Power Optimization in Heterogenous Datapaths

Alberto A. Del Barrio*, Seda Ogrenci Memik#, Maria C. Molina,* Jose M. Mendias*, Roman Hermida*
*Architecture and Technology of Computing Systems, Universidad Complutense de Madrid (UCM), Spain
#Department of Electrical Engineering and Computer Science (EECS), Northwestern University, Evanston, Illinois
albertodbg@fdi.ucm.es, seda@eecs.northwestern.edu, {cmolinap, mendias, rhermida}@dacya.ucm.es

***Abstract*[1][2]—Heterogenous datapaths maximize the utilization of functional units (FUs) by customizing their widths individually through fragmentation of wide operands. In comparison, slices in large functional units in a homogenous datapath could be spending many cycles not performing actual useful work. Various fragmentation techniques demonstrated benefits in minimizing the total functional unit area. Upon a closer look at fragmentation techniques, we observe that the area savings achieved by heterogenous datapaths can be traded-off for power optimization. Our specific approach is to introduce choices for functional units with power/area trade-offs for different fragmentation and allocation choices, for reducing power consumption while satisfying the area constraint imposed on the heterogenous datapath. As low power FUs in literature produce an area penalty, a methodology must be developed in order to introduce them in the HLS flow while complying with the area constraint. We propose an allocation and module selection algorithms that pursue a trade-off between area and power consumption for fragmented datapaths under a total area constraint. Results show that it is possible to reduce power by 37% on average (49% in the best case). Moreover latency and cycle time will be equal or nearly the same as in the baseline case, which will lead to an energy reduction, too.**

Keywords: low-power, area, HLS

I. INTRODUCTION

The power dissipated by a circuit can be optimized at different levels of abstraction. However, the potential impact of strategic decisions made at the higher levels is likely to be most significant [1-2]. High-Level Synthesis (HLS) techniques have therefore tackled power minimization in various ways. Majority of these techniques focuses on the dynamic power consumption. Although dynamic power still dominates the total power envelope, leakage power is becoming an increasingly larger fraction of total power. Leakage optimizations are generally addressed through lower level design optimizations such as dual threshold gates or reverse body biasing [3-4]. HLS also has some indirect impact on power, for example by minimizing the number of clock cycles at which a given resource is idle or by minimizing the total amount of resources employed in a datapath. Dynamic power, on the other hand, is a function of switching activity, supply voltage level, and the switched capacitive load. Various HLS techniques address these parameters for optimization. However those focused on reducing voltage [19] or frequency [20] require significant changes in the design process and impact technology parameters. Alternatively, other techniques only aim to minimize switching activity or the effective amount of resources required without the need to impose any limitations on circuit and technology parameters. For instance, binding operations with correlated switching activity on the same resource in consecutive cycles diminishes switching activity [21]. However, this may not be sufficient for modules composed of a large amount of logic, e.g. a combinational multiplier. Internal signals within such a complex component can behave differently depending on the specific implementation, even though consecutive inputs supplied to the component at the primary ports are correlated.

Similar to some aforementioned HLS techniques, fragmentation [16-18] would have an indirect impact on power; mainly thanks to the fact that the total effective amount of hardware used at a given clock cycle is reduced. Useless switching activity is produced when executing a narrow operation in a bigger FU. This is the case of heterogeneous specifications, where different sizes and data types are taken into account. Traditional HLS allocation techniques select FUs able to execute the widest operations in the specification and hence, there will be some wasted FU parts when computing narrower operations. Hence, removing these useless parts is a way of reducing both area and power. Fragmentation techniques have been developed in order to tackle this problem. Fragmentation mainly aims to allocate a set of functional units with various width configurations that can execute operations in the specification. The goal is to minimize the total amount of hardware used and in this process some wide operations can be divided into fragments such that they can be executed on a narrow width FU. Scheduling techniques have been proposed to accompany fragmentation, where fragments of the same operation could be scheduled in non-consecutive cycles [17,18]. Thereby, a small set of effective FUs would be utilized to the maximum extent across all clock cycles.

Fragmentation and associated scheduling techniques mentioned above indeed help reduce total area, however, they will also increase circuit latency and frequency. The increase in frequency is obvious, since reduced cycle time (thanks to reduced critical path lengths of the fragmented FUs) will produce this effect. The circuit latency will increase when at least one operation in the critical path is divided into fragments, each of them scheduled in different csteps, for the sake of balancing and maximizing the use of the FU set.

