UNIVERSITY OF CALIFORNIA,

IRVINE

Low Power Division and Square Root

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Engineering

by

Alberto Nannarelli

1999

The thesis of Alberto Nannarelli is approved

and is acceptable in quality and form

for publication on microfilm:

Committee Chair

University of California at Irvine
1999

To Chiara Elisa and Tanja,

for their patience.

"E quindi uscimmo a riveder le stelle."

"And we walked out to see again the stars."

La Divina Commedia - Inferno XXXIV, 139.

Dante Alighieri (1265-1321)

Contents

List of Figures

List of Tables

Acknowledgement

I am indebted to Professor Tomás Lang for his insight and direction during these years at UCI. I am grateful for his confidence in me, and for having had the opportunity to work with him. Our work was partially funded by the National Science Fundation (grant MIP 9314172) and by the state of California with the industrial support of Sun Microsystems Inc., through UC MICRO 97-084.

I also wish to thank the members of my thesis committee, Professors Nader Bagherzadeh, Nikil Dutt, Rajesh Gupta, and Fadi Kurdahi, for their helpful comments and recommendations regarding my work.

A special thank you goes to Professor Enrico Macii from Politecnico di Torino, Italy, for providing me with a cell library suitable for synthesis for low-power.

Curriculum Vitae

Alberto Nannarelli

1988	Eng. Degree in Electrical Engineering, Universita' "La Sapienza", Roma (Italy).
1989	Military Service in the Italian Army.
1990-91	Design Engineer, SGS-Thomson Microelectronics, Agrate (Italy).
1991-93	System and Software Engineer, Ericsson Telecom, Roma (Italy), Stockholm (Sweden).
1994-99	Research Assistant, Dept. of Electrical and Computer Eng., University of California, Irvine.
1995	M.S. in Engineering, University of California, Irvine.
1995	Summer Intern, Rockwell Semiconductor Systems, Newport Beach.
1999	Ph.D. in Engineering, University of California, Irvine. Dissertation: Low Power Division and Square Root Professor Tomas Lang, Chair

FIELD OF STUDY

Numerical Processors and VLSI Design

Professor Tomas Lang

PUBLICATIONS

A. Nannarelli and T. Lang. "Low-Power Radix-4 Divider". *Proc. of International Symposium on Low Power Electronics and Design*, pages 205-208, Monterey, CA. August 1996.

A. Nannarelli and T. Lang. "Power-Delay Tradeoffs for Radix-4 and Radix-8 Dividers", *Proc. of International Symposium on Low Power Electronics and Design*, pages 109-111, Monterey, CA. August 1998.

A. Nannarelli and T. Lang. "Low-Power Radix-8 Divider", *Proc. of International Conference on Computer Design*, pages 420-426, Austin, TX. October 1998.

A. Nannarelli and T. Lang. "Low-Power Divider", *IEEE Transactions on Computers*, pages 2-14, January 1999.

A. Nannarelli and T. Lang. "Low-Power Division: Comparison among implementations of radix 4, 8 and 16". *Proc. of 14th Symposium on Computer Arithmetic*, pages 60-67, Adelaide (AUS), April 1999.

A. Nannarelli and T. Lang. "Low-Power Radix-4 Combined Division and Square Root", *to appear in Proc. of International Conference on Computer Design*, Austin, TX. October 1999.

Abstract of the Dissertation

Low Power Division and Square Root

by

Alberto Nannarelli

Doctor of Philosophy in Engineering

University of California, Irvine, 1999

Professor Tomás Lang, Chair

The general objective of our work is to develop methods to reduce the energy consumption of arithmetic modules while maintaining the delay unchanged and keeping the increase in the area to a minimum. Here, we present techniques for dividers and square root units realized in CMOS technology. The energy dissipation reduction is carried out at different levels of abstraction: from the algorithm level down to the implementation, or gate, level. We describe the use of techniques such as switching-off not active blocks, retiming, dual voltage, and equalizing the paths to reduce glitches. Also, we describe modifications in the on-the-fly conversion and rounding algorithm and in the redundant representation of the residual in order to reduce the energy dissipation. The techniques and modifications mentioned above are applied to several division and square root schemes, realized with static CMOS standard cells, for which a reduction in the energy dissipation of about 40 percent is obtained with respect to the standard implementation optimized for minimum delay. This reduction is expected to be even larger if low-voltage gates, for dual voltage implementation, are available.

Introduction

In recent years the demand for low-power electronic systems has increased due to both the massive advent of portable devices, which require small and light batteries, and the increased densities on chip and the consequent necessity of reducing the energy dissipated.

In digital systems the number of transistors on a chip doubles every two years and the smaller device size allows the use of faster clocks. As a consequence, the charging and the discharging of many devices due to more frequent transitions of the signals causes an increase of energy dissipation. This increase in energy consumption results in many side effects that can increase the cost of the system or even compromise its functionality.

It is reported that current microprocessors, such as the Pentium or the Alpha, dissipate about 30 W [1]. A system that dissipates more than 2 W cannot be placed in a plastic package and the use of ceramic packaging, heat sinks, and coolant fans raises significantly the cost of the product. Moreover, a chip that dissipates 30 W at 3.3 V requires wires on the circuit board that can deliver a current of about 10 A.

More serious problems can arise in case of large current densities, since electromigration, caused by large currents flowing in narrow wires, might produce gaps or bridges in the power-rails of the chip with a consequent permanent damage of the system.

The possibility to put entire systems on a chip and the miniaturization of I/O devices (displays, sensors, etc...), brought on the market a variety of portable products such as cellular phones, laptop computers, personal digital assistants (PDA), GPS receivers, and medical devices. The critical resource of these systems is the battery lifetime, which can be lengthened by reducing the energy dissipation. This reduction also enables the use of smaller, and lighter, batteries. A cellular phone that requires a battery recharge every hour or a laptop computer powered by a car battery are not very practical. For this reason it is essential that portable systems be designed for low-power.

This work investigates the implementation of low-power double-precision floating point division and square root units compliant with the IEEE standard [2]. These units are common in general-purpose processors, but the results obtained can be extended also to units with a different number of bits implemented in DSP cores or other application-specific processors.

Although division and square root are not very frequent ignoring their implementations can result in system performance degradation [3]. We briefly summarize some facts stated in [3]. Table 0.1 shows the average frequency of floating-point (FP) operations in the SPECfp92 benchmark suite. By assuming a machine model of a scalar processor, where the FP-adder and FP-multiplier have both a latency of 3 cycles and the FP-divider has a latency of 20 cycles, a distribution of the excess CPI (cycles per instruction) due to stalls in the FP-unit is shown in Figure 0.1. The stall time is the period during which the processor was ready to execute the next instruction, but the dependency upon an unfinished FP-operation prevented it from continuing. This excess CPI reduces the overall performance by increasing the total CPI. Figure 0.1 shows that, although division is less frequent than addition and multiplication, its longer latency accounts for 40% of the performance degradation. For this reason, many general-purpose microprocessors implement division in hardware and try to make it fast enough not to compromise the overall performance.

Percent of all FP instructions	
division	3 %
square root	0.3 %
multiplication	37 %
addition	38 %
other	21 %

Table 0.1: Instruction mix.

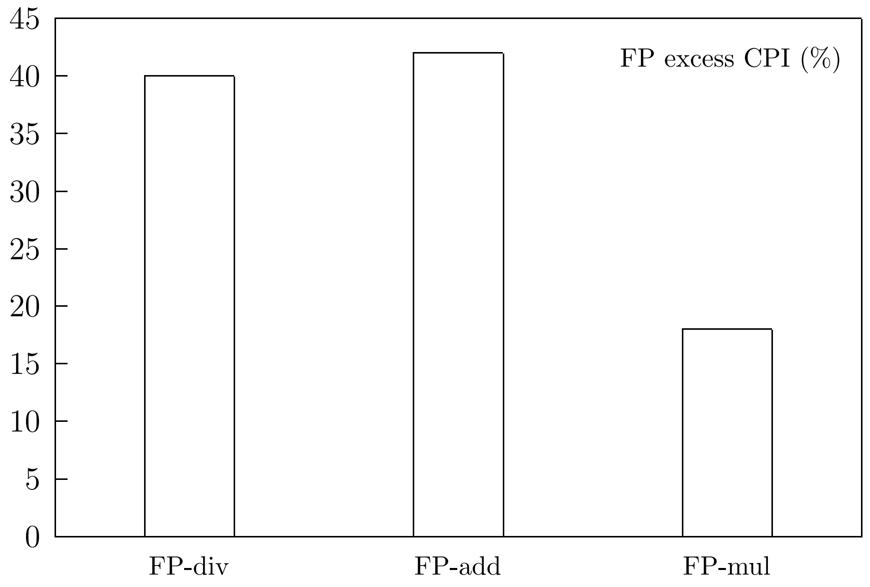


Figure 0.1: FP-unit stall time distribution.

	FP-adder	FP-multiplier
technology	CMOS 0.5 μm, 3.3 V	CMOS 0.5 μm, 3.3 V
f_{MAX}	164 MHz	286 MHz
Area	$2.5 \times 3.5\ mm^2$	$4.2 \times 5.1\ mm^2$
n. pipeline stages	5	5
Energy per	3.24 mW/MHz	5.10 mW/MHz
operation	$E_{add} = 16\ nJ$	$E_{mul} = 25\ nJ$

Table 0.2: Data on implementations [5] and [6].

As for the energy dissipation, no data on the comparison of division with other FP operations are available in literature. In [4] (pages 194-198), a quite coarse evaluation is done using only the number of transitions to estimate the energy dissipation of a radix-2 and a radix-4 divider, without an actual implementation. Because this evaluation did not take into account the switching capacitance and only the recurrence part of the low radices 2 and 4 is evaluated, the results obtained do not illustrate very well the design issues of low-power dividers. The implementations of a FP-adder and FP-multiplier, realized by the same group of people with the same technology, are described in [5] and [6], respectively. Table 0.2 summarizes the data of the two implementations.

In order to evaluate what percentage of the energy consumption is dissipated in the divider, we implemented with our library the 54×54-bit multiplier described in [7] and determined the energy consumed per multiplication which resulted to be $E_{mul} = 15\ nJ$. By assuming the same ratio $\frac{E_{mul}}{E_{add}}$ as in Table 0.2, we estimated the energy consumed in a FP addition as $E_{add} = 10\ nJ$. Finally, we computed the energy dissipation for the radix-4 divider of Section 4.2, which resulted to be $E_{div} = 40\ nJ$. Combining these values with the instruction mix of the program *spice* (see Table 0.3 [8]), we obtained the breakdown for the energy dissipated in the FP-unit shown in Figure 0.2.

Figure 0.2 shows that although division is less frequent than addition and mul-

Instruction	Percent	Unit
division	8 %	FP-div
multiplication	26 %	FP-mul
addition	14 %	
subtraction	22 %	
comparison	7 %	
move	2 %	
	45 %	FP-add
other	20 %	-

Table 0.3: Instruction mix in program *spice*.

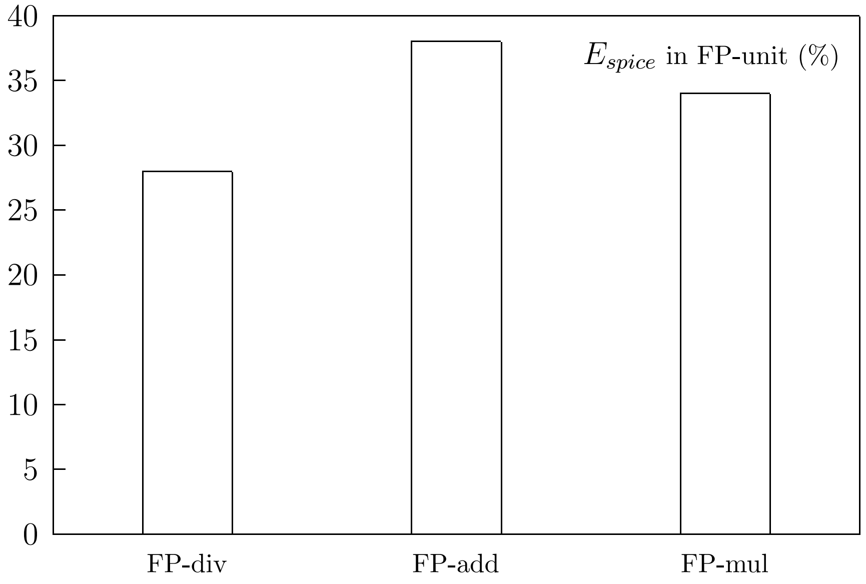


Figure 0.2: Breakdown of energy in FP-unit.

tiplication, because of its longer latency, it dissipates about 30% of the total energy consumed in the floating-point unit when running the program *spice*. Consequently, it is important that division unit be designed for low-power. For these reasons, we explore the possibilities of reducing the energy dissipated in division and square root units. Our main objective is to reduce the energy consumption without increasing the execution time. However, we also consider tradeoffs between delay and energy in some cases. Furthermore, we study the relation between energy dissipation and the radix of division and square root implementation.

The research is carried out by implementing, with a static CMOS standard cell library, a set of division and square root units, and by applying several techniques aiming to reduce the energy dissipation. Since the energy dissipated in CMOS cells is proportional to the number of transitions, to the output load, and to the square of the operating voltage [9], we reduce the number of transitions, the capacitance and estimate the impact of using a lower voltage. For the energy reduction, we separate the units into two portions, the recurrence and the on-the-fly conversion and rounding [10]. In the first portion, we retime the recurrence to reduce the glitches and to constrain the critical path to the most-significant slice. This allows the replacing in the non-critical slice of the radix-2 carry-save adder cell by a radix-r version to reduce the number of flip-flops. Moreover, in the non-critical slice we use low-drive and low-voltage cells. Finally, we equalize the signal paths to reduce the glitches. For the on-the-fly conversion and rounding, we modify the algorithm to reduce the number of flip-flops and their activity. For these low-activity flip-flops, we use individual gated clocks to reduce the energy of the flip-flops that do not change. In addition, we implement gated trees to reduce the energy in the distribution of signals. The energy dissipation is computed from the actual implementations in most cases, and estimated in others.

Results show that the energy dissipated to complete one operation is almost constant for several radices, and that in most cases it is possible to reduce the energy dissipation between 40 and 60 percent without increasing the latency.

This work is organized as follows. Chapter 1 introduces background concepts related to energy dissipation and the standards used in the number representation. Chapter 2 presents the algorithms used to perform division and square root. Chapter 3 describes the techniques and methodologies used to reduce the energy dissipation. Chapter 4 presents the actual implementations of the units and the application of the techniques of Chapter 3. Chapter 5 summarizes the results obtained and discusses some of the tradeoffs among different implementation. Finally, Chapter 6 draws the conclusions.

Chapter 1
Background

Introduction

The main purpose of this chapter is to provide the necessary background for the concepts and the methods presented in this work. First, we introduce the metrics used to evaluate the energy and power dissipation and illustrate the main sources of energy consumption in VLSI circuits based on static CMOS technology. Then, we discuss different approaches aiming to reduce the energy dissipation, and a list of simulation and optimization tools, at different levels of abstraction, is presented. In the last part of the chapter, the IEEE format for floating-point and its utilization for division and square root are briefly described.

1.1 Metrics

In this work a common measure of the energy dissipation is required in order to evaluate and compare different approaches in low-power design. Because the algorithms are in general different and the latency of the operations varies from case to case, it is convenient to have a measure of the energy dissipated to complete an operation. This energy-per-operation is given by

$$E_{op} = \int_{t_{op}} vi\ dt \qquad [J]$$

where t_{op} is the time elapsed to perform the operation. The energy-per-operation is computed on a cell basis as the sum of the energy E_i dissipated in the ith-cell during t_{op}

$$E_{op} = \sum_{i=1}^{N} E_i \qquad [J] \qquad \text{with } E_i = \int_{t_{op}} vi_i\ dt \qquad [J].$$

Operations are usually performed in more than one cycle and the expression of t_{op} is typically

$$t_{op} = T_{cycle} \times (no.\ of\ cycles) \qquad [s].$$

By dividing the energy-per-operation by the number of cycles we obtain the energy-per-cycle

$$E_{pc} = \frac{E_{op}}{no.\ of\ cycles} \qquad [J].$$

The average power dissipation is the product of E_{pc} and the clock frequency

$$P_f = E_{pc} f = \frac{E_{op}}{t_{op}} = V_{DD} I_{ave} \qquad [W] \tag{1.1}$$

where V_{DD} is the supply voltage and I_{ave} is the average current.

