

Design methodology for runtime reconfigurable FPGA: From high level specification down to implementation

Florent Berthelot , Fabienne Nouvel , Dominique Houzet
CNRS UMR 6164 IETR/INSA

Rennes 20 av des Buttes de Coesmes, 35043 Rennes, France
florent.berthelot@ens.insa-rennes.fr, {fabienne.nouvel, dominique.houzet}@insa-rennes.fr

Abstract—In this paper we present an automatic design generation methodology for heterogeneous architectures composed of processors, DSPs and FPGAs. This methodology is based on an Adequation Algorithm Architecture where application is represented by a control data flow graph and architecture by an architecture graph. We focus on how to take into account specificities of partially reconfigurable components during the adequation process and for the design generation. We present a method which generates automatically the design for both fixed and partially reconfigurable parts of a FPGA. This method uses prefetching technic to minimize reconfiguration latency of runtime reconfiguration and buffer merging to minimize memory requirements of the generated design.

I. INTRODUCTION

Recent developments in radio technology have introduced software defined radio paradigm SDR [1]. This evolution has emerged with the proliferation of wireless standards, including both local area networks 802.11 to 2,5G, 3G, and future 4G telecommunication standards. Future telecommunication systems will be software reprogrammable radios, which could be reconfigured to adapt for changing communication protocols and channels.

Such systems require heterogeneous architectures based on digital signal processors (DSPs), general purpose processors and reconfigurable devices like field programmable gate arrays (FPGAs). The whole application is split between Hardware/Software (HW/SW) components during the partitioning process. This step leads to a compromise between system's performance of an hardwired solution and flexibility of a software solution.

These components use different levels of reconfigurations. Harvard-based architectures (DSPs, general purpose processors) are reconfigurable at system-level and use a temporal scheme implementation of operations. Configuration of the data path requires few data and can be performed at each cycle. This high flexibility is well adapted to control based applications but suffers from its power efficiency for repetitive and high throughput computations. The introduction of Dynamically Reconfigurable Systems (DRS) can deliver higher levels of performance, flexibility and power efficiency. Reconfigurable devices, including FPGAs, can fill the gap between hardwired and software technology. Recently runtime reconfiguration (RTR) of partial FPGA parts has led to the

concept of virtual hardware. Its allows to change only a specified part of the chip while other areas remain operational and unaffected by the reconfiguration [2]. So RTR allows more sections of an application to be mapped into hardware, a larger part of the application can be accelerated by contrast with a processor computation. By changing dynamically the functionality performed by the FPGA over the time, we can address SDR constraints and obtain a scalable system which can evolved in agreement with its environment requirements, and hence using a reconfigurable hardware platform across multiple standards. However reconfiguration latency is a major drawback of runtime reconfiguration on commercial devices and must be considered during HW/SW partitioning process. The objective is to obtain a near optimal scheduling of tasks in time over an heterogeneous architecture [3]. These more and more complex systems must be handled by an appropriate design flow to reduce development time under strong time-to-market pressure. These steps can be achieved by modeling application operations through control data flow graph (DFG) and operate an Adequation Algorithm Architecture (AAA) methodology.

In this paper we focus on codesign systems with DSPs and commercial FPGAs for telecommunication applications. The first section describes the SynDex tool [4] used for the application partitioning stage. We describe the impact of run-time reconfigurable component utilization on algorithm-architecture adequation in the second section. Next the way of modeling a partially runtime reconfigurable part of a FPGA with SynDex is exposed. Then we present the automatic VHDL design generation for a runtime reconfigurable component. We discuss how to generate an automatic management of runtime reconfiguration over the time with SynDex in section 4. A case study based on a runtime reconfigurable multicarrier code-division multiple-access (MC-CDMA) transmitter is presented in section 5 followed by the conclusion.

