Dynamically Reconfigurable Architectures for Digital Signal Processing Applications

Gilles Sassatelli, Lionel Torres, Pascal Benoit, Gaston Cambon,

Michel Robert, Jérôme Galy

LIRMM, UMR University of Montpellier II-CNRS C5506,
161 rue Ada, 34392 Montpellier Cedex 5, France
{sassate torres diou cambon robert galy}@lirmm.fr

Abstract: Tomorrow's pocket devices will all have Internet-based communication capabilities. The advent of mobile phones, PDAs (Pocket Data Assistant) and pocketPC's joint to the newcomer's third generation wireless networks such as UMTS will soon allow everyone to be connected, everywhere. In this competitive marketplace where many similar products compete for the consumer attention, performances level is a very important criterion.Videoconferencing, digital music broadcast, speech recognition are a few example of the new features allowed by the new third generation networks. This kind of multimedia, data oriented content requires highly efficient architectures; and nowadays mobile system-on-chip solution will no longer be able to deal with the critical constraints like area, power, and data computing efficiency. In this paper we will propose a new dynamically reconfigurable network, dedicated to data oriented applications such as the one targeted on third generation networks. Principles, realisations and comparative results will be exposed for some classical applications, targeted on different architectures.

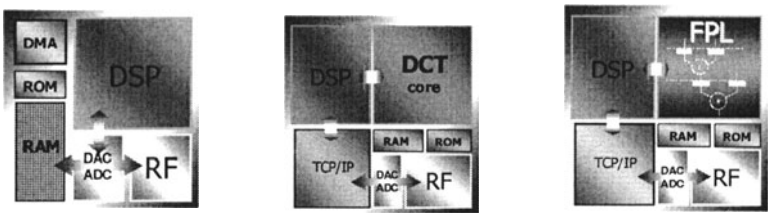
Key words: Reconfigurable computing, Data flow, Digital Signal Processing

1. INTRODUCTION

Nowadays pocket devices are mostly based on a SoC (System on Chip) approach (Figure 1). On the same silicon die are grouped heterogeneous IP (Intellectual Property) modules.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35597-9_40

M. Robert et al. (eds.), *SOC Design Methodologies*



1.1 *1.2* *1.3*

Figure 1. The three main SoC approaches

There are different ways to face these new problems:

- The easiest, and actual way to deal with this increasing computing requirements is naturally to use a more powerful DSP/μP (figure 1.1) than the ones used today; but it will probably not be feasible for the most demanding applications, as the resulting processor will grow until the size of a Pentium (such as the ones which take place in the most powerful PDA or pocket PCs), with the corresponding area, cost and consumption problems.
- Another way is to try to identify the future application field, and use a dedicated core to compute the common parts of the corresponding algorithms (Figure 1.2). For example, if JPEG and MPEG based applications are targeted, we will make the choice of implementing a wired IDCT (Inverse Discrete Cosine Transform) core, which is known to be the common most time consuming part of both algorithms[7][8]. An interesting, but restrictive solution as the application field is thus not extensible.
- Yet another way is the reconfigurable computing [2][3][10]. For example, integrate a FPGA core[1][3], where, depending on the target application different algorithm/architecture solutions could be synthesized (figure 1.3). Here, if we target JPEG applications, we will choose to synthesize the IDCT core in the FPGA, and also an application dependant part of the algorithm, like Huffman coding, or quantization. But in the other way if we target MPEG[9] applications, we will still make the choice of a wired IDCT, but this time we will also select the motion estimation[6], which is one of the most demanding part of the MPEG.

This kind of approach seems to be quite interesting; we can thus imagine, depending on a given application, a video streaming one for example, that the mobile could directly connect to the vendor's site to download the corresponding applet, which is nothing else than the configuration file of the considered reconfigurable network.

2. RECONFIGURABLE SOLUTIONS

A closer look to the kind of tomorrow's mobile applications shows a very data oriented, data intensive trend: the multimedia content needs a very high number of arithmetic operations; which would naturally imply to synthesize numbers of arithmetic operators in case of using fine grained reconfigurable logic (FPGA for example).

Arithmetic operators synthesis is known to be very area costly on fine grained reconfigurable networks. Due to the highly combinational character of adders and multipliers, the resulting functional frequencies are also often very low making FPGA-like architectures bad candidates for arithmetic level data computing.

Coarse grained reconfigurable architectures[2][3] featuring hardwired arithmetic operators are much more adapted to dataflow oriented computations.

