Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 916867, 11 pages
doi:10.1155/2008/916867

Research Article

Multiple Word-Length High-Level Synthesis

Philippe Coussy, Ghizlane Lhairech-Lebreton, and Dominique Heller

Lab-STICC (CNRS), European University of Brittany, The Université de Bretagne-Sud, Centre de Recherche, BP 92116, F-56321 Lorient Cedex, France

Correspondence should be addressed to Philippe Coussy, philippe.coussy@univ-ubs.fr

Received 29 February 2008; Revised 5 May 2008; Accepted 21 July 2008

Recommended by Markus Rupp

Digital signal processing (DSP) applications are nowadays widely used and their complexity is ever growing. The design of dedicated hardware accelerators is thus still needed in system-on-chip and embedded systems. Realistic hardware implementation requires first to convert the floating-point data of the initial specification into arbitrary length data (finite-precision) while keeping an acceptable computation accuracy. Next, an optimized hardware architecture has to be designed. Considering uniform bit-width specification allows to use traditional automated design flow. However, it leads to oversized design. On the other hand, considering non uniform bit-width specification allows to get a smaller circuit but requires complex design tasks. In this paper, we propose an approach that inputs a C/C++ specification. The design flow, based on high-level synthesis (HLS) techniques, automatically generates a potentially pipeline RTL architecture described in VHDL. Both bitaccurate integer and fixed-point data types can be used in the input specification. The generated architecture uses components (operator, register, etc.) that have different widths. The design constraints are the clock period and the throughput of the application. The proposed approach considers data word-length information in all the synthesis steps by using dedicated algorithms. We show in this paper the effectiveness of the proposed approach through several design experiments in the DSP domain.

Copyright © 2008 Philippe Coussy et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. INTRODUCTION

Electronic devices are more and more oriented towards multimedia and communication applications. The design of system-on-chip (SoC) is currently achieved by using system level description, electronic system level (ESL) tools, and by reusing predesigned IP-cores. Typical MPSoC architectures include several processors, memories, I/O devices, communication media (bus, network-on-chip), and dedicated HW accelerators. Indeed, the increasing complexity, the low-power design constraints, and the growing data rates of applications from the digital signal processing (DSP) domain still often require hardwired implementation to be used as a dedicated accelerator in the final SoC. Due to both the complexity of today's DSP applications and the shrinking time-to-market, designers need a more direct path from the functionality down to the silicon. Layered design flow and associated CAD tools to manage DSP system complexity in a shorten time are thus needed. This led to the development of environments that can help the designer to explore the design space thoroughly and to find optimized designs rapidly.

Typically, the design of a DSP application begins by a high-level specification capture of the desired functionality using MATLAB-/Simulink-like environment [1] and/or C/C++ language. This first step consists thus in writing an algorithmic specification with a purely transformational semantics, that is, a function consumes all its input data simultaneously, performs all of computations without any particular timing behavior, and provides all its output data at the same time. At this abstraction level, variables are purely functional (structure, array, etc.), and the data types (typically floating point and/or 16, 32, or 64 bits integer) are not related to the hardware design domain (bit, bit vector). Realistic hardware implementation thus requires the following: (1) to convert the floating-point data types into arbitrary length data types (fixed-precision) while keeping acceptable computation accuracy and (2) to design an optimized hardware architecture starting from this bit-accurate algorithmic specification.

Word-length determination has been early addressed in the literature (see, e.g., [2–4]). This complex and major task in the design flow of DSP applications can be done by using

the following approaches [5–10] when targeting hardware components (DSP, ASIC/FPGA, etc.). Next, the design flow has to take into account bit-width information (i.e., data and operation word-length) in order to carry out an optimized architecture. The design can be done by hand or by using high-level synthesis (HLS) tool. As to be mentioned in Section 2, combining word-length optimization and high-level synthesis allows the hardware implementation cost reduction. This assumes that efficient bit-width aware HLS tools are provided.

In this paper, we present a high-level synthesis approach that inputs specification of digital signal processing application using both bit-accurate integers and fixed point integers. Our approach optimizes area of hardware architectures by taking into account the bit-width information during all the HLS steps. This paper is organized as follows: first, Section 2 presents related work around bit-width aware design steps; Section 3 introduces the general design flow we propose and details the design steps that allow synthesizing optimized multiple word-length architecture; finally, Section 4 shows the effectiveness of our methodology and tool through several experiments in the DSP domain.

2. RELATED WORK

A lot of high-level synthesis work has been presented for two decades [11–13]. However, conventional techniques usually rely on uniform bit-width specification. The works that address the design of multiple bit-width architectures often focus on particular steps of the synthesis flow (see Figure 1). Moreover, to our knowledge, no work has been published on the HLS of fixed-point specification: only bit-accurate integer data type is considered in the literature even if the term “fixed-point” is used.

When the word-length determination is partial, the bit-width refinement, which is the first step of HLS flow, determines bit-width requirements for all integer variables and operations which sizes have not been defined in the input specification. In [14], a forward and a backward propagations are used to infer the minimum bits required.

Next, in order to keep scheduling and binding bit-width unaware (and thus use traditional HLS flows), a clustering is realized. This clustering step groups the operations according to their characteristics into *clusters*. Two operations that belong to the same *cluster* will be able to share an operator if they are not scheduled in the same control step. In order to cluster operations, works from [9, 14, 15] use greedy heuristics while an iterative approach is proposed in [16]. In [14], a bit-width unaware iterative modulo scheduling is performed, afterwards, to satisfy a throughput constraint.

These approaches consider both computational function and area as partitioning criteria and assume latency of operators to be uniform. Unfortunately, larger bit-width operators have longer latency than smaller bit-width operators (see Figure 3): the longest latency will be assigned to all the compatible operations. This can lead to oversized parallel architecture when latency or throughput constraints are considered.

