Design of multi-mode application-specific cores based on high-level synthesis

Emmanuel Casseau, Bertrand Le Gal

Abstract

In a mobile society, more and more devices need to continuously adapt to changing environments. Such devices require flexibility to satisfy increasing context-switching needs, that is to say to implement different algorithms at different times. When the specifications for any application are given, they can be executed on a software processor (i.e. on a general-purpose processor) or on a hardware processor (i.e. on an ASIC or on an FPGA). A software processor contains all the basic blocks needed to build any logic of mathematical function imaginable but the limitations are memory latencies and available parallelism. Performance and power consumption requirements cannot meet mobile device requirements. In this article we propose a methodology to map selected groups of DSP tasks to multi-mode cores using conventional hardware technologies.

1. Introduction

Mobile devices require flexibility to satisfy increasing context-switching needs, that is to say to implement different algorithms at different times. When the specifications for any application are given, they can be executed on a software processor (i.e. on a general-purpose processor) or on a hardware processor (i.e. on an ASIC or on an FPGA). A software processor contains all the basic blocks needed to build any logic of mathematical function imaginable but the limitations are memory latencies and available parallelism. Performance and power consumption requirements cannot meet mobile device requirements. In this article we propose a methodology to map selected groups of DSP tasks to multi-mode cores using conventional hardware technologies.

The main purpose of HLS design flows is to share resources during the execution of the algorithm (intra-mode resource sharing). One of the goals of multi-mode system design is to minimize the area reusing the resources effectively among the different modes. The main idea of this article is to make use of high-level synthesis for the design of digital signal processing multi-mode systems, i.e. to take benefit from intra-mode as well as inter-mode resource sharing.
This article is organized as follows. Section 2 presents related work around multi-mode system design. Section 3 is dedicated to the problem formulation. Our design flow is detailed in Section 4. The high-level synthesis process and the associated algorithms are presented in this section. Experimental results are reported in Section 5. Finally, conclusions are drawn in Section 6.

2. Related work

One way to design multi-mode systems is to first perform graph merging before synthesis [6–11]. Each mode is represented by a graph. Subgraph isomorphism, i.e. inter-graph node compatibility, is assessed based on the difference between the cost of the nodes before merging and the cost of the resulting nodes. The datapath merging process identifies the similarities among the data-flow graphs and produces a single datapath that can be configured according to the mode to run [8]. Merged vertices are executed by the same functional unit to reduce the computing resource area usage. Datapath merging is then performed based on the compatibility graph. However to reduce the total area cost, the interconnection cost (multiplexers) has to be taken into account when vertices are merged and it can become a time-consuming process [9,10]. The main goal of datapath merging is to perform inter-mode resource sharing. The total area can be reduced if intra-mode resource sharing is also performed. However if intra-mode resource sharing is taken into account during datapath merging, the datapath merging approach becomes much more complex. To cope with this issue, in [11] data-flow graphs are first merged to create a datapath that reduces the resource area usage by sharing resources between the modes. Then, to optimize the resource area usage, functional units and interconnection units are merged inside the datapath using a modified maximum weighted clique algorithm.

Both intra and inter-mode resource sharing can be performed using multi-mode dedicated high-level synthesis processes [12–14]. The approach proposed in [12] targets ASICs. It makes use of small scale reconfigurability along with high-level synthesis. 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 a small portion in the reconfigurable area. While not fully reconfigurable, in contrast with an FPGA, the design incorporates reconfigurability up to the desired extent.

To cope with the hardware reconfigurability required by the previous approach, the SPA-tially Chained Transformation (SPACT) based approach has been proposed in [13]. The different modes are represented by data-flow graphs (DFGs). Each DFG is scheduled separately. The scheduling can be time constrained using the SPACT-MR approach (minimum resources are used under time constraints) or can be resource constrained using the SPACT-RC approach (minimum latency is obtained under resource constraints). The scheduled DFGs are then concatenated into a single DFG. Finally this DFG is bound to resources. This technique leads to a low complexity binding. However because modes are scheduled separately, similarities between configurations are not taken into account. Extra sharing cost (interconnection resources like multiplexers, and controller complexity) can be reduced.

To reduce the number of control steps, path-based scheduling [15] can be performed. This algorithm targets applications with many control constructs such as conditional branches and loops that emphasize fast schedules. First each path is scheduled independently for speed. A path corresponds to one possible execution sequence, i.e. one configuration of a multi-mode system in our case. Then the schedules for each path are overlapped to find the minimum number of states for all the paths. The aim is to obtain the minimum number of control states for all execution paths but register and interconnection cost issues are ignored.

The approach proposed in [14] completes the SPACT approach taking into account the increase of both the controller and the interconnection cost. This approach is dedicated to throughput constrained syntheses and targets pipelined datapaths. DFGs are scheduled in a sequence according to a priority function that takes into account the average parallelism: DFGs are sorted such that the DFG with the greatest ratio between the number of compatible operations and the throughput constraint is scheduled first and so on. The scheduling algorithm aims at maximizing the similarities within the control steps of each pipeline stage and each mode to minimize the controller complexity. Operations are then bound to operators in a way that benefits from a priori similarities between datapaths. However, because register sharing and interconnection binding is completed later, interconnection and controller cost can
be further reduced: scheduling and operation binding decisions would not have been made if actual interconnection paths had been known.

3. Problem formulation and contribution

We can formulate our synthesis problem: given a set of timewise mutually exclusive modes defined before synthesis, generating the dedicated hardware implementation, which can be configured on the fly according to the mode to run, satisfying time as well as resource constraints and minimizing the total area cost.

In this article, we focus on the design based on high-level synthesis. Targeted applications are digital signal processing (DSP) applications.

3.1. Illustrative example

The behavioral kernels of the DSP applications to be synthesized are represented as data-flow graphs (DFG), which clearly exhibit the data dependencies [1,2]. A DFG is a directed acyclic graph \(G(VE)\) where a node represents an operation and an edge \((u,v)\) represents the data dependency between operations \(u\) and \(v\). The intrinsic parallelism between operations can be easily exploited so that the syntheses can be constrained for performance. A single mode system can be represented by only one DFG. A multi-mode system can be represented by a set of DFGs, each corresponding to a particular configuration the system has to execute.