3. SYSTEM OVERVIEW

Our architecture follows the classical bi-layer FPGA principles. Here are the main characteristics:

- The operative layer is no longer CLB (Configurable Logic Block) based, but use a coarse-grained granularity component: The Dnode (Data node). It is a datapath component, with an ALU and a few registers, as shown in figure 3. This component is configured by a microinstruction code.
- The configuration layer follows the same principle as FPGAs, it's a RAM which contains the configuration of all the component of the operative layer.
- We also use a custom RISC core [5] with a dedicated instruction set; its task is to manage dynamically the configuration of the network and also to control the data transfers between the reconfigurable core and the host CPU.

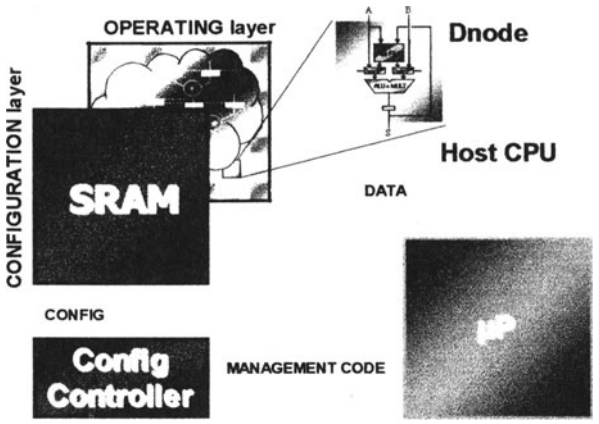


Figure 2. System Overview

This architecture is thus not intended to be a stand-alone solution, rather an IP core for dataflow oriented computing, which would take place in a SoC. Figure 2 shows schematically our system in a SoC context. The µP can thus confide the most demanding part of a given application to our IP core. So it downloads to the RISC memory the corresponding configuration program (which manages the dynamical reconfiguration).

From a functional point of view:

- The host processor first sends the management code to the configuration controller memory (the custom RISC has its own program memory). This is a object code, ready to be executed, and specially designed to manage dynamically the configuration of the network (the content of the RAM thus changes from one cycle to another), as to say, the functionality of the operating layer. Each clock cycle, the configuration controller is able to change up to the entire content of the RAM thanks to its dedicated instruction set.
- Once done, our core is ready to compute. The host processor sends the data to the operating layer via a specific scheme and then get back the computed data. As the configuration is dynamically managed, it is possible to multiplex the sent data, and to compute them by several sequential (hardware multiplexing) or concurrent (static) synthesised datapaths.

4. OPERATING LAYER ARCHITECTURE

In this section we will describe more precisely the operating layer architecture.

4.1 4.1 Dnode architecture

It essentially consists in an ALU-Multiplier, able to make all the classical arithmetic and logic operations : addition, multiplication, subtraction, roll, shift and so on. This optimised architecture is able, in the same clock cycle, to make all possible operations, even between two different registers. Its corresponding microinstruction code, the configuration code, comes from a memory location in the configuration layer. As previously said, this code evolves during the computing phase, the functionality can thus be changed from one clock cycle to another, from an addition to a multiplication, load to register, etc.

Each Dnode has in fact two execution modes :

- Global mode (normal mode), already described : the Dnode executes the microinstruction code which comes from the configuration layer, managed by the RISC configuration controller.
- Local mode : The stand-alone mode : Each Dnode has 7 registers, a up to 6-states counter and a 6 to 1 multiplexer forming a small local controller. Each one of the 6 first registers can contain a Dnode microinstruction code, and each clock cycle the counter increases the value on the multiplexer address input, thus sending the content of a register to the datapath part of the Dnode.

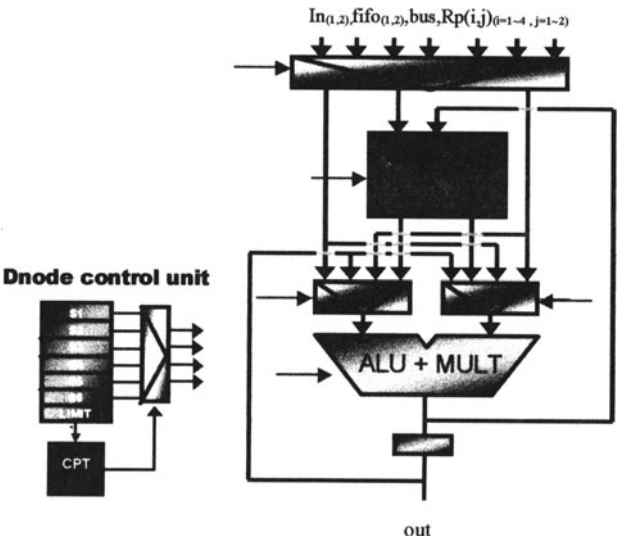


Figure 3. The Dnode Architecture

In this last mode the Dnode is like a basic RISC CPU able to compute various (otherwise control intensive) algorithms like MACs, serial digital filters, FIFO/LIFO emulation. This scheme, joint to a specific input/output Data controller allows very efficient, high bandwidth dataflow oriented computation.

