A More Efficient and Flexible DSP Design Flow from MATLAB-SIMULINK
Philippe Coussy, Gwenolé Corre, Pierre Bomel, Eric Senn, Eric Martin

To cite this version:

HAL Id: hal-00077301
https://hal.archives-ouvertes.fr/hal-00077301
Submitted on 31 May 2006

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.
HIGH-LEVEL SYNTHESIS UNDER I/O TIMING AND MEMORY CONSTRAINTS

P. Coussy, G. Corre, P. Bomel, E. Senn, E. Martin

ABSTRACT

The design of complex Systems-on-Chips implies to take into account communication and timing constraints but also internal memory architecture and mapping for the integration of dedicated hardware accelerator. We present a methodology and a tool that permit the High-Level Synthesis of DSP applications, under both I/O timing and memory constraints. Based on formal models and a generic architecture, this tool helps the designer in finding a reasonable trade-off between the circuit’s latency and its architectural complexity. The efficiency of our approach is demonstrated on the case study of a FFT algorithm.

1. INTRODUCTION

Due to the complexity of today’s digital signal processing (DSP) applications, it is extremely complex and time consuming to design digital systems at the Register Transfer Level (RTL). In this context, to satisfy the time-to-market, designers need a more direct path from the functionality down to the silicon: they need a layered design flow and associated CAD tools to manage SoC complexity in a short time. These needs lead to the development of an environment that can help the designer to explore the design space thoroughly and to find optimized designs.

Hence, in [1], [2], [3] authors proposed approaches that use Matlab/Simulink/Stateflow tools [4] for the system specification and that produce a VHDL RTL architecture of the system. Based on hardware macro generators that use the “generic”/“generate” mechanisms, the synthesis process can be summarized as a block instantiation. However, though such a component may be parameterizable, it relies on a fixed architectural model with very restricted customization capabilities. This lack of flexibility in RTL blocks is especially true for both the communication unit, which sequence orders and/or timing requirements are defined, and the memory unit, which data distribution is set. High-Level Synthesis (HLS) can be used to increase flexibility. Several scheduling techniques even include memory issues. Among them, most try to reduce the memory cost by estimating the needs in terms of number of registers for a given scheduling: those work only with scalars [5][6]. Only a few really schedule the memory accesses [7][8]. They include precise temporal models of those accesses, and try to improve performances without considering the possibility of simultaneous accesses that would ease the subsequent task of register and memory allocation. Works in [9] include the memory during HLS, but is dedicated to control intensive applications. In SystemC Compiler [10] from Synopsys, and Catapult C from Mentor Graphics, memory accesses are represented as multi-cycle operations in a Control and Data Flow Graph (CDFG). Memory vertices are scheduled as operative vertices by considering conflicts among data accesses. Memory accesses are regarded as Input/Output. The I/O behavior and number of control steps are managed in function of the scheduling mode [11]. In practice, the number of nodes in their input specifications must be limited to obtain a realistic and satisfying architectural solution. This limitation is mainly due to the complexity of the algorithms that are used for the scheduling. These two tools, in addition to the “super state” mode, propose a synthesis mode called “cycle-fixed mode” that maintains a fixed I/O timing behavior that is exactly the same before and after synthesis [11]. Communication is specified using wait statements and is mixed with the computation specification.

In the domain of real-time and data-intensive applications, processing resources have to deal with ever growing data streams. The system/architecture design has therefore to focus on avoiding bottlenecks in the buses and I/O buffers for data-transfer, reducing the cost of data storage and satisfying strict timing constraints and high-data rates. The design of such applications thus needs methodology that relies on (1) constraint modeling for both I/O timing and internal data memory, (2) constraint analysis steps for feasibility checking and (3) synthesis.

In [12] and [13], we proposed a SoC design methodology based on algorithmic IP core re-using. Based on high-level synthesis techniques under I/O timing constraints, our approach aims to optimally synthesize the IP by taking into account the system integration constraints: the data rate, technology, bus format, and I/O timing properties specified by timing frames of transfers. In [14], we introduced a new approach to take into account the memory architecture and the memory mapping in the behavioral synthesis of real-time VLSI circuits. We formalized the memory mapping as a set of constraints for the synthesis, and defined a Memory Constraint Graph and an accessibility criterion to be used in the scheduling step. We used a memory-mapping file to include those memory constraints in our HLS tool GAUT [15]. In this paper, we propose a design flow based on formal models that allow high-level synthesis under both I/O timing and memory constraints for digital signal processing algorithms. DSP systems designers hence specify the I/O timing, the computation latency, the memory distribution and the application’s data rate requirements that are the constraints used for the synthesis of the hardware components.