In this paper, we propose a new flow for allocation and binding for fragmented datapaths to explicitly address power optimization instead of relying indirectly on area reduction for power improvement. Besides, to lessen the abovementioned

[1] This work was supported by the Spanish Government Research Grant TIN 2008/00508, the 2009 UCM-Santander Bank Distinguished Visitors Program (AY16/09) and the National Science Foundation Grant No. 0546305

adverse impacts of existing fragmentation-based flows on power we propose a different scheduling approach to comply with the original latency constraint of the homogenous datapath. We start out by utilizing the main principle of fragmentation as a tool for minimizing hardware used per cycle [18]. However, instead of scheduling several fragments of the same operation in different control steps (csteps), as practiced by prior techniques [17,18], FUs are fragmented, but operations are executed in a set of linked FUs in the csteps when they were originally scheduled [16]. In this way we keep a similar execution time to the common case, which combined with the power reduction will produce an overall energy decrease.

Next, we observe that the area savings achieved by fragmentation can be traded-off systematically for power reduction. Our specific approach is to introduce choices for FUs with power/area trade-offs for different fragmentation and allocation styles. A resource allocation algorithm has been developed, which pursues a trade-off between area and power consumption for fragmented datapaths under total area constraints. Our experimental results show that it is possible to reduce power by 37% on average (49% in the best case).

The rest of the paper is organized as follows: section II discusses the related work, section III presents an example in order to motivate our techniques, section IV explains in more detail the allocation and module selection algorithms, section V describes the area and power models, and the FU library used in the module selection algorithm and finally sections VI and VII present our experimental results and final conclusions.

II. Related Work

Usually in datapaths multipliers are the biggest and most power consuming modules. Previous works [5-9] try to minimize power produced by multipliers. Authors propose to reduce switching activity in the partial product matrix with by-passing logic in one dimension [5,7,8], two dimensions [6], or by using 2's complement for some operands [9]. All these works reduce switching activity inside the multipliers for reducing power around 20-30% at the expense of an area increase that ranges between 10-25%, except for one case [6] where due to the bidimensional bypassing a significant 75% power reduction is achieved, but with 125% area overhead. These approaches that sacrifice a piece of area for diminishing power are becoming widespread and also adders present a low power version [10], reducing 55% power with 8% area penalty. Note that these LP-FUs present a similar, or even lower delay than the corresponding non-LP-FUs. Therefore achieving the same latency as in the common implementation will be enough for complying with the baseline execution time.

Another approach [11] proposes to bind operations that share some operands to the same FU and in consecutive csteps, thus reducing switching activity. Nevertheless, as this technique can only be applied to a specific subset of operations, it is quite restricted. In [12] authors present a methodology for reducing area, power or energy, but there is always one target function at a time. It is not clear how several parameters, e.g. area and power can be combined. Remaining parameters often suffer significant degradation while trying to reach optimality in the target metric. Area overhead when reducing power or energy exceeds 50% for some cases, and 25% on average. In [13] authors tradeoff area and power by different clock selections, but they relax the timing constraint *T*, producing circuits with 1.5T-3.5T.

In [14,15] authors propose to customize DSP or FPGA multipliers depending on the constants of the applications, obtaining area, power and latency reductions. This idea is more suitable for structures with abundant resources such as FPGAs and DSPs. This is not usually the case of ASICs, where datapaths must be highly optimized in order to comply with the designer constraints. Besides, this approach creates highly instruction-specific FUs. However, our objective is to use non-specific FUs, because the instruction-specific ones will diminish FU sharing and therefore it could introduce some additional modules, with the corresponding area penalty.

Therefore, the inclusion of Low Power (LP) modules in the design flow seems to be the most suitable technique for reducing power, while maintaining cycle time and latency constraints. However, the area overhead should be traded-off in a systematic way. In order to diminish area, existing techniques [16-18] apply fragmentation techniques while executing operations in different [17,18] or in their original cstep over a set of linked FUs [16]. Some of these algorithms [17,18] are especially oriented to diminish cycle time. Executing all fragments of operations in the same csteps as the corresponding non-fragmented operation in the original scheduling specification [16] is more suitable, if our target is to maintain both cycle time and circuit latency. Our approach aims to use fragmentation to save area and give it back to introduce LP-modules.

III. Motivational Example

In order to describe how to achieve a low power design in a fragmented datapath while respecting area constraints, consider the example given in figure 1a). A DFG scheduling is shown. Operands are labeled as X1 through X16, while the operand size is depicted on the edge that feeds into the operand.