1.2 Energy Dissipation in CMOS

Over the past decade, CMOS technology has played a dominant role in the market of digital integrated circuits, and it is expected to continue in the near future. For this reason, this work is focused on CMOS systems. Two components characterize the amount of energy dissipated in a CMOS circuit [9]:

- Dynamic dissipation due to the charging and discharging of load capacitances and to the short-circuit current.
- Static dissipation due to leakage current and other current drawn continuously from the power supply.

The total energy dissipation for a CMOS gate can be written as

$$E_{gate} = E_{load} + E_{sc} + E_{leakage}\ . \tag{1.2}$$

The quantity E_{load} is the energy dissipated for charging and discharging the capacitive load C_L when n_i output transitions occur. If in a gate (like the one in

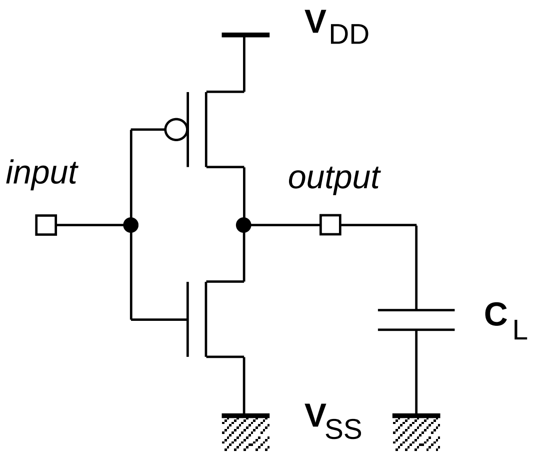


Figure 1.1: CMOS inverter loaded with C_L.

Figure 1.1) one transition from the logic level "low" ($V_{SS} = 0\ V$) to "high" (V_{DD}) occurred[1] at time t, we can write

$$E_t = \int_0^t vi\ dt = \int_0^t v\ C_L \frac{dv}{dt}\ dt = C_L\ \int_0^{V_{DD}} v\ dv = \frac{1}{2} C_L V_{DD}^2\ . \tag{1.3}$$

Consequently, for n_i output transitions we have:

$$E_{load} = \frac{1}{2} C_L V_{DD}^2 n_i\ . \tag{1.4}$$

The energy due to the short-circuit current is E_{sc}. In a CMOS inverter (Figure 1.1), during a transition both the n and the p-transistors are on for a short period of time. This results in a short current pulse from the power supply voltage (V_{DD}) to ground (V_{SS}). With no loading the short-circuit current is quite relevant, while by increasing the output loading the current drawn for charging or discharging the capacitance, becomes dominant. E_{sc} depends on V_{DD}, the transition time, the gate design, the load C_L and n_i ([11] pages 92-97).

The energy due to leakage currents $E_{leakage}$ is small and usually neglected, unless the system spends a large amount of time in stand-by or sleep status.

[1]One transition from V_{DD} to V_{SS} produces identical results.

In the analysis of more complex gates, especially in standard cells libraries, the energy is usually split into two contributions:

- energy dissipation due to the loading of the cell, which coincides with E_{load}
- energy dissipated internally, which is the sum of E_{sc} and the energy dissipated in charging and discharging the internal capacitances.

Therefore, the expression of the average energy dissipated in a cell is

$$E_i = E_{load} + E_{int} = (\ \frac{1}{2} C_L V_{DD}^2 + E^{int} \ ) \ n_i \ . \tag{1.5}$$

in which E^{int} is the energy dissipated internally per transition and the term between parenthesis represents the energy per transition.

For a circuit composed of several cells, the energy dissipation can be computed as the sum of the energy dissipated in each cell. That is,

$$E_{total} = \sum_{i=1}^{N} E_i = \sum_{i=1}^{N} (\ \frac{1}{2} C_{L_i} V_{DD}^2 + E_i^{int} \ ) \ n_i \ . \tag{1.6}$$

1.3 Approaches to Energy Dissipation Reduction

Several techniques have been developed to reduce the energy dissipation of CMOS systems. By expression (1.2) and expression (1.4), the minimization can be carried out by reducing the supply voltage, the capacitance, the number of transitions (e.g. the activity in the circuit), and by optimizing the timing of the signals and the design of the gate to reduce the energy due to short-circuit currents.

A large impact on energy is made by the supply voltage. By reducing V_{DD} the energy dissipation decreases quadratically, but the delay increases and the performance is degraded. A possible solution is that of using different supply voltages in different parts of the circuit [12]. The parts not in the critical path are supplied by lower voltages, while the critical one by the higher voltage [13]. Another technique is to compensate the loss of performance by replicating the hardware (parallelism) to keep the throughput [14].

Capacitance can be reduced at different levels. At transistor, or layout, level by keeping the size of the device small and by optimizing the wire interconnection capacitance during the floor-planning and the routing. At gate level, by using gates specially designed for low-power and by merging a set of gates into a more complex cell eliminating the interconnection capacitance [15]. It is important to note that by reducing the capacitance, not only the energy dissipation, but also the performance will be improved.

The number of transitions can be reduced at transistor level, by equalizing the delay of the different paths to avoid the generation of glitches [16], and at register-transfer (RT) level, by disabling both combinational and sequential blocks not used at a particular time [17]. Combinational logic can be disabled by forcing a constant logic value at its inputs, while in sequential circuits this can be obtained by disabling the clock [18]. This last technique, known as clock gating, can be also implemented at gate-level by gating the clocks to individual flip-flops [19]. Retiming is the circuit transformation that consists in re-positioning the registers in a sequential circuit without modifying its external behavior [20]. By retiming it is possible to stop the propagation of glitches reducing the activity in the system. A combined optimization of number of transitions and capacitance is obtained by swapping a pin whose activity is high with a pin with lower capacitance [15].

Further reduction are achieved by changing the data encoding and the algorithm [21], [13].

The energy dissipation due to short-circuit currents can be reduced by careful design at gate level and by buffering in order to avoid long transition (rise/fall) times [11].

Finally, energy dissipation can be reduced by changing the fabrication process to support very low-voltages, copper interconnects, and insulators with low dielectric constants [1].

In this work, we reduce the energy by applying minimization technique at RT-level and gate-level. Optimization of short-circuit energy dissipation and transistor level techniques are not covered.

1.4 Asynchronous Systems

Recently there has been a renewed interest in asynchronous circuits due to the potential better power efficiency over the traditional synchronous (clocked) systems ([11] pages 461-492).

Clocked circuits waste energy by clocking all parts of the chip whether or not they are doing useful work. Clock trees are also responsible for a significant portion of the energy dissipated in the chip. In asynchronous circuits the number of transitions is reduced, but the self-timing requires the use of additional logic for control signals. There is a tradeoff between number of transitions and capacitance (extra logic).

In this work, the research on low-power division and square root is limited to synchronous circuits.

Examples of a self-timed divider and of a self-timed shared division and square root unit are presented in [22] and [23], respectively. The area of the latter unit, as stated in [23], is about 1.7 larger than the corresponding synchronous implementation. However, no information on power or energy dissipation is provided in the articles in question, and a comparison with the corresponding synchronous units is undoable because of unknown parameters such as circuit activity and switching capacitance.

1.5 Tools for Low-Power Design

Computer-aided design (CAD) tools are used to speed-up the design process and improve the productivity. As mentioned above, techniques for low-power integrated circuits (IC) design can be applied at every level of abstraction and some CAD tools that take into account power constraints, in addition to the traditional delay and area constraints, start to be available [11].

In the design of a system two fundamental aspects are analysis and optimization. CAD tools analyze a system to extract information on performance, area and power dissipation. This information is then used to evaluate if the designed system met the constraints and/or to optimize the design. Estimators for average energy dissipation can be either based on simulation or on probabilistic models of the energy dissipated in a circuit, or on statistical estimation techniques [24].

Methods based on simulation give good accuracy and are straightforward to implement. Simulations at transistor level monitor the power supply current waveform, at higher level the number of transitions is counted and energy is estimated by expression (1.6), or equivalent. However, simulation methods are pattern-dependent and in an early phase of the design, patterns generated by several functional blocks might be still unknown. Furthermore, the simulator and the energy estimator can either be tightly-coupled or loosely-coupled [25]. In tightly-coupled systems the estimation is done at run time, while in loosely-coupled systems the simulator outputs the transition statistics on a file for the energy estimator. The main advantage of the latter is the flexibility: different simulators can be used in different design stages.

The estimation using probabilities alleviates the pattern-dependency problem.

Instead of simulating the circuit for a large number of patterns and then averaging the result, one can assume a distribution of the probability of the inputs and use that information to estimate how often internal nodes switch. Signal probabilities are propagated into the circuit assuming different timing, probability propagation and energy models that, depending on the specific tools, take into account temporal and spatial correlation of the signals, short-circuit energy and so on. To some extent, the process is still pattern-dependent because the user has to supply the probabilities of the inputs. However, this information might be more readily available than specific input patterns. The drawback of these estimators is that they use simplified models, so that they do not provide the same accuracy as circuit simulations. Better accuracy can be obtained at expenses of more complicated models and longer execution times. There is a tradeoff between accuracy and speed.

Statistical methods do not require specialized models. They use traditional simulation models and simulate the circuit for a limited number of randomly generated input vectors while monitoring the energy. Those vectors are generated from user-specified probabilistic information about the circuit inputs. Using statistical estimation techniques, one can determine when to stop the simulation once a specified estimation error is obtained. Details of these methods are given in Section 4.1.1.

In general, it is not clear which is the best approach, but statistical methods offer a good mix of accuracy, speed and ease of implementation [24].

CAD tools can be differentiated by the level of abstraction at which they operate. We describe below, tools to perform analysis and synthesis for low-power.

1.5.1 Transistor Level

Tools for estimation at transistor level achieve the best accuracy, but require the longest run time. At this level, energy evaluation is done by simulations and SPICE is the reference among the simulators. However, other commercial tools claim an accuracy within 5% of SPICE and execution times up to x1000 faster [25]. Transistor level estimators are typically used to characterize cells and modules for use at the higher abstraction levels.

Optimization at this level is done by tools which resize the transistors according to given power/delay/area constraints [25].

1.5.2 Gate Level

Energy estimation at gate level is less accurate than energy estimation at the transistor level, but it is faster and can be done in an earlier stage of the design with good accuracy (10-15%). Energy values can typically be reported by signal, gate or blocks of gates.

Optimization is done by using several techniques (refer to Section 1.3) to reduce the energy under given timing constraints. One popular commercial tool with power optimization capability is Synopsys Power Compiler [26].

1.5.3 Architectural Level

At this level estimation is mainly done with probabilistic models by analyzing VHDL or Verilog descriptions of the system. The accuracy is in the range 20-25%, but large circuits can be analyzed in a short time at an early stage of the design [1]. A commercial tool available for estimation at this level is Sente WattWatcher/Architect [27].

Optimization at this level is currently an interactive process, consisting in the evaluation of various design alternatives and the subsequent choice of the design that best fits the project constraints [1].

1.6 Floating-Point Division and Square Root

1.6.1 IEEE Floating-Point Standard

The IEEE floating-point standard 754 defines formats for binary representation of floating-point numbers [2]. The two basic formats are the single-precision 32-bit format and the double-precision 64-bit format. We now, briefly describe the double-precision format which is the one used in the rest of this work.

The 64 bits of the double-precision format are divided into three fields: 1-bit field representing the sign S, a 11-bit field representing the biased exponent E, and a 52-bit field f which represents the fractional part of the significand $(1.f)$. Thus, the floating-point number F is represented by the following expression

$$F = (-1)^S 1.f \; 2^{E-1023} \; .$$

Because the significand is normalized in the range $1 \leq 1.f < 2$, its integer bit is always 1 and is omitted (hidden bit) in the binary representation. The IEEE standard also describes rounding schemes that are necessary when the number of bits required for the representation of a number exceeds the total allowed by the format. The round-off schemes are the following: truncation, round-to-nearest-even, round to $+\infty$, and round to $-\infty$ [28].

1.6.2 Division and Square Root

When performing the division of two floating-point numbers X and D, such as:

$$X = (-1)^{S_x} \; x \; 2^{E_x-1023} \text{ and } D = (-1)^{S_d} \; d \; 2^{E_d-1023}$$

three different operations have to be performed on sign, exponent, and significand to produce the quotient of the division Q

$$Q = \frac{X}{D} = (-1)^{S_q} \; q \; 2^{E_q-1023} \; .$$

The sign of Q is $S_q = S_x \oplus S_d$, its exponent is given by the subtraction $E_q = E_x - E_d$, and the significand by the division $q = x/d$. The quotient q produced by the division of the two significands is not normalized, but in the range $\frac{1}{2} < q < 2$, and a step of post-normalization is required when $x < d$. This post-normalization step consists in shifting q one position to the left and decrementing the exponent E_q by one.

An alternative to post-normalization is pre-shifting. Pre-shifting is done before performing the division by shifting one of the operands to obtain $x \geq d$ and consequently, q is already normalized in $[1, 2)$.

In square root,

$$S = \sqrt{X} = s\ 2^{E_s - 1023}\ ,$$

the sign of the radicand is always positive, the exponent must be halved and the square root operation has to be performed on the significand. The operation to perform on the exponent is the following:

$$E_s = \left\lfloor \frac{E_x - 1023}{2} \right\rfloor + 1023$$

and the significand x must be shifted one position to the right (pre-shifting) if E_x is even. For the significand, we compute:

$$s = \begin{cases} \sqrt{x} & \text{if } E_x \text{ is odd} \\ \sqrt{\frac{x}{2}} & \text{if } E_x \text{ is even.} \end{cases}$$

In the rest of this work, we describe only the operations (division and square root) to be performed on the significands and we treat rounding assuming that the operands are pre-shifted.

Chapter 2 Algorithms

Introduction

In this chapter the division and square root algorithm are summarized. In the first part of the chapter, we describe the digit-recurrence algorithm for division, the on-the-fly conversion and rounding algorithm and give an example of division performed with the two algorithms. Then two modifications to the division algorithm are discussed to make it suitable for high radices. Finally, the square root algorithm and its combination with division are described.

2.1 Division Algorithm

Digit-recurrence algorithms for division and square-root give probably the best tradeoff delay-area [29], and are the focus of this work. Digit-recurrence algorithms produce a fixed number of result bits every iteration, determined by the radix. Higher radices reduce the number of iterations to complete the operation, but increase the cycle time and the complexity of the circuit.

The division algorithm, described in detail in [10], is implemented by the residual recurrence

$$w[j+1] = rw[j] - q_{j+1}d \qquad j = 0, 1, \ldots, m-1 \tag{2.1}$$

with initial value $w[0] = x$, where r is the radix, x the dividend, d the divisor, and q_{j+1} the quotient digit at the j-th iteration, such that the quotient is

$$q = \frac{x}{d} = \sum_{j=1}^{m} q_j r^{-j} \tag{2.2}$$

where m is the number of iterations needed to produce the $n+1$ bits of the conventional representation (53 for IEEE double-precision format + one rounding bit). Both d and x are normalized in $[0.5, 1)$ and $x < d$[1].