Let us consider a system that can execute two modes: \(mode_1\) and \(mode_2\). In Fig. 2, \(mode_1\) is represented by DFG1 and \(mode_2\) is represented by DFG2. \(Mode_1\) and \(mode_2\) are timewise mutually exclusive, i.e. \(mode_1\) and \(mode_2\) cannot be executed at the same time. Let us schedule DFG1 and DFG2 independently. If maximum performance is targeted, assuming adder latency is 1 cycle and multiplier latency is 2 cycles, then DFG1 and DFG2 require five control steps each to be executed. Fig. 3 shows the scheduling of DFG1 and DFG2 based on a conventional list-scheduling algorithm [1].

We first assume that both DFG1 and DFG2 are implemented separately, i.e. each in a single mode way. HLS makes use of resource sharing during the execution of the mode. Resources can be operators, storage resources and interconnection resources. Fig. 4 shows examples of datapath implementations for DFG1 and DFG2 that an HLS tool can generate. Datapaths consist of operators, registers and interconnection resources. \(Mode_1\) would require two multipliers (MULT) and one adder (ADD) and \(mode_2\) would require one multiplier and one adder. So the total number of operators required to implement these two modes with two single mode architectures is three multipliers and two adders in this case. However \(mode_1\) and \(mode_2\) are timewise mutually exclusive. Thus they can be executed on a multi-mode architecture in which components may be shared by both modes. Such an implementation would require only two multipliers and one adder, and the proper mode would be invoked based on the mode to be executed. Similarly, storage elements (registers (reg) in Fig. 4) can be shared by the different configurations of the multi-mode system. In Fig. 3 let us assume that inputs to DFG1 and DFG2 are registered inputs. So \(a, b, c\) and \(d\) are registered inputs to nodes \(v_1\) and \(v_2\) of DFG1 and similarly \(e, f, g\) and \(h\) are registered inputs to nodes \(n_1\) and \(n_2\) of DFG2. If both DFG1 and DFG2 are implemented separately then eight registers are required to store these inputs. However with a multi-mode architecture, only four registers are needed if register sharing is used for example between ‘a’ and ‘e’, ‘b’ and ‘f’, ‘c’ and ‘g’, ‘d’ and ‘h’.

For the time being it is clear that with a multi-mode system both the number of operators and the number of storage elements can be reduced compared to single mode based systems. However resource sharing involves extra costs: interconnection cost and controller cost. Roughly, the more resources are shared, the more interconnection resources are required and the more complex is the controller. A multi-mode system design implements both intra and inter-mode resource sharing. This means resource sharing cost may become very expensive. Therefore, while designing multi-mode systems, further steps are necessary to reduce these extra costs as much as possible.

3.2. Interconnection cost

Every data transfer needs an interconnection path from its source to its sink. The interconnection path may include multiplexers, buses, wire links, etc. In the datapath, interconnection resources make it possible to drive data from registers to operator inputs and from operator outputs to registers. Two data transfers can share all or part of an interconnection path if they do not take place simultaneously. When designing multi-mode systems, whenever the resources are shared by different configurations, extra multiplexers/buses may be required to support this sharing. For example, in the case of the shared input registers for \(a, b, c, d\) and \(e, f, g, h\), if DFG1 and DFG2, since they are not connected to the same operators (\(a\ and \(b\) are linked to a multiplier whereas \(e\ and \(f\) are linked to an adder), additional interconnector elements are required. These extra elements increase area, consume power and cause extra delay. It is thus necessary to take into account this extra cost. The actual interconnection cost results from the interconnection binding, i.e. the conventionally last HLS step.
Because in most of the HLS design flows, operation scheduling and computational resource binding are based on estimated interconnection costs, accurate resource sharing techniques are required during the synthesis of multi-mode systems to avoid interconnection resource dominated architectures.

3.3. Controller cost

An HLS-based methodology targets an RTL architecture implemented by a set of register-transfer components [4]. This architecture includes a datapath and a controller that orchestrate the flow of data in the datapath (Fig. 1). The controller usually consists of a finite state machine (FSM) and a decoder, which steers the multiplexers and loads the registers depending upon the tasks (transfers, storages) to be executed at a particular clock cycle. For a single mode hardware design generated using HLS, the controller area cost is 10% of the datapath area on average. The complexity of a controller for a multi-mode system can be relatively high compared to a single mode system. As discussed earlier, due to the sharing of the operators and the registers, multi-mode systems use more multiplexers or multiplexers with more input ports than single mode systems. As the number of multiplexers or multiplexer input ports increases, the number of control signals also increases. Furthermore, merged register clock enable signals are more complex: original single mode register clock enable signals are combined based on an OR logic function. Extra logic may thus be required. In both the cases (increase of the number of control signals for the multiplexers and complexity increase of the register clock enable signals), the complexity of the decoder part of the controller is increased.

3.4. Contribution

In this article, we present an approach that aims at maximizing both the similarities between datapaths to reduce the extra interconnection cost and the similarities between operations of the different configurations within each control step to reduce the complexity of the decoding part of the controller. A joint ad-hoc scheduling and binding algorithm is proposed. DFGs are processed concurrently. The proposed scheduling aims at scheduling the same computations at the same control state among the modes and the binding favors the same paths to be used rather than new paths to be created. A close-to-actual interconnection cost is computed since a symbolic interconnection cost cannot be used to accurately compute the area cost savings. The synthesis process takes care of data word-length and both time and resource constraints are supported. For example, if we have two modes, one can be optimized for area and the other one for performance.

4. Multi-constraint, multi-mode architecture design

4.1. Design flow overview

Fig. 5 shows our proposed design flow for the synthesis of multi-mode systems. Inputs are the Matlab behavioral descriptions, which specify the behavior of the applications to implement and their constraint and profile files. If an application can be parameterized, its profile file sets the values of the parameters. For example, for a sum of absolute difference (SAD) algorithm, the size of the macro-blocks is given in this file.