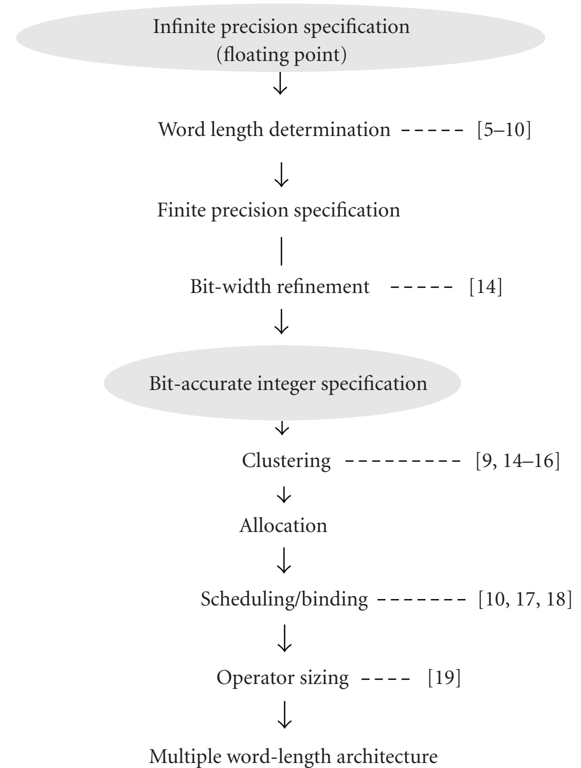

Figure 1: High-level synthesis flow and multiple word-length architecture design approaches.

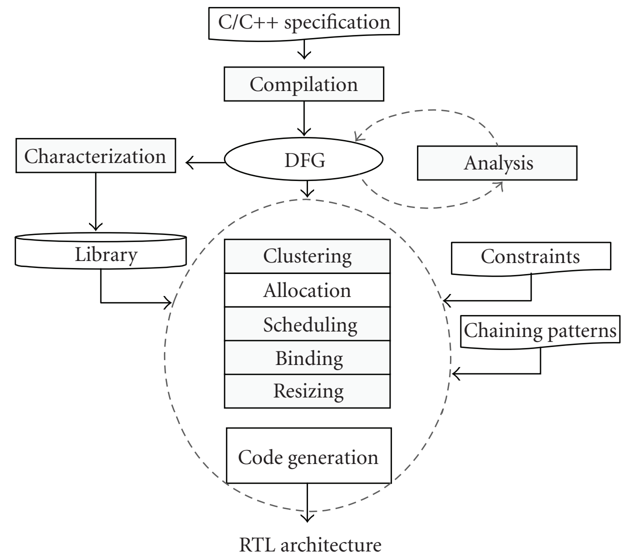

Figure 2: Proposed design flow.

In [10, 16, 17], the word-length optimization is coupled with the high-level synthesis to minimize the hardware implementation cost. In [16], the operations are list-scheduled by decreasing order of timing mobility and no detail on the binding algorithm is provided. Work from [10, 17] is based on a bit-width aware high-level synthesis. A list-scheduling and a register binding algorithm based on the

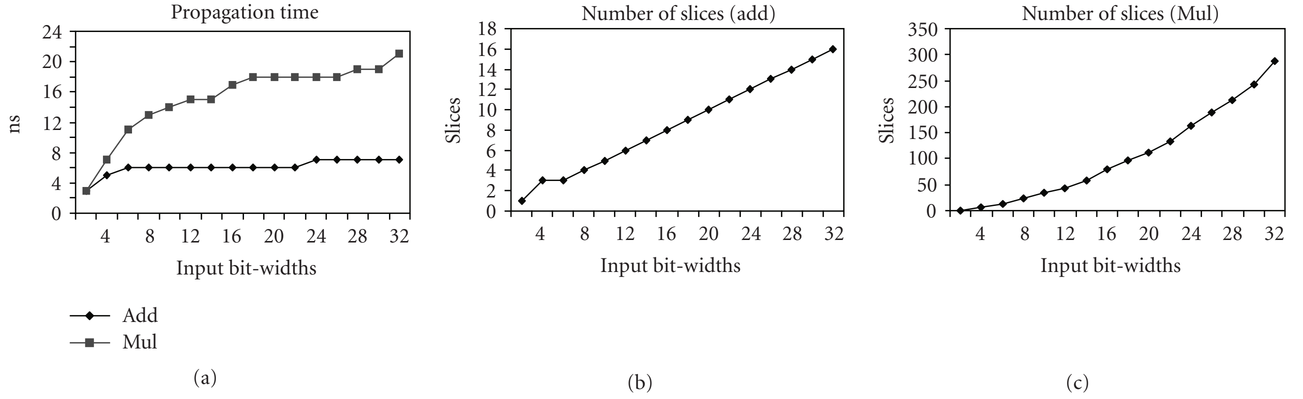

FIGURE 3: Propagation time and area of multiplier and adder according to the size of their inputs.

clique partitioning are described in [10]. The ready operation list with the largest word-length has the highest priority, and the largest scheduled operations are bound first. In [17], an ILP-based approach and a heuristic are presented. Both proposed methods are introduced to perform scheduling with incomplete word-length information and to combine binding and word-length selection. The refinement of word-length information is based on the critical path analysis.

Work from [18] proposes a bit-width aware HLS flow which does not involve clustering step. Authors first compute an estimated area lower-bound to next schedule and bind operation under latency constraint.

Finally, in [19], authors only propose to resize the operators. They first use an HLS tool which does not take into account bit-width information. The proposed optimization step is done after both the scheduling and the binding tasks. To resize the operators and the registers, a forward propagation of the value range of variables is realized.

To our knowledge, none of the previous works jointly

(i) develop a fully automated design flow that inputs specification mixing both bit-accurate integer and fixed-point variables;

(ii) analyze the specification by combining both bit-width and value range bidirectional propagation;

(iii) use propagation time of operators to cluster multiple word-length operations;

(iv) propose bit-width aware scheduling and binding algorithms;

(v) automate operator reusing between bit-accurate integer and fixed-point operations;

(vi) support operator chaining;

(vii) resize operators to optimize the logic synthesis to generate a multiple-word length architecture.

3. BIT-WIDTH AWARE DESIGN FLOW

The proposed high-level synthesis flow is presented in Figure 2. Starting from a purely functional specification, a technological library and both a throughput (iteration period) and a clock period constraints, our tool extracts the potential parallelism before clustering, allocating, scheduling, and assigning operations. It generates a potentially pipelined architecture composed of a processing unit, a memory unit, a communication and multiplexing unit with a globally asynchronous locally synchronous/latency insensitive system (GALS/LIS) interface (see [20] for more details).

The following subsections detail each step of the bit-width aware HLS flow we propose.

3.1. Compilation and bit-width analysis

The input description is a C/C++ function wherein Algorithmic C™ class library from Mentor Graphics [21] is used. This allows the designer to specify signed and unsigned bit-accurate integer and fixed-point variables by using ac_int and ac_fixed data types. This library, like SystemC [22], hence provides fixed-point data-types that supply all the arithmetic operations and built-in quantization (rounding, truncation, etc.) and overflow (saturation, wrap-around, etc.) functionality. For example, an ac_fixed ⟨5, 2, true, AC_RND, AC_SA⟩ is a signed fixed-point number of the form bb.bbb (5 bits of width, 2 bits integer) and with quantization mode set to rounding and overflow mode set to saturation.

