efficient system-level design methodologies to cope with these difficulties and so to simplify the implementation task from the specification to the final prototype.

In this field, several system-level design methodologies and their associated tools have been suggested during the last years. SPADE /1/ methodology enables modelling and

exploration of signal heterogeneous processing systems. The result is the definition of a heterogeneous architecture able to execute these applications with respect real-time constraints. SPARK /2/ is high-level synthesis framework that provides a number of code transformations techniques. The back-end of the SPARK system generates synthesizable RTL VHDL (control synthesis is a finite sate machine controller). GRAPE-II /3/ is a system-level development environment for specifying, compiling, debugging, simulating and emulating digital-signal processing applications on heterogeneous target platforms consisting of DSPs and FPGAs. After specification, resources requirement, mapping architecture, the last phase generates C or VHDL code for each of the processing devices. POLIS /4/ system implements a HW/SW codesign using the CFSM (the Codesign Finite State Machine formal model). A complete codesign environment, based on POLIS system, which combines automatic partitioning and reuse of a module database is presented in polis. The SPARCS design system /5/ is an integrated design environment for automatically partitioning and synthesizing behavioural specifications (in the form of task graphs) on multi-FPGA architecture. The SPARCS contains a temporal partitioning tool to temporally divide and schedule the tasks on the architecture and high-level synthesis tool to synthesize register_transfer level designs for each set of tasks.

Each of the above tools has its own features (for example several models can be used for application and architecture specification) and innovative aspects but none of them support the entire implementation process onto mixed architecture using an unified model as well to specify the application algorithm, as to deduce the possible implementation onto multicomponent architecture.

To achieve this goal, we have developed, in the one hand, the AAA (Algorithm-Architecture Adequation) rapid prototyping methodology /6/ which helps the real-time application designer to obtain rapidly an efficient implementation of his application algorithm onto his heterogeneous multiprocessor architecture and to generate automatically the corresponding distributed executive /7/. This methodology uses an unified model of graphs as well to modelize the application algorithm, the available architecture as to deduce the implementation which is formalized in terms of transformations applied on the previous graphs. In the other hand we aim to extend our AAA methodology to the hardware implementation onto specific integrated circuits in order to finally provide a methodology allowing to automate the implementation of complex application onto multicomponent architecture using an unified approach.

This paper presents the design methodology based upon graph transformation from algorithm specification to hardware implementation. This methodology automates the hardware implementation of an application algorithm specified as a Conditioned Factorized Data Dependence graph in the case of reconfigurable integrated circuits (FPGA). This methodology is illustrated through all the sections with a conditioned matrix-vector product case study that involve a moderately complex control flow involving both conditioning and loops. We first present the conditioned factorized data dependence graph model proposed to specify the application algorithm in section 2. In section 3 we present the implementation model describing the result obtained by applying a set of rules that allows to automate the synthesis of data and control paths from the algorithm specification. Following that, the principles of optimization by defactorization are described in section 4. In this section we present the using of the simulated annealing technique to obtain an optimized implementation on mono and multi circuit architecture. The proposed algorithms guided by the cost functions find the best solution that respects the real time constraint while minimizing the resources consumption.

2. AAA methodology: Algorithm model

According to the AAA methodology, the algorithm model is an extension of the directed data dependence graph, where each node models an operation (more or less complex, e.g. an addition or a filter), and each oriented hyperedge models a data dependence, where the data produced as output of a node is used as input of an other node or several other nodes (data diffusion). The set of data dependences defines a partial order relation on the execution of the operations, which may be interpreted as a "potential parallelism".

This extended data dependence graph, called Conditioned Factorized Data Dependence Graph (CFDDG) allows to specify loops through factorization nodes, and conditioned operations (operation executed, or not, depending on its conditioning input) through conditioning edge. In this CFDDG graph, each oriented dependence edge is either a data dependence or a conditioning dependence, and each node is either a computation operation, an input-output operation, a factorization operation or a selection operation.

This algorithm graph may be specified directly by the user using the graphical or textual interface of the SynDEx software /7/ or it may be generated by the compiler from high level specification languages, such as the synchronous languages, which perform formal verifications in terms of events ordering in order to reject specifications including deadlocks /8/.