In figure 1b) the number of FUs, estimated FU-area, and FU-power are shown. In the leftmost column the allocation method is shown. Four different allocation methods have been considered: heterogeneous allocation, heterogeneous allocation + LP (considering Low Power FUs, in this case we include one LP multiplier in the resource set), an allocation with fragmentation (fragmented allocation), and fragmented allocation + LP (including one LP multiplier in the resource set). Note that the last, i.e., the fifth allocation follows the same allocation as the fourth, but this time including two LP multipliers in the resource set. In the remaining columns the number of adders, multipliers, estimated FU-area and estimated FU-power are depicted. We will explain in detail how the area and power values are derived in our actual experiments in section V. However, in order to simplify the illustration in this example, we will assume that both area and power are computed in terms of the number of Full Adder (FA) cells used across all units.

In the heterogeneous allocation, it is clear that 2 adders and 2 multipliers are needed. Their sizes will be decided later during the binding stage, selecting the greatest size among the operations bound to the FU under question. In this example independently from the binding, a 24-bit and a 16-bit adder, and a 16x8 and a 8x8 multiplier are required. In the fragmented

allocation, the 24-bit adder is divided into a 16-bit adder and an 8-bit one. However, only two 8x8 multipliers are needed. This is due to the fact that the original 16x8 multiplier has been fragmented in two 8x8 multipliers and an additional 16-bit adder for composing the final result.

Figures 1 c) and d) show the scheduling and binding for heterogeneous and fragmented allocations. In figure 1 c) the operands that are smaller than the FU where they have been bound are extended with '0's. With signed numbers the sign would be extended. In the second cycle of figure 1 d) we observe how the original product **X5*X6** has been decomposed into two sub-products and one final addition that are executed in the same cstep where the original product was scheduled. Similarly, the original addition **X7+X8** in cycle 3, has been divided into two fragments of lengths 16 and 8 bits, respectively, but executed in the same cstep. Therefore, after applying fragmentation, operations will maintain their original scheduling. The difference is that they will be bound to a set of linked FUs, instead of only one FU.

Finally let us examine the estimated FU-area and FU-power shown in the two rightmost columns of figure 1 b). We have considered that a LP multiplier occupies 20% more area, while consuming 30% less power, similar to a design presented in literature [5]. In the heterogeneous+LP allocation a 16x8 LP multiplier is considered. In the case of fragmented+LP allocations, we consider two cases, in the fifth and sixth rows of the table in figure 1b), respectively: the use of only one 8x8 LP multiplier, and the use of two LP 8x8 multipliers. As it is observed, the utilization of LP-FUs in heterogeneous datapaths increases area with respect to the original implementation, i.e. it violates the area constraint. Fragmentation reduces the area, so on the one hand there will be less resources consuming power, especially in the case of big modules such as multipliers, and on the other hand this area reduction can be exploited in order to save even more power through use of LP-modules while still respecting the initial area constraint.

Therefore, assuming that the area constraint is 208 basic cells (the total area for the heterogenous design), using LP-FUs in combination with a heterogeneous allocation is not enough to satisfy the area constraint. However, after optimizing the design with a fragmented allocation, there will be some available FU-area slack for including LP FUs. 113.6 and 177.6 power and area units, respectively, are estimated for the Fragmented+LP and Fragmented+LP(2) allocations, in the last row of the table, which is around 45.5% less power and still 14.5% less area than the baseline heterogeneous datapath.

This example illustrates the opportunities available in the design process for fragmented datapaths. In this paper, we aim to develop a module selection algorithm to take advantage of this opportunity to design low power datapaths.

IV. PROPOSED TECHNIQUES

In order to reduce power we propose two strategies. On the one hand, useless area and switching FU parts will be removed by means of operator fragmentation in the allocation stage. On the other hand, low power FUs will substitute the baseline modules while respecting area constraints.

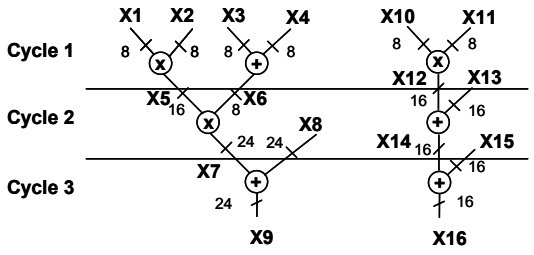
A. Fragmentation in the allocation stage

In the allocation stage operator fragmentation is performed in order to remove the useless switching parts. In our case the basic fragementation step has been carried out with an algorithm similar to existing techniques [16]. In our evaluations, we will then compare our low power module selection algorithm built on top of this, with an alternative flow which uses the fragmentation approach described in [18] in its initial step. If fragmented operations are still executed following the original scheduling it is possible to maintain a similar cycle time and not to increase latency. By contrast, dividing an operation in different fragments and rescheduling them may produce a latency increase, but on the other hand it decreases cycle time and benefits the FUs sharing, diminishing area too. Nevertheless there could appear some extra registers for keeping intermediate results and routing elements in order to complete the original operation, which would lessen this area reduction.