The role of the compiler is to transform this initial specification into a formal representation which exhibits the data dependencies between the operations. The front end of our tool derives GCC 4.2 [23] to extract a data flow graph (DFG) representation of the application annotated with the bit-width information. The code optimizations (such as dead-code elimination, false data-dependency elimination, loop transformations, etc.) performed by the compiler will not be presented in this paper. For the quantization/overflow functionality of a fixed-point variable, the compiler generates dedicated operation nodes into the DFG. As described later in Section 3.4, this allows to share (i.e., reuse) (1) arithmetic operators between bit-accurate integer operations and fixed-point operations and (2) quantization/overflow operators between fixed-point operations. Timing performance optimization is addressed through the operator chaining (see Section 3.4).

```
Inputs:
  DFG, timing constraint II and resource allocation

Output:
  A scheduled DFG

Begin
  c-step = 0;
  Repeat until the last node is scheduled
    Determine the ready operations RO;
    Compute the operation mobility;
    While there are RO
      If there are available resources avail(OPR)
        Schedule the operation ops with the highest priority;
        Remove resource opr from available resource set;
        If the current operation ops belongs to a chaining pattern
          Update the ready operations RO;
          If there are available resources
            Schedule the operations corresponding to the pattern;
            Remove resources from available resource set;
          End if
        End if
      Else
        If the operations can be delayed
          Delay the operations;
        Else
          Allocate resources();
          Schedule the operations;
        End if
      End if
    End while
    Bind all the scheduled operations;
    c-step++;
End
```

Algorithm 1: Pseudocode of the scheduling algorithm.

The starting point of the proposed design flow is an already refined specification (the floating-point to fixed-point conversion is not addressed in this paper). However, even if the specification has been already refined (by using existing approaches like [5–10], etc.) the sizes of some variables and constants (#define, int, etc.) and/or variable DFG nodes automatically inferred by the compiler can remain undefined. In order to define the size of such data, the bit-width analysis has been proposed. This step operates on the DFG the two following tasks.

(i) *Constant bit-width definition*: the compiler carries out a DFG representation wherein the constants are represented by nodes with a 16, 32, or 64 bit sizes. This first analysis step defines for each integer (fixed point) constant the exact number of bits needed to represent its value (integer part). We use the following formula for unsigned and signed values:

$$\text{Number of bits} = \lfloor \log_2 |\text{Value}(\text{Cst_}X)| \rfloor + 1 + \text{S}_{\text{igned}}(\text{Cst_}X), \quad (1)$$

where Value(Cst_X) is the numeric value of the constant Cst_X and S_{igned}(Cst_X) is set to "1" when Cst_X is signed and set to "0" otherwise.

(ii) *Bit-width and value range propagation* infers the bit-width of each variable of the specification by coupling work from [14, 19]. A bit-width analysis is hence performed to optimize the word-length of both the operations and the variables. This step performs a forward and a backward propagation of both the value ranges and the bit-width information to figure out the minimum bits required.

3.2. Library characterization and clustering

Library characterization uses a DFG, a technological library, and a target technology (typically the FPGA model). This fully automated step, based on commercial logic synthesis tools like ISE from Xilinx [24] and Quartus from Altera [25], carries out a library of time characterized operators to be used during the following HLS steps. The technological library provides the VHDL behavioral description

of operators and the DFG provides the set of operation to be characterized with their bit-width information. The characterization step synthesizes each operator from the technological library which is able to realize one operation of the DFG. It next retrieves synthesis results in terms of logical cell number and propagation time to generate a characterized operator library (see Figure 3).

For clustering operations, we propose to combine the computational function and the operation delay. This allows considering indirectly operation's bit-width since the propagation time of an operator depends on its input size. In order to maximize the use of operator, one operation that belongs to a cluster C1 with a propagation time t1 can be assigned to operators allocated for a cluster C2 if the propagation time t2 is greater than t1.

3.3. Resource allocation

Allocation next defines the number of selected operators needed to satisfy the design constraints. In our approach, in order to respect the throughput requirement specified by the designer, allocation is done for each a priori pipeline stage. The number of a priori pipeline stage is computed as the ratio between the minimum latency of the DFG *Latency* (i.e., the longest data dependency path in the graph) and the iteration interval II (i.e., the period at which the application has to iterate): $\lceil \text{Latency}/\text{II} \rceil$. Thus, we compute the average parallelism of the application extracted from the DFG dated by a as soon as possible ASAP scheduling [11]. The average parallelism is calculated separately for each *type* of operation and for each pipeline stage s of the DFG, comprising the set of the date operations belonging to $[s \cdot \text{II}, (s+1) \cdot \text{II}]$.

This first allocation is considered as a lower bound. Thus during the scheduling phase, supplementary resources can be allocated and pipeline slices may be created if necessary, subsequent to operation scheduling on the previously allocated operators.

3.4. Operation scheduling

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 its earliest time. Whenever two ready operations need to access the same resource (this is a so-called resource conflict), the operation with the highest priority is scheduled. The other is postponed.

Traditionally, bit-width information is not considered and the priority function depends on the mobility only. The operation mobility (mobility in the following formula) is thus only defined as the difference between the as-late-as possible (ALAP as late as possible) time and the current c-step (see Algorithm 1).