This paper is organized as follows: First in section 2 we formulate the problem of synthesis under I/O timing and memory constraints. Section 3 presents the main steps of our approach, and its underlying formal models. In section 4, we demonstrate the efficiency of our approach with the didactic example of the Fast Fourier Transform (FFT).
3.1. Timing Constraint Graph

In a first step, we generate an Algorithmic Constraint Graph \( ACG \) from the operator latencies and the data dependencies expressed in the \( SFG \). The latencies of the operators are assigned to operation vertices of the \( ACG \) during the operator's selection step of the behavioral synthesis flow.

Starting from the system description and its architecture model, the integrator, for each bus or port that connects the component to design to others SoC components, specifies I/O rates, data sequence orders and transfer timing information. We defined a formal model named \( IOCG \) (IO Constraint Graph) that supports the expression of integration constraints for each bus (id. port) that connects the component to the others in the SoC.

Finally, we generate a Global Constraint Graph \( GCG \) by merging the \( ACG \) with the \( IOCG \) graph. Merging is done by mapping the vertices and associated constraints of \( IOCG \) onto the input and output vertices set of \( ACG \). A minimum timing constraint on output vertices (earliest date for data transfer) of the \( IOCG \) are transformed into the \( GCG \) in maximum timing constraints (latest date for data computation/production).

After having described the behavior of the component and the design constraints in a formal model, we analyze the feasibility between the application rate and the data dependencies of the algorithm, in function of the technological constraints. We analyze the I/O timing specifications according to the algorithmic ones: we check if the required constraints on input data are always verified with the behavior specified for input data.

The entry point of the IP core design task is the global constraint graph \( GCG \).

3.2. Memory Constraint Graph

As outlined in the previous subsection, A Signal Flow Graph \( SFG \) is first generated from the algorithmic specification. In our approach, this \( SFG \) is parsed and a memory table is created. All data vertices are extracted from the \( SFG \) to construct the memory table. The designer can choose the data to be placed in memory and defines a memory mapping. For every memory in the memory table, we construct a weighted Memory Constraint Graph \( MCG \). It represents conflicts and scheduling possibilities between all nodes placed in this memory. The \( MCG \) is constructed from the \( SFG \) and the memory mapping file. It will be used during the scheduling step of the synthesis.

Fig. 3(b) shows a \( MCG \) for the presented example with one simple port memory bank. The variable data \( var2 \) and \( var1 \) are placed consecutively in one bank. Dotted edges represent sequential accesses (two adjacent memory addresses) and plain edges for random accesses (non adjacent addresses). More details on the \( MCG \) can be found in [14].

Further information about the formal models and the memory design can be found in [12][13][14].
3.3. Scheduling under I/O and Memory Flow Constraints

The classical “list scheduling” algorithm relies on heuristics in which ready operations (operations to be scheduled) are listed by priority order. An operation can be scheduled if the current cycle is greater than the ASAP As Soon As Possible time. In our tool, an early scheduling is performed on the GCG. In this scheduling, the priority function depends on the mobility criterion. This mobility is statically computed as the difference between ASAP and ALAP. ASAP operation is computed using ASAP input data arrival time and the latency of the operator allocated for the operation. ALAP operation is computed using ALAP output data deadline and the latency of the operator allocated for the operation. For operations that have the same mobility, the priority is defined using the operation margin. Operation margin is defined as the difference (in cycles) between the current cycle and the operation deadline. An operation deadline is the minimum ALAP (As Late As Possible) time allowed on the outputs that have data dependency with this operation. Whenever two ready operations need to access the same resource (this is a so-called resource conflict), the operation with the lower mobility has the highest priority and is scheduled. The other is postponed. When the mobility is equal to zero, one new operator is allocated to this operation. To perform a scheduling under memory constraint, we introduce fictive memory access operators and add an accessibility criterion based on the MCG. A memory has as much access operators as access ports. The memory bank0. MCG.1 indicates that the sequence var2 → var1 is shorter than var1 → var2. We then schedule (b*var2) at c.step cs.1 and (a*var1) at c.step cs.2 to favour the sequential access (see Fig. 3(c)).