2.1 Conditioned Factorized Data Dependence Graph

Typically an algorithm specification based on data dependence contains regular parts (repetitive subgraph) and non-regular parts. As described in /9/, these spatial repetitions of operation patterns (identical operations that operate on different data) are usually reduced by a factorization process to reduce the size of the specification and to highlight its regular parts. Graph factorization consists in replacing a repeated pattern, i.e. a subgraph, by only one

instance of the pattern, and in marking each edge crossing the pattern frontier with a special "factorization" node, and the factorization frontier itself by a doted line crossing these nodes. The type of factorization nodes depends on the way the data are managed when crossing a factorization frontier: 1. A Fork ***F*** node factorizes array partition in as many subarrays as repetitions of the pattern. 2. A Join ***J*** node factorizes array composition from results of each repetition of the pattern. 3. A Diffusion ***D*** node factorizes diffusion of a data to all repetitions of the pattern. 4. An Iterate ***I*** node factorizes inter-pattern data dependence between iterations of the pattern.

Moreover, the user may want to specify that some operations will be executed depending on some condition. In our CFDDG model, we provide a conditioning process such that the execution of operations of the algorithm graph may be conditioned by a conditioning dependence, which is represented on the algorithm graph by a dashed edge. In this case, the conditioned operation is executed only if its inputs data are present and its condition of activation is satisfied. In order to indicate the end of the conditioned sub-graph in the algorithm graph that corresponds to the 'EndIf' of the typical control primitive IF-THEN-ELSE, we need a specific node 'select'. It allows to select among the data it receive the one that will be sent to its output. The input data of a select node correspond to the data produced by the conditioned operations with their condition of activation satisfied. As the parallel execution of these conditioned operations, that are not necessarily exclusive, can lead to simultaneous presence of several input data at the select node, we introduced priorities between its data which will be specified in an explicit way with labels on the input edges (p_1, p_2, ...,p_n). The input data having the highest priority p_i will be selected and sent to its output.

2.2 Specification of Conditioned Matrix-Vector by using Conditioned Factorized Data Dependence Graph

Figure 1 represents use a Conditioned Matrix-Vector Product example (C-MVP) specifying by CFDDG model: depending on the value of the input data C, this algorithm will compute either the product of the matrix $M \in R^m \times R^n$ by the vector $V \in R^n$ and will return the resulting vector or will directly return the input vector V. The computation of the product of the matrix M (composed of m vectors M_i: $M=(M_i)_{1\le i\le m}$ by the vector V can be decomposed into m scalar products $PS=(M_iV)_{1\le i\le m}$ (loop for i) each PS can then be decomposed into a sum of n products $M_iV=M_{i1}V_1+.....+M_{in}V_n$ (loop for j).

This decomposition process generates repetitions of operations patterns; that we often prefer to specify in a factorized form as described in Fig.1. Therefore, the final conditioned Factorized Data Dependence Graph (CFDDG) will include the two imbricated frontiers FF2 and FF3 corresponding to the two imbricated for loops, in addition to the factorization frontier FF1 which correspond to the factorization of the infinitely repeated pattern of the graph since we deal with reactive applications that interact infinitely with the physical environment.

3. AAA methodology: Implementation model

Implementing applications onto specific integrated circuits requires system designers to generate the data path responsible for the core of the computation as well as the control path to provide the appropriate control signals for the computations. The resulting RTL design containing both data and control paths is then characterized in order to estimate time and area performance. This allows the exploration of different hardware implementations, seeking for an ideal compromise between the area and the response time of the circuit.

Then, we propose a seamless flow based on graph transformation to transform the algorithm graph into an implementation graph containing both data-path graph and control-path graph. As will see, data-path transformations are quite simple, but control-path transformations are not trivial and require to build first a neighborhood graph.

