AUTOMATED MULTIMODE SYSTEM DESIGN FOR HIGH PERFORMANCE DSP APPLICATIONS

Bertrand LE GAL[1] *and Emmanuel CASSEAU*[2]

1. IMS Laboratory, CNRS UMR 5218 - 351 Cours de la Libération, 33405 Talence, France
email : bertrand.legal@ims-bordeaux.fr
2. INRIA/IRISA – ENSSAT Université de Rennes1, 6 Rue de Kérampont - 22305 Lannion, France
email: emmanuel.casseau@irisa.fr

ABSTRACT

In a mobile society, more and more devices need to continuously adapt to changing environments. Such mode switches can be smoothly done in software using a general purpose or digital signal processor. However hardware components are required to cope with throughput and power constraints. Reconfigurable hardware technologies like FPGA offer partial reconfiguration at run-time but require too long reconfiguration times to rapidly changing applications. In this paper we propose an automated design flow to implement multiple configuration and multi-constraint systems into a single circuit using conventional hardware technologies.

1. INTRODUCTION

In many applications, such as personal computers or high-performance computing, the main focus of processor design is to increase the computing performance and efficiency. Fast growing increase in the development of DSP, multimedia and communication applications causes the widespread research on efficient platforms for integrating complex systems on a chip. General purpose processors contain major limitations for DSP applications like memory latencies, limits in parallelism exploitation, i.e. performance, and power consumption. ASIC allows designers to optimize the hardware for one or more parameters. However due to lack of flexibility, ASIC can not cope with the evolving standards and applications of today's world. FPGA provides flexibility but at the cost of large performance, area, power and reconfiguration time penalties. So the search for flexibility and performance at the same time is the requirement in digital design.

Keeping in view the requirement of digital design, multimode systems are proposed to realize a set of selected configurations into a single system. The main intention of multimode systems is to implement multiple configurations or modes using conventional hardware technologies. Although the reconfigurability of a multimode system is not as much much high as of FPGA but on the fly it can be reconfigured to one of the set of configurations for which it is designed for. Multimode systems provide both reconfigurability and efficiency in terms of area, performance, power consumption and reconfiguration time.

One of the goals of multimode system design is to minimize area by reusing hardware resources effectively among different configurations. High-level synthesis (HLS) is an automated process that generates a register transfer level (RTL) architecture from an algorithmic specification and user defined constraints [1, 2, 3]. Conventional scheduling and binding algorithms used in HLS can accomplish resource sharing efficiently. The main idea of this paper is to make use of such algorithms in the synthesis of multimode systems to share hardware resources among the configurations.

The paper is organized as follow. Section II presents the advantages and the related work on multimode architecture design. An overview of the multi-constraint multimode system design flow we propose is presented in section III. Section IV detailed our high-level synthesis process. Experimental results are reported in section V. Finally conclusions are presented in section V.

2. INTRODUCTION TO MULTIMODE SYSTEMS

Hardware implementation of a system (SoC for system on chip) is becoming more and more popular. However in most of SoC implementations, component utilization is low due to the high number of dedicated cores which are used in a timewise mutually exclusive way. For example with a multi-standard decoder circuit (GSM, EDGE, etc.) only one standard is executed per time frame, so only one part of the circuit is used at a particular time. One way to increase the component utilization is to design a multimode architecture i.e. a single architecture to execute several applications (so called configurations or modes) as shown in figure 1.

