Synthesis of Multimode digital signal processing systems

Caaliph Andriamisaina, Emmanuel Casseau, Philippe Coussy

▶ **To cite this version:**

Caaliph Andriamisaina, Emmanuel Casseau, Philippe Coussy. Synthesis of Multimode digital signal processing systems. NASA/ESA Conference on Adaptive Hardware and Systems, 2007, Edinburgh, United Kingdom. pp.7. hal-00153086

HAL Id: hal-00153086
https://hal.science/hal-00153086

Submitted on 8 Jun 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Synthesis of Multimode digital signal processing systems

Caaliph Andriamisaina*, Emmanuel Casseau**, Philippe Coussy*
* *LESTER Lab.- FRE CNRS 2734, Université de Bretagne Sud, France*
** *R2D2 - IRISA Lab. - UMR CNRS 6074, Université de Rennes1, France*

Abstract

In this paper, we propose a design methodology for implementing a multimode (or multi-configuration) and multi-throughput system into a single hardware architecture. The inputs of the design flow are the data flow graphs (DFGs), representing the different modes (i.e. the different applications to be implemented), with their respective throughput constraints. While traditional approaches merge DFGs together before the synthesis process, we propose to use ad-hoc scheduling and binding steps during the synthesis of each DFG. The scheduling, which assigns operations to specific time steps, maximizes the similarity between the control steps and thus decreases the controller complexity. The binding process, which assigns operations to specific functional units and data to specific storage elements, maximizes the similarity between datapaths and thus minimizes steering logic and register overhead. First results show the interest of the proposed synthesis flow.

Keywords: Flexible devices, high-level synthesis, multimode systems.

1. Introduction

The increasing demand for high performance and reconfigurability of embedded systems has led to the research on efficient hardware devices to adapt rapidly to changing environments. Field programmable gate arrays (FPGAs) provide partial reconfiguration at runtime but they have a weakness for performance. Furthermore, FPGAs require too long reconfiguration times to rapidly changing applications.

Multimode or multi-configuration cores are specifically designed for a set of time-wise mutually exclusive user-specified applications and target conventional hardware technologies. Multimode architectures are typically dedicated to multi-standard applications, for example applications implementing one algorithm which parameters can be changed like in a mobile system a Viterbi decoder for the channel decoding part whose constraint length and code rate may be changed from one communication standard to another one.

In this paper, we propose a methodology to implement multimode systems based on high-level synthesis. The starting point of the design flow is a set of behavioural level time-wise mutually exclusive specifications. From the set of data flow graphs representing the different modes, a single hardware architecture implementing these specifications is generated using high-level synthesis.

The paper is organized as follows: section 2 presents related work around high-level synthesis and multimode system design. Section 3 focuses on the problem formulation and section 4 presents our design flow. Experimental results based on illustrative examples are presented in section 5.

2. Related work

In order to design a multimode architecture, a designer, thanks to its own knowledge and experience, can identify similar properties and computation patterns between each mode and handcrafts multi-mode architectures. Such efficient flexible design examples can be found in [1-3].

To automate the process, methodologies that generate hardware architectures from behavioral specifications have to be used. High level synthesis (HLS) is analogous to software compilation transposed to the hardware domain [4, 5]. HLS tools automate the design process that generates a register transfer level architecture from an algorithmic behaviour of the specification. Single mode HLS design experiments can be found in [6, 7]. It is worth noticing that HLS basics were developed 15 years ago but HLS tools were not mature enough. HLS tools are being used in practice only at present time.

Applying HLS basics to generate multimode architectures seems natural. Four kinds of approaches (figure 1) can be identified: graph merging, datapath merging, inter-mode resource sharing during the binding process and joint scheduling.