Multiplier allocation is performed first. This will produce a set of multipliers and a set of adders as well, due to the fact that after fragmenting one multiplier, some additions are needed in order to compose the final result. Next, adder fragmentation is realized. This will produce a list of FUs, which are able to comply with the scheduling performed before allocation, and binding performed after allocation.

This stage is really useful because it will allow substituting

a)



b)

Allocation	Adders	Multipliers	Area	Power
Heterogeneous	+24, +16	16x8, 8x8	208	208
Heterog + LP	+24, +16	LP 16x8, 8x8	233.6	169.6
Fragmentation	2 +16, +8	2 8x8	152	152
Frag + LP	2 +16, +8	LP 8x8, 8x8	164.8	132.8
Frag + LP (2)	2 +16, +8	2 LP 8x8	177.6	113.6

c)

	+24	+16	16x8	8x8
Cycle 1		X3 + X4	0X1 * 0X2	X10 * X11
Cycle 2		X12 + X13	X5 * X6	
Cycle 3	X7 + X8	X14 + X15		

d)

	+16	+16	+8	8x8	8x8
Cycle 1			X3 + X4	X1 * X2	X10 * X11
Cycle 2	X12 + X13	Comp(X5 * X6)		X5[7..0] * X6	X5[15..8] * X6
Cycle 3	X14 + X15	X7[23..8] + X8[23..8]	X7[7..0] + X8[7..0]		

Figure 1. a) DFG scheduling, b) FUs, and estimated area and power with different allocation techniques c), d) Scheduling, binding after applying heterogeneous and fragmented allocation, respectively

smaller FUs when introducing LP-FUs. Hence area penalty will be lower.

B. Using low power FUs

After the binding is performed, we need to decide on how many low power resources to use. In order to minimize power while complying with the area constraints, we have developed a variation of the greedy version of the knapsack problem [24]. In the knapsack problem there is a set of objects $\boldsymbol{O}=\{o_i\}$, each of them with a weight w_i and a value v_i, and a knapsack with a maximum allowed weight W_{max}. The objective is to maximize the sum $\sum(v_i) = V$, so that the sum $\sum(w_i) = W <= W_{max}$.

In our case power must be minimized while complying with the area constraints, so our values v_i will be the inverse of power. Hence, the problem can be reformulated as:

Given a set of FUs, $\boldsymbol{F}=\{f_i\}$, each of them with an area a_i and a power p_i, and given a maximum area A_{max}, the objective is to maximize $\sum(1/p_i)=P_{inverse}$, so that $\sum(a_i)=A <= A_{max}$.

Obviously, maximizing $\sum(1/p_i)$ is not the same as minimizing $\sum(p_i)$. However, using $1/p_i$ as the values v_i's is a good approach because the main target is to use FUs with low power consumption. As in the greedy knapsack problem the solution consists in ordering the objects according to their quality, q_i. For every object o_i, the quality is defined as $q_i=v_i/w_i$. After ordering the objects in decreasing order of quality, we choose those that satisfiy the weight constraint. In our case we order the FUs according to their quality $q_i=1/(p_i*a_i)$. Now, supposing that there are $\boldsymbol{N}$ FUs, the best $\boldsymbol{N-1}$ non-repeated f_i's are chosen while respecting the A_{max} constraint. The $\boldsymbol{N^{th}}$ f_i will be the one which minimizes the violation of the A_{max} constraint. In this way the algorithm can achieve good solutions in general, while keeping a low complexity. Finally, note that this algorithm can be extended to consider several FUs with different power-area (P,A) tradeoffs.