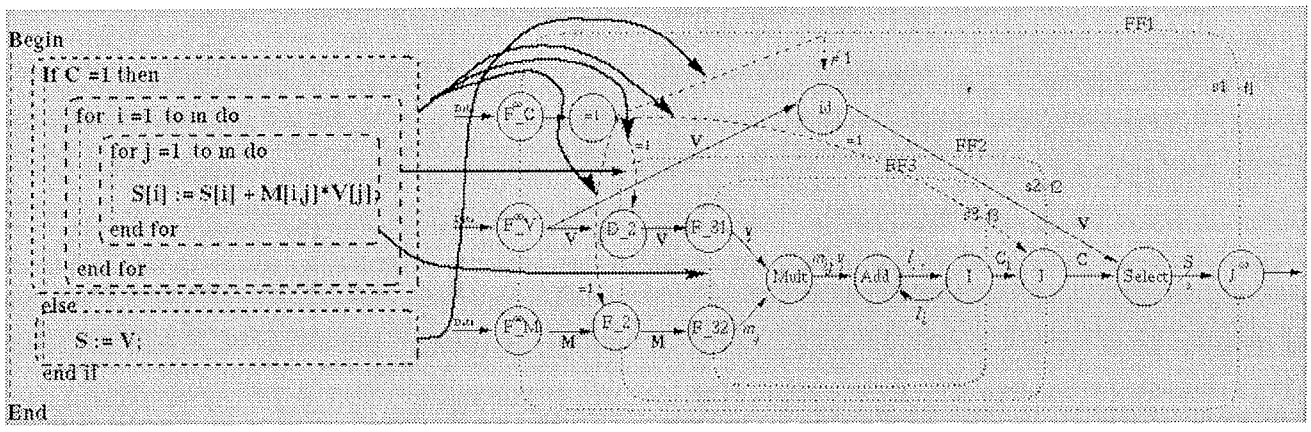

Figure 1: Conditioned and Factorized Data Dependence Graph of C-MVP

3.1 Neighborhood graph

Every factorization frontier may be a consumer (located downstream) or/and a producer (located upstream) relatively to another frontier according to the data dependences relating them. Two frontiers are neighbors if there is at least one relation of direct dependence that does not cross a third frontier. Based on these neighborhood relations, we build a neighborhood graph. The nodes of such graph represent the factorization frontiers and the oriented edges represent the data flow between factorization frontiers.

The edge orientation describes the consumption/production relation: an edge starts at a producer and ends at a consumer. As producing/consuming frontiers may be them selves conditioned (e.g. FF2 on Fig.1), data production/consumption between frontiers are consequently conditioned. To take into account such conditioned data flow, we will represent conditioned consumption/production by dashed edges.

In the case of a sequential implementation of factorization nodes, every factorization frontier, called FF, separates two regions, the first one called "fast" (f), being repeated relatively to the second one, called "slow" (s). These slow and fast sides of a frontier are due to the difference of the data transfer rate on each side of the frontier. Every node of the neighborhood graph is then subdivided in four parts (slow-downstream, fast-upstream, fast-downstream and slow-upstream) /10/. The neighborhood graph is deduced automatically from the CFDDG and it is used during the implementation in order to establish the control relationships between frontiers leading to a part of the control-path.

The neighborhood graph built from the CFDDG specifying the C-MVP algorithm (Fig.1) comprises three nodes corresponding to the three factorization frontiers FF1, FF2, FF3. The factorization frontier FF1 is infinite, it does not have neighbors on its "slow" side which corresponds to the physical environment. FF1 is, either a producer compared to the conditioned frontier FF2 or a producer to itself and a consumer compared either to itself or to the conditioned frontier FF2. FF2 is also a producer and a consumer compared to FF3. FF3 is a producer and a consumer, compared to itself through the arithmetic operations mul and add.

3.2 Data-path graph generation

The hardware implementation of the Conditioned Factorized Data Dependence Graph consists in providing a matching operator for every node, and a matching connection between operators for each data dependence edge relating the corresponding operations. The resulting graph of operators and their interconnections compose the data path of the circuit. This hardware translation process defines then a graph isomorphism between Conditioned Factorized Data Dependence Graph and the data path graph.

The matching operator node is a logic function in the case of a computation operation node, or it is composed of a multiplexer and/or registers in the case of a factorization node (i.e: ***F***, ***I*** and ***J*** nodes) or it is composed of a priority encoder and a multiplexer for the select node to encode priority and to select data.

3.3 Control-path graph generation

The control path corresponds to the logic functions that must be added to the datapath, in order to control the multiplexers and the transitions of the registers composing the operators. It is then obtained by data transfer synchronization between registers. However, two conditions must be satisfied to allow a register to change state: the new upstream data to the register must be stable, and all downstream consumers of the register must have finished the utilization of previous data. Moreover, if upstream data of a circuit comes from various producers with different propagation time, it is necessary to use a synchronized data transfer process. This synchronization is possible through the use of a request/acknowledge communication protocol. Consequently, the synchronization of the circuit implementing the whole algorithm is reduced to the synchronization of the request/acknowledge signals of the set of factorization operators.

These operators are gathered in factorization frontier and their data consumption and production are done in a synchronous way at the level of the frontier. We propose then a local control system where each factorization frontier will have its own control unit.

This delocalized control approach allows the CAD tools used for the synthesis to place the control units closer to the operators to control rather then a centralized control approach.