The graph merging technique consists in merging graphs before the main synthesis steps (allocation, scheduling and binding). In [8], the focus is to merge several signal flow graphs (SFG) by using a locality based search algorithm such as simulated annealing or iterative refinement, in order to generate a multifunctional data path. In this approach, it is assumed that the number of multiplexers is proportional to the number of edges and thereby the number of edges is used as a measure of the cost of an

operator assignment. Thus, the used algorithm focuses on minimizing the number of edges by sharing as many as possible edges in the different SFGs. In [9] an ILP algorithm is used to merge several graphs. This approach focuses on maximizing an edge gain, defined as the difference between the cost of the nodes before merging and the cost of the resulting node after merging, as well as an interconnection gain, based on the multiplexer cost decrease. A multifunctional datapath is also generated. A similar approach has been proposed in [10]. However, the graph merging algorithm is different. This algorithm begins by constructing a compatibility graph, for a pair of DFGs, whose nodes are previously weighted by a cost function similar to the one defined in [9]. Afterwards, a search of the maximal weighted clique is performed and finally, the resultant graph is constructed from this clique. Moreover, the operation commutativity is taken into account, which allows to increase the number of compatible operations to be merged and thereby limits the number of multiplexers to add. In this approach, the generated structure is a data flow graph (DFG).
Although all these graph-merging algorithms could be efficient for the area reduction, they cannot be used when several tasks have to respect different throughput constraints. The register merging and the datapath generation are not realized: the architecture cost (controller, registers…) is thus not taken into account.

In [11] and [12], an approach that consists in unifying scheduled DFGs during the binding step has been proposed. In [11], both datapath and controller generation is presented. The first step consists in scheduling each DFG under a latency constraint. In the second step, a control step (c-step) matching across the scheduled DFGs is performed by using a maximal weighted matching. This matching allows to maximize the same-c-step component usage, thus decreasing the functional difference between the different controllers. Afterwards, the DFG that lastly updated the resource allocation is assigned by using a binding algorithm developed in [13]. Finally, the others DFGs are assigned trying to minimize hardware resource overhead. In [12], the methodology begins by the scheduling of each DFG under the given timing/resource constraints. Then the scheduled DFGs are concatenated (chained) into a single DFG. Finally, the resource binding of the concatenated DFG is performed using the maximal weighted bipartite matching algorithm. This technique allows to reduce the binding complexity. However, the authors did not take into account the effect of the proposed methodology on the controller area.

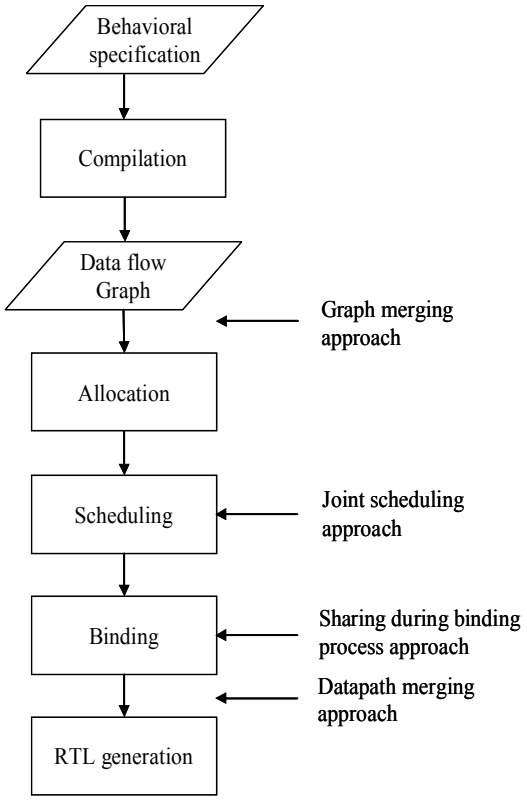

Figure 1. High-level synthesis flow and multimode system design approaches

The generation of a multifunction loop accelerator has been developed in [14]. This method applies the scheduling and binding process to kernel loops in order to generate datapaths for each kernel loop. These datapaths are then merged, using an ILP method, to provide a multifunction loop accelerator. A datapath merging method based on the graph merging algorithm defined in [10] is also developed in [15].