The quotient digit is in signed-digit representation $\{-a, \ldots, -1, 0, 1, \ldots, a\}$ and the residual $w[j]$ is stored in carry-save representation (w_S and w_C). The quotient digit is determined, at each iteration, by a selection function

$$q_{j+1} = SEL(d_\delta, \hat{y})$$

where d_δ is d truncated after the δ-th fractional bit and

$$\hat{y} = rw_{St} + rw_{Ct}$$

where rw_{St} and rw_{Ct} refer to the carry-save representation of the shifted residual truncated after t fractional bits. The quotient digit is selected so that

$$-\frac{a}{r-1}d < w[j] < \frac{a}{r-1}d \tag{2.3}$$

Since expression (2.3) is the condition for convergence for the algorithm, x might need to be shifted one bit to the right to have a bounded residual $w[0]$ in case $a < r-1$. Moreover, for simplicity of implementation, it is convenient to have the rounding bit produced in the least-significant bit of the quotient digit (i.e. in the last iteration we compute both bits of the quotient and the bit to be rounded), and to achieve this, x is shifted to the right accordingly. A correction step is required at the end if the final residual is negative. In addition, rounding to the nearest-even is done by adding 1 in the last bit of the partial quotient. To perform this correction and rounding, we need to determine the sign of the final residual and if it is zero (necessary for the round-to-nearest-even scheme).

[1]Because in IEEE standard floating-point quantities are normalized in $[1, 2)$ it is necessary to right-shift the operands one position. Furthermore, if $x \geq d$, x is right-shifted an extra position (pre-shifting).

The signed-digit representation of the quotient must be converted to the conventional representation in 2's complement; the on-the-fly convert-and-round algorithm performs this conversion as the digits are produced and does not require a carry-propagate adder.

A possible scheme to perform the division algorithm is shown in Figure 2.1. The recurrence is implemented with the selection function (SEL), the multiple generator (MULT), the carry-save adder (CSA) and two registers (REG) to store the carry-save representation of the residual. The number of bits in the recurrence (s), depends on the radix r and on the redundancy factor $\rho = \frac{a}{r-1}$. Because of the carry-save representation of the residual, the selection function in Figure 2.1 is composed by a b-bit carry-propagate adder and a logic function.

The conversion block performs the conversion from the signed-digit quotient and the rounding according to the sign of the final residual and the signal that detects if it is a zero, which are produced by the sign-zero-detection block (SZD). The scheme is completed by a controller (not depicted in the figure).

This algorithm is used effectively for radix 2, 4 and 8. For higher radices the selection function is too complex and its delay too high.

2.2 Conversion and Rounding Algorithm

We summarize the on-the-fly convert-and-round algorithm described in full detail in [10]. Three registers are needed to store Q, QM, and QP (Figure 2.2). However, as explained later, when the rounding is done in the least-significant position, and $a < r - 1$ two registers Q and QM are sufficient.

The rounding is done using the round-to-nearest-even scheme, which is mandatory in the IEEE standard [28]. The other rounding schemes are not discussed here, but they can be realized with the same unit used for the round-to-nearest-even case.

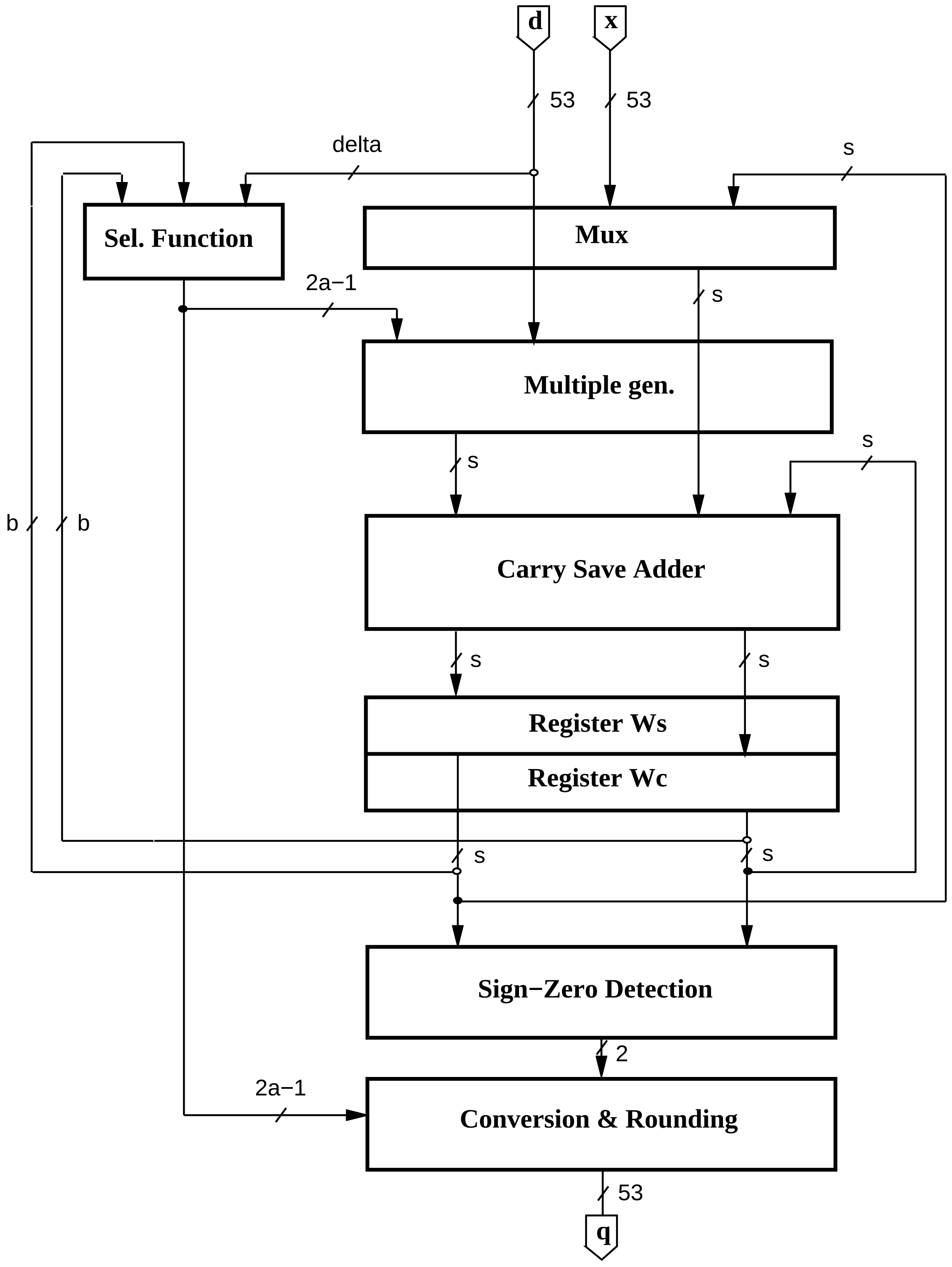


Figure 2.1: Block diagram of radix-r division.

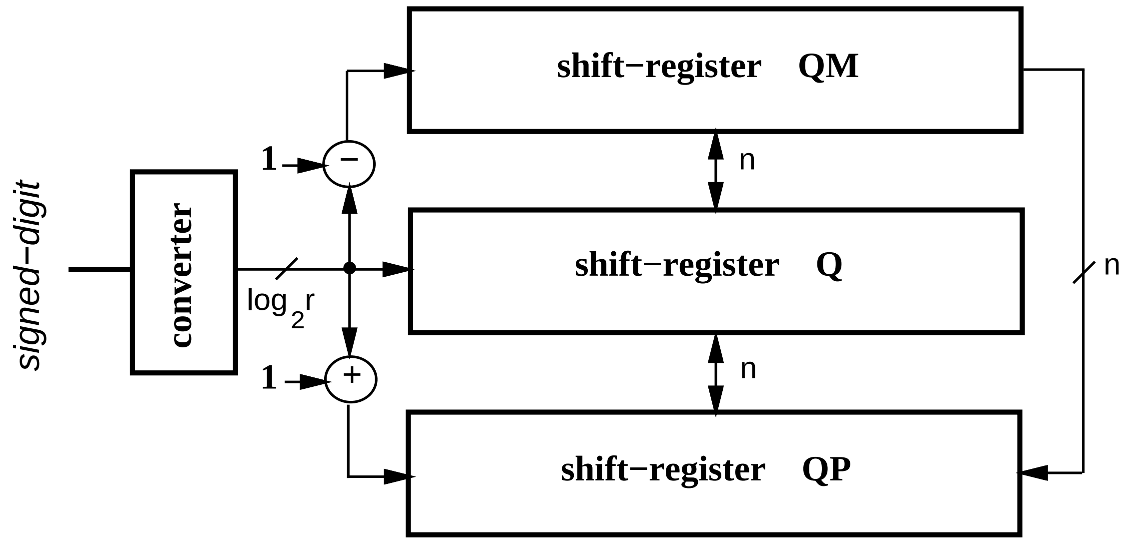


Figure 2.2: Convert and round unit.

After iteration j the three registers contain

$$Q[j] = q[j]\ r^{j-m} \quad , \quad QM[j] = (q[j]-1)\ r^{j-m} \quad \text{, and} \quad QP[j] = (q[j]+1)\ r^{j-m}$$

with

$$q[j] = \sum_{k=1}^{j} q_k r^{-k}.$$

Registers Q and QM are updated every iteration by the following rules:

$$\begin{aligned} Q[j] &\Leftarrow (\ shl(\ Q[j-1]\ ),\ q_j\ ) && \text{if } q_j > 0 \\ QM[j] &\Leftarrow (\ shl(\ Q[j-1]\ ),\ q_j - 1\ ) \end{aligned}$$

$$\begin{aligned} Q[j] &\Leftarrow (\ shl(\ Q[j-1]\ ),\ 0\ ) && \text{if } q_j = 0 \\ QM[j] &\Leftarrow (\ shl(\ QM[j-1]\ ),\ r - 1\ ) \end{aligned}$$

$$\begin{aligned} Q[j] &\Leftarrow (\ shl(\ QM[j-1]\ ),\ r - |q_j|\ ) && \text{if } q_j < 0 \\ QM[j] &\Leftarrow (\ shl(\ QM[j-1]\ ),\ (r-1) - |q_j|\ ) \end{aligned}$$

where, for example, $Q[j] \Leftarrow (\ shl(\ Q[j-1]\ ),\ q_j\ )$ means that the register Q at iteration j is shifted one digit to the left and the last digit is loaded with q_j. In

j	q_j	Q	QM
1	1	xxxxxxxx1	xxxxxxxx0
2	2	xxxxxxx12	xxxxxxx11
3	0	xxxxxx120	xxxxxx113
4	-1	xxxxx1133	xxxxx1132
5	0	xxxx11330	xxxx11323
6	0	xxx113300	xxx113233
7	2	xx1133002	xx1133001
8	-2	x11330012	x11330011
9	-1	113300113	113300112

Table 2.1: Example of radix-4 conversion.

QM, the current digit is given by $q_j - 1 \pmod{r}$. Table 2.1 shows an example of conversion for radix-4 and $a = 2$.

Register QP is updated every iteration by the following rules:

$$\begin{aligned} QP[j] &\Leftarrow (\ shl(\ QP[j-1]\ ),\ 0\ ) && q_j = r-1 \\ QP[j] &\Leftarrow (\ shl(\ Q[j-1]\ ),\ q_j + 1\ ) && -1 \le q_j \le r-2 \\ QP[j] &\Leftarrow (\ shl(\ QM[j-1]\ ),\ r - |q_j| + 1\ ) && q_j < -1 \end{aligned}$$

In the last iteration the rounding of the bit in the least-significant position is performed as follows. First the quotient digit q_m is converted into

$$g_m = q_m \pmod{r}\ .$$

Then, the rounded digit p is computed according to Table 2.2, where $SIGN = 0$ if the final residual is positive, $ZERO = 1$ if it is zero, and $G1$ represents the bit before the least-significant in g_m. Two operations have to be performed in the rounding step:

1. If the remainder is negative, the quotient must be decremented by 1 (in rounding position).
2. To round-to-the-nearest 1 has always to be added in rounding position.

q_m	$SIGN$	$ZERO$	p	case
$r-1$	0	0	0	1
	0	1	0	1
	1	-	$r-1$	2
$0 \leq q_m < r-1$	0	0	$g_m + 1$	2
	0	1	$g_m + G1$	2
	1	-	g_m	2
-1	0	-	0	2
	1	-	$r-1$	3
$q_m < -1$	0	0	$g_m + 1$	3
	0	1	$g_m + G1$	3
	1	-	g_m	3

Table 2.2: Values of p in the rounding step.

Finally, the quotient is obtained by

$$q = \begin{cases} shl(\ QP[m-1]\ ),\ p_t\ ) & \text{if case 1} \\ shl(\ Q[m-1]\ ),\ p_t\ ) & \text{if case 2} \\ shl(\ QM[m-1]\ ),\ p_t\ ) & \text{if case 3} \end{cases} \tag{2.4}$$

where

$$p_t = \left\lfloor \frac{p}{2} \right\rfloor \ .$$

2.3 Example of Division

We now show an example of application of the division algorithm for the case of radix-4 with $a = 2$. The selection function is given in Table 2.3. The quotient digit h selected is the one satisfying the expression $m_h \leq \hat{y} < m_{h+1}$. The example, shown in Table 2.4, is for the division x/d with $x = 0.5$ and $d = 0.6 = 0.10011001...$ (binary) which produces $q = 0.8\overline{3}$. The binary value of d_δ $(\delta = 3)$ is 001. The bit of weight 2^{-1} is always omitted because it is always 1 for the normalization $1/2 \leq d < 1$. Values in table, except q_{j+1}, are given as hexadecimal vectors.

m_h	d_δ							
	8/16	9/16	10/16	11/16	12/16	13/16	14/16	15/16
m_2	0C	0E	0F	10	12	14	14	18
m_1	04	04	04	04	06	06	08	08
m_0	7C	7A	7A	7A	78	78	78	78
m_{-1}	73	71	70	6E	6C	6C	6A	68

Values in table (multiplied by 16) are in hexadecimal

Table 2.3: Selection function for radix-4 division.

2.4 Division by Overlapping Stages

As the radix increases the number of iterations needed to compute the quotient of the division are reduced, but the selection function becomes more complicated.

Higher radices can be obtained by executing several recurrence iterations in the same cycle. This produces more bits of the result per cycle. However, the cycle time is lengthened and its longer delay offsets the benefit of having a reduced number of cycles. The only reduction in time is due to register loading that is done once for cycle. As an alternative, lower radices stages can be overlapped to reduce the cycle time and the latency of division [10, 30, 31]. When the delay in the selection function is dominant over the delay of the other components of the recurrence (carry-save adders, multiple generators, multiplexers) it might be convenient to replicate and overlap more selection functions.

