Scheduling and resource binding for low power

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 the scheduling and resource-binding steps of high-level synthesis. Algorithms for these steps targeting at low-power data-paths and trading off, in some cases, speed and area for low power are presented.

The algorithms focus on reducing the activity of the functional units (adders, multipliers) by minimizing the transitions of their input operands. The power consumption of the functional units accounts for a large fraction of the overall data-path power budget.

1 Introduction

Current VLSI technology allows circuits with more and more functionality to be integrated in just one chip. Nowadays, portable applications are not only wrist clocks or calculators but multi-media terminals, mobile telephones and other real-time systems. These new applications are based on intensive data-path tasks such as video compression, speech recognition and other digital signal processing tasks. The portable feature of these applications imposes a limit on power consumption whereas the real-time characteristic forces the designer to comply with the required throughput.

Power consumption can be taken into account at different levels in the design process [4]: technological, topological, architectural and algorithmic levels. High-level synthesis (HLS) comprises techniques at the architectural and algorithmic level. Design decisions taken in the HLS process have a significant impact on the quality of the final implementation. Traditionally, HLS has been applied to obtain small and fast designs, but including power consumption as one of the design parameters or constraints has rarely been addressed.

Preliminary studies in the HLS steps of scheduling and resource binding [9] targeting at low power reported in [14] have guided the algorithms presented in this paper.

The main target for reducing power consumption is the set of functional units (adders, multipliers) because its power consumption accounts for a large fraction of the overall data-path power budget. The algorithms attempt to reduce the activity of the functional units by minimizing the switching activity of their input operands.

Models derived from switch-level simulations of the main data-path components (functional, interconnection and storage units) [14] will be used to estimate the power reduction achieved with the algorithms.

The paper is organized as follows: in Section 2, previous work on low-power circuits with special insight in high-level techniques is briefly presented. Section 3 discusses how the functional units consume power in data-path intensive systems. It briefly describes the scheduling and resource-binding tasks along with the basic ideas behind the algorithms presented in the paper. Sections 4 and 5 describe how the scheduling and resource-binding algorithms for low power are implemented. Results are presented for some benchmarks. Power reduction results are obtained by comparing traditional scheduling and resource-binding methods with ours targeting at low power. Section 6 concludes the paper.

2 Previous work

Most of the efforts in HLS for low power propose models and estimations of power consumption at algorithmic and architectural level [12, 13, 2, 6, 15].

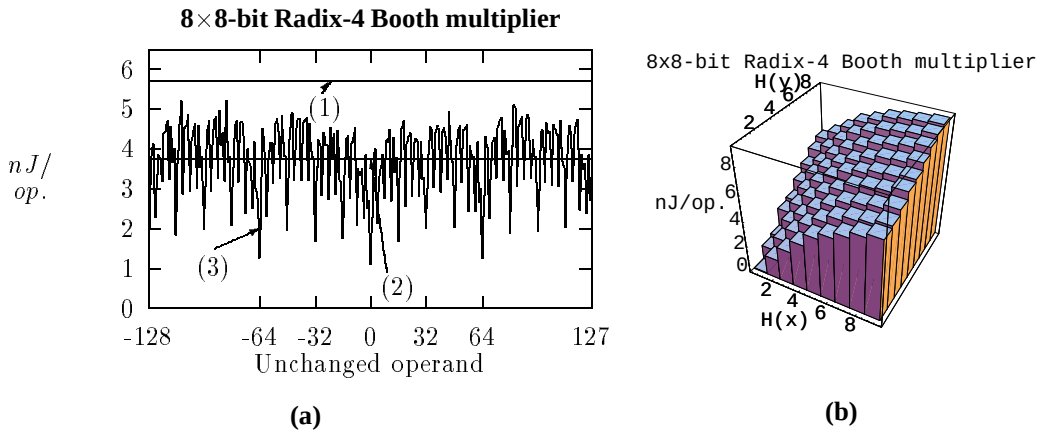
Few authors have addressed the set of transformations at algorithmic and architectural level to obtain lower-power designs. In [5],the power consumption of additions and constant multiplications as a function of the operand activity is studied. From this study, a data flow graph transformation is derived for a typical operation in signal processing applications. In [21], some memory transformations for low power systems are hinted. The aim of these transformations is to reduce the number of off-chip references. In [3], 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.

