

Research Article

Latency-Sensitive High-Level Synthesis for Multiple Word-Length DSP Design

Bertrand Le Gal¹ and Emmanuel Casseau²

¹IMS Laboratory UMR-CNRS 5218, Polytechnic Institute of Bordeaux (IPB), University of Bordeaux, 33405 Talence CEDEX, France

²IRISA-CAIRN laboratory, ENSSAT Engineering School, University of Rennes 1, BP 80518, 22305 Lannion CEDEX, France

Correspondence should be addressed to Bertrand Le Gal, bertrand.legal@ixl.fr

Received 28 June 2010; Revised 21 October 2010; Accepted 19 January 2011

Academic Editor: Juan A. López

Copyright © 2011 B. Le Gal and E. Casseau. 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.

High-level synthesis (HLS) currently seems to be an interesting process to reduce the design time substantially. HLS tools actually map algorithms to architectures. Conventional HLS techniques usually focus on uniform-width resources according to the worst-case data requirements, that is, the largest word length. HLS techniques have been reviewed for the last few years to benefit from multiple word-length fixed point description of the algorithms to be implemented. Aims were to save design area and power consumption. Unfortunately, data-width timing issues over the operation's latency have not been taken into account accurately. In this paper, an HLS process that takes care of the delay of the operators according to the data width is presented. Experimental results show that our approach achieves significant design latency saving or area decrease compared to a conventional synthesis.

1. Introduction

Multimedia, communications, and, more generally, consumer electronics applications are witnessing a rapid development towards integrating a complex system on a chip (SoC). The increasingly demanding requirements for digital signal processing applications (like multimedia, new generations of wireless systems, etc.) lead to the implementation of more and more complex algorithms and systems. To handle this increase in complexity and the time-to-market pressure, design methodologies based on high-level synthesis (HLS) are nowadays required [1–3]. These methodologies allow to generate circuits from the behavior of the application to implement and from a set of constraints. Digital signal and video processing applications usually require a large number of computations. Data-width requirements are not the same during the processing. When an ASIC or a FPGA implementation is targeted, area cost, latency, and power consumption can be reduced if redundant bits are identified. Efficient usage of resources requires efficient synthesis methods. However, previous related works usually consider area optimizations. Data width impacts resource area but also impacts the delay of the operators.

In this paper, an HLS process that takes into account operators with variable latency is proposed. It makes it possible to save computation clock cycles, that is, to reduce the design latency when the synthesis is constrained by the number of resources. When the synthesis is constrained for latency, it makes it possible to save area. The methodology we propose manages both area- and time-constrained syntheses. ASIC and FPGA technologies can be targeted.

The paper is organized as follows. Section 2 presents related works about multiple word-length high-level synthesis. Section 3 presents our motivations with an example. Section 4 is dedicated to the proposed methodology. The models and the techniques we use are presented in this section. Experimental results are reported in Section 5.

2. Related Works

Fixed point DSP algorithm implementation based on high-level synthesis mainly consists of two steps: word-length allocation and high-level synthesis. In [4], the benefits of the multiple word-length design approach over the traditional uniform word-length design approach are presented.

Implementation cost may be notably reduced with multiple word-length fixed point description of the algorithms. Several high-level synthesis techniques have been proposed during the last two decades. Conventional HLS techniques usually focus on uniform-width resources. Worst-case data size, that is, the largest word length, is thus considered. Although operation scheduling and resource binding are more complex, optimizations are achieved when multiple word-length HLS is performed. It is due to the fact that resource costs depend on the size of the handled data. Combining both word-length allocation and high-level synthesis makes it possible to explore the dependencies between word lengths, resources, and the quantization error criteria. As shown in [5–7], significant area reduction and latency saving can be achieved, but complexity, which impacts runtime, is increased. Sequential or two-step design approaches firstly perform word-length allocation then high-level synthesis. Provided designs may be optimized, but the overall complexity is reduced. In this paper, we address such design approaches, and we focus on multiple word-length HLS.

Multiple word-length high-level synthesis usually focuses on area optimization [8–12]. For example, in [11], a bit-aware design flow, including data-width analysis, scheduling, and binding, is proposed. The data range analysis introduced in [13] is used to determine the minimum data width required for each operation and memorization. In the second step, MCAS architectural synthesis system [14] performs scheduling, binding, and placement without considering data-width information. In the final step, data-width-aware operation rescheduling and rebinding are performed to minimize the area cost of the processing units.

Pipeline design syntheses are addressed in [15, 16]. The authors [15] perform scheduling and binding in order to minimize interconnection resource cost without first considering data-width information. Based on a data range analysis, resource word-length optimization is performed later during the hardware architecture generation process. This approach has been extended in [16] taking symbolic resource costs into account during scheduling and binding.

In [17], operations are handled at the bit level. One operation can be decomposed into several smaller ones that may be executed in several inconsecutive cycles and over several functional units.

Except [17], previous works assume a fixed propagation delay for an operator whatever the size of the data it handles. Worst-case delay is thus always considered. Our work introduces a formalized way to deal with variable propagation delays when resources process multiple data widths. The flow is based on the fact that the delay required to execute an operation depends on the width of the input data whereas the most significant bits (MSB) of the result are discarded.

3. Problem Formulation

3.1. Hardware Design and Performance. For a general purpose or DSP processor, each operation requires a fixed latency (number of clock cycles) to be executed disregarding the input data width; for example, computing a 11-bit fixed point operation takes the same number of clock cycles as

computing a 16-bit one because, in practice, short integers are used in the source code. This characteristic is linked with the single computing resource-based structure of microprocessor datapath that is still reused all over the computations. In contrast, hardware designs are specifically designed. On ASIC or FPGA technologies, resources can be sized depending on the requirements. Furthermore, an operator can have various implementations providing different performance tradeoffs (propagation delay, area, power consumption, etc.). For example, for addition computations, the designer may choose between various architectural possibilities [18, 19]. Propagation delay comes from the critical path that is to say the MSB computation due to carry propagation. For example, Ripple Carry Adder implementation is cheaper but quite slow while Carry Select Adder is faster but area expensive. This remark on adders can be made also for many other operators such as multipliers which are based on adder trees [20]. Although implementation characteristics depend on architectural choices, the larger the word length is, the slower the operators with binary representation. To increase the clock frequency, that is, to increase the throughput and/or the usage ratio, a commonly used technique is to consider multi cycle operations: slower operations require more than one clock cycle to be executed. The required number of clock cycles is computed according to the propagation delay of the resource and the clock frequency. However, the operation's latency depends on the width of the data. Let us consider for example an adder. If a 16-bit addition is executed on a 32-bit adder, the useful 16-bit result (15 down to 0) is available before the useless MSBs (31 down to 16) are computed. Delay can be saved according to the data-width. Efficiency can thus be improved if the number of clock cycles required to compute an operation is not taken the same whatever the data width. This number of clock cycles depends on both the operator and the data width.