In the case shown in Figure 2.3, two stages are overlapped. The first stage produces q_{j+1} which is used to select q_{j+2} among all the possible combination of $\hat{y}_{j+1} = trunc\ (rw[j] - q_{j+1}d)$. Because only a few bits of the carry-save representation of w are needed in the selection function, all $\hat{y}$s can be obtained by small CSAs at the input of the selection functions (one for each possible value of q_{j+1}). For example, for $a = 2$ five selection functions and four small CSAs are required to generate q_{j+2}. The resulting quotient digit, for the scheme of Figure 2.3, is

j	$\hat{y}$	q_{j+1}	$-q_{j+1}d$	$w_s[j]$	$w_c[j]$	q
0	00	0	000000000000000	020000000000000	000000000000000	00000000000000
1	08	1	366666600000000	1E66665FFFFFFFF	000000000000001	00000000000001
2	79	-1	099999A00000000	100000DFFFFFFF8	133332400000008	00000000000003
3	0C	1	366666600000000	1AAAAC20000003F	088886BFFFFFFC1	0000000000000D
4	0C	1	366666600000000	1EEECC200000007	044465BFFFFFFF9	00000000000035
5	0C	1	366666600000000	1CCCC0200000007	06666DBFFFFFFF9	000000000000D5
6	0C	1	366666600000000	1CCCD0200000007	06664DBFFFFFFF9	00000000000355
7	0C	1	366666600000000	1CCC10200000007	0666CDBFFFFFFF9	00000000000D55
8	0C	1	366666600000000	1CCD10200000007	0664CDBFFFFFFF9	00000000003555
9	0C	1	366666600000000	1CC110200000007	066CCDBFFFFFFF9	0000000000D555
10	0C	1	366666600000000	1CD110200000007	064CCDBFFFFFFF9	00000000035555
11	0C	1	366666600000000	1C1110200000007	06CCCDBFFFFFFF9	000000000D5555
12	0C	1	366666600000000	1D1110200000007	04CCCDBFFFFFFF9	00000000355555
13	0C	1	366666600000000	111110200000007	0CCCCDBFFFFFFF9	00000000D55555
14	77	-1	099999A00000000	1EEEEFDFFFFFFF8	022221400000008	00000003555553
15	03	0	000000000000000	13333A7FFFFFFC0	11110A000000040	0000000D55554C
16	10	2	2CCCCCC00000000	04440D4000001FF	1999D17FFFFFE01	00000035555532
17	77	-1	099999A00000000	1EEEE95FFFFFFF8	02222B400000008	000000D55554C7
18	03	0	000000000000000	1333087FFFFFFC0	11114A000000040	0000035555531C
19	10	2	2CCCCCC00000000	0445C54000001FF	1998517FFFFFE01	00000D55554C72
20	77	-1	099999A00000000	1EEFC95FFFFFFF8	02222B400000008	000035555531C7
21	03	0	000000000000000	1337887FFFFFFC0	11104A000000040	0000D55554C71C
22	10	2	2CCCCCC00000000	0453C54000001FF	1998517FFFFFE01	00035555531C72
23	77	-1	099999A00000000	1EB7C95FFFFFFF8	02922B400000008	000D55554C71C7
24	04	1	366666600000000	06F1EE20000003F	149C4ABFFFFFFC1	0035555531C71D
25	6D	-2	133333400000000	1A85A13FFFFFFF8	06E675800000008	00D55554C71C72
26	05	1	366666600000000	07E934A0000003F	142D8CBFFFFFFC1	035555531C71C9
27	6F	-2	133333400000000	1C21D33FFFFFFF8	076C65800000008	0D55554C71C722
28	0D	1	366666600000000	1B50BCA0000003F	094E8CBFFFFFFC1	35555531C71C89
29	rounding step:			sign (ws + wc) = 0	--> add 1	1
						35555531C71C8A

hex 35555531C71C8A trunc. in last bit = 1AAAAA98E38E45 = $0.8\overline{3}$ decimal

Table 2.4: Example of radix-4 division.

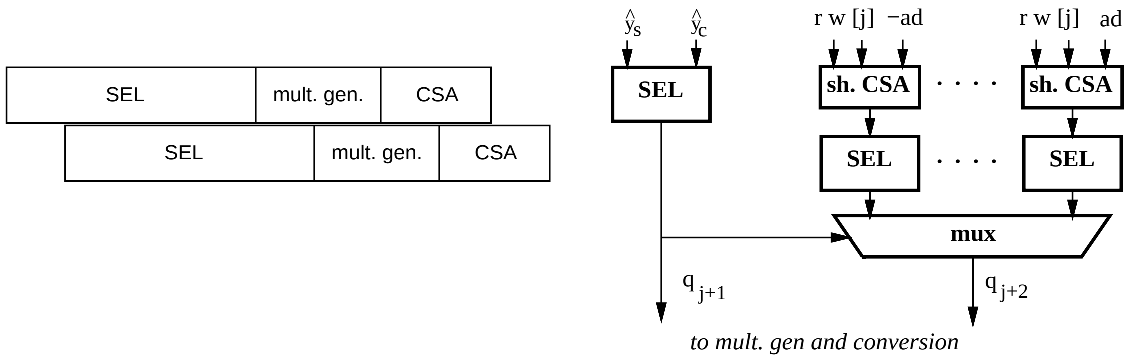


Figure 2.3: Selection function with overlapped stages.

$rq_{j+1} + q_{j+2}$.

Because of the replication of the selection function, which number is proportional to a, the radices which are suitable to be overlapped are 2 and 4. The drawback of this scheme is the use of hardware duplication and, therefore, a resulting larger area.

2.5 Very High Radix Division

Another division unit studied is the digit-recurrence algorithm radix-512 with scaling and quotient-digit selection by rounding, presented in [10, 32]. The unit, implemented in [33], showed a speed-up of about 2.0 over the radix-4 divider. Although radix-512 belongs to the category of the digit-recurrence algorithms, the implementation is quite different from the ones for lower radices and some structures such as recoders and trees of adders are present.

For radix-512 nine bits of the quotient are produced every iteration. To apply the quotient-digit selection by rounding, the divisor must be within a determined range. To achieve this, both operands are scaled by a quantity $M \approx \frac{1}{d}$ so that:

$$z = Md$$

and

$$w[0] = Mx$$

and the condition to be satisfied is:

$$1 - \frac{r-2}{4r(r-1)} < z < 1 + \frac{r-2}{4r(r-1)}$$

that for the specific case of radix-512 is

$$0.9995127 < z < 1.0004873 \quad .$$

The recurrence to be executed, for $r = 512$, is:

$$w[j+1] = rw[j] - q_{j+1}z \qquad\qquad j = 0, 1, \ldots 5$$

with initial value $w[0] = Mx$ and quotient-digit selection:

$$q_{j+1} = \lfloor \hat{y} + 1/2 \rfloor$$

where $q_{j+1} = \{-511, \ldots, 0, \ldots, 511\}$ is the quotient digit generated in iteration j and $\hat{y} = \{rw[j]\}_2$, that is $rw[j]$ truncated to its 2nd fractional bit. The quantity M can be calculated with different methods. By using linear approximation we obtain

$$M = -\gamma_1 d_{15} + \gamma_2$$

where M is truncated to its 13th fractional bit and the two coefficients

$$\gamma_1 = \frac{1}{{d_6}^2 + d_6 2^{-6} + 2^{-15}}$$

$$\gamma_2 = \frac{2d_6 + 2^{-6}}{{d_6}^2 + d_6 2^{-6} + 2^{-15}}$$

are also truncated to their 13th fractional bit and in the range:

$$1 < \gamma_1 < 4 \qquad\qquad 2 < \gamma_2 < 4 \quad .$$

Moreover, d_{15} and d_6 are the divisor d truncated to its 15th and 6th bit respectively.

The residual $w[j]$ is in carry-save representation to avoid carry-propagation in the addition and the quotient-digit q_{j+1} is also in carry-save representation before the recoding. The multiplication $q_{j+1}z$ is performed by recoding one of the operands. This recoding is done from the carry-save representation of the shifted residual and the recoder also produces the quotient-digit obtained by the rounding of two fractional bits of the shifted residual [32]. The recoded operand is in signed digit representation and each digit can assume the values $\{-2, -1, 0, 1, 2\}$.

The algorithm, represented by the block diagram in Figure 2.4, is divided into four parts:

1. M calculation (1 iteration),
2. scaling of the two operands (2 iterations),
3. execution of the recurrence (6 iterations),
4. final rounding (1 iteration),

for a total of ten iterations needed to perform one division.

An example of application of the radix-512 division algorithm is shown in Table 2.5, The division of $x = 0.5$ and $d = 0.6$ produces $q = x/d = 0.8\overline{3}$. Values in table, except q_{j+1} and z, are given as hexadecimal vectors.

2.6 Square Root Algorithm

The algorithm to compute the square root is quite similar to the division one. It is implemented, as described in [10], by the recurrence

$$w[j+1] = rw[j] - (2S[j]s_{j+1} + s_{j+1}^2 r^{-(j+1)}) \qquad j = 0, 1, \ldots m \tag{2.5}$$

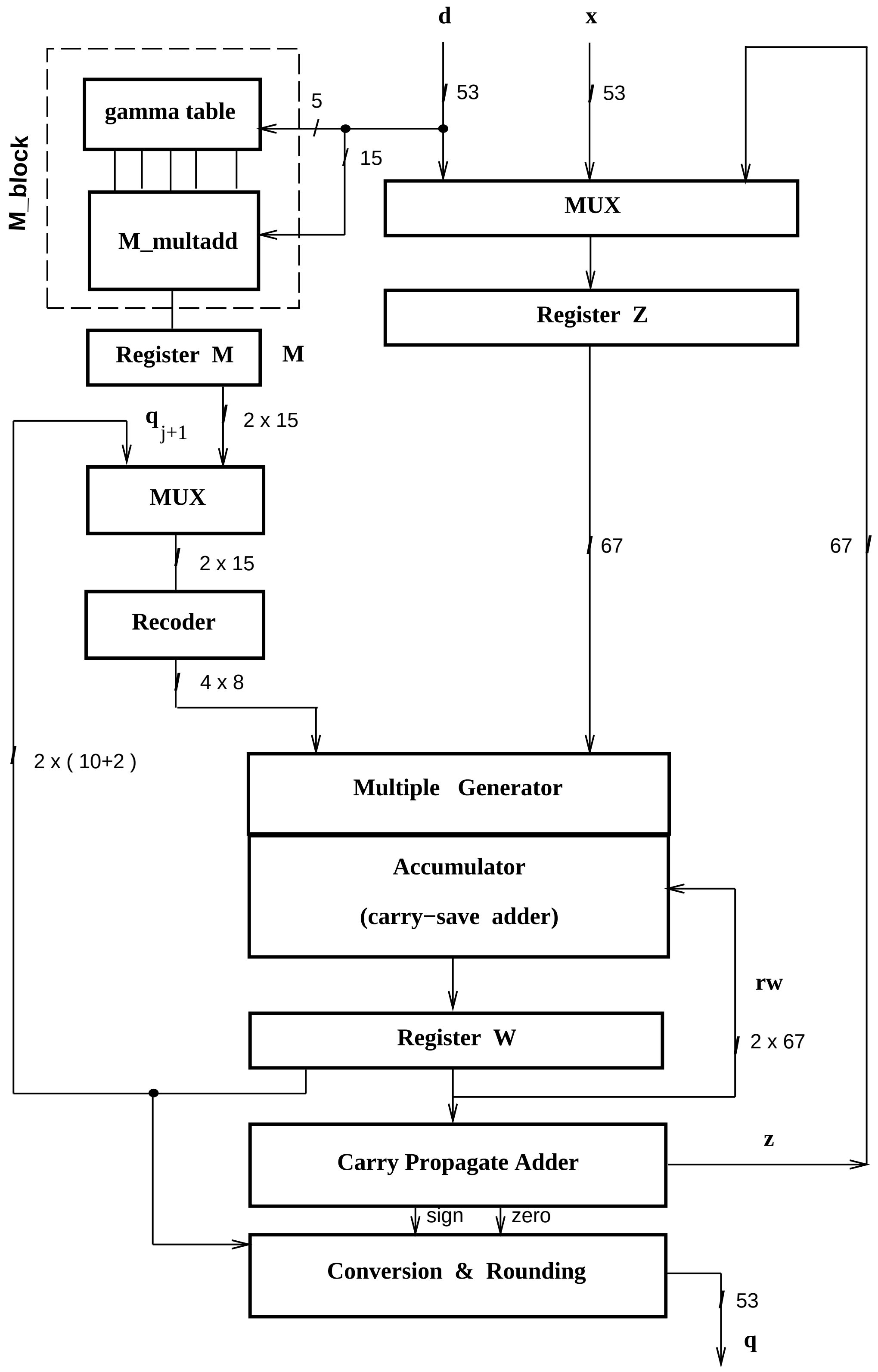


Figure 2.4: Block diagram of radix-512 divider.

j	q_{j+1}	$\hat{y}_s$	$w_s[j]$	$\hat{y}_c$	$w_c[j]$	q	z
-						Ms = 7557	Mc = 5550
-	-	379B	262DD97FFFFFC0	1065	66A44900000040	00000000000000	1.0002595 dec
0	-	3540	DFFFFFFFFFFFC0	116A	40000000000040	00000000000000	
1	427	3A2A	E07F0B7FFFF000	032A	3D016900001000	000000000001AB	
2	-341	2D66	FEF9F4807FFFF0	1532	124C16FF800010	00000000035555	
3	166	3E30	F52F5EFFFFFFC0	0396	46A54200000040	00000006AAAAA6	
4	114	3986	89595CFFFFFFE0	04B2	6A554600000020	00000D55554C72	
5	-398	3DD5	C83612FFFFFF80	0450	4AA34A00000080	001AAAAA98E38E	
6	136	3D05	A1B2768003FFF0	06D4	B49312FFFC0010	35555531C71C89	
rounding step: sign (ws + wc) = 0 --> add 1						1	
						35555531C71C8A	

hex 35555531C71C8A trunc. in last bit = 1AAAAA98E38E45 = $0.8\overline{3}$ decimal

Table 2.5: Example of radix-512 division.

with initial value $w[0] = x - 1$ and $S[0] = 1.0$. The quantity $S[j]$ is the partial result and s_{j+1} is the result digit chosen at the j-th iteration by the selection function

$$s_{j+1} = SEL(S[j]_\delta, \lfloor rw[j] \rfloor_t) \ .$$

The condition for convergence is

$$-2\rho S[j] + \rho^2 r^{-j} < w[j] < 2\rho S[j] + \rho^2 r^{-j} \ .$$

In general, it is not possible to have single selection function for all values of j. For a more accurate description, refer to [10].

By comparing expression (2.5) with expression (2.1), the term inside ( ) in expression (2.5) substitutes $q_{j+1}d$ in expression (2.1). Because of these similarities in the recurrence, it is convenient to implement division and square root in the same unit as discussed next.

2.7 Combined Division and Square Root Algorithm

Because of the similarities in the algorithm, division and square root can be effectively implemented in the same unit [31, 10, 34]. The combined division and square root, described in detail in [10], is implemented by the residual recurrence

$$w[j+1] = rw[j] + F[j] \qquad j = 0, 1, \ldots, m \tag{2.6}$$

in which

$$F[j] = \begin{cases} -q_{j+1}d & \text{(division)} \\ -(S[j]s_{j+1} + \frac{1}{2}\, r^{-(j+1)} s_{j+1}^2) & \text{(square root)} \end{cases} \tag{2.7}$$

Since the partial result is initialized to $Q[0] = 1.0$ and $S[0] = 1.0$,

$$w[0] = \begin{cases} x - d & \text{(division)} \\ \frac{1}{2}(x - 1) & \text{(square root)} \end{cases}$$

where x is the dividend/radicand, and d the divisor. Both d and x are normalized in $[0.5, 1)$ and $x < d$ for division, while x is normalized in $[0.25, 1)$ for square root. The result digit (q_{j+1} for division and s_{j+1} for square root) are determined, at each iteration, by a selection function

$$\begin{aligned} q_{j+1} &= SELC(d_\delta, \hat{y}) && \text{(division)} \\ s_{j+1} &= SELC(\hat{S}[j], \hat{y}) && \text{(square root)} \end{aligned}$$

where d_δ and $\hat{S}[j]$ are respectively d and $S[j]$ truncated after δ fractional bits, and $\hat{y}$ is an estimate of $rw[j]$. The result digit is in signed-digit representation and the residual $w[j]$ is stored in carry-save representation (w_S and w_C) to reduce the iteration time. In order to use $S[j]$ in the iterations, we need to convert the result digits from signed-digit to conventional representation. The on-the-fly conversion algorithm is used to perform this conversion. In the on-the-fly conversion, two variables A and B are required. They are updated, in every iteration, as follows:

$$A[j] = S[j] \qquad \text{and} \qquad B[j] = S[j] - r^{-j}$$

The recurrence is implemented, as shown in Figure 2.5 with the selection function (SEL), the block to form F (FGEN), a block (DSMUX) which provides FGEN with the appropriate bit vectors[2] (depending on the operation selected by signal OP), a carry-save adder (CSA), and two registers to store the carry-save representation of the residual. The conversion block performs the conversion from signed-digit to conventional representation and the rounding. The result is rounded in the last iteration according to the sign of the final residual and the signal that detects if it is zero, which are produced by the sign-zero-detection block (SZD).

[2]In special cases, such as for radix-2 and 4, one bit vector is sufficient.