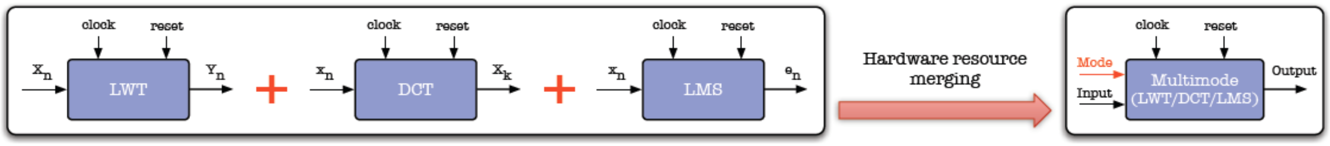

Figure 1 - Multimode architecture approach

A multimode architecture can be obtained through designer knowledge and experience to identify similar patterns in different configurations and design the circuit in such a way to utilize the similar patterns again and again for the different configurations. In [4] Kumar et al. proposed an HLS-based approach to automate the design. This technique makes use of small scale reconfigurability along with high-level synthesis of multimode systems. In small scale reconfigurability the chip area is divided into fixed logic and reconfigurable logic areas. High performance along with reconfigurability is achieved by implementing most of the circuitry in the fixed logic and only small portion in the reconfigurable area. Although the design is not fully reconfigurable like FPGA, it incorporates the reconfigurability up to the desired extend. To cope with the hardware reconfigurability required by the previous approach, Chiou et al. proposed SPACT approach [5]. The different modes are represented by data-flow graphs (DFGs). Each DFG is scheduled separately under its own given resource or timing constraint. The scheduled DFGs are then concatenated into a single DFG. Finally this DFG is bound to resources. This technique leads to low complexity binding. However because modes are scheduled separately, similarities between configurations are not taken into account. Sharing cost (interconnection resources like multiplexers, and controller) can be improved. Approach proposed by Chavet et al. in [6] aims at improving SPACT approach for timing constrained modes. DFGs are scheduled consecutively. Scheduling algorithm aims at maximizing the similarities within control steps of each mode depending on the previously scheduled DFGs to minimize the controller complexity. Operations are then bound to arithmetic resources in such a way to minimize resource sharing cost.

Our proposed approach aims at handling timing as well as resource constraint modes. Previous synthesis methodologies perform the scheduling step and then try to optimize the binding step. However, we believe that processing the binding step after the scheduling is completed is not a good solution to reduce the sharing cost: scheduling decisions would not have been taken if the actual interconnect cost has been known. To limit extra sharing cost, we developed a join ad-hoc scheduling and binding algorithm based on similarities between datapaths. The aim is not to focus on the minimization of the arithmetic resources but the overall architecture, including registers and interconnection resources.

3. DESIGN FLOW OVERVIEW

The multimode system design approach is presented in Figure 2. Inputs are the Matlab behavioural descriptions which specify the behaviour of the applications to implement, and their constraint. Our approach makes it possible to generate multimode architectures that can support different kinds of constraints. For example, if we have two configurations, one can be optimized for area and the other for a particular throughput. Each behavioural description is first transformed into a formal representation model. Usually DSP applications are regular and predictable. The modelling of such applications is generally performed using *Data Flow Graph (DFG)* or *Signal Flow Graph (SFG)* models [1;2] which clearly exhibit the data dependencies. In the proposed design flow, the behavioural description is compiled to a SFG as internal representation. Loops are unrolled and conditional structures are flattened. The intrinsic parallelism among operations can thus be easily exploited so that real time constraints (throughput) can be satisfied. Then a 3-step process is performed:

1. The first step performs the analysis of the applications. Each configuration (couple application-constraints) is analysed independently. According to the target hardware technology, delays can be associated to each operation of the internal representation (SFG) of a particular mode. Thus, for each mode, node mobility is found out by calculating the as soon as possible (ASAP) and, for timing constrained modes, as late as possible (ALAP) execution times.
2. The second step is dedicated to the graph model merging. All of the annotated graphs are merged into a single unified signal flow graph. Timewise mutually exclusive relationship between the different modes is implemented by a conditional node statement to emphasis mutually exclusive branches.
3. The third step carries out the synthesis process of the unified SFG. High-level synthesis is used to formally transform this graph into a hardware architecture. HLS not only meets all the constraints corresponding to the different applications but also try to minimize the total cost of the multimode architecture.