High-level synthesis for low power has been addressed in [17, 7, 14]. In [17], an allocation method that attempts to reduce both the capacitance and switching activity of the synthesized design is presented. In [7], a scheduling and binding technique for reducing the activity in the buses is described.

The algorithms presented in this paper are based on preliminary results reported in [14], where high-level synthesis techniques for reducing the activity of functional units are also described and their potential benefits evaluated.

3 Power consumption of the functional units

Power consumption in the data-path accounts for a large fraction of the overall system power budget. Among the



(a) **(b)**

Figure 1: Energy of a multiplier as a function of the **(a)** operand repetition and **(b)** operand activity.

	Benchmark	+/∗	⊗, ⊕	idle
1	*AR filter [11]*	12/16	1 p. ⊕ (2 cycles) 1 ⊗ (1 cycle)	27%
2	*4th-order Daubechies Wavelet filter [16]*	12/16	1 ⊗ (2 cycles) 1 ⊕ (1 cycle)	37%
3	*1-D 8-input Lee DCT [18]*	28/13	4 ⊗ (2 cycles) 3 ⊕ (1 cycle)	40%
4	*4 × 4 matrix multiplication*	4/8	2 ⊗ (2 cycles) 1 ⊕ (1 cycle)	33%
5	*loop-unrolled low-pass image filter [14]*	24/0	4 ⊕ (1 cycle)	25%
6	*LMS adaptive filter [19]*	8/9	2 ⊗ (2 cycles) 1 ⊕ (1 cycle)	45%
7	*pixel interpolation [1]*	5/0	3 ⊕ (1 cycle)	44%
8	*5th-order Wave filter [8]*	26/8	1 ⊗ (2 cycles) 1 ⊕ (1 cycle)	33%

Table 1: Idle time spent by the functional units for some high-level synthesis benchmarks assuming a schedule with the number of functional units in the third column. The total number of operations for each benchmark is shown in second column.

different types of units that compose a data-path, power consumption is mainly considered in the functional units due to their large contribution to the power consumption of the data-path.

The power consumption of a functional unit depends on the operand variability of its inputs. Figure 1 illustrates this fact for an 8×8 radix-4 Booth multiplier [10].

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(b) represents the energy in $nJ/operation$ as a function of the average Hamming distance (AHD) of the operands defined as $AHD(x) = \lim_{n\to\infty} \frac{\sum_{i=1}^{n} H(x_i, x_{i-1})}{n}$, where $H(p,q)$ is the Hamming distance between p and q and x_i is the value of operand x in cycle i. Obviously, the power consumption tends to zero when the AHD of both operands tends to zero. The power consumption in the multiplier with an AHD of its operands of 4 and 2 is approx. 25% less than with AHD values of 4 and 6.

A functional unit in a data-path consumes both *useful* and *useless* power. It consumes useful power when it is executing an operation and consumes useless power when there is an input operand transition while the functional unit is idle. The control unit is usually synthesized using *don't care* values to minimize area or increase speed. Thus, an idle functional unit may have input operand changes due to the variation of the selection signals of multiplexers.

Useless power is specially important in data-paths synthesized from *sparse* schedules. A schedule is said to be *sparse* if its unit occupation is relatively low. Table 1 presents the functional unit occupation for some benchmarks.

The power consumption of a functional unit (idle or not) depends on the operand variability of its inputs. In the sequel, we will distinguish between *operand activity* and *operand repetition*. Both concepts are related to the variability of the bit-pattern that represents the operand. *Operand activity* relates to the variability of the bit-pattern of one operand from one cycle to the next. *Operand repetition* relates to the coarse-grained variability of the operand, i.e. the operand may or may not change between two consecutive cycles.

Figures 1(a) and 1(b) illustrate how the power consumption of a multiplier can be studied as a function of its operand repetition and operand activity respectively.

Simple power-consumption models have been derived for each of the main data-path components as a function of the operand repetition and operand activity [14].

Since we focus on data-path circuits, whenever we refer to the *power consumption* of a design we mean the *energy per operation* executed by that design. Data-path circuits have a fixed throughput and, therefore, the energy/operation is the best metric that quantifies the energy efficiency for these type of circuits [2].