Control units and their interconnections: As mentioned above (section 3.1), each factorization frontier has upstream and downstream relations on both sides, "slow" and "fast". The relations between upstream/downstream and request/acknowledge signals on both sides of a frontier are implemented by the "control unit" of the factorization frontier. This control unit contains a counter C with d states (corresponding to the d factorized repetitions) and an additional logic function in order to generate, in the one hand the communication protocol between frontiers (the slow/fast, request/acknowledge signals at the upstream and downstream sides), and in the other hand the counter value (cpt) and the enable signal (en), that control the frontier operators.

Thus, the control path will mainly be composed of the set of control units associated to the corresponding frontiers nodes of the neighborhood graph. These control units are then inter-connected in a systematic way as follows: for each oriented dependence edge, we generate a request signal transmitted between the corresponding control units. And for each generated request signal, the associated acknowledge signal is transmitted, in the opposite direc-

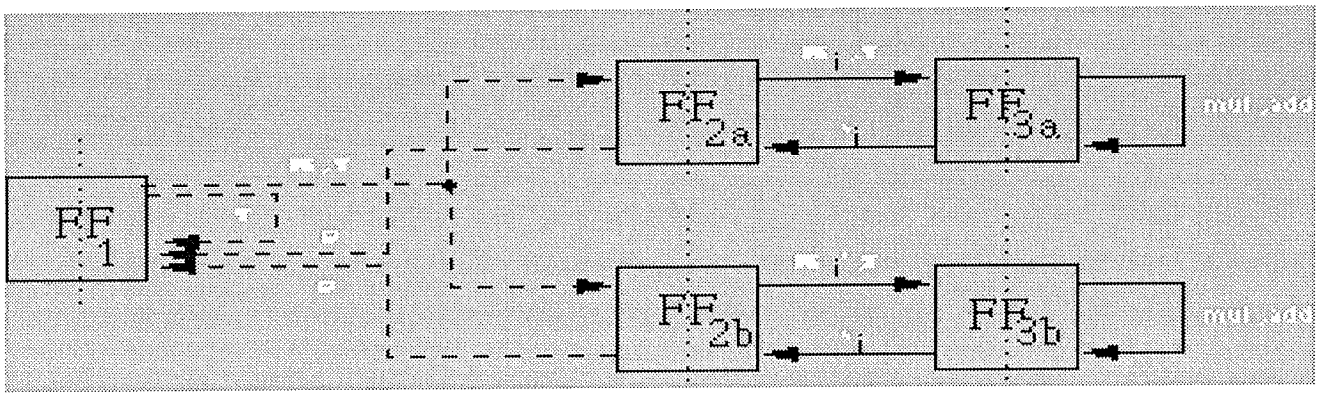

Figure 2: Neighborhood graph of defactorized C-MVP (see paragraph 2.2)

tion. When several signals occur at the same input of a control unit, the conjunction of these signals is performed by a logical AND gate. Note that, the generated request signals associated to conditioned dependences must first be send to a multiplexer controlled by a priority encoder which will send in turn the request signal with the highest priority to its output /11/.

4. AAA methodology: Implementation optimization

4.1 Optimization principle based upon defactorization process

If the implementation of the factorized specification onto an application specific integrated circuit or an FPGA does not meet the real time constraints, we need to defactorize the implementation graph corresponding to the specification. The defactorization process is the reverse transformation of the factorization and therefore it does not change the operational semantic of the data dependence graph. The goal is to obtain a more parallel implementation in order to reduce the latency and improve the temporal performances in spite of increasing hardware resources. Thus the optimized implementation of a conditioned factorized algorithm graph onto the target architecture is formalized in terms of graph defactorization transformation.

Figure 3 represents a defactorized by a factor 2 of C-MVP graph. Defactorized solution allows to reduce the latency of the implementation, but increase the number of required hardware resources. FF2 is defactorized in two frontiers FF2a and FF2b, and FF3 is then duplicated in FF3a and FF3b.

The implementation space which must be explored in order to find the best solution is then composed of all the possible defactorizations of a factorized graph specifying the algorithm. For instance, for a given algorithm graph with n frontiers, we have at least 2^n defactorized implementations. Moreover, each frontier can be partially defactorized: a factorization frontier of *r* repetitions can be decomposed in *f* factorization frontiers of *r/f* repetitions. Consequently, for a given algorithm graph, there is a large, but finite, number of possible implementations which are more or less defactorized, and among which we need to select the most efficient one, i.e. which satisfies the real-time constraints (upper bound on latency), and which uses as less as possible the hardware resources, logic gates for ASIC and number of Configurable Logic Blocks CLB for FPGA. This optimization problem is known to be NP-hard, and its size is usually huge for realistic applications. This is why we use heuristic guided by a cost function, in order to compare the performances of different defactorizations of the specification. This heuristic allows us to explore only a small subset of all the possible defactorizations into the implementation space. The heuristic needs to define a cost function based on the critical path length metric of the implementation graph: it takes into account both the latency and the resources consumption of the implementation which are obtained by a preliminary step of characterization.

