Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 703267, 14 pages
doi:10.1155/2009/703267

Research Article

Architectural Synthesis of Fixed-Point DSP Datapaths Using FPGAs

Gabriel Caffarena,[1] Juan A. López,[1] Gerardo Leyva,[2] Carlos Carreras,[1] and Octavio Nieto-Taladriz[1]

[1] *Departamento de Ingeniería Electrónica, Universidad Politécnica de Madrid, Ciudad Universitaria s/n, 28040 Madrid, Spain*
[2] *Departamento de Sistemas Electrónicos, Universidad Autónoma de Aguascalientes, Ciudad Universitaria s/n, 20100 Aguascalientes, Mexico*

Correspondence should be addressed to Gabriel Caffarena, gabriel@die.upm.es

Received 25 February 2009; Accepted 28 August 2009

Recommended by Cesar Torres

We address the automatic synthesis of DSP algorithms using FPGAs. Optimized fixed-point implementations are obtained by means of considering (i) a multiple wordlength approach; (ii) a complete datapath formed of wordlength-wise resources (i.e., functional units, multiplexers, and registers); (iii) an FPGA-wise resource usage metric that enables an efficient distribution of logic fabric and embedded DSP resources. The paper shows (i) the benefits of applying a multiple wordlength approach to the implementation of fixed-point datapaths and (ii) the benefits of a wise use of embedded FPGA resources. The use of a complete fixed-point datapath leads to improvements up to 35%. And, the wise mapping of operations to FPGA resources (logic fabric and embedded blocks), thanks to the proposed resource usage metric, leads to improvements up to 54%.

1. Introduction

This paper addresses the architectural synthesis (AS) of Digital Signal Processing (DSP) algorithms implemented using modern FPGAs. High levels of optimization are achieved by means of the use of Multiple wordlength (MWL) fixed-point descriptions of the algorithms and also the use of both LUT-based and embedded FPGA resources. The former reduces implementation costs notably, and the latter minimizes area in FPGA implementations.

The MWL implementation of fixed-point DSP algorithms [1–4] has proved to provide significant cost savings when compared to the traditional uniform wordlength (UWL) design approach. The introduction of MWL issues in AS increases optimization complexity, but it opens the door to significant cost reductions [2, 3, 5, 6].

FPGA devices have been extensively used in the implementation of DSP algorithms, especially due to the recent introduction of specialized embedded blocks (i.e., memory blocks, DSP blocks, etc.). Traditional approaches to estimate FPGA resource usage do not apply to modern FPGAs, which present a heterogeneous architecture composed of both logic fabric and embedded blocks, since they only account for lookup table- (LUT-) based resources [7]. This situation calls for new resource usage metrics that can be integrated as part of automatic synthesis techniques to fully exploit the possibilities that embedded resources offer [8–10].

The current approaches to perform MWL-oriented architectural synthesis are not tuned to modern FPGAs [2, 3] or an efficient distribution between logic fabric and specialized embedded blocks is not applied [11, 12]. Also, the resource set used during the optimization process does not include the multiplexers necessary to transfer data from memory elements to arithmetic resources.

The main contributions of this paper are the following.

(i) The presentation of a novel resource usage metric that guarantees minimum resource usage for heterogeneous FPGA implementations if integrated within an optimization framework.

(ii) The presentation of an architectural synthesis procedure tuned to fixed-point implementations, that handles a complete datapath (functional units, multiplexers, and registers).

(iii) A novel strategy for fixed-point data multiplexing.

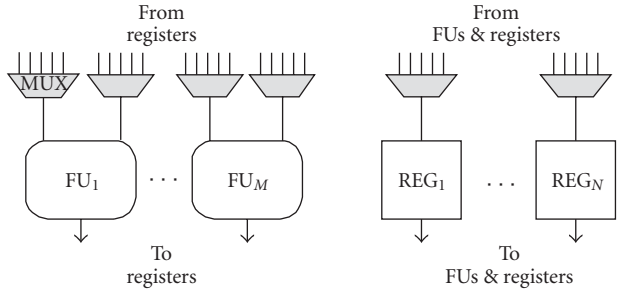


Figure 1: Datapath architecture: FUs, registers, and multiplexers.

The paper is divided as follows. In Section 2, the architectural synthesis of DSP datapaths using multiple wordlength systems and modern FPGAs is introduced. Section 3 deals with the implementation results from synthesizing several DSP benchmarks for different latency constraints and output noise constraints. Finally, in Section 4, the conclusions are drawn.

2. Synthesis of Fixed-Point Datapaths

2.1. Formal Description. This work focuses on the time-constrained resource minimization problem [13]. The notation used is based on [13], and it is similar to that in [2, 4, 6].

Given a sequencing graph $G_{\text{SEQ}}(O, S)$, a maximum latency λ, and a set of resources R (e.g., functional units R_{FU}, registers R_{REG}, and steering logic R_{MUX}), it is the goal of AS to find the time step when each operation is executed (*scheduling*), the types and number of resources forming R (*resource allocation*), and the binding between operations and variables to functional units and registers (*resource binding*) that comply with the constraints, while minimizing cost (i.e., area). As a result, a datapath able to compute the algorithm's operations (see Figure 1) as well as the required control logic is generated.

$G_{\text{SEQ}}(O, S)$ is a formal representation of a single iteration of an algorithm, where O is the set of operations and $S \subset O \times O$ is the set of signals that determine the data flow. We consider $O = O_M \cup O_G \cup O_A \cup O_D \cup O_I \cup O_O$ composed of typical DSP operations: multiplications, gains (multiplication by a constant value), additions, unit delays, and input and output nodes. Signals are in two's complement fixed-point format, defined by the pair (p, n), where p is number of integer bits [4] and n is the wordlength of the signal not including the sign bit (see Section 2.5). The values of the couples (p, n) have been computed during a previously performed wordlength optimization (WLO) [1, 14–16]. See Section 2.5 for more information about the wordlength optimization process.

Functional units (R_{FU}) are in charge of executing the set of operations from O. Registers (R_{REG}) store the data produced by FUs and some intermediate values. Finally, steering logic (R_{MUX}) interconnects FUs and registers by means of multiplexers. The set of functional units $R_{\text{FU}} = R_{A_{\text{LUT}}} \cup R_{M_{\text{LUT}}} \cup R_{M_{\text{EMB}}}$ is composed of LUT-based adders, LUT-based generic multipliers, and embedded multipliers. This set of FUs covers a representative set of modern FPGA devices. An FU $r \in R_{\text{FU}}$ is defined by its type $\text{type}(r) = \{\text{Adder}_{\text{LUT}}, \text{Multiplier}_{\text{LUT}}, \text{Multiplier}_{\text{EMB}}\}$ and by its *size*, that depends on the input wordlengths. An operation is compatible with an FU if they have compatible types and if the size of the operation is smaller than or equal to the size of the FU [4, 6].

Scheduling is expressed by means of function $\varphi : O \to Z^+$, which assigns a start time to each operation. Resource binding is divided into *FU binding* and *register binding*. *FU binding* makes use of the compatibility graph $G_{\text{COMP}}(O \cup R, C)$ [2], which indicates the compatible resources for each $o \in O$ by means of the set of edges $C \subset O \times R$. The binding between operations and resources is expressed by means of function $\beta : O \to R \times Z^+$, where $\beta(o) = \{r, i\}$ indicates that operation o is bound to the ith instance of resource r. The compatibility rules impose that $(o, r) \subset C$. In a similar fashion, register binding links variables $d \in D$ to registers $r \in R_{\text{REG}}$ by means of function $\gamma : D \to R_{\text{REG}} \times Z^+$. The set of variables D is extracted from O considering that there is a variable assigned to the output of each operation from the subset $O_M \cup O_G \cup O_A$ and to each delay o_D connected to another delay. Registers have an associated size n_r that determines the maximum allowed wordlength of the variables bound to them.

The steering logic consists of multiplexers connected at the inputs of FUs and registers. They are in charge of sending data to and from these two types of resources. R_{MUX} is determined by φ, β, and γ, since φ determines when data are generated, β when data are used by FUs, and γ where data are stored.