II. AAA APPROACH AND SYNDEX REPRESENTATION

Some partitioning methodologies based on various approaches are reported in the literature [5]. They are characterized by the granularity level of the partitioning, metrics, target hardware, support of runtime-reconfiguration, flow automation and on-line / off-line scheduling policies. Few tools, based

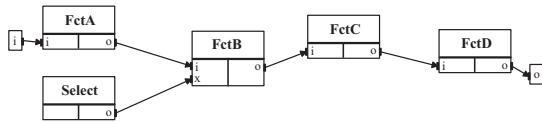


Fig. 1. Algorithm graph

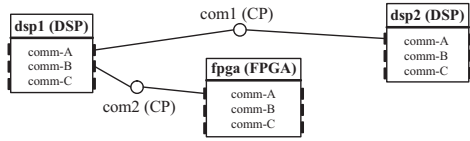


Fig. 2. Architecture graph

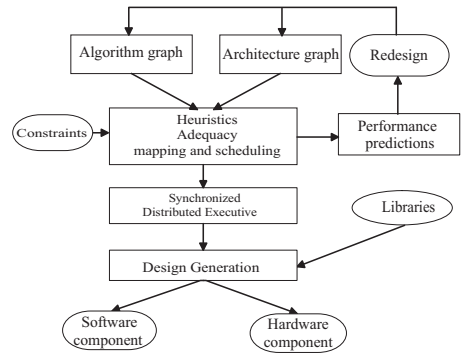


Fig. 3. SynDEx methodology flow

on these partitioning methodologies, provide a seamless flow from the specification to the implementation.

Choice of candidates for a dynamic hardware implementation can be guided by some metrics: execution time, memory constraints, power efficiency, reconfiguration time, configuration prefetching capabilities and area constraints.

Among these methods we have chosen SynDEx already used in [6]. SynDEx is an academic system-level CAD tool which supports AAA methodology. This free tool has been developed by the INRIA Rocquencourt France laboratory and several others laboratories contribute on its development over different research topics. AAA methodology aims at finding the best matching between an algorithm and an architecture while satisfying time constraints. SynDEx automatically generates a distributed and optimized synchronized executive.

Application algorithm is represented by a data flow graph (DFG) to exhibit the potential parallelism between operations as represented by Figure 1. The algorithm model is a direct data dependence graph. An operation is executed as soon as its input are available, and this DFG is infinitely repeated. SynDEx includes hierarchical algorithm representation, conditional statements and iteration of algorithm parts.

Architecture is also modeled by a graph, which is a directed graph where the vertices are operators (e.g processors, DSP, FPGA) or media (e.g OCB busses, ethernet) and edges are connections between them. Operators have no internal parallelism computation available but the architecture exhibits the actual parallelism between operators. An example is shown in Figure 2. In order to perform the adequation between these two graphs, operations and data dependencies have to be characterized (time execution) on each vertices of the architecture graph.

Adequation consists in performing a mapping and a scheduling of the operations and data transfers onto the operators and the communication media. It is carried out by a heuristic which takes into account durations of computations and inter-component communications. The result is a synchronized executive represented by a macro-code for each vertices of the architecture. Figure 3 depicts the overall methodology flow [7]. Each macro-code is then translated toward a high level language for each HW/SW components. This translation

produces an automatic dead-lock free code generation, macro-code directives are replaced by a corresponding code given in libraries (C/C++ for software components, VHDL for hardware components). Many libraries have been developed for heterogeneous platforms. Today this tool is used on heterogeneous architecture based on DSP and FPGA. Then we want to extend SynDEx capacities to runtime reconfigurable components.

III. RUN TIME RECONFIGURATION CONSIDERATIONS FOR ADEQUACY

A. Minimizing reconfiguration cost

Runtime partially reconfigurable components must be handled with a special processing during the adequation step. A major drawback of using runtime reconfiguration is the significant delay of hardware configuration. The total runtime of an application includes execution delay of each task on the hardware along with the total time spent for hardware reconfiguration between computations. The length of the sequence of reconfiguration is proportional to the reprogrammed area on the chip. Partial reconfiguration allows to change only a specified part of the design hence decreasing the reconfiguration time. An efficient way to minimize reconfiguration overhead is to overlap it as much as possible with the execution of others operations executed on others components. It is known as configuration prefetching [8]. Figure 4 illustrates our purpose. In this example we want to map and schedule an algorithm graph onto an architecture composed of two resources, one of them is dynamically reconfigurable. The algorithm, composed of six functions, is split into two branches. Function D needs the results of C and Y functions. Operations $\{X, Y\}$ are assumed to be executed by the dynamic reconfigurable component of the architecture successively. Operations $\{A, B, C, D\}$ are implemented on the non-reconfigurable component. According to this algorithm, among all operations scheduling possibilities, one potential selection is:

$$SI:\{A,X,B,Y,Y,C,D\}$$

which is infinitely repeated. With this scheduling two reconfigurations are needed: one after computation of X to implement Y , one after the second computation of Y to