In order to optimize the final architecture area, we modified the classical priority function to take into account the operation bit-width in addition to its mobility. Hence the priority of an operation ops, priority(ops_i), is a weighted sum of (1) the inverse of its mobility mobility(ops_i) (i.e., its timing priority) and (2) the inverse of the overcost inferred by the pseudoassignment of the largest available and compatible operator opr_j with the operation ops_i. The operator opr_j is returned by the maxsize(avail(OPR, ops_i)) function, where OPR is the set of allocated operators and avail(X, y) is a function that returns from the set of operators X, the set of available operators compatible with at least one of the operations y is as follows:

$$\begin{aligned}
\text{priority}(\text{ops}_i) &= \frac{\alpha}{\text{mobility}(\text{ops}_i)} + \frac{1-\alpha}{\text{overcost}(\text{ops}_i, \text{opr}_j)} \\
\text{opr}_j &= \text{maxsize}(\text{avail}(\text{OPR}), \text{ops}_i) \\
&\text{overcost}(\text{ops}_i, \text{opr}_j) \\
&= \text{Min}\left\{ \left(\frac{\text{opr}_{j,\text{in1}} - \text{ops}_{i,\text{in1}}}{\text{opr}_{j,\text{in1}}} + \frac{\text{opr}_{j,\text{in2}} - \text{ops}_{i,\text{in2}}}{\text{opr}_{j,\text{in2}}} \right), \right. \\
&\qquad \left. \left(\frac{\text{opr}_{j,\text{in2}} - \text{ops}_{i,\text{in1}}}{\text{opr}_{j,\text{in2}}} + \frac{\text{opr}_{j,\text{in1}} - \text{ops}_{i,\text{in2}}}{\text{opr}_{j,\text{in1}}} \right) \right\},
\end{aligned} \tag{2}$$

where $\text{ops}_{i,\text{in}k}$ ($\text{opr}_{j,\text{in}k}$) is the bit-width of the kth input of the operation ops_i (operator opr_j).

The overcost function returns the lowest sum of the gradients of operation input's bit-width and of operator input's bit-width. This means that for a same mobility, the priority will be given to the operation that best minimizes the overcost. For different mobility, the user defined factor α allows to increase the priority of an operation ops_i having more mobility than an operation ops_k if overcost(ops_i, opr_j) is less than overcost(ops_k, opr_j). In the overcost computation, the reuse of an operator (already used) is avoided through a pseudoassignment made during the scheduling. A pseudoassignment is a preliminary binding which allows to remove the largest operator from the available resource set. Once the operations can be no more scheduled in the current cycle, the resource binding is performed.

Operation chaining

To respect the specified throughput and clock constraints while optimizing the final area, operator chaining can be used. In our approach, the candidates for chaining are identified by using templates in a library. Through a dedicated specification language (see Figure 4), the user defines chaining patterns with their respective maximum delays. The latency constraint is expressed in number of clock cycles which allows to be bit-width independent in the pattern specification.

In order to automate arithmetic operators sharing between bit-accurate and/or fixed-point operations from a specification using the Algorithmic C$^{\text{TM}}$ from Mentor Graphics [21], our compiler generates for fixed-point operations two nodes in the DFG (see Section 3.1): one node for the arithmetic operation and one other for the quantization/overflow functionality. Indeed, the difference between integer and fixed point comes from the operator implementation. Hardware operators have the same architecture and the same area only if one considers the

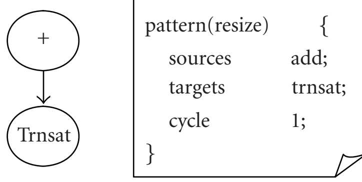

Figure 4: Chaining pattern specification language.

computation part. However, for fixed-point, quantization (rounding, truncation, etc.) and overflow (saturation, wrap-around, etc.) operations can be specified. Of course, the quantization/overflow operations can be part of all the operators (integer and fixed-point). But, this would affect the latency of all the operators and thus the overall latency of the final architecture. This would also increase the overall area. In order to share the computation part only, the compiler has to generate specific operation nodes in the internal representation for the quantization/overflow operations. This allows improving both the area and the timing performance of the final area.

Figure 5(a) depicts a fixed-point dedicated operator, where computational part is merged with the quantization/overflow functionality. This kind of operator architecture allows sharing neither the arithmetic logic nor quantization/overflow part between bit-accurate and/or fixed-point operations.

Figure 5(b) shows the resulting architecture when the compiler generates dedicated nodes for a fixed-point operation and that chaining is not used.

Figure 5(c) presents an architecture wherein the arithmetic part and the quantization/overflow functionality have been chained by coupling both the compiler results and fixed-point templates.

3.5. Resource binding

The assignment of an available operator with a candidate operation has to respond to the minimization of interconnections (steering logic) between operators and to the minimization of the operator's size (see Figure 3 and [18]). Given the set of allocated operators, our binding algorithm assigns all the scheduled operations of the current step (see Algorithm 1). The pipeline control of each operator is managed by a complementary priority on assignment. When an operator is allocated, but not yet used, its priority for assignment is primarily inferior to that of an operator already utilized.

The first step consists in constructing a bipartite weighted graph $G = (U, \text{avail}(\text{OPR}, U), E)$ with:

(i) U, the set of operations in c-step S_k of the DFG;

(ii) avail(OPR, U) is a function that retruns from the set of operators OPR, the set of available operators compatible with the operations from U;

(iii) E, the set of weighted edges (U, avail(OPR)) between a pair of an operation $\text{ops}_i \in U$ and an operator $\text{opr}_j \in \text{avail}(\text{OPR}, U)$;

The edge weight $w_{\text{ops}_i,\text{ops}_j}$ is given by the following equation:

$$w_{\text{ops}_i,\text{opr}_j} = \beta * \text{con}(\text{ops}_i, \text{opr}_j) + (1 - \beta) * \text{dis}(\text{ops}_i, \text{opr}_j), \quad (3)$$

where

(i) $\text{con}(\text{ops}_i, \text{opr}_j)$ is a weighted summation of the maximum number of existing connections between opr_j and each operator assigned to the predecessors of ops_i, and a possible future connection with an operator that could be assigned to the successors of ops_i (i.e., operation/operator compatibility);

(ii) $\text{dis}(\text{ops}_i, \text{opr}_j)$ is the reciprocal of the positive difference between bit-widths of ops_i and opr_j inputs;

(iii) β is user defined factor which allows minimizing either steering logic area or computational area.

The second step consists in finding the maximal weighted edge subset by using the maximal bipartite weighted matching (MBWM) algorithm described in [26].

Let us now consider the following example.

Assuming

(i) the scheduling and binding of the operations of the DFG in Figure 6(a) on c-step1 and c-step2 have been already done,

(ii) the operations ops_1 and ops_4 have been scheduled in c-step3,

(iii) allocated operators are opr_1, opr_2, and opr_3,

(iv) ops_9, ops_7 have been bound to opr_1,

(v) ops_3, ops_0 have been bound to opr_3,

we will focus on ops_1 and ops_4 binding. Our algorithm first constructs the bipartite weighted graph (Figure 6(b)) taking β equal to 1 for the sake of simplicity (i.e., only steering logic is considered). Afterwards, the MBWM algorithm is applied to identify the best edges. Thus operation ops_1 is assigned to opr_1 thanks to the edge weight 3 in Figure 6(b). Edges connected to opr_1 node are then removed (see Figure 6(c)). In other word, connection between opr_3 (FU implementing the ops_1 predecessor) and opr_1 is maximized. Thereby, the creation of multiplexers is avoided, and thus the final architecture has been optimized.