Figure 1 shows the delay required to get the result on the output for 32-bit adders and 32-bit multipliers on an ASIC standard cell 65 nm technology (CORE65LPLVT_NOM_1.00_25C from ST Microelectronics). Results show that the delay is approximately linear to the data width for these two operators. Figure 2 shows the delay for the same operators on the Altera Cyclone-III FPGA technology. Delay increases by step depending on the data width. This is due to the internal structure of FPGA devices based on look-up table (LUT) elements.

3.2. Performance Impact on Delay Modeling. In this section, we present an example to show the interest of an efficient delay modeling during the synthesis process.

3.2.1. Impact on Resource-Constrained Syntheses. Figure 3 presents a basic specification that handles multiple data widths. In this example, a , b , and c are 16-bit data. t (\times_1) and q (\times_2) computations require 16-bit multipliers whereas y (\times_3) require a 32-bit multiplier.

Let us assume a single multiplier is used for the design and the clock period is 5 ns while targeting an Altera Cyclone-III FPGA platform. Because the multiplier is shared, the largest data-width is to be used for the multiplier's

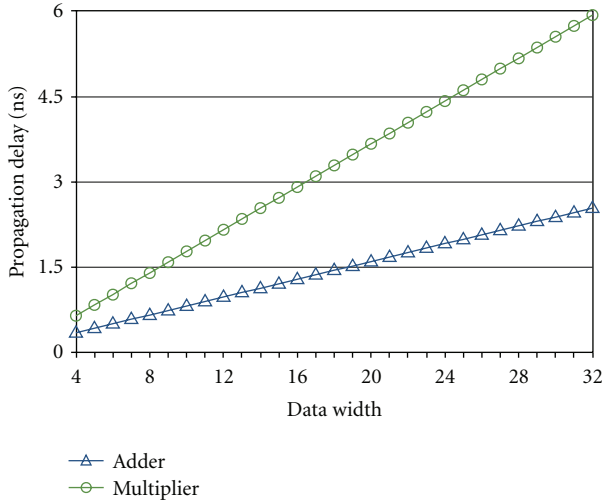


FIGURE 1: Delay for 32-bit resources (65 nm ASIC).

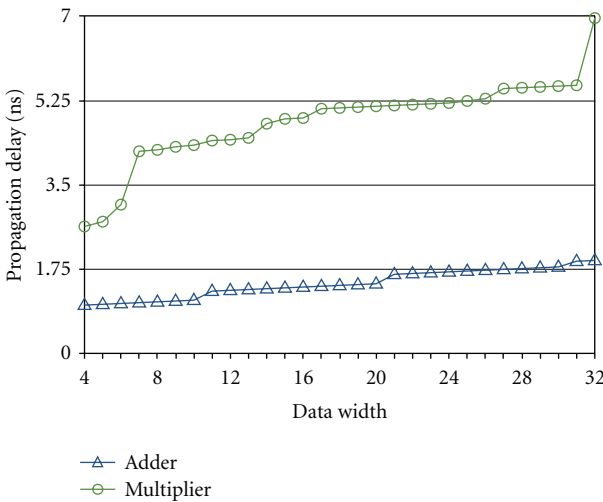


FIGURE 2: Delay for 32-bit resources (Altera Cyclone-III).

word-length. A 32-bit multiplier is thus required. With a Cyclone-III platform, the 32-bit multiplier delay is 6,9 ns.

With a conventional approach, data width is not taken into account for the delay, so the computation delay depends only on the operator’s word length. The multiplier is seen as a multi-cycle operator requiring two clock cycles for a multiplication. Figure 4(a) shows the scheduling of the specification. Multiplications are sequentially scheduled on the multiplier: the two 16-bit multiplications \times_1 and \times_2 are scheduled, respectively, at clock cycles $\{1, 2\}$ and clock cycles $\{3, 4\}$. The 32-bit multiplication \times_3 is scheduled at clock cycles $\{5, 6\}$. Design latency is thus 6 clock cycles.

Using accurate timing models for the operators, propagation delay can be considered individually for each operation depending on its data width. A 16-bit multiplication requires one clock cycle whereas a 32-bit multiplication requires two clock cycles. Figure 4(b) shows the scheduling obtained using accurate delays. The two 16-bit multiplications (\times_1 and \times_2) are scheduled, respectively, at clock cycles $\{1\}$ and $\{2\}$. The

-
- (1) $a, b, c = 16$ bits
 - (2) $t = a \times a;$ (\times_1)
 - (3) $q = b \times c;$ (\times_2)
 - (4) $y = t \times q;$ (\times_3)
-

FIGURE 3: Specification with multiple data-width requirements.

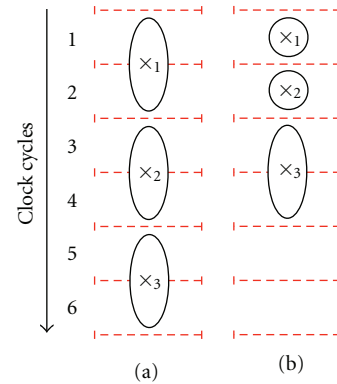


FIGURE 4: Resource-constrained syntheses: (a) scheduling assuming operators with fixed delay, (b) scheduling assuming operators with variable delays depending on data width.

32-bit multiplication (\times_3) is scheduled at clock cycles $\{3, 4\}$. Design latency is thus reduced to 4 cycles.

3.2.2. Impact on Time Constrained Syntheses. Area reduction may also be achieved when the synthesis is constrained for latency. Let us still consider the specification presented in Figure 3. We assume the design latency constraint is 4 clock cycles. Using a conventional approach, every multiplication requires two clock cycles. The two 16-bit multiplications (\times_1 and \times_2) are thus scheduled at clock cycles $\{1, 2\}$ as shown in Figure 5(a) and the 32-bit multiplication (\times_3) is scheduled at cycles $\{3, 4\}$. Two multipliers are required: one 16-bit multiplier to compute \times_1 for example and one 32-bit shared multiplier to compute \times_2 and \times_3 .

