

A Dynamically Reconfigurable Processor for Dataflow Graph Execution

Lorenzo Verdoscia

Research Center on Parallel Computing and Supercomputers - CNR

Via Castellino, 111

80131 Napoli - Italy

verdoscia.l@cps.na.cnr.it

ABSTRACT

This paper presents a dynamically reconfigurable processor for dataflow graph execution. It constitutes the core for CODACS (COConfigurable DATAflow Computing System) prototype, a project under development at Research Center on Parallel Computing and Supercomputers (CPS) of the National Research Council-Italy. The objective of this project is to build up a high performance reconfigurable computing system prototype providing highly scalable architecture and transparent hardware reconfiguration for the node processor. Main characteristics of this processor are: a) it is constituted by a set of identical Elemental Computing Elements (ECE) able to directly execute any elemental operation defined in CHIARA [14], an FP-like functional programming language; b) it executes dataflow graphs without using memory to store partial results when tokens flow from an ECE to another; c) graph execution happens in a completely asynchronous manner; d) no control tokens are generated during this computation.

Thanks to its particular architecture, such processor can also execute pipeline operations without any hardware cost.

Keywords

Dataflow, FPGA, reconfigurable computing

1. INTRODUCTION

FPGA (Field Programmable Gate Array) components are large, fast integrated circuits which can be modified or configured, almost any point by the end user. An FPGA can be programmable at the gate level thank to its spatial connection of general purpose computing elements (cells). In a custom FPGA based computing machine, the execution core is composed of an array of cells interconnected by reconfigurable structures such as buses and switch matrices. This feature, that has introduced a new computing paradigm [12], provides a straightforward and powerful development tool to design reconfigurable systems. Furthermore, treating such

components as coprocessors, several researchers are investigating the effectiveness of combining reconfigurable hardware with general purpose processor for word-level, computation intensive applications [21, 24, 23, 7, 19].

Reconfigurable Computing paradigm, applied to design of general purpose processors, seems to be a valid execution platform in prototyping programmable HPC systems employing thousand reconfigurable processing units. In particular, the dataflow execution model is promising when applied to this platform because of its fine grain nature. In fact, instead of carrying out in sequence a set of operations like a standard processor does, the dataflow execution model calculates a function connecting and configuring a number of actors (macrocells) as a dataflow graph.

However, despite the model simplicity, in the past technology limits drove the development of general purpose dataflow architectures based on the *search mode* approach (dynamic or tagged token architectures) respect to *interconnect mode* ones (static architectures) [2]. But, the proposed dataflow architectures did not give the longed for results [25] due to difficulty to efficiently manage and directly execute dataflow graphs in hardware. Consequently, dataflow nodes became more and more like processing elements supporting the von Neumann model [22] because the real problem resided just in the model actor heterogeneity. So, to overcome these difficulties, we introduced a new dataflow model [15]. In Appendix we show the difference between the classical model and ours.

Although the past result have generated a general scepticism about these machines, we believe that dataflow model is still a valid proposal to increase performance and seriously attack the von Neumann model at least at processor level.

Why a new processor? There are at least four reasons that have motivated this proposal and consequently driven the design process: the first is the demand to directly map in hardware dataflow graphs in dataflow mode [3]; the second is the need to dispose of a straightforward data flow control and actor firing mechanism at a minimal hardware cost; the third is the requirement to reduce the continual LOAD and STORE operations and augment performance; the last is the possibility to adopt primitive functions of a functional language as assembler language for a processor.