3.1 Scheduling and resource binding for low power: basic ideas

The HLS process is divided in three basic tasks [9]: allocation, scheduling and resource binding. The latter task is

[1] Although data is correlated for some of the HLS applications, we focus on fairly compare the relative benefits of different circuit descriptions.

itself decomposed into functional, storage and interconnection unit binding steps, all of them tightly related to each other. They are usually ordered and executed sequentially due to the high complexity of the resource-binding task.

Two traditional approaches for the scheduling and resource-binding tasks have been modified to target at low-power designs and their algorithms are presented in this paper. Both algorithms attempt to reduce power consumption only in the functional units. They do not address the reduction of power in I/O, clocks or data transfers.

The scheduling algorithm for low power uses a list-scheduling approach where the priorities of the operations of the ready-operation queue are set in such a way that operations sharing the same operand are scheduled in control steps as close as possible. Thus, the potential for a functional unit to reuse the same input value (and, therefore, to decrease its input activity) is higher.

The resource-binding algorithm for low power is based on a clique partition of a restricted variable-lifetime compatibility graph to obtain a register set that, for each functional unit, reduces the power consumption during idle cycles. Power consumption in functional units during non-idle cycles is further decreased by taking into account the AHD among the variables of the behavioral description and the commutative property of some operations.

Although the scheduling technique will obtain better improvements if applied to dense schedules (e.g. schedules where the functional unit occupation is high) and the resource-binding technique is more suitable to sparse schedules, both techniques are compatible and complementary.

4 Scheduling for low power

The goal of the scheduling algorithm for low power is to increase the potential for a functional unit (FU) to reuse an operand. Henceforth, we will call operand reutilization (OPR) the fact that an operand is reused by two operations consecutively executed in the same FU.

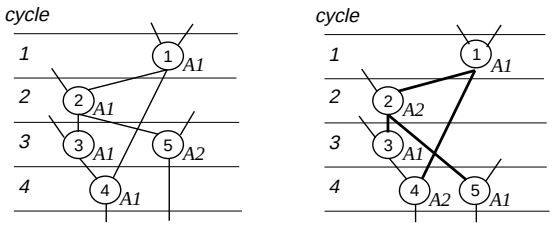


Figure 2: **(a)** One possible schedule and FU binding with no OPRs assuming two adders and **(b)** improved schedule and FU binding with 2 OPRs.

Figure 2, where two schedule and FU bindings of a simple data-flow graph (DFG) are shown, illustrates the OPR concept. There are some operations in the DFG whose result is the input for more than one operation. For example, the result of addition 1 is input for additions 2 and 4. Assume that additions 2 and 4 are assigned to the same adder A. Assume also that between the execution of addition 2 and 4 there is no other use of adder A. Then, one of the operands of adder A will not change from addition 2 to addition 4.

Figure 2(a) shows a schedule and an FU binding with two adders obtained with a traditional list-scheduling algorithm (LS) for the scheduling task and a clique-partitioning approach with weights to minimize the number of interconnection units for the FU-binding task. None of the two OPRs are achieved.

Figure 2(b) shows the schedule and FU binding obtained with the list-scheduling algorithm for low power (LPLS) for the scheduling task and a slightly different approach to the clique-partitioning for the FU-binding task. Now both OPRs are achieved.

LPLS also trades off latency for OPRs. This idea is also illustrated in Figure 2. If addition 5 happens to be in the critical path, the schedule and FU binding in Figure 2(c) has one more cycle of latency than the one in Figure 2(b).

4.1 LPLS key features

Some heuristics have been included in the traditional list-scheduling algorithm (Figure 3(a)) to obtain its low-power version (see Figure 3(b) for a simplified algorithm). Algorithms in Figure 3 follow the notation in [9].

Those operations that share an operand are grouped in *operand-sharing sets* (henceforth, SS) (CREATE_ALL_SS()). All operations of a group (IS_OSS()) can be executed on the same FU. An operation of an SS is able to reserve the FU where it is going to be assigned for the rest of its SS in case it has not one reserved yet (RESERVE_FU_IN_SS()). Given an SS and its reserved FU, in the best case $|\text{SS}| - 1$ OPRs can be obtained. All these consecutive OPRs on the same FU are called an *operand-sharing chain*. LPLS attempts to schedule as many operations as possible of the SS on its reserved FU. It also attempts not to execute other operations on it in order to prevent breaking the operand-sharing chain (OBTAIN_FREE_AND_NOT_RESERVED_FU()). The scheduling of the operations of an SS is guided by giving more priority to the operations in the operand-ready queue whose SS has already a reserved FU (UPDATE_PRIORITIES()). The priority of an operation is decreased (i.e. will be scheduled later) if it is going to be assigned to an FU not reserved by its SS. If the operation scheduled in a later cycle happens to be in the critical path, the final latency is increased.

