High-level synthesis techniques for reducing the activity of functional units

E. Musoll and J. Cortadella
Department of Computer Architecture
Universitat Politècnica de Catalunya
08071-Barcelona, Spain

Abstract

Decisions taken at the earliest steps of the design process may have a significant impact on the characteristics of the final implementation. This paper illustrates how power consumption issues can be tackled during high-level synthesis (high-level transformations, scheduling and binding). Several techniques pursuing low power are proposed and the potential benefits evaluated.

The common idea behind these techniques is to reduce the activity of the functional units (e.g. adders, multipliers) by minimizing the changes of their input operands. Preliminary evaluations obtained from switch-level simulations show that significant improvements can be achieved.

1 Introduction

Power consumption can be taken into account at different levels [5]: technological, topological, architectural and algorithmic level.

High-level synthesis (HLS) comprises techniques at the architectural and algorithmic level. Traditionally, HLS has been applied to obtain small and fast designs. But little has been done to include power consumption as one of the design parameters or constraints.

In this paper we present some HLS techniques for power reduction bearing in mind that design decisions taken at the architectural and algorithmic level can have a significant impact on the quality of the final implementation. No methods to implement the techniques are presented. In order to evaluate the efficiency of the techniques, power-consumption models derived from switch-level simulations of the basic functional units (e.g. adders and multipliers) will be used. The proposed techniques attempt to reduce the activity of the functional units by minimizing the changes of their input operands.

The paper is organized as follows: in Section 2 the previous work on high-level techniques for low power is briefly presented. Section 3 presents the power-consumption models of adders and multipliers along with an introduction of the proposed techniques. Sections 4-8 describe the techniques for power reduction. Section 9 concludes the paper.

2 Previous work

Research in low-power circuits has been devoted to the power consumption estimation of relatively small circuits [18, 8, 12, 22]. The design of arithmetic circuits aiming at minimizing power consumption has been rarely addressed [3, 27, 10].

Most of the efforts in HLS for low power propose models and estimations of power consumption at algorithmic and architectural level. In [15], a model that accounts for the random behavior of the LSB bits and the correlated behavior of the MSB bits is presented. In [1] the impact of the cache architecture in power consumption is studied. In [19] a technique to evaluate a lower bound of the throughput and cost during algorithm selection is introduced. In [2] different processor models that account for the energy for the major modes of computation are described.

Few authors have addressed the set of transformations at algorithmic and architectural level to obtain lower-power designs. In [6] the power consumption of additions and constant multiplications as a function of the operand activity is studied. From this study, a data flow graph transformation is described for a typical operation in signal processing applications. In [26] some memory transformations for low power systems are hinted. The aim of these transformations is to reduce both the activity of the address lines and the number of off-chip references. In [4] the traditional transformations for faster and smaller circuits are applied in order to evaluate the power consumption savings. Whenever the resulting circuit is faster than the required throughput, power-supply reduction can be applied to take advantage of its quadratic impact on consumption.

3 Power consumption models and power reduction techniques

This section describes the power-consumption models used to evaluate the techniques presented in the paper. A summary of all the techniques is also included.

3.1 Power consumption models

Power consumption has been considered only in the arithmetic components of the data-path and simple power-consumption models have been derived for each basic functional unit (adder, multiplier). Power consumption in the data-path accounts for a large fraction of the overall system power budget. The tool used in the estimations is *sls* [24], a switch-level simulator. The designs of the functional units are based on library cells.

In these models the number of operands that remain unchanged with respect to the previous operation is taken into account. Figure 1 illustrates this concept for an 8×8 radix-4 Booth multiplier [13].

In Figure 1(a), plot (3) represents the energy of the multiplier in $nJ/operation$ when one operand remains unchanged (x axis) with respect to the previous operation and the other operand varies randomly[1]. Line (2) is the average of plot (3) and line (1) is the average energy when both operands vary randomly with respect to the previous operation. Comparing lines (1) and (2), the average power consumption of the multiplier is approx. 35% less when one operand remains unchanged.

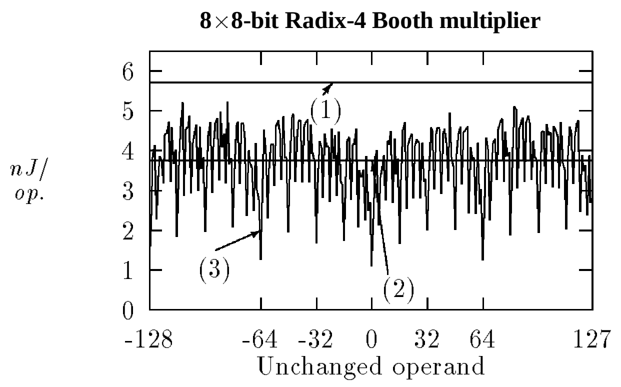


Figure 1: Plot (3) represents the energy of the multiplier when one operand remains unchanged (x axis) with respect to the previous operation and and the other operand varies randomly. Line (2) is the average of plot (3) and line (1) is the average energy when both operands vary randomly.

The techniques proposed in this paper will use the notation in Table 1. Factor β denotes the power consumption relation among the adder and multiplier whereas factors α_{add} (α_{mul}) denote the

[1]Although data is correlated for some of the HLS applications, we have found the random distribution to be a good firt approximation.