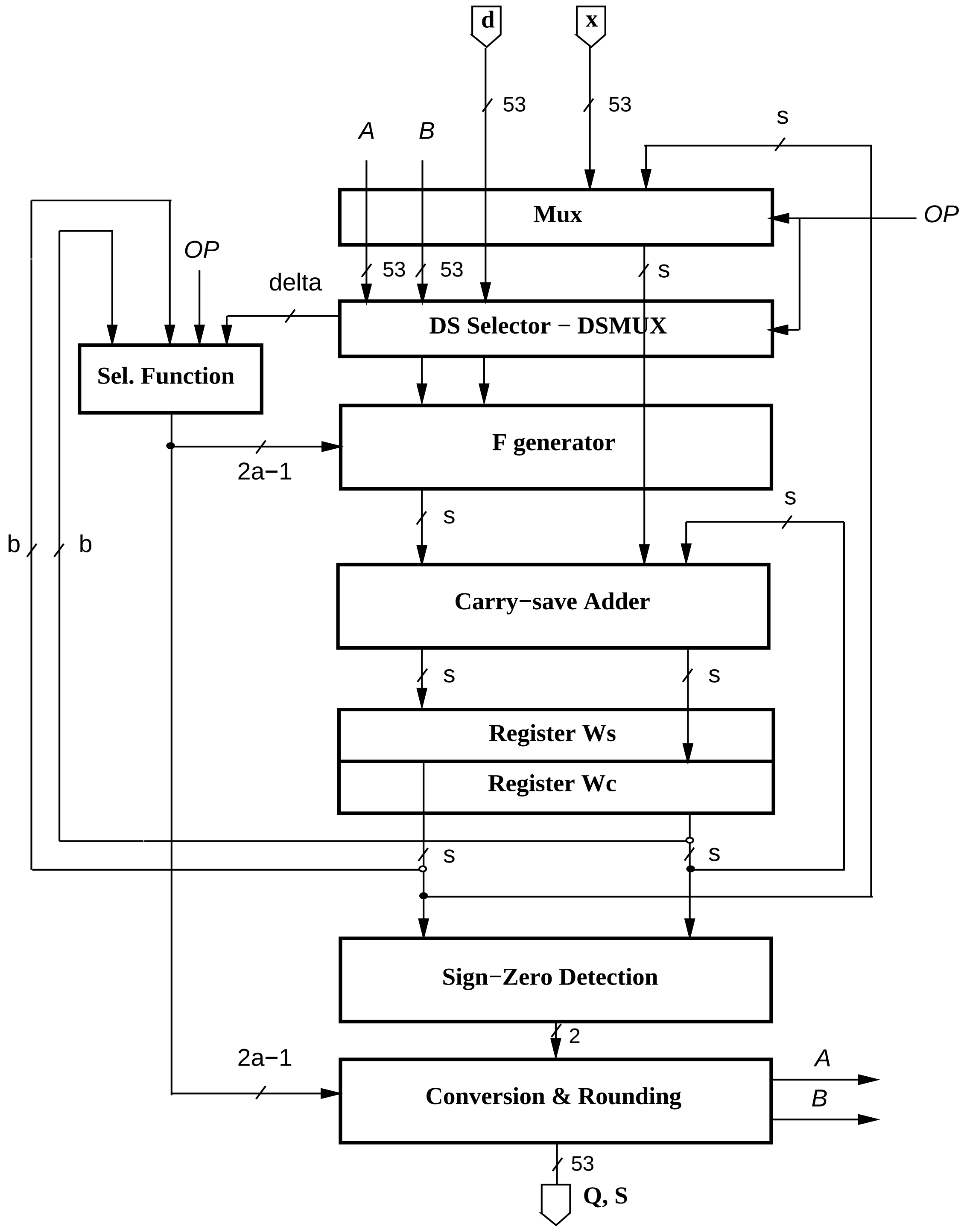


Figure 2.5: Combined division/square root unit.

Chapter 3
Techniques to Reduce Energy Dissipation

Introduction

In this chapter we describe the techniques used to reduce the energy consumption. The radix-4 divider is presented in this chapter to establish a "standard" implementation of the digit-recurrence algorithm and better explain the application of energy reduction techniques to the unit. More detail on the implementation of the blocks is given in Appendix A.

3.1 Radix-4 Division Algorithm and Basic Implementation

For radix-4 the recurrence is

$$w[j+1] = 4w[j] - q_{j+1}d \qquad j = 0, 1, \ldots, 28$$

with the initial value $w[0] = x$ and with the quotient-digit selection

$$q_{j+1} = SEL(d_4, \hat{y}) \qquad q_j = \{-2, -1, 0, 1, 2\}$$

where d_4 is d truncated after the 4-th fractional bit, but only 3 bits are needed for the selection, being the most-significant bit (MSB) 1 because d is normalized. The estimated residual, $\hat{y} = 4w_{S4} + 4w_{C4}$, is truncated after 4 fractional bits and with the 3 integer bits gives a total of 7 bits required by the selection function. The selection function for radix-4 and $\rho = \frac{2}{3}$ is shown in Table 3.1. The quotient digit h selected is the one satisfying the expression $m_h \leq \hat{y} < m_{h+1}$. To have the

m_h	d_δ							
	8/16	9/16	10/16	11/16	12/16	13/16	14/16	15/16
m_2	12	14	15	16	18	20	20	22
m_1	4	4	4	4	6	6	8	8
m_0	-4	-6	-6	-6	-8	-8	-8	-8
m_{-1}	-13	-15	-16	-18	-20	-20	-22	-24

Values in table are multiplied by 16

Table 3.1: Selection function for radix-4 division.

divider compliant with IEEE standard for double-precision while operating with fractional values, 1-bit shifts are performed on the operands. Furthermore, to have a bounded residual in the first iteration (expression (2.3) with $w[0] = x$), we shift x one position to the right obtaining 54 bits for the representation of its mantissa. Since it is convenient to have the extra bit required for the rounding produced in the last position of the last digit, we shift x by an extra position to the right, obtaining a total of 55 fractional bits and 1 sign bit for the recurrence ($w[j]$). Each division requires 28 cycles to compute the quotient digits plus one cycle to initialize the recurrence and one cycle to perform the rounding.

The block diagram of the basic radix-4 divider is shown in Figure 3.1. The datapath shown in Figure 3.1 is completed by a controller and by a tree to distribute the clock signal (not depicted in the figure). The critical path, shown in Figure 3.2, is $7.0ns$. The energy dissipation of the unit is shown in the first column of Table 4.1 at page 80. The largest part of the energy is consumed in the registers and in the convert-and-round unit.

3.2 Classification of Techniques

In our approach to the reduction of the energy dissipated in the division or square root unit, we consider two main portions: the recurrence and the conversion and

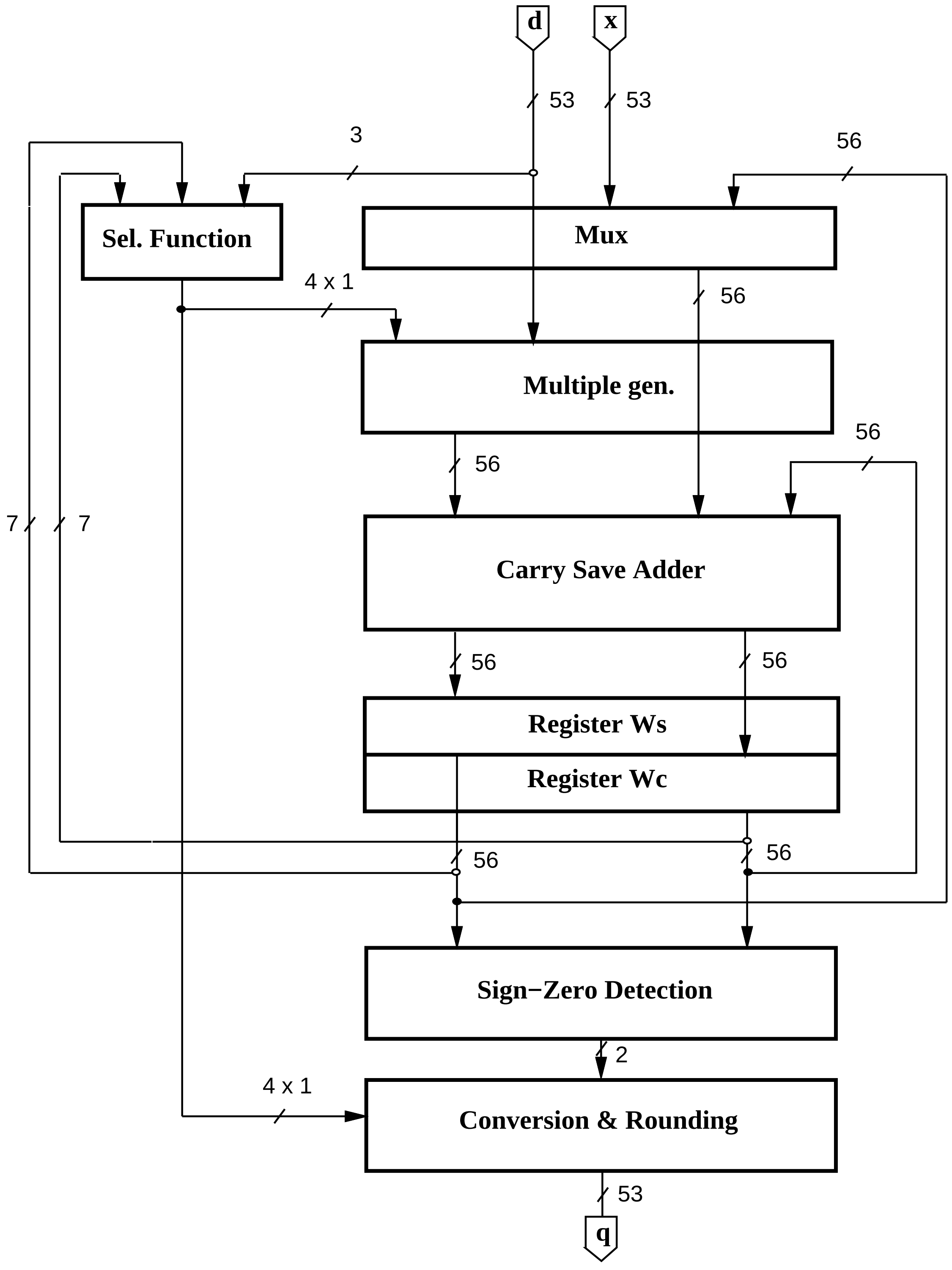


Figure 3.1: Implementation of radix-4 divider.

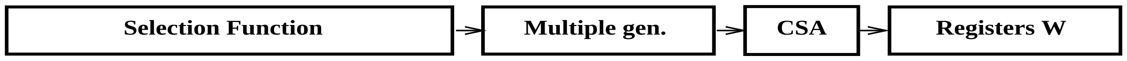


Figure 3.2: Critical path for radix-4 implementation in Figure 3.1.

rounding. The following techniques are applied to the recurrence part:

- retiming the recurrence
- changing the redundant representation to reduce the number of flip-flops in the registers
- using gates with lower drive capability for gates not in the critical path
- applying dual voltage to portions of the circuit not in the critical path
- equalizing the paths to reduce glitches
- partitioning and disabling the selection function
- glitch filtering and suppression.

For the conversion and rounding part, the following techniques are applied:

- on-the-fly conversion algorithm modification
- disabling the clock in not changing flip-flops
- gating the trees to distribute signals
- applying dual voltage.

In addition we switch off not active blocks, when possible.

Most of the techniques described above do not alter the critical path and therefore do not increase the execution time of the operation performed. However, the following techniques affect the critical path:

- partitioning and disabling the selection function
- glitch filtering and suppression.

For those techniques, tradeoffs between delay and energy consumption are considered.

3.3 Retiming the Recurrence

The position of the registers in a sequential system affects the energy dissipation. Retiming is the circuit transformation that consists in re-positioning the registers in a sequential circuit without modifying its external behavior [20]. By retiming the recurrence we reduce the number of spurious transitions, reduce the switching activity in some blocks, and change the critical path. The retiming is done by moving the selection function from the first part of the cycle to the last part of the previous cycle (see Figure 3.3). We have to introduce a new register to store the quotient digit, but the register q_j is quite small, a few bits, and it does not compromise the energy saving obtained by retiming.

Since now the quotient digit is stored in a register, this has the effect of reducing the glitches in the multiple generator and in the carry-save adder.

After the retiming, the critical path is limited to a few most-significant bits in the recurrence. Since the path through the least-significant bits of the multiple generator and the CSA does not include the selection function (Figure 3.4), these bits can be redesigned for low-power, as discussed in the next sections.

As shown in Table 3.2, the retiming does not increase the number of cycles needed to complete the operation.

Furthermore, by eliminating buffering for the few most-significant bits in the critical path in MULT, we can reduce the critical path (Figure 3.5).

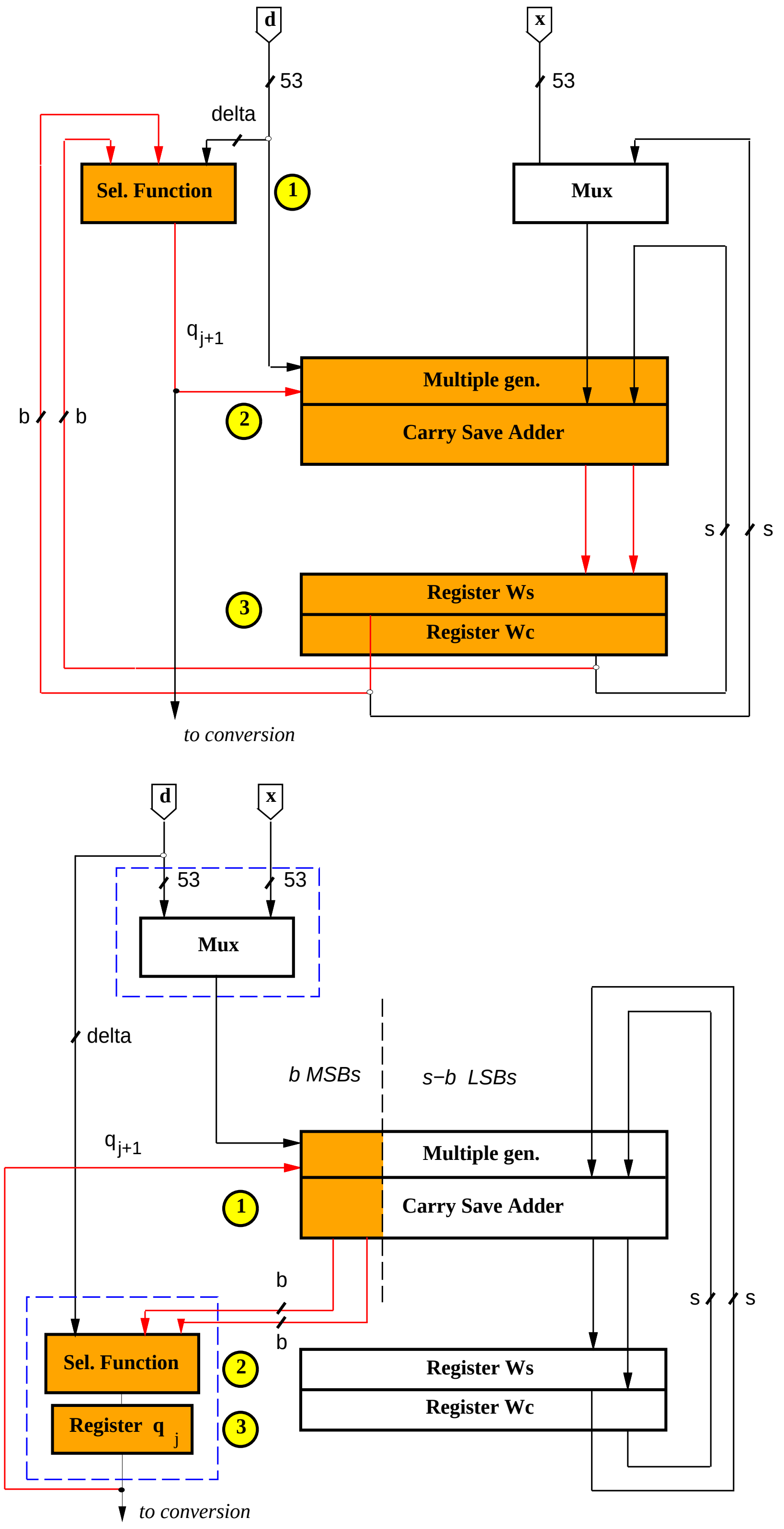


Figure 3.3: Retiming of recurrence.