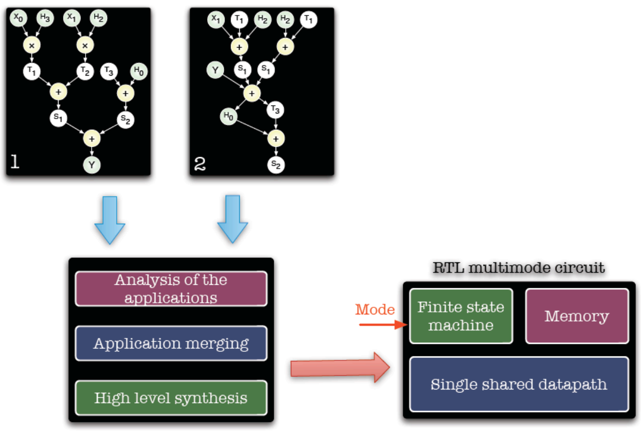

Figure 2- Design flow overview from application to circuit.

At the end of the process a VHDL Register Transfer Level description of the architecture implementing all the applications under constraints is generated. The architecture is composed of three main units : the *data-path unit* which performs the computations of the applications, the *memory unit* for the data storage and the *controller unit* (FSM) which controls the behaviour of the circuit depending on the selected mode.

4. HIGH-LEVEL SYNTHESIS PROCESS

The high-level synthesis design flow is shown in figure 3. From the unified graph model the synthesis process starts with the selection of the hardware operators using a technology-specific characterized library. Then allocation step is performed. According to the average number of operations per cycle, the allocation step defines the minimum number of arithmetic operators required for each kind of operation in order to satisfy the designer timing constraints. In the case of a multimode architecture, every resource can be shared without distinction of mode so the number of arithmetic operators is optimized at this step. Multimode system operation scheduling is then performed. Conventional high-level synthesis methodologies perform the scheduling step and then try to optimize the binding step. Sharing of resources increases along with the number of modes. To limit extra cost (muxes and controller) due to this increased resource sharing, in our approach scheduling and binding steps are performed concurrently and associate to each scheduled operation an arithmetic operator, registers and interconnection resources. Register sharing is performed on the fly.

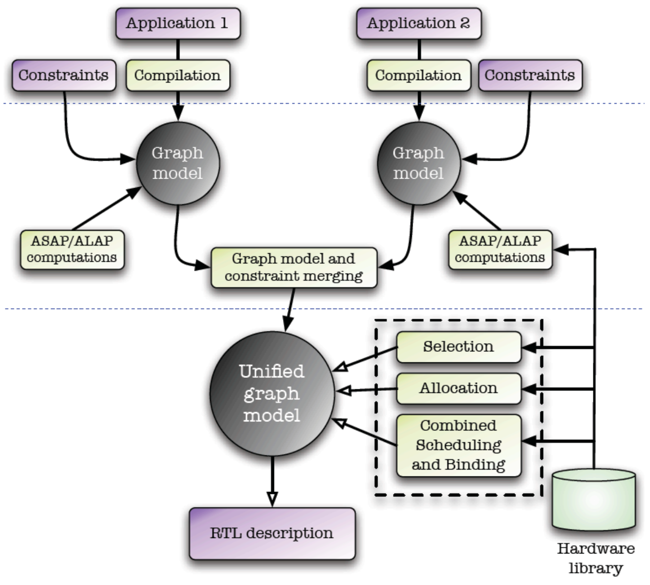

Figure 3 - Multimode system design flow

Figure 4 shows an overview of the combined scheduling and binding algorithm. Compare to previous approaches, the main differences are 1) modes are not scheduled consecutively but concurrently, 2) scheduling decisions depend on on the fly binding cost evaluation 3) the binding of an operation and its associated resources (interconnections and registers) is performed as soon as this operation has been selected for scheduling. The aim is to minimize the overall architecture, including registers and interconnection resources while satisfying the designer constraints.