Using accurate timing models for the operators according to the width of the data they handle, fewer operators are required. In our case, only one 32-bit multiplier is required. A first 16-bit multiplication (\times_1) is scheduled at clock cycle $\{1\}$ and the second one (\times_2) is scheduled at clock cycle $\{2\}$. The 32-bit multiplication (\times_3) is scheduled at clock cycles $\{3, 4\}$ (Figure 5(b)). The utilization rate of the operators is increased so the area is reduced.

Moreover, compared to a conventional approach where data width is not taken into account for the delay, minimum latency can be reduced. With a conventional approach, minimum latency is 4 clock cycles because multiplications require 2 clock cycles (Figure 5(a)). With the proposed approach, minimum latency is 3 clock cycles (Figure 5(c)). The two 16-bit multiplications (\times_1 and \times_2) are scheduled at clock cycle $\{1\}$ and the 32-bit multiplication (\times_3) is scheduled at cycles $\{2, 3\}$. In both cases, operator’s requirements are the same

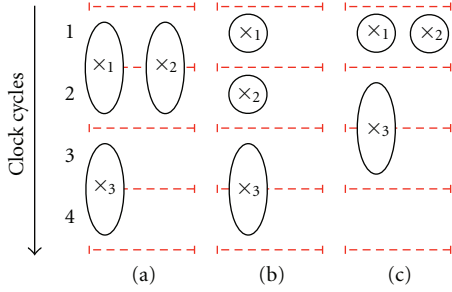


FIGURE 5: Time-constrained syntheses. (a) scheduling assuming operators with fixed delay, (b) Scheduling assuming operators with variable delays depending on data width, and (c) minimum latency scheduling assuming operators with variable delays depending on data width.

(one 16-bit multiplier and one 32-bit shared multiplier) but design latency is reduced with the proposed approach.

3.2.3. Characterized Library. The high-level synthesis steps make use of a characterized library dedicated to the technology the designer targets. This library includes data about delay and area of the resources. With our approach, because the delays of the operators are not fixed, for each operator, a propagation delay function that links the propagation delay to the data width is required (see Section 4.1). The propagation delay function can be automatically extracted from an automated process, based on logic synthesis and simulation tools.

It should be noticed that some operations cannot take advantage of the proposed approach; for example, for the divider operation, the delay is associated to the least significant bit (LSB) computation. In such cases, delay is taken as a constant so a propagation delay function is not required.

4. HLS Design Flow

Our work has been integrated in the GraphLab high-level synthesis tool (http://www.enseirb.fr/~legal/wp_graphlab). This CAD tool is based on a usual high-level synthesis design flow. Its starting point is a MATLAB behavioral description of the algorithm to implement. The synthesis process can be constrained by the designer using different parameters: the target technology, the clock frequency, the design latency, and so forth. The synthesis process initially takes operation's data width into account but to size the resources and minimize area only, that is, the propagation delay of an operator is fixed whatever the width of the data it handles [21]. In order to generate area-efficient designs, a joint scheduling and binding algorithm is used during the synthesis process based on accurate area cost models that depend on data width.

4.1. Path Delay Computation. In HLS, the behavioral description of the application to synthesize is usually translated into an internal representation such as trees or graphs. Data flow graph (DFG) or signal flow graph (SFG) are often used

for DSP applications. It is assumed a data-width analysis of the specification has been previously performed such that each node of the graph can be annotated with its data width $w(n) \rightarrow Z$, that is, the minimum number of bits required to implement node n without loss of precision.

For example, if the extreme values (minima and maxima) are known, data width $w(n)$ required for a signed integer variable node n in two's complement representation can be calculated based on the following equation:

$$w(n) = \left\lceil \log_2(\max(\text{abs}(\beta_{\min}(n)) - 1, \text{abs}(\beta_{\max}(n)))) \right\rceil + 2, \quad (1)$$

where $\beta_{\min}(n)$ and $\beta_{\max}(n)$ are the minimum and maximum values that the variable can be worth.

In the proposed approach, fixed point representation is considered. Data width is thus made up of an integer part plus a fractional part. To avoid binary point alignment, uniform fractional part word length is used.

Assuming f is the operator that implements node n , the characterized library includes, for each operator f , a function $\Gamma_f(w(n)) \rightarrow \mathfrak{R}$ such that $\Gamma_f(w(n))$ corresponds to the propagation delay required by f to compute node n whose data width is $w(n)$. We assume $d_f \in [\text{true}, \text{false}]$ indicates, for each operator f , if node n implemented on operator f may be delay optimized or not. The propagation delay $\delta(n)$ of operator f that implements operation node n can be computed using the following

$$\delta(n) = \begin{cases} \Gamma_f(w(n)) & \text{if } d_f = \text{true} \\ \Gamma_f(w_{\text{op}}) & \text{if } d_f = \text{false}, \end{cases} \quad (2)$$

where w_{op} is the operator word length. $\Gamma_f(w_{\text{op}})$ is thus the delay of operator f when it is assumed operator's delay is fixed.

According to the architecture model targeted by GraphLab tool, the usual computation path from register to register after the logical synthesis is made of an operator and two multiplexers. Equation (3) provides a first estimate of the computation path delay $\delta_{\text{path}}(n)$ required for the computation of node n on operator f . δ_{mux} is the propagation delay of a 2-to-1 multiplexer, and δ_{reg} is the register load delay. (For multiplexer and register resources, propagation delay does not depend on data-width. It should be noticed that propagation delays depend on the load capacitance for each output. In (3), delays take it into account based on the usual computation path);

$$\delta_{\text{path}}(n) = \delta(n) + 2 \times \delta_{\text{mux}} + \delta_{\text{reg}}. \quad (3)$$

In practice, the computation path delay is not only due to logical gates. Part of the computation path delay is also due to interconnection wires. Equation (4) provides the computation path delay $\theta_{\text{path}}(n)$ required for the computation of node n on operator f and including wire delay cost. ε is a routing weight which users can adjust based on their knowledge about the target technology and the complexity of the design ($0 \leq \varepsilon \leq 1$ usually);

$$\theta_{\text{path}}(n) = (1 + \varepsilon) \times \delta_{\text{path}}(n). \quad (4)$$

Finally, the following equation gives the number of clock cycles $\lambda(n)$ required for the execution of operation node n ;

$$\lambda(n) = \left\lceil \frac{\theta_{\text{path}}(n)}{\text{clock period}} \right\rceil. \quad (5)$$

4.2. Resource Allocation. The proposed methodology supports both area- and time-constrained syntheses. For an area constrained synthesis, the designer performs the allocation itself giving the number of each type of operator. For a time-constrained synthesis, the allocation is performed by the HLS tool. The main objective of the resource allocation step is to calculate the right number of each type of operator while meeting the timing constraint. With GraphLab tool, the timing constraint is given as the design latency T to get the result. T is given in number of clock cycles. Two methods are commonly used for the allocation: the average allocation and the interval-based one [22]. We extend these two methods for our proposed approach.