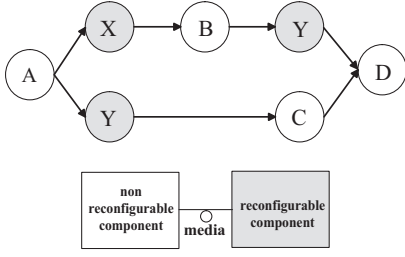


Fig. 4. Algorithm and architecture example

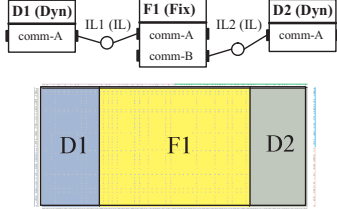


Fig. 5. Model of runtime reconfigurable parts of a FPGA with SynDEX

implement X again. We have to define the following terms:

- C_k : Computation cost of operation k .
- $T_{j,k}$: Cost of data transmission between operations j and k through the media.
- R_k : The reconfiguration delay of operation k .
- $D_{j,k}$: The time between the end of an operation j to the beginning of the operation k , both executed on the reconfigurable component.
- P_k : Prefetching cost of operation k .

The prefetching cost of operation k is $P_k = R_k - D_{j,k}$, with j a previous different operation. So considering scheduling $S1$, we have to add the following prefetching cost to the first operation Y:

$$P_y = R_y - D_{x,y}$$

$$\text{with } D_{x,y} = T_{x,b} + C_b + T_{b,y}$$

In order to improve P_y , R_y must be similar to the delay between two reconfigurable operations. Hence a way to minimize prefetching overhead is to overlap a maximum of computations or communications with reconfigurations of the dynamic part. The heuristic of SynDEX has to be improved to take into account this overhead on reconfigurable operations.

B. Architecture graph modeling of runtime reconfigurable components

Runtime reconfigurable parts of an component must be considered as vertices in the architecture graph. As shown in the example in Figure 5, runtime reconfigurable parts of a FPGA (D1 and D2) and fixed parts (F1) can be represented as hardware operators of the architecture. An internal media (IL) allows data exchanges between them. (D1 and D2) will integrate the dynamic operator and the control to manage it.

IV. AUTOMATIC DESIGN GENERATION

A. General FPGA synthesis scheme

Once mapping and scheduling of the algorithm are performed, macro-code is automatically generated and each one must be translated. The translation generates the VHDL code, both for the static and dynamic parts of a FPGA. We use the GNU macro processor M4 and macros embedded in libraries. Different kinds of macro are developed for communications, memory allocations, operator instantiations and synchronization. The final FPGA design is based on several dedicated processes to control:

- communication sequencings,
- computation sequencings,
- operator behaviour,
- activation of reading and writing phases of buffers,

A same operator can be used at different time in the data flow, only one instantiation of each kind of operator is done in VHDL. We have defined an uniform interface for each operator through encapsulation. As described in the next section, we use global buffers which allow us to merge different kinds of buffer (depth and data width) filled by operators to store computation results. These two points lead to build complex data paths automatically. Our libraries are able to perform these constructions by using conditional VHDL signal assignment. In next section, we deal with two important points: optimization of memory for exchanged data and control of reconfigurations.

B. Macro-code preprocessing for memory minimization

The goal is to merge as much as possible independant buffers to minimize the total memory requirement, this is known as buffer merging technics [9]. We operate a macro-code preprocessing which analyzes data life of variables stored in these buffers and results in a list of buffers which must be merged into a global one. Buffers can be different as they must store computation results of operations working on various depth and data width (we consider only 8,16 or 32 bits width). Hence global buffers are automatically generated in agreement with operators data type computation results associated. We denote by:

- $L : \{b_1, b_2, ..b_n\}$: A list of n buffers which can be merged, where $L_{(k)} = b_k$.
- D_k : The depth of buffer k .
- W_k : The data width of buffer k .