Parameter	Description	8-bit	12-bit	16-bit
P_{add2}	Avg. consumption of an adder when both operands change	0.35 $nJ/op.$	0.53 $nJ/op.$	0.90 $nJ/op.$
P_{add1}	Avg. consumption of an adder when only one operand changes	0.26 $nJ/op.$	0.4 $nJ/op.$	0.70 $nJ/op.$
P_{mul2}	Avg. consumption of a multiplier when both operands change	5.7 $nJ/op.$	13.68 $nJ/op.$	28.9 $nJ/op.$
P_{mul1}	Avg. consumption of a multiplier when only one operand changes	3.7 $nJ/op.$	8.88 $nJ/op.$	19.9 $nJ/op.$
α_{add}	P_{add1} / P_{add2}	0.74	0.75	0.77
α_{mul}	P_{mul1} / P_{mul2}	0.65	0.65	0.68
β	P_{add2} / P_{mul2}	0.06	0.04	0.03

Table 1: Notation used in the presented techniques. The values have been obtained for 8, 12 and 16-bit-wide functional units.

ratio of power in an adder (multiplier) between operations with one and two operand changes with respect to the previous operation.

We have found that good estimations of the factors α_{add}, α_{mul} and β are 0.75, 0.65 and 0.04 respectively for 12-bit-wide functional units. In DSP applications, a bit-width of 12 is considered accurated enough. For example, the value 0.65 for factor α_{mul} indicates that the average power consumption of a multiplication when one of its operands remains unchanged with respect to the previous operation is 35% less than when both operands change.

Factors α_{add} and α_{mul} hardly change with the bit-width of the operands. Although the values of these factors are realistic enough, they must be derived for each cell-library if more accurate estimations are pursued.

Although the models presented are simplistic, they provide an easy way to estimate the power consumption in high-level synthesis. The authors are currently working in a more precise model based on not only the number of operand changes, but on the variability of the bit-pattern of the operands. With this model, the correlation presented in the data is taken into account.

3.2 Power-reduction techniques

The techniques proposed in this paper are summarized as follows: **loop interchange**: takes advantage of data locality to reduce the activity of the inputs of the functional units; **operand reordering**: seeks an appropriate operand order for commutative operations to reduce the switching activity; **operand sharing**: attempts to schedule and bind operations to functional units in such a way that the activity of the input operands is reduced; **idle units**: tries to minimize the useless power consumption of the idle units and **operand correlation**: uses the information of the correlation among the variables and constants of the algorithm in the scheduling and register-binding steps.

4 Loop Interchange

The loop-interchange technique has been traditionally implemented in compilers to obtain dependency graphs with a higher degree of parallelism or to increase data locality and, thus, reduce memory traffic [26].

We apply loop interchange with the goal of minimizing the number of operand changes on the functional unit inputs. This technique will be applied to the *motion estimation* algorithm for image compression [17] (Figure2(a)) to illustrate its efficiency.

4.1 Application of loop interchange

In the algorithm of Figure 2(a) we observe three operations in the inner loop: absolute value, addition and subtraction. For simplicity, we will consider a subtraction to be the same as an addition in terms of power consumption.

The absolute value in 2's complement arithmetic has two steps: (a) to check whether the value is negative and (b) complement the number and add 1 in this case. The first step represents negligible contribution to the total power consumption: just check if the MSB bit is one. In average, the second step will be executed half of the times.

In the algorithm of Figure 2(a) we observe also that: (1) both operands of the accumulation usually change with respect to the previous iteration of the algorithm and (2) both operands of the subtraction inside the absolute value operator also change because both are fetched from memory in the inner loop (where the absolute value operation is executed).

If we use instead the algorithm of Figure 2(b), we find out that: (1) the total number of operations remains the same (approx. $P \times L \times M \times N$ additions, $P \times L \times M \times N$ subtractions and $(P \times L \times M \times N)/2$ increments), (2) both operands of the accumulation also change at each iteration and (3) now one operand of the subtraction inside the absolute value operator remains the same during $M \times N$ iterations.

With the notation in Table 1, the power consumption estimation of algorithm (a) is roughly

$$P_a = PLMN\ (2\ P_{add2} + \frac{P_{abs}}{2})$$

and the power consumption estimation of algorithm (b) is

$$P_b = PLMN\ (P_{add2} + P_{add1} + \frac{P_{abs}}{2})$$

We have estimated by simulation the average power consumption of the increment operation executed on an adder as $P_{abs} \approx 0.45\ P_{add2}$. Thus, the estimated reduction factor on power consumption is

$$R\ (\alpha_{add}) = \frac{1 - \alpha_{add}}{2.225}$$

With the value for α_{add} in Table 1 for 12-bit-wide functional units, we obtain a reduction of the power consumption of 11%.

The power consumption has only been estimated for the functional units of the data-path. The increase in the control logic can reduce the savings achieved.

With algorithm (b) the off-chip reference order has changed, although the total number remains the same. Of course, a wise use of the local registers is expected in order to minimize the off-chip references. This is important particularly in the motion estimation algorithm, where the data working-set is considerably large. In order to minimize off-chip references, the most frequently referenced data can be stored in an internal cache. This implies that the algorithm must adapt its structure to the size of this internal cache to properly exploit data locality.