For the average allocation-based approach, an average parallelism is assumed. The minimum number of resources of type f required to implement the operation nodes resource f can execute is given by

$$N(f) = \left\lceil \frac{\eta(f)}{T} \right\rceil \quad \text{where} \quad \eta(f) = \sum_{n \in G_f} \lambda(n). \quad (6)$$

$\eta(f)$ represents the number of clock cycles required to compute sequentially all nodes $n \in G_f$. G_f is a subgraph of graph G including the set of nodes operator f can execute.

For the interval-based technique, the minimum number of resources required for a time interval is calculated using the ASAP(n) and ALAP(n) times. (ASAP(n)/ALAP(n): as soon as possible/as late as possible time operation n can be computed.) Such as in our extended average allocation approach, the accurate number of clock cycles $\lambda(n)$ required for the execution of operation node n is taken into account, for example, to compute ASAP(n) and ALAP(n). For a time interval $[p, q] \in [1, T]$, the minimum overlap between the execution of operation n and the interval is denoted as $W(n, p, q)$ and is calculated as follows:

$$\begin{aligned} W(n, p, q) &= \min \left(\left| \text{ASAP}(n), \text{ASAP}(n) + \lambda(n) \right| \cap [p, q], \right. \\ &\quad \left. \left| \text{ALAP}(n), \text{ALAP}(n) + \lambda(n) \right| \cap [p, q] \right). \end{aligned} \quad (7)$$

The overlap value $W(n, p, q)$ is added to the list $A_f(p, q)$ that gives the rate of operators f required at time interval $[p, q]$. When every overlap value has been computed, the minimum number of each type of operator required during time interval $[p, q]$ is given by

$$N_f(p, q) = \left\lceil \frac{A_f(p, q)}{(q - p + 1)} \right\rceil. \quad (8)$$

4.3. Combined Scheduling and Binding. The join scheduling and binding approach we use is based on the list-scheduling algorithm. A list-based scheduling algorithm maintains a priority list of ready nodes. A ready node represents an operation which can be scheduled, that is, whose predecessors have already been scheduled. A priority function is used to sort the ready operation nodes: nodes with highest priorities are scheduled first. Thus, the priority function resolves the resource contention among operations. The algorithm goal is to consider in the same time an efficient use of the parallelism of the application, data-width information, and datapath cost. The scheduling priority function is based on the following metrics: the operation mobility, the operation data-width to favor first the scheduling of operations associated with costly datapath, and the number of operations which can be fired (immediate successors waiting for the result of the current operation; see [21] for details).

To reduce the area of the overall architecture, including registers and interconnection resources, scheduling and binding are processed concurrently. The binding cost of a particular node over a particular operator is thus required. Binding an operation to an operator involves the operator itself as well as the resources required to drive the input data to this operator (see computation path in Section 4.1). Binding cost thus includes the operator cost, the register cost, and the interconnection cost. When scheduling and binding are performed concurrently, these costs can be accurately computed from previously scheduled nodes and previously bound resources. Weighted bipartite graphs are used to efficiently select minimum binding cost taking data width into account.

The join-scheduling and binding algorithm has been reviewed to support variable latency operations. Main change comes from the computation of the ready to schedule time. When an operation node has been scheduled, output data are tagged as computed and the successors of this node may become ready nodes. The time operation node n_i is ready is given by

$$\text{ready}(n_i) = \max_{n \in \text{pred}(n_i)} (\text{exec}(n) + \lambda(n)), \quad (9)$$

where $\text{exec}(n)$ is the time (clock cycle) operation node n , is executed, n_i is a successor of node n and pred_{n_i} is the set of nodes which are predecessors of node n_i .

With a conventional synthesis process that takes into account operators with fixed propagation delay, data width $w(n)$ is not taken into account to compute the number of clock cycles $\lambda(n)$ required to compute n (5) because propagation delay $\delta(n)$ of operator f that implements operation node n is taken as a constant (Equation 2b). With our proposed approach, $\lambda(n)$ is accurately computed depending on data width.

Moreover, because the priority function used to schedule the nodes depends on the mobility, priority function results may be different compared to the approach with fixed latency operators. Actually, mobility of node n is based on ASAP(n) and ALAP(n) times. With our approach, these times depend on n 's data width because they are computed based on (5).

Both changes make it possible to increase the utilization rate of the resources avoiding clock cycle waste.

5. Experiments

To evaluate the effectiveness of the proposed methodology, experiments on a JPEG decompression description were carried out. Two syntheses have been done.

- (i) A conventional approach [21] using a data-width-aware high-level synthesis flow in which data width is used to size the resources and minimize area only, that is, the propagation delay of an operator is fixed whatever the width of the data it handles. This approach is denoted EXP₁.
- (ii) The second one, denoted EXP₂, corresponds to the approach proposed in this paper. It extends the first approach including variable latency operators depending on data width during the join scheduling and binding step.

Two different technologies were targeted: an ASIC standard cell 65 nm technology (CORE65LPLVT_NOM_1.00_25C from ST Microelectronics.) and an Altera Cyclone-III FPGA platform. Synthesis libraries were characterized using Design Compiler from Synopsys for the ASIC technology and Quartus II v9.01 for the FPGA platform.

5.1. JPEG Decompression Description. The JPEG compression process is a well-known technique used to compress pictures. JPEG compression and decompression algorithms are parts of most video compression standards like MPEG-x and h26x. The processing requires more than five thousand computations. Input data comes from the arithmetic coding bloc and red, green, blue data are generated. The following computations are processed: (1) inverse ZigZag permutation, (2) invert quantization computation, (3) invert 2D-discrete cosine transform, and (4) color space conversion (YCbCr to RGB color space). Input data (three 8×8 data blocs for Y, Cb, Cr) are signed and are coded using 12 bits. The constant coefficients are 12-bit signed. Output data are unsigned and are coded with 8 bits.

Two data-width profiles for the JPEG decompression core have been generated to evaluate the performance of the proposed methodology.

- (i) A first profile with small data-width requirements; data widths range from 16 bits up to 41 bits. This experiment is named low-precision profile.
- (ii) A second profile with larger data-width requirements; data widths range from 16 bits up to 59 bits. This experiment is named high-precision profile.