2.2. Handling Resource Heterogeneity. The recent appearance of specialized blocks in FPGAs calls for new design methods to efficiently exploit their advantages. In [8], it is proposed to use a normalized resource usage vector. Given an FPGA with M different types of resources R_i ($i = 0 \cdots M-1$), each type with a maximum number of $|R_i|$ resources, the resource requirements of a particular design implementation d can be expressed as the following normalized area vector:

$$\hat{\mathbf{A}} \equiv \left\langle \frac{\#r_0}{|R_0|}, \frac{\#r_1}{|R_1|}, \ldots, \frac{\#r_{M-1}}{|R_{M-1}|} \right\rangle, \tag{1}$$

where $\#r_i$ is the number of resources of type R_i used. Two useful norms are the ∞-norm and the 1-norm:

$$\left\|\hat{\mathbf{A}}\right\|_\infty = \max\left\{ \frac{\#r_0}{|R_0|}, \frac{\#r_1}{|R_1|}, \ldots, \frac{\#r_{M-1}}{|R_{M-1}|} \right\}, \tag{2}$$

$$\left\|\hat{\mathbf{A}}\right\|_1 = \sum_{i=0}^{i=M-1} \frac{\#r_i}{|R_i|}. \tag{3}$$

The inverse of ∞-norm represents the number of times that the same implementation of design d can be replicated within the FPGA device (see [8]), and the 1-norm gives information about the overall resource usage of the implementation. Each norm is interesting on its own, but they have some pitfalls. On the one hand, if two implementations have the same ∞-norm this implies that they can be replicated

the same number of times, but there is no way to know which implementation requires less resources. On the other hand, the 1-norm can tell if a design implementation requires less resources than other, but that does not guarantee that the implementation with less resources can be replicated more times than the other. In the work presented here a combination of ∞-norm and 1-norm , called +-norm (*plus-norm*), is proposed and applied. A metric $\|\cdot\|_+$ that exploits the benefits of both norms but none of the drawbacks should fulfill the following conditions:

$$\forall i,j : \left\|\hat{\mathbf{A}}_i\right\|_+ < \left\|\hat{\mathbf{A}}_j\right\|_+ \Longrightarrow \left(\left\|\hat{\mathbf{A}}_i\right\|_\infty < \left\|\hat{\mathbf{A}}_j\right\|_\infty\right) \vee \left(\left(\left\|\hat{\mathbf{A}}_i\right\|_\infty = \left\|\hat{\mathbf{A}}_j\right\|_\infty\right) \wedge \left(\left\|\hat{\mathbf{A}}_i\right\|_1 < \left\|\hat{\mathbf{A}}_j\right\|_1\right)\right). \tag{4}$$

This can be expressed by means of a combination of the two norms

$$\left\|\hat{\mathbf{A}}\right\|_+ = K \cdot \left\|\hat{\mathbf{A}}\right\|_\infty + \left\|\hat{\mathbf{A}}\right\|_1. \tag{5}$$

A feasible solution for K can be found by trying to comply with (6) for areas $\mathbf{A}_1$ and $\mathbf{A}_2$, such that $\mathbf{A}_1$ requires only one type of resource, $\|\hat{\mathbf{A}}_1\|_\infty > \|\hat{\mathbf{A}}_2\|_\infty$ and $\|\hat{\mathbf{A}}_2\|_1$ has the biggest value that $\|\hat{\mathbf{A}}_1\|_\infty$ allows

$$\left\|\hat{\mathbf{A}}_1\right\|_+ > \left\|\hat{\mathbf{A}}_2\right\|_+. \tag{6}$$

First let us find upper bounds for $\|\hat{\mathbf{A}}_2\|_\infty$ and $\|\hat{\mathbf{A}}_2\|_1$

$$\begin{aligned} \left\|\hat{\mathbf{A}}_2\right\|_\infty &\le \left\|\hat{\mathbf{A}}_1\right\|_\infty - \frac{1}{\max(|R_i|)}, \\ \left\|\hat{\mathbf{A}}_2\right\|_1 &\le M \cdot \left(\left\|\hat{\mathbf{A}}_1\right\|_\infty - \frac{1}{\max(|R_i|)}\right). \end{aligned} \tag{7}$$

Substituting (5) and (7) into (6) allows

$$K \cdot \left\|\hat{\mathbf{A}}_1\right\|_\infty + \left\|\hat{\mathbf{A}}_1\right\|_1 > (K+M) \cdot \left(\left\|\hat{\mathbf{A}}_1\right\|_\infty - \frac{1}{\max(|R_i|)}\right). \tag{8}$$

Since $\|\hat{\mathbf{A}}_1\|_1 = \|\hat{\mathbf{A}}_1\|_\infty$ and $\|\hat{\mathbf{A}}_1\|_\infty \le 1$, a possible range of values of K that complies with (4) is expressed in terms of the number of types of resources (M) and the maximum number resources of any type ($\max(|R_i|)$)

$$K > (M-1) \cdot (\max(|R_i|)) - M. \tag{9}$$

K guarantees that for any two implementations d_i and d_j: (i) if $\|\mathbf{A}_i\|_+ < \|\mathbf{A}_j\|_+$ then d_i can be replicated more times than d_j; (ii) if $\|\mathbf{A}_i\|_+ \le \|\mathbf{A}_j\|_+$ then d_i can be replicated more times than d_j, or the same number of times consuming less resources. Therefore, minimizing +-norm implies that the design can be replicated within the FPGA the maximum possible number of times while using the minimum possible number of resources.

The metric +-norm has a low computational cost and it is suitable for integer linear programming approaches [4, 15] and heuristic approaches [6, 17].

2.3. Resource Modeling. Resources are divided into three types: functional units (R_{FU}), registers (R_{REG}), and steering logic (R_{MUX}). The area and latency of FUs and registers (i.e., $\mathbf{A}(r)$ and $l(r)$) are expressed as functions of the input and output wordlength information (p and n). They are obtained by applying curve fitting to hundreds of synthesis results. The use of accurate delay cost functions proved to provide significant performance improvements compared to some other existent naive approaches (from 12% to 63%, see [6]). Registers are assumed to have a zero latency in terms of clock period, which is true as long as the clock frequency enables to comply with setup and hold times.

Note that $\mathbf{A}$ is a vector with as many components as types of FPGA resources. Thus, it is possible to apply the +-norm to $\mathbf{A}$ in order to optimize the total datapath area. Multiplexers and wiring latencies are neglected, which could be easily overcome by means of multiplying the clock period by an empirical factor [18].

2.3.1. MWL Multiplexers. The area of multiplexers in UWL systems is only affected by the data wordlength, which sets the multiplexer size, and by the number of different data sources (e.g., registers or FUs), which determines the multiplexer width. An estimation of the area of an N-input multiplexer of wordlength M for Virtex-II devices is given by

$$A_{\mathrm{MUX}} = M \cdot \frac{N}{4} \quad \text{slices.} \tag{10}$$

This estimation is specific for Virtex-II, Spartan-3, and Virtex-4, since the implementation of multiplexers relies on the combination of 4-input LUTs and dedicated multiplexers. Another FPGA architectures (i.e., Altera's Stratix-II) that make use only of 4-input LUTs would require a different estimation.

In MWL systems, data must be aligned before being processed by FUs or stored by registers. In [19] the problem of data alignment and multiplexing is tackled by means of alignment blocks introduced before multiplexers. In this work, multiplexers are used for both data multiplexing and data alignment, since the combination of these two tasks leads to a reduction in the number of control signals, and therefore, control logic. In addition, the chances for logic optimization are greater than if two separate blocks (an alignment block and a multiplexer) are used.

Alignment is required at the inputs of adders and at the outputs of both adders and multipliers. On one hand, adders require the alignment of their inputs in order to obtain a meaningful result. If an adder is shared to compute several additions (i.e., a_1 and a_2), an alignment block is required to arrange the MSB of the inputs in the right position for each operation (different alignments will be necessary for a_1 and a_2). On the other hand, the output of the different arithmetic operations—both additions and multiplications—in an algorithm can have the MSBs in different positions. Again, if the FUs are shared the output's MSB changes its position depending on the operation executed, therefore, it is necessary to dynamically align the FU's output using in order to store the data in a register.

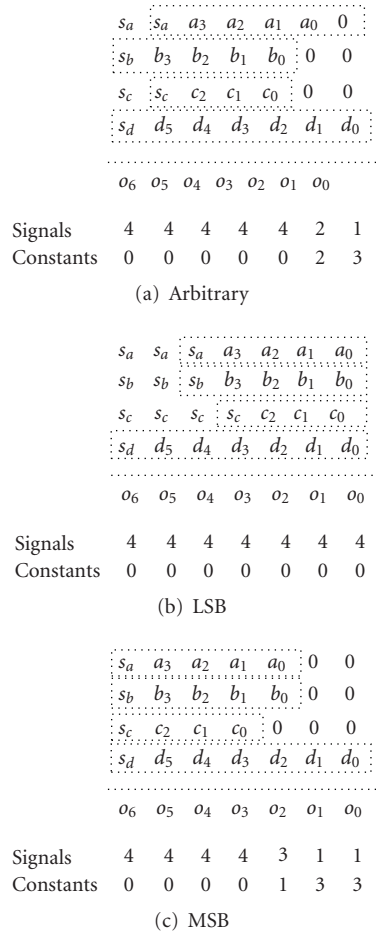


Figure 2: Signal alignment.

Figure 2 presents three different types of alignments for a 4-input multiplexer with inputs signals a, b, c, and d and output o: arbitrary alignment (see Figure 2(a)), least significant bit (LSB) alignment (see Figure 2(b)), and most significant bit (MSB) alignment (see Figure 2(c)). Note that sign extension (see Figures 2(a) and 2(b)) does not offer any opportunity for logic optimization, while zero padding (see Figures 2(a) and 2(c)) does offer it, due to the reduction in the number of signals and the introduction of constant bits (zeros) that can be hard-wired into the multiplexer logic. In fact, it is MSB alignment (see Figure 2(c)) the option which allows a greater logic reduction. Therefore, it is recommended to apply this alignment whenever possible.

A lower bound on the multiplexers' area if the MSB alignment is adopted can be computed as

$$\underline{\mathbf{A}_{\mathbf{MUX}}} = \frac{1}{4}\sum_{i=0}^{N-1}(n_i + 1) \quad \text{slices}, \tag{11}$$

where M is the maximum wordlength present and n_i is the wordlength of signal i.