The design flow is based on four major steps:

1. Data-width analysis: the behavioral description of each application is transformed into its data-flow graph. To avoid resource wastage, data-width analysis is performed. Each node (data and operations) of the DFG is annotated with its data-width, i.e. the minimum number of bits required to store or to compute each data. A fixed-point representation is considered. Range analysis is performed based on a static method: from the input data-width information given in the profile file, maximum values of the data are estimated by considering the propagation of data ranges through the DFG [16]. It should be noted that a data range propagation approach may lead to pessimistic results (oversized data-widths): it is a worst case analysis. However with the design framework we use, the designer can manually restrict the word-length for particular operation or variable nodes: nodes of the graph can be specifically annotated. This step is performed before the data range propagation through the DFG. Moreover this first step of the design flow can be replaced easily by another user-defined approach like [17–19] for example to perform the data-width analysis.

2. Timing analysis: according to the hardware technology the designer targets and according to the operations that the nodes have to execute, a set of hardware operators are selected. Delays can thus be associated to each operation. For each mode, the node mobility is found by calculating the as soon as possible (ASAP) execution time and, for time constrained modes, the as late as possible (ALAP) execution time.

3. Unified graph modeling: a unified graph, which represents the different configurations to be executed, is used for the synthesis. It is made of all of the annotated DFGs. Each node is thus annotated with both its data-width and its mobility. The time-wise mutually exclusive relationship between the
different modes is implemented by a conditional node statement to emphasize mutually exclusive branches.

4. Unified high-level synthesis: the high-level synthesis process is applied to the unified graph. HLS not only meets the constraints corresponding to the different applications but also tries to minimize the total area cost of the multi-mode architecture.

4.2. High-level synthesis process

The synthesis starts with the allocation step taking into account that in a multi-mode architecture, every resource can be shared without distinction of mode. The multi-mode system operation scheduling is then processed. Conventional high-level synthesis methodologies first perform the scheduling step and then try to optimize the binding step. However, processing the binding step after the scheduling is completed is not a good solution to reduce the interconnection cost: scheduling decisions would not have been made if the actual interconnection cost had been known. To limit the extra sharing cost, a joint ad-hoc scheduling and binding algorithm based on similarities between datapaths and control steps is implemented. The aim is to focus the minimization not just on the operators but on the overall architecture, including registers, interconnection resources and controller. Finally a VHDL-RTL multi-mode architecture is generated.

4.2.1. Resource allocation

The main objective of resource allocation is to calculate the number of operators required while meeting the constraint imposed on each mode. In our approach, the allocation step is performed before the scheduling step. The allocation is first done individually for each mode, i.e. each mutually exclusive branch of the unified graph. In the second step, the allocated operators are combined to obtain the total number of operators needed for the implementation of the set of modes (Eq. (1))

\[
\text{Number (operator\_i)} = \max \{\text{number (operator\_i)} \text{ for each mode}\}
\]

Since modes are timewise mutually exclusive, all the operators will be shared without distinction of the mode. In the case of a resource constrained mode, it is the task of the designer to specify the number of each type of operator. On the contrary, in the case of a time constrained mode, the allocation is performed by the synthesis process. In our approach, the lower bound of the operators required for each kind of operation is computed according to the interval-based allocation technique [20].

4.2.2. Combined scheduling and binding

The pseudo-code of our scheduling and binding algorithm is shown in Fig. 6. A concurrent operation scheduling and global binding algorithm (operators, registers and interconnection resources) is performed. This algorithm is based on the joint scheduling and binding algorithm introduced in [21] to carefully take into account the interconnection cost. Compared to previous approaches dedicated to multi-mode system design, the main differences are (1) modes are not scheduled in a sequence but concurrently, (2) the binding of an operation and its associated...
The minimum binding cost is computed. The binding is performed and an intermediate datapath is generated. Resources are updated if required: resource width increase, insertion of additional inputs to multiplexers and multiplexer/register allocation (L12). Then, at this step of the process, the aim is to reuse this datapath at this particular clock cycle among the other modes to benefit from both datapath and controller similarities. This is the second part of the process. Thus, for every other mode, a list of the $M$ highest priority nodes is selected to be scheduled (L13, L14). Their binding cost is computed over the resources and nodes are bound to the operators with the lowest sharing cost (L15) (see Section 4.2.4).

The intermediate datapath is then updated to execute every mode (L16). Resources are thus updated if required: resource width increase, insertion of additional inputs to multiplexers and multiplexer/register allocation.

Then, the next clock cycle can be processed. The combined scheduling and binding process ends when all nodes from all of the modes have been processed.

### 4.2.3. Scheduling priority function

The scheduling approach we use is based on the list-scheduling algorithm [1,2]. A list-based scheduling algorithm maintains a priority list of ready nodes. A ready node represents an operation that can be scheduled, i.e. whose predecessors have already been scheduled. The priority function is used to sort the ready nodes: the nodes with the highest priorities are scheduled first. Thus the priority function resolves the resource contention among the operations. To reduce the computational complexity of the proposed approach, the scheduling and binding step is applied on a set of ready nodes only (see Section 4.2.4). This set of nodes is selected based on the priority function. Because the extra binding cost is hard to estimate before scheduling, the priority function favors latency cost over datapath cost. Next, operation binding targets binding cost minimization.

The scheduling priority function (Eq. (2)) is based on the following metrics:

- **operation mobility**, as in a traditional list-scheduling,
- **operation word-length**, to favor first the scheduling of operations associated with costly data-width,
- **the number of operations that can be fired (immediate successors waiting for the result of the current operation)**. It is shown in [22] that scheduling ahead the nodes with the highest number of successors reduces the variable lifetime, i.e. reduces the data storage cost.

$$
\text{Priority} = \frac{1}{\text{mobility}} + \alpha_1 \times \text{bit_width} + \alpha_2 \times \text{nb_successors}
$$

$\alpha_1$ and $\alpha_2$ are the weights that the users can adjust based on their requirements to trade off resource reduction and critical path overhead against one another.