Range analysis was processed based on a static method considering the propagation of data ranges through the graph [13]. (It should be noticed that this approach leads to pessimistic results (it is a worst-case analysis) [23]. It

was used because of its ease of implementation. More accurate data scaling can be used.) The difference between the two profiles comes from the fixed point data rounding that is performed only on RGB outputs in the high-precision profile whereas rounding is also performed all over the computations for the low-precision profile. (The JPEG decompression behavioral description is translated into a data flow graph. The graphs for the low-precision profile and the high-precision profile are the same except the annotations of the nodes, data-width requirements.) Figures 6(a) and 6(b) show the distribution of the operation's word-length requirements for the low-precision profile and the high-precision profile, respectively.

5.2. Resource-Constrained Synthesis. Figures 7(a) and 7(b) show the distributions of the computation path delay $\theta_{\text{path}}(n)$ required to compute the operations for the low-precision profile and the high-precision profile, respectively, for the 65 nm ASIC technology (Figures 7(a) and 7(b); it is assumed an n -bit operation is executed on an n -bit operator). For these experiments, routing weight was set to 0, 5. The same kind of distributions is obtained for the FPGA technology, but computation path delays range from 4 ns up to 17 ns. Depending on the clock period and the operator's word length, a particular operation requires more or less clock cycles to be executed. For example, for the high-precision profile, assuming clock period is 1,5 ns, the computation path delay of the addition ranges from 1 clock cycle up to 3 clock cycles.

Two resource constraints have been set to implement the JPEG decompression core.

- (1) A small set of operators: 6 adders, 6 subtractors, and 10 multipliers are allocated.
- (2) A large set of operators: 20 adders, 20 subtractors, and 30 multipliers. This configuration allows a higher computation parallelism rate.

Figures 8(a) and 8(b) show the design latency obtained after the synthesis of the JPEG decompression description constrained by the small set of operators for the low-precision profile and the high-precision profile, respectively. Design latency is the number of clock cycles required to execute the JPEG processing on $8 \times 8 [Y, C_b, C_r]$ data blocs. The clock period is specified by the user. Based on the computation path delay distribution (Figure 7), the clock period constraint has been set from 1 ns up to 4,5 ns for the low-precision profile and from 1 ns up to 6,5 ns for the high-precision profile. Figures 9(a) and 9(b) show the design latency when the synthesis is constrained by the large set of operators.

Compared to the conventional approach [21] for which the propagation delay of an operator is fixed whatever the width of the data it handles, the proposed methodology reduces the design latency from 4% up to 35% for the low-precision profile (average saving is 16%) and from 17% up to 39% for the high-precision profile (average saving is 30%). When the clock period is longer than the most important computation path delay, every operation can be scheduled in

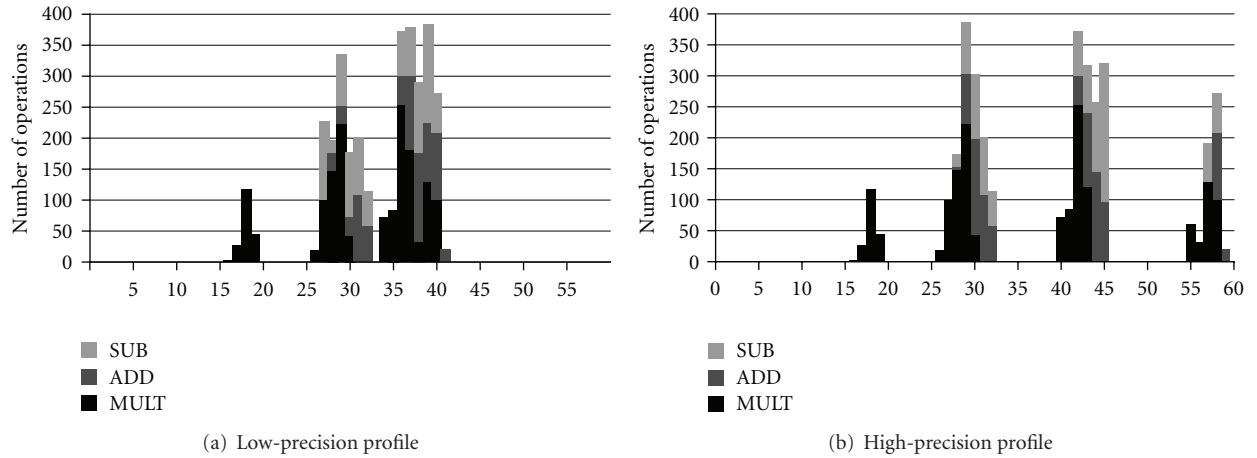


FIGURE 6: Operation's word-length requirements for the JPEG decompression description.

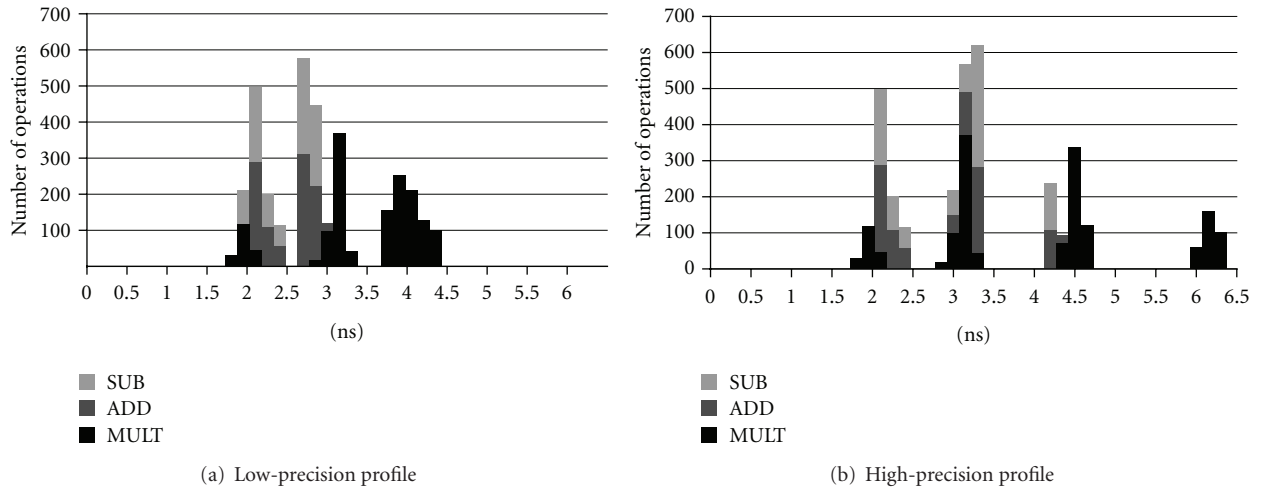


FIGURE 7: Computation path delay distribution—65 nm ASIC technology.