The joint scheduling method of several graphs has been mentioned in [14] but the authors gave few details. The proposed approach consists in jointly scheduling loop bodies by choosing a scheduling decision that maximizes the hardware resource sharing across loops. An ILP is used to solve this problem. The limit of this approach is the combinatorial explosion that related to the DFG complexity increase. The use of exact methods like ILP for the joint scheduling is limited to small size graphs.

In this paper, we focus on digital signal and image processing applications. These applications have often as main constraint the throughput to satisfy a real time constraint. Graph merging approaches are efficient to decrease area but they cannot support the synthesis under different throughput constraints. Except [14] and

[15], the approaches described above take into account only area and latency constraints. Apart from [11] other works only consider datapath area. Datapath merging based approaches can be improved taking into account the functional similarities across the control steps. In current designs, the cost of the control part is ever growing. This trend becomes critical for multimode systems.

In this paper, we propose an approach that aims to reduce both datapath and controller area of multimode systems. The approach supports modes with different throughput constraints. It is based on an ad-hoc scheduling algorithm that improves the functional similarities across the c-steps of the DFGs in order to reduce the controller complexity and steering logic.

3. Problem formulation

Let us consider two tasks represented by their DFGs: DFG1 in figure 2(a) and DFG2 in figure 2(b). We assume these tasks are time-wise mutually exclusive, that is, they can not be executed at the same time. The executing condition for each task is determined by the "mode" value.

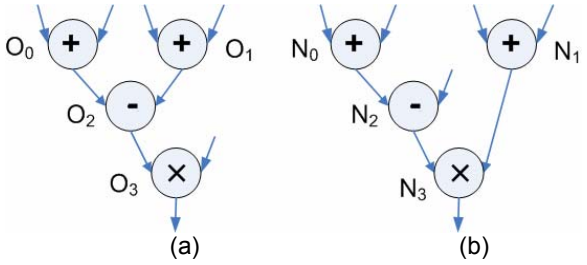

Figure 2. (a) DFG1, (b) DFG2

Let us schedule DFG1 and DFG2 using a list scheduling [16] under a throughput constraint. Supose the adder and the subtractor latency is 1 cycle, the multiplier latency is 2 cycles. If the throughput constraints of both DFGs set an output sample every 2 cycles, then 2 pipeline stages delimitated by the bold lines in figure 3 are required.

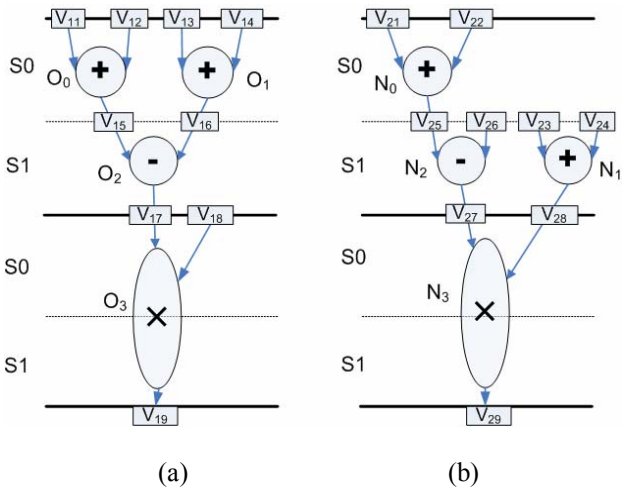

Figure 3. (a) DFG1 scheduling, (b) DFG2 scheduling

We consider that no multifunction operator is available. Thereby 2 adders, 1 subtractor and 1 multiplier are required for DFG1 scheduling. At least 1 adder, 1 subtractor and 1 multiplier are required for DFG2 scheduling. Assuming that no register sharing is performed during the register allocation of each scheduled DFG, 9 registers are required for each architecture. Finally if the both tasks are implemented in two different hardware architectures, 2 multipliers, 3 adders, 2 subtractors and 18 registers are required for the datapath.