All the information about achieved OPRs gathered during the execution of LPLS is transferred to the FU-binder as a set of binding constraints. The FU-binder first complies with all these constraints (i.e. achieves all OPRs already obtained by LPLS) and after that proceeds as the traditional FU-binder with weights to minimize the number of interconnection units (multiplexers).

LS has a complexity of $O(n)$, where n is the number of operations. LPLS has a complexity of $O(n^2m)$, where m is the number of unit types.

4.2 Results

LS is compared with its low-power version LPLS over some data-path benchmarks. With LS, many of the OPRs are achieved because the FU binder already forces some OPRs in its attempt to minimize the number of multiplexers.

Several results are shown in Table 2. The benchmarks have been scheduled with the resources reported in Table 1. To estimate power consumption, 12-bit-wide FUs are assumed.

The effect of an OPR on the power consumption of an FU has been evaluated by measuring the energy of the FU as a function of the operand repetition (see Section 3). The

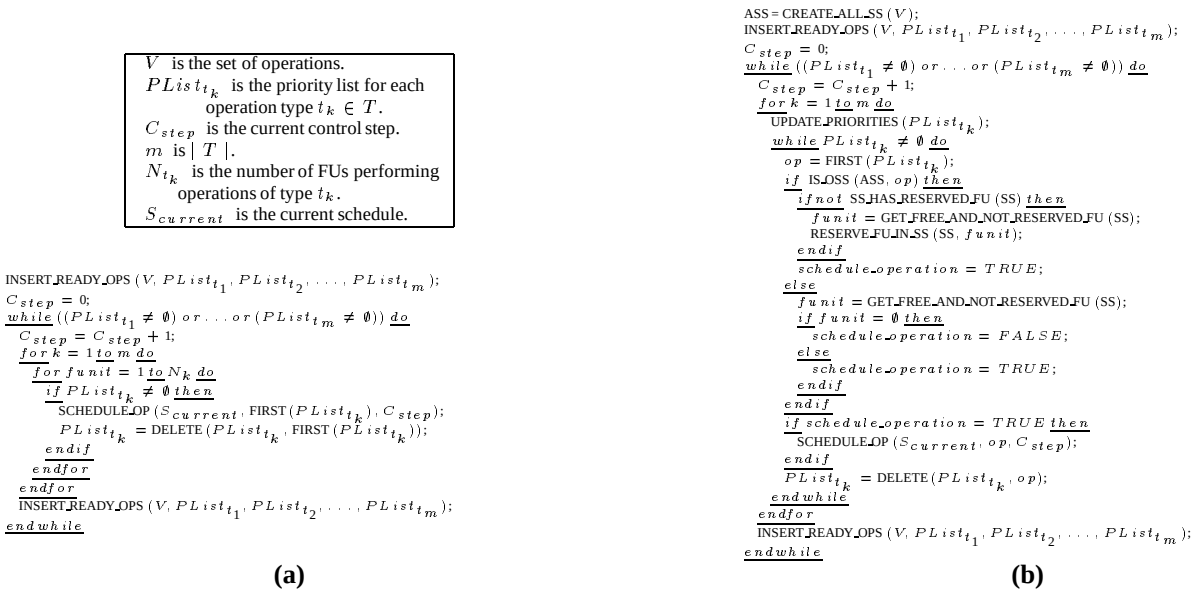


Figure 3: **(a)** Traditional list-scheduling algorithm **(b)** list-scheduling algorithm for low-power.