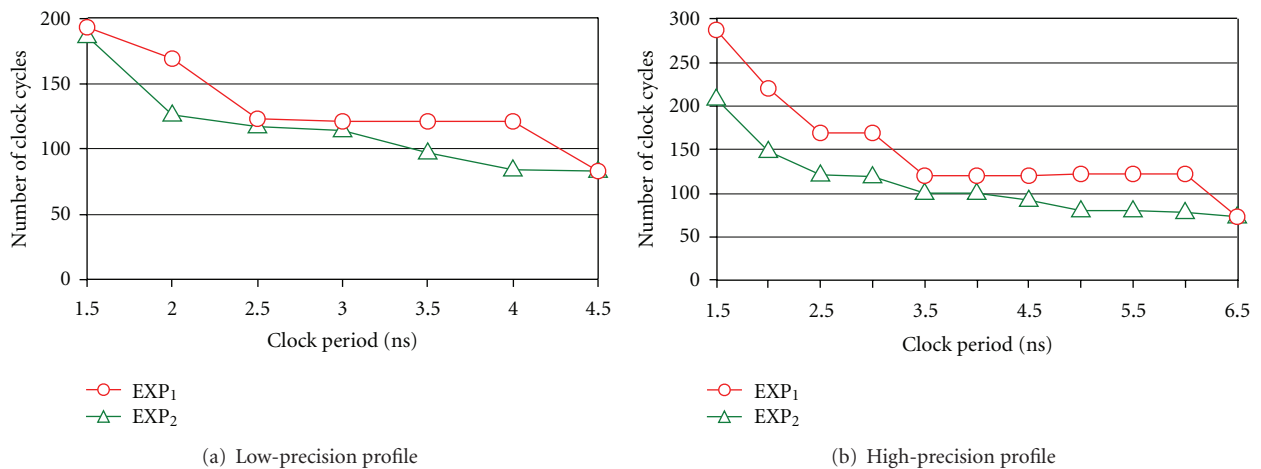


FIGURE 8: Design latency when the synthesis is constrained by the small set of operators—65 nm ASIC technology.

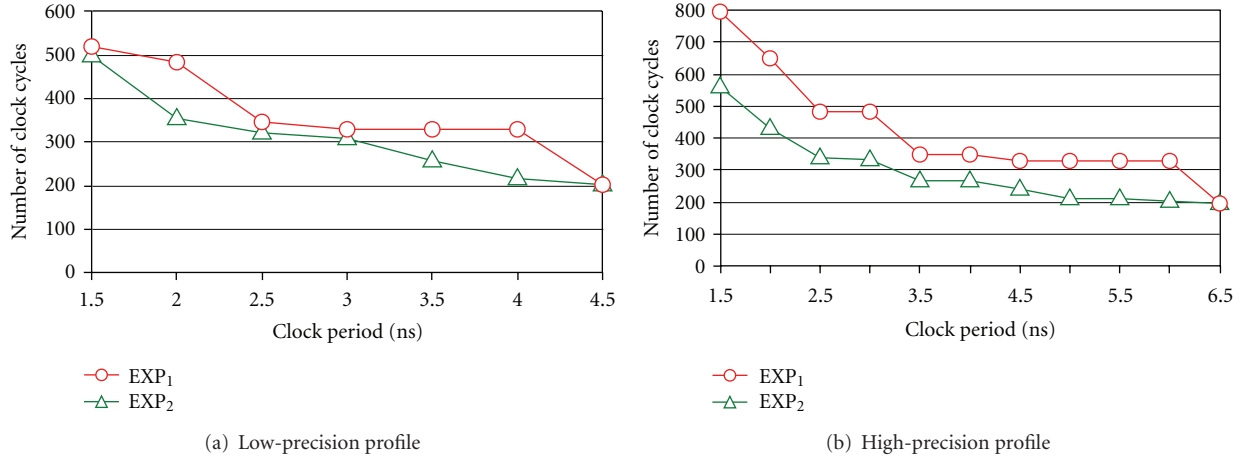


FIGURE 9: Design latency when the synthesis is constrained by the large set of operators—65 nm ASIC technology.

one clock cycle. In this case, there is no design latency saving as it can be seen in Figures 8 and 9 for a 4,5 ns clock period and a 6,5 ns clock period. (This means clock frequency is not very well chosen in this case: operator's utilization rate is low).

It should be noticed that when the clock period is a little bit shorter than the computation path delay of a particular operation, there is no clock cycle saving with this operation. For example, let us consider an operation n whose computation path delay on operator f is $\theta_{\text{path}}(n) = 4$ ns and maximum computation path delay of operator f is 5 ns. Assuming clock period is chosen to be 3 ns, with the conventional approach the number of clock cycles required to compute the operation is $\lambda(n)_{\text{EXP1}} = \lceil 5/3 \rceil = 2$ cycles whereas with the proposed approach $\lambda(n)_{\text{EXP2}} = \lceil 4/3 \rceil$ is also 2 cycles. In this case, the number of clock cycles required to compute operation n is the same whatever the approach. On the contrary, if the clock period is set to 4 ns, the number of clock cycles is reduced to one clock cycle with the proposed approach. The choice of the clock period is thus important.

Similar design latency savings were obtained when targeting ALTERA Cyclone-III FPGA technology. In these experiments, the clock period was set from 4 ns up to 27 ns based on the computation path delay distribution of this technology. Design latency is reduced from 2% up to 27% for the low-precision profile (average saving is 7%) and from 6% up to 36% for the high-precision profile (average saving is 20%).

A logical synthesis has been performed after the high-level synthesis to get area- and energy-consumption results. Design Vision from Synopsys was used and the 65 nm ASIC technology was targeted. Energy consumption was obtained from power consumption. Power consumption was estimated using Prime Power from Synopsys using the statistical-based approach. The complete architectures have been synthesized, that is, the datapath, its controller, and the storage elements required for temporary data and to buffer input and output data. Similar areas are

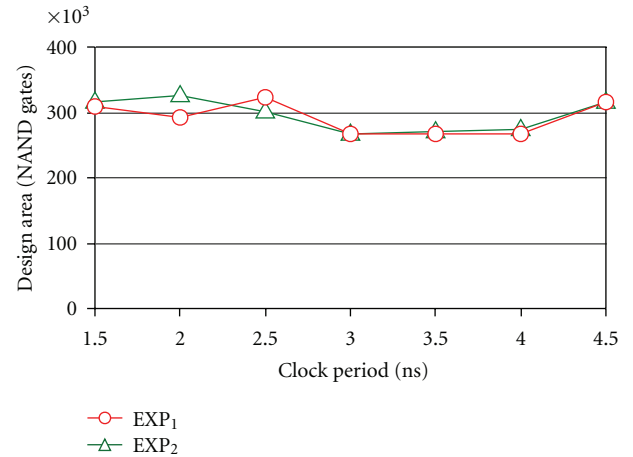


FIGURE 10: Area when the synthesis is constrained by the small set of operators for low-precision profile—65 nm ASIC technology.