4.2 Architecture characterization: area and latency estimation

To estimate the total area we use the neighborhood graph to calculate the data path area and control path area. This total area used by the implementation is given by:

$$S_t(FF,FG)=\sum_{i=1}^{n} D_{path}(f_i)+\sum_{i=1}^{n} S_{Cpath}(f_i) \qquad (1)$$

Where f_i designs the frontier i.

The total area is calculated in term of FF (Flip Flop) and FG (Function Generator). One can use equation 2 to deduce the area in terms of CLB (Control Logic Block):

$$S_t(CLB)=\frac{\max(S_t(FF),S_t(FG)}{2} \qquad (2)$$

For the calculation of latency, one deduces from neighbourhood graph the various relations between the frontiers (frontiers in series, parallel, inclusive). These relations enable us to determine the number of cycles, thus the number of cycles multiplied by the time cycle gives the execution time of the algorithm.

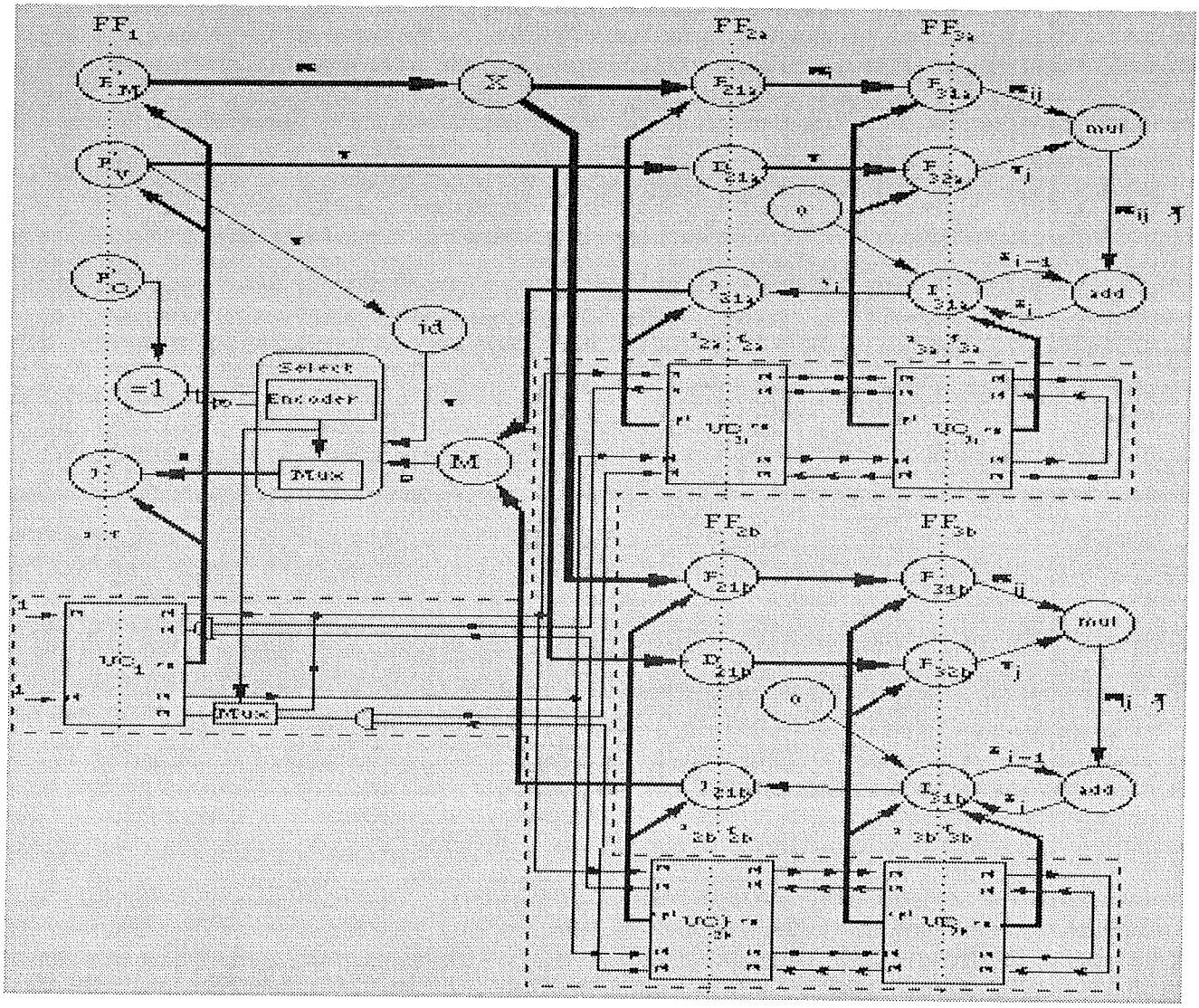

Figure 3: A defactorized implementation graph of C-MVP

4.3 Algorithm of the proposed Simulated annealing heuristic for mono-FPGA architecture

4.3.1 Simulated annealing principle

This is a technique /12/ for solving combinatorial optimisation problems such as minimising functions of many variables. The conventional ANNEALING algorithm operates by starting with an initial solution to the combinatorial optimization problem and improving the solution through a series of changes. This involves finding some configuration of parameters that minimises some objective function. It is based on a modification of iterative improvement, which involves starting with some existing suboptimal configuration and perturbing it in some small way. If this new configuration is better than the old one the new configuration is accepted and we start again. Unlike a greedy algorithm which only accepts system configuration resulting from better changes, annealing probabilistically accepts inferior changes.

4.3.2 Application the simulated annealing technique in the AAA methodology

The application of simulated annealing consists in finding the global minimum (i.e. the optimized implementation graph) of an objective function by avoiding its local minima (metastable states of the system). This function has two variables: resources used for the hardware implementation and execution time of the algorithm.

Let a vector containing a set of variables, each one of these variables indicates the state of each component of the system. In our case the system is represented by the CFGDD of the algorithm to be implemented, each frontier of this graph defines a component of the system and a variable X defines the defactorization factor. The defactorization process implies a change of the system. For example: if the V1 vector = {X1=3, X2=1, X=3,X4=2) changes into in V1 = {X1=3, X2=2, X3=3, X4=2), the new state of V1 means that the frontier which is defined by X2 was defactorized by 2. As we described in paragraph 4.1, this defactorization process generates an increasing/diminution of the