Using a multimode architecture, the number of functional unit can be reduced to 1 multiplier, 2 adders, 1 subtractor and the number of register number can be 9. That is to say the architecture area decrease can be very interesting as well as the power consumption saving.

However, extra cost is to be considered. Resource and register sharing between two time-wise mutually exclusive graphs involve:

1) extra control logic owing to the load commands of the merged registers,

2) extra steering logic in the datapath basically owing to the multiplexer generation associated with the resource sharing (functional unit and registers).

However this extra control logic increase can be overcome by making a suitable scheduling. In the examples in figure 3, if O_0 and N_0 share a single adder (adder1) and O_1 and N_1 share another adder (adder2)[1], an efficient design implies the sharing of the left input register of O_0 (respectively O_1) and N_0 (respectively

[1] With the set of functional units allocated for this multimode architecture, if a single adder is used to implement DFG2, the control unit and extra steering logic are more costly.

N_1) to a single register R_1 (respectively R_3) and their right input registers to R_2 (respectively R_4) as shown in figure 4. If we assume that the input variables of O_0 (respectively O_1) and N_0 (respectively N_1) come from the same source, no multiplexer is necessary during the register allocation.

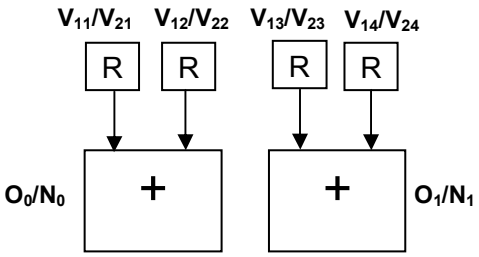

Figure 4. Partial multimode datapath

The merged register load command is by definition the OR logic function of the original single mode register load commands. In our case, the load command is the combination of the c-step value and the "mode" value. Examples in figure 3 have 2 states each other, S0 and S1[2]. Therefore a unique controller with 2 states (S0 and S1) is enough to control these two tasks.

Figure 5 represents the register load commands of both adders. Mode0 (mode) refers to the task represented by DFG1 and mode1 (not mode) refers to the one represented by DFG2. Actually ld_R_1 and ld_R_2 do not require more logic gates whereas ld_R_3 and ld_R_4 generate extra logic gates (figure 5c).

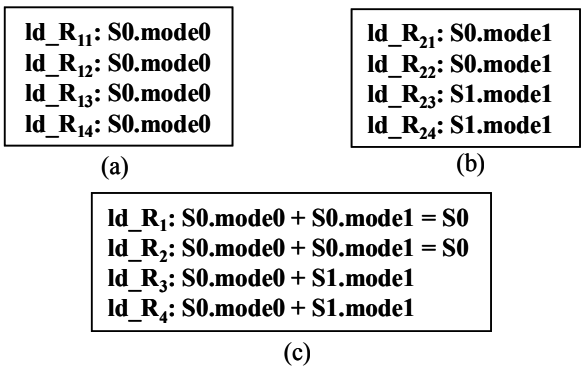

Figure 5. (a) Original register load commands with mode0, (b) Original register load commands with mode1, (c) Merged register load commands

[2] In these cases, two pipeline stages execute in parallel. Final state *Si* is thus the combination of the sub-states *Si* associated with each pipeline stage.

Let us now consider another approach. DFG1 is still scheduled using a list scheduling as shown in figure 3(a). DFG2 is now scheduled considering the number of each type of operations per c-steps in DFG1 and trying to maximize the similarity between compatible operations[3] per c-steps in DFG1 and DFG2. Thereby operation N_1 in DFG2 is scheduled in c-step S0 rather than in c-step S1. Figure 6 shows the register load commands if the functional unit and register sharing is the same as the one described in figure 4. In that case no extra logic gate is required for the merged register load commands (figure 5e).

These two basic examples show that the complexity of a multimode architecture does not only depend on the number of functional and memorization units but also on the scheduling that drives the complexity of the extra control cost. Ad-hoc scheduling, based on c-steps similarity, is of major interest.