From the unified graph model, the list of operation nodes to be scheduled is first extracted (line 1 (L1)). Each clock cycle, when nodes can be scheduled, the process is the following. Timing constrained modes are first processed (L4). Ready nodes[1] are sorted in such a way nodes that can not be delayed (zero mobility nodes) are first processed then other ready nodes are sorted based on the list-scheduling algorithm [2] using mobility as priority function. Highest priority node is selected disregarding its mode (L5). The binding cost of this node is computed over available resources. Binding cost includes the *operator cost* and the *path cost* (interconnection and register resource cost as well as controller cost). Operator with lowest cost is selected (L6) based on the minimal weighted matching algorithm proposed in [7] for logic synthesis. In our case, weight of edges is the sharing overcost. Now, at this step of the process, aim is to reuse this particular operator at this particular clock cycle among the other modes in order to benefit from the multimode architecture approach. Thus, for every other mode, including resource constrained modes, the cost of binding ready nodes to the selected operator is computed (L7). For each mode, node with lowest cost is extracted and scheduled (L8). Once again, minimal weighted matching algorithm is used. The set of nodes is then bounded to the selected operator (L9) and interconnection resources and registers are updated or allocated if required. This process (L5-L11) is carried out as long as ready nodes of timing constrained modes still remain and arithmetic resources are available. Finally if arithmetic resources are still free, remaining ready nodes of resource constrained modes can be processed. Aim is to benefit as much as possible of available arithmetic resources. Same multimode combined scheduling and binding process (L5-L11) is performed until no more resource is available at current clock cycle. Then next clock cycle can be processed. The combined scheduling and binding process ends when every operation node from all of the modes has been processed.

```
SchedulingAndBinding( Graph g, ListOpr res )
01. NodeListByMode ln = SortNodeByMode( g );
02. While hasNodeToSchedule( ln ) then
    |
03. | // Processing time and resource constrained modes
04. | While hasWaitingTimingConstrainedNode( ln )
    | |                        or FreeResource( res ) then
05. | | Node     n     = GetHighestPriorityNode ( ln );
06. | | Operator o     = GetBestFreeOperator( n, res );
07. | | ListNode nodes = SearchBestCompatibleNodes( n, o );
08. | | ScheduleNodes( nodes, clock_cycle );
09. | | LinkNodesToResourcesAndCreatePaths( nodes, o );
10. | | RemoveNodes( ln, nodes);
11. | | SpecifiedOperatorAsUsed( res, o );
12. | End while
    |
13. | FreeResources( res );
14. | clock_cycle += 1;
15. End while;
```

Figure 4 - Combined scheduling and binding algorithm overview

5. EXPERIMENTS

To evaluate the effectiveness of the proposed methodology, several experiments were made. These experiments aim to compare our approach with:

[1] A ready node represents an operation which can be scheduled, i.e. whose predecessors have already been scheduled.

- a conventional approach which performs the high-level synthesis of each configuration independently. In this case, one architecture per mode is produced, i.e. there is no sharing of resources among the modes. We call it cumulative approach (CA).
- the synthesis of a multimode architecture based on SPACT approach [5].

Results were obtained using the design flow we presented including graph analysis, graph merging and high-level synthesis process. For this we used GraphLab[2] tool.

5.1 Hardware resource allocation

Table 1 shows the number of allocated resources for several multimode architectures: a FIR filter with different number of taps, a FIR/LMS design, a DCT/LMS/LWT design. Cumulative, SPACT and our proposed approaches are performed. Syntheses are constrained to get same throughput. This experiment allows to focus on the resource sharing efficiency because the control cost is not included.

Variable tap-length FIR filter experiment shows the benefits of a multimode architecture. Compare to the cumulative approach which requires a dedicated architecture per mode, the number of arithmetic resources and registers (REG) is small whatever the multimode approach. However, resource sharing among the modes involves extra cost like multiplexers. This number of multiplexers may become greater than the cumulative approach one if particular attention is not paid. Compared to SPACT approach, the number of multiplexers is reduced with our proposed approach because similarities between modes is taken into account.