The intent of the CODACS processor implementation is to

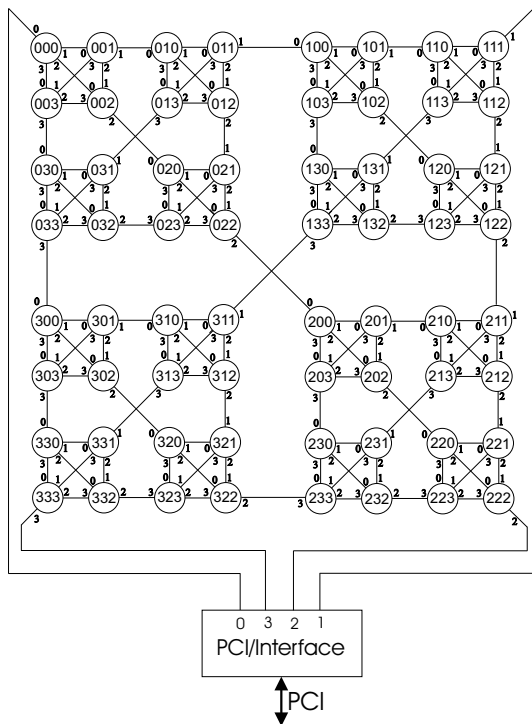


Figure 1: CODACS Architecture connected as WK-recursive with $N_d = 4$ and $Level = 3$.

study the management of dataflow graphs directly into VLSI to satisfy the increasing demand for low cost stream or frame data processing needed for important application classes, like video and image processing, multimedia, digital signal processing, and data encryption. Thanks to the flexibility offered by FPGA components, we are making a processor able to execute static dataflow graphs mapping them directly in hardware. Furthermore, thanks to its internal architecture, such processor can execute also pipelined operations.

2. CODACS ARCHITECTURE

CODACS general architecture, shown in Figure 1, has a direct network architecture organized as a set of identical nodes regularly connected by communication paths. Each node produces, consumes, and routes messages that are delivered to the destination node through intermediate nodes. The choice of the interconnect topology has been oriented towards realizing a processor network without boundaries to deal with problems requiring more than the available hardware resources, in terms of overall computational tasks partitioning and distributing into available processors. Among direct network topologies we chose a WK-Recursive one [8, 9], a tradeoff between wiring and performance [20].

From the communication point of view, in a direct network there exists an efficiency problem because each node must route, in parallel with its activities, packets directed to other computing nodes. These packets rapidly produce node overhead in an architecture that scales to thousands of nodes. The solution to the above mentioned problem requires a suitable hardware support for a fast and parallel execution

of dataflow graphs and able to provide kernel and routing functions.

The programming language for CODACS is CHIARA [14], an FP-like language. The FP style [13] was chosen instead of lambda style (like LISP) since the former produces new functions applying the metacomposition rule to more elemental functions. CHIARA peculiarity is that the elemental operator set ADD, SUB, MULT, DIV, LPS, EQ, NEQ, GE, GT, LE, LT, AND, OR, and NOT is complete (able to generate any other more complex function) and implemented directly in hardware. Therefore, CHIARA constitutes not only CODACS high level programming language, but also the low level programming language (assembly) for the dataflow processor. After compiled, a program is translated in two tables: the *graph description table* and the *input token value table*. The graph description table contains all information needed to map the dataflow graph onto the processor. For each Elemental Computing Element (ECE) of the processor, this table details its operation code, to which ECE it will send its result, and from it will receive its right and left token. The input token value table details which ECE will receive the starting tokens (the program initial and constant values) to activate the computation. Since these two tables split data and instructions connection, the graph description table loading can take place at a different time from the input token value one. Consequently, it is possible to overlap loading operations and processor activities.

2.1 Node Functional Description

To avoid a rapid overhead increase, each CODACS node, whose functional block representation is shown in Figure 2, has been partitioned into three concurrently operating subsystems:

- Routing Subsystem (RS), devoted to provide all routing functions;
- Kernel Subsystem (KS), devoted to manage processor configurations and I/O data tokens;
- Processing Subsystem (PS), devoted to execute dataflow graphs.

Routing Subsystem (RS). On the basis of the system network topology, communication among nodes can take place along four directions. This module has four input and output serial links to connect the node to its neighbors and two dedicated lines to connect the RS to the KS. The RS main task is to process incoming and outgoing messages. When a message reaches the RS, this module reads its destination and routes it towards either the appropriate output link or the KS. The RS performs the following actions:

- reads the forthcoming message;
- determines its destination;
- sends the message to the appropriate output link or (/and if required) to the KS.

The routing strategy adopted is the adaptive one described in [16].

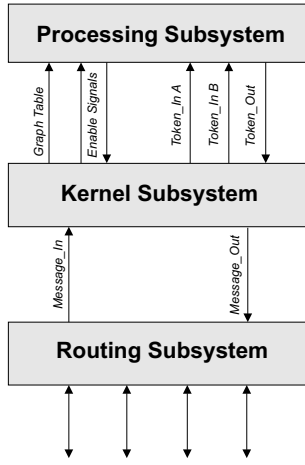


Figure 2: Node functional blocks.