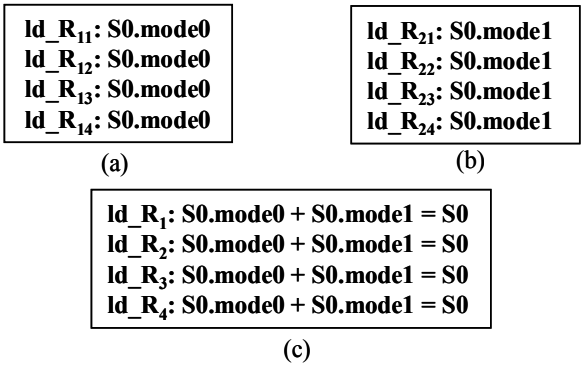

Figure 6. (a) Original register load commands with mode0, (b) Original register load commands with mode1, (c) Merged register load commands

4. Design flow

The proposed design flow is presented in figure 7. The starting point is the several DFGs with their respective throughput constraints. The first step aims to identify the "main" DFG. First for each DFG the number of compatible operations is computed. The DFG which has the greatest ratio between its number of compatible operations and its throughput constraint is considered as the main DFG. The same rule is applied for ordering the other DGFs (secondary

[3] Compatible operations are operations that can be executed using the same type of operator (example 1: additions are compatible each other because they can be executed by an adder; example 2: additions and subtractions are compatible if adder/subtractor multifunction operators are allocated).

DFGs)[4]. The main DFG is scheduled first by using a list scheduling under a throughput constraint. A resource reservation table is obtained from this scheduling. For each c-step, this table states if an operator has been used. The scheduling of the next DFGs, ordering according to their own ratio, takes into account the throughput constraint and the resource reservation table of the main DFG (respectively the updated resource reservation table that comes from the previously scheduled DFGs). Appendix details the scheduling algorithm. Using the resource reservation table as a constraint aims to benefit from the similarities between the c-steps of the DFGs. It permits to decrease, as discussed in section 3, extra control logic.

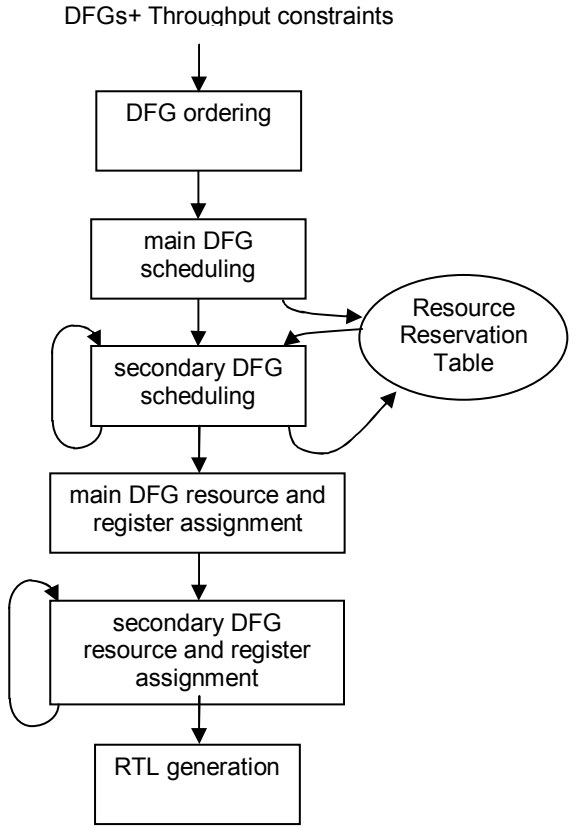

Figure 7. Proposed design flow

When all DFGs have been scheduled, operations and variables of the main DFG are bounded to functional units and registers. The next step is the functional unit and register assignment of the secondary DFGs using the bipartite weighted matching [13]. Compatible operations are first bounded to the resources required for the previously DFGs trying to minimize multiplexor cost. Then other operations (non compatible operations) are bounded to other resources. The last step consists in generating the register transfer level (RTL) description of the multimode architecture.

