ROMA

D1.1 Intermediate Progress Report

Contract No	ANR-06-ARFU-004
Workpackage	SP1
Date	September 1st, 2008
Author, company	E. Casseau, F. Charot, A. Floc'h, S. Khan, D. Ménard, O. Sentieys, C. Wolinski: IRISA S. Guyetant, S. Chevobbe : CEA A. Tisserand : LIRMM H. Heijnen, J-P. Le Glanic, E. Raffin : Thomson
Short abstract	Progress report after 18 months

History:

Release	Date	Changes	Distribution
R1.0	July 27, 2008	Creation	
R1.1	August 29, 2008	Update	ANR + website (public domain)

Table of content

Introduction

The ROMA project proposes to develop both a design methodology and a reconfigurable processor able to adapt its computing structure to video and image processing applications. The processor is built around a pipeline of coarse grain reconfigurable operators exhibiting efficient power and performance features. A particular attention is given to the operator such that their granularity matches the domain-specific (video) computation patterns as well as the number system and the word-length used for the representation of the data. The configuration of the processor is dynamically done all along the application depending on the tasks that are to be carried out. Because an efficient processor is useless if it is hard to use, the ROMA project also intend to provide a software framework to configure the processor. From the program source code, the programming environment gets the different patterns of calculation as well as their successive arrangements and completes transformations to optimize the processor mapping.

1. Hardware Platform Design - Concepts and options

Main goal of the ROMA project is to provide a flexible platform design for reconfigurable processor design space exploration. The proposal to design a reconfigurable processor arose with the need to answer the flaws of processors and of reconfigurable architectures. Processors have long increased their ILP with over pipelining and with costly execution speculation (branch prediction, cache replacement policies) and superscalar techniques where parallelism is made explicit at run time by the hardware itself. The great advantage of general purpose processors is their compatibility with the Von Neumann model, because compilers know how to program them from a procedural code of an algorithm. On the contrary, reconfigurable arrays are programmed with architectural descriptions, which needs a skilled and long human intervention, although High Level Synthesis (HLS) methods tend to appear. Such spatial descriptions are quite efficient for regular processing and data flows, even with a high interconnect overhead, but handle poorly control-rich algorithms. So basically, the ROMA reconfigurable processor design goals are to have a reactive control processor that can reconfigure quickly computational patterns dedicated to an applicative domain.

When dealing with a given application, a good way to enhance the performance is to spot the most reused sequence of operations and to create dedicated instructions in addition to the processor instruction set. This ISA extension is particularly promoted by Tensilica, and numerous examples of multimedia application acceleration with this technique exist in the literature. The drawback is that the extensions are fixed, so that any new application with a different algorithm will not be accelerated by the same hardware. This is one of the limitations that the proposed reconfigurable processor will overcome. Stretch proposed a reconfigurable extension that is meant to be general purpose, in contrast to the ROMA proposal which targets flexibility inside a given application domain. For more details on this topic, please refer to ROMA deliverable D2.1, entitled “State of the art of reconfigurable architectures”.

1.1 Operator interface

The operator interface was specified early in the project as it is a common specification to WP2 and WP3. The interface has two input flows, one output flow, a control and status interface and also a simplified bus interface used to configure the operators. Multiple input flows were discussed but profiling on multimedia applications proved that most computations are binary and unary operations; anyway, the patterns that can be mapped on the ROMA processor template permits to create more complex operators. The VHDL code of the interface is given below:

```
entity op_roma is
 generic (
   ComplexAHB   : integer range 0 to 2   := 0;
   nb_flow      : integer range 1 to 4   := 2;
   SizeDataFlow : integer range 1 to 64  := 32;
   SizeSigCtrl  : integer range 16 to 64 := 16;
   Id_OP        : integer range 0 to 15  := 0
   );
```

```
  port (
    CLK         : in  std_logic;
    nRST        : in  std_logic;
    FLOW1_data  : in  std_logic_vector(SizeDataFlow-1 downto 0);
    FLOW1_vld   : in  std_logic;
    FLOW1_rdy   : out std_logic;
    FLOW2_data  : in  std_logic_vector(SizeDataFlow-1 downto 0);
    FLOW2_vld   : in  std_logic;
    FLOW2_rdy   : out std_logic;
    FLOWO_data  : out std_logic_vector(SizeDataFlow-1 downto 0);
    FLOWO_vld   : out std_logic;
    FLOW0_rdy   : in  std_logic;
    OP_commands : in  std_logic_vector(SizeSigCtrl-1 downto 0);
    OP_status   : out std_logic_vector(SizeSigCtrl-1 downto 0);
    AMBA_IN     : in  ahb_slv_in_type;
    AMBA_OUT    : out ahb_slv_out_type
    );
end op_roma;
```

1.2 Template Operator design:

A basic operator has been defined in order to validate the interface, to serve as a template for WP3 advanced operators and also to be used as an example for the developments done in WP2, therefore it was not optimized. This operator is composed of usual ALU operations plus a 16*16 multiplier. In order to test the architecture with classical signal and image processing, a barrel shifter is added to accelerate floating points operations, and a 40 bits accumulator is used to perform SAD and MAC operations with the same sample bandwidth. The operator was synthesized for a Virtex-4 technology at 130MHz and 1500 LUTs (not using the DSP blocks), and for a 130nm ASIC technology at 200MHz, 0.06mm². It is depicted in the next figure:

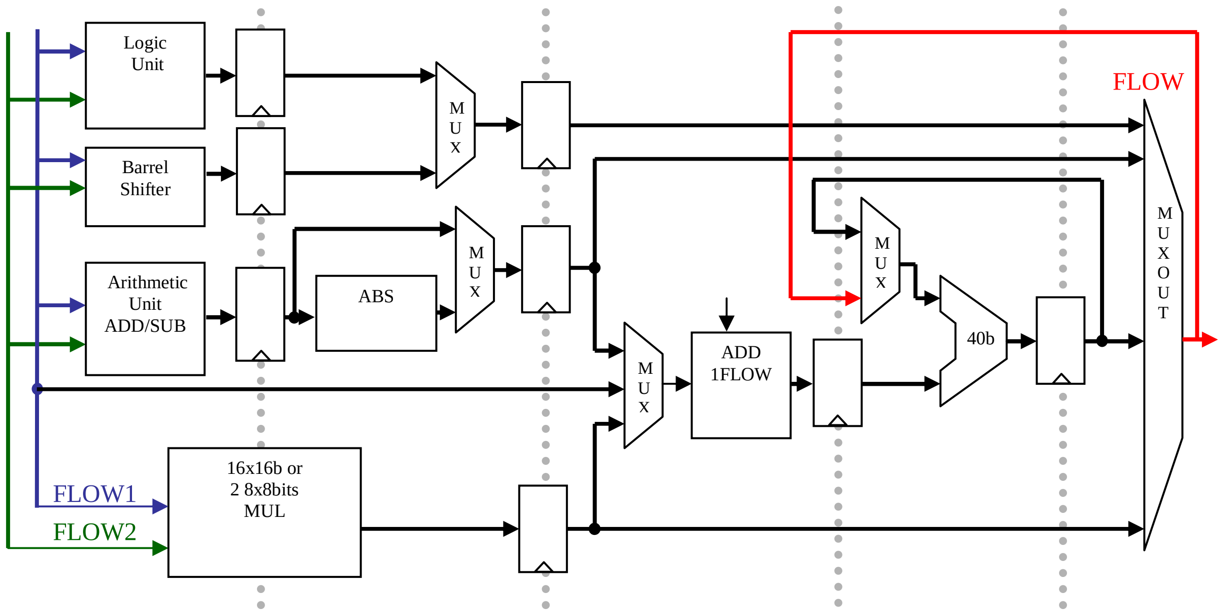

Figure 1.1

1.3 Top level architecture options:

The main discussion with a great impact on the overall design is the granularity of the operators, and the atomic tasks that are executed on these operators. Here are the two opposite and extreme points of the design space to be explored: first when these operators are simple ALUs, the processor is no more than a VLIW processor. Then the memory elements have to be registers files and the control has to be updated every cycle. A VLIW compiler is needed to program such architecture. The opposite point consists in having coarse tasks that are autonomous and can run a continuous data flow. The control is no more a major issue, because updates take place every hundreds or thousands of cycles.

The hardware components can be dedicated, or reconfigurable arrays programmed by hand or by means of HLS (high level synthesis) methodology. The ROMA reconfigurable processor that is proposed in this project deals with operators that have a latency of a few cycles. The operators can be chained in a pipeline in order to improve the instruction parallelism. By analysing the Control and Data Flow Graphs of an application, the control statements have to be executed by the controller and not inside the operators. Small memory chunks are used to bufferize the data that is processed by a given "configuration" of the processor. These configurations are meant to be small so that there is no overhead due to their loading onto the operating part of the processor, and their storage is not problematic compared to a classical instruction flow.
The global architecture is depicted below. The number of operators is configurable: N=1 is the model of a scalar processor. Complexities of 4 and 8 are of particular interest, while larger processors will be very hard to achieve high operator usage (with other mode than SIMD of course). The number M of memory banks is related to the maximum number of concurrent accesses, because dual ported memories are rather area-inefficient. Thus M is comprised between 2 (pure pipelined operators) and 3N (pure SIMD mode); the ideal range will be refined with a profiling of composite operators patterns, and is a compromise between access flexibility, data allocation complexity, and area of memories and interconnect. The interconnect is split in two parts, because the requirements are different: the memory interconnect can tolerate some latency, but with the so called pattern interconnect, the goal is to achieve the smallest latency.