(1)	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*	*(7)*
1	20	20	4 ⊗	3 ⊗	4 ⊗	2%
2	20	22	10 ⊗	7 ⊗	10 ⊗	17%
3	14	15	11 ⊕/3 ⊗	5 ⊕/2 ⊗	5 ⊕/2 ⊗	0%
4	11	11	6 ⊗	0	4 ⊗	7%
5	9	9	9 ⊕	2 ⊕	9 ⊕	10%
6	17	17	3 ⊗	3 ⊗	3 ⊗	0%
7	5	5	2 ⊕	1 ⊕	1 ⊕	0%

Table 2: Latency and number of OPRs (for both type of FUs) achieved. *(1)* benchmark; *(2/3)* latency obtained with LS/LPLS; *(4)* max. OPRs; *(5/6)* achieved OPRs with LS/LPLS and *(7)* power reduction in the functional units.

last column of Table 2 accounts for the savings in power consumption in the FUs due to the increment of achieved OPRs obtained with LPLS. The power consumption due to an operation of the benchmark depends on the type of FU where this operation is scheduled and on how many operand changes that FU has when it executes the operation. A 17% of power reduction is achieved in the Daubechies filter and a 7% in the 4×4 matrix multiplication. The rest of the benchmarks present a small or null power-consumption reduction due to the following reasons: (a) the maximum number of OPRs is too small compared to the number of operations of the benchmark and (b) the null or little increase in OPRs achieved by LPLS with respect to LS.

5 Resource binding for low power

The goal of the resource-binding algorithm for low power (LPRB) is to reduce power consumption in the FUs once the scheduling and FU-binding tasks have been done. LPRB tackles both useful and useless power consumption of FUs.

LPRB assumes that the control unit maintains, for each FU, the same registers on its inputs during idle cycles.

The LMS benchmark (see Figure 4(a) for its DFG) will illustrate how LPRB works.

5.1 Reducing useless power

LPRB addresses the reduction of useless power consumption by building up a register set that minimizes the number of input changes on the idle units. All this process is represented in the first part of the algorithm in Figure 5.

A traditional approach for building up a register set (register binding) is the clique-partitioning method. After applying this method to a lifetime compatibility graph for the variables (CG), each clique of the partition corresponds to one register. LPRB uses the same traditional approach but applied to a different variable-compatibility graph (LPCG). To build up the LPCG, the register-binding for low power first constructs the CG (CREATE_CG()). In a second step, a set of edges of the CG are removed (REMOVE_EDGE()). Each edge removed from the CG connects two compatible variables with the following property: should both be assigned to the same register, an idle FU would have an input change.

Figure 4(b) illustrates this concept. It shows the schedule and FU binding for the DFG of Figure 4(a). The shadowed slots represent the cycles in which the FUs are idle. For each FU, the variables in parenthesis in the shadowed slots force the control unit to maintain the same registers on its inputs during idle cycles. Let us consider what happens with FU $A0$ in cycle 10. An input change will occur at the inputs of idle unit $A0$ if, for example, variables $v16$ and $v21$ are assigned to the same register because multiplier $M0$ will modify the value of that register in cycle 9. The same happens with variable pair $(v20 - v21)$. But not all the variables of these two pairs have compatible lifetime between them. In this example, only the pair $(v20 - v21)$ does. Thus, for the FU $A0$ in cycle 10, this edge is removed from the CG. If the same procedure is applied to all the idle slots of Figure 4(b), 6 edges will be removed.

The drawback in removing edges is the possibility to obtain a larger register set, as it will be confirmed later with the results.