Multimode	Design	ADD	SUB	MUL	DIV2	REG	MUX
FIR 8/16/32/64	FIR 8	4		4		16	4
	FIR 16	4		4		25	19
	FIR 32	4		4		42	46
	FIR 64	4		4		74	83
	C.A.	16		16		157	152
	SPACT approach	4		4		110	159
	Proposed appro.	4		4		76	86
FIR16 + LMS16	FIR 16	4		4		25	19
	LMS 16	4	1	4		45	77
	C.A.	8	1	8		70	96
	SPACT approach	4	1	4		65	100
	Proposed appro.	4	1	4		61	82
DCT8 + LMS16 + LWT8	DCT 8	8	8	8		37	48
	LMS 16	4	1	4		45	77
	LWT 8	4	4		4	16	14
	C.A.	16	13	12	4	98	139
	SPACT approach	8	8	8	4	65	121
	Proposed appro.	8	8	8	4	57	119

Table 1 - Allocated resources

For the two other experiments, the arithmetic resource requirements are different. Specific types of operators, i.e. operators which can not be used by other modes, are necessary. Behaviour is also different. In that case, difference between the number of registers of the cumulative approach and multimode design approaches is less because data lifetime does not match so conveniently. Sharing cost is also greater compared to FIR filter experiment.

5.2 Logical synthesis results

We experimented two kinds of sets of modes targeting everyday signal and image processing an embedded system may implement:

1) one single application with different profiles. We have considered FIR filtering with various numbers of taps, Viterbi decoding with different constraint lengths according to software radio needs, and sum of absolute differences SAD for different macro-block sizes based on video standards.
2) different applications. We have combined transformations with their inverse like two-dimensional discrete cosine transform DCT and inverse DCT used in video processing systems, fast Fourier transform FFT and inverse FFT used in communication systems. We have also combined less similar applications like FFT and DCT, FIR and LMS filtering, SAD and sum of square differences SSD, DCT and local wavelet transform LWT and LMS filtering.

To analyse the overall multimode architecture efficiency, a logical synthesis was performed after the high-level synthesis using Synopsys Design Vision. We target ASIC standard cell CMOS 65nm technology from ST Microelectronics. For these experiments, word-length of resources was 16 bits except for Viterbi decoders where 12-bit resources only are required. Fixed-point format was used. Some applications have also been synthesized using IEEE-754 floating-point hardware resources.

Table 2 shows experimental results: cumulative approach area (number of gates), area and dynamic power consumption for SPACT approach and our approach. Area results correspond to the complete architecture area, i.e. the datapath including storage resources and its controller. We set heterogeneous synthesis constraints: timing constraint (T), resource constraint (R) and no constraint (N) that is to say an "as fast as possible" implementation using resources of other modes is targeted.

Compared to the cumulative approach, area decreases from 14% up to 68% (average saving 40%) with our approach. Best results are obtained when application behaviour is similar (FIR filtering). For configurations without obvious behaviour similarities like 2dDCT/FFT64 or LWT/DCT/FFT, results are still interesting. When a floating-point data representation is investigated, a multimode architecture is even more attracting. Architecture remains the same but area saving is increased because floating-point operators are more costly. Compared to SPACT approach, area saving is 9,4% on average and dynamic power consumption is reduced by about 12%. Actually, our approach takes similarities between modes into account. The number of input ports of the multiplexers is reduced on average and by the way the complexity of the decoding part of the controller is also decreased. This leads to area saving as well as power saving.