execution time of the algorithm (i.e. latency of the circuit) and thus a diminution/increasing of the surface of the circuit. We defined a cost function F(X) for a given state X of the system. This function estimates the variations of the system for a given state X. It permits to choose the defactorization which satisfies the time constraint while minimizing the resources consumption (i.e. CLB in case FPGA):

$$F(X)=\begin{cases} S & \text{if } t \prec T \\ S+k(t-T) & \text{if } t \geq T \end{cases} \quad (3)$$

Where:

- T indicates the time constraint,
- t is the execution time for a given defactorization,
- S is the surface of the hardware implementation for a given defactorization,
- k defines the penalty factor.

Based on the technique of simulated annealing, the algorithm that we propose starts with the calculation of the control parameter C_0, then one starts by gradually decreasing his value and for each one of these values one carries out a certain number of changes of states of the system (defactorization). With each reduction of the control parameter of control, one increases the number of changes of states of the system by a factor β.

In the algorithm below:

- X_0 presents the initial solution of the system,
- L_0 is the initial value of a number of changes of system state,
- *iter* defines the number of change of a control parameter,
- L_k is the modification number of system state for a given control parameter c_k,
- X_i is a given solution or given state vector of the system,
- X_j is state vector after a defactorization process, it is a solution close to X_i,
- c_k is indicates the value of a control parameter during the kth iteration, is the initial value of a control parameter,
- *F(X)* indicates the cost function for a system state X.

Algorithm: simulated annealing for mono circuit architecture

```
1. initialize (c0,L0,X0 )
2. for iter =1 to num_iter do
3.     for n-modif to Lk do
4.         Xj=V(Xi)
5.         if (F(Xi)≺F(Xj)) then Xi=Xj
6.         else
7.             if(exp(-(F(Xj)-F(Xi))/
               c)≻Random.float(0,1)) then Xi=Xj
8.             endif
9.         endif
10.    endfor
11. CK+1=CK*α
12. LK+1=LK*β
13. endfor
```

The simulated annealing algorithm returns the optimized implementation graph which satisfies real-time constraints and uses as less as possible the hardware resources. The algorithm begins by performing the initial value of X_0, L_0 and c_0 (c_0 is equal to 2*n, where n defines the number of defactorizable frontiers) and then decreases gradually a control parameter. For each control parameter, the state system is modified and for each control parameter reduction, the number of the system change is increased by β. One chooses β equal to the edges number of the CFDDG. The algorithm must find and accepts the solution close to X_i if it is not possible another solution is accepted with certain probability.