Not all the useless power consumption in the idle FUs

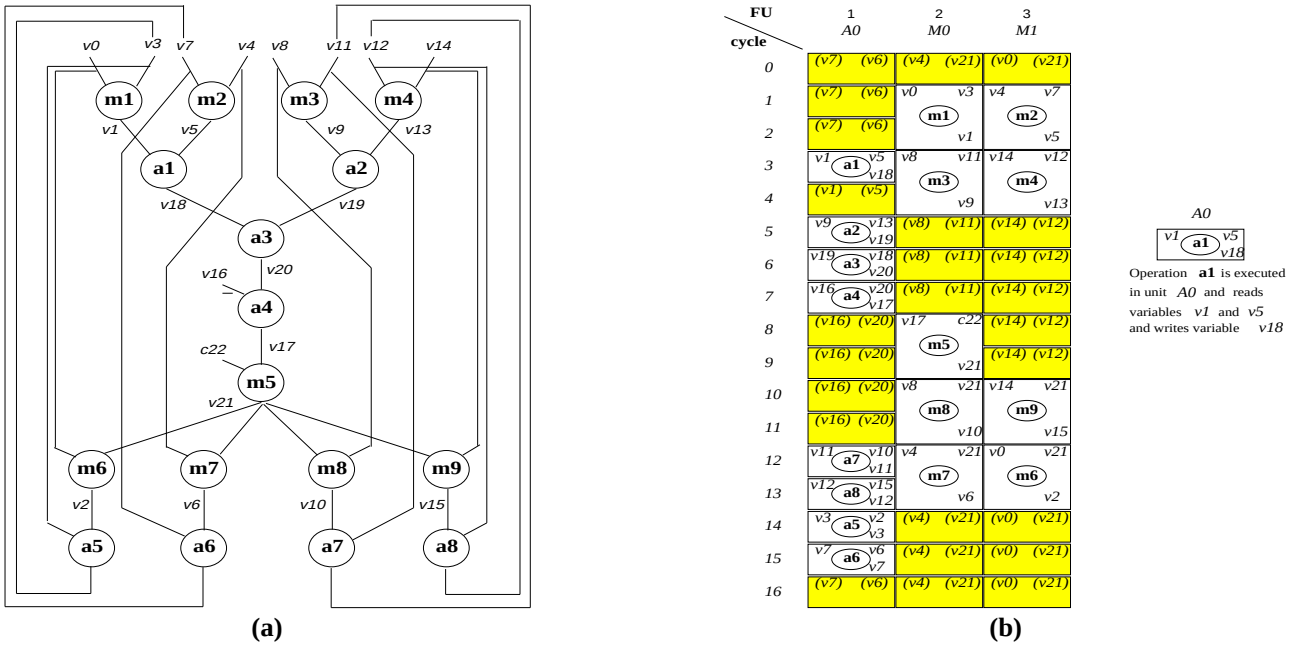


Figure 4: **(a)** DFG of the LMS filter and **(b)** Schedule and FU binding with one adder (one cycle) and two multipliers (two cycles).

```
/* reduce power consumption in idle functional units */
CG = CREATE_CG (V);
for c = 1 to MAX_CYCLES do
  for fu = 1 to MAX_FUs do
    if IDLE (fu, c) then
      for each operation op whose result
      is in cycle (c − 1) MOD MAX_CYCLES do
        op_source = OPERATION_IN_FU (fu);
        REMOVE_EDGE (CG, <VAR_DEST(op_source), VAR_A(op) >);
        REMOVE_EDGE (CG, <VAR_DEST(op_source), VAR_B(op) >);
      endforeach
    endif
  endfor
endfor
REGISTER_BINDING(CG);
/* reduce power consumption in non − idle functional units */
for each FU fu do
  OBTAIN_BEST_VARIABLE_ORDER (fu, AHD);
endforeach
INTERCONNECTION_UNIT_BINDER();
```

Figure 5: Resource binding algorithm for low power.

is eliminated with this technique. As an example, let us consider FU $A0$ in cycle 16 in Figure 4(b). Because the previous operation executed in FU $A0$ has variable $v7$ as an operand and as the result, FU $A0$ has in cycle 16 an input change.

5.2 Further reduction of useful power

Once the register set has been derived, the useful power consumption in FUs may be reduced if the commutative property of some operations and the average Hamming distance (AHD) among the variables are taken into account. The process to reduce the power consumption in non-idle units is shown in the second part of the algorithm in Figure 5.

As an example, consider additions $a1$ and $a2$ of Figure 4(b). With the variable input order shown, the FU $A0$ has an AHD on one of its inputs of $H(v1, v9)$ and on the other input of $H(v5, v13)$. Recall from Section 3 how the power consumption of an FU depends on the AHD of its inputs. If the AHD information among the variables is available, the reduction in power can be evaluated if the variable order in addition $a2$ is changed. The problem of obtaining the best variable order for all operations requires an exhaustive exploration. Thus, for simplicity, LPRB follows a greedy approach (OBTAIN_BEST_VARIABLE_ORDER()).

By defining a variable order, the degrees of freedom for the interconnection-unit binder are reduced because the correct variable order (which implies the correct register order) has to be satisfied. This implies that the number of multiplexers will be at least equal to the number obtained if no useful power is reduced.