5 Operand Reordering

The goal of this technique is to find an appropriate input operand order for commutative operations in such a way that switching activity is reduced. In order to estimate its efficiency, this technique will be applied to the the multiply-accumulate (MAC) unit.

5.1 The MAC structure

Digital filters are basic components in DSP systems. A typical substructure of a filter is the MAC structure, which performs the operation $\sum_{i=0}^{p-1} x_i y_i$, where p multiplications and $p-1$ additions are executed.

One possible data-flow graph (DFG) of the operation is shown in Figure 3(a). Three adders and four multipliers are used to implement the MAC unit. There are other ways to reorganize the additions, but the balanced structure of Figure 3(a) implies less power consumption [4].

Figure 3(b) shows a 4th-order LMS adaptive filter [23]. In the LMS filter, and in some other digital filters (g.e. FIR and IIR filters), the MAC structure plays an important role and, therefore, minimizing its power consumption will decrease the total power consumption of the filter.

5.2 Application of operand reordering

For power consumption purposes, the MAC unit is classified into three cases: (a) both the x and y values change from one iteration to the next one (the general case); (b) either x or y values are constant and (c) either the x or y values of iteration i are the same as those of iteration $i-1$ but shifted one position. The IIR and FIR filters follow cases (b) and (c). In the LMS filter, the MAC unit follows case (c).

In order to propose a better operand reordering for cases (a) and (b), the *activity* of the operands is taken into account whereas

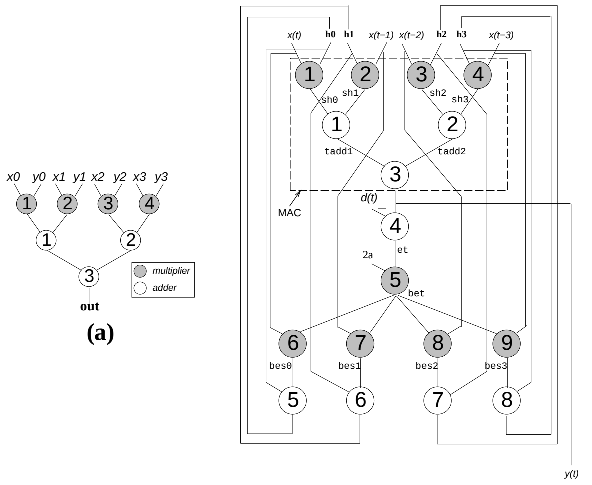
```
for $g = 0$ to $\lceil \frac{P}{m} \rceil - 1$
  for $h = 0$ to $\lceil \frac{L}{n} \rceil - 1$
    $\triangle_{optimal}(g, h) = \infty$
    for $i = -\lfloor \frac{M}{2} \rfloor$ to $\lfloor \frac{M-1}{2} \rfloor$
      for $j = -\lfloor \frac{N}{2} \rfloor$ to $\lfloor \frac{N-1}{2} \rfloor$
        $\triangle_{part}(i, j) = 0$
        for $k = 0$ to $m - 1$
          for $l = 0$ to $n - 1$
            $CV = CF(m \cdot g + k, n \cdot h + l)$
            $RV = RF(m \cdot g + i + k, n \cdot h + j + l)$
            $\triangle_{part}(i, j) = \triangle_{part}(i, j) + |CV - RV|$
        if $\triangle_{part}(i, j) < \triangle_{optimal}(g, h)$ then
          $\triangle_{optimal}(g, h) = \triangle_{part}(i, j)$
          $MV(g, h) = [i, j]^T$
```

(a)

```
for $i = -\lfloor \frac{M}{2} \rfloor$ to $\lfloor \frac{M-1}{2} \rfloor$
  for $j = -\lfloor \frac{N}{2} \rfloor$ to $\lfloor \frac{N-1}{2} \rfloor$
    $\triangle_{part}(i, j) = 0$
for $g = 0$ to $\lceil \frac{P}{m} \rceil - 1$
  for $h = 0$ to $\lceil \frac{L}{n} \rceil - 1$
    for $k = 0$ to $m - 1$
      for $l = 0$ to $n - 1$
        $CV = CF(m \cdot g + k, n \cdot h + l)$
        for $i = -\lfloor \frac{M}{2} \rfloor$ to $\lfloor \frac{M-1}{2} \rfloor$
          for $j = -\lfloor \frac{N}{2} \rfloor$ to $\lfloor \frac{N-1}{2} \rfloor$
            $RV = RF(m \cdot g + i + k, n \cdot h + j + l)$
            $\triangle_{part}(i, j) = \triangle_{part}(i, j) + |CV - RV|$
    $\triangle_{optimal}(g, h) = \infty$
    for $i = -\lfloor \frac{M}{2} \rfloor$ to $\lfloor \frac{M-1}{2} \rfloor$
      for $j = -\lfloor \frac{N}{2} \rfloor$ to $\lfloor \frac{N-1}{2} \rfloor$
        if $\triangle_{part}(i, j) < \triangle_{optimal}(g, h)$ then
          $\triangle_{optimal}(g, h) = \triangle_{part}(i, j)$
          $MV(g, h) = [i, j]^T$
        $\triangle_{part}(i, j) = 0$
```

(b)

Figure 2: **(a)** Motion estimation algorithm and **(b)** motion estimation algorithm with two loop interchanges. Notation: P and L, bit-length and bit-width of the current image frame; M and N, maximum horizontal and vertical vector coordinate; m and n, bit-length and bit-width of the current block; CV and RV, current and reference image frame value; CF and RF, current and reference frame; $MV(g, h)$, motion vector of block (g, h).