2.4. Optimization Procedure. In this subsection we extend on the work presented in [6, 17], where the optimization was steered by the ∞-norm and registers and multiplexers were not considered. The optimization procedure is based on Simulated Annealing (SA) [20] and it is shown in Algorithm 1. The inputs are the sequential graph $G_{\text{SEQ}}(O, S)$ and the total latency constraint λ. The optimization procedure determines the set of resources of the datapath $R = R_{\text{FU}} \cup R_{\text{REG}} \cup R_{\text{MUX}}$, the scheduling φ, the FU binding β, and the register binding γ, which define the datapath, the steering logic, and the timing of the control signals.

Table 1: Simulated annealing parameters and conditions.

λ	Latency constraint (in clock cycles)
α_T	Annealing factor (0.95)
$\lvert O\rvert$	Number of operations in algorithm
Equilibrium state	Accepted $> \lvert O\rvert \times \lvert O\rvert$
Frozen state	Iterations $> k \cdot \lvert O\rvert \times \lvert O\rvert$, $k > 1.0$
	3 consecutive times
Restart condition	$\mathbf{A}_{\text{last}} > 1.5 \cdot \mathbf{A}_{\text{min}}$

2.4.1. Simulated Annealing. First, the set of functional units R_{FU}, the set of registers R_{REG}, and the compatibility graph $G_{\text{COMP}}(O, R_{\text{FU}})$ are extracted (line 1). An initial resource mapping m_0 is selected by mapping each operation to the fastest LUT-based resource among the available compatible resources for that operation (line 2), and the area $\mathbf{A}_0$ occupied by the resulting datapath is used as the initial area (line 3). From this point (lines 5–30), the optimization proceeds following the typical SA behavior: the algorithm iterates while producing changes (line 8)—also referred to as *movements*—that modify the value of the cost function (i.e., area) until a certain exit condition is reached. If these changes lead to a cost reduction, they are accepted (line 11), if not, they are accepted with a certain probability which depends on the current temperature T (line 15). The temperature starts at a high value and decreases with time. Most movements are accepted at the beginning of the process, thus enabling a wide design space exploration. As temperature decreases, only those movements which produce small cost deviations are accepted. The temperature is decreased when the *equilibrium state* is reached (line 19). Sporadic *restarting* [21] is also allowed (line 27), which repositions the optimization variables at the last minimum state found.

A summary of simulated annealing parameters and conditions is in Table 1. The annealing factor of 0.95 was chosen empirically aiming at balancing the tradeoff between optimality and solving time.

The variation in cost ΔA is normalized with respect to the initial area $\mathbf{A}_0$ (line 10). This is a simple way to control that the behavior of SA is not affected by the complexity of the algorithm [22], which it is approximated by $\|\widehat{\mathbf{A}}_0\|_n$. The value of n must be set to 1 (or ∞) for homogeneous-architecture FPGAs, and to "+" for heterogeneous-architecture FPGAs.

Input: $G_{\text{SEQ}}(O, S)$, λ
Output: $R = R_{\text{FU}} \cup R_{\text{REG}} \cup R_{\text{MUX}}$, φ, β, γ
(1) Extract $G_{\text{COMP}}(O, R_{\text{FU}})$, R_{FU}, R_{REG}
(2) Find initial mapping m_0
(3) Compute initial area $\mathbf{A}_0$ from m_0
(4) $\mathbf{A}_{\text{min}} = \mathbf{A}_{\text{last}} = \mathbf{A}_0$
$m_{\text{min}} = m_{\text{last}} = m_0$
$T = T_0$
iteration = accepted = exit = 0
(5) **while** ¬ *exit condition* **do**
(6) $m = m_{\text{last}}$
(7) iteration = iteration + 1
(8) Perform change to current m
(9) Compute area $\mathbf{A}$ from m (Algorithm 2)
(10) $\Delta\text{A} = (\|\hat{\mathbf{A}}_{\text{last}}\|_n - \|\hat{\mathbf{A}}\|_n)/\|\hat{\mathbf{A}}_0\|_n$
(11) **if** $\Delta A < 0$ **then**
(12) $\hat{\mathbf{A}}_{\text{min}} = \hat{\mathbf{A}}_{\text{last}} = \mathbf{A}$
$m_{\text{min}} = m_{\text{last}} = m$
accepted = accepted + 1
(13) **else**
(14) $p = e^{\Delta A/T}$, $r = \text{rand}[0 \cdots 1)$
(15) **if** $r \leq p$ **then**
(16) $\mathbf{A}_{\text{last}} = \mathbf{A}$, $m_{\text{last}} = m$
accepted = accepted + 1
(17) **end if**
(18) **end if**
(19) **if** *equilibrium state* **then**
(20) $T = \alpha_T \cdot T$
(21) iterations = accepted = exit = 0
(22) **else if** *frozen state* **then**
(23) $T = \alpha_T \cdot T$
(24) iterations = accepted = 0
(25) exit = exit + 1
(26) **end if**
(27) **if** *restart condition* **then**
(28) $\mathbf{A}_{\text{last}} = \mathbf{A}_{\text{min}}$
$m_{\text{last}} = m_{\text{min}}$
(29) **end if**
(30) **end while**

Algorithm 1: Optimization procedure.

The changes on the cost function (line 8) are performed by applying with equal probabilities the following movements to the resource mapping function m:

(i) M_A: map an operation $o \in O$ to a non mapped resource,

(ii) M_B: map an operation o to another already mapped resource,

(iii) M_C: swap the mapping of two compatible operations mapped to different resources.

2.4.2. Area Computation. The computation of the area cost is shown in Algorithm 2. First, it is checked whether the current resource mapping m complies with latency λ (lines 1–4). If it does not, the actual latency λ' is computed. Later on (line 26), any deviation from the design constraints is penalized by means of increasing the area cost of the solution. Thus, solutions that do not meet the latency constraint are included within the design space exploration [23]. Even though these solutions are never accepted as valid, their inclusion allows a wider architecture exploration than rejecting solutions that do not comply with λ.

Then, the resource allocation and resource binding that minimizes FU area is sought by means of a loop where several list-based scheduling operations are performed (lines 5–18). The purpose of the loop is to check different combinations on the number of instances of the resources. Both lower [24] and upper bounds on the number of instances for each resource are computed (line 6). All combinations of possible instances are computed and stored in the set of vectors I. The list-based scheduling performs an ASAP scheduling to the operations sorted by mobility in ascendant order, providing a fast way to find a valid solution. Note that the size of I is being pruned while the loop iterates; all combinations of FU instances that require areas greater than the minimum found so far are removed (line 15). Thus, resource allocation is sped up.

Once the minimum FU area scheduling is found, the datapath is defined. The tasks of register binding and multiplexer allocation are not commonly included within the optimization loop, in spite of their impact in the final architecture. In this work, these two tasks are part of the optimization procedure.

Register binding is performed by applying a *left-edge* algorithm [13]. Inputs signals are supposed to be available for all λ cycles and do not require storing. Each variable assigned to a delay is initially assigned a register, and after that, the left-edge algorithm is applied as usual.

From sets R_{FU} and R_{REG} and functions φ, β, and γ it is possible to extract the steering logic resources R_{MUX}. Registers have a single multiplexer (see Figure 1), while FUs have two. A goal of multiplexer definition is to maximize the use of the MSB alignment. This aligment can be applied directly to registers and multipliers. However, adders require that the inputs must be aligned to each other. Thus, if an MSB alignment if applied to the mux connected to one of the inputs, it is not possible to do so for the remaining mux, and vice versa. Finally, the control signals can be easily extracted from the scheduling contained in φ.

The area vector is computed by adding the area of each resource multiplied by the number of instances required (line 25). If $\lambda' > \lambda$ the area is penalized by means of factor α_λ. If the implementation does not comply with the latency constraint and if the resulting penalized area is smaller than A_{min}, then the area is forced to be bigger than A_{min} (see line 28).

Summarizing, the optimization procedure is actually controlled by changing iteratively the mapping between operations and FUs. These changes impact on the structure of the datapath and, therefore, on its area cost, which is the function to be minimized. This method provides a robust way to simultaneously perform the tasks of scheduling, resource allocation, and resource binding for multiple wordlength systems. This procedure was satisfactorily applied in [25].