Kernel Subsystem (KS). This subsystem has eight dedicated lines: two to communicate with the RS and six to communicate with the PS. We have adopted this solution to augment the throughput and overlap I/O operations towards/from the other two modules.

Once received a message, the KS unpacks it, evaluates its content, and stores the corresponding information into four buffer set: the Graph Setter List (GSL), that contains the configuration table list assigned to the processor (context assignment); the Destination List (DL), that contains the node list to which processor results will be delivered; the Input Transfer Token Environment (ITTE) constituted by two distinguish buffers for the right and left ECE tokens (Token_In A and B). Furthermore, the KS generates enabling signals for transfer operations to/from the processor.

When results are ready, the KS receives an enabling signal from the PS and performs the following actions:

- stores results into the Output Token Transfer Environment (OTTE);
- scans the destination node set from the DL and associates results to destination nodes;
- prepares new messages and transfers them to the RS for routing activities.

Processing Subsystem (PS). This subsystem is the node processor. It executes the dataflow graph assigned to that node on the basis of information received from the Kernel Subsystem. Five functional blocks constitute this subsystem: the I/O token buffer ensemble, the Control Section, the ECE Interconnect, the set of ECE, and the Graph Setter.

Once received the graph description table (configuration code) from the KS, the PS control starts configuring its computing environment (processor context), checks if the input tokens are ready in the corresponding buffers, and enables them, so the computation can take place. When the graph computation ends, the produced results are transferred to the OTTE.

After result transfer to the KR, a new context can be activated. If we do not need to change context but the processor

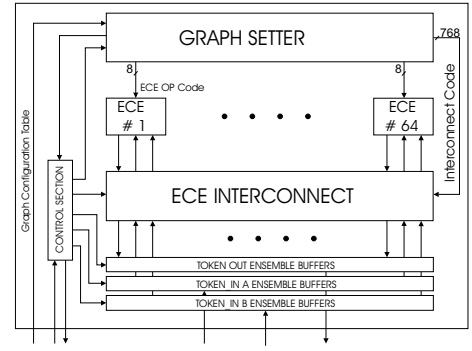


Figure 3: Dataflow Processor Architecture.

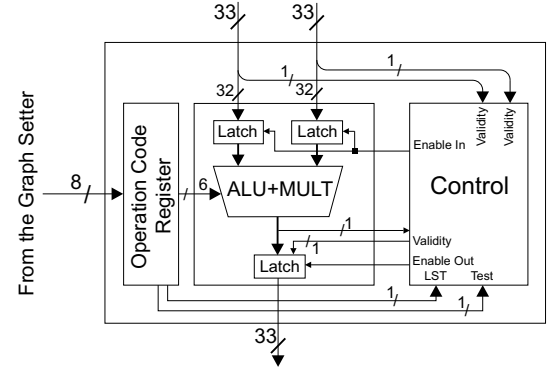


Figure 4: ECE Architecture.

only requires a new input token set (e.g. pipeline activity to compute matrix inner product), the processor control checks only for their availability.

3. THE DATAFLOW PROCESSOR

The dataflow processor is the programmable core of CO-DACS. As shown in Figure 3 it is composed of 64 identical Elemental Computing Elements (ECE), the ECE Interconnect, the Control, three groups of I/O buffers for token transfer, and the Graph Setter. From the FPGA point of view the proposed processor is static because its composition, once configured, does not change anymore. From the operational point of view it is dynamic because changing the ECE Interconnect and ECE operation codes, it can execute any dataflow graph.