3.6. Operator sizing

In this design step, the operators have to be sized according to the operations which have been assigned on. In order to get correct computing results, the width of the operator inputs/outputs have to be greater or equal to the width of the operation variables.

In the available literature, the input's width of an operator is used to be the maximum of all its inputs (see [14, 18], e.g.). This computing method increases considerably the final area (see Figures 3, 7, and [16]). However, an operator can have different input width. Thus operator sizing task

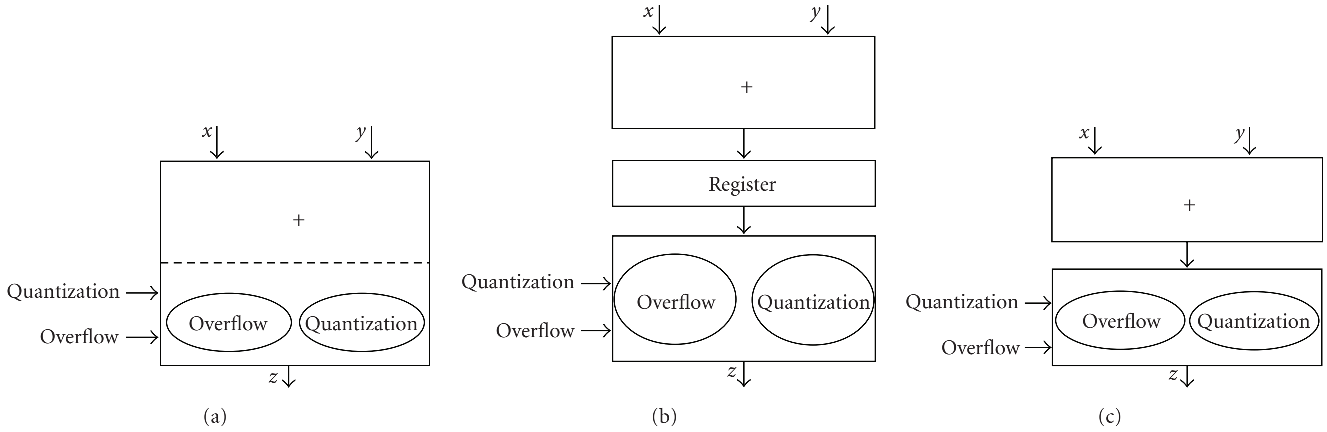

Figure 5: (a) Monolithic fixed-point operator, (b) “unchained” fixed-point operator, and (c) chained fixed-point operator.

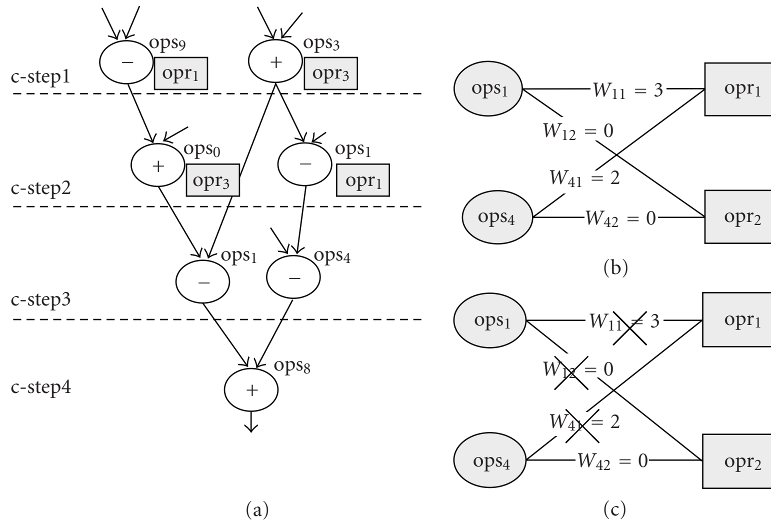

Figure 6: (a) DFG example, (b) bipartite weighted graph, and (c) maximal-weighted edge matching.

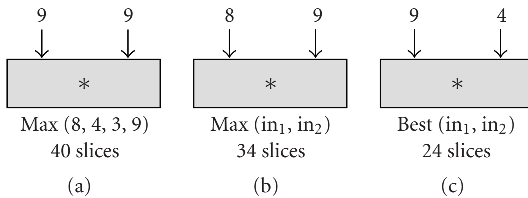

Figure 7: Sizing results.

can optimize the final operator area by (1) computing the maximum width for each input, respectively, (Figure 7(b)) or (2) computing the optimal size for each input by considering commutativity (Figure 7(c)). However, swapping inputs can infer steering logic.

Let us consider a multiplier that executes two operations ops_1 and ops_2. Their respective input widths are ($in_1 = 8$, $in_2 = 4$) and ($in_1 = 3$, $in_2 = 9$) and output width is 12.

Figure 7 shows, respectively, for each approach the synthesis results obtained by using a Xilinx Virtex2 xc2v8000 FPGA device and the ISE 8.2 logic synthesis tool. Considering different widths for each input can thus reduce considerably the operator area.

4. EXPERIMENTS

To show the effectiveness of our approach and its associated high-level synthesis tool [20], several experiments with some well-known DSP applications have been made. All the experiments have been realized on Xilinx Virtex2 xc2v8000-4 FPGA device using ISE 8.2 logic synthesis tool. All the algorithmic specifications have been done using the C language and the Algorithmic C class library from Mentor Graphics [21]. All the syntheses have been realized using a 10 nanoseconds clock period constraint. Experiments have been done using a 2.59 GHz AMD Opteron Processor 252, with 3G RAM and running Windows XP. The execution time of each synthesis using our tool was less than 2 minutes for the considered examples. The RTL descriptions have been simulated using Modeltech’s ModelSim6.1b simulator and

the functional validation has been completed by comparing RTL results to the C/Algorithmic C model ones.

These experiments objective is to present the interest of the full bit-aware HLS flow presented in this paper. First, we show the effectiveness of our bit-width aware synthesis flow with only our resizing step. For this purpose, comparisons have been made against bit-width unaware flows. Second, we highlight the interest of our proposed "bit-width aware" binding and scheduling algorithms, as well as the benefit of the combination of all the synthesis steps. Finally, the last experiment outlines the usefulness of arithmetic operators sharing between bit-accurate and/or fixed-point operations.