2.5. Wordlength Optimization: A Case Study. Let us introduce this section through a simple LTI case study (Algorithm 3).

Input: $G_{\text{SEQ}}(O, E)$, λ
Output: $R = R_{\text{FU}} \cup R_{\text{REG}} \cup R_{\text{MUX}}$, φ, β, γ, $\mathbf{A}$
(1) Compute minimum latency λ' for mapping m
(2) **if** $\lambda' < \lambda$ **then**
(3) $\lambda' = \lambda$
(4) **end if**
(5) Find set of functional units $R'_{\text{FU}} = \{r'_o, \ldots, r'_{N-1}\}$ with mapped operations
(6) for all $r' \in R'_{\text{FU}}$: Compute instances lower bound $\underline{\text{inst}}(r')$ [24] and upper bound $\overline{\text{inst}}(r')$
(7) $I = \{$all vectors ranging from $\langle \underline{\text{inst}}(r_0), \ldots, \underline{\text{inst}}(r_{N-1})\rangle$ to $\langle \overline{\text{inst}}(r_0), \ldots, \overline{\text{inst}}(r_{N-1})\rangle\}$
(8) $\mathbf{A}_{\text{FUmin}} = \infty$
(9) **for** $\mathbf{i} \in I$ **do**
(10) for all $r'_j \in R'_{\text{FU}}, \text{inst}(r') = \mathbf{i}[j]$
(11) $\mathbf{A}' = \sum_{r' \in R'_{\text{FU}}} \mathbf{A}(r') \cdot \text{inst}(r')$
(12) **if** $\|\hat{\mathbf{A}}'\|_n < \| \hat{\mathbf{A}}_{\text{FUmin}} \|_n$ **then**
(13) **if** list_scheduling(λ', m) **then**
(14) $\mathbf{A}_{\text{FUmin}} = \mathbf{A}$
$\hat{i}_{\text{min}} = \mathbf{i}$
$\varphi_{\text{min}} = \varphi$
$\beta_{\text{min}} = \beta$
(15) $I = \{\mathbf{i} \in I : \| \sum_{r' R'_{\text{FU}}} \hat{\mathbf{A}}(r') \cdot \text{inst}(r')\|_n < \| \hat{\mathbf{A}}_{\text{FUmin}} \|_n\}$
(16) **end if**
(17) **end if**
(18) **end for**
(19) for all $r'_j \in R'_{\text{FU}}, \text{inst}(r') = \mathbf{i}_{\text{min}}[j]$
(20) $\varphi_{\text{min}} = \varphi$
$\beta_{\text{min}} = \beta$
(21) Extract D, R_{REG}
(22) Compute register binding
(23) Extract R_{MUX}
(24) $R = R_{\text{FU}} \cup R_{\text{REG}} \cup R_{\text{MUX}}$
(25) $\mathbf{A}' = \sum_{r \in R} \mathbf{A}(r) \cdot \text{inst}(r)$
(26) $\alpha_\lambda = (\lambda' - \lambda)/\lambda$
(27) $\mathbf{A} = \mathbf{A}' \cdot (1 + \alpha_\lambda)$
(28) **if** $\|\hat{\mathbf{A}}\|_n < \|\hat{\mathbf{A}}_{\text{min}}\|_n$ **then**
(29) $\mathbf{A} = \mathbf{A}_{\text{min}} \cdot (1 + \alpha_\lambda)$
(30) **end if**

Algorithm 2: Computation of area cost.

Input: $a, b, c \in (-1/2, 1/2)$, uniformly distributed
Output: d
(1) **while** true **do**
(2) Get new value of a, b, and c
(3) $m1 = a * 2.384$
(4) $m2 = b * 0.0036$
(5) $s1 = m1 + m2$
(6) $s2 = s1 + c$
(7) New value of output: $d = s2$
(8) **end while**

Algorithm 3: Case study.

Algorithm 3 performs the weighted summation of three signals. The operations involved are two constant multiplications (i.e., gains) and two additions. There are a total of 8 signals.

The goal of WLO is to define the fixed-point format for each signal that enables to produce a hardware implementation of the algorithm. The fixed-point format, as mentioned in Section 2.1, is composed of the pair (p, n). Thus, the ultimate goal of WLO is to find the proper set of (p, n) pairs to optimize the hardware realization of an algorithm. Figure 3 depicts the meaning of this parameters: p is the distance in bits from the fractionary point to the MSB (a zero distance implies that there is no integer part in the number); n is the number of bits used to represent the number without considering the sign bit. A common way to address WLO is to split it into two sequential subtasks: *scaling*, where the values of p are selected, and *wordlength selection*, where the values of n are chosen.

Scaling is performed by means of performing a floating-point simulation and gathering the maximum absolute value of each signal s and computing:

$$p_s = \lfloor \log(\max|s|) \rfloor + 1. \tag{12}$$

Once scaling is accomplished, the values of p are fixed and the values of n are obtained through an optimization process (wordlength selection). The number of bits assigned

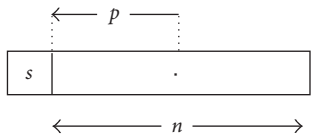


Figure 3: Fixed-point format.

Table 2: UWL versus MWL wordlength optimization.

Signal	$\sigma^2 = 10^{-5}$		$\sigma^2 = 10^{-6}$	
	UWL (p,n)	MWL (p,n)	UWL (p,n)	MWL (p,n)
a	(1,9)	(−1,8)	(1,11)	(−1,10)
b	(1,9)	(−1,2)	(1,11)	(−1,5)
c	(1,9)	(−1,7)	(1,11)	(−1,8)
$m1$	(1,9)	(1,9)	(1,11)	(1,10)
$m2$	(1,9)	(−5,3)	(1,11)	(−5,4)
$s1$	(1,9)	(1,9)	(1,11)	(1,10)
$s2$	(1,9)	(1,8)	(1,11)	(1,10)
d	(1,9)	(1,8)	(1,11)	(1,10)

to a signal (i.e., n) determines the quantization noise that the signal introduces, and, therefore, it has a high impact in the final precision of the system, producing an error in the output signal. During the optimization process different combinations of n are tried in order to look for a particular set that minimizes cost (i.e., area, speed, power) while complying with the output error constraint. The error of the system is typically measured in terms of the peak error value [5, 26], the signal to quantization noise ratio (SQNR) [11, 27], and the variance of the output error [15]. In this work, we adopt the variance of the output error (σ^2).

Table 2 contains the fixed-point formats (i.e., (p,n)) of the signals of Algorithm 3 for both UWL and MWL WLO approaches for different error constraints ($\sigma^2 = 10^{-5}$ and $\sigma^2 = 10^{-6}$). The UWL synthesis is achieved by computing the minimum values of p and n that if applied to all signals the fixed-point realization complies with the noise constraint [15]. The MWL synthesis is achieved by means of an SA-based approach, which minimized the area of a resource-dedicated implementation (with no resource sharing) [28].

Let us focus on the results for $\sigma^2 = 10^{-5}$. The UWL approach clearly requires longer wordlengths than the MWL approach. The main reason for this is that the UWL optimization is far too simple. Also, note that some signals' wordlengths are decreased considerably in the MWL approach (b and $m2$). This is due to the fact that signal b is multiplied by a small constant, so the quantization noise introduced is also small. Similar results are also present for $\sigma^2 = 10^{-6}$. In this case the values of n are bigger since the error constraint is more restrictive.

Summarizing, the MWL approach enables the generation of fixed-point realizations that require a small number of bits. The only drawback is that the complexity of the design process is increased and techniques, such as the proposal in this section, are required.

3. Results

Here, the implementation results are presented. The following benchmarks are used:

(i) ITU RGB to YCrCb converter (ITU) [15],

(ii) 3rd-order lattice filter (LAT_3) [29],

(iii) 4th-order IIR filter (IIR_4) [30],

(iv) 8th-order linear-phase FIR filter (FIR_8).

All algorithms are assigned 8-bit inputs and 12-bit constant coefficients. The algorithm implementations have been tested under different latency and output noise constraint scenarios assuming a system clock of 125 MHz. In particular, the noise constraints were $\sigma^2 = \{10^{-k}, 10^{-(k+1)}, 10^{-(k+2)}\}$, where k is the minimum number that makes 10^{-k} as close as possible to the variance of the quantization noise that would present the output of the benchmark if quantized to 8 bits ($\sigma^2 = (2^{-2n+2p}/12)|_{n=7}$).