(a) **(b)**

Figure 3: **(a)** MAC structure for $p = 4$ and **(b)** DFG of the 4th-order LMS adaptive filter.

for case (c), the *repetition* of the operands will determine the new operand reordering.

Operand activity relates to the variability of the bit-pattern of one operand from one iteration to the next (power consumption is somehow related to the Hamming distance of consecutive bit-patterns). *Operand repetition* relates to the coarse-grained variability of the operand, i.e. the operand may or may not change between two consecutive iterations.

Case (b) has been addressed in [6], and the conclusion is that the minimum average activity over all nodes of the balanced MAC unit is obtained when the constant operands (e.g. the y values) satisfy $y_0 \le y_1 \le \cdots \le y_n$ or $y_0 \ge y_1 \ge \cdots \ge y_n$.

5.2.1 Input reordering for case (c)

As previously explained, operand repetition will determine the new reordering. In the MAC structure of the LMS filter of Figure 3(b) we observe that all multiplications receive different operands at each iteration: the x values are shifted one position to the left and the first position is the new operand value; the h values are recalculated at each iteration and, therefore, are different. This fact is clearly shown in Table 2 (reordering A).

Table 2 (reordering B) shows a different operand reordering that takes advantage of the shift-wise behavior of the x values. With this new reordering, each multiplier will have one fixed operand (the x value) during four consecutive iterations.

iter.	**reordering A**			
	M_0	M_1	M_2	M_3
i	(x_t, h_0)	(x_{t-1}, h_1)	(x_{t-2}, h_2)	(x_{t-3}, h_3)
$i+1$	(x_{t+1}, h_0)	(x_t, h_1)	(x_{t-1}, h_2)	(x_{t-2}, h_3)
$i+2$	(x_{t+2}, h_0)	(x_{t+1}, h_1)	(x_t, h_2)	(x_{t-1}, h_3)
$i+3$	(x_{t+3}, h_0)	(x_{t+2}, h_1)	(x_{t+1}, h_2)	(x_t, h_3)

iter.	**reordering B**			
	M_0	M_1	M_2	M_3
i	(x_t, h_0)	(x_{t-1}, h_1)	(x_{t-2}, h_2)	(x_{t-3}, h_3)
$i+1$	(x_t, h_1)	(x_{t-1}, h_2)	(x_{t-2}, h_3)	(x_{t+1}, h_0)
$i+2$	(x_t, h_2)	(x_{t-1}, h_3)	(x_{t+2}, h_0)	(x_{t+1}, h_1)
$i+3$	(x_t, h_3)	(x_{t+3}, h_0)	(x_{t+2}, h_1)	(x_{t+1}, h_2)

Table 2: Two different input reordering for the 4-input MAC unit. M_i represent the multiplications of the MAC unit.

Using the notation in Table 1 the estimated power consumption of the MAC operation with reordering A after p iterations is

$$P_A(\beta) = p\,(p\,P_{mul2} + (p-1)\,P_{add2}) = p\,P_{mul2}\,(p + \beta\,(p-1))$$

and the estimated power consumption with reordering B after p iterations is

$$P_B(\alpha_{mul}, \beta) = p\,(p\,P_{mul1} + (p-1)\,P_{add2}) =$$

$$= p\,P_{mul2}(p\,\alpha_{mul} + \beta\,(p-1))$$

Thus, the estimated power-consumption reduction factor from reordering A to B is

$$R(\alpha_{mul}, \beta) = \frac{p\,(1-\alpha_{mul})}{p\,(1+\beta) - \beta} \approx \frac{1-\alpha_{mul}}{1+\beta}$$

With the values in Table 1 for 12-bit-wide functional units, a 34% of power-consumption reduction is achieved.

6 Operand Sharing

The operand-sharing technique attempts to schedule and bind operations to functional units in such a way that the activity of the input operands is reduced. Operations sharing the same operand are scheduled in control steps as near as possible. Thus, the potential for a functional unit to reuse the same operand value (and, therefore, to decrease its input activity) is higher. This technique is efficient when it is applied to a DFG with variables used by more than one operation. The AR filter [14] will be used to illustrate this technique. The DFG of the AR filter is presented in Figure 4(a).