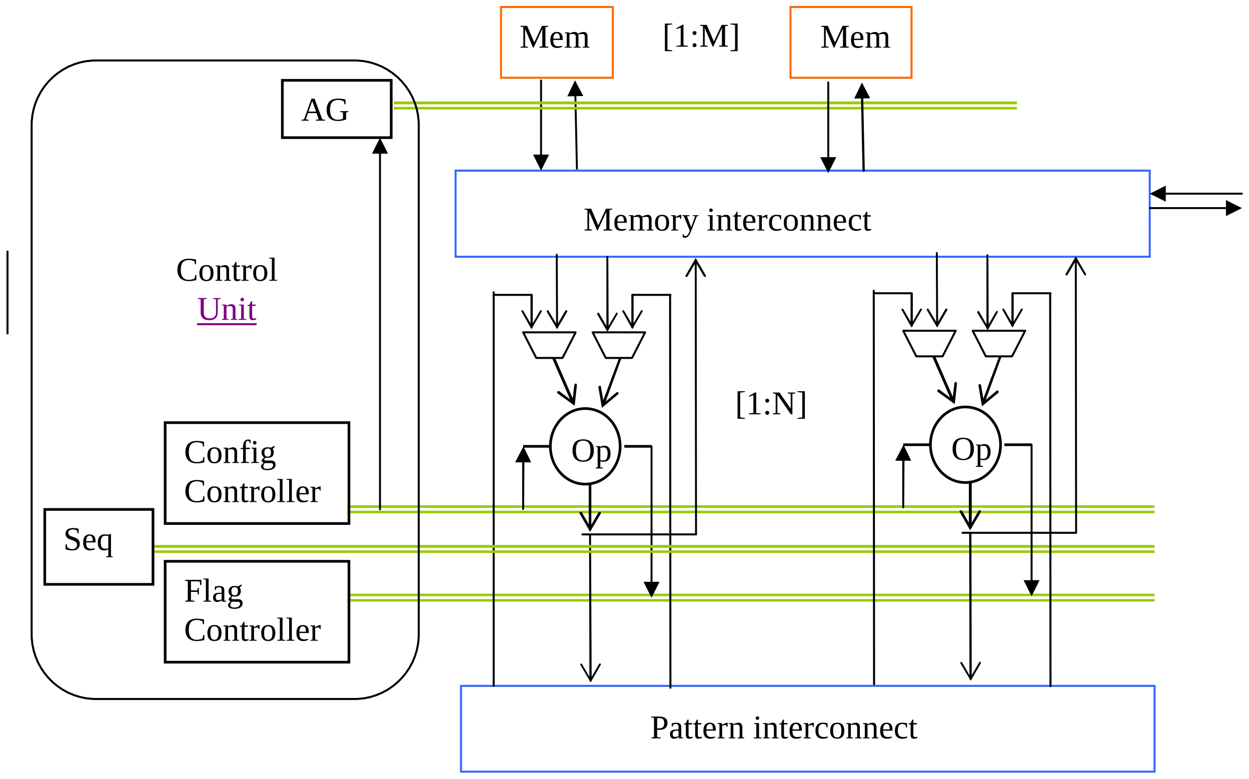

Figure 1.2

1.4 Controller options and specification:

The control part is composed of four main functionalities:

- a sequencer that drives the algorithmic execution;
- a configuration controller, that fetches the various configurations, eventually decompresses or recompose them, and load them inside the configuration interfaces;
- an address generator (AG) that points the right data location in memory according to access patterns;
- a flag controller that centralizes the configuration and execution status of each operator

To get the most performing and flexible solution, it has been decided to merge into a single control unit all the controlling parts. This unit is a small and efficient programmable controller whose IS is dedicated to control operations.

In order to design a well adapted control unit able to execute these functionalities, dedicated control codes have been analyzed. From this analysis a first version of the instruction set (IS) of the dedicated controller had been defined. As the application code of the consortium is not available, a dedicated benchmark had been created. It has been extracted from c code based on applications dedicated to control available in the laboratory such as:

- configuration management
- parsing of control data flow graph
- very small operating system dedicated to reconfigurable computing
- uCoS

The structure analysis of these codes allowed to evaluate the best IS needed to implement code dedicated to the control. Then the static profiling allowed to define a first encoding of the IS. This controller processor is based on a classical well known five stages pipeline. The implementation of the pipeline is in progress.

2. Targeted applications

Several applications were extracted from our target domain: the encoding, decoding video loop process were studied, described and exposed to the partners:

- Motion estimation, hierarchical algorithm HME
- Motion compensation, pixel recursive algorithm
- Deblocking filter
- Motion Vector derivation (decoding),
- Up_scaling algorithm
- Entropy coding, CABAC and VLC algorithms

	Operation types	**Specification availability**	**Existing implementation**	**Constraint and limitation**
Motion estimation	MB loop, SAD8x8, filter 6 tap	Thomson	Fire8x8 + dedicated HW	Thomson algorithm property
Motion compensation	Pixel recursive	Thomson	Dedicated HW	Thomson algorithm property
Deblocking filter	MB loop, H and V filtering, data transfer	Normative H264	Fire 4x4	no
Motion Vector derivation	MB loop Mult(discalfactor)	Normative	Dedicated HW	no
up scaling	Filtering,	Thomson	Fire 4x4	Thomson algorithm property
Entropy coding	arithmetic coding, normalization	Normative H264	Dedicated HW	no

The selected applications will be Motion Compensation and up-scaling. Due to first analysis: control graphs, required operators and complexity.

2.1 Motion Compensation

Motion Compensation main features are:

- Improving display by Motion compensation for matrix display: Plasma, LCD
- Rotated scan mode for slim tube application (interleaved)
- Frame Rate Conversion: generic converter nHz to pHz:
  - standard conversion 50Hz -> 60Hz
  - 75Hz, 100Hz up conversion
  - Film -> Video conversion
- Noise reduction for mpeg coding (professional uses)

➔ What is available today
- o Implementation on one FPGA for MPEG2 encoder
  - ▪ rtl netlist available
- o Detailed specification
- o C model available and used for IP validation

2.2 Up scaling

Nowadays, with the accession of the High Definition TV (HDTV), the Standard Definition TV (SDTV) is doomed to disappear. There are a lot of algorithms already developed for SD to HD conversion. The simplest ones are based on linear filtering (spatial and temporal filtering), some of them combine non linear filtering (median filter) and the most complex ones include motion compensation (Ex.: for de-interlacing to retrieve the missing video information).

Our method is based on the extraction of local characteristics of an image. It requires differential operators of the first order:

- The local orientation of “isophotes” (=line or surface which joins points of same light intensity =~contour orientation)
- The interpolation is done in accordance with the orientation of these “isophotes”.

This proposed method is a good compromise between complexity of implementation and quality and reliability of the conversion results.

Main ideas of the algorithm is to interpolate pixels of a frame in the direction of the estimated isophotes in order to make format conversion SD to HD

The up scaling algorithm enables to interpolate lines and rows, functions of the size ratio between input and output. The following figure illustrates the general process of interpolation by inpainting.

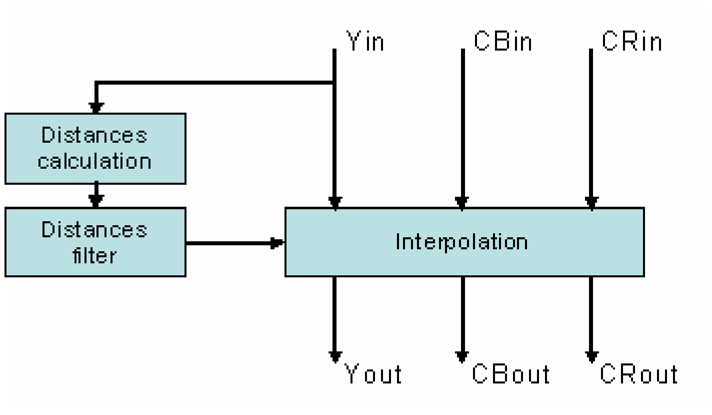

Figure 2.1

➔ What is available today
- o Implementation on the FIRE processor
  - ▪ Assembly code
- o Detailed specification of the algorithm
- o C model available

3. Reconfigurable operators

In standard processors or circuits, arithmetic operators and supported number systems (or representations) are fixed at design time. The high level of optimization of those operators leads to high performances but without any flexibility. Implementing features out of the scope of the initial specifications (operations and/or representations) must be handled in software at a very high cost (memory, computation time and power consumption).