4.4 Algorithm of the proposed Simulated annealing heuristic for Multi-FPGA architecture

Let the number of frontiers composing the graph algorithm CFGDD, each frontier border of this graph contains a set of edge and has a factor of factorization.

A X vector defines the different states of the frontier defactorization. Let FF2 frontier which is defined by X2 variable. If FF2 frontier has a factor defactorization equal to 3 then X2 contains three elements: X21, X22 and X23. Each of these elements corresponds to a defactorization state. For

$$F(X)=\begin{cases} S_{tot}+I/O_{tot} & if \ t_{tot}<T_{contr} \ and \ \forall S_i<S_{contr} \ and \ \forall I/O_i<I/O_{contr} \\ S_{tot}+I/O_{tot}+k(t_{tot}-T_{contr}) & if \ t_{tot}\geq T_{contr} \ and \ \forall S_i<S_{contr} \ and \ \forall I/O_i<I/O_{contr} \\ S_{tot}+g.(I/O_i-I/O_{contr})+k(t_{tot}-T_{contr}) & if \ t_{tot}\geq T_{contr} \ and \ \forall S_i<S_{contr} \ and \ \exists I/O_i\geq I/O_{contr} \\ l(S_i-S_{contr})+I/O_{tot}+k(t_{tot}-T_{contr}) & if \ t_{tot}\geq T_{contr} \ and \ \exists S_i\geq S_{contr} \ and \ \forall I/O_i<I/O_{contr} \\ l(S_i-S_{contr})+g(I/O_i-I/O_{contr})+k(t_{tot}-T_{contr}) & if \ t_{tot}\geq T_{contr} \ and \ \exists S_i\geq S_{contr} \ and \ \exists I/O_i\geq I/O_{contr} \\ l(S_i-S_{contr})+g(I/O_i-I/O_{contr}) & if \ t_{tot}<T_{contr} \ and \ \exists S_i\geq S_{contr} \ and \ \exists I/O_i\geq I/O_{contr} \\ l(S_i-S_{contr})+I/O_{tot} & if \ t_{tot}<T_{contr} \ and \ \exists S_i\geq S_{contr} \ and \ \forall I/O_i<I/O_{contr} \\ S_{tot}+g(I/O_{tot}-I/O_{contr}) & if \ t_{tot}<T_{contr} \ and \ \forall S_i<S_{contr} \ and \ \exists I/O_i\geq I/O_{contr} \end{cases} \quad (4)$$

example, if one defactorises FF2 by 2 then X22 is equal to 2 and X21 and X23 are equal to 0. X22 contains a set of frontiers which their edges.

To each frontier, one associates in random way a given circuit of the architecture by applying the simulated annealing method. In order to estimate the partionning we defined a cost function:

Where :

- t_{tot} is the execution time of the algorithm after applying defactorization,
- S_{tot} is the area occupied by all the circuits after the defactorization process
- S_i defines the area used in the circuit i,
- k,l,g are the penalty coefficients (varying from 10 to 100) used if respectively the time, area and I/O constraint are not respected,
- T_{contr} is the time constraint,
- S_{contr} and I/O_{contr} define respectively the time and I/O constraints for every implementation,
- I/O_{tot} is the total number of the I/O used for all the circuits used by a given implementation,
- I/O_i is the I/O number for the circuit *i*.

The proposed algorithm is similar than the mono circuit case, we only change the cost function. This algorithm takes the neighbourhood graph as an input and returns the partionning graph. This partinoning graph corresponds to muli-circuit implementation which respects the real time constraints and uses as less possible the hardware resources (i.e. area and I/O).

We applied this algorithm to an example which describes a five Conditioned Product Matrix Vector calculation (the CFDD graph contains 60 edges). The hardware implementation uses two circuits: Xilinx XC4013E/X2. The parameters bellow defines the main characteristics of XC 4012 E/ X2 circuit:

Device	Logic Cells	Max Logic Gates	Max RAM bits	Typical Gate Range	CLB Matrix	Total CLBs	Number of Flip - Flops	Max User I/O
XC4013E/X2	1,368	13,000	18,432	10,000 - 30,000	24x24	576	1,536	192

To compare the optimisation results, we applied the simulated annealing algorithm for 5 different time constraints: 1000, 1500, 2000, 2500 and 3000 ns. The table below presents the results for each time constraint. The hardware implementation needs two circuits: circuit 1 and circuit 2. For each circuit table 1 gives the CLB and I/O used at different time constraint.