4.1. The proposed operator resizing approach

In this first experiment, we compare the three following methodologies:

(i) a pure C ANSI specification with a *classical* HLS flow. All the variables and the constants have thus been defined as integers, that is, their size depends only on the compiler and on the technological library. Standard integer widths have thus been considered, that is, 16 and 32 bits;

(ii) a C/C++ specification using bit-accurate integers (through the Algorithmic C^{TM} class library) with a *uniform* HLS approach that does not consider word-length information. The *uniform synthesis* approach generates thus an oversized architecture where all the components (logical and arithmetic operators, registers, multiplexers, etc.) have the same width, that is, the width of the largest data of the specification;

(iii) a C/C++ specification using bit-accurate integers (through the Algorithmic C^{TM} class library) with the *proposed bit-width aware* HLS approach applying only the resizing step (i.e., the proposed bit-with aware scheduling and binding steps have not been used in this first experiment).

In the *classical* approach, the specification was written by using ANSI C, where the data were of type *int*. Each data has thus been represented with 16 bits in the DFG. In the *uniform* synthesis, we set (by using the Algorithmic C data types) the width of all the variables equal to the width of the largest data of the input specification. For this purpose, we used the bit-width analysis tool presented in Section 3.1 to infer output data widths by propagating the input data bit-widths.

Figure 8 presents the results provided by the three approaches previously described through three well known DSP applications. The first application is a low-pass video line filter. The filter has five nonzero taps with symmetric coefficients, 96 pixels and the image is padded with the boundary pixels. The dataflow graph provided by the compilation step contains 1354 nodes (725 data and 629 operations) with a width varying from 4 to 16 bits. From the figure, standard 16 bits architecture obtained with *classical* flow and *uniform* one (set to 16 bits to respect the computing dynamic of the example) have obviously the same area cost. In contrast, *bit-accurate* architecture is almost two-thirds and thus achieves 27% area reduction (see Figure 8(a)).

The second application is an elliptic filter with 25 operation nodes and 35 data nodes. Data bit-width varies between 4 and 15. *Uniform* architecture is slightly smaller than *classical* 16 bits one, saving only 4.5% amount of area whereas our *bit-accurate* architecture exploits data word-length to reach 40% area saving (see Figure 8(b)).

For the third application (Figure 8(c)), we chose a four taps infinite impulse response IIR filter containing 23 data nodes and 8 operation nodes shared between addition and multiplication operations. Data width varies from 2 up to 14 bits. This experiment shows that *uniform* approach is sometimes unprofitable and leads to an architecture larger than the 16 bits *classical* one. This is due to the fact that the logic synthesis tool uses, for standard widths, optimized hardwired resources [24]. Using our bit-accurate approach reduces noticeably the total area which is two times smaller than the *classical* 16 bits architecture (i.e., 50% area saving).

When using a compiler that considers the integers as 32 bits data, the *classical* approach generates obviously a 32 bit-width architecture. In this case, area waste is considerable. The final area obtained for the line filter and elliptical filter examples is almost three times lager than bit-accurate area (area reduction ratio is around 70%). The most noticeable area reduction occurs for the IIR filter where area is reduced to the fifth reaching 80% area saving. However, when at least one data requires more than 16 bits, the 32-bits compiler has to be used with the *classical* approach.

We synthesized a 16-taps finite impulse response FIR filter where 28-bits data were needed. Figure 9 shows, respectively, 82% and 80% of area reduction obtained by the proposed *bit-accurate* approach compared to the 32-bits *classical* and *uniform* (28 bits) approach.

This first experiment set has shown the interest of the resizing algorithm, we proposed in this paper, which provides alone a large area reduction. Further optimizations can be obtained by using scheduling and binding bit-width aware algorithms as presented in Section 4.2.

4.2. Proposed scheduling and binding algorithms

The scheduling step is a list-scheduling algorithm with a priority criterion combining both mobility and operation overcost (see Section 3.4). A parameter α controls the impact of operation overcost in the operation selection. The binding step has to respond to the minimization of interconnections (steering logic) and to the minimization of the components size (operators and registers). As described in Section 3.5, our binding algorithm provides a parameter β to find a tradeoff between the computation and the routing area. The following experiments were performed with α set to 0.3, meaning that the mobility has greater impact than the operation overcost on the scheduling priority. β has been set to 0.6, meaning that routing area has a greater priority than the combinatorial area. These values were chosen out from a dozen of experiments and outcome, in average, the best area results.

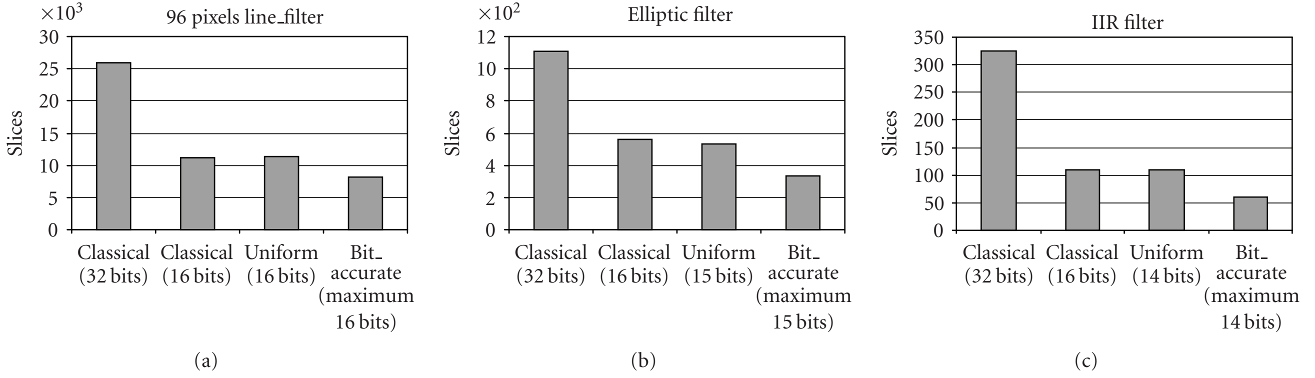

Figure 8: Uniform versus bit-accurate architecture area for (a) line filter, (b) elliptic filter, and (c) IIR filter.