The purpose of the reconfigurable operator design task in the ROMA project is the study and the prototyping of reconfigurable arithmetic operators (or units) with the support of multiple arithmetic operations and various (simple and advanced) numbers representations. In most of video applications, high-level programs are divided into consecutive steps, each step performing one “image-level” operation1 on the complete image. Typical examples of such steps are: image filtering, edge detection and motion estimation. At each step the number of required representations and operations is limited but there are repeated with a very high level of parallelism (for all pixels in the image). Using dedicated operators for each step leads to very large circuits with functional blocks used at a low rate. Implementing all operations and representations in one block leads to very large general purpose operators with low performances (complex instruction decoding).

The configuration of the operator specifies the small set of operations and representations supported during the step. The control during the step is then simple (programming model close to the high-level description of the algorithm). Once the reconfigurable operator is configured the internal activity is limited to the useful computations.

One goal of the project is the study and prototyping of arithmetic operators with variable levels of parallelism. Word-level parallelism and sub-word parallelism (SWP) is used. The word-level parallelism is achieved using several independent arithmetic operators at the same time. At this level SIMD or MIMD solutions can be used. SWP capabilities allow to execute several operations in parallel on different sub-word data using a SIMD programming scheme.

3.1 Numbers representations and arithmetic algorithms exploration :

The optimization of a circuit can be done at different levels: system, algorithm, architecture, circuit and technology. Arithmetic operators have no concern with the system and technology levels. Algorithm, architecture and circuit levels widely impact the design of arithmetic operators and vice versa. More precisely, there is a complex trade-off between:

- the **number system(s)** used to represent the data (width, number coding, mathematical properties, achievable accuracy...);
- the **algorithms** used to compute the mathematical operations (evaluation methods, internal radices, speed/area trade-offs, internal parallelism, fused operations...);
- the **characteristics of data** (range, required accuracy, signal activity, space/time correlations...);
- and some **circuit constraints** (specific cells in the standard library, timing and/or area constraints, logic style...).

Our application field requires the support of standard number representations for the operands and the results of the reconfigurable operators: unsigned binary integers and fixed-point values, 2's complement integers and fixed-point values and sign/magnitude integers and fixed-point values. For internal representation of numbers in the operator, we plan to use other representations such as redundant number systems or the logarithmic number system. We do not plan to use floating-point representations w.r.t. our application domain. The use of redundant number systems allows constant time addition with solutions such as carry-save or borrow-save adders. This property is very useful for reconfigurable arithmetic operators. Not for speed purpose but for the limited propagations of carries. This simplifies significantly the decomposition of operators into smaller parts (for configuration and/or SWP aspects).

[1]Usually composed of a few arithmetic, control and memory basic operations for each pixel.

The *carry-save* (CS) number system is widely used. This is a radix-2 representation in which the digits belong to {0,1,2}. The representation is said redundant because some numbers have several representations. For instance, the integer 6 can be represented in carry-save using (0110)cs or (0102)cs. In CS, each digit x of {0,1,2} is represented by the sum of 2 bits: x = x_c + x_s where x_c and x_s are in {0,1}. The conversion from a value represented using a redundant number system into a value represented using a non-redundant number system can be done using a non-redundant addition. While a redundant number system allows to compute the addition of two arbitrary large numbers in a constant time, it has some drawbacks. First, more bits are required to represent the values than using a non-redundant number system. Second, due to the possible multiple different representations of a value, comparisons are more complex than standard comparisons. Therefore, redundant addition is mainly used as an internal representation. One frequent use of redundant adders is to perform the sum of 3 or more values. Radix 4 or 8 redundant number representations may lead to interesting solutions for internal representation of values in the reconfigurable operator. The use of those higher radix representations will be done latter in the project.

Assuming basic representations of numbers, the addition and the subtraction operations are very close. In the following, we denote addition both operations. The simplest addition architecture is based on a linear array of FA cells. It is known as sequential adder or ripple carry adder (RCA). This adder is the slowest useful adder, but it is also the smallest and it has a regular layout (this simplifies some internal decompositions).

Carry select adders (CSeA) split a sequential adder into two parts and perform the computation of the MSB part with the two possible carry-in bits (0 and 1) in parallel and select the right one using the carry-out bit of the LSB part. The MSB part is doubled. Recursively applied, this method leads to a logarithmic time adder at the algorithmic level. But CSeA are not so fast in practice because of high fan out problems. This type of adder is used in combination with a faster scheme in some multiple size operators. For SWP operators, CSeA may be an interesting starting point.

Carry skip adders (CSkA) are based on a linear structure of blocks of sequential adders and additional logic used to skip blocks when all ranks propagate the carry inside the block. They are constant or variable block widths CSkAs. The speed of those adders is O(sqrt(n)) for n-bit operands. Although this adder has not the highest theoretical speed, it is widely used for fast but not critical additions because of its small area and regular layout. Fixed and equal block widths CSkA are well suited for SWP operators.

There are several kinds of logarithmic adders such as carry lookahead adders (CLA) or parallel-prefix ones. Parallel prefix adders are based on the computation of the carry-in bits for all ranks using partially shared generate/propagate trees. The sharing of the different trees can be done using different schemes. This leads to the various types of parallel-prefix adders. Decomposition of parallel prefix adder is not simple. We plan to study specific trees for SWP aspects in a latter part of the project.

Multiplication involves two basic operations: the generation of the partial products and their sum. Therefore, there are two possible ways to accelerate the multiplication: reduce the number of partial products or speed up their sum. Both solutions can be applied simultaneously. Reducing the number of partial products also reduces the time required to perform their sum. In practice, most of high-performance multipliers are based on the three following steps: parallel generation of the partial products, tree reduction of the partial products into a redundant sum (e.g. carry-save tree), and conversion of the redundant sum into a non-redundant representation using a fast adder.