Figure 2 illustrates an example, which explains the stages of our algorithm. Let us suppose that after the allocation process two FUs are required, and suppose we have the (P,A) tradeoffs given by the topmost box. First, we perform a normalization of (P,A) values with respect to the baseline case, and second, quality values are calculated and ordered. Finally, supposing an area constraint A_{max}=84 we would choose the pairs (FU2,1.28) and (FU1,1), corresponding to the pairs (40,80) and (4.4). Note that normalization with respect the baseline values of area and power is needed in order to select the most power-area efficient modules. If no normalization were performed, due to numeric reasons the smallest modules would always be located first in the list, which could avoid bigger and more consuming modules to be substituted.

There are still two points to be explained: on the one hand, how to choose the A_{max} constraint, and on the other hand how to perform the area and power estimations. In order to choose properly a value for A_{max} we must take into account that after the fragmentation in the allocation stage, area will be reduced with respect to the original implementation, so A_{max} should be the estimated area as if a common heterogeneous allocation had been performed. The area and power models will be explained in more detail in the next section.

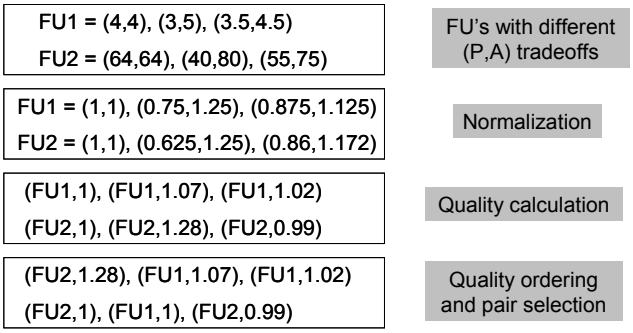


Figure 2. Functional Units selection algorithm stages

V. MODELLING FUNCTIONAL UNITS

Area and power models have been developed in order to give area and power information to the algorithm before synthesizing the datapaths. This will be described in subsections A and B, while in subsection C a brief overview of the considered modules will be given.

A. Area model

We make an assumption similar to prior work [5-9] based on the replication of Full-Adders (FAs) in the structure of a FU. Thus, the FUs area is estimated in terms of number of FAs. Hence, an *n*-bits *Ripple Carry Adder* [22] occupies *n* basic cells, while a *m*x*n* parallel multiplier, like the *Carry Save* or *Braun* multiplier [22], occupies *m*x(*n-1*) basic cells. For low power modules, we have used area data reported in literature. The respective area penalties in LP designs reported in those studies have been used to scale the area costs of the baseline designs that are measured in terms of FAs.

B. Power model

The proposed power model estimates the dynamic power of the FUs that will be deployed in the datapath. As LP-FUs [5-10] are mostly based on the reduction of switching activity, estimating dynamic power will be a good metric in order to select the best candidate with the FU selection algorithm. Hence, we have developed a program that simulates the datapath at RT level after receiving the information given by the allocation, scheduling, and binding stages.

Dynamic power is calculated taking into account the size and switching activity of every module under consideration. Size is computed as the number of basic cells, like in the area model, while switching activity is a fraction that depends on the Hamming Distance between two consecutive operations bound to the same resource, which is computed with our simulator. Thereby, for every module dynamic power is estimated as the product of the switching activity and the size. Finally, this number is multiplied by a constant, which may be dependant on the technology. Note that this is similar to the traditional dynamic power $P_{switching} = \alpha * V_{DD}^2 * f * C_L$ formula [1]. Switching activity is α, while size should be proportional to C_L. V_{DD} and f are defined by the target technology, such as the constant in our formula. The respective power penalties reported in the [5-10] studies have been used to scale the power costs with respect the baseline designs.

C. FU library

Our modules library is composed of the following ones:

- Adders. We have used RCAs [22] as the baseline adder, and a LP version taking the (P,A) tradeoff information from published data [10]. Thus the LP-RCA will consume 55% less power in exchange for an 8% area overhead.
- Multipliers. The Braun multiplier [22] has been our baseline multiplier. LP modules have been taken from published results [5-6]. [5-6] multipliers offer a 33.8% and 75% power reduction with an area penalty of 21% and 125%, respectively.

VI. EXPERIMENTS

In this section first we describe our experimental framework. Next, we discuss our experimental results.

A. Framework

We have used a Force Directed Scheduling based scheduling algorithm [23] and afterwards we have performed the allocation stage. As our baseline, a traditional allocation algorithm for heterogeneous datapaths [25] has been used. We also implemented the fragmented allocation algorithm explained in section IV.A. Finally a traditional resource constrained FU binding and a left-edge based register binding [1] is applied to both allocation solutions.