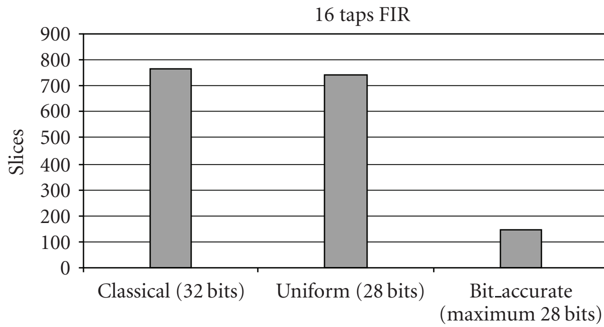

Figure 9: 16 taps FIR design.

This second experiment set shows the interest of considering bit-width information during synthesis steps through tree DSP applications; (1) 96 pixels line filter, described previously; (2) Volterra which consists in first order, nonlinear, differential equations; and (3) a 32 points fast Fourier transform FFT. Figure 10 shows the synthesis results of the presented applications at different throughput constraints.

The line filter design was kept in this set of experiments in order to highlight the interest of optimization algorithms. Actually, the reduction ratio obtained in the first experiment, that is, 70% (see Section 4.1) is improved. The proposed clustering algorithm performs an extra reduction by 6%. The most optimization ratio, 9% is obtained by combining all steps. Thus total area saving is 76.3%.

The Volterra design was synthesized under two different throughputs constraints; 50 Mb/s and 10 Mb/s which leads, respectively, to a 2-stage pipeline architecture and a non-pipeline one. When throughput is equal to 50 Mb/s, area reduction reaches 25% by combing the proposed clustering and binding algorithms. Otherwise, the binding algorithm alone improves area cost by 14% and the clustering algorithm provides an improvement about 18%. Under low throughput constraints, that is, 10 Mb/s, operators are hardly shared. Scheduling and binding algorithms have no effect on total area. Only clustering algorithm manages to reduce area by 10%.

From the 32 points FFT design, we notice that the classical clustering (which does not take into account the propagation time of operators) can lead to oversized area. Indeed, Figure 10(c) shows for abscise $X = (1)$, (2), (3), and (4) a circuit area with a throughput of 4 Mb/s greater or equal that a circuit area that satisfies a 6.66 Mb/s constraint. The proposed time clustering algorithm alone has reduced the area, respectively, for throughputs 4 Mb/s and 6.66 Mb/s, by 6% and 1.6%. Moreover, coupled with the binding algorithm, the gain is raised, respectively, to 6.7% and 3.4%.

The results point out that making each step of the HLS flow "bit-width aware" allows to optimize the circuit area. Actually, the area obtained with an HLS flow which only resizes the operators is reduced utmost by 25% for the Volterra application. Furthermore, a mean reduction of 13% is observed.

4.3. *Operator reusing between bit-accurate integer and fixed-point operations*

Decoupling the arithmetic part and the quantization/overflow functionality (see Section 3.1) for fixed point operators allows the resources sharing as described in Section 3.4. Performance optimizations are addressed through the operator chaining. In this subsection, efficiency of operator reusing is shown through the synthesis of a 32 taps least mean squares (LMSs) filter. This application consists of two distinct computations (1) adaptive coefficient computation which is described with fixed-point data types (ac_fixed) and (2) filtered sample computation specified with bit-accurate integer data types (ac_int).

Figure 11 compares total area of designs produced by the traditionalapproaches and the proposed one. The traditionalapproaches consider fixed-point operator as monolithic. Thus they generate an architecture composed by one integer and one fixed-point subpart which cannot share the operators. Otherwise, the *proposed* approach leads to a unified circuit where operators are shared between bit-accurate integer and fixed-point operations. The proposed approach hence provides a reduction ratio of 10%.

5. CONCLUSION

We have presented in this paper a high-level synthesis flow, which allows the design of multiple word-length