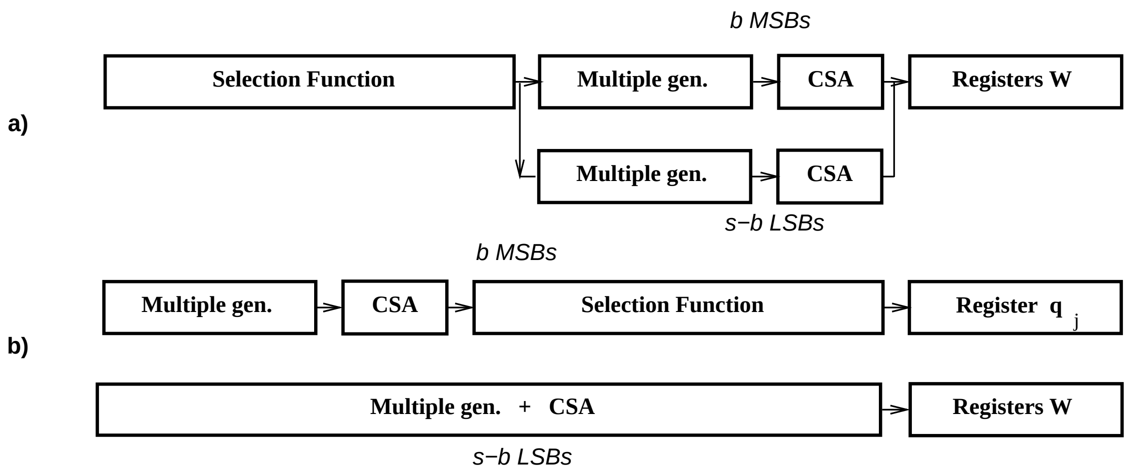


Figure 3.4: Change in the critical path. Before **a)** and after **b)** retiming.

j	cycle	before retiming	after retiming
0	1	$w[0] = x$	$w[0] = x$ $q_1 = SEL(d_\delta, r\hat{x})$
1	2	$q_1 = SEL(d_\delta, r\hat{x})$ $w[1] = rw[0] - q_1 d$	$w[1] = rw[0] - q_1 d$ $q_2 = SEL(d_\delta, r\hat{w}[1])$
		$\ldots\ldots$	
$j+1$	$j+2$	$q_{j+1} = SEL(d_\delta, r\hat{w}[j])$ $w[j+1] = rw[j] - q_{j+1} d$	$w[j+1] = rw[j] - q_{j+1} d$ $q_{j+2} = SEL(d_\delta, r\hat{w}[j+1])$
		$\ldots\ldots$	
m	$m+1$	$q_{m+1} = SEL(d_\delta, r\hat{w}[m])$ $w[m+1] = rw[m] - q_{m+1} d$	$w[m+1] = rw[m] - q_{m+1} d$ $q_{m+2} =$ not used

Table 3.2: Retiming does not increase number of cycles.

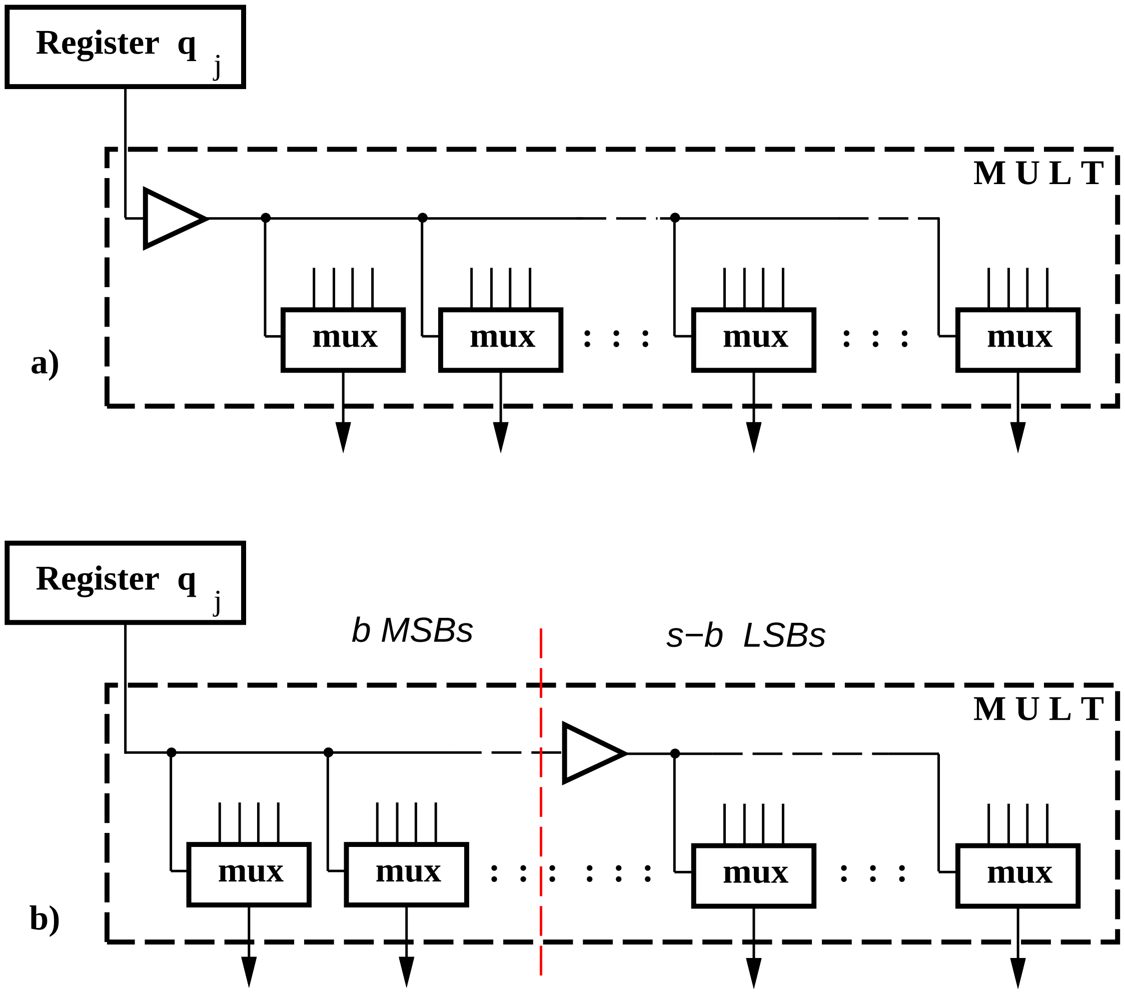


Figure 3.5: Removing buffers from MSBs. **a)** before, **b)** after.

3.3.1 Reducing the Transitions in the Multiplexer

In this modified unit, the retiming allows the re-positioning of the multiplexer out of the recurrence (Figure 3.3). In the first iteration the input x of the multiplexer is selected, while the input d is selected in the remaining iterations. The operations in the first cycle are modified by resetting register q_j to 1 and allowing the input x to be stored in registers W as the first residual $w[0] = 1 \cdot x$.

The multiplexer is now in the critical path because it provides the value of either x or d to the multiple generator, which inputs are otherwise connected to

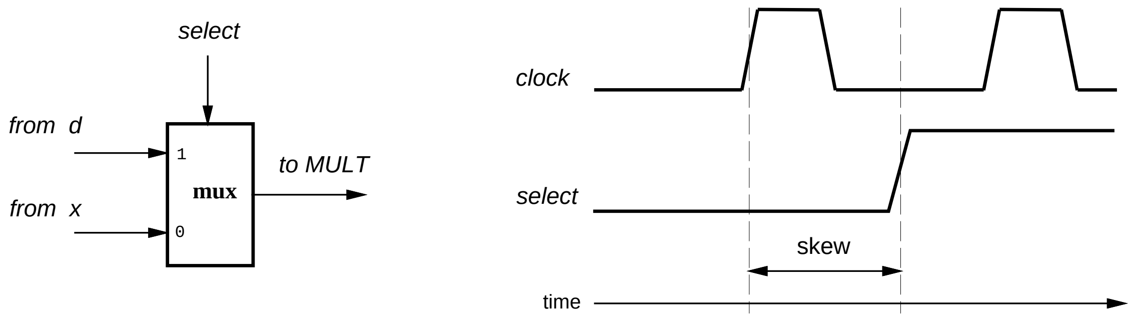


Figure 3.6: Skewing of the *select* signal.

the registers. However, because the output of the multiplexer is changed once per division, its delay can be masked by earlier switching. In fact, the mux-select is the only signal sent from the controller to the recurrence and it can be skewed (anticipated) at the end of the first cycle masking the delay of the multiplexer. The mux-select signal can be skewed by adding the appropriate delay (e.g. some buffers) in the distribution tree as shown in Figure 3.6.

3.4 Changing the Redundant Representation

Since the contribution of flip-flops to both energy dissipation and area is significant, it is useful to change the redundant representation of the residual (w_S and w_C) to reduce the number of flip-flops in the registers. By using a radix-r carry-save representation with $log_2 r$ sum bits and one carry bit for each digit, we can reduce the number of flip-flops. With this modification we only need to store one carry bit for each digit, instead of $log_2 r$.

The change in the redundant representation requires a redesign of the carry-save adder to propagate the carry inside the digit (Figure 3.7). In Figure 3.7, each radix-2 CSA (left in figure) is actually a full-adder (FA) implemented with two half-adders (HA). The propagation of the carry increases the delay so that this modification cannot be made for those cells (digits of w) that are in the critical

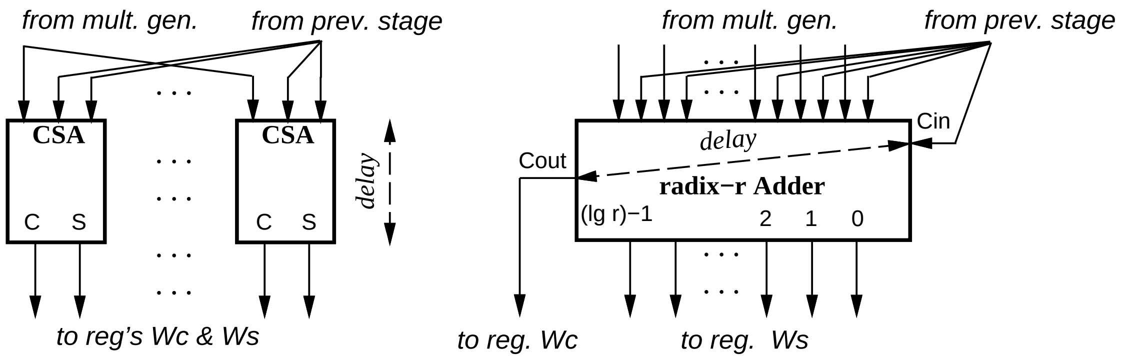


Figure 3.7: Replacing CSAs with radix-r CSAs.

path. After the recurrence has been retimed, the critical path is limited to the b MSBs. The difference between the paths through the MSBs and the LSBs is

MSBs:	MULT	HA	SEL	REG
LSBs:	MULT	HA		REG

For the LSBs in the recurrence we can redesign the CSA into a radix-r carry-save adder (r-CSA) that satisfies the following condition on delays:

$$t_{r-CSA} \leq t_{HA} + t_{SEL} \ .$$

Furthermore, because the b MSBs of the residual are assimilated in the selection function, in the retimed scheme these bits can be stored in register w_S and the corresponding b flip-flops in register w_C eliminated (Figure 3.8).

3.5 Using Gates with Lower Drive Capability

Another reduction in the energy dissipation is achieved by minimizing the energy in the gates not in the critical path by using cells with lower drive capability. In the retimed recurrence, this is done for the least-significant bits (not in the critical path) of the multiple generator and the carry-save adder (Figure 3.9).

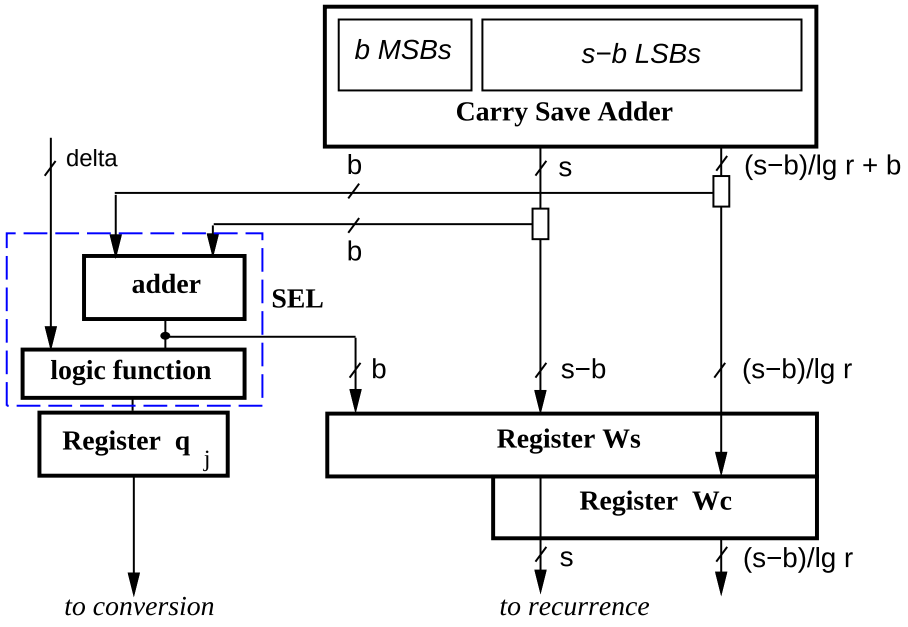


Figure 3.8: b MSBs assimilated in selection function.

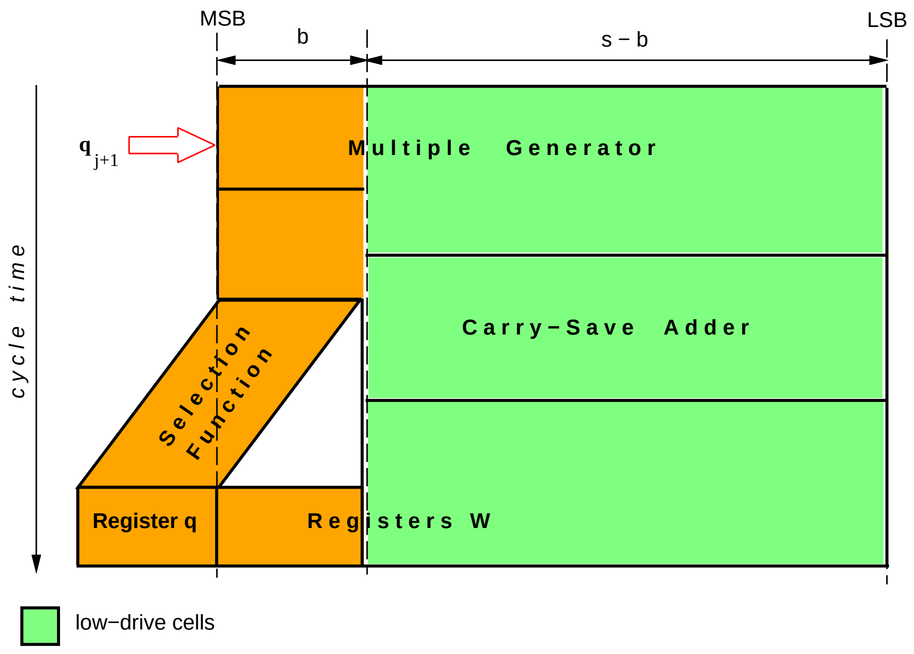


Figure 3.9: Low-drive cells in the recurrence.