The proposed architecture of the reconfigurable operator is depicted in Figure 3.1 without the programmable routing resources. The replication of the basic (small) arithmetic operators allows word-level parallelism. The SWP aspects are not represented on this figure.

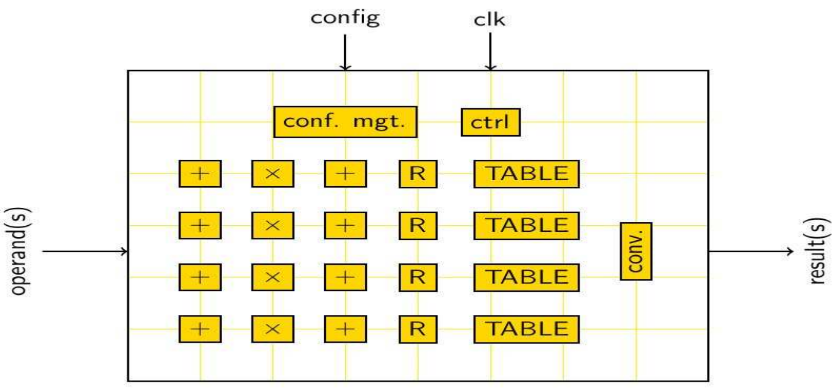

Figure 3.1

The configuration management block provides all the resources required for the configuration of the operator. The control block (also configured by the configuration management block) provides control capabilities during one step (for one configuration). This control corresponds to the decoding of the few instructions executed during one step. The control block can be implemented using a programmable finite state machine. The lookup tables stores coefficients or values used during one step. The size of these tables must be defined by the analysis of some applications examples. The reconfigurable operator integrates a conversion block when the internal representation must be converted before transmission to the output. The conversion block can be bypassed.

3.2. Sub word parallelism operator design :

Subword parallelism (SWP) is a form of parallel vector processing which utilizes data level parallelism. In SWP single instruction initiates parallel execution on data organized in parallel. As a result of SWP, the same data path and computation units perform more than one computation on a composite word. This composite word consists of more than one adjacent subwords. Thus a more efficient use of the processor data path is obtained when the size of the operands is smaller than the word size. Figure 3.2 shows a simple example of an adder with SWP capability. This adder is similar to a conventional adder which accepts two n bits numbers (words) as operands and produces n bit sum and 1 bit carry. If words are splited in 4 subwords for example, by blocking the carry chains that propagate the carry from bit (n/4)-1 to bit (n/4), from bit 2.(n/4)-1 to bit 2.(n/4), and from bit 3.(n/4)-1 to bit 3.(n/4), this adder is able to perform simultaneous four n/4 bit additions.

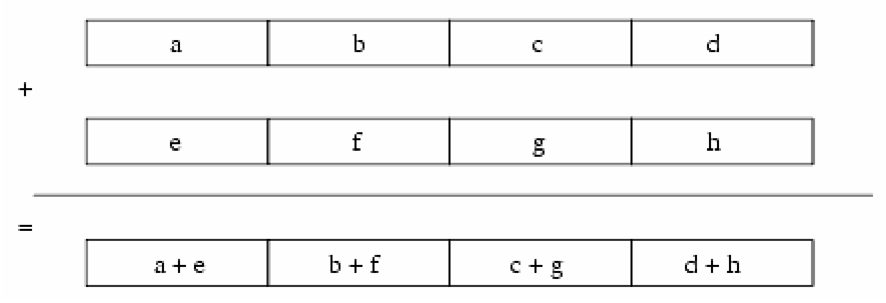

Figure 3.2 Parallel Subword ADD Instruction

The multimedia data at the lowest component level tends to be 8, 10, 12 or 16 bits of precision. However general purpose processors are generally optimized for processing data in unit of words, where a word is at least 32 or 64 bits. Hence SWP is an efficient and flexible solution for accelerating multimedia processing because the algorithms exhibit a great deal of data parallelism on lower precision data. Furthermore a more efficient use of memory also results, since a single load or store word instruction moves multiple packed subwords between memory and processor registers. SWP has been introduced as multimedia extensions to major general-purpose microprocessors. DSP with SWP capability include tigerSHARC from Analog Devices, TMS320C64x from Texas Instruments etc. GPP with multimedia extension include MAX-I and MAX-II in PA RISC, AltiVec in Motorola's PowerPC, MMX in Intel IA-32 and IA-64.

In order to obtain a low power efficient reconfigurable hardware operator dedicated to multimedia applications, SWP enabled basic operators are required. These operators include adder, subtractor, multiplier, MAC (multiply and accumulate) etc. Research work has been done on the implementation of SWP enable operators. In [1] SWP enabled adders is proposed that introduces SWP capability by breaking the carry chain of carry look ahead adder using carry propagate and carry generate signals. In [2] an efficient architecture for SWP enabled multiplier is proposed. This multiplier implements SWP capability with the minimum increase in area resources and critical path. It uses simple AND operation for generation of partial products and then arrange them efficiently for each subword size. In [3] variable precision multiplier is proposed for an FPGA platform. It uses minimum FPGA resources for implementing a variable precision multiplier. In [4] a 64-bit fixed point vector MAC unit is proposed which supports multiple precision (one 64x64, two 32x32, four 16x16 or eight 8x8). Its design is based on shared segmentation method. In all these SWP enabled operator designs, classical subword sizes (8, 16, 32) are considered rather than multimedia oriented subword sizes. From these papers, we have design basic DSP dedicated operators (ADD, MULT, MAC) using architectures whose efficiency matches SWP technique. Simple and SWP versions of each operator are implemented using different algorithms. Based on data sizes of current multimedia applications, subword sizes of 8, 10, 12 and 16 bits are chosen. Area and critical path (delay) are examined. The targeted technologies are ASIC standard cells HCMOS9GP 130nm (CORE9GPLL 4.1 low leakage standard cell library from ST Microelectronics) and 90nm (fsd0t_a standard performance low voltage threshold cell library from UMC) and FPGA (Xilinx Virtex II). Implementation results [P1] show that although SWP enabled operators consume more area and CP compared to simple designs but they increase the efficiency through the parallel execution of data.