### 4.2.4. Binding cost

Our combined scheduling and binding algorithm relies on operation node binding costs: in Fig. 6, in lines (L11) and (L15), couples (node, resources) providing the minimum cost are selected. In practice, to reduce runtime, weighted bipartite graphs are used [23,24].

The binding cost of a particular node over a particular operator is thus required. The binding cost used in this section is based on [21]. Binding an operation to an operator involves the operator itself as well as the resources required to drive the input data to

---

1 In this case, to obtain a solution in a short run-time, no re-scheduling is realized. The set of resources is increased and the scheduling of the next operations benefits of this update.

2 As said in Section 4.2.2, the priority function is also used to choose the mode to process first each clock cycle.
this operator. The binding cost (Eq. (3)) thus includes the operator cost and the path cost. The path cost is made of the interconnection and register resource cost as well as the controller cost (Eq. (5)). Because in our approach scheduling and binding are performed concurrently, these costs can be accurately computed from previously scheduled nodes and previously bound resources. Of course the binding cost is evaluated only when the operator can implement the current operation represented by a node.

The cost of binding operation node \( n_j \) to available operator \( \theta_i \) is given by Eq. (3), where \( o_{\text{operator}} \) is the area overcost and \( o_{\text{path}}(\theta_i, n_j) \) is the path overcost.

\[
\zeta(\theta_i, n_j) = o_{\text{operator}}(\theta_i, n_j) + o_{\text{path}}(\theta_i, n_j)
\]  

(3)

\( \zeta(\theta_i, n_j) \) represents an accurate measure of the operation binding cost. It takes into account the overall design overcost (operator, interconnection resources, registers, controller) required to bind an operation to a particular resource and not just the operator overcost as is usually done.

For each hardware function \( \theta_i \), we defined the resource overcost \( o_{\text{operator}}(\theta_i, n_j) \) (Eq. (4)), such that the resource overcost is the area overcost resulting from the transformation of the particular hardware resource \( \theta_i \) to compute/store/transmit the operation/variable node \( n_j \) to be bound.

\[
o_{\text{operator}}(\theta_i, n_j) = \Gamma(\Omega_{\theta_i}) - \Gamma(\Omega_{\theta_i,n_j})
\]  

(4)

\( \Gamma(\Omega) \) models the area cost of the hardware function \( \theta \) depending on the data-width requirements \( \Omega \). \( \Gamma(\Omega_{\theta_i}) \) is the area of the hardware function \( \theta \) whose width is node \( n_j \)'s and \( \Gamma(\Omega_{\theta_i,n_j}) \) is the area of the hardware function \( \theta \) whose width is resource \( n_j \)'s.\(^3\) When \( o_{\text{operator}}(\theta_i, n_j) \) is negative, i.e. the resource \( \theta_i \) is oversized according to the data-width requirements, \( o_{\text{operator}}(\theta_i, n_j) \) is set to zero because the width of the resource does not need to be modified.\(^4\) \( o_{\text{operator}}(\theta_i, n_j) \) is a particular case where no resource will be shared, hence an extra resource will be allocated. Resource overcost in this case is the area of this extra resource whose data-width is \( \Gamma(\Omega_{\theta_i,n_j}) \).

As said previously, the path overcost \( o_{\text{path}}(\theta_i, n_j) \) includes interconnection and storage resources as well as controller overcosts (mux, reg and ctrl, respectively, in Eq. (5)).

\[
o_{\text{path}}(\theta_i, n_j) = o_{\text{mux}}(\theta_i, n_j) + o_{\text{reg}}(\theta_i, n_j) + o_{\text{ctrl}}
\]  

(5)

The interconnection resource overcost \( o_{\text{mux}}(\theta_i, n_j) \) requires particular care because the multiplexer area depends on both the port-width and the input port number. Eq. (4) is thus updated for multiplexers: the interconnection resource overcost \( o_{\text{mux}}(\theta_i, n_j) \) is given by Eq. (6).

\[
o_{\text{mux}}(\theta_i, n_j) = \Gamma_{\text{mux}}(\Omega_{\theta_i}, \varphi_{n_j}) - \Gamma_{\text{mux}}(\Omega_{\theta_i,n_j}, \varphi_{n_j})
\]  

(6)

\( \Gamma(\Omega, \varphi) \) models the interconnection resource area cost according to the bit-width requirements \( \Omega \) and the input port number \( \varphi \).

The controller overcost \( o_{\text{ctrl}} \) is very complex to estimate during the binding step and its complexity will increase depending on future scheduling and binding decisions. Furthermore controller optimizations deeply depend on the logical synthesis tool used, which may choose from several state encodings to minimize the controller area and/or the critical path. An accurate controller overcost cannot be given. We thus decided not to use the controller overcost as it appears in Eq. (5): it is used instead as a secondary metric to decide between same path overcosts (\( o_{\text{path}} \)). In this case, a symbolic cost is used. Actually, in practice, the controller state encoding that minimizes the complexity of the decoding is the one hot encoding. We chose this state-encoding scheme as a reference to compute the controller overcost. Adding one more controller state to the set of states that sets to “1” the value of one control signal requires one OR gate (without logic optimization). Hence \( e_{\text{ctrl}} \) is defined as the number of resources in the path that are to be controlled to orchestrate the flow of the data (multiplexer and/or register) multiplied by the area cost of one OR gate.

In practice, the datapath is divided into clusters. One cluster consists of one operator, the registers that are connected to the inputs of this operator and the interconnection resources that drive the data to these registers or drive the data from these registers to the operator. For example, in Fig. 7a, three connected clusters are shown (MULT1, ADD1 and ADD2 are operators, reg_1--reg_4 are registers and M1--M5 are multiplexers). Register sharing is applied inside the clusters and not on the overall architecture. This reduces not only the computational complexity but also interconnection cost, by allocating multiplexers with fewer input ports. Such a clustering step makes it possible to limit the propagation delay after the place and route step \(^{25}\). Actually, a hierarchical RTL description enables local placement of resources. The clustering step consists of providing hierarchy in the RTL description. Inside a cluster, interconnections between resources are local wires with small lengths (hence with short propagation delays)\(^5\).