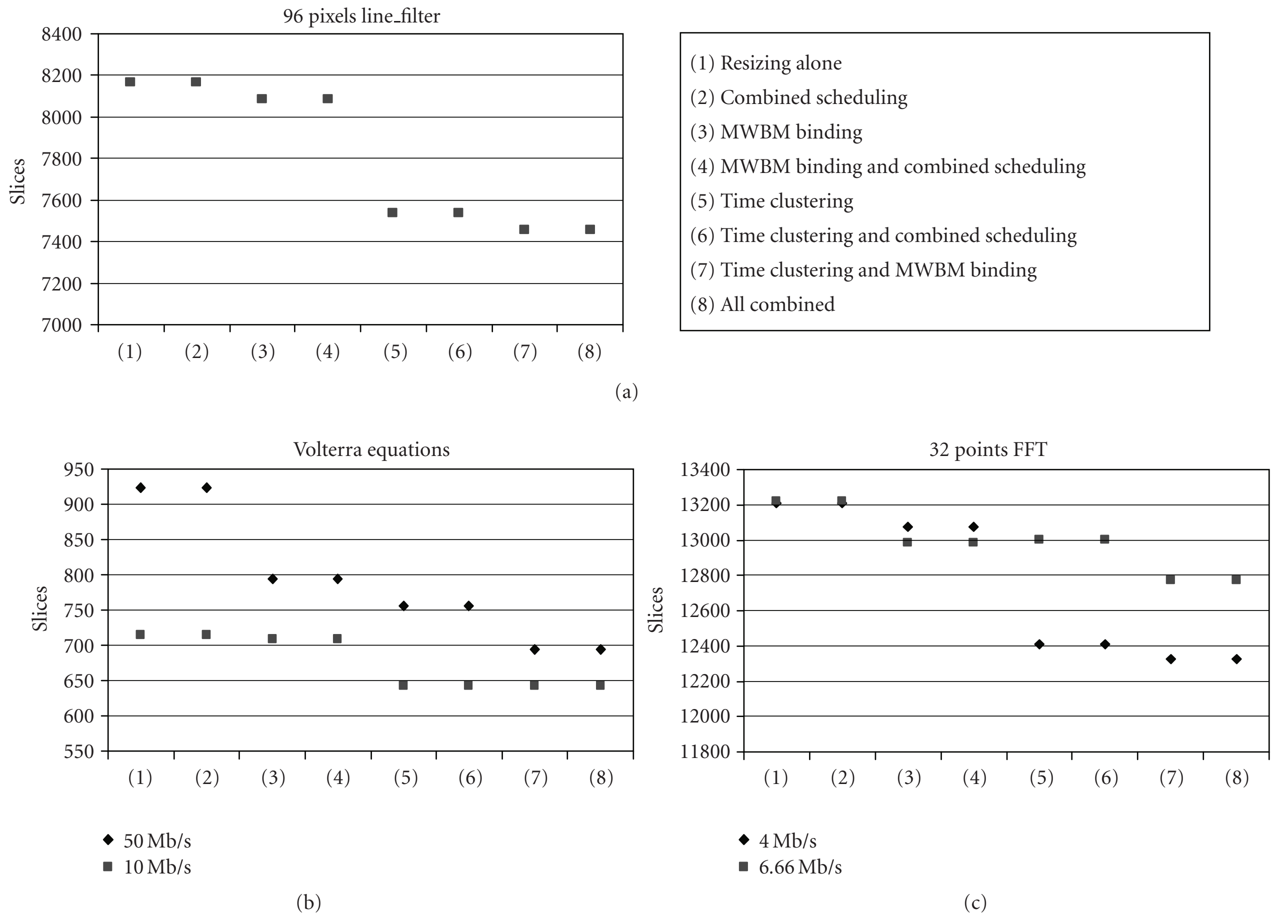

Figure 10: Impact of the bit-width aware design steps for (a) line filter, (b)Volterra, and (c) 32-points FFT.

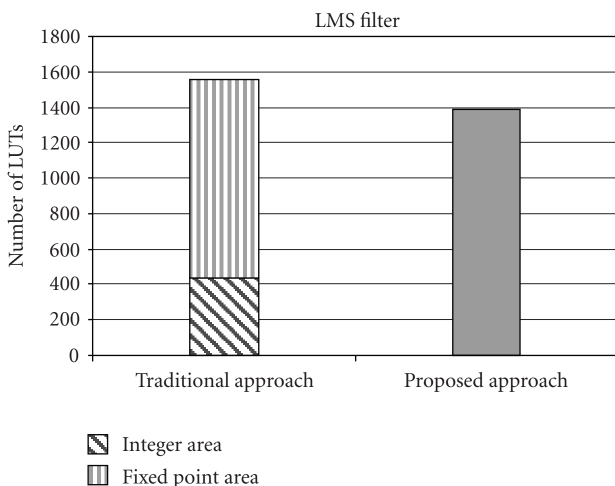

Figure 11: Operator sharing approach versus monolithic approach.

architectures. The approach is based on dedicated compilation, scheduling, binding, and sizing steps that allow optimizing the final architecture area. The experiments have shown promising results for overall area reduction through well known DSP applications on a Xilinx Virtex-2 FPGA.

In the future work, the effects on power consumption will be investigated and the results will be compared with traditional HLS design flows.

REFERENCES

[1] Mathworks, http://www.mathworks.com/.

[2] L. B. Jackson, "Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form," *IEEE Transactions on Audio and Electroacoustics*, vol. 18, no. 2, pp. 107–122, 1970.

[3] L. B. Jackson, "Roundoff-noise bounds derived from coefficient sensitivities for digital filters," *IEEE Transactions on Circuits and Systems*, vol. 23, no. 5, pp. 481–485, 1976.

[4] C. T. Mullis and R. A. Roberts, "Synthesis of minimum roundoff noise fixed point digital filters," *IEEE Transactions on Circuits and Systems*, vol. 23, no. 9, pp. 551–562, 1976.

[5] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: a fixed-point design and simulation environment," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE '98)*, pp. 429–435, Paris, France, February 1998.

[6] L. Wanhammar, *DSP Integrated Circuits*, Academic Press, New York, NY, USA, 1999.

[7] M. Budiu, S. Goldstein, K. Walker, and M. Sakr, "Bitvalue inference: detecting and exploiting narrow bitwidth computations," in *Proceedings of the 6th International Euro-Par*

Conference on Parallel Processing, A. Bode, T. Ludwig, W. Karl, and R. Wism"uller, Eds., vol. 1900 of *Lecture Notes in Computer Science*, pp. 969–979, Springer, Munich, Germany, August-September 2000.

[8] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '00)*, pp. 108–120, Vancouver, Canada, June 2000.

[9] S. A. Wadekar and A. C. Parker, "Accuracy sensitive word-length selection for algorithm optimization," in *Proceedings of IEEE International Conference on Computer Design (ICCD '98)*, pp. 54–61, Austin, Tex, USA, October 1998.

[10] K.-I. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, pp. 921–930, 2001.

[11] D. D. Gajksi, N. Dutt, S. Y.-L. Lin, and A. Wu, *High Level Synthesis: Introduction to Chip and System Design*, Springer, Berlin, Germany, 1992.

[12] G. De Micheli and D. ku, *High Level Synthesis of ASICs under Timing and Synchronization Constraints*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992.

[13] P. Coussy and A. Morawiec, Eds., *High-Level Synthesis: From Algorithm to Digital Circuit*, Springer, Berlin, Germany, 2008.

[14] S. Mahlke, R. Ravindran, M. Schlansker, R. Schreiber, and T. Sherwood, "Bitwidth cognizant architecture synthesis of custom hardware accelerators," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, vol. 20, no. 11, pp. 1355–1371, 2001.

[15] J.-L. Choi, H.-S. Jun, and S.-Y. Hwang, "Efficient hardware optimisation algorithm for fixed point digital signal processing ASIC design," *Electronics Letters*, vol. 32, no. 11, pp. 992–994, 1996.

[16] N. Hervé, D. Ménard, and O. Sentieys, "Data wordlength optimization for FPGA synthesis," in *Proceedings of IEEE Workshop on Signal Processing Systems: Design and Implementation (SiPS '05)*, pp. 623–628, Athens, Greece, November 2005.

[17] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimal datapath allocation for multiple-wordlength systems," *Electronics Letters*, vol. 36, no. 17, pp. 1508–1509, 2000.

[18] J. Cong, Y. Fan, G. Han, et al., "Bitwidth-aware scheduling and binding in high-level synthesis," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '05)*, vol. 2, pp. 856–861, Shanghai, China, January 2005.

[19] B. Le Gal, C. Andriamisaina, and E. Casseaut, "Bit-width aware high-level synthesis for digital signal processing systems," in *Proceedings of IEEE International Systems-on-Chip Conference (SOCC '06)*, pp. 175–178, Austin, Tex, USA, September 2006.

[20] http://web.univ-ubs.fr/lester/www-gaut.

[21] www.mentor.com/.

[22] http://www.systemc.org.

[23] http://gcc.gnu.org/gcc-4.2.

[24] http://www.xilinx.com/ise.

[25] http://www.altera.com.

[26] Z. Galil, "Efficient algorithms for finding maximum matching in graphs," *ACM Computing Surveys*, vol. 18, no. 1, pp. 23–38, 1986.