4. Software framework

4.1 Architecture models

The architecture models supported by the compilation design flow are presented on figure 4.1.

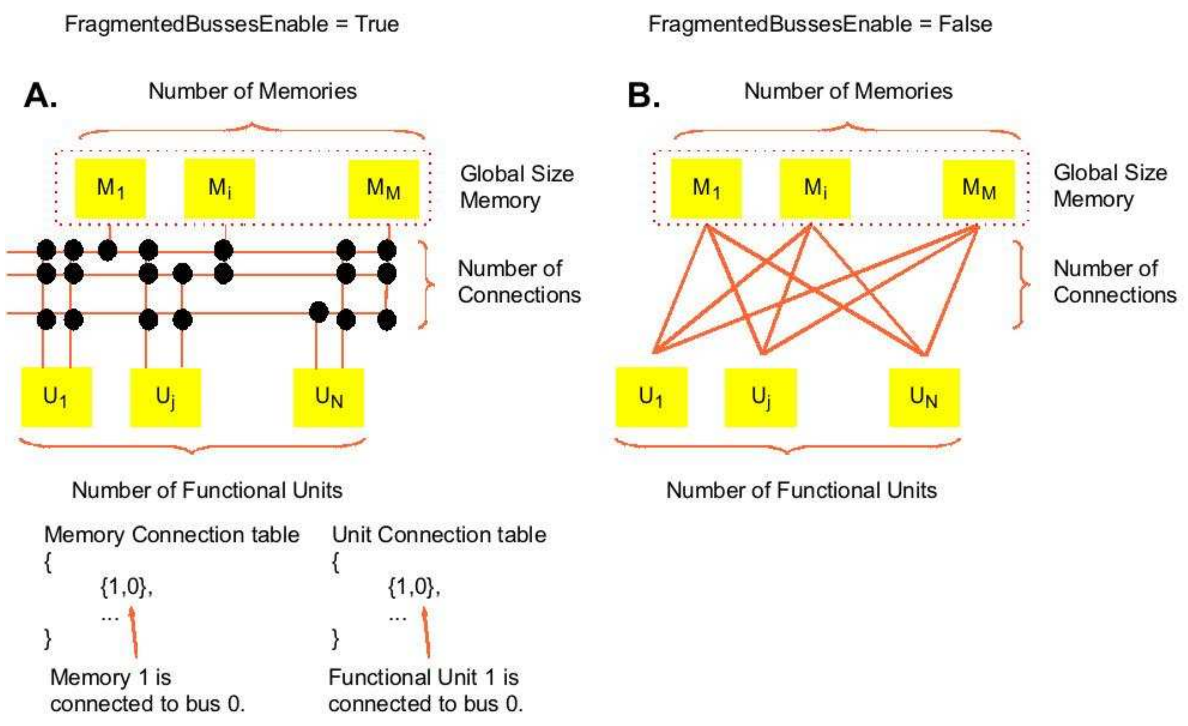

Figure 4.1. Architecture models.

The first architecture (on the left side of figure 4.1) is a reconfigurable architecture with fragmented buses. In this architecture the processing units and memories can be connected together only through specific connections. Thus the connections are not uniform inside the architecture. The “Memory Connection Table” specifies the possible connections for memories while the “Unit Connection Table” gives information about the connections supported by processing units.

The second architecture is more uniform. Each unit (in unit set U) and each memory (in memory set M) can be connected together without restrictions by N shared busses.

The selection of a specific architecture supported by the design tool is made by assigning a corresponding value to the “FragmentedBussesEnable” parameter.

The architecture is fully parametric: number of processing units, number of memories, number of connections and number of busses can be defined by the user.

4.2 Application execution

Figure 4.2 shows an example of the execution of the application on the reconfigurable architecture from figure 4.1.

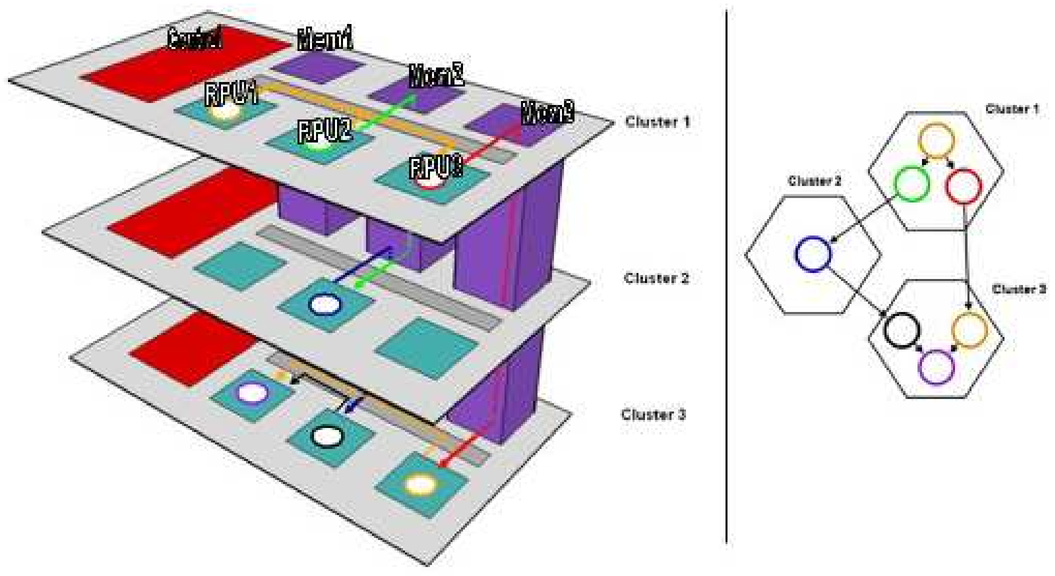

Figure 4.2. Execution of the application principle.