3.6 Dual Voltage

The energy dissipated in a cell depends on the square of the voltage supply (V_{DD}) and a significant amount of energy can be saved by reducing it [14]. However, by lowering the voltage the delay increases, so that to maintain the performance this technique is applied only to cells not in the critical path. Different power supply voltages require level-shifting circuitry that contribute to the total energy dissipation. As a consequence, it is convenient to apply this technique only if the number of cells not in the critical path is quite large, and the energy increase in the level-shifting circuitry does not offset the reduction due to voltage scaling. However, by using two voltages we only need to level-shift when going from the lower to the higher voltage [35]. A more complete description of the level shifter for dual voltage is presented in Appendix A.

In the case of the divider, as shown in Figure 3.10, the $s - b$ least-significant bits, can be redesigned for low-voltage. The voltage-level shifters are not needed until a specific digit moves towards the b-MSBs, by shifting across iterations, and into the critical path. By placing the voltage-level shifters in the digit immediately before the b-MSBs the cycle time is not increased. In order to evaluate the possible lower voltage V_2 to be used in a dual voltage implementation we need to determine the time slack available for the LSBs. The time slack is the difference between the delay in the paths through the MSBs and LSBs, and it gives the amount of time available for the delay of gates whose voltage is scaled to V_2.

The delay of the least-significant portion depends on the type of CSA adder used, since the delay of the radix-r CSA is larger than that of the radix-2 CSA. Since the reduced voltage can be lower for radix-2 CSA, this might result in a reduction of the total energy. There is a tradeoff between the following:

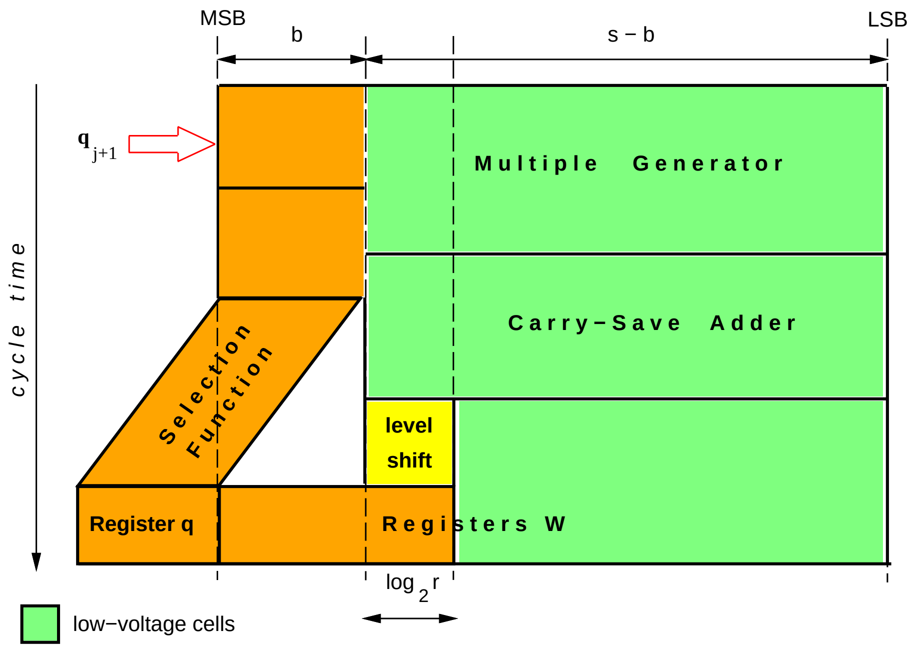
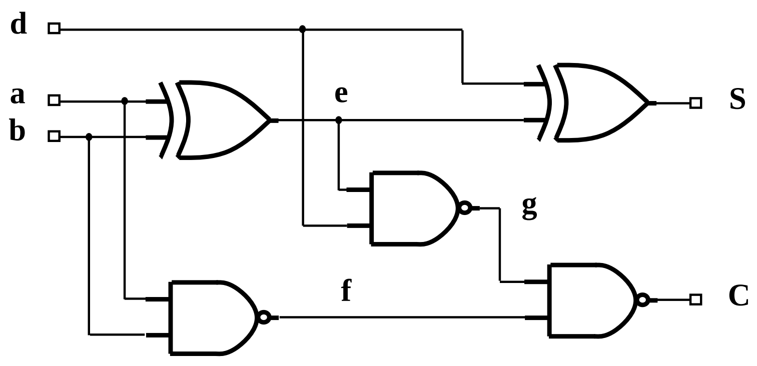


Figure 3.10: Low-voltage cells in the recurrence.

- The voltage can be lower for radix-2 CSA
- There is a reduction in the number of flip-flops by using the radix-r CSA.

3.7 Equalizing the Paths to Reduce Glitches

By equalizing the paths of the input signals of the blocks we reduce the generation of glitches [16]. Because of different delays, both gate and interconnection delay, the input signals to the carry-save adder (CSA) arrive at different times, creating spurious transitions inside the adders. For instance, Figure 3.11 shows, in the upper part, a possible implementation of one of the full-adders composing the carry-save adder. Pins a and b are directly connected to the registers, whereas pin d is connected to the output of the multiple generator. If the input signals a and b arrive at different times, glitches might be produced in e and f. Also, if there is a difference between the arrival times of d and e, glitches might be produced in S, g and C.



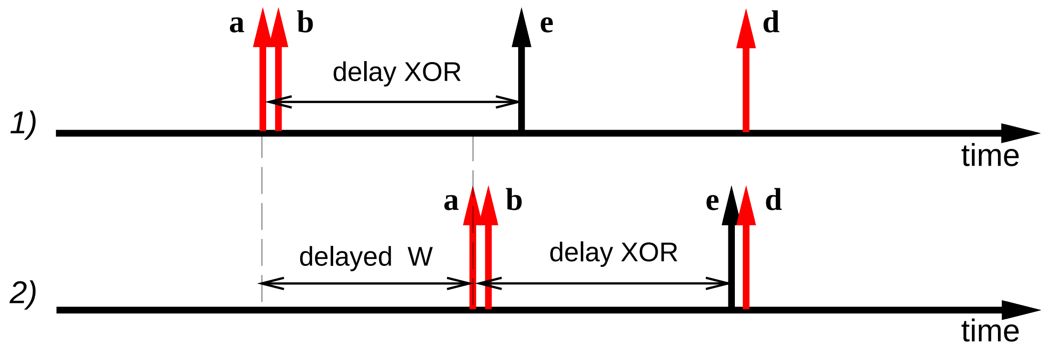


Figure 3.11: Equalizing paths in CSA.

Time diagram 1) in Figure 3.11 shows an example of the distribution of the arrival times for signal a, b, d, and e. In order to eliminate the spurious transitions, we delay the clock to the Ws and Wc registers (which produce a and b) so that the signals e and d, overlap, as shown in time diagram 2) in Figure 3.11. However, it is impossible to eliminate all the glitches because due to the different delays of the XOR and NAND gates, signals at nodes f and g always arrive at different times.

3.8 Partitioning and Disabling the Selection Function

The quotient-digit selection is a function of a few bits of the divisor and of the residual. Since the divisor is fixed for the whole division operation, from the point of view of energy consumption it is convenient to decompose the function into

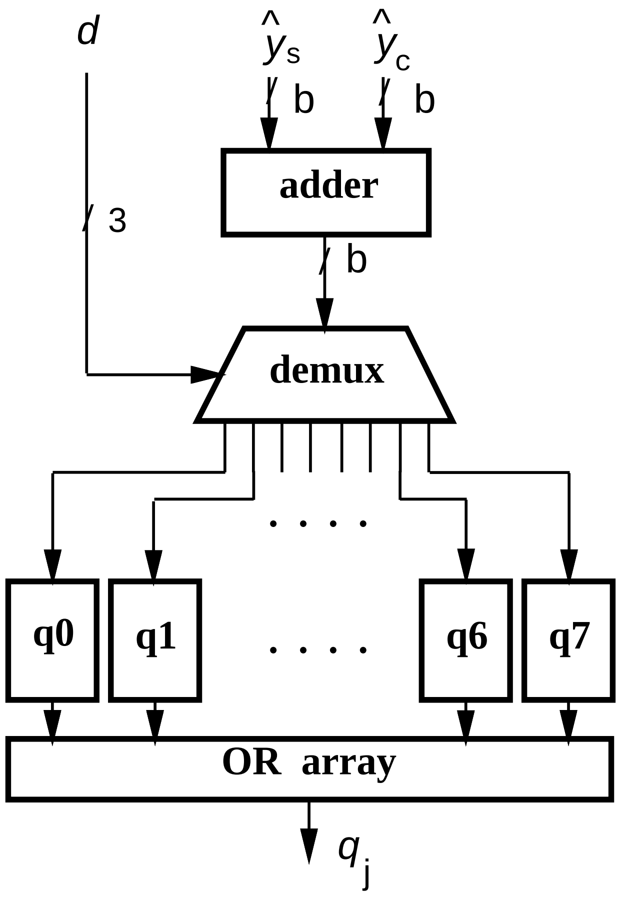


Figure 3.12: Partitioned selection function.

subfunctions and to enable only the subfunction corresponding to the actual value of the divisor. This is specially convenient for higher radices, because the quotient-digit selection is more complex and therefore is responsible for a significant portion of the energy.

Figure 3.12 shows an example ($\delta = 3$ bits of the divisor are required) in which the selection function is partitioned in $2^\delta = 8$ parts (all the possible values of d_3). The demultiplexer transmits the assimilated value of $\hat{y}$ to the selected pair of selection tables and forces to zero the output of the others. Finally an array of OR gates concentrates again the value of the quotient-digit.

Experimental results showed that the partioned selection function dissipates less

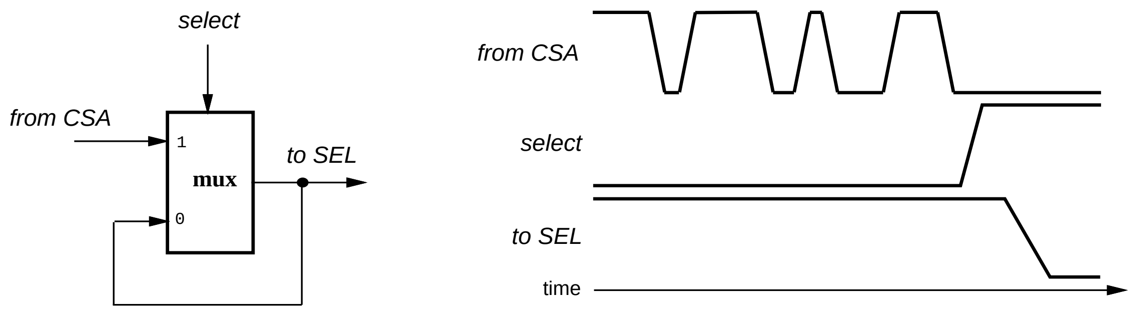


Figure 3.13: Glitch suppression using multiplexers.

energy, but because of the demux and the OR gates the critical path increased.

3.9 Glitch Filtering and Suppression

In the retimed implementation, the selection function is connected to the output of the carry-save adder, instead of directly after the registers (Figure 3.3). As a consequence of its repositioning, there is an increase in the number of glitches in the selection function. One way to filter those glitches is to buffer the selection function with multiplexers acting as latches, as described in Figure 3.13. The select signal is driven by a different clock (same period, different phase) that enables the multiplexers to transmit the value from the CSA when it is stable, and hold the current value otherwise. However, in this case the delay of the multiplexer affects the critical path. More precisely, the additional delay in the critical path is due to two contributions:

1. the intrinsic delay of the multiplexer (from input to output),
2. the delay of the *select* signal with respect to the time the output of the CSA is stable.

This second contribution can be eliminated by triggering the *select* signal before the output of the CSA is stable. However, in this case, some glitches might not be suppressed.

3.10 Reductions in Conversion and Rounding

3.10.1 On-the-fly Conversion Algorithm Modification

In the conversion and rounding part of the divider, we both modified the algorithm and applied gate-level energy reduction techniques.

We now describe the modifications in the conversion algorithm for the two cases: $a < r - 1$ and $a = r - 1$.

When $a < r-1$, in the original algorithm, two registers (n bits each) are needed to store Q and QM. The registers are filled with digits starting from the least-significant position and then shifted towards the most-significant position. The large number of flip-flops used and the shifting result in a large energy consumption in the convert-and-round unit.

As a first step to reduce the energy dissipated, we load each digit in its final position. In this way we avoid the need to shift digits along the registers. To determine the load position we use an m-bit ring counter. The algorithm starts the computation from the most-significant digit. In iteration j

- If $q_j \geq 0$ then load q_j in Q and $q_j - 1$ in QM, both in position $i = m - j$.
- If $q_j < 0$ then load $r- \mid q_j \mid$ in Q and $r- \mid q_j \mid -1$ in QM, both in position i.

As a second step, we eliminate register QM. When $q_j < 0$ it is necessary to propagate a borrow. In the original algorithm, QM is used to avoid this propagation. Instead of the register, to propagate this borrow (without actually doing the subtraction) the digits which change because of this propagation are marked. These digits correspond to the last sequence of zeros plus the last nonzero digit before this sequence. These are marked by the same ring counter by keeping a 1 for those digits that might be changed by a borrow.

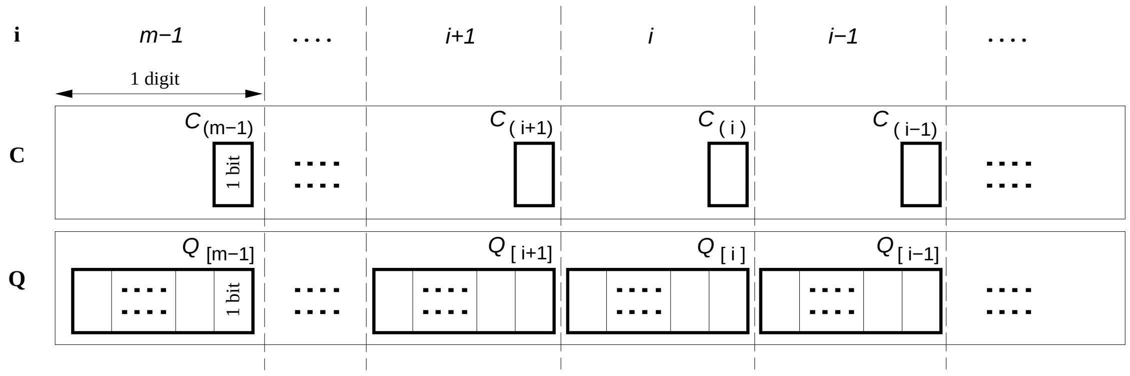


Figure 3.14: Registers C and Q in the new converter.

If $q_j > 0$	then	$Q_{[i]} \Leftarrow q_j$ AND set $C_{(i-1)} \Leftarrow 1$ AND reset all other bits in C
If $q_j = 0$	then	$Q_{[i]} \Leftarrow 0$ AND set $C_{(i-1)} \Leftarrow 1$ /∗ no resetting in C ∗/
If $q_j < 0$	then	$Q_{[i]} \Leftarrow r- \mid q_j \mid$ AND $\left\{ \begin{array}{lll} Q_{[k]} \Leftarrow Q_{[k]} & \text{if } C_{(k-1)} = 0 & k = i+1, \ldots, m \\ Q_{[k]} \Leftarrow [Q_{[k]} - 1]_{mod\ r} & \text{if } C_{(k-1)} = 1 & k = i+1, \ldots, m \end{array} \right.$ AND set $C_{(i-1)} \Leftarrow 1$ AND reset all other bits in C

Table 3.3: Modified algorithm.

We refer with $Q_{[i]}$ to the digit position in the register Q and with $C_{(i)}$ to the bit position in the ring counter (Figure 3.14). The modified algorithm is shown in Table 3.3.