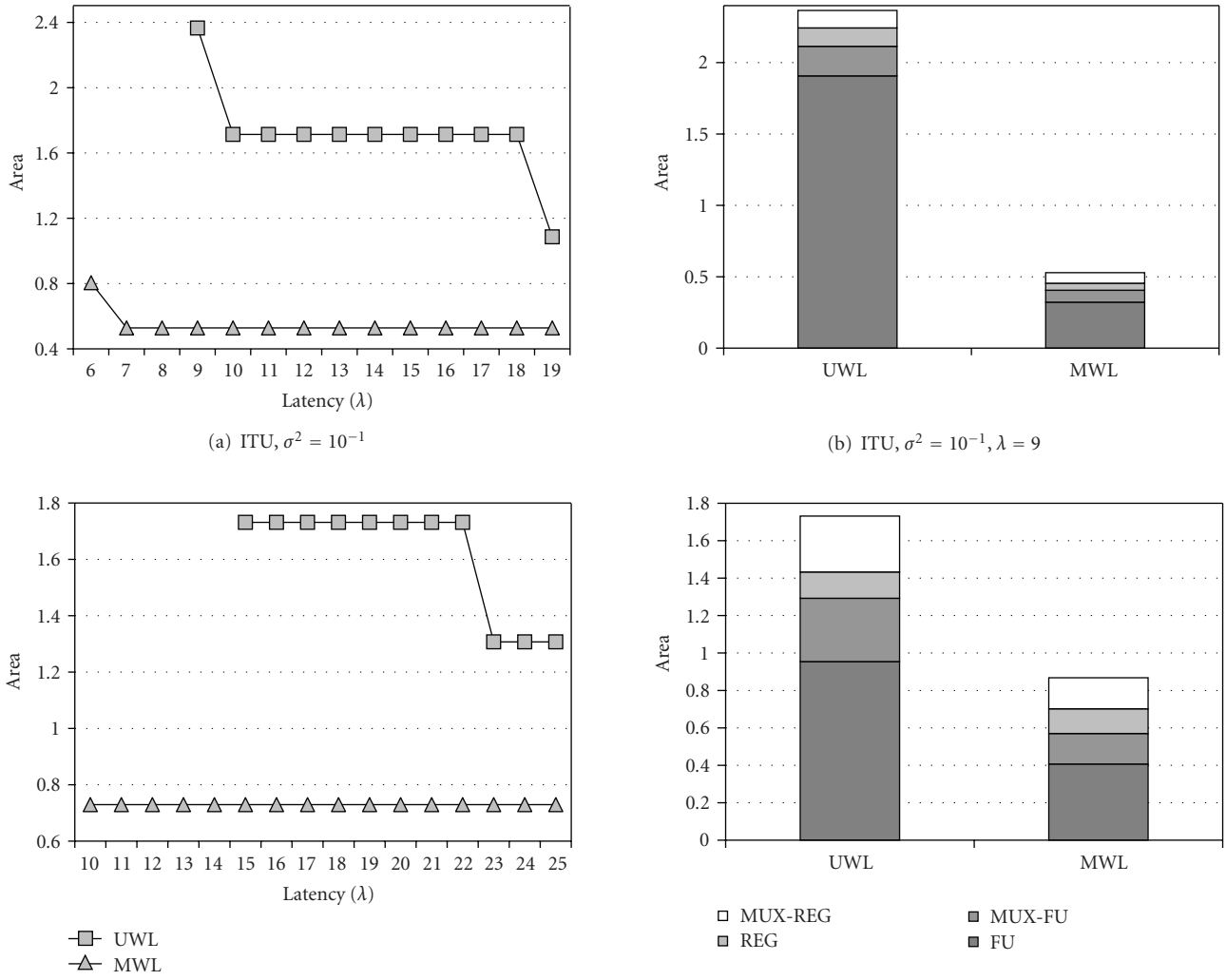
The target devices belong to the Xilinx Virtex-II family. The area results are normalized with respect to the XC2V40 device (256 slices, 4 embedded 18 × 18 multipliers) and expressed according to (2). For instance, an area vector with ∞-norm equal to or smaller than 1.0 implies that the device XC2V40 is the smallest-cost device able to hold the design; whereas a ∞-norm greater than 1.0 and equal to or smaller than 2.0 implies that the smallest-cost device able to hold the design is the XC2V80, and so on.

Before AS, each algorithm is translated to a fixed-point specification by means of two wordlength optimization procedures, that follow a UWL approach and an MWL approach, respectively.

The area results in this section are computed using the resource model explained in Section 2.3, which provides a good estimation of actual synthesis results.

3.1. Uniform Wordlength versus Multiple Wordlength Synthesis: Homogeneous Architectures. Figures 4 and 5 display results on the comparison of UWL versus MWL synthesis using a homogeneous-resource architecture (i.e., only LUT-based resources). Note that the subfigures are arranged in couples, which are related to the same benchmark. The left subfigures depict the area versus latency curves for a particular output noise constraint (see Figures 4(a), 4(c), 5(a), and 5(c)), while the right subfigures contain the detailed resource distribution graph of a particular point of its counterpart (see Figures 4(b), 4(d), 5(b), and 5(d)). Let us define $\lambda_{\min}^{\text{UWL-HOM}}$ as the minimum latency attainable for a UWL-homogeneous implementation of an algorithm, and $\lambda_{\min}^{\text{MWL-HOM}}$ the equivalent for an MWL-homogeneous implementation. The latency used for the experiments ranges from $\lambda_{\min}^{\text{MWL-HOM}}$ to $\lambda_{\min}^{\text{UWL-HOM}} + 10$.

Figures 4(a) and 4(b) contain the implementation results of the ITU benchmark with an output noise variance of 10^{-1}. Figure 4(a) depicts how both the UWL and MWL areas decrease as long as the latency increases. This is expected since the greater the latency the greater the chance of FU reuse. The comparison of the two implementation curves yields that the improvement obtained by means of



(a) ITU, $\sigma^2 = 10^{-1}$

(b) ITU, $\sigma^2 = 10^{-1}, \lambda = 9$

(c) LAT_3, $\sigma^2 = 10^{-5}$

(d) LAT_3, $\sigma^2 = 10^{-5}, \lambda = 22$

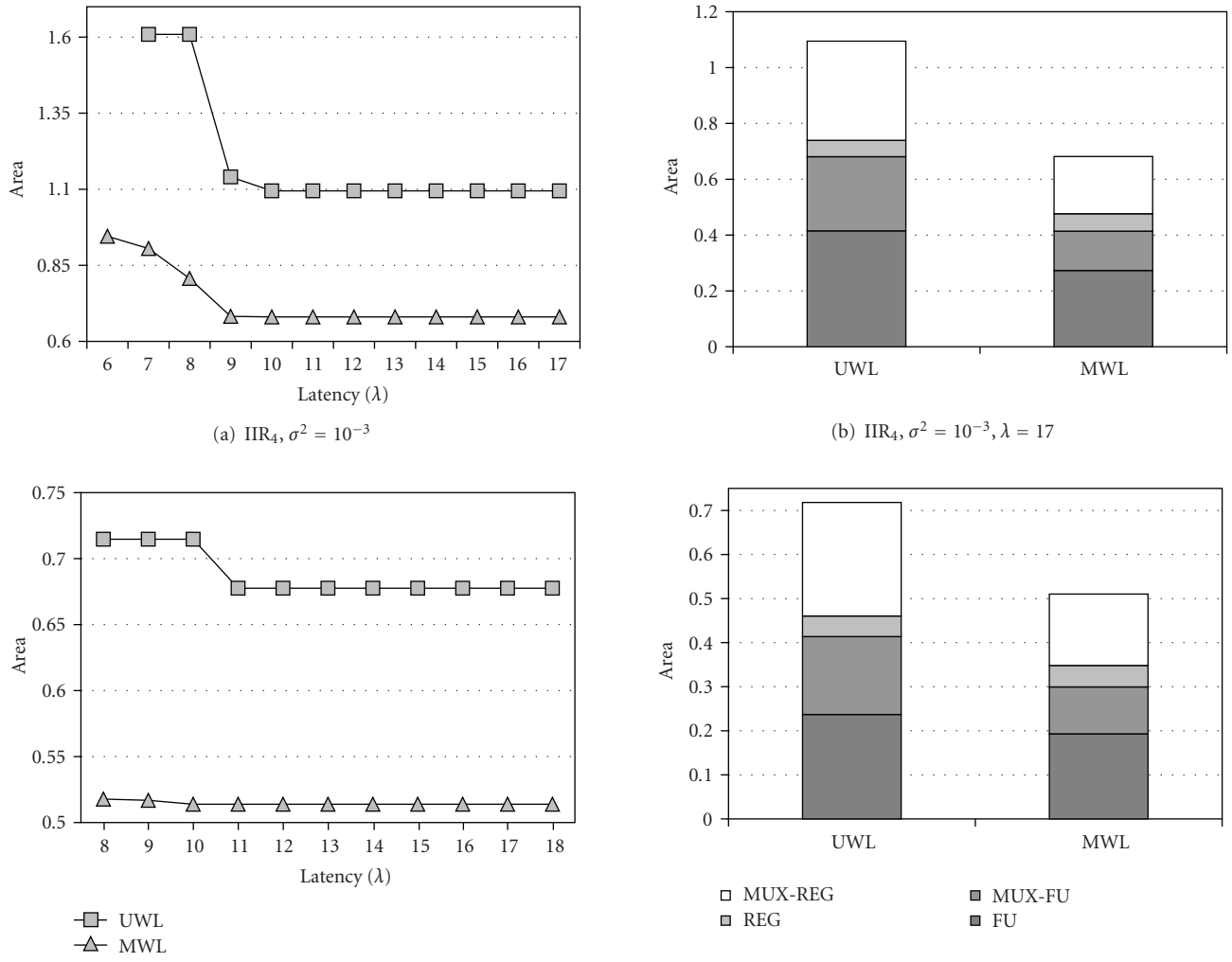
Figure 4: UWL versus MWL: homogeneous implementations (I).

using an MWL approach ranges from 51% to 77%. Also, the minimum latency that each implementation achieves differs considerably. The fine-grain tradeoff between area and quantization noise performed by the MWL approach allows important area reductions when compared to the UWL approach. Figure 4(b) displays the detailed resource distribution for the ITU UWL and MWL implementations correspondant to $\sigma^2 = 10^{-1}$ and $\lambda = 9$. The overall area savings are 77%, and it is due to the fact that the wordlengths of the majority of signals, which impact on FUs, multiplexers and registers, have been highly reduced; FUs' area has been reduced 83%, FU's multiplexers 59%, registers 62%, and registers' multiplexers 39%. It is important to highlight that the area due to multiplexers and registers, although smaller than the FUs' area, makes up a significant part of the total area (20% for UWL and 39% for MWL). Hence the importance of including its cost within the optimization loop is analyzed in Section 3.4.

The other benchmarks also show large area improvements: LAT_3 up to 49% (see Figure 4(c)), IIR_4 up to 49% (see Figure 5(a)), and FIR_8 up to 28% (see Figure 5(c)). As observed in the detailed resource distribution subfigures (see Figures 4(d), 5(b), and 5(d)), the area of the majority of the resources has been highly decreased. Also, it is noted that the percentage of area devoted to multiplexation and data storing is high in proportion to the overall implementation area. The minimum latency is also improved (see Figures 4(a), 4(b), and 5(a)).