![Diagram](image)

**Fig. 2: Proposed Synthesis Flow**

4. EXPERIMENTAL RESULTS

We described in the two previous sections our synthesis design flow and the scheduling under I/O timing and memory constraints. We present now the results of synthesis under constraints obtained using the HLS tool GAUT [15]. The algorithm used for this experience is a Fast Fourier Transform (FFT). This FFT reads 128 real input X(t(k)) and produces the output Y(k) composed of two parts: one real Yr(k) and one imaginary Yi(k). The SFG includes 16897 edges and 8451
vertices. Several syntheses have been realized using a 200MHz clock frequency and a technological library in which the multiplier latency is 2 cycles and the latency of the adder and the subtractor is 1 cycle.

4.1 Synthesis under I/O timing constraints

In this first experiment we synthesized the FFT component under I/O timing constraints and analyzed the requirements on memory banks. In order to generate a global constraint graph GCG, minimum and maximum timing constraints have been introduced between I/O vertices of the ACG graph using IOCG model. The FFT latency is defined by a maximum timing constraint between the first input and the first output vertices. The specified latency (that is the shortest one according to the data dependencies and the operator latencies) corresponds to a delay of 261 cycles. The FFT component is constrained to read one Xr sample and to produce one Y sample every cycle. The resulting FFT component contains 20 multipliers, 8 adders and 10 subtractors (Table 1). 8 memory banks are required for those I/O timing constraints. However, the internal coefficients are mapped in a non-linear scheme in memory. A large amount of memory bank is needed to get enough parallel access guaranteeing hence the specified latency. Moreover coefficient can be present in multiple banks what requires the design of a complex memory unit.

The specified latency imposed by the memory mapping and the operator latencies corresponds to a delay of 261 cycles. The FFT component is constrained to read one Xr sample and to produce one Y sample every cycle. The resulting FFT component contains 20 multipliers, 8 adders and 10 subtractors (Table 1). 8 memory banks are required for those I/O timing constraints. However, the internal coefficients are mapped in a non-linear scheme in memory. A large amount of memory bank is needed to get enough parallel access guaranteeing hence the specified latency. Moreover coefficient can be present in multiple banks what requires the design of a complex memory unit.

Table 1: Synthesis under I/O timing constraints

<table>
<thead>
<tr>
<th>Memory bank</th>
<th>Input buses</th>
<th>Output buses</th>
<th>Sub.</th>
<th>Add.</th>
<th>Mult.</th>
<th>Latency (in cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td>20</td>
<td>261</td>
</tr>
</tbody>
</table>

4.1. Synthesis under memory constraints

In this second experiment we synthesized a FFT component only under memory constraints. Nevertheless, only the maximal number of concurrent access to memory banks limits the minimal latency value. Hence, with a large amount of operators, a latency equal to the critical path delay of the SFG could be obtained. For the reason we synthesized the FFT component with the amount of operators presented in the first experiment. We then analyzed the requirement on I/O ports and computation latency. The memory constraints are the following: 2 memory banks respecting a simple mapping constraint: the 128 real coefficient Wr in bank0 and the 128 imaginary coefficient Wi in bank1.

The shortest latency imposed by the memory mapping and the amount of operators corresponds to a delay of 215 cycles (Table 2) what is shorter that those obtained in the previous experiment. This architecture requires 36 input busses and 14 output. However, a large amount of busses which data orders are not trivial (non-linear data index progression) is needed. If the environment required the component to exchange data over few I/O busses, this requires the design of a communication unit. This communication unit can add extra latency to serialize data.

Table 2: Synthesis under memory constraints

<table>
<thead>
<tr>
<th>Memory bank</th>
<th>Input buses</th>
<th>Output buses</th>
<th>Sub.</th>
<th>Add.</th>
<th>Mult.</th>
<th>Latency (in cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>36</td>
<td>14</td>
<td>10</td>
<td>8</td>
<td>20</td>
<td>215</td>
</tr>
</tbody>
</table>

5. CONCLUSION

In this paper, a design methodology for DSP component under I/O timing and memory constraints is presented. This approach, that relies on constraints modeling, constraints analysis, and synthesis, help the designer to efficiently implement complex applications. Experimental results in the DSP domain show the interest of the methodology and modeling, that allow trade-offs between the latency, I/O rate and memory mapping.

We are currently working on heuristic rules that could help the designer in exploring more easily different architectural solutions, while considering memory mapping and I/O timing requirements.

Acknowledgements

These works have been realized within the French RNRT Project ALIPTA.

6. REFERENCES