ECE Architecture. As shown in Figure 4, an ECE is composed of a 32 bit fixed-point ALU-multiplier unit, an operation code register, two input and one output token latches, and a control. When a token reaches the ECE, its validity bit is sent to the control to match the partner. As soon as the match happens, that is both validity bits have the 1 value, an enabling signal activates the two input latches that catch the data tokens allowing the operation stored in the register to take place. After the delay needed to produce the result (in our case $1\mu\text{sec}$), the control generates the bit validity, enables the output token latch, and resets the two input validity bit, thus a new processing starts. The operation code register sends also two bits the control. The first informs the control if the ECE is involved in a logical

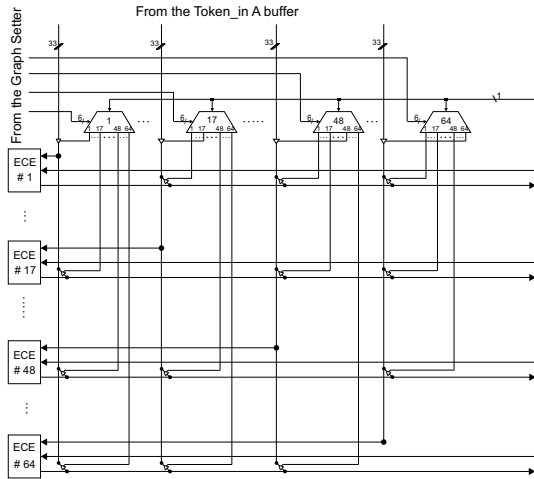


Figure 5: ECE interconnection network.

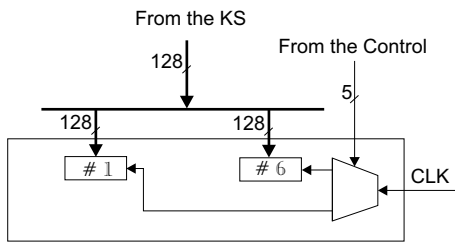


Figure 6: The interconnect code register block inside the Graph Setter.

or comparative operation (Test bit). In this case if the condition is not satisfied, the control generates the \emptyset value for the validity. The second informs the control if the ECE is involved in a loop start operation (LST bit). In this case a token virtual presence is generated.

Interconnection Network. The ECE interconnect is organized as a crossbar switch. It consists of a grid of wires, rows and columns connected by a set of three-states, and a set of decoders as shown in Figure 5. Column wires are grouped in three sets to allow tokens flow towards and from ECE and/or Token Ensemble Buffers. When each decoder receives its own code, it enables the connection between the corresponding ECE output and the ECE input devoted to receive it. Decoder control signals come directly from dedicated registers of the Graph Setter. The processor control provide for them output enabling.

Graph Setter. The graph Setter is essentially organized into two blocks that store the processor context. In one there is the interconnect code, in the other there are the ECE operation codes. Each block has a dedicated bus and is organized in groups of registers to simplify the graph description table from the KR to the PS. Figure 6 shows the group registers for the interconnection code.

4. RELATED WORKS

Even though there exist several projects that investigate new architectural proposals using FPGA computation model, CO-

DACS project is unique with respect to them because its processors, that are statically configured to implement internal block functionality (buffers, interconnect, control, and computational elements), are dynamically configured at ECE and interconnect level to execute any dataflow graph, included cycles, without changing their contexts. Among these projects, GRD chip [18], Morphosys [1], and PADDI [6] are the those ones comparable with ours.

GDR chip is specialized for neural network applications and constituted by a RISC processor and 15 PFUs (Programmable Functional Units), the DSP processors, connected in a reconfigurable network of a binary tree shape while CODACS processor does not support any classical processor to execute sequential tasks of the application, and its interconnect is organized like a crossbar.

Morphosys chip is constituted by the 8×8 RC Array, an array of Reconfigurable Cells (SIMD coprocessor), and its Context Memory, TinyRISC main processor that executes sequential tasks, and a high-bandwidth memory interface. Furthermore, it uses 2D mesh and a hierarchical bus network. In contrast, CODACS processor exhibits MIMD functionality, its interconnect is like a crossbar, and its context switch is managed by the Kernel Subsystem according to the operations to be executed (e.g., in pipeline operations, the context does not change).

PADDI chip has a VLIW nature because each EXU uses a 53-bit instruction word and employs a crossbar interconnection network. It cannot be dynamically configured since the instruction decoders are SRAMs that are loaded at setup time and does not employ any buffer to speed up I/O transfers. Finally, PADDI is a stand alone system targeting only DSP applications.