5.3 Results

TRB is compared with its low-power version LPRB over three data-path benchmarks for which we have representative input data. The AHD among the variables has been obtained by means of profiling the benchmarks[2]. In all of them, the scheduling and FU-binding tasks have been done with the low-power methods described in Section 4. The benchmarks have been scheduled with the resources reported in Table 1.

By means of switch-level simulations [20] of the basic functional units, multiplexers and registers, power-consumption models similar to the one in Figure 1(b) have been obtained. 12-bit-wide FUs are assumed in the power results.

For both resource-binding algorithms, useful and useless

[2]It is important to notice that the AHD among the variables highly depends on the input data. The AHD of the benchmarks related to image processing has been obtained using the well-known Lena benchmark. We have observed that the AHD values converge fast (in approx. 500 iterations of the algorithm).

Bench.	LPLS and TRB						LPLS and LPRB						Power Red.
	(1)	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*	*(1)*	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*	
3	21	14.9	65	33.2	73.0/182.1	303.2	23	14.7	67	34.2	48.5/184.0	281.4	7%
6	12	4.6	23	13.3	1.4/104.3	123.6	12	4.5	24	13.9	0.2/98.4	117.0	5%
7	6	1.21	5	0.85	0.33/1.45	3.84	7	1.38	4	0.68	0.0/1.46	3.52	8%

Table 3: Comparison between the traditional resource-binding algorithm (TRB) and its low-power version (LPRB). All power estimations are in $nJ/iteration$. *(1)* number of registers; *(2)* power due to registers; *(3)* number of multiplexers; *(4)* power due to multiplexers; *(5)* useless/useful power of FUs and *(6)* total data-path power.

power consumption of FUs, and the number of registers and multiplexers[3] along with estimations of their power consumption are reported in Table 3.

In the 1-D 8-input Lee DCT and pixel interpolation benchmarks, no improvement has been observed when applying the algorithm for reducing the useful power consumption in FUs. The greedy method used did not change the variable order for any FU.

In the pixel interpolation benchmark, only two adders are used. This implies that the power consumption due to the registers and multiplexers plays an important role in this benchmark.

It is worth noticing the area-power trade-off: in two benchmarks the number of registers and multiplexers has increased when applying LPRB. Although the total area has increased, the power consumption has has been reduced.

6 Conclusions

Algorithms that reduce the activity of the functional units by minimizing the switching activity of their input operands have been presented for the high-level synthesis tasks of scheduling and resource binding.

Significant power-consumption reduction is obtained in the scheduling task with little increase or no increase at all in latency. Further power reduction is achieved in the resource-binding task by increasing the number of storage and interconnection units and taking into account both the commutative property of some operations and the average Hamming distance among the variables of the data-flow graph to be synthesized.

In this paper, the impact of the number of functional units on the power consumption has not been addressed. Our future work is devoted to the evaluation of this impact.

Acknowledgment

We would like to thank to Rosa Badía for her constructive comments which were instrumental in improving this paper.

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

References

[1] C. Brown and B. Shepherd. *Graphics File Formats: reference and guide*. Prentice-Hall, 1995.

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

[3] 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.

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

[5] 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.

[6] R. M. D. Marculescu and M. Pedram. Information theoretic measures of energy consumption at register transfer level. In *Int. Symp. on Low Power Design*, pages 81–86, Apr. 1995.

[7] A. Dasgupta and R. Karri. Simultaneous scheduling and binding for power minimization during microarchitectural synthesis. In *Int. Symp. on Low Power Design*, pages 69–74, Apr. 1995.

[8] 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.

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

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

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

[12] 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.

[13] P. Landman and J. Rabaey. Activity-sensitive architectural power analysis for the control path. In *Int. Symp. on Low Power Design*, pages 93–98, Apr. 1995.

[14] E. Musoll and J. Cortadella. High-level synthesis techniques for reducing the activity of functional units. In *Int. Symp. on Low Power Design*, pages 99–104, Apr. 1995.

[15] F. Najm. Towards a high-level power estimation capability. In *Int. Symp. on Low Power Design*, pages 87–92, Apr. 1995.

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

[17] A. Raghunathan and N. Jha. Behavioral synthesis for low power. In *Proc. of the Int. Conf. on Computer Design*, pages 318–322, Oct. 1994.

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

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

[20] 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.

[21] 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.

[3] The equivalent number of 2-input multiplexers