To compute the path overcost \( o_{\text{path}}(\theta_i, n_j) \) (Eq. (5)), three cases have to be considered:

1. A register of the cluster is available and there is already a path between this register and the currently investigated input of the operator. In this case, the path overcost comes from the increase of the data-width (if required). The resource overcost (Eq. (4)) is thus applied to the register and the interconnection resource. For example, in Fig. 7a, let us assume ADD1 input data1 comes from MULT1. Register1 (reg_1) is free and there is already a path from MULT1 to ADD1 through multiplexer M1 (Fig. 7b). This binding does not require any extra resource, only data-width requirements have to be addressed.

2. A register of the cluster is available but there is no path between this register and the currently investigated input of the operator. To drive the data to the operator, an extra interconnection resource has to be allocated (i.e. a two-input multiplexer) or an already allocated interconnection resource can be reused but it is necessary to add an input port to this interconnection resource. For example, in Fig. 7a, let us assume ADD2 input data1 comes from MULT1. Register3 (reg_3) is free but there is no path from MULT1 to register3. An extra multiplexer is required as shown in Fig. 7c (M6). The path overcost due to the register width increase has to be computed if required and the interconnection resource overcost \( o_{\text{mux}}(\theta_i, n_j) \) is replaced by \( o_{\text{mux}}(0, n_j) \) in Eq. (5) to take into account the extra two-input multiplexer. If an already allocated interconnection resource is used but with one more input port, the interconnection resource overcost due to this extra input port is computed (Eq. (6)) and the interconnection resource width increase is also addressed if required.

3. There is no free register. An extra register is thus allocated. In this case, the path overcost is made of this extra register but...
the interconnection path to drive the data to the operator must also be taken into account. For example, in Fig. 7a, let us assume ADD1 input data1 comes from MULT1 and no register is available. A new register is thus required (reg_5) and a new multiplexer is allocated (M6) as shown in Fig. 7d. In practice, three cases may occur as follows: (a) the operator was never used before (not yet bound), so no interconnection resource is required, (b) an extra interconnection resource has to be allocated or (c) an already allocated interconnection resource can be reused but it is necessary to add an input port to this interconnection resource. Eq. (5) is still used for the path overcost. In case (a) \( \omega_{\text{mux}}(t_{\text{mux}}, n_i) \) is replaced by zero and in case (b) by \( \omega_{\text{mux}}(0, n_i) \). Because a new register is allocated, \( \omega_{\text{reg}}(t_{\text{reg}}, n_i) \) is replaced by \( \omega_{\text{reg}}(0, n_i) \).

To compute the minimum cost to bind operation \( n_i \) to operator \( \theta_j \), a weighted bipartite graph \( B = (S \cup T, E) \) is built. The path overcost \( \omega_{\text{path}}(\theta_j, n_i) \) takes into account the path of each input of the operation. Commutativity is investigated when it is implemented. Each vertex \( s_j \in S \) represents an input \( \delta_i \) of operator \( \theta_j \) and each vertex \( t_m \in T \) represents an available register \( t_{\text{reg}} \) of the cluster \( C_i \), or an extra register. \( E \) is the set of weighted edges \( e \) between \( s_i \) and \( t_m \). The weight \( w_{\text{reg}} \), associated with edge \( t_{\text{reg}} \) is the path overcost associated with input \( \delta_i \) of operator \( \theta_j \) and register \( \text{reg}_{\text{reg}} \). Couples (input, register) providing the minimum path cost are found according to the minimal weighted matching for \( B \). The path overcost \( \omega_{\text{path}}(\theta_j, n_i) \) for the operator \( \theta_j \) is the sum of the selected path of every operator input. The algorithmic complexity is \( O(n^2 \log(n) + n \cdot e) \) where \( n \) corresponds to the total number of nodes and \( e \) corresponds to the total number of weighted edges. Conventional operators have two inputs. The number \( e \) is thus less than or equal to twice the number of registers being considered.

The runtime increases with the number of registers taken into account. Applying register sharing taking into account only registers of cluster \( C_i \) associated to operator \( \theta_j \) rather than every register of all the clusters makes it possible to reduce the algorithmic complexity in order to reduce runtime.

When the path overcost \( \omega_{\text{path}}(\theta_j, n_i) \) has been computed, the aim is to select the operator, which will implement the particular operation node \( n_j \) at clock cycle \( k \) (lines L11 and L15 in Fig. 6).

Again, a weighted bipartite graph \( B = (S \cup T, E) \) is built. Each vertex \( s_j \in S \) represents an operation \( n_j \) selected to be scheduled and each vertex \( t_m \in T \) represents an allocated operator \( \theta_m \) of the set of weighted edges \( e_{\text{reg}} \) between \( s_j \) and \( t_m \). Obviously there is an edge \( e_{\text{reg}} \) only if the operator \( \theta_m \) can implement the operation of node \( n_j \). The weight \( w_{\text{reg}} \) associated with the edge \( e_{\text{reg}} \) is given by \( \frac{1}{2}(\mu_{\text{reg}}, n_j) \) (Eq. 3). Operations are bound to operators according to the minimal weighted matching for \( B \) (lines L12 and L16 in Fig. 6), i.e. the minimum binding cost.

Fig. 8 shows the bipartite graph process to compute the minimal binding cost of the first mode to process (first part of the multimode combined scheduling and binding algorithm). As an example Fig. 8a shows the bipartite graph \( B \) for mode1 at a particular clock cycle: two additions and two multiplications are assessed to be bound to three adders and two multipliers. Fig. 8b shows the bindings with the lowest cost. The intermediate datapath provided according to these bindings is then used for the second part of the combined scheduling and binding algorithm. The minimal weight matching procedure is applied over the other modes to benefit as much as possible from this intermediate datapath and to minimize the area increase. For example, Fig. 9 shows the resource binding process to bind scheduled operations of mode1 and mode2 to the intermediate datapath.