Figure 4(b) shows a possible schedule of the AR filter with two adders (one cycle) and one pipelined multiplier (two cycles). We observe there are some operations whose result is the input for more

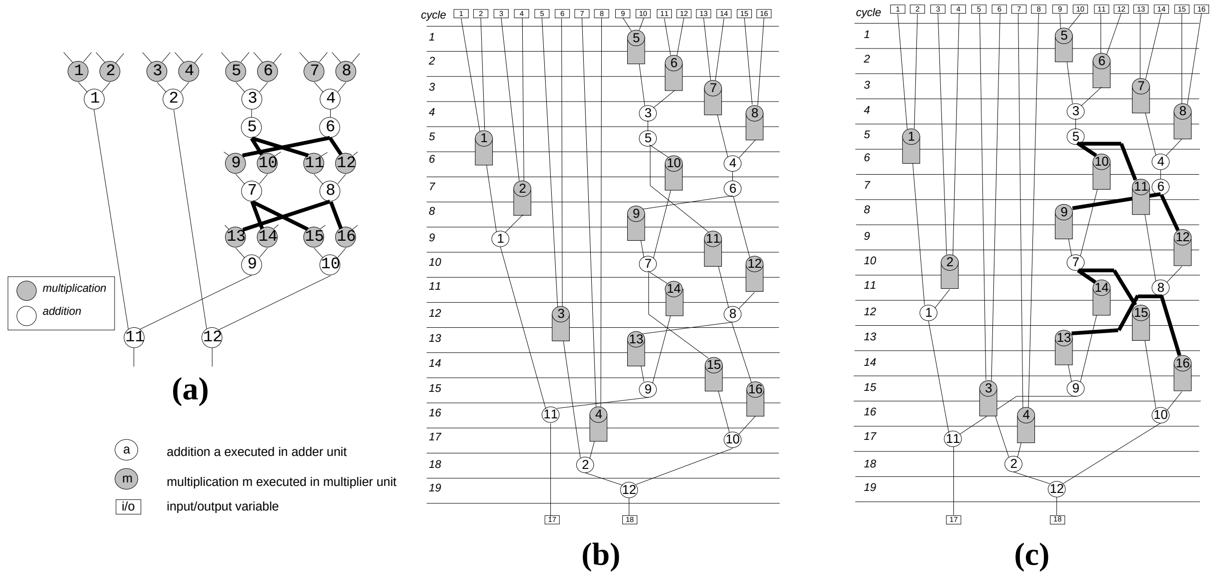

multiplication
addition
addition a executed in adder unit
multiplication m executed in multiplier unit
input/output variable
cycle
(a)
(b)
(c)

Figure 4: **(a)** DFG of the AR filter; **(b)** one possible schedule and binding of **(a)** with one adder (one cycle) and one pipelined multiplier (two cycles) and **(c)** improved schedule with 4 achieved OPRs.

than one operation (thick lines in Figure 4(a)). For example, the result of addition 5 is input for multiplications 10 and 11. Assume we schedule multiplications 10 and 11 to the same unit U. Assume also that between the execution of multiplication 10 and 11 there is no other use of unit U. Then, one of the operands of unit U will not change from multiplication 10 to multiplication 11. Henceforth, we will call operand reutilization (OPR) the fact that an operand is reused by two operations consecutively executed in the same functional unit. In Figure 4(a), 4 multiplication OPRs can be potentially obtained.

An alternative schedule and unit binding is presented in Figure 4(c) with 4 achieved OPRs. In the schedule and unit binding of Figure 4(b) no OPRs can be obtained.

Thus, the estimated power consumption of one iteration in schedule (b) is

$$P_b\,(\beta) = 12\ P_{add2} + 16\ P_{mul2} = P_{mul2}\,(16 + 12\,\beta)$$

and the estimated power consumption of one iteration in schedule (c) is

$$P_c\,(\alpha_{mul}, \beta) = 12\ P_{add2} + 12\ P_{mul2} + 4\ P_{mul1} =$$
$$= P_{mul2}\,(12 + 4\,\alpha_{mul} + 12\,\beta)$$

The estimated power-consumption reduction is

$$R(\alpha_{mul}, \beta) = \frac{1 - \alpha_{mul}}{4 + 3\,\beta}$$

With the values in Table 1 for 12-bit-wide functional units, a 8.5% reduction is achieved.

6.1 Application of loop unrolling for operand sharing

The operand-sharing technique is applied when some operations share the same operand in the same iteration of the algorithm. But it can also be applied even if operands feed more than one operation in different iterations. We just need to unroll the loop.

The low-pass image filter [16] will be used to illustrate this technique.

A DFG for the low-pass image filter is shown in Figure 5(b) [2]. We see that no OPR is possible. But if we unroll the inner loop twice (the loop body contains now three iterations), the DFG of Figure 5(c) is obtained, where some OPRs are possible.

With one adder, the schedule of the DFG in Figure 5(c) can be obtained in 24 cycles and the one in 5(b) in 8. Therefore, the total latency of the algorithm is the same in both schedules. All 9 OPRs are achieved.

The estimated power-consumption reduction is now

$$R\,(\alpha_{add}) = \frac{3}{8}(1 - \alpha_{add})$$

With the value of α_{add} in Table 1 for a 12-bit-width adder, the reduction obtained is 9.4%.

6.2 Application of the technique to other benchmarks

Table 3 shows the results obtained when applying the operand-sharing technique to other high-level synthesis benchmarks.