Next, the information obtained from scheduling, allocation and binding, is introduced into our switching simulator in order to estimate both area and power and apply our FU selection algorithm described in section IV. B. Afterwards we generate the HDL via a VHDL code-generator that receives the scheduling, allocation, binding and module selection information. Then, *Synopsys Design Compiler* is used for synthesizing designs with a 65 nm library, and finally *Synopsys Power Compiler* calculates power.

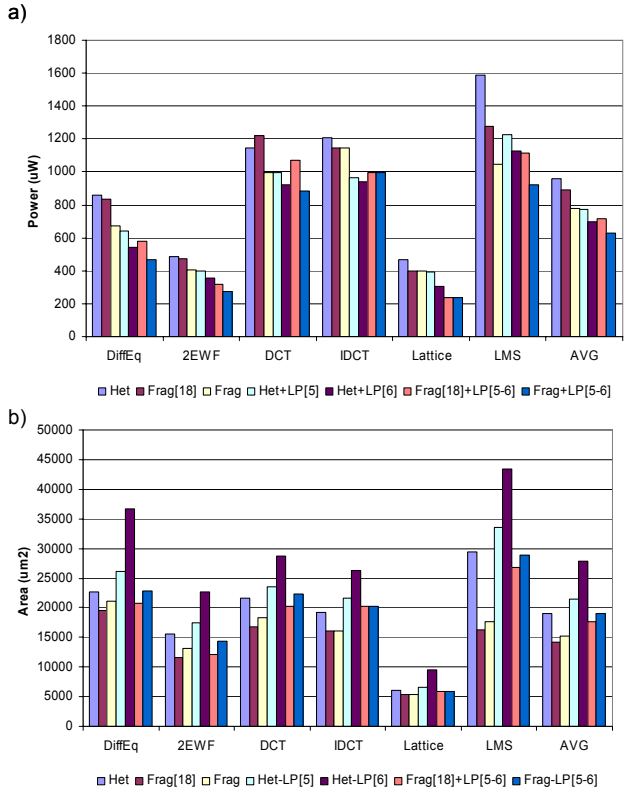


Figure 3. a) Power and b) Area results with several allocation styles and LP-FUs

B. Synthesis Results

In order to test the efficiency of our methods we have performed several experiments. Six benchmarks have been used for our evaluations.

A study of the power consumption and area overhead has been performed. Figures 3a) and 3b) depict the power and area of the six benchmarks and the average values for seven allocation and module selection cases, namely: heteroneneous allocation (*Het*), that will be our baseline case, fragmented allocation utilizing an algorithm like the one described in [18] (*Frag[18]*), our fragmented allocation (*Frag*), heterogeneous allocation using LP multipliers from [5] (*Het+LP[5]*), heterogeneous allocation using LP multipliers from [6] (*Het+LP[6]*), a [18] fragmentation algorithm using LP multipliers from [5-6] (*Frag[18]+LP[5-6]*) and our fragmented allocation using LP multipliers from [5-6] (*Frag+LP[5-6]*).

LP-modules are selected with our FU selection algorithm. In both *Frag[18]+LP[5-6]* and *Frag+LP[5-6]* the A_{max} constraint is the estimated area of the *Het* case. Utilizing this algorithm with a heterogeneous allocation, such as *Het+LP[5]* and *Het+LP[6]*, implies to choose an A_{max} value that violates the baseline area. Then we have chosen an A_{max} value of 10% and 40% with respect the baseline area estimation, respectively. In the case of *Het+LP[5]* as the area penalty is not too high, all the multipliers can be substituted by their LP version. On the other hand in *Het+LP[6]*, as [6] area is too high, only the most switching multiplier of the datapath has been substituted.

As it can be observed in figures 3a) and 3b), *Frag* achieves around 19% power reduction, which is nearly the same as the *Het+LP[5]* case. However, the LP-FUs introduce 12% area overhead with respect to the baseline case. This does not happen with *Frag*, which saves 20% in area with respect to *Het*. Thus, this area reduction can be given back in order to incorporate the LP-FUs. On the other hand, *Frag[18]* reduces area by 25.4%. Since resulting fragments are rescheduled in different csteps by *Frag[18]*, some FUs can be removed, which would not be possible in our proposed method. However, we note that this reduction in FU area does not reflect to the final

Table 1. a) Latency and Cycle Time and b) Execution time and Energy per iteration with heterogeneous, fragmented [18] and our fragmented allocation stages

a)