The application data flow graph (right side of figure 4.2) is passed over the reconfigurable architecture (knowing that the architecture can execute in the same cycle a sub-graph of the data flow graph) in such a way that the number of needed reconfigurations is minimal. In order to execute an entire application in the case of the example presented on figure 4.2 three reconfigurations are necessary. For this reason the figure shows the same architecture during the execution of three sub-graphs indicated by clusters on figure 4.2. It needs to be pointed out that the data goes through memories between successive configurations.

4.3 Design flow

The design flow supporting the execution model presented in section 4.2. and the reconfigurable architecture models from section 4.1 are shown on figure 4.3. It is illustrated by a real life FIR example.

The entry of the design flow is an application data flow graph and an architecture model (section 4.1).

The reconfigurations are obtained after three steps.

- Step 1. A research of interesting patterns in the data flow graph is made. The interesting patterns could be frequent computational patterns or other patterns of interest. Patterns identification uses advanced software technologies that include graph-matching techniques [P2,P3]. Pattern identification is necessary for the gathering of information about the locality of the processing.

- Step 2. In this step the pattern selection and scheduling are executed. In result, new nodes called super nodes replace the matches of the selected patterns inside the data flow graph. This task uses flexible scheduling techniques developed in the context of the UPaK system [P4,P5].

- Step 3. During this step the super node expansion, node placement and data dependencies routing inside the reconfigurable architecture are executed in such a way that the number of reconfigurations is optimized. These tasks are performed for the nodes coming from the sliding window which slides through the data flow graph obtained in the second step.

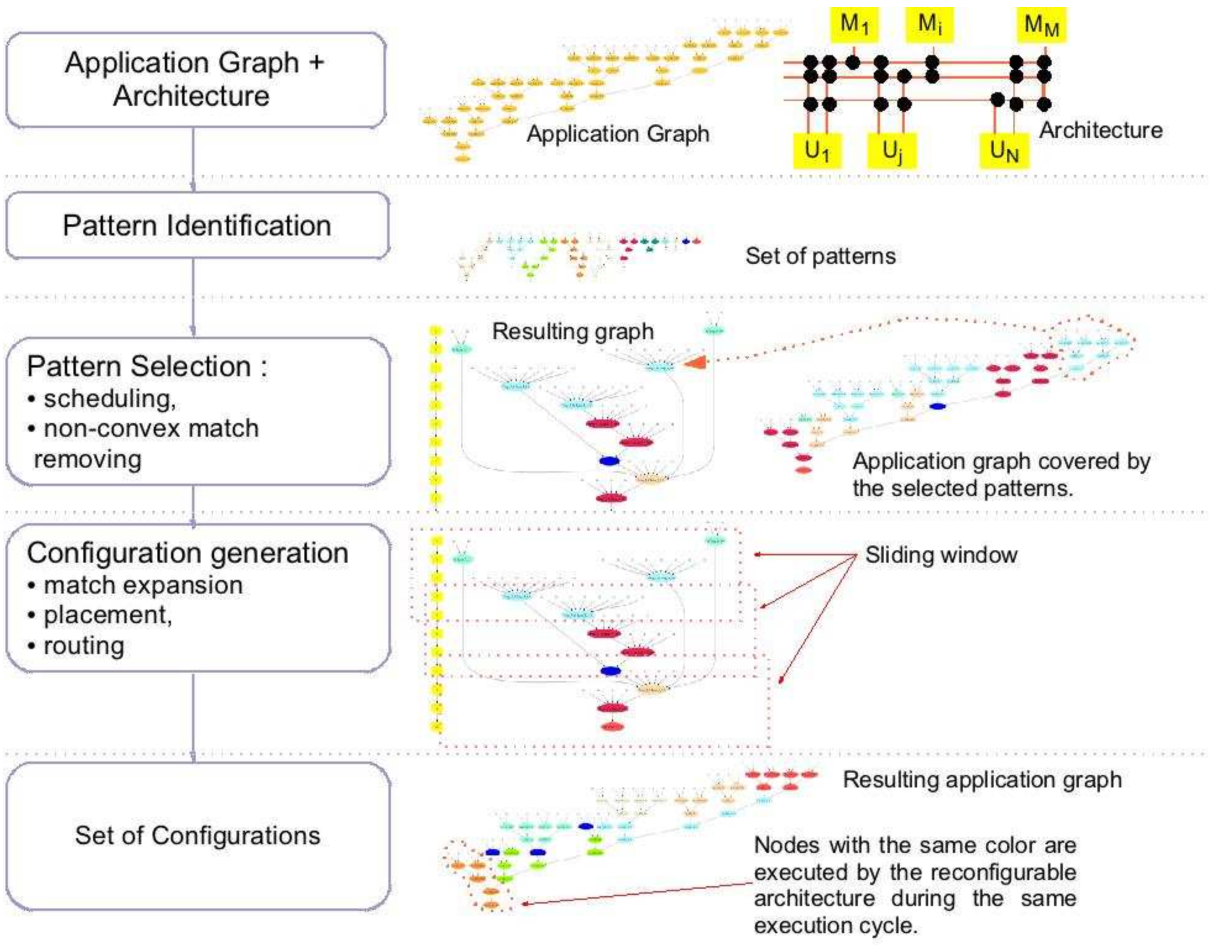
Application Graph + Architecture
Application Graph
M1
Mi
MM
U1
Uj
UN
Architecture
Pattern Identification
Set of patterns
Pattern Selection :
• scheduling,
• non-convex match removing
Resulting graph
Application graph covered by the selected patterns.
Configuration generation
• match expansion
• placement,
• routing
Sliding window
Set of Configurations
Resulting application graph
Nodes with the same color are executed by the reconfigurable architecture during the same execution cycle.

Figure 4.3. Design flow

4.4 Data word-length optimization

In the data word-length optimization process, the goal is to select the data word-length combination that maximizes the performances, minimizes the energy consumption and fulfils the computation accuracy constraint. Thus, the architecture cost is minimized under accuracy constraint. This optimization process is made-up of two main parts corresponding to the data-word-length selection and the computation accuracy evaluation.