Benchmark	+/$*$	FUs	Red.
5th-order Wave filler [9]	26/8	1 $\otimes$ (2) / 2 $\oplus$ (1)	12%
4th-order Daubechies filter [20]	12/12	1 $\oplus$ (2) / 1 $\otimes$ (1)	21%
SHARF [23]	11/12	1 $\oplus$ pipel. (2) / 2 $\otimes$ (1)	10%
1-D 8-input Lee DCT [21]	29/13	2 $\oplus$ (2) / 2 $\otimes$ (1)	6%
1-D 8-input Chen DCT [21]	26/16	2 $\oplus$ (2) / 2 $\otimes$ (1)	19%
4 $\times$ 4 *matrix multiplier*	4/8	2 $\oplus$ (2) / 1 $\otimes$ (1)	26%

Table 3: Results obtained by applying the input-sharing technique over a wide range of benchmarks. The number and type of operations, the number and type of functional units (FUs) used and the power consumption reduction is shown. The numbers in parenthesis are the latency in cycles of the functional units.

In all benchmarks except for the Wave filter, the results have been obtained by comparing the power consumption estimation of the schedule with fewest OPRs and the schedule with the largest number of OPRs, having both schedules the lowest possible latency.

In the Wave filter we have detected a tradeoff between the speed and the consumption of the final design: it is possible to obtain a design with more latency but also with more number of achieved OPRs.

7 Idle units

Not all resources of a data-path are always used during all cycles. Some remain idle when no operation is available for them. The technique presented here tries to minimize the useless power consumption of the idle functional units. It is specially efficient

[2]For clarity, the division of the sum by nine is omitted and the input operands are assumed to be in registers.

```
for i = 0 to M
  for j = 0 to N
    out =
     ( A[i-1][j-1]+ / * a0 * /
       A[i-1][j]+   / * a1 * /
       A[i-1][j+1]+ / * a2 * /
       A[i][j-1]+   / * b0 * /
       A[i][j]+     / * b1 * /
       A[i][j+1]+   / * b2 * /
       A[i+1][j-1]+ / * c0 * /
       A[i+1][j]+   / * c1 * /
    A[i+1][j+1])/9  / * c2 * /
```
(a)

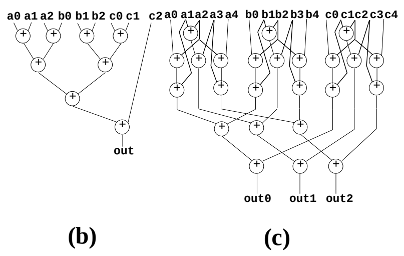

a0 a1 a2 b0 b1 b2 c0 c1 c2
out
a0 a1a2a3a4 b0 b1b2b3b4 c0c1c2c3c4
out0 out1 out2
(b)
(c)

Figure 5: **(a)** Low-pass image filter algorithm; **(b)** DFG of the inner loop of **(a)** and **(c)** DFG after loop unrolling.

for *sparse* schedules. A schedule is said to be *sparse* if the unit utilization is relatively low.

Some approaches to minimizing the useless power consumption of the idle units are: (a) with a proper register binding that minimizes the activity of the functional units (this technique is addressed in Section 8); (b) by wisely defining the control signals of the multiplexors during the idle cycles in such a way that the changes at the inputs of the functional units are minimized (this may result in defining some of the *don't care* values of the control signals) and (c) latching the operands of those units that will be often idle.

In this section, approach (c) is evaluated. It consists of the insertion of latches at the inputs of the functional units to store the operands only when the unit requires them. Thus, in those cycles in which the unit is idle no consumption in produced. The control unit has to be redesigned accordingly, in such a way that input latches become transparent during those cycles in which the corresponding functional unit must execute an operation.

This technique has been evaluated with the 5th-order Wave filter. With an schedule with two adders (one cycle) and one multiplier (two cycles) a final latency of 21 cycles has been obtained. During one iteration of the algorithm, the adders become idle during 16 cycles and the multiplier becomes idle during 5 cycles.

With the notation in Table 1 the power consumption generated by the idle units (*useless* consumption) is

$$P_{useless}\,(\alpha_{add}, \alpha_{mul}, \beta) = 16\; P_{add1} + 5\; P_{mul1} =$$

$$= 16\; \alpha_{add}\,\beta + 5\;\alpha_{mul}$$

and the power consumption due to the useful calculations (*useful* consumption) is

$$P_{useful}\,(\alpha_{add}, \alpha_{mul}, \beta) = 20\; P_{add2} + 6\; P_{add1} + P_{mul1} + 7\; P_{mul2} =$$

$$= 20\;\beta + 6\;\alpha_{add}\,\beta + \alpha_{mul} + 7$$

The estimated reduction in power consumption is

$$R\,(\alpha_{add}, \alpha_{mul}, \beta) = \frac{P_{useless}}{P_{useless} + P_{useful}} =$$

$$= \frac{16\;\alpha_{add}\,\beta + 5\;\alpha_{mul}}{22\;\alpha_{add}\,\beta + 14\;\alpha_{mul} + 20\;\beta + 7}$$

With the values in Table 1 for 12-bit-wide functional units, a 21% reduction is achieved. For simplicity in the evaluation (and to avoid synthesizing every control unit), we have assumed that, in average, only one of the operands changes in each idle unit at each cycle. This assumption may be optimistic or pessimistic depending on the final implementation.

Efficient latches (both in area and power) are integrated using Clocked CMOS gates (C^{2}MOS [25]) in the operand-selection multiplexers.