	Latency (cycles)			Cycle Time (ns)		
Benchmark	Het	[18]	Ours	Het	[18]	Ours
DiffEq	4	5	4	23.31	18.05	23.71
2EWF	14	16	14	17.86	13.99	18.14
DCT	10	12	10	17.55	13.74	17.83
IDCT	10	10	10	17.34	17.34	17.62
Lattice	9	9	9	11.89	11.89	11.89
LMS	9	12	9	23.58	18.19	24.14
AVG	9.33	10.67	9.33	18.59	15.58	18.89

b)

	Ex. time per iteration (ns)			Energy per iteration (pJ)		
Benchmark	Het	[18]	Ours	Het	[18]	Ours
DiffEq	93.24	90.25	94.84	80.32	52.10	44.48
2EWF	250.04	223.78	253.96	121.29	70.76	69.18
DCT	175.50	164.87	178.30	201.40	176.67	157.38
IDCT	173.40	173.40	176.20	209.48	172.35	175.14
Lattice	107.01	107.01	107.01	50.21	25.52	25.52
LMS	212.22	218.24	217.26	337.11	243.74	200.58
AVG	168.57	162.92	171.26	166.64	123.53	112.05

overall area because of the additional registers and routing elements required by *Frag[18]*. However, due to the increase in frequency, power is only reduced by 7%. *Het+LP[6]* reaches a 28% power reduction, but since the area penalty of the module LP[6] is high, the average area overhead increases by up to 47%. *Frag+LP[5-6]* saves power by up to 37% on average (49% best case), and with a 0.61% less area than the original case (8.62% area reduction in the best case). By contrast, *Frag[18]+LP[5-6]* only decreases power by 28.5%, but with 7% area reduction. Finally, note that *Frag[18]+LP[5-6]* and *Frag+LP[5-6]* saves 19.4% and 19% power in comparison with *Frag[18]* and *Frag*, respectively, at the expense of 24.1% and 25.1% increase in area.

Therefore a fragmented allocation stage works fine in order to reduce both area and power, but more power can be saved up by introducing LP-modules carefully with the FU selection algorithm. Besides the saved area that can be given back to include LP-FUs, the area penalty due to the LP-FUs themselves is diminished in the case of a fragmented allocation, because the size of the utilized FUs uses to be smaller than in a traditional allocation process.

C. *Latency and cycle time*

We have compared our latency and cycle time results with those obtained with [18] techniques, where fragmented operations are executed in different csteps, combined with LP-modules. Results are shown in table 1a). Columns 2 to 4 and 5 to 7 depict the latency of every benchmark for a heterogeneous case, a [18]-like implementation and ours. Latency is given in cycles and cycle time in nanoseconds. The alternative method [18] reduces cycle time by around 16.4%, while it increases latency by 14.3%. On the other hand, in our approach there is some delay produced by the routing elements used to connect the involved fragments. This is shown in column 7. However, the increase in cycle time does not exceed 2% on average.

The consequences of these results in combination with the power ones from figure 3 are summarized in table 1b), where execution time and energy per iteration are shown. Columns are distributed as in table 1a). Execution time is given in nanoseconds and energy in picojoules. The alternative technique *Frag[18]* reduces execution time, while in our case execution time is slightly increased. However, the differences are negligible. This will lead to an energy reduction in our proposed solution, because on the one hand execution time is nearly equal among the three options, but on the other hand we achieve a power reduction, as shown in figure 3. In conclusion, power reduction is accompanied by an energy reduction of 25.9% and 32.8% in the case of *Frag[18]* and our proposed fragmentation techniques, respectively.

VII. CONCLUSIONS

A power aware allocation and module selection algorithms are presented. Using fragmentation techniques in the allocation stage an area reduction is achieved. Afterwards this will be given back by substituting the less efficient FUs in terms of power-area in order to achieve an additional power reduction while complying with the initial area constraint. Results show that is possible to reduce power a 37% in average (49% best case) while achieving almost the same area than in the baseline case. Moreover, as execution time is close to the baseline case, this power reduction is accompanied by an energy reduction that demonstrates the efficiency of our algorithms.

REFERENCES

[1] P. Coussy, A. Morawiec, "High-Level Synthesis. From Algorithm to Circuit Design.", Springer, 2008.

[2] S.P. Mohanty, N. Ranganathan, E. Kougianos, P. Patra. "Low-Power High-Level Synthesis for Nanoscale CMOS", Springer, 2008.

[3] V. Kursun, E.G. Friedman, "Low swing dual threshold voltage domino logic", Proc. Of the 12th ACM GLSVLSI, 2002, pp. 47-52.