As said previously, the minimal weighted matching can be computed by \( O(n^2 \log(n) + n \cdot e) \) where \( n \) is the total number of nodes and \( e \) is the total number of weighted edges. The best joint scheduling and binding algorithm must handle every ready node (nodes whose predecessors have already been scheduled) for minimal weighted matching for \( B \). With our approach, to reduce the algorithmic complexity, only a part of the ready nodes is processed. This is the reason why a scheduling priority function has been used (lines L9 and L13 in Fig. 6 and Section 4.2.3), thus reducing the number of nodes of graph \( B \) to process the number of already allocated operators.\(^6\)

---

\(^6\) For the second part of the combined scheduling and binding algorithm, the algorithmic complexity of minimal weighted matching is \( O(L \sum_{i=1}^{L} \log(n_i) \cdot n_i e) \) where \( L \) corresponds to the number of nodes processed in this part of the algorithm and \( n_i \) and \( e \) correspond to the total number of nodes and the total number of weighted edges of mode \( i \), respectively.
5. Experimental results

To evaluate the effectiveness of our proposed approach, several experiments targeting digital signal processing applications were made. These experiments compare our approach with:

- a basic approach that 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 the resources among the modes. Areas are then added. We call this approach the cumulative approach (CA).
- the synthesis of a multi-mode architecture based on the SPACT approach [13], which we presented in Section 2. The SPACT-MR approach was used for time constrained syntheses and the SPACT-RC approach was used for resource constrained syntheses.

Results were obtained using the design flow presented in this article including the graph analysis, the graph modeling and the high-level synthesis process. For this we used GraphLab tool. The binding cost presented in Section 4.2.4 was used for the binding step without distinction of the approach.

5.1. Resource allocation

Table 1 shows the number of allocated resources for several multi-mode architectures: a SAD operator that computes the sum of absolute differences for different macro-block sizes based on video standards, an architecture dedicated to the SAD and the sum of square differences (SSD) for an 8 × 8 macro-block size, and

Fig. 8. Resource binding to get the intermediate datapath. (a) Bipartite weighted graph showing every possible binding, (b) minimal weighted matching for minimum binding cost.

Fig. 9. Resource binding to get the multi-mode datapath.

http://www.enseirb.fr/~legal/wp_graphlab/

The approach proposed in [14] was not used for comparison. This approach is dedicated to throughput constrained syntheses and targets pipelined datapaths. Unfortunately, when pipelining is not possible due to the latency constraint, the scheduling algorithm dedicated to pipelined datapaths allocates resources with a low usage rate, which increases the area. Moreover, resource constrained syntheses cannot be processed.
a JPEG\(^9\) and inverse JPEG decoder for image compression. The three approaches – cumulative, SPACT and ours – have been applied to these examples. Synthesizes are constrained to get the same throughput and uniform data-width resources are allocated. This experiment allows focusing on resource sharing because the controller cost is not included.

For example, for the SAD, compared to the cumulative approach, which requires a dedicated architecture per mode, the number of operators (adders (ADD), subtractors (SUB) and absolute value operators (ABS)) and registers (REG) is small whatever the multi-mode approach because these resources can be shared among the modes. However, resource sharing involves extra costs such as multiplexers. The number of multiplexers\(^9\) (MUX) may become greater than for the cumulative approach. Special attention is required to get an efficient sharing.

For the SAD/SSD experiment, the operator requirements are different. Specific types of operators, i.e. operators that cannot be used by the other mode, are required. In this case, the difference between the number of registers of the cumulative approach and the multi-mode design approaches is smaller because register sharing is applied only inside clusters associated to one operator rather than to the overall architecture (see Section 4.2).

For the JPEG/iJPEG architecture, all the operators except the divider (DIV) can be shared among the two modes. However, the amount of data to be stored is high compared to the number of operators and the data lifetime does not match so conveniently. Thus register sharing is hard to do and the number of interconnection resources is high.

These experiments show the benefits of a multi-mode architecture. The number of resources is smaller for the proposed approach than for the cumulative approach, with the exception of the interconnections for some applications. The sharing process has to take this cost into account. Compared to the SPACT approach, the number of multiplexers is reduced with our approach because datapath similarities between modes are taken into account.

5.2. Data-width unaware synthesis

To analyze the overall multi-mode architecture efficiency, a logical synthesis was performed after the high-level synthesis step of the design flow. Synopsys Design Compiler was used for the logical synthesis. We target an ASIC standard cell CMOS 65 nm technology from ST Microelectronics. Because the SPACT approach we use for comparison is data-width unaware, for a fair comparison in this section a uniform data-width has been implemented in our approach. Thus the data-width analysis step of the proposed design flow was not performed. 16-bit resources were used and the binding process does not take different widths into account in the resource overcost equations (Eqs. (4) and (6)). These first experiments make it possible to validate our approach based on a joint scheduling and binding algorithm.

We experimented three kinds of sets of modes targeting the kind of signal and image processing an embedded system may implement:

1. **A 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\(^1\); the sum of absolute differences SAD for different macro-block sizes based on video standards.

2. **Different applications that handle the same set of operations**, i.e. every operator can be shared without distinction of mode. We have combined transformations with their inverse like the two-dimensional discrete cosine transform DCT and the inverse DCT used in the video processing systems, the fast Fourier transform FFT and the inverse FFT used in communication systems, and different applications like the DCT and the FFT.

3. **Different applications that handle different sets of operations**, i.e. only part of the operators can be shared: the FIR and the LMS filtering; the SAD and the sum of square differences SSD; the DCT and the local wavelet transform LWT; the DCT and the LWT and the LMS; the JPEG and the inverse JPEG.

Table 2 shows the experimental results: the cumulative approach (CA) area, the area of the largest single mode core, and the area and the dynamic power consumption for the SPACT approach and our approach. Area and power savings obtained using the Synopsys Design Compiler framework are also presented. Power estimation has been carried out based on a statistical power estimation. Area results correspond to the total area, i.e. the datapath including storage resources and its controller. The area is given in a number of gates. We set heterogeneous synthesis constraints: a time constraint (T), a

---

9 The JPEG decoder includes the RGB to YUV color space conversion, the 2d-DCT, the quantization and the ZigZag coding. The arithmetic coding is not included. 8 x 8 macro-blocks are processed.