Hence the total amount of memory needed is:

- Without buffer merging: $M_{bm}^- = \sum_{i=1}^n D_{L(i)} * W_{L(i)}$
- With buffer merging: $M_{bm} = \max(D_{L(i)} * W_{L(i)})$ for $1 \leq i \leq n$.

So the saved memory is:

$$S_{mem} = M_{bm}^- - M_{bm}.$$

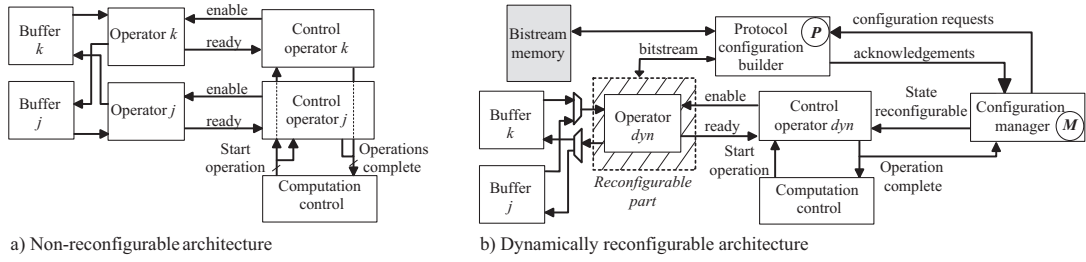


Fig. 6. Architecture comparison between fixed/runtime reconfigurable solutions

C. Design generation for runtime reconfigurable components

In order to perform reconfiguration of the dynamic part we have chosen to divide this process into two sub-parts: a configuration manager and a protocol configuration builder. Figure 6 shows a simple example based on two operations (j and k) which are executed successively. Case a) shows the design generated for a non-reconfigurable component, the two operators are physically implemented. Case b) is based on a dynamically reconfigurable component which implement successively the two operations. A configuration manager is in charge of the bitstream which must be loaded on the reconfigurable part by sending configuration requests. These requests are sent only when an operation has completed its computation and if a different operation has to be loaded after. So reconfigurations are performed as soon as the current operation is complete to enable configuration prefetching as described before. This functionality provides also information on the current state of the reconfigurable part, this is useful to start operator computations (with signal 'enable') only when the reconfiguration process is ended. Configuration requests are sent to the protocol configuration builder which is in charge to construct a valid reconfiguration stream in agreement with the used protocol mode (e.g selectmap). Encapsulation of operators with a standard interface allows to reconfigure only the area containing the operator without altering the design around. Functionalities involved in the general control of the dynamic area and the operator remain on a static part of the circuit with buffers. That allows to reduce the size of the bitstream which must be loaded and decrease the time needed to reconfigure.

Now this way to proceed must be adapted with architecture considerations. There are many ways to reconfigure partially a FPGA, Figure 7 shows three solutions of architectures for this purpose. Case a) shows a standalone self reconfiguration where the fixed part of the FPGA reconfigures the dynamic area. This case can be adapted for small amounts of bitstream data which can be stored by on-chip FPGA memory. However bitstreams which contain partial configurations require often a lot of memory and can't fit within the limited embedded memory provided by FPGAs, so bitstreams are stored by an external memory as depicted in the case b). The last case c), shows the used of a processor to perform the reconfiguration. In this case the FPGA sends reconfiguration requests to the

processor through hardware interruptions for example. This processor can be viewed as a slave for the FPGA. Either it can act as a master by reconfiguring directly the dynamic area of the FPGA. The CPLD is used to interface these two components.

Labels M and P show where functionalities 'Configuration manager' and 'Protocol configuration builder' respectively are implemented. Locations of these functionalities have a direct impact on the reconfiguration latency. Case c) has the highest reconfiguration latency since the 'protocol configuration builder' is a task of the processor which can be activated through an hardware interruption. Moreover external memory accesses are often costing. Macro-codes generated for runtime reconfigurable components are handled by a special library. The 'Configuration manager' is automatically generated by our libraries in agreement with the sequencing of operations expressed in the macro-code. The reconfigurable part provides a virtual hardware, so at some time only one operator is physically implemented on this dynamic part.

Operations of the DFG implemented on this reconfigurable part are viewed as a global operation from the point of view of the control computation process. Computation invocations of these operations are renamed with a generic name in the macro-code. This renaming is feasible because there is no ambiguity on which operator is addressed when signal 'enable' is driven (see Figure 6). That allows to have only one control functionality for managing tasks implemented by the dynamically reconfigurable part.