5. First experiments

5.1 Illustrative example

Let us take two basic computations to illustrate the proposed approach:

1) x=((a+b)*(c-d)+(e*f)-shr(g,h))*(i+j) represented by DFG1
2) y=((a*b)+(c-d)+(e+f))*((g+h)*(i-j)) represented by DFG2.

Compatible operations between DFG1 and DFG2 are multiplications, additions and subtractions. DFG1 has 8 compatible operations whereas DFG2 has 9 ones. Assuming the throughput constraint is the same for both DFGs. Thus DFG2 is said to be the main DFG. Hence it is scheduled first using a list scheduling under a throughput constraint. We assume the adder and the subtractor latencies are 1 cycle, the multiplier and the shifter latencies are 2 cycles. Assuming the timing constraint sets an output sample every 2 cycles, 3 pipeline stages are required with 2 c-steps S0 and S1 (figure 8a). Table 1a depicts the resource reservation table of DFG2. 3 multipliers are required for S0 c-step as well as for S1, 2 adders are required for S0 and S1, and 1 subtractor is required for S0 and S1.

The scheduling of DFG1 is done by respecting its throughput constraint and DFG2 reservation table. Assuming the throughput constraint is 2 cycles, 4 pipeline stages with 2 c-steps are required. Inside each pipeline stage, macro scheduling is performed to maximize similarities between DFG1 c-steps and DFG2 c-steps and trying to avoid new resource allocation. The resulting scheduling is presented in figure 8b. In this case, except the shifter, *i.e.* a non compatible operator, no more functional unit is required. The resource reservation table is then updated (table 1b).

After the scheduling, operations of DFG2 are first bounded to the allocated resources. Then compatible operations of DFG1 are bounded to the resources used for DFG2 mapping. Afterwards, other DFG1 operations are then bounded to the other resources. Finally, the register transfer level (RTL) description of the multimode architecture is generated.

[4] The more this ratio is important the greater the inter-mode functional unit sharing

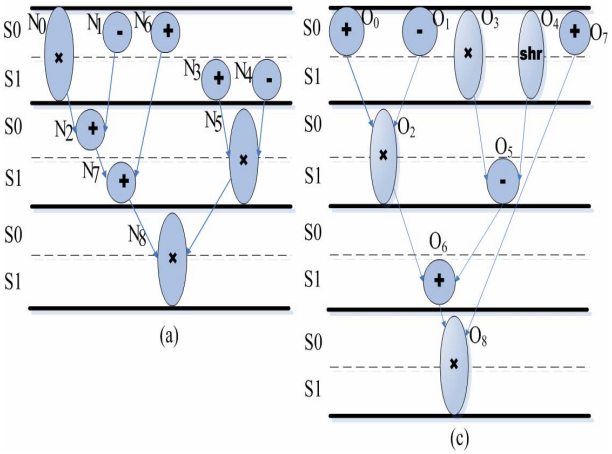

Figure 8. (a) List scheduling of DFG2, (b) Scheduling of DFG1

a)

	Resources		
States	*	+	-
S0	3	2	1
S1	3	2	1

b)

	Resources			
States	*	+	-	shr
S0	3	2	1	1
S1	3	2	1	1

Table 1. (a) Resource reservation table of DFG2, (b) Updated resource reservation table

Equations 1 and 2 have been synthesised to evaluate our approach. Two different methods were also completed. The first one consists in implementing the two tasks in two separate hardware architectures (single mode architectures) that execute in parallel. Their areas were then added. The second one consists in applying the graph merging method before synthesis. This latter is known to be very efficient for area optimization.

We used a Xilinx Virtex-II Pro XC2VP100 FPGA as a target technology and ISE 7.1 logic synthesis tool from Xilinx. Area results including both datapath and controller unit are presented in table 2. The area improvement is 34% compared to the single mode method. Compared to the less area costly single mode architecture, the area overhead of the multimode architecture is 15%. The proposed approach and the graph merging method are similar.