8 Operand Correlation

In the techniques previously presented, the main idea was to maximize the operand locality or, in other words, the *operand repetition* in the functional units. The operand-correlation technique takes into account the *operand activity* [3]. This technique uses the information of the correlation among the variables and constants of the algorithm in the scheduling and register-binding steps.

We will show how the activity of the input operands affect the power consumption of the design. Two examples will be presented to illustrate this technique: a low-power schedule for the finite impulse response filter (FIR filter) [23] and a low-power register binding for the Differential Equation Solver [11].

8.1 Input operand activity and its effect in power consumption

There are algorithms that present correlation among their variables and constants. A high correlation between two variables does not imply a low activity between them; for example, in the expression $x_t = 2\; y_{t-1}$ both variables x and y are highly correlated but if y always takes the value 010101 or 101010, then the Average Hamming Distance (AHD) between x and y is maximum (6).

Thus, a profiling of the algorithm to be synthesized is needed in order to determine the activity (measured with the AHD) among its variables and constants. As an example, let us consider the least-mean square adaptive filter (LMS filter) [23] of Figure 3(b).

Two experiments have been performed: in experiment A one of the input signals of the LMS filter is random; in experiment B the input is a waveform calculated as the sum of two sines. In both experiments, the second input signal has a triangular shape and the operation frequency is 0.5 kHz. The AHD among the variables assuming 12-bit operands have been obtained. When two variables have no correlation at all, their AHD is 6.

Experiment A implies that the variables $x(t)$ to $x(t-3)$ have an AHD of 6 among them, whereas in B the AHD reduces to 3.5 because of the smoother transition between one input data and the next one.

This difference in the AHD affects the power consumption of the filter. After simulations with *sls* [7] we have observed that experiment B is 22.6% less power consuming than experiment A.

The difference in power consumption obtained is only produced by the input data pattern. This difference increases with the sampling frequency. With a higher sampling frequency, the input data in experiment B is smoother than with a lower one. A higher sampling frequency implies a lower AHD in the input data [4]. The design is the same in both experiments and it has been scheduled with one adder (one cycle) and two multipliers (two cycles).

A similar experiment has been performed with the 4th-order FIR filter. A 7% power-consumption reduction has been observed.

8.2 Example 1: scheduling of the FIR filter

A FIR filter follows the equation $\sum_{i=0}^{p-1} x_i c_i$ where c_i are constants.

When speed is not a major issue, a significant reduction in hardware complexity is achieved by performing multiplications over several clock cycles as a series of shift-add operations. When speed is important, the multiplications must be executed by multipliers. We will focus on this case and will show how a different multiplication execution order can influence over the final power consumption.

As an example, assume $p = 4$, the values -1870, 1867, -740 and -1804 for the constants c_0 to c_3 and a bit-width of 12. Assume also that the input data is a waveform calculated as a sum of two sines. If this 4-order FIR filter is scheduled with one multiplier and one adder, different minimum-latency schedules are possible with different multiplication execution order. In one of those schedules, the multiplier observes the following changes in one of its operands (numbers on the arrows indicate the AHD between constants): $c_0 \xrightarrow{10} c_1 \xrightarrow{7} c_2 \xrightarrow{6} c_3 \xrightarrow{3} c_0 \xrightarrow{10} \cdots$ whereas in another schedule, it may observe the following changes: $c_0 \xrightarrow{10} c_1 \xrightarrow{11} c_3 \xrightarrow{6} c_2 \xrightarrow{7} c_0 \xrightarrow{10} \cdots$.

[3]See Section 5 for the definition of *operand repetition* and *operand activity*.

[4]The power consumption is calculated as the energy per iteration of the algorithm. Indeed, if we double the operation frequency, the overall power consumption is also doubled, but not the energy per iteration.

By means of switch-level simulations, the calculated power consumption for the functional units associated to the first schedule is 6.3% less than the one associated to the second. This reduction has been achieved only with the change of the schedule of two operations.

8.3 Example 2: register binding for the Differential Equation Solver

The experiments done in Section 8.1 for the LMS filter of Figure 3(b) showed that the AHD among the variables h_i is lower than among the other variables. The same occurs for $x(t-i)$, sh_i, bes_i and $tadd_i$.

This information can be used in register-binding algorithms to obtain a register set where activity of individual registers is minimized. As a side effect, those idle units that observe the changes in the registers will also reduce its consumption. Furthermore, the operand-correlation information along with the commutative property of some operations can be used also to decrease the power consumption in the non-idle functional units by swapping their operands.

The Differential Equation Solver has been scheduled with one adder (one cycle) and two multipliers (two cycles). The AHD between all pairs of variables has been obtained by means of simulations of the algorithm with different input data. The final AHD used has been obtained as the average of all simulations.

Two different register bindings (A and B) have been obtained. Both bindings use 5 registers. The reduction of the register activity of binding B produces an average power savings of 7.5% in the functional units with respect to A. This is obtained by the reduction of the operand activity at the inputs of the functional units during the idle cycles.

Intuitively, power consumption can be further reduced by increasing the number of registers (i.e., there exists a power-area tradeoff). The worst case, in terms of area, is to allocate one register for each variable. In this case, the idle units will have almost no operand changes on their inputs. But increasing the number of registers also increases the number of control signals, implying a more complicated control logic and interconnection, which may then offset the power savings achieved in the functional units.