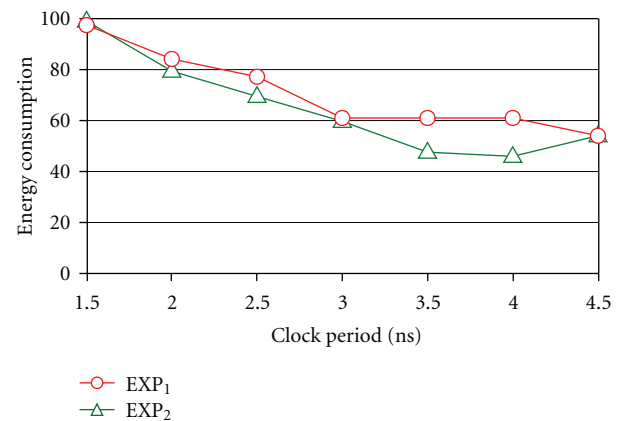


FIGURE 11: Energy consumption when the synthesis is constrained by the small set of operators for low-precision profile—65 nm ASIC technology.

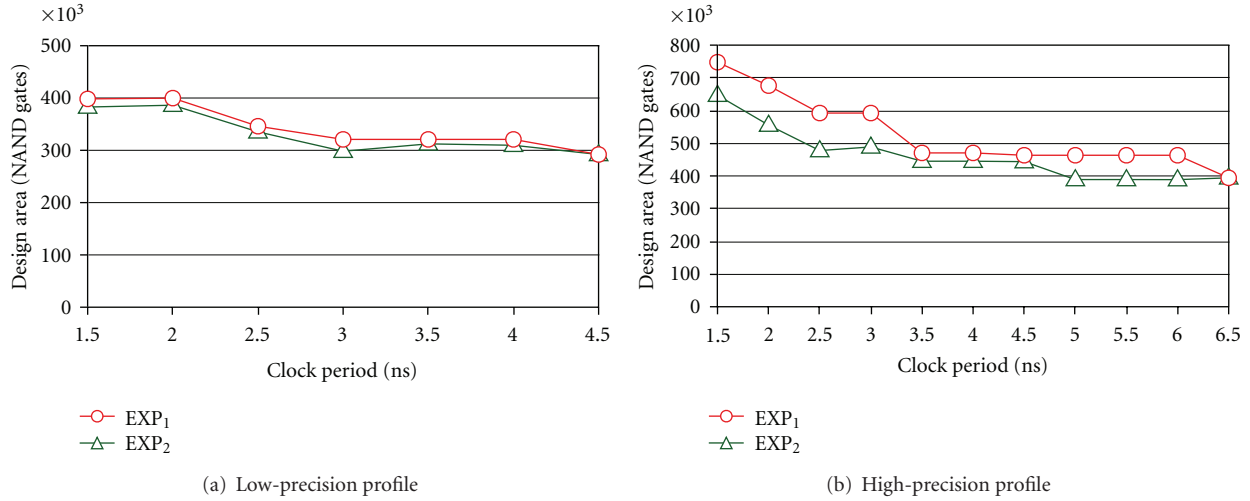


FIGURE 12: Area when the synthesis is constrained by a 240-clock cycle latency constraint—65 nm ASIC technology.

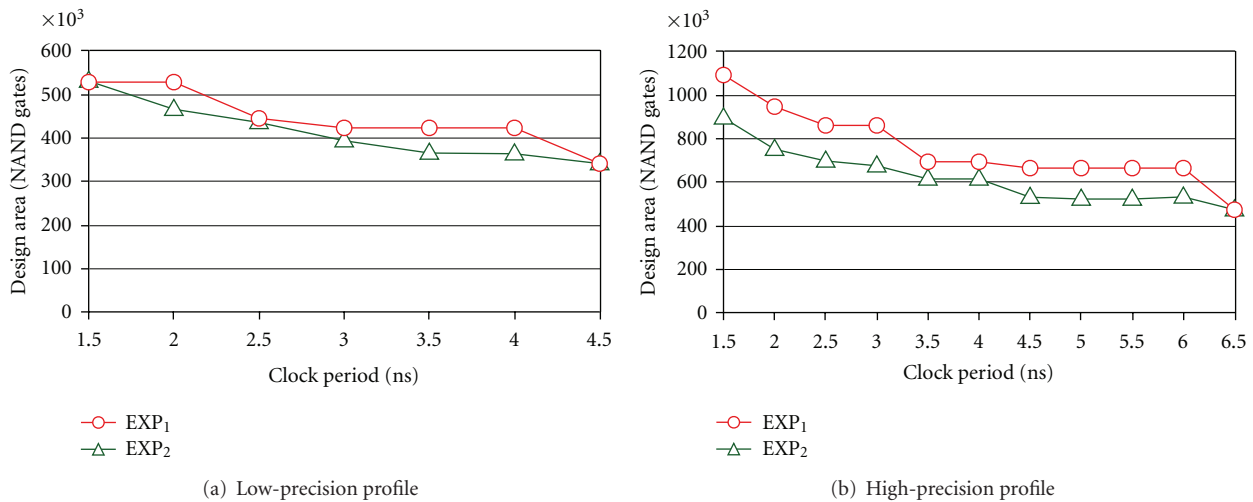


FIGURE 13: Area when the synthesis is constrained by a 120-clock cycle latency constraint—65 nm ASIC technology.

obtained with EXP₁ and EXP₂. For example, Figure 10 shows the area obtained after the logical synthesis of the JPEG decompression description constrained by the small set of operators for the low-precision profile. Area is in number of equivalent NAND gates. On average, the proposed approach is 2% more expensive, that is, similar areas are obtained while design latency is reduced. Furthermore, although resource utilization rate is increased, that is to say switching activity increases, energy consumption decreases with the proposed approach. It is due to the design latency saving. For example, Figure 11 shows the energy consumption of the JPEG decompression core when the synthesis is constrained by the small set of operators for the low-precision profile. Average energy consumption saving is 12%.