In Figure 4(c) the MWL area does not decrease as long as the latency increases. This is due to the fact that the wordlengths are small enough as to allow maximum resource sharing for all latencies, thus the coincidence in the area results for the MWL implementations. This situation might change if a different error constraint (σ^2) is applied during WLO.

Table 3 contains the implementation results for all the benchmarks corresponding to three different quantization noise scenarios. For each quantization scenario the latency ranges from $\lambda_{\min}^{\text{UWL-HOM}}$ to $\lambda_{\min}^{\text{UWL-HOM}} + 10$, and the minimum, maximum, and mean values of the area improvements



(a) IIR_4, $\sigma^2 = 10^{-3}$

(b) IIR_4, $\sigma^2 = 10^{-3}$, $\lambda = 17$

(c) FIR_8, $\sigma^2 = 10^{-4}$

(d) FIR_8, $\sigma^2 = 10^{-4}$, $\lambda = 8$

Figure 5: UWL versus MWL: homogeneous implementations (II).

Table 3: UWL versus MWL for homogeneous architectures.

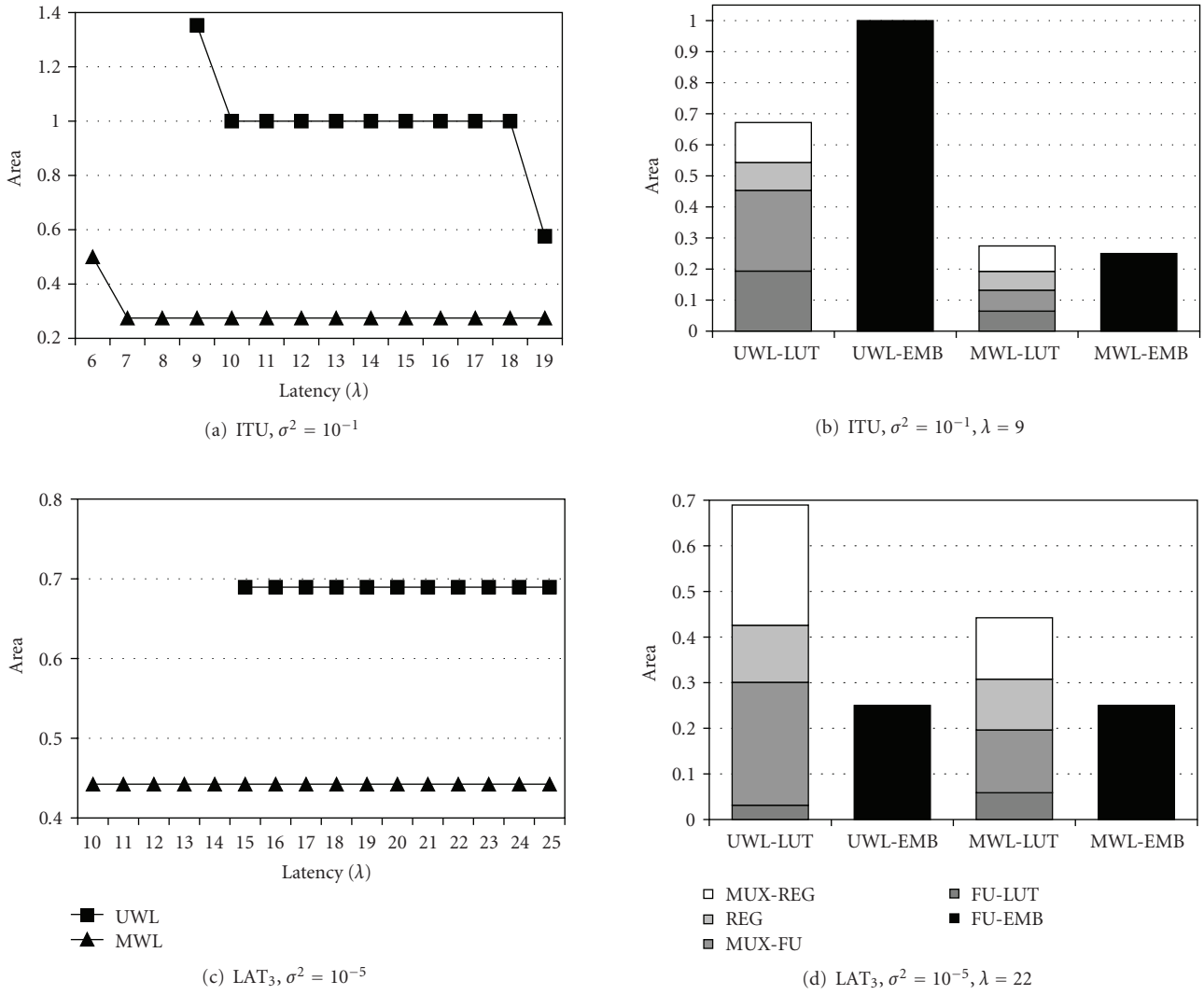
Bench.	σ^2	Area improvement (%)		
		Min	Max	Mean
ITU	10^{-1}	51.36	77.66	68.32
	10^{-2}	48.44	76.31	66.41
	10^{-3}	46.51	75.40	65.13
LAT_3	10^{-3}	44.07	44.07	44.07
	10^{-4}	33.66	33.66	33.66
	10^{-5}	33.62	49.89	45.42
IIR_4	10^{-3}	37.85	49.86	39.69
	10^{-4}	34.30	63.55	50.65
	10^{-5}	37.08	64.99	52.72
FIR_8	10^{-3}	40.28	47.68	43.54
	10^{-4}	24.16	28.10	25.14
	10^{-5}	22.04	25.73	22.82
All		22.04	77.66	46.46

obtained by the MWL implementations in comparison to the UWL implementations are computed. The first column in the table contains the name of the benchmark. The second, the output noise variance. And the third column contains area improvement values. The last row holds the minimum, maximum, and average improvements considering all results simultaneously.

The area improvements obtained are remarkable: mean improvements range from 47% to 77%. Note that the minimum improvements obtained for all benchmarks are quite close to both the maximum and the mean. The results clearly show that an MWL-based AS approach achieves significant area reductions.

Regarding latency, the minimum latency achievable by UWL implementations is reduced in average a 22% by means of MWL AS.

3.2. Uniform Wordlength versus Multiple Wordlength Synthesis: Heterogeneous Architectures. Figures 6 and 7 contain results on the comparison of UWL versus MWL synthesis



(a) ITU, $\sigma^2 = 10^{-1}$

(b) ITU, $\sigma^2 = 10^{-1}, \lambda = 9$

(c) LAT_3, $\sigma^2 = 10^{-5}$

(d) LAT_3, $\sigma^2 = 10^{-5}, \lambda = 22$

Figure 6: UWL versus MWL: homogeneous implementations (I).

using a heterogeneous-resource architecture (i.e., both LUT-based and embedded resources present). The arrangement of figures is similar to that of the previous subsection. Now, the latency ranges from $\lambda_{\min}^{\text{MWL-HET}}$ to $\lambda_{\min}^{\text{UWL-HET}} + 10$ (HET indicates heterogeneous implementations).