5. Conclusion

We showed that from an application algorithm specified with a conditioned factorized data dependence graph it is possible to obtain a hardware implementation onto a reconfigurable integrated circuit following a set of graphs transformations, leading to a seamless design flow. These transformations allow to automatically generate the data-path and the control-path for designs with moderately complex control flow involving both conditioning and loops. The proposed delocalized control approach allows the CAD tools used for the synthesis to place the control units closer to the operators to control. We have presented an optimization heuristic based upon simulated annealing and we applied this technique for Conditioned and factorized Data Dependence Graph by using a defactorization process guided by cost function. We defined two cost functions for mono and multi Circuit architectures. We used these two algorithms to obtain automati-

TIME Constraint [ns]	1000	1500	2000	2500	3000
circuit 1 CLB used	60	433	298	202	306
Circuit 2 CLB used	458	114	210	317	172
circuit 1 I /O used	34	30	41	35	43
Circuit 2 I/O used	37	27	39	30	47
Total latency [ns]	397	1231	985	1280	1231
Total CLB	518	547	508	519	478
Total I/O	71	57	80	65	90

Table 1: number of circuit, CLB and I/O used at different time constraint.

cally the best solution at the given time constraint. The optimization heuristics will address both defactorization and partitioning issues. Moreover, this extension of the AAA methodology to the hardware implementation of algorithm onto integrated circuit, provides a global methodology in order to tackle complex hardware/software co-design problems involved by multicomponent architecture.

6. References

/1/ P. Lieverse, P. van detr Wolf, Ed Deprettere, K. Vissers, "A Methodology for architecture exploration of heterogeneous signal processing systems", Proc. 1999 IEEE Workshop on Signal Processing Systems.

/2/ S. Gupta, N. Dutt, R. Gupta, A. Nicolau, "SPARK, High-Level Synthesis Framework For Applying Parallelizing Compiler Transformations", 7th Intl.Conference on VLSI design, January 5-9, 2004, Mumbai, India.

/3/ R. lauwereins, M. Engels, M. Adé, J. Peperstraete, "Grape-II : A system-level Prototyping Environment For DSP applications", IEEE Computer, Vol. 28,No 2, pp. 35-43, Feb. 1995.

/4/ S. Edwards, L. Lavagno, E.A. Lee, A. Sangiovanni-Vincentelli, "Design of embedded systems: formal models, validation, and synthesis", Proc. of IEEE, v.85, n.3, March 1997.

/5/ M. Kaul , R. Venuri, "Optimal Temporal Partitioning and Synthesis for Reconfigurable Architectures", Design, Automation and Test in Europe, February 1998.

/6/ T. Grandpierre, C. Lavarenne, Y. Sorel, "Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors", CODES'99 7th Intl. Workshop on Hardware/Software Co-Design, Rome, May 1999.

/7/ T. Grandpierre, Y. Sorel, "From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations", MEMOCODE03, Intl. Conference on Formal Methods and Models for Codesign, Mont Saint-Michel, France, June 2003.

/8/ N. Halbwachs, "Synchronous programing of reactive systems", Kluwer Academic Publishers}, Dordrecht Boston, 1993.

/9/ L.Kaouane, M. Akil, Y. Sorel, T. Grandpierre, "From algorithm graph specification to automatic synthesis of FPGA circuit: a seamless flow of graphs transformations", FPL'03, Intl.Confrence on Field Programmable Logic and Applications}, Lisbon, Portugal, September 1-3, 2003.

/10/ L.Kaouane, M. Akil, Y. Sorel, T. Grandpierre, "A methodology to implement real-time applications on reconfigurable circuits", ERSA'03, Intl. Conference on Engineering of Reconfigurable Systems and Algorithms}, Las vegas, USA, June 2003.

/11/ R. Vodisek, M. Akil, S.Gailhard, A.Zemva, "Automatic Generation of VHDL code for SynDEx v6 software", Electro technical and Computer Science conference}, Portoroz, Slovenia, September 2001.

/12/ S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, „Optimization by Simulated Annealing", Science, 220(4598): 671-680, May 1983.

Mohamed Akil
Laboratoire A2SI, Groupe ESIEE
Cité Descartes, 2 Bld Blaise Pascal - BP 99
93162 Noisy Le Grand cedex, France
akilm@esiee.fr

Prispelo (Arrived): 15.09.2003 *Sprejeto (Accepted): 03.10.2003*