[4] A. Kshavarzi, S. Narendra, S. Borkar, C. Hawkins, K. Roy, V. De, "Technology scaling behavior of optimum reverse body bias for standby leakage power reduction in CMOS IC's", ISLPED, 1999, pp. 252-254.

[5] K-C. Kuo, C-W. Chou, "Low Power Multiplier with Bypassing and Tree Structure", IEEE APCCAS, 2006, pp. 602-605.

[6] C. Wang, G. Sung, "Low-Power Multiplier Design Using a Bypassing Technique", Journal of Signal Processing Systems, 2009, pp. 331-338.

[7] G. Economakos, K. Anagnostopoulos, "Bit Level Architectural Exploration Technique for the Design of Low Power Multipliers", IEEE International Symposium on Circuits and Systems, 2006, pp- 1483-1486.

[8] M.C. Wen, S. J. Wang, Y.N. Lin, "Low-Power parallel multiplier with column bypassing", Electronic Letters, vol 41, no 12, pp. 581-583, 2005.

[9] M. Nickary, M. Dehyadgari, A. Sobhani, A. Afzali-kusha, "Multiplier for Correlative Input Patterns", 17th ICM , 2005.

[10] V. Navarro-Botello, J.A. Montiel-Nelson, S. Nooshabadi, "HIgh performance low power CMOS dynamic logic for arithmetic circuits", Microelectronics Journal, 2007, Vol. 38, no. 4-5, pp. 482-488.

[11] E. Musoll, R. Cortadella, "High-level synnthesis techniques for reducing the activity of functional units", ISLPED, pp. 99-104, 1995.

[12] W. Wang, T. K. Tan, J. Luo, Y. Fei, L. Shang, K.S. Vallerio, L. Zhong, A. Raghunathan and N.K. Jha, "A Comprehensive HLS System for Control-Flow Intensive Behaviors", GLSVLSI, pp. 11-14, 2003.

[13] M. A. Ochoa-Montiel, B. M. Al-Hashimi, P. Kollig. "Exploiting Power-Area Tradeoffs in Behavioural Synthesis through Clock and Operations Throughput Selection". Proceedings of ASP-DAC, 2007.

[14] M. Mukherjee, R. Vermuri, "A Novel Synthesis Strategy Driven by Partial Evaluation Based Circuit Reduction for Application Specific DSP Circuits", Proc. Of the 21st ICCD, 2003, pp. 436-440.

[15] G. Caffarena, J.A. Lopez, C. Carreras and O. Nieto-Taladriz, "High-level synthesis of multiple word-length DSP algorithms using heterogeneous-resource FPGAs", FPL, 2006, pp. 675-678.

[16] A.A. Del Barrio, M.C. Molina, J.M. Mendias, E. Andres, R. Hermida, "Restricted Chaining and Fragmentation Techniques in Power Aware High-Level Synthesis", 11th Euromicro DSD, 2008, pp. 267-273.

[17] M. C. Molina, J. M. Mendías, R. Hermida, "Behavioural specification allocation to minimize bit level waste of functional units," Proc. Inst. Elect. Eng.—Comput. Digit. Tech., vol. 150, no. 5, pp. 321–329, 2003

[18] M.C. Molina, R. Ruiz-Sautua, J.M. Mendias, R. Hermida, "Bitwise Scheduling to Balance the Computational Cost of Behavioral Specifications". IEEE Trans. On CAD. Vol 25. no. 1, pp. 31-46, 2006.

[19] H. Yang, and L. Dung. "On multiple-voltage high-level synthesis using algorithmic transformations". Proceedings of ASP-DAC, 2005.

[20] S. P. Mohanty, N. Ranganathan. "Energy-efficient datapath scheduling using multiple voltages and dynamic clocking". ACM TODAES, 10(2):330-353, 2005.

[21] L. Benini, A. Bogliolo, and C. De Micheli. "A survey of design techniques for system-level dynamic power management". IEEE Trans. on VLSI systems, June 2000.

[22] I. Koren, "Computer Arithmetic Algorithms", A K Peters, 2nd ed, 2002.

[23] P.G. Paulin, J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs", IEEE Trans. on CAD. Vol. 8, no. 6, 1989, pp. 661-679.

[24] G. Brassard, P. Bratley, "Fundamentals of Algorithmics", Prentice Hall, 1996.

[25] S. Devadas, A.R. Newton, "Algorithms for hardware allocation in data path synthesis", IEEE Trans. On CAD, 1999, no. 8, pp. 768-781.