5. PERFORMANCE

To evaluate the processor goodness, we have simulated its behavior. We have used ALTERA Quartus II [4] development software and the FPGA component APEX20K15-C for the processor project. Each processor is organized as a cluster with 64 interconnected ECE. Each ECE executes operations on 32 bit integer operands while the clock speed is 133 MHz. Since the number of I/O pins for an APEX20K15-C is 808, we have used 132×3 pins for the token transfer (4 tokens at a time for Token_InA, Token_InB, and Token_Out) 128 pins for the interconnect configuration transfer, 256 pins for the ECE operation code transfer, and the others for clock and control signals.

Time evaluation. Since at the moment we have not yet realized the Kernel Subsystem, we have set an average transfer time from it towards the processor $t_a = 8$ nsec. t_a has been evaluated transferring a 66 bit pin-to-pin from a register on an FPGA to another. Instead, the execution time for an ECE has been measured $t_{ECE} = 1\mu\text{sec}$ using the ALTERA Quartus II evaluation tool. If n_{ECE} is the number of ECE in a processor, the interconnect configuration requires a number of bits:

$$N_{bit_{conf}} = 2 \times n_{ECE} \times \log_2 n_{ECE} \quad (1)$$

Table 1: IP time evaluation. They include the processor configuration and token transfer time

n	T_{MULT} μsec	T_{ADD} μsec
64	18.0	26.1
256	288.8	331.5
1024	4620.3	5158.6

while the operation codes (one byte for each of them) require a number of bits

$$Nbit_{opcd} = 8 \times n_{ECE} \quad (2)$$

and for the tokens we need

$$Nbit_{Token} = 3 \times 33 \times n_{ECE}. \quad (3)$$

Processor performance is related to the Inner Product (IP) between two matrices $A(n, n)$ and $B(n, n)$ where each element of the resulting matrix $C(n, n)$ is:

$$c_{ik} = \sum_{j=1}^n a_{ij} \times b_{jk} \quad i, k = 1 \dots n. \quad (4)$$

The corresponding CHIARA program is

```
def IP = ! + &* trans
```

and dataflow graph is a reverse binary tree where, for each c_{ik} , the first level needs n parallel multiplications and $\lceil \log_2 n \rceil$ sequential steps for the $n - 1$ sums. Table 1 shows the multiplication and sum time evaluation for some values of n .

6. CONCLUDING REMARKS

In this paper a reconfigurable processor is presented. Principal features that distinguish it from the others are: the processor is able to execute dataflow graphs, included loops, according to the static model but using only actors with homogeneous I/O conditions (no control token requirement); ECE assembly language is a high level programming language, simplifying thus graph configurations into the processor; graph execution and communication environments are separated, overlapping thus data and configuration transfer and computation; no memory usage is required during a graph execution, reducing thus latency penalty; highly scalable level of the general architecture.

At the moment our demonstrator is constituted by 4 nodes interconnected in a WK-Recursive with node degree 4. Each node uses the ALTERA APEX20K15-3C [5] component as building block for the Processing Subsystem and some extra hardware for the Routing and Kernel Subsystems. For our purpose we are employing a GIDEL Proc20KE board [17], a PCI and stand-alone board, with 5 APEX20K15-3C. One component is devoted for kernel and routing functions. The host provides the compilation and mapping environment and all libraries to program the GIDEL board.

Acknowledgments

This research is supported by Regione Campania under contract (POP-FESR Azione 5.4.2) "Sviluppo di Metodologie e

Tecniche per la Progettazione e la Realizzazione dei Sistemi Avanzati di Supporto alle Decisioni".