Description of the tool for fixed-point computation accuracy evaluation :

In the digital signal processing domain, the most commonly used criteria for the evaluation of fixed-point specification accuracy is the Signal-to-Quantization-Noise-Ratio (SQNR). The aim of our method is to compute the application output SQNR expression by using an analytical method based on the theoretical approach presented in [P6]. The flow of the methodology is presented in figure 4.4. The input of the tool is the application description based on a signal flow graph with an XML syntax. This single graph is called Gs. The tool determines the analytical SQNR expression. It consists of three successive transformations (T1 to T3) of the application graph as represented in figure 4.4.

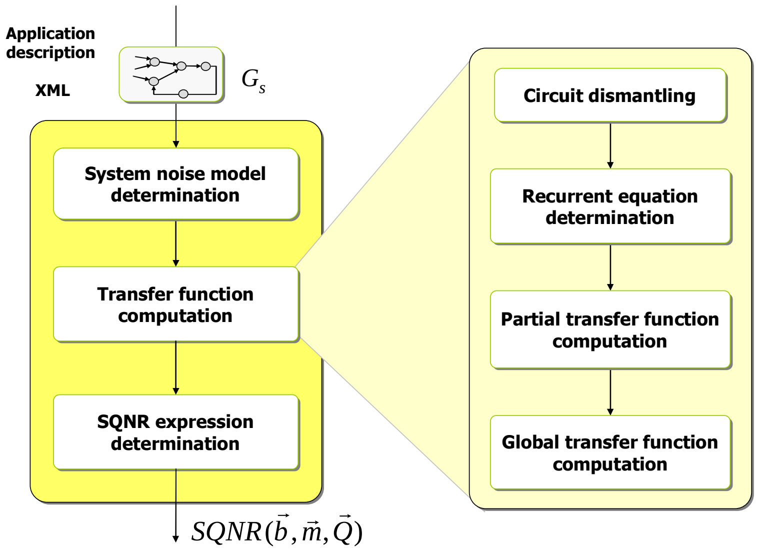
Application description
XML
G_s
System noise model determination
Transfer function computation
SQNR expression determination
$SQNR(\vec{b},\vec{m},\vec{Q})$
Circuit dismantling
Recurrent equation determination
Partial transfer function computation
Global transfer function computation

Figure 4.4. Methodology for the SQNR expression determination

The goal of transformation T1 is to represent the application at the quantization noise level. First, the noise sources are inserted in the graph. Secondly, the data and operations are replaced by their noise propagation model.

The goal of transformation T2 is to determine the transfer function set which defines the system. In our approach, the transfer functions are obtained from the Z transform of the recurrent equations representing the system. The recurrent equations are built by traversing the graph from the inputs to the output. But, this technique is unusable if cycles are present in the graph as in our case when recursive structures are considered. Consequently, this technique requires first of all, the transformation of this graph into several directed a-cyclic graphs.

The goal of transformation T3 is to compute the SQNR expression. The gain between the output and the noise source are computed from the transfer functions obtained in the previous transformation. The statistical parameters of the noise sources are computed from the data word-length (b), the binary position (m) and the quantization laws (Q). These three kinds of elements are the variables of the SQNR function. The noise power expression is available through a C function. Then, this C code is compiled and dynamically linked to the fixed-point conversion tool.

The transformation T1 has been developed and validated. The transformation T2 is more complex and is under development. A first version for non recursive systems will be first provided. In this case, the circuit dismantling step and the global transfer function computation step can be bypassed.

5. Events where the ROMA project was presented

[E1] Colloque STIC, nov 5-7, 2007, Paris, parc de La Villette, France

[E2] Semaine de l'innovation", june 23, 2008, Rennes, France.

6. Publications linked with the ROMA project

[P1] S. Khan, E. Casseau, D. Menard
"SWP for multimedia operator design"
Colloque GDR SOC-SIP, Paris, France, june 4-6, 2008

[P2] C. Wolinski, K. Kuchcinski
"Identification of Application Specific Instructions Based on Subgraph Isomorphism Constraints"
ASAP Canada, Montreal, 2007

[P3] C. Wolinski, K. Kuchcinski
"Computation Patterns Identification for Instruction Set Extensions Implemented as Reconfigurable Hardware"
ERSA 2007, Las Vegas, USA

[P4] C. Wolinski, K. Kuchcinski, A.Postola
"UPaK : Abstract Unified Pattern Based Synthesis Kernel for Hardware and Software Systems"
(University Booth) DATE 2007, Nice, France

[P5] C. Wolinski, K. Kuchcinski
"Automatic Selection of Application-Specific Reconfigurable Processor Extensions"
DATE 2008, Munich, Germany

[P6] D.Menard, R. Rocher, and O. Sentieys.
"Analytical Fixed-Point Accuracy Evaluation in Linear Time-Invariant Systems"
Accepted for publication in IEEE Transactions on Circuits and Systems I, 2008.

References :

[1] A. A. Farooqui, V.G. Oklobdzija and F. Chechrazi *64-Bit Media Adder* in proc. IEEE Int. Symp. Circuits and Systems, Orlando, Fl, May 1999.

[2] S. Krithivasan and M.J. Schulte, *Multiplier Architectures for Media Processing,* Thirty seventh Asilomar Conference on signals, systems and computers, 2003, pp 2193-2197 vol.2

[3] P Corsonello1, S Perri, M.A Iachino1 and G Cocorullo *Variable Precision Arithmetic Circuits for FPGA Based Multimedia Processors,* IEEE Transactions on very large scale integration (VLSI) systems, VOL. 12, No.9, September 2004.

[4] A. Danysh, D. Tan, *Architecture and Implementation of a Vector/SIMD Multiply-Accumulate Unit*, IEEE Computer Society, volume 54, Issue 3, March 2005 Page(s): 284 – 293