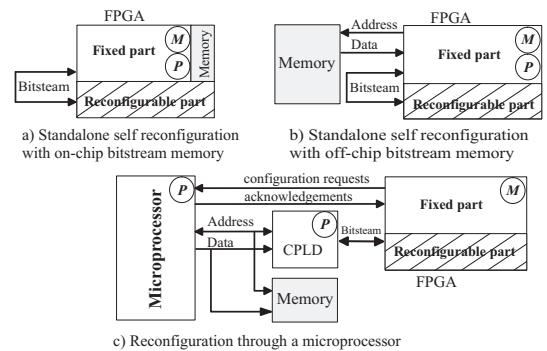


Fig. 7. Different ways to reconfigure dynamic parts of a FPGA

V. IMPLEMENTATION EXAMPLE

Our implementation application is a transmitter system for future wireless networks for 4G air interface [10]. This transmitter is based on MC-CDMA modulation scheme. MC-CDMA combines Orthogonal Frequency Division Multiplexing (OFDM) with spreading. SynDex algorithm graph, depicted by Figure 8, shows the basic numeric computation blocks of this transmitter. Block *modulation* performs either a QPSK or QAM-16 modulation. This adaptive modulation is selected by the conditional entry *Select* which defines the modulation of each OFDM symbol according to the signal to noise ratio. *Spreading* block implements a Fast Hadamard Transform (FHT). Next a frequency interleaving is done in order to take into account the frequency diversity offered by OFDM modulation. The OFDM modulation is performed by an Inverse Fast Fourier Transform thanks to *IFFT* block which also implements zero-padding process. As this paper focus on dynamic reconfiguration we consider blocks Spreading, Interleaving and IFFT as a unique global block not reconfigurable. We have to implement this transmitter over a prototyping board from Sundance technology [11]. This board is composed of one DSP C6201 from Texas Instrument and one FPGA Xilinx Xc2v2000 partially reconfigurable. Its model with SynDex is represented by Figure 9. Mapping of functionalities over the architecture gives the following results. The DSP is in charge of data generation and modulation selection. Block *Modulation* is constrained to be executed on the dynamic part of the FPGA (*fpga_dyn*). *Spreading*, *Interleaving* and *IFFT* are executed on the fixed part of the FPGA. Table I sums-up the most significant parameters of this transmitter.

A. Design generation

Figure 10 represents the computation sequencing for the partially reconfigurable part as expressed by the macro-code. This computation sequencing has to be implemented on the reconfigurable part of the FPGA, hence some transformations must be made. We merge buffers $\{B, C, D\}$ and operations $\{QPSK, MAQ16\}$ to generate operation *Op_Dyn* before the VHDL synthesis.

B. Implementation results

1) *Xilinx design flow for partial reconfiguration*: This application has been implemented on a Virtex II FPGA from Xilinx. The code, both for fixed and dynamic part has been automatically generated with SynDex, thanks to the libraries.

Sampling frequency	: 20MHz	
Total/Used subcarriers	: 256/192	
Numbers of users (full-load system)	: 32	
Guard interval	: 0.4 μ s	
Frame duration	: 1.32 ms	
Modulation	: QPSK	: MAQ16
Net bit rate per user (streaming mode)	: 0.909 Mb/s	: 1.818 Mb/s