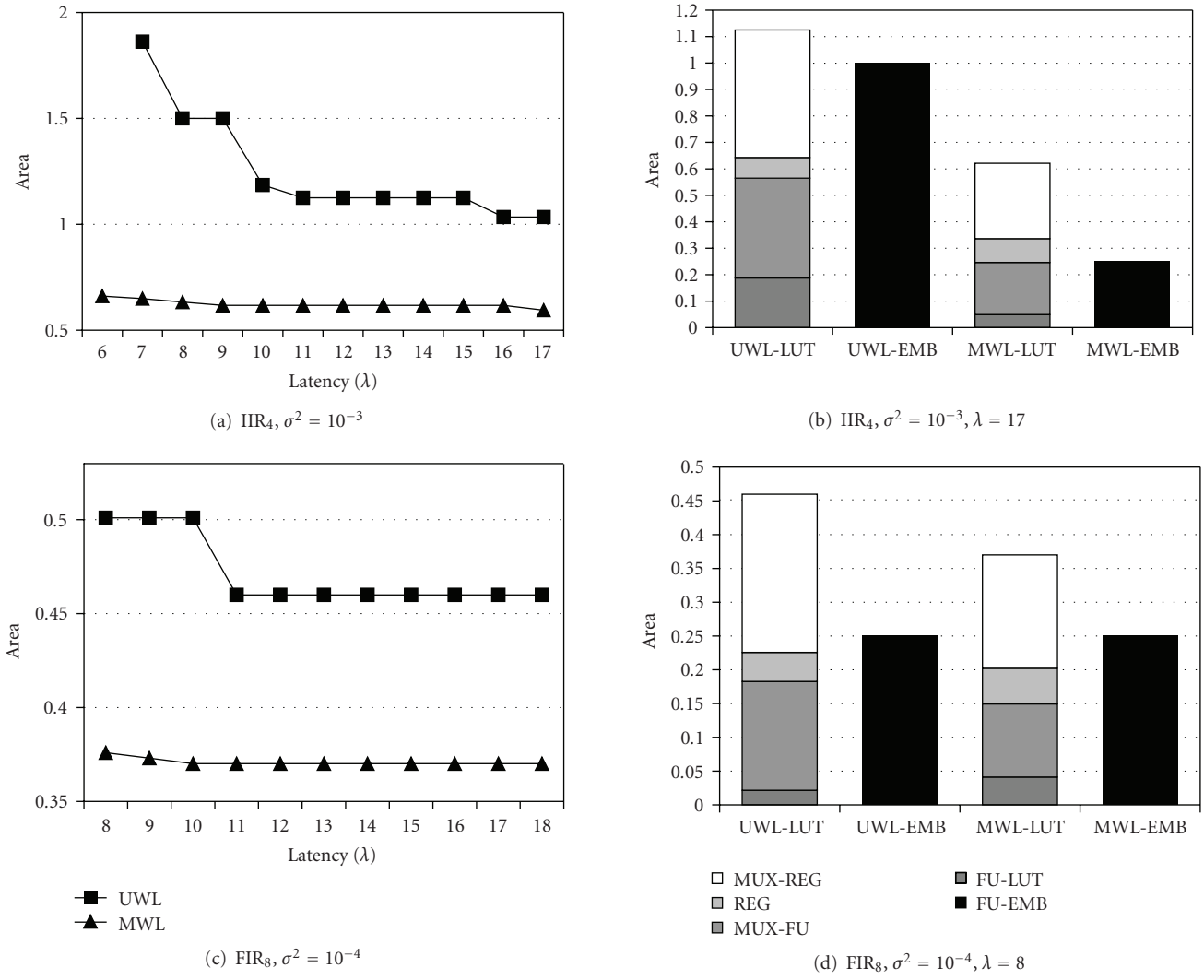
Figures 6(a), 6(c), 7(a), and 7(c) contain the implementation area versus latency curves. The graphs clearly show how the area is reduced by means of an MWL synthesis: ITU up to 79% (see Figure 6(a)), LAT_3 up to 35% (see Figure 6(c)), IIR_4 up to 40% (see Figure 7(a)), and FIR_8 up to 26% (see Figure 7(b)). The detailed resource distribution in Figures 6(b), 6(d), 7(b), and 7(d) shows how the majority of resources are decreased, and in particular the embedded multipliers and the FUs' multiplexers are clearly optimized. For instance, the ITU resource distribution for $\sigma^2 = 10^{-2}$ and $\lambda = 10$ (see Figure 6(b)) shows an overall area reduction of 72%. The LUT-based resources are reduced 59% (LUT-based FUs' area has been reduced 32%, FU's multiplexers 74%, registers 32%, and registers' multiplexers 36%); while the embedded FUs are reduced 75%.

Note that the area of embedded resources for Figures 6(d) and 7(d) is the same for both the UWL and MWL approaches, in fact a single multiplier is being used (1 out of 4). This happens because the wordlengths involved in multiplications, though not the same, are small enough for both UWL and MWL approaches as to enable the use of a single embedded multiplier. However, the LUT-based areas are quite different, and, as a result, the overall resource usage is much smaller for the MWL implementation.

In Figure 6(c) the UWL and MWL areas do not decrease as long as the latency increases. Again, this is due to the fact that the particular wordlengths obtained allow maximum resource sharing for all latencies. Different error constraints (σ^2) might change this situation.

Again, the figures show how the minimum latency can be highly improved by means of an MWL approach. Also, it can be seen that the LUT-based resources are devoted almost entirely to data multiplexing and storing.

Table 4 contains the implementation results of all the benchmarks corresponding to three different quantization



(a) IIR_4, $\sigma^2 = 10^{-3}$

(b) IIR_4, $\sigma^2 = 10^{-3}$, $\lambda = 17$

(c) FIR_8, $\sigma^2 = 10^{-4}$

(d) FIR_8, $\sigma^2 = 10^{-4}$, $\lambda = 8$

Figure 7: UWL versus MWL: homogeneous implementations (II).

noise scenarios. For each quantization scenario the latency ranges from $\lambda_{\text{min}}^{\text{UWL-HET}}$ to $\lambda_{\text{min}}^{\text{UWL-HET}} + 10$, and the minimum, maximum, and mean area improvements obtained by the MWL implementations in comparison to the UWL implementations are computed considering ∞-norm area, the LUT-based area, and the embedded FUs area. The first column in the table contains the name of the benchmark. The second, the output noise variance applied. The third column contains the minimum, maximum, and mean ∞-norm area improvement values. The fourth column contains the minimum, maximum, and mean values of the LUT-based resource area. And the last column contains the minimum, maximum, and mean values of the embedded FUs area.

The area improvements obtained are considerable; ITU obtains up to 80.77%, LAT_3 up to 48.87%, IIR_4 up to 65.13%, FIR_8 up to 38.01%. Note that the minimum improvements obtained for most of the benchmarks are again quite close to both the maximum and the mean.

The LUT-based area reductions are up to 81.07% for ITU, up to 48.87% for LAT_3, up to 65.13% for IIR_4, and up to 43.83% for FIR_8. The embedded resources are only reduced for benchmarks ITU (up to a 75.00%) and IIR_4 (up to 83.33%). Benchmarks LAT_3 and FIR_8 use the minimum possible number of embedded resources (1 embedded multiplier), hence the 0% improvement.

Area improvements up to 80.77% are achieved. The average improvement is 44.88% for the overall area, 42.76% for the LUT-based resources, and 24.03% for the embedded resources. The results clearly show that an MWL-based approach for AS leads to significant area reductions.

As a final note regarding these area results, the authors would like to emphasized that the *plus-norm* has been used during the optimization process, but it is not used to present the results as it cannot be directly related to the percentage of occupation of the FPGA. Thus, the ∞-norm is used instead.

The latency analysis throws that the minimum UWL latency is reduced an average 19% by means of MWL AS.

3.3. MWL Synthesis: Heterogeneous versus Homogeneous. Table 5 contains the implementation results of all the

Table 4: UWL versus MWL for heterogeneous architectures.

Bench.	σ^2	$\|\hat{\mathbf{A}}\|_\infty$ %			A_{LUT} %			A_{EMB} %		
		Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
ITU	10^{-1}	54.85	80.77	73.69	55.56	81.07	63.19	50.00	75.00	72.73
	10^{-2}	52.37	79.71	71.37	52.37	79.71	60.41	50.00	75.00	72.73
	10^{-3}	47.80	77.76	66.71	47.80	77.76	56.33	50.00	75.00	72.73
LAT_3	10^{-3}	48.87	48.87	48.87	48.87	48.87	48.87	0.00	0.00	0.00
	10^{-4}	35.84	35.84	35.84	35.84	35.84	35.84	0.00	0.00	0.00
	10^{-5}	31.70	31.70	31.70	31.70	31.70	31.70	0.00	0.00	0.00
IIR_4	10^{-3}	37.47	41.27	38.33	37.47	45.10	39.05	0.00	0.00	0.00
	10^{-4}	32.97	40.70	34.74	32.97	40.70	34.74	0.00	0.00	0.00
	10^{-5}	40.32	65.13	49.57	40.32	65.13	48.55	50.00	83.33	71.67
FIR_8	10^{-3}	32.45	38.01	33.97	38.79	43.83	39.68	0.00	0.00	0.00
	10^{-4}	27.53	32.83	28.62	27.53	32.83	28.62	0.00	0.00	0.00
	10^{-5}	19.53	26.12	21.17	19.53	26.12	21.17	0.00	0.00	0.00
All		19.53	80.77	44.88	19.53	81.07	42.76	0.00	83.33	24.03

benchmarks corresponding to three different quantization noise scenarios. For each quantization scenario the latency ranges from $\lambda_{\text{min}}^{\text{MWL-HOM}}$ to $\lambda_{\text{min}}^{\text{MWL-HOM}}+10$, and the minimum, maximum, and mean values of the area improvements, in terms of ∞-norm, obtained by the MWL implementations in comparison to the UWL implementations are computed. The first column of the table contains the name of the benchmark. The second, the output noise variance applied. And, the third column contains the minimum, maximum and mean area improvement values.

The area improvements obtained are remarkable; ITU obtains up to 54.76%, LAT_3 up to 43.09%, IIR_4 up to 48.79%, FIR_8 up to 44.68%. Note that, again, the minimum improvements obtained for all benchmarks are quite close to both the maximum and the mean. Area improvements up to 54.76% are achieved, being the average improvement 40.23%. The results clearly show that the inclusion of embedded resources within AS leads to higly optimized DSP implementations.

Regarding latencies, the minimum latency achievable by both homogeneous and heterogeneous implementations is the same for the experiments performed. This is due to the fact that the latency of resources is very similar in the particular conditions used for the tests. The same experiments presented in this section were repeated increasing the constant wordlength to 16 bits, obtaining that heterogeneous implementations reduced 7% the minimum latency in constrast to homogeous implementations.

3.4. Effect of Registers and Multiplexers. In this subsection the effect of including the cost of registers and multiplexers within the optimization loop is investigated. As in the previous experiments, the analysis is performed implementing the benchmarks using different noise and latency constraints. Before AS is applied a gradient-descent quantization [28] is applied according to the given noise constraint. The comparison is done by using Algorithm 1 to perform the AS using two different area cost estimation solutions: (i) Algorithm 2, which is referred to as the *complete* area estimation algorithm, and (ii) a simplified version of Algorithm 2 (*simplified* area estimation algorithm) where the cost of registers and multiplexers is neglected. When the simplified area estimation is used, the cost of registers and multiplexers is included after the optimization loop has finished its execution, using the complete area estimation (Algorithm 2).