9 Conclusions

The use of high-level synthesis techniques for low power can have a significant impact on the resulting implementations. In this paper, several strategies to tackle the problem of power consumption at high level have been presented. The potential benefits have been evaluated in different examples for DSP. All techniques focus on the minimization of the activity of the functional units by properly selecting the operands used at each cycle.

The promising results obtained from the preliminary estimations should endorse further research on this area. Forthcoming efforts must be devoted to automate these techniques and incorporate them into synthesis systems. The authors of the paper are currently pursuing this goal.

Acknowledgments

We are indebted to Prof. Tomás Lang for insight discussions and helpful comments on this paper.

This work has been partially supported by CICYT TIC94-0531-E and Dept. d'Ensenyament de la Generalitat de Catalunya.

References

[1] J. Bunda, W. Athas, and D. Fussell. Evaluating power implications of CMOS microprocessor design decisions. In *Proc. Int. Workshop on Low Power Design*, pages 147–152, Apr. 1994.

[2] T. Burd and R. Brothersen. Energy efficient CMOS microprocessor design. In *Proc. 28th Hawaii Int. Conf. on System Sciences*, Jan. 1995.

[3] T. Callaway and E. Swartzlander. Estimating the power consumption of CMOS adders. In *Proc. of the Custom Integrated Circuit Conf.*, pages 210–216, 1993.

[4] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. Brodersen. HYPER-LP: A system for power minimization using architectural transformations. *IEEE Trans. on CAD*, pages 300–303, Nov. 1992.

[5] A. Chandrakasan, S. Sheng, and R. Broderssen. Low power CMOS digital design. *IEEE Trans. on SSC*, 27(4):473–483, Apr. 1992.

[6] A. Chatterjee and R. Roy. Synthesis of low power linear DSP circuits using activity metrics. In *Proc. of the Int. Conf. on VLSI Design*, pages 265–270, Jan. 1994.

[7] A. de Graaf and A. van Genderen. SLS: Switch-level simulator user's manual. Technical report, Delft Univ. of Tech., 1987.

[8] S. Devadas, K. Keutzer, and J. White. Estimation of power dissipation in CMOS combinational circuits using boolean function manipulation. *IEEE Trans. on CAD*, 11(3):373–383, Mar. 1992.

[9] P. Dewilde, E. Deprettere, and R. Nouta. *Parallel and pipelined VLSI implementation of signal processing algorithms*, chapter 15, pages 257–264. VLSI and Modern Signal Processing. Prentice-Hall, Inglewood Cliffs, NJ, 1985.

[10] M. Ercegovac and T. Lang. Reducing transition counts in arithmetic circuits. In *Proc. Int. Symp. on Low Power Electronics*, pages 64–65, Oct. 1994.

[11] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-level synthesis: introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[12] A. Ghosh, S. Devadas, K. Keutzer, and J. White. Estimation of average switching activity in combinational and sequential circuits. In *Proc. DAC*, pages 253–259, 1992.

[13] I. Koren. *Computer Arithmetic Algorithms*. Prentice-Hall, 1993.

[14] S. Kung. On supercomputing with systolic/wavefront array processor. In *Proc. of the IEEE*, pages 867–884, July 1984.

[15] P. Landman and J. Rabaey. Black-box capacitance models for architectural power analysis. In *Proc. Int. Workshop on Low Power Design*, pages 165–170, Apr. 1994.

[16] J. Lim. *Two-Dimentional Signal and Image Processing*. Signal Processing Series. Prentice-Hall, 1990.

[17] C. Lin and S. Kwatra. An adaptive algorithm for motion compensated colour image coding. *IEEE Globecom*, 1984.

[18] F. Najm. Transition density, a stochastic measure of activity in digital circuits. In *Proc. DAC*, pages 644–649, 1991.

[19] M. Potkonjak and J. Rabaey. Algorithm selection: A quantitative computation-intensive optimization approach. In *Proc. of the IEEE Int. Conf. on Computer Aided Design*, pages 90–95, 1994.

[20] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[21] K. Rao and P. Yip. *Discrete Cosine Transform*. Academic Press, 1990.

[22] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. On average power dissipation and random pattern testability of CMOS combinational logic networks. In *Proc. of the IEEE Int. Conf. on Computer Aided Design*, 1992.

[23] J. Treichler, C. Johnson, Jr., and M. Larimore. *Theory and Design of Adaptive Filters*. New York: John Wiley & Sons, 1987.

[24] A. van Gerenden. SLS: An efficient switch-level timing simulator using min-max voltage waveforms. In *Proc. VLSI 89 Conf.*, pages 79–88, Aug. 1989.

[25] N. Weste and Eshragian. *Principles of CMOS VLSI Design: A systems Perspective*. Addison-Wesley, 1988.

[26] S. Wuytack, F. Catthoor, F. Franseen, L. Nachtergaele, and H. D. Man. Global communications and memory optimizing transformations for low power. In *Proc. Int. Workshop on Low Power Design*, pages 203–208, Apr. 1994.

[27] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimizu. A 3.8-ns CMOS 16x16-b multiplier using complementary pass-transistor logic. *IEEE JSSC*, 25(2):388–395, Apr. 1990.