TABLE I

MC-CDMA TRANSMITTER PARAMETERS- INDOOR SCHEME

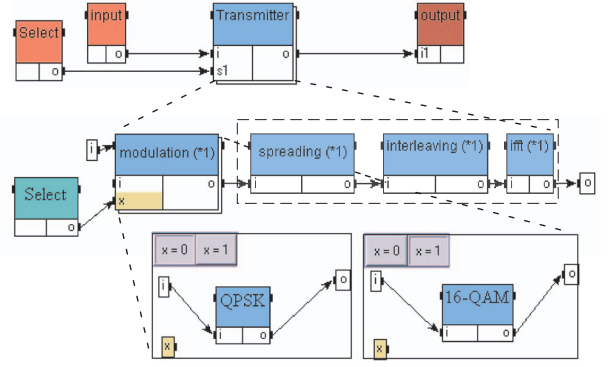


Fig. 8. Algorithm graph of a MC-CDMA transmitter

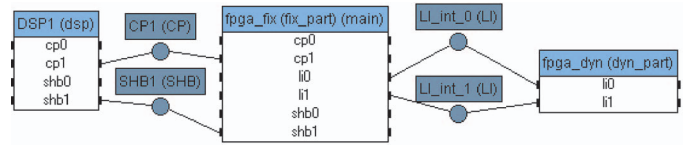


Fig. 9. Architecture graph

However, the generation of bitstreams needs a specific flow called *modular design*. This design flow is broken down into three main phases: creating the floorplan and constraints for the overall design, implementing each module through the place and route process, assembling individual modules together into a complete design. Reconfigurable modules communicate with other modules, both fixed and reconfigurable, by using a special bus macro (3-state buffers) and must be top-level modules.

2) *Numerical results of implementation*: Virtex II integrates an internal reconfiguration access port (ICAP) for partial reconfiguration. Scheme *b* of Figure 7 is applied.

Figure 11 shows the resulting design of the reconfigurable transmitter. The FPGA is divided in two parts. The first one is static and implements non reconfigurable logic, the second one takes 8% of the FPGA and is dedicated to the dynamic operator. The design automatically generated in agreement with SynDex macro-code is framed by a gray rectangle. We can distinguish blocks executed by the part *fpga_fix* of Figure 9 such as *Spreading*, *Interleaving*, *IFFT* and communications process (*Interface_IN_OUT*). This non-reconfigurable part also integrates the design automatically generated for block *modulation*. As modulation functionality is runtime reconfigurable, design generated by SynDex for this block doesn't integrate the main functionality which is implemented by operator *Op_Dyn*.

The DSP can select modulation performed by the dynamic part by sending this value to module *Interface_IN_OUT*. Next this value is sent to block modulation through communication link *LIO*. *Interface_IN_OUT* module receives DSP data from SHB bus. Receiving process can be locked-up during partial reconfigurations thanks to signal *In_Reconf*. If *Select* register value is changed, block *modulation* sends a reconfiguration

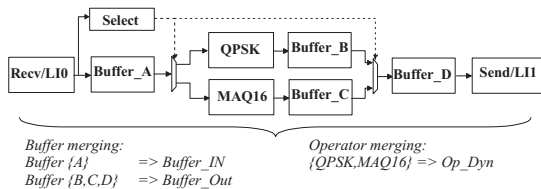


Fig. 10. Representation of macro-code computation sequencing for the partially reconfigurable part of the FPGA

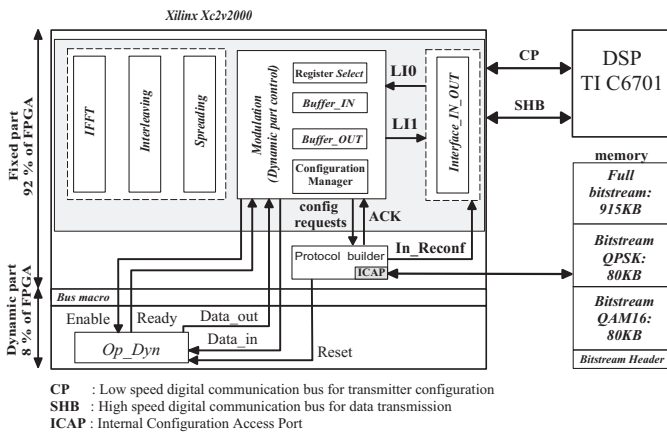


Fig. 11. Reconfigurable MC-CDMA transmitter architecture

request to the protocol builder component which is next in charge to address external memory and drive ICAP. This self reconfiguration operates at 50Mhz, one bistream byte is loaded each cycle by the ICAP. So the reconfiguration time needed to reconfigure *Op_Dyn* takes about 4ms. As noted by Table I the maximum net bit rate per user in streaming mode is about of 0.909Mbits/s with QPSK modulation. Our Xilinx IFFT core implementation allows us to reach a maximum bit rate per user of 0.296 Mbits/s. Time to reconfigure the modulation is of the order of some data frames, hence this system is enough reactive to allow fast adaptations with time-varying channels. Other computation blocks such as *spreading* or *interleaving* can be implemented using this method to obtain a full scalable system.