Table 5: MWL synthesis: homogeneous versus heterogeneous architectures.

Bench.	σ^2	$\|\hat{\mathbf{A}}\|_\infty$ %		
		Min	Max	Mean
ITU	10^{-1}	40.73	52.69	51.60
	10^{-2}	43.29	53.98	53.01
	10^{-3}	50.45	54.76	54.32
LAT_3	10^{-3}	42.68	43.09	42.72
	10^{-4}	39.36	39.36	39.36
	10^{-5}	38.73	39.74	38.82
IIR_4	10^{-3}	34.83	44.77	36.04
	10^{-4}	32.92	48.23	35.96
	10^{-5}	33.21	48.79	35.79
FIR_8	10^{-3}	21.42	41.46	28.37
	10^{-4}	27.02	44.68	33.24
	10^{-5}	27.46	44.62	33.52
All		21.42	54.76	40.23

Table 6 contains the results of this analysis. The latencies range from $\lambda_{\text{min}}^{\text{ARCH}}$ to $\lambda_{\text{min}}^{\text{ARCH}}$ + 10, where ARCH refers to the type of FPGA architecture used (homogeneous or heterogeneous). The noise constraints are the same used in the previous subsection (three σ^2 for each benchmark), though the results have been combined into a single row. The first column contains the type of FPGA architecture.

Table 6: Complete versus simplified cost estimation: area improvement (%).

Arch.	Bench.	Area improvement		
		Min	Max	Mean
HOM	ITU	0.00	0.95	0.30
	LAT_3	0.71	3.50	1.53
	IIR_4	0.00	5.35	1.26
	FIR_8	0.00	1.77	0.31
HET	ITU	0.00	25.85	1.89
	LAT_3	1.15	5.77	2.52
	IIR_4	0.00	35.57	8.09
	FIR_8	0.00	8.11	0.83
All		0.00	35.57	2.09

The second column indicates the benchmark used. And the fourth column contains the minimum, maximum, and average area improvements obtained by the complete area estimation synthesis in contrast to the simplified area estimation synthesis. The last row includes the minimum, maximum, and mean improvements for all benchmarks.

The average improvements for the different benchmarks range from 0.00% to 8.09%, being the overall average improvement of 2.09%. The maximum improvement found is 35.57%. These results clearly show that failing to include the cost of registers and multiplexer during the optimization procedure can lead to unwanted area penalties.

4. Conclusions

In this paper an architectural synthesis procedure able to produce optimized fixed-point implementations using modern FPGA devices is presented. The key to success is provided by the use of highly accurate models of the datapath resources, a complete datapath resource set that includes multiplexer and registers, a novel method to handle fixed-point data alignment and multiplexing, and also the introduction of a novel resource usage metric that can cope with LUT-based and embedded FGPGA resources.

The AS procedure produces area improvements of up to 80% when compared to uniform-wordlength implementations, and latency improvement of up to 22%. The efficient use of embedded resources achieves area improvements of up to 54% when compared to homogeneous implementations. Also, the inefficiency of current FPGA architectures to implement data steering was exposed.

These results are intented to be further improved by means of tightly combining the fixed-point refinement process within the architectural synthesis [4, 31]. Also, the inclusion of the control logic in the resource model is regarded as a future research line.

Acknowledgment

This work was supported by the Spanish Ministry of Education and Science under Research Project TEC2006-13067-C03-03.

References

[1] 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.

[2] G. Constantinides, P. Cheung, and W. Luk, "Heuristic datapath allocation for multiple wordlength systems," in *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE '01)*, pp. 791–796, Munich, Germany, 2001.

[3] 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, Shanghai, China, 2005.

[4] 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*, vol. 53, no. 5, pp. 339–343, 2006.

[5] S. A. Wadekar and A. C. Parker, "Accuracy sensitive word-length selection for algorithm optimization," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '98)*, pp. 54–61, San Jose, Calif, USA, 1998.

[6] G. Caffarena, J. A. López, C. Carreras, and O. Nieto-Taladriz, "High-level synthesis of multiple word-length DSP algorithms using heterogeneous-resource FPGAS," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 675–678, Madrid, Spain, 2006.

[7] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Accurate area and delay estimators for FPGAs," in *Proceedings of the 39th Design Automation Conference (DAC '02)*, pp. 862–869, New Orleans, La, USA, June 2002.

[8] C.-S. Bouganis, G. A. Constantinides, and P. Y. K. Cheung, "A novel 2D filter design methodology for heterogeneous devices," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 13–22, Napa, Calif, USA, April 2005.

[9] X. Liang, J. S. Vetter, M. C. Smith, and A. S. Bland, "Balancing FPGA resource utilities," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '05)*, pp. 156–162, Las Vegas, Nev, USA, June 2005.

[10] A. M. Smith, G. A. Constantinides, and P. Y. K. Cheung, "Fused-arithmetic unit generation for reconfigurable devices using common subgraph extraction," in *Proceedings of the International Conference on Field Programmable Technology (FPT '07)*, pp. 105–112, Kitakyushu, Japan, December 2007.

[11] R. Rocher, D. Menard, N. Herve, and O. Sentieys, "Fixed-point configurable hardware components," *EURASIP Journal of Embedded Systems*, vol. 2006, Article ID 23197, 13 pages, 2006.

[12] N. Hervé, D. Ménard, and O. Sentieys, "About the importance of operation grouping procedures for multiple word-length architecture optimizations," in *Proceedings of the International Workshop on Applied Reconfigurable Computing (ARC '07)*, pp. 191–200, March 2007.

[13] G. De Michelli, *Synthesis and Optimization of Digital Circuits*, Electrical and Computer Engineering series, McGraw-Hill, New York, NY, USA, 1994.

[14] M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie, "An automatic word length determination method," in *Proceedings*

of the IEEE International Symposium on Circuits and Systems (ISCAS '01), vol. 5, pp. 53–56, Sydney, Australia, May 2001.

[15] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1432–1442, 2003.

[16] M. Holzer, B. Knerr, P. Belanoviç, and M. Rupp, "Efficient design methods for embedded communication systems," *EURASIP Journal of Embedded Systems*, vol. 2006, Article ID 64913, 2006.

[17] G. Caffarena, J. A. López, C. Carreras, and O. Nieto-Taladriz, "Optimized implementation of DSP cores on FPGAs using logic-based and embedded resources," in *Proceedings of the International Symposium on System-On-Chip (SoC '06)*, pp. 103–106, Tampere, Finland, November 2006.

[18] R. Enzler, T. Jeger, D. Cottet, and G. Tröster, "High-level area and performance estimation of hardware building blocks on FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FBL '00)*, pp. 525–534, Villach, Austria, August 2000.

[19] K. Schoofs, G. Goossens, and H. De Man, "Bit-alignment in hardware allocation for multiplexed DSP architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '93)*, pp. 289–293, October 1993.

[20] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[21] N. Benvenuto, M. Marchesi, and A. Uncini, "Applications of simulated annealing for the design of special digital filters," *IEEE Transactions on Signal Processing*, vol. 40, no. 2, pp. 323–332, 1992.

[22] H. Orsila, T. Kangas, E. Salminen, and T. D. Hämäläinen, "Parameterizing simulated annealing for distributing task graphs on multiprocessor SoCs," in *Proceedings of the International Symposium on System-On-Chip (SoC '06)*, pp. 1–4, Tampere, Finland, November 2006.

[23] M. Lopez-Vallejo, J. Grajal, and J. Lopez, "Constraintdriven system partitioning," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '00)*, pp. 411–416, Paris, France, March 2000.

[24] S. Y. Ohm, F. J. Kurdahi, and N. D. Dutt, "A unified lower bound estimation technique for high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 5, pp. 458–472, 1997.

[25] G. Caffarena, J. A. López, G. Leyva, C. Carreras, and O. Nieto-Taladriz, "Optmized architectural synthesis of fixed-point datapaths," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '08)*, pp. 85–90, Cancun, Mexico, 2008.

[26] M. López-Vallejo and J. C. López, "On the hardware-software partitioning problem: System modeling and partitioning techniques," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 8, no. 3, pp. 269–297, 2003.

[27] J. A. López, G. Caffarena, C. Carreras, and O. Nieto-Taladriz, "Fast and accurate computation of the round-off noise of linear time-invariant systems," *IET Circuits, Devices and Systems*, vol. 2, no. 4, pp. 393–408, 2008.

[28] M.-A. Cantin, Y. Savaria, and P. Lavoie, "A comparison of automatic word length optimization procedures," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '02)*, vol. 2, pp. 612–615, May 2002.

[29] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons, New York, NY, USA, 1999.

[30] K.-I. Kum, J. Kang, and W. Sung, "AUTOSCALER for C: an optimizing floating-point to integer C program converter for fixed-point digital signal processors," *IEEE Transactions on Circuits and Systems II*, vol. 47, no. 9, pp. 840–848, 2000.

[31] G. Caffarena, *Combined word-length allocation and high-level synthesis of digital signal processing circuits*, Ph.D. dissertation, Universidad Politécnica de Madrid, Madrid, Spain, 2008.