7. REFERENCES

- [1] Sing A., Lee M., Lu G., Kurdahi F.J., Bagherzadeh N., and Chaves Filho E.M. Morphosys: An integrated reconfigurable system for data-parallel and computation intensive applications. *IEEE Trans. Computers*, 49(5):465–480, May 2000.
- [2] Arvind, Bic L., and Ungerer T. *Advanced Topics in Data-Flow Computing*, chapter Evolution of Data-Flow Computers, pages 3–33. Prentice Hall, 1991.
- [3] Lee B. Dataflow architectures and multithreading. *IEEE Computer*, pages 27–39, August 1994.
- [4] ALTERA Corporation. Quartus programmable logic development system and software. San Jose, CA, May 1999.
- [5] ALTERA Corporation. Apex 20k devices: System-on-a-programmable-chip solutions. <http://www.altera.com/products/devices/apex/apx-index.html>, 2001.
- [6] Chen D. and Rabaey J. Reconfigurable multi-processor ic for rapid prototyping of algorithmic-specific high-speed datapaths. *IEEE J. Solid-State Circuits*, 27(12), December 1992.
- [7] Babb J. et alii. Logic emulation with virtual wires. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 16(6):609–626, June 1997.
- [8] Della Vecchia G. and Sanges C. Recursively scalable networks for message passing architectures. In Chiricozzi E. and D'Amico A., editors, *Proceedings of Int. Conf. Parallel Processing and Applications*, pages 33–40, L'Aquila, Italy, 1987. Elsevier Science Publishers.
- [9] Chen G.H. and Du D.R. Topological properties, communication, and computing on wk-recursive networks. *Networks*, 24:303–317, 1994.
- [10] Dennis J.B. Dataflow computation controlflow and data flow: Concepts of distributed programming. *NATO ASI Series F: Computer and system sciences*, vol.14, 1985.
- [11] Dennis J.B., Gao G.R., and Todd K.W. Modelling the weather with a dataflow supercomputer. *IEEE Trans. Computer*, pages 592–603, July 1984.
- [12] Gray J.P. and Kean T.A. Configurable hardware: A new paradigm for computation. In *Proc. Decennial CalTech Conf. VLSI*, pages 277–293, Pasadena, CA, March 1989.
- [13] Backus J.W. Can programming be liberated from von neumann style a functional style and its algebra of programs. *Communications of the ACM*, pages 613–641, August 1978.

- [14] Verdoscia L. and Esposito R. Introduction to chiara language. Technical Report 20, CNR Research Center on Parallel Computing and Supercomputers, Via Castellino, 111 - 80131 Napoli - Italy, September 2001.
- [15] Verdoscia L. and Vaccaro R. A high-level dataflow system. *Computing*, pages 285–305, 1998.
- [16] Verdoscia L. and Vaccaro R. An adaptive routing algorithm for wk-recursive topologies. *Computing*, (63):171–184, 1999.
- [17] GIDEL LTD. Proc20ke board. www.gidel.com, May 1999.
- [18] Murakawa M., Yoshizawa S., Kajitani I., Kajihara N. Yao X., Iwata M., and Higuchi T. The grd chip: Genetic reconfiguration of dsp for neural network processing. *IEEE Trans. on Computers*, 48(6):628–639, June 1999.
- [19] Platzner M. Reconfigurable accelerators for combinatorial problems. *Computer*, 33(4):58–60, April 2000.
- [20] Raghunath M.T. *Interconnection Network Design Based on Packaging Considerations*. Phd. thesis, University of California at Berkeley, Berkeley, CA, 1993. UCB CSD-93-782.
- [21] Albahama O.T., Cheung P., and Clarke T.J. On the viability of fpga-based integrated coprocessors. In Pocek K.L. and Arnold J., editors, *Proc. IEEE Symp. FPGAs for Custom Computing Machines*, pages 206–215, April 1996.
- [22] Iannucci R.A. Toward a dataflow/von neumann hybrid architecture. In *Proc. of the 15th Annual Symposium on Computer Architecture*, pages 131–140, Honolulu, Haw., May-June 1988. IEEE Computer Society.
- [23] Haynes S.D., Stone J., Cheung P.Y.K., and Wayne L. Video image processing with the sonic architecture. *Computer*, 33(4):50–57, April 2000.
- [24] Callahan T.J., Hauser J.R., and Wawrzynek J. The garp architecture and c compiler. *Computer*, 33(4):62–69, April 2000.
- [25] Sterling T.L. and MacDonald M.J. The realities of high performance computing and dataflow's role in it: Lessons from the nasa hpcc program. In Cosnard M., Ebcioğlu K., and Gaudiot J.L., editors, *Proc. of the IFIP WG10.3 on Architectures and Compilation Techniques for Fine and Medium Grain Parallelism*, pages 165–176, Orlando, FL, January 20-22 1993. North-Holland.