4.2 The ring architecture

Related works[1][2][3] propose mesh, array or crossbar-based operating layer architecture.

Mesh-based architectures[2][3], even very flexible usually suffers from routing problems. Each reconfigurable block must features full routing capabilities with the nearest neighbours for direct communications. Routing over longer distances are achieved by dedicated lines, and with new silicon technologies allowing giant reconfigurable architectures, this requires important routing capabilities, with no-more maintainable propagation delays (general SoC problem, die-long interconnections cause hard timing problems).

Crossbar based arrays[3]. The routing capabilities are again usually quite satisfying, but area costly. The scalability of these architectures is also limited for the same reasons as mesh-based networks; and more specially FPGAs. The largest ones are facing propagation delay problems implying P&R tools to spend lot of time in routing phase.

Linear array-based architectures[3]. Aiming to map pipeline character of datapaths, they are often bi-dimensionals. Feedback operations (opposite dataflow direction, figure 5) of all kinds of digital signal processing like algorithm require additional routing resources and are often area and performance costly thus limiting the scalability for next generations.

Our approach proposes an original linear array like architecture to solve routing relative problems. This one is based on curled bi-datapath structure.

4.2.1 Forward : The main Dataflow

We use a curled, pipelined systolic structure as shown in figure 4. All the Dnodes form a ring, which length (Dnodes layers number) and width (Dnodes per-layer number) can easily be scaled.

We use a curled, pipelined systolic structure as shown in figure 4. All the Dnodes form a ring, which length (Dnodes layers number) and width (Dnodes per-layer number) can easily be scaled.

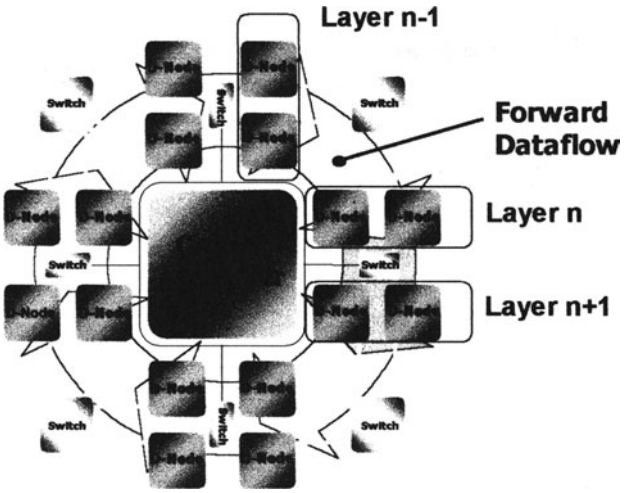


Figure 4. The ring architecture

The Dnodes are organised in layers; a Dnode layer is connected to the two adjacent ones by also dynamically reconfigurable switch components able to make any interconnection between two stages. These switches also manage data transfers with the host by dedicated FIFOs, and optional RISC communications via a shared bus.

In normal mode, each Dnode can be seen as an arithmetic operator of a datapath which computes a data each clock cycle. In stand-alone mode each Dnode can be seen as a autonomous CPU. The structure is also flexible in the way that all Dnodes have not to run in the same mode, allowing the Systolic Ring to compute either in global mode (normal mode), local mode (stand-alone) or hybrid (normal and stand-alone) mode.

4.2.2 Reverse : The secondary flow

The data feedback problem is addressed here: we use special feedback pipelines (figure 5), forming a reverse Dataflow to avoid complex routing structures. The last task that accomplishes each switch is to write unconditionally (no control needed) the computed result of the previous D-node layer in a dedicated pipeline (each switch owns its pipeline), which allows the feedback of each data to the previous stages. These ones can then choose to get these data through the switches, which have direct access to all the pipelines. This technique ensures a good scalability of the architecture, as the routing problem is thus removed.