5.3. Time-Constrained Synthesis. Time constrained synthesis was also experimented. Low- and high-precision profiles were used for the JPEG decompression synthesis. Two timing constraints have been set.

- (1) Output data are to be provided under a 240-clock cycle latency constraint.
- (2) Output data are to be provided under a 120-clock cycle latency constraint, that is to say a faster design is required.

Figures 12(a) and 12(b) show the area obtained after the synthesis of the JPEG decompression description constrained by a 240-clock cycle latency for the low-precision profile and the high-precision profile, respectively. Target technology is 65 nm ASIC and Design Vision from Synopsys was used for the logical synthesis we performed after the high-level synthesis. As it was already done for resource constraint syntheses, the clock period constraint has been set from 1 ns up to 4,5 ns for the low-precision profile and from 1 ns up to 6,5 ns for the high-precision profile based on the computation path delay distribution (Figure 7). Figures 13(a) and 13(b) show the area when the synthesis is constrained by a 120-clock cycle latency.

Compared to the conventional approach [21] for which the propagation delay of an operator is fixed whatever the width of the data it handles, area is decreased from 0% up to 14% with the proposed methodology for the low-precision profile (average area saving is 6%) and from 2% up to 22% for the high-precision profile (average area saving is 13%). As it was already observed for the resource constrained syntheses, when the clock period is longer than the most important computation path delay, every operation can be scheduled in one clock cycle. In this case, there is no area saving (for a 4,5 ns clock period and for a 6,5 ns clock period, respectively, in Figures 12 and 13).

Similar area savings were obtained when targeting ALTERA Cyclone-III FPGA technology. The clock period was set from 4,5 ns up to 27 ns and both the 120- and 240-clock cycle latency constraints were experimented. Area is decreased from 0% up to 18% with the proposed methodology for the low-precision profile (average area saving is 5%) and from 0% up to 25% for the high-precision profile (average area saving is 12%).

6. Conclusion

In this paper, we have presented a high-level synthesis flow that takes into account operators with variable latency depending on data-width. Both ASIC and FPGA platforms can be targeted. Accurate computation path delay models are used for the allocation and scheduling steps. The synthesis process makes it possible to increase the utilization rate of the resources avoiding clock cycle waste. Design latency can be reduced for resource constrained syntheses. In our experiments, design latency saving is about 19% in comparison to a conventional approach for which the propagation delay of an operator is fixed whatever the width of the data it handles. Energy consumption is also reduced. For time-constrained syntheses, area can be reduced. Area saving is about 9% in comparison to a conventional approach.

References

- [1] E. Casseau, B. Le Gal, S. Huet, P. Bomel, C. Jogo, and E. Martin, "C-based rapid prototyping for digital signal processing," in *Proceedings of the 13th European Signal Processing Conference*, Antalya, Turkey, September 2005.
- [2] P. Urard, J. Yi, H. Kwon, and A. Gouraud, *High-Level Synthesis: From Algorithm to Digital Circuit*, Springer, New York, NY, USA, 2008.
- [3] G. Martin and G. Smith, "High-level synthesis: past, present, and future," *IEEE Design and Test of Computers*, vol. 26, no. 4, pp. 18–25, 2009.
- [4] G. Constantinides, P. Cheung, and W. Luk, "The multiple wordlength paradigm," in *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '01)*, pp. 51–60, April 2001.
- [5] K. I. 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.
- [6] N. Hervé, D. Ménard, and O. Sentieys, "Data wordlength optimization for FPGA synthesis," in *Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation (SiPS '05)*, pp. 623–628, November 2005.
- [7] G. Caffarena, G. A. Constantinides, P. Y. K. Cheung, C. Carreras, and O. Nieto-Taladriz, "Optimal combined word-length allocation and architectural synthesis of digital signal processing circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 5, pp. 339–343, 2006.
- [8] K.-I. Kum and W. Sung, "Word-length optimization for high level synthesis of digital signal processing systems," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, pp. 569–578, 1998.
- [9] V. Agrawal, A. Pande, and M. M. Mehendale, "High level synthesis of multi-precision data flow graphs," in *Proceedings of the 14th International Conference on VLSI Design (VLSI '01)*, pp. 411–416, January 2001.
- [10] S. Tallam and R. Gupta, "Bitwidth aware global register allocation," in *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '03)*, pp. 85–96, 2003.
- [11] 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)*, pp. 856–861, 2005.
- [12] G. Caffarena, J. A. Lopez, G. Leyva, C. Carreras, and O. Nieto-Taladriz, "Architectural synthesis of fixed-point DSP datapaths using FPGAs," *International Journal of Reconfigurable Computing*, vol. 2009, Article ID 703267, 14 pages, 2009.
- [13] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," in *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI '00)*, pp. 108–120, June 2000.
- [14] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and Synthesis for On-Chip Multicycle Communication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 550–564, 2004.
- [15] B. Le Gal, C. Andriamisaina, and E. Casseau, "Bit-width aware high-level synthesis for digital signal processing systems," in *Proceedings of the IEEE International Systems-on-Chip Conference (SOC '06)*, pp. 175–178, September 2006.
- [16] P. Coussy, G. Lhairech-Lebreton, and D. Heller, "Multiple word-length high-level synthesis," *Eurasip Journal on Embedded Systems*, vol. 2008, no. 1, Article ID 916867, 2008.
- [17] M. Molina, R. Ruiz-Sautua, J. Mendias, and R. Hermida, "Exploiting bitlevel design techniques in behavioural synthesis," in *High-Level Synthesis, From Algorithm to Digital Circuit*, Springer, New York, NY, USA, 2008.
- [18] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs*, Oxford University Press, Oxford, UK, 2000.
- [19] I. Koren, *Computer Arithmetic Algorithms*, CRC Press, Natick, Mass, USA, 2nd edition, 2002.
- [20] W. J. Townsend, E. E. Swartzlander, and J. A. Abraham, "A comparison of Dadda and Wallace multiplier delays," in *Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, vol. 5205 of *Proceedings of SPIE*, pp. 552–560, 2003.
- [21] B. Le Gal and E. Casseau, "Word-length aware DSP hardware design flow based on high-Level synthesis," *Journal of Signal Processing Systems*, pp. 1–17, 2010.

- [22] A. Sharma and R. Jain, "Estimating architectural resources and performance for high-level synthesis application," in *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pp. 355–360, June 1993.
- [23] C. Carreras, J. A. Lopez, and O. Nieto-Taladriz, "Bit-width selection for data-path implementations," in *Proceedings of the 12th International Symposium on System Synthesis (ISSS '99)*, pp. 114–119, Washington, DC, USA, 1999.