10 in number of equivalent 2-to-1 multiplexers in Table 1.
resource constraint (R) and no constraint (N) meaning that an “as fast as possible” implementation using the set of resources allocated for the other modes is targeted.

Compared to the cumulative approach, the area savings are 31% on average with our approach. Best results are obtained when the applications have similar behavior. For example, the area savings are 58% for the FIR filtering with various numbers of taps. For configurations without obvious behavior similarities like the discrete cosine transform DCT and the local wavelet transform LWT, the area savings are less: 11% in this case. The area of a multi-mode architecture is often near to that of the largest single mode architecture (increase is 18% on average), which emphasizes the interest of a multi-mode core design. Compared to the SPACT approach, area decreases from 6% up to 16% (average saving: 11%) and the dynamic power consumption is reduced by about 13%. Actually, our approach takes similarities among the applications to multi-mode cores is presented. The methodology used so the resource overcost takes this into account during the scheduling and binding process.

To show further the interest of multi-mode architectures, the set of applications has been re-synthesized taking into account the data-width requirements. Thus, in these experiments, the data-width analysis step of the design flow was applied to size the data and the operations according to the input word-length. For comparison, the first syntheses were performed based on the proposed approach but with a uniform data-width such as in Section 5.2. In essence every resource implements the same mode architecture based on a data-width aware synthesis: the largest required data-width to avoid overflow errors but in this case some resources can be oversized. The binding process does not take different widths into account in the resource overcost equations (Eqs. (4) and (6)) in this case. Then, the next syntheses were performed using the complete proposed data-width aware approach. In this case, resources of different data-widths can be used so the resource overcost takes this into account during the scheduling and binding process. Table 3 shows the synthesis results. The input word-length is given and the largest required data-width is presented taking into account the resource overcost equations. The symbol refers to the cases whose data-width is specifically limited to a common processor data-width, i.e. 32 bits. The total area and the dynamic power consumption for the multi-mode architectures with uniform data-width resources (first experiments) and multiple data-width resources (second experiments) are presented. Results show the interest of the design of a multi-mode architecture based on a data-width aware synthesis: compared to the approach based on uniform data-width resources, the area savings are 30% on average and the dynamic power consumption decreases by 41%.

6. Conclusion

In this article a methodology for implementing groups of applications to multi-mode cores is presented. The methodology is based on high-level synthesis. Because of the sharing of the resources among the different modes, efficient performance/area/power consumption trade-off is achieved. The different configurations of the multi-mode architecture can be optimized for time or resource constraints. Multi-mode system design implies extra costs mainly due to extra sharing resources and the increase of the controller complexity. To limit these extra costs, a combined

### Table 2
Synthesis results using a data-width unaware synthesis.

<table>
<thead>
<tr>
<th>Modes</th>
<th>Synthesis constraints</th>
<th>Thoughtput (Msample/s)</th>
<th>C.A. (gates)</th>
<th>Largest core (gates)</th>
<th>SPACT approach</th>
<th>Proposed approach</th>
<th>%C.A.</th>
<th>% Largest core</th>
<th>% SPACT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Area (gates)</td>
<td>Power (mw)</td>
<td>Area (gates)</td>
<td>Power (mw)</td>
<td>Area savings</td>
</tr>
<tr>
<td>1 8/16/32/64</td>
<td>N/H/T/T</td>
<td>32/20/13/8</td>
<td>8828</td>
<td>3393</td>
<td>4194</td>
<td>1.17</td>
<td>3677</td>
<td>0.81</td>
<td>58</td>
</tr>
<tr>
<td>Viterbi 16/64/256</td>
<td>N/N/T</td>
<td>26/7/2</td>
<td>20.572</td>
<td>14.375</td>
<td>19.780</td>
<td>4.31</td>
<td>17.689</td>
<td>3.85</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>N/N/T</td>
<td>53/22/7</td>
<td>27.981</td>
<td>16.648</td>
<td>23.083</td>
<td>5.03</td>
<td>20.428</td>
<td>4.45</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>N/N/T</td>
<td>53/40/13</td>
<td>35.081</td>
<td>20.180</td>
<td>27.424</td>
<td>5.97</td>
<td>24.088</td>
<td>5.25</td>
<td>31</td>
</tr>
<tr>
<td>SAD 4 x 4 8 x 8 16 x 16</td>
<td>T/R/R</td>
<td>20/7/2</td>
<td>2489</td>
<td>928</td>
<td>1190</td>
<td>0.46</td>
<td>1076</td>
<td>0.42</td>
<td>66</td>
</tr>
<tr>
<td>2 DCT8–IDCT8</td>
<td>T/T</td>
<td>176/208</td>
<td>6868</td>
<td>3491</td>
<td>4735</td>
<td>1.43</td>
<td>4357</td>
<td>1.20</td>
<td>37</td>
</tr>
<tr>
<td>2d DCT 8 x 8 2d IDCT2d</td>
<td>T/N</td>
<td>512/512</td>
<td>46736</td>
<td>24325</td>
<td>35740</td>
<td>5.97</td>
<td>33105</td>
<td>5.43</td>
<td>24</td>
</tr>
<tr>
<td>8 x 8</td>
<td>FFT64</td>
<td>T/N</td>
<td>384/384</td>
<td>65810</td>
<td>41814</td>
<td>10.10</td>
<td>36395</td>
<td>9.25</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>FFT64</td>
<td>T/N</td>
<td>256/512</td>
<td>56675</td>
<td>42034</td>
<td>9.08</td>
<td>38156</td>
<td>7.99</td>
<td>26</td>
</tr>
<tr>
<td>3 FIR16 LMS16</td>
<td>T/T</td>
<td>20/10</td>
<td>4798</td>
<td>3060</td>
<td>3664</td>
<td>0.82</td>
<td>3094</td>
<td>0.67</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>SAD 8 x 8 SSD 8 x 8</td>
<td>T/T</td>
<td>7/7</td>
<td>2242</td>
<td>1838</td>
<td>0.58</td>
<td>1737</td>
<td>0.55</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>SAD 8 x 16/16 SSD</td>
<td>N/T/N/T</td>
<td>7/7/2/2</td>
<td>4729</td>
<td>2420</td>
<td>0.72</td>
<td>2072</td>
<td>0.63</td>
<td>56</td>
</tr>
<tr>
<td>8 x 8 16 x 16</td>
<td>DCT8/LWT8</td>
<td>T/R</td>
<td>176/208</td>
<td>4345</td>
<td>4512</td>
<td>1.22</td>
<td>3875</td>
<td>1.01</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>DCT8/LWT8/LMS8 JPEG/JPEG</td>
<td>R/N/T</td>
<td>176/208/16</td>
<td>6517</td>
<td>5548</td>
<td>1.19</td>
<td>5009</td>
<td>1.06</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T/J</td>
<td>109/109</td>
<td>1338/49</td>
<td>106362</td>
<td>19.78</td>
<td>9832/12</td>
<td>18.11</td>
<td>29</td>
</tr>
</tbody>
</table>