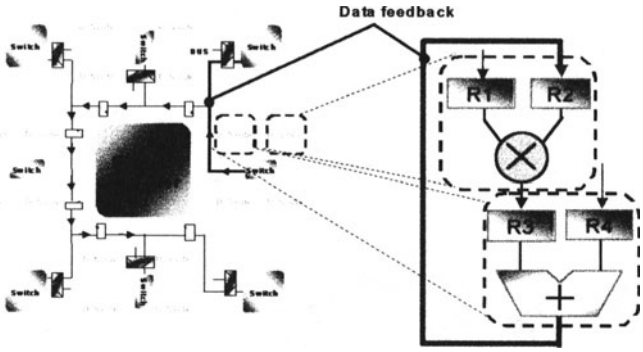


Figure 5. The feedback network

5. COMPARISONS & REALISATIONS

5.1 Comparative Results

A 8 Dnodes version has a maximal computing power of 1600 MIPS at the typical 200 MHz evaluated functional frequency, quite impressive compared to the 400 MIPS of a Pentium II 450 MHz processor. The theoretical maximum bandwidth of this version of the structure is about 3 Gbytes/s, however often limited by the communication protocol between the host CPU and the core. To program this structure we wrote an assembling tool, which parse both configuration controller level (for the control) and Ring level assembler primitives. It directly generates the machine object code, ready to be executed in the architecture.

5.1.1 Motion estimation algorithm implementation

In the application field targeted by third generation systems we can find lots of video-relative techniques. One of these well known computing intensive algorithm is the motion estimation. Widely used in video compression techniques for broadcasting, storing, and videoconferencing, his task is to remove the temporal redundancy in video streams, as the DCT's is to remove the spatial redundancy.

Block matching and specially Full Search Block Matching (FSBM) algorithm is the most popular implementation, also recommended by several standard committees (MPEG (video) and H.261 (videoconferencing) standards).

The Mean Absolute Difference (MAD) criterion, used to estimate the matching of the current block can be formulated as follows:

$$MAD(m,n) = \sum_{i=1}^{N}\sum_{j=1}^{N}\left|R(i,j) - S(i+m, j+n)\right|$$

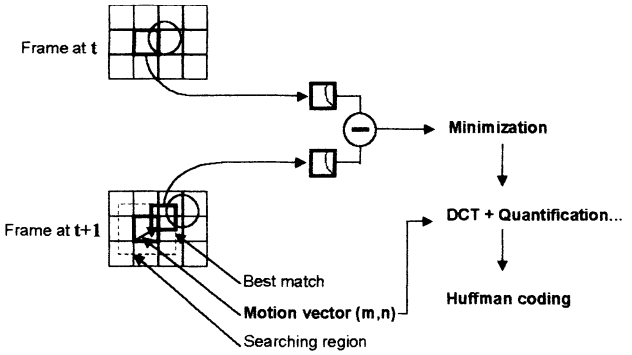


Figure 6. The motion estimation algorithm

R(i,j) is the reference block (figure 6) of size N x N and S(i+m,j+n) the candidate block within the search area determined by p and q which are the maximum horizontal and vertical displacements. The size of this area is (N+p) (N+q) pixels; and the displacement vector represented by (m,n) is determined by the least MAD(m,n) among all the (p+1)(q+1) possible displacement within the search area.

Let's consider the following common specifications: An image size of 352 x 240 pixels at 15 frames/s with a block size of 8 x 8 pixels and a maximum displacement of 8 pixels in horizontal and vertical directions.

For each candidate block the first summation (j=1 to 8) requires N operations and the accumulation N-1 operations, thus a total of 2N-1 operations. The second summation requires to compute N times the previous one account of operations and again N-1 operations for the accumulation of the partials sums. The total amount of arithmetic operations to compute is so $2N^2 - 1$.

The (2N-1).N first operations can be achieved within (2N-1).N / (0,75.Nx) clock cycles in a Nx Nodes version of our structure, as there are no dependencies on these data and one node over four is in wait state (layer n: 2 nodes computing two R()-S() operations; layer n+1: 1 node accumulating of the two previous computed results).

The last N-1 operations (accumulation) are achieved in int(ln(N))+1 clock cycles for N <= Nx.

In a 16 Nodes version of our structure, and with the previous specified codec (N=8) the computation of the MAD for a candidate block requires 13 clock cycles. Each reference block requires the computation of 289 candidate blocks and there are 1320 reference blocks in each frame. The total processing time of an image frame is 1320x289x13=4959240 cycles. At the 200MHz estimated frequency the computation time would be 24ms, which is two times smaller than the frame period (1/15s).

Table 1 shows the performances of the Systolic Ring compared with the ASIC architecture implemented in [12] and Intel MMX instructions[13] using the criterion of the number of cycles (the three architectures can achieve comparable functional frequencies) needed for matching a 8x8 reference block against its search area of 8 pixels displacement.