Architecture	Single mode	Graph merging	Our approach
Cycle time (ns)	21	21	21
Area (slices)	748	445	444

Table 2. Synthesis results

5.2 FFT synthesis

With equations 1) and 2), the operation dependencies between the two graphs are almost similar. In this section, computations whose dependencies differ a lot are investigated.

The Discrete Fourier Transform (DFT) is a digital signal processing basic computation. The following equation describes the DFT of a N-points sequence *x(n)*:

$$X(k) = \sum_{n=0}^{N-1} x(n) \times W_N^{k.n} \quad , \quad k = 0,1, \ldots, N-1$$

where $W_N = e^{-j2\pi/N}$ is called the *twiddle factor*.

An Fast Fourier Transform (FFT) algorithm is usually used to reduce the computation complexity of the DFT which requires N^2 operations where N is the transformation size. FFT algorithms actually reduce the DFT complexity from N^2 to $N.log_2N$.

Two kinds of FFT algorithm can be used. The first one is called decimation "in time" (DIT) FFT and the second one is called decimation in frequency (DIF) FFT.

The synthesis of a multimode architecture that implements both kinds of FFT algorithms has been performed to experiment our approach. A 4-points real FFT was chosen. With this basic FFT, 4 additions, 4 subtractions and 4 multiplications are required no matter the kind of the FFT algorithm. However, the dependencies differ completely between these two kinds of FFT [9].

For our experiments, we have chosen a timing constraint of 3 cycles and no multifunction resource has been used. As we did before, a first synthesis was performed considering the two kinds of FFT separately (synthesis of 2 single mode architectures). With this timing constraint, 8 multipliers, 4 adders, 4 subtractors are allocated considering the combined datapath. The multimode architecture requires 4 multipliers, 2 adders and 2 subtractors.

A Xilinx Virtex-II Pro XC2VP100 FPGA as a target technology and ISE 7.1 logic synthesis tool from Xilinx were used. Area results including both datapath

and controller unit are presented in table 3. An area improvement of 36% compared to the single mode architecture is obtained. The areas of both single mode architectures are actually almost similar (≅ 740 slices). Compared to one of this single mode architecture, the area overhead of the multimode architecture is only 20% whereas both FFT kinds of algorithms can be considered with the latter.

Architecture	Single Mode	Multimode
Timing cycle (ns)	31	31
Area (Slices)	1472	934

Table 3. FFT DIT-DIF single mode architecture versus multimode architecture

6. Conclusion and work in progress

We have presented a high-level synthesis based design flow to automate the design of multimode systems. The approach is based on an ad-hoc scheduling to limit extra control logic owning to resource and register sharing between time-wise mutually exclusive graphs. Based on a first data flow graph list scheduling, an Control Similarity Based List Scheduling is performed for the other DFGs by considering as scheduling constraints both the user specified throughput constraint and the resource reservation table of the previously scheduled DFGs.

First experiments with basic computations give promising results for graphs with similar operations dependencies as well as for graphs where dependencies differ a lot. Work in progress focuses on the synthesis of more complex DSP computations. Binding algorithm improvements are also investigated.

7. References

[1] I. Krikidis, J.L. Danger, L. Naviner "An iterative reconfigurability approach for WCDMA high-data-rate communications", *IEEE Wireless Communications*, June 2006.

[2] E. Piriou, C. Jego, P. Adde, M. Jezequel, "A flexible architecture for block turbo decoders using bch or reed-solomon components codes", *ISVLSI*, Mars 2006.

[3] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol and R. Yan, "A unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18-µm CMOS", *IEEE Journal of Solid-State Circuits*, 2002, pp. 1555-1564

[4] D. D. Gajski, N. D. Dutt, Allen C-H. Wu, Steve Y-L. Lin, High-Level Synthesis: *Introduction to Chip and System Design*, Kluwer Academic Publishers, Boston, MA, 1992.