| Average                |                       |                       |              |                      | 31           | 18         | 11           | 13        |

**E. Cassseau, B. Le Gal / INTEGRATION, the VLSI journal 45 (2012) 9–21**

---

5.3. Data-width aware synthesis

Our approach takes into account the data-width during the scheduling and binding process to avoid resource usage wastage.
ad-hoc scheduling and binding algorithm based on similarities between datapaths is implemented. Data-width information is also used to limit resource wastage. Our multi-mode design flow gives promising results. Using uniform data-width resources, total area savings (datapath and controller) amount to about 11% and dynamic power consumption is reduced by about 13% in comparison to a conventional high-level synthesis based multi-mode design approach. Taking data-width into account during the scheduling and binding process, area and dynamic power consumption can be further reduced by about 30% and 41%, respectively.

References


Table 3: Synthesis results using a data-width aware synthesis.

<table>
<thead>
<tr>
<th>Modes</th>
<th>Input word-length</th>
<th>Largest required data-width</th>
<th>Architecture with uniform data-width resources</th>
<th>Architecture with multiple data-width resources</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Area (gates)</td>
<td>Power (mw)</td>
<td>Area (gates)</td>
</tr>
<tr>
<td>FIR 8/16/32/64</td>
<td>8 bits</td>
<td>25</td>
<td>6577</td>
<td>1.339</td>
<td>2869</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>32*</td>
<td>9183</td>
<td>1.77</td>
<td>4815</td>
</tr>
<tr>
<td>SAD 4 x 4/8 x 8/16 x 16</td>
<td>8 bits</td>
<td>14</td>
<td>966</td>
<td>0.367</td>
<td>750</td>
</tr>
<tr>
<td></td>
<td>12 bits</td>
<td>18</td>
<td>1208</td>
<td>0.474</td>
<td>993</td>
</tr>
<tr>
<td>DCT8–IDCT8</td>
<td>8 bits</td>
<td>22</td>
<td>6921</td>
<td>1.79</td>
<td>4433</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>30</td>
<td>12130</td>
<td>2.66</td>
<td>7146</td>
</tr>
<tr>
<td>FFT64–IFFT64</td>
<td>8 bits</td>
<td>32*</td>
<td>84368</td>
<td>22.31</td>
<td>71355</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>32*</td>
<td>84368</td>
<td>22.31</td>
<td>76882</td>
</tr>
<tr>
<td>2d DCT 8 x 8–FFT64</td>
<td>8 bits</td>
<td>32*</td>
<td>104333</td>
<td>17.31</td>
<td>89637</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>32*</td>
<td>104333</td>
<td>17.31</td>
<td>98028</td>
</tr>
<tr>
<td>FIR16–LMS16</td>
<td>8 bits</td>
<td>23</td>
<td>5049</td>
<td>1.023</td>
<td>2514</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>32*</td>
<td>8126</td>
<td>1.422</td>
<td>5095</td>
</tr>
<tr>
<td>SAD 8 x 8–SAD 16 x 16</td>
<td>8 bits</td>
<td>24</td>
<td>3700</td>
<td>1.082</td>
<td>1718</td>
</tr>
<tr>
<td></td>
<td>12 bits</td>
<td>30</td>
<td>5176</td>
<td>1.493</td>
<td>2704</td>
</tr>
<tr>
<td>SSD 8 x 8–SSD 16 x 16</td>
<td>8 bits</td>
<td>19</td>
<td>6339</td>
<td>1.36</td>
<td>4419</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>32*</td>
<td>9357</td>
<td>1.76</td>
<td>8948</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>84368</td>
<td>22.31</td>
<td>71355</td>
</tr>
</tbody>
</table>
Emmanuel Casseau received his M.S. degree in Electrical Engineering in 1990 and Ph.D. degree in Electrical and Computer Engineering from the University of West Brittany, France, in 1994. From 1994 to 1996 he was a Research Engineer in the French National Telecom School, ENST Bretagne, France. From 1996 to 2006 he was an Associate Professor in the Electronic Department at the University of South Brittany, France. He is currently a Professor in IRISA/INRIA (French National Institute for Research in Computer Science and Control), University of Rennes1, France. His research interests include system design, high-level synthesis, SoCs design methodologies and reconfigurable architectures for multimedia applications.

Bertrand Le Gal was born in 1979, in Lorient France. He received his Ph.D. degree in information and engineering sciences and technologies from the Université de Bretagne Sud, Lorient, France, in 2005 and the DEA (M.S. Degree) in Electronics in 2002. He is currently an Associate Professor in the IMS Laboratory, ENSEIRB-MATMECA Engineering School, Talence, France. His research focuses on system design, high-level synthesis, SoCs design methodologies, Application Specific Instruction Processors (ASIPs) and security issues in embedded devices such as Virtual Component Protection (VCP).