As shown in Table II, FPGA resources utilization needed to implement QPSK and QAM-16 modulations are more important with a dynamic reconfiguration scheme. This overhead is due to the generic VHDL structure generation, based on the macro code description, which is more adapted for medium-complex data path control. However this gap is decreasing with the number of different reconfigurations needed and the ability of runtime reconfiguration to provide virtual hardware. However the flexibility given by this methodology and the automatic VHDL generation can overcome hardware resource overhead. For instance we can easily add a Forward Error Correction (FEC) process, such as Reed-Solomon encoder, in addition with modulation process. So we obtain a new configuration for the dynamic part. Since the size of the reconfigurable part is fixed by the designer, any design able

to meet this area constraint can be implemented.

Modulation implementation	Fix (QPSK+QAM16)	Dynamic		
		Dynamic Part control	Protocol Builder	Reconfigurable area capacity
CLB Slices :	17	600	107	870
Dff - Latches :	32	300	123	-
Fct generator :	33	274	213	-
Block RAM (18Kbits) :	0	2	0	14
FPGA area :	-	-	-	8 %
Switching latency :	-	-	-	4ms

TABLE II
FIX-DYNAMIC MODULATION IMPLEMENTATION COMPARISON

VI. CONCLUSION

We have described a methodology flow to manage automatically partially reconfigurable parts of a FPGA. It allows to map applications over heterogeneous architectures and fully exploit advantages given by partially reconfigurable components. The AAA methodology and associated tool SynDEx have been used to perform mapping and code generation for fixed and dynamic parts of FPGA. Either, SynDEx's heuristic needs additional developments to optimize time reconfiguration. Furthermore, complex design and architecture can support more than one dynamic part. This design flow has the main advantage to target as well as software components as hardware components to implement complex applications from a high level functional description. This methodology can easily be used to introduce dynamic reconfiguration over already developed fixed design. As well as for fast IP block integration on fixed or reconfigurable architectures.

REFERENCES

- [1] J. Mitola, "Software radio architecture evolution: Foundations, technology tradeoffs, and architecture implications," *IEICE Trans. Commun.*, vol. E83-B, no. 6, pp. 1165–1172, June 2000.
- [2] E. L. Horta, J. W. Lockwood, D. E. Taylor, and D. Parlour, "Dynamic hardware plugins in an fpga with partial runtime reconfiguration," *Design Automation Conference (DAC)*, 2002.
- [3] J. Noguera and R. Badia, "A hw/sw partitioning algorithm for dynamically reconfigurable architectures," *Proc. Design Automation and Test in Europe*, vol. pp. 72934, 2001.
- [4] C. Sorel and Y. Lavarenne, "From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations," *Formal Methods and Models for Code Design Conference, France*, pp. 123–132, June 2003.
- [5] J. Harkin, T. McGinnity, and L. Maguire, "Partitioning methodology for dynamically reconfigurable embedded systems," *IEE Proc-Comput. Digit. Tech.*, vol. 147, November 2000.
- [6] V. Fresse, O. Deforges, and J.-F. Nezan, "Avsyndex: A rapid prototyping process dedicated to the implementation of digital image processing applications on multi-dsps and fpga architectures," *EURASIP JASP*, pp. 990–1002, september 2002.
- [7] F. Berthelot, F. Nouvel, and D. Houzet, "Design methodology for dynamically reconfigurable systems," *JFAAA, Dijon France*, pp. 47–52, January 2005.
- [8] S. Hauck, "Configuration prefetch for single context reconfigurable coprocessors," *ACM/SIGDA International Symposium on FPGAs*, vol. E83-B, no. 6, p. 6574, June 1998.
- [9] P. K. Murthy, S. S. Bhattacharyya, and E. A. Lee, "Minimizing memory requirements for chain-structured sdf graphs," *Proc. of ICASSP, Australia*, vol. E83-B, no. 6, p. 6574, June 1994.
- [10] S. Lenours, F. Nouvel, and J.-F. Helard, "Design and implementation of mc-cdma systems for future wireless networks," *EURASIP JASP*, pp. 1604–1615, August 2004.
- [11] S. M. T. Ltd, "http://www.sundance.com."