Systolic Ring	ASIC[12]	MMX [13]
3757	*581*	*28900*

Table 1 : Motion Estimation performance comparison (cycles)

Our structure shows again its efficiency in a such computing intensive context. The ASIC implementation is much faster than our solution at the price of flexibility: The Systolic Ring provides the advantage of hardware reuse and is also almost 8 times faster than a MMX solution.

5.2 Synthesis results & future work

The entire architecture (reconfigurable core and configuration controller) has been described in both behavioural and structural VHDL. A 8 Dnodes, 16 bits data width version has been fully simulated, and synthesised in both HCMOS7 and HCMOS8, respectively 0.25µm and 0.18µm ST technology.Table 3 shows the comparative synthesis results in both technologies.

	0.25µm	0.18µm
Dnode area	0.06 mm^2	0.04 mm^2
Ring-8 area	0.9 mm^2	0.7 mm^2
Estimated Frequency	180 MHz	200 MHz

Table 3 : Synthesis Results

The low area of each Dnode, joint to the exposed specific architecture shows that this one could easily be scaled to larger realizations. Figure 7 shows a foreseeable .18µm technology, 12 mm^2 die area SoC for high constrained embedded solution. Our specific architecture allows the

integration of a powerful 64 Dnodes version of our core (3.4 mm^2 on-die area) with a widely used ARM7 CPU, able to run various operating systems like windows CE, Linux. This kind of solution could provide a great computation power/cost ratio, which combines the flexibility of a CPU / reconfigurable architecture couple with the efficiency of applications dedicated cores.

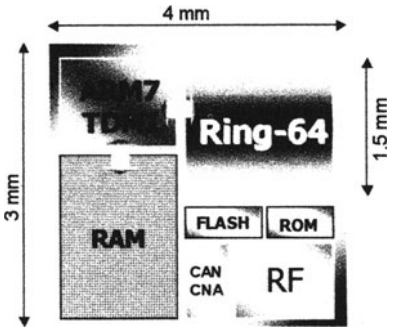


Figure 7. A forseeable SoC

6. CONCLUSION

We have proposed a new coarse grain dynamically reconfigurable architecture which proves its efficiency in data oriented processing. Its scalability shows that its field of applications is not limited to high-constrained embedded applications, but can also make be worth its faculties in other contexts, where high data bandwidth processing remains critical. A small 8-Dnodes version of this structure already provides up to 1600 MIPS of raw power for data dominated applications with a sustained data rate of 3 Gbytes/s at 200 MHz, either in global or local mode.

Our future work takes place in the translation of the structure to floating point and also in the writing of an efficient compiling tool, the key to industrial success for coarse grain reconfigurable architectures.

7. REFERENCES

[1] Stephen Brown and J. Rose, "Architecture of FPGAs and CPLDs: A Tutorial," IEEE Design and Test of Computers, Vol. 13, No. 2, pp. 42-57, 1996.

[2] Why reconfigurable computing, Department of Computer Science, Computer Structures Group, http://xputers.informatik.uni-kl.de/.

[3] R. Hartenstein, H. Grünbacher (Editors): The Roadmap to Reconfigurable computing Proc.FPL2000, Aug.27-30,2000;LNCS,Springer-Verlag2000.

[4] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Re-configurable Co-processor," Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[5] A. Abnous, C. Christensen, J. Gray, J. Lenell, A. Naylor and N. Bagherzadeh, " Design and Implementation of the Tiny RISC microprocessor," Microprocessors and Microsystems, Vol. 16, No. 4, pp. 187-94, 1992.

[6] C. Hsieh and T. Lin, " VLSI Architecture For Block-Matching Motion Estimation Algorithm," IEEE Trans. on Circuits and Systems for Video Technology, vol. 2, pp. 169-175, June 1992.

[7] N. Ahmed, T. Natarajan, and K.R. Rao, "Discrete cosine transform," IEEE Trans. On Computers, vol. C-23, pp. 90-93, Jan 1974.

[8] ISO/IEC JTC1 CD 10918. Digital compression and coding of continuous-tone still images – part 1, requirements and guidelines, ISO, 1993 (JPEG).

[9] ISO/IEC JTC1 CD 13818. Generic coding of moving pictures and associated audio: video, ISO, 1994 (MPEG-2 standard).

[10] High Productivity Computing Systems (HPCS), Defense and Advanced Research Projects Agency, http://www.darpa.mil/ito/research/hpcs/index.html.

[11] Xilinx, the Programmable Logic Data Book, 1994

[12] A.Bugeja and W. Yang, "A Re-configurable VLSI Coprocessing System for the Block Matching Algorithm", IEEE Trans. On VLSI systems, vol. 5, September 1997.

[13] Intel Application Notes for Pentium MMX, http://developer.intel.com/.