APPENDIX

Comparison between the two dataflow models

Since the classical model originally proposed by Dennis [10] is useless if we want to take advantage of VLSI technology, a new dataflow model has been introduced [15] in which only actors with homogeneous I/O conditions are present. Homogeneous I/O conditions mean that any actor has one output and two input arcs and consumes and produces only

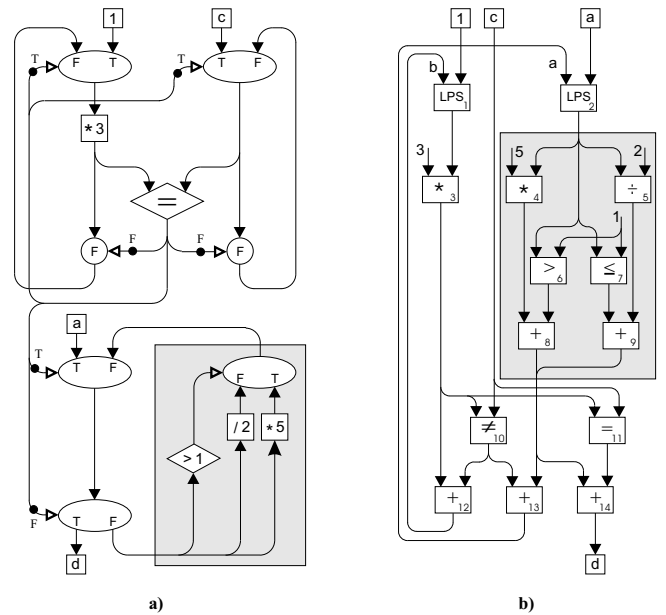


Figure 7: Graphs comparison

data tokens.

Despite their simplicity, with these actors it is possible to generate determinate graphs, including iterative and conditional constructs, where

- 1) no feedback interpretation is needed to correctly execute a dataflow program graph. This means that no check needs to verify whether the output token of an actor has been consumed, thus only one-way token flow is present,
- 2) no extra synchronization mechanism is required to control the token flow, thus the model is completely asynchronous.

We show the difference between the two models with the following program example:

```

input (a, c)
  b := 1;
  repeat
    if a > 1 then a := a \ 2
    else a := a * 5
    b := b * 3;
  until b = c;
output (d)

```

where a and c are the input variables and d is the output variable. Figure 7 shows the equivalent well-behaved DFGs adopting a) the classical and b) the new model.

The DFG inside the grey rectangle represents the body of the program, the conditional structure *if ... then ... else*, while the other part represents the iteration control.

Observing Figure 7.a the DFG has heterogeneous I/O conditions of actors and heterogeneous links and values (data

links to hold data tokens and control links to hold control tokens). For its comprehension, we have to follow the flow of two different kinds of tokens and the behavior of actors with different numbers of input and output arcs that consume and produce different kinds of tokens. Although sharing the same control link, the initial behavior of actors like F-gate, Switch, and Merge depends on their position in the DFG rather than on the program input values and are a programmer's trick to allow the computation to start correctly. In fact, the control tokens for the Switch and the three Merge actors inside the iteration control are initially set to false and true values while for the Merge actor in the conditional structure the control is related Decider actor. Besides, some actor functions are defined in a domain but assume values in a different codomain. For example, let \mathbf{D} be a subset of real numbers, \mathbf{B} be the set of boolean values, and \mathbf{W} be the set $\mathbf{D} \times \mathbf{B}$, arithmetical functions are defined and assume a value in \mathbf{D} , the Decider function is defined in \mathbf{D} but assumes a value in \mathbf{B} , and the Merge, Switch, and T-gate functions are defined in $\mathbf{D} \times \mathbf{B}$ but assume a value in \mathbf{D} . Finally, the implicit definition of static dataflow, for this model, requires that each arc is constituted by a pair of links: value and signal [11], to control the actor enabling rule. This control, obtained by means of data/acknowledgement mechanism between sender and receiver actors, augments the communication overhead originating two opposite flows of tokens (data and signal).

In opposite, the DFG shown in Figure 7.b employs only actors with homogeneous I/O conditions, their initial behavior depends on the program input values rather than on its position in the graph, the well-behavedness check, without generating opposite token flows, is guaranteed if subgraphs are macroactors [15], actors consume and produce only data tokens, and, finally, these DFGs can be directly mapped onto hardware.