[5] S. Gupta, R. Gupta, N. Dutt, A. Nicolau, "SPARK : a parallelizing approach to the high-level synthesis of digital circuits", *Ed. Springer*, 2004, ISBN 1402078374.

[6] E. Casseau, B. Le Gal, P. Bomel, C. Jégo, S. Huet, E. Martin, "C-based rapid prototyping for digital signal processing", Proceedings of the EUSIPCO, 2005

[7] http://deepchip.com/items/else06-08.html

[8] Van der Werf, M.J.H Peek, E.H.L. Aarts, J.L. Van Meerbergen, P.E.R. Lippens, W.F.J. Verhaegh, "Area optimization of multi-functional processing units", *Proceedings of the ICCAD*, 1992.

[9] W. Geurts, F. Catthoor and H. De Man, "Quadratic Zero-One Programming-based synthesis of application-specific data paths", *Proceedings of the ICCAD,* 1993.

[10] N. Moreano, E. Borin, C. de Souza and G. Araujo, "Efficient datapath merging for partially reconfigurable architectures", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2005.

[11] V. V. Kumar and J. Lach, "Highly flexible multi-mode system synthesis", *Proceeding of CODESS+ISSS*, 2005

[12] L. Chiou, S. Bhunia and K. Roy, "Synthesis of Application-Specific Highly Efficient Multi-mode Cores for Embedded Systems", *ACM Transactions on Embedded Computing Systems*, February 2005

[13] C. Huang, Y. Chen, Y. Lin and Y. Hsu, "Data path allocation based on bipartite weighted matching", *Proceedings of the Design Automation Conference*, 1990, 499-504.

[14] K. Fan, M. Kudlur, H. Park and S. Mahlke, "Increasing Hardware Efficiency with Multifunction Loop Accelerators", *Proceedings of the CODESS+ISSS*, October 2006

[15] Z. Huang, S. Malik, N. Moreano and G. Araujo, "The design of dynamically reconfigurable datapath coprocessors", *ACM Transactions on Embedded Computing Systems,* 2004

[16] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994

Appendix : Control Similarity Based List Scheduling (CSBLS) algorithm

Definitions

Scheduling delay of operation ($\delta(O_i)$): allowable delay of operation O_i before its scheduling.

Ready operations of type "type" in state "cstep" ($RO_{cstep,type}$): set of operations whose ancestors are already scheduled.

Time frames of operations O_i ($\mu(O_i)$): difference between the as soon as possible time of O_i and the current step "cstep".

C-step similarity between the previously scheduled DFGs (pDFG) and the currently scheduled (secondary) DFG (sDFG) (CS(type,cstep,pDFG,sDFG)): difference between the number of resource type "type" at state "cstep" of the previously DFGs and the secondary DFG.

Algorithm :

Inputs :

- a secondary DFG (sDFG)
- the timing constraint of the sDFG (T_s)
- the resource reservation table of the previously scheduled DFGs (pDFG)

Output :

- a scheduled sDFG

Begin
cstep=0;
$\delta(Oi)=0, \forall O_i \in O$ (O: set of operations in the sDFG);
repeat until (O_n is scheduled)
for each resource type
Determine $RO_{cstep,type}$;
Compute $\mu(O_j), \forall O_j \in RO_{cstep,type}$;
Compute CS(type,cstep,pDFG,sDFG);
while (CS(type,cstep,pDFG,sDFG) > 0 or card ($RO_{cstep,type}$) > 0)
Schedule O_j by increasing time frames;
end while;
if (CS(type,cstep,pDFG,sDFG) ≤ 0 and card ($RO_{cstep,type}$) > 0) *then*
if ($\delta(O_j) < T_s$-1) *then*
Delay O_j *until* (card ($RO_{cstep,type}$) = 0);
else
Schedule O_j;
end if;
end if;
end for;
cstep++;
if (cstep=last_step) *then* cstep=0 *end if*;
end repeat;
end;