[2] http://www.enseirb.fr/~legal/wp_graphlab

	Modes	Synthesis constraints	Data format	Throughtput (Msample/s)	C.A. (gates)	SPACT approach		Proposed approach		% C.A.	% SPACT	
						Area (gates)	Power (mw)	Area (gates)	Power (mw)	Area saving	Area saving	Power saving
1	FIR 8/16/32/64	N/H/T/T	16 bits	32/20/13/8	14758	7026	2,59	5300	2,05	*64,1 %*	*24,6 %*	*20,9 %*
			IEEE-754		47529	19349	5,45	15474	4,56	*67,4 %*	*20,0 %*	*16,3 %*
	Viterbi 16/64/256	N/N/T	12 bits	26/7/2	131660	126592	44,70	113209	39,98	*14,0 %*	*10,6 %*	*9,8%*
		N/N/T	12 bits	53/22/7	179077	147731	52,17	130742	46,17	*27,0 %*	*11,5 %*	*10,9%*
		N/N/T	12 bits	53/40/13	224517	175511	61,98	154162	54,44	*31,3 %*	*12,2 %*	*11,1%*
	SAD 8x8 16x16	T/R	16 bits	7/2	2667	1470	0,68	1458	0,68	*44,9 %*	*0,8 %*	*0,3 %*
2	DCT - iDCT	T/T	16 bits	176/208	12217	7070	1,69	6906	1,64	*43,5 %*	*2,3 %*	*2,8 %*
			IEEE-754		52493	28750	4,37	28044	4,214	*45,2 %*	*2,5 %*	*3,6 %*
	FFT64 - iFFT64	T/N	16 bits	384/384	69997	46176	18,79	37232	12,16	*46,8 %*	*19,4 %*	*35,3 %*
			IEEE-754		272503	174525	56,15	140488	35,06	*36,0 %*	*19,5 %*	*37,6 %*
	2d DCT - FFT64	T/N	16 bits	256/512	68480	48315	13,12	45448	12,57	*29,4 %*	*5,9 %*	*4,2 %*
			IEEE-754		259645	178047	33,89	171520	32,82	*31,4 %*	*3,7 %*	*3,2 %*
	FIR16 - LMS16	T/T	16 bits	20/10	7008	4895	1,71	4621	1,63	*34,1 %*	*5,6 %*	*4,7 %*
			IEEE-754		23535	15154	3,7113	14629	3,6182	*35,6 %*	*3,5 %*	*2,5 %*
	SAD 8x8/16x16 SSD 8x8/16x16	N/T/N/T	16 bits	7/7/2/2	8181	3597	0,93	3389	0,88	*58,6 %*	*5,8 %*	*5,5 %*
	DCT8 / LWT8 / LMS8	R/N/T	16 bits	176/208/16	9300	7500	1,84	6664	1,49	*28,3 %*	*11,2 %*	*19,0 %*
			IEEE-754		44256	28752	4,091	28316	3,65	*35,0 %*	*1,5 %*	*10,8 %*
										39,6 %	*9,4 %*	*11,9 %*

Table 2 - Synthesis results

6. CONCLUSION

Multimode architectures, due to the good performance/area/power consumption trade-off they can achieve, have a strong interest for embedded devices. In this paper a methodology to automate the design of such architectures is proposed. It is based on high-level synthesis. The different configurations of the multimode architecture can be optimized for heterogeneous constraints (performance, area) which further increases the interest of the proposed approach. Multimode system design implies extra costs mainly due to the increase of the controller complexity and extra sharing resources. To limit these costs a combined ad-hoc scheduling and binding algorithm based on similarities between modes is used. Area decrease shows the effectiveness of our proposed approach. Area saving is about 40% compared to a conventional cumulative approach.

REFERENCES

[1]. J. P. Elliott, *Understanding Behavioral Synthesis. A Practical Guide to High-Level Design*, Kluwer Academic Publishers, 2000.

[2] S. Gupta, R. Gupta, N. Dutt, A. Nicaulo, *SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits*, Springer, 2004

[3]. *High-Level Synthesis: From Algorithm to Digital Circuit*, P. Coussy, A. Morawiec, (Eds), Springer, 2008

[4]. V.V. Kumar, J. Lach, “Highly flexible multimode digital signal processing systems using adaptable components and controllers”, EURASIP Journal on Applied Signal Processing, Volume, Issue 1, January 2006, pp.73-82.

[5]. L.-Y. Chiou, S. Bhunia, K. Roy, “Synthesis of application-specific highly efficient multi-mode cores for embedded systems”, ACM Transactions on Embedded Computing Systems, Volume 4, Issue 1, February 2005, pp.168-188.

[6]. C. Chavet, C. Andriamisaina, P. Coussy, E. Casseau, E. Juin, P. Urard, E. Martin, “A design flow dedicated to multimode architectures for DSP applications”, Int. Conf. on Computer-Aided Design, ICCAD, pp. 604-611, 2007.

[7]. C-Y. Huang, Y. Chen, Y. Lin, Y. Hsu, "Data path allocation based on bipartite weighted matching", Proc. ACM/IEEE Design Automation Conf. (DAC), pp. 499-504, 1990.