The updating expression for the ring counter is

$$C_{(i)} \Leftarrow C_{(i+1)}\ \overline{C_{(i)}} + Z\ C_{(i)} \tag{3.1}$$

where $Z = 1$ if $q_j = 0$.

Table 3.4 shows how the conversion is modified, for the example presented in Table 2.1.

In the final rounding, the last digit is loaded with p_t as in expression (2.4). If the last digit is negative the update (to propagate the borrow) is done as in the other iterations. The only exception is when $q_m = -1$, and by rounding it $p = 0$ is

j	q_j	Q	C
1	1	1xxxxxxxx	010000000
2	2	12xxxxxxx	001000000
3	0	120xxxxxx	001100000
4	-1	1133xxxxx	000010000
5	0	11330xxxx	000011000
6	0	113300xxx	000011100
7	2	1133002xx	000000010
8	-2	11330012x	000000001
9	-1	113300113	100000000

Table 3.4: Example of radix-4 modified conversion.

obtained. In this case, the register Q is not updated.

When $a = r - 1$ three registers Q, QM and QP are needed for the conversion and rounding (Section 2.2). The register QP is eliminated by recoding the quotient digit into the digit set $\{-(r-1), \ldots, -1, 0, 1, \ldots, r-2\}$. The value $r-1$ is recoded into -1 and the previous digit incremented by one. This recoding requires to store the current quotient digit in a temporary register T ($\log_2 r$ bits + 1 sign bit) as sketched in Figure 3.15. No additional cycle is needed since the conversion of the last digit is done together with the rounding. Table 3.5 shows an example of conversion, for radix-8 and $a = 7$, using this recoding.

With the implementation of the new algorithm we reduce the number of flip-flops in the convert-and-round unit from $2n$ to $(1 + \frac{1}{\log_2 r})n$ when $a < r - 1$ and from $3n$ to $(1 + \frac{1}{\log_2 r})n + \log_2 r + 1$ when $a = r - 1$.

Summarizing, the algorithm is modified by eliminating the shifting of the digits previously loaded and by replacing registers QM and QP with two additional, but smaller registers: C, which is introduced to keep track of the digits to update, and T, which is used for the recoding.

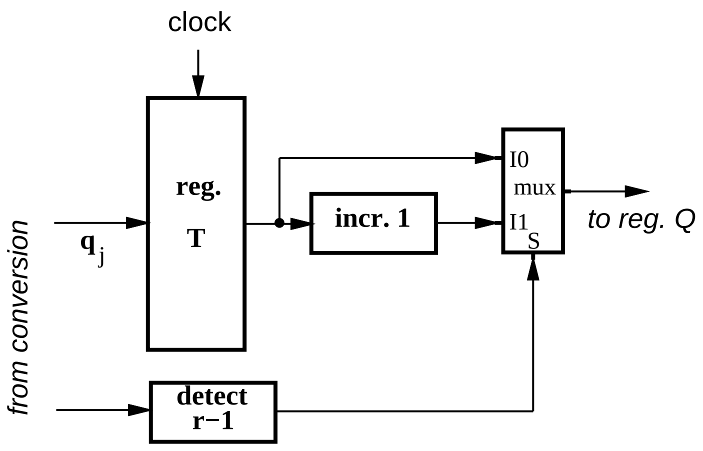


Figure 3.15: Use of register T.

q_j	T	Q	C
4	4	xxxxxxxx	10000000
7	-1	5xxxxxxx	01000000
-1	-1	47xxxxxx	00100000
7	-1	470xxxxx	00110000
0	0	4677xxxx	00001000
7	-1	46771xxx	00000100
7	-1	467710xx	00000110
7	-1	4677100x	00000111
Final rounding			
add 1	x	46771000	00000111
add 0	x	46770777	00000111

Table 3.5: Example of radix-8 recoding.

3.10.2 Disabling the Clock

As a further step to reduce the energy dissipation in the convert-and-round unit, we switch off the clock signal for the flip-flops in the register that do not have to be updated. Figure 3.16 shows an application of the gated flip-flop technique [19]. We introduce the activation function F, that enables the clock of the flip-flop only when it is needed. As described in [19], F must be ANDed with the clock signal (clk) for trailing-edge-triggered flip-flops. For leading-edge-triggered (rising edge) flip-flops an AND gate cannot be used, to avoid a malfunctioning of the circuit if the delay (d) of F is shorter than the period the clock is high (h), as shown in Figure 3.16.a ($d < h$). By making the flip-flop clock signal

$$cp = \overline{F} + clk \tag{3.2}$$

we obtain the desired result for leading-edge-triggered flip-flops (Figure 3.16.b). Note that the problem is still present if F changes when clk is low, but in the case of the converter the delay d is shorter than the clock pulse width h.

With this technique, in the ring counter (refer to Figure 3.14) the clock of flip-flop $C_{(k)}$ is enabled when

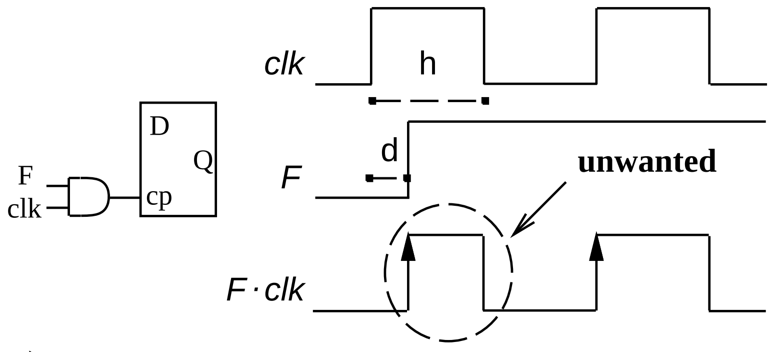
- the normal update of the ring counter that occurs when $C_{(k+1)} = 1$ and $C_{(k)} = 0$.
- the reset which occurs when $C_{(k)} = 1$ and $Z = 0$.

The resulting enabling function is

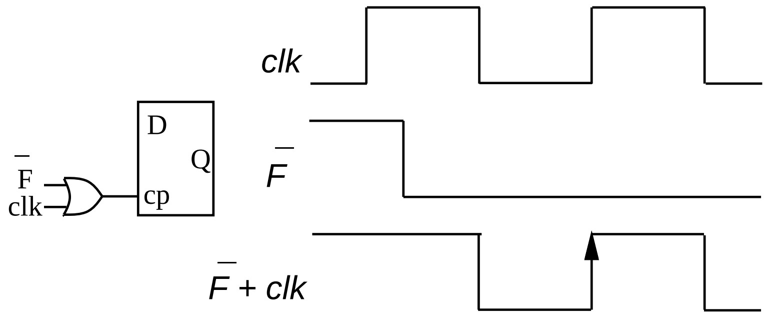
$$F_{(k)} = C_{(k+1)}\ \overline{C_{(k)}} + C_{(k)}\ \overline{Z}$$

By De Morgan's theorem we can write expression (3.2) as

$$cp = \overline{F} + clk = \overline{F\ \overline{clk}}$$



a)



b)

Figure 3.16: Gated flip-flop enabling function.

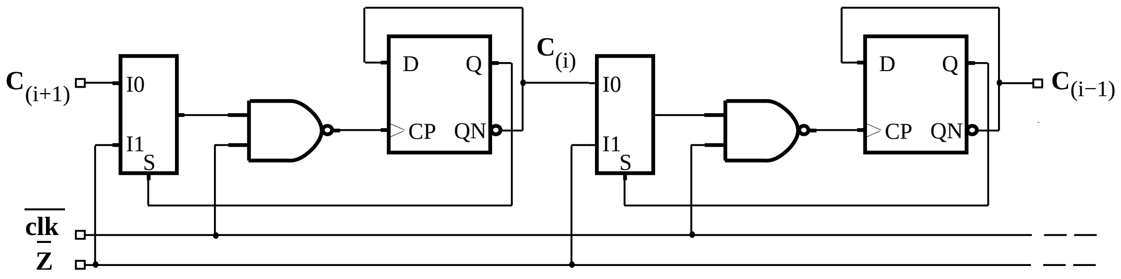


Figure 3.17: Two consecutive bits in the ring counter.

and substituting F we get the expression for the clock of the k-th flip-flop

$$cp_{C(k)} = \overline{(\ C_{(k+1)}\ \overline{C_{(k)}} + C_{(k)}\ \overline{Z}\ )\ \overline{clk}}$$

Because of the selective activation of the clock, the updating for $C_{(k)}$ is reduced from expression (3.1) to

$$C_{(k)} \Leftarrow \overline{C_{(k)}}$$

Figure 3.17 shows the implementation for two consecutive flip-flops.

In register Q, the current digit is presented to the input of all the flip-flops, but only the digits modified at that iteration are loaded, by enabling the clock signal of the corresponding flip-flops. If at start-up register Q is initialized to 0, no load is needed when q_{j+1} is zero. The clock signal of digit $Q_{[i]}$ is enabled when

- the current digit $q_j \neq 0$ and it must be loaded in the i-th position. In this situation $C_{(i)} = 1$, $C_{(i-1)} = 0$, and $Z = 0$.
- the current digit $q_j < 0$ ($q^{SIGN} = 1$) and the borrow must be propagated. In this case, all the digits whose corresponding bit $C_{(k-1)} = 1$, for $k > i$, must be updated.

The enabling function $E_{[k]}$ is

$$E_{[k]} = \overline{P_{[k]}}\ \overline{Z} + C_{(k-1)}\ q^{SIGN} \qquad \text{with } \overline{P_{[k]}} = C_{(k)}\ \overline{C_{(k-1)}}$$

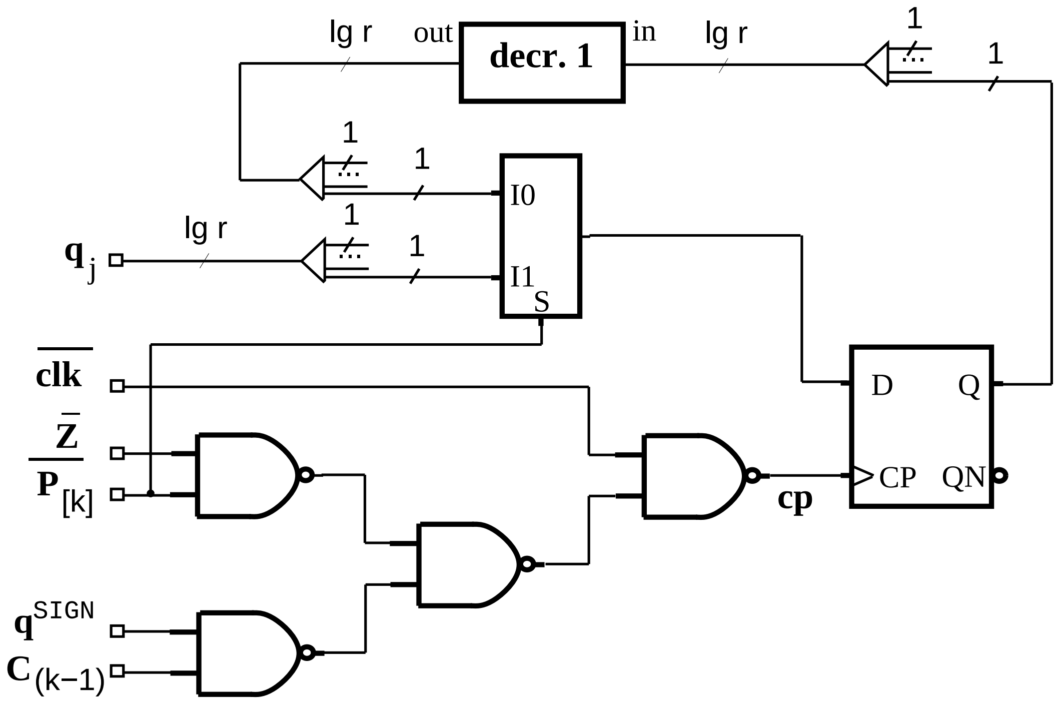


Figure 3.18: Clock enabling function and loading in register Q.

the expression for the clock of the k-th digit of Q is

$$cp_{Q[k]} = \overline{\overline{(\ C_{(k-1)}\ q^{SIGN} + \overline{P_{[k]}}\ \overline{Z}\ )}\ \overline{clk}} \tag{3.3}$$

and the value to be loaded is

$$Q_{[k]} \Leftarrow \begin{cases} q_j & \text{if } P_{[k]} = 0 \\ Q_{[k]} - 1 \pmod r & \text{if } P_{[k]} = 1 \end{cases}$$

Its implementation is shown in Figure 3.18.

3.10.3 Gating the Trees

The modified conversion algorithm requires that the converted quotient digit be presented to the full array of flip-flops in register Q, and then, only $\log_2 r$ of them are loaded with this digit. To distribute the digit and the clock we need a tree (Figure 3.19.a) that dissipates a significant amount of energy. Because of the particular structure of the algorithm, by dividing the register Q into two portions, upper ($m/2$ most-significant digits) and lower ($m/2$ least-significant digits), we can switch-off a part of the tree for half the number of the iterations. This is obtained by dividing the tree into two halves and by propagating the signal to the upper array of flip-flops when executing the first $m/2$ iterations and to the lower array in the rest as shown in Figure 3.19.b. Signals AU and $AL = \overline{AU}$ select the half array to feed and g represent a generic bit to be loaded in the flip-flops. To keep track of which part of the array is computed we use an additional flip-flop that is set after the $m/2$-th iteration. By implementing this gated-tree we can save about 50% of the energy dissipated to distribute the signals (the gates introduce extra capacitance, and also the number of transitions are not equally distributed in the two portions of the array).

On the other hand, when computing the digits in the lower portion we might need to update the digits in the upper array, for this reason we cannot switch-off the clock (for example) for the upper part. But the clock can be disabled for the lower part in the first $m/2$ iterations. In this case the reduction is about 25% (Figure 3.19.c).

As a further refinement, we can switch-off the clock and digit-sign in the upper part after a digit different from 0 has shown up in the lower array. This requires an additional flip-flop to mark the state "second part of the array and digit different from zero occurred".

3.10.4 Dual Voltage

Because the convert-and-round unit is not in the critical path, we can use low-voltage cells to realize it. The number of level-shifter required is n: one for each bit of the final quotient that must be raised from the lower voltage to the higher (V_{DD}). Note that in the last cycle of the division, when the result is produced, the larger delay of the low-voltage flip-flops will produce q at a later time than in the

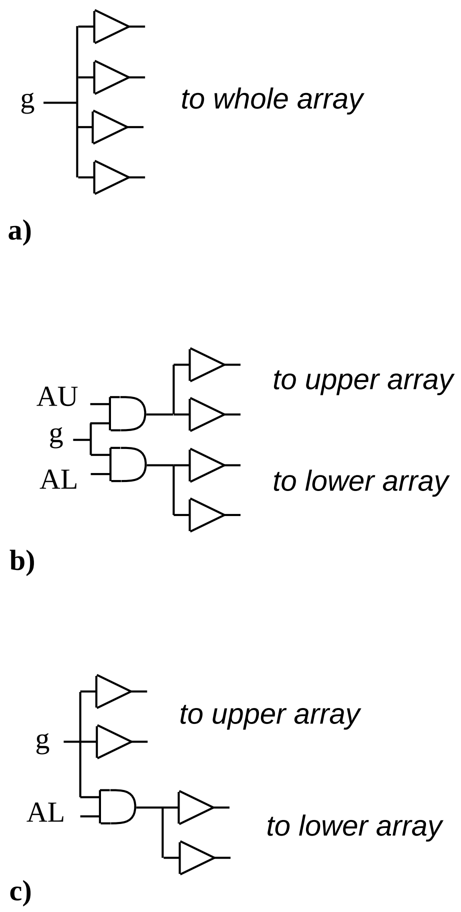


Figure 3.19: Gated tree. a) before, b) 50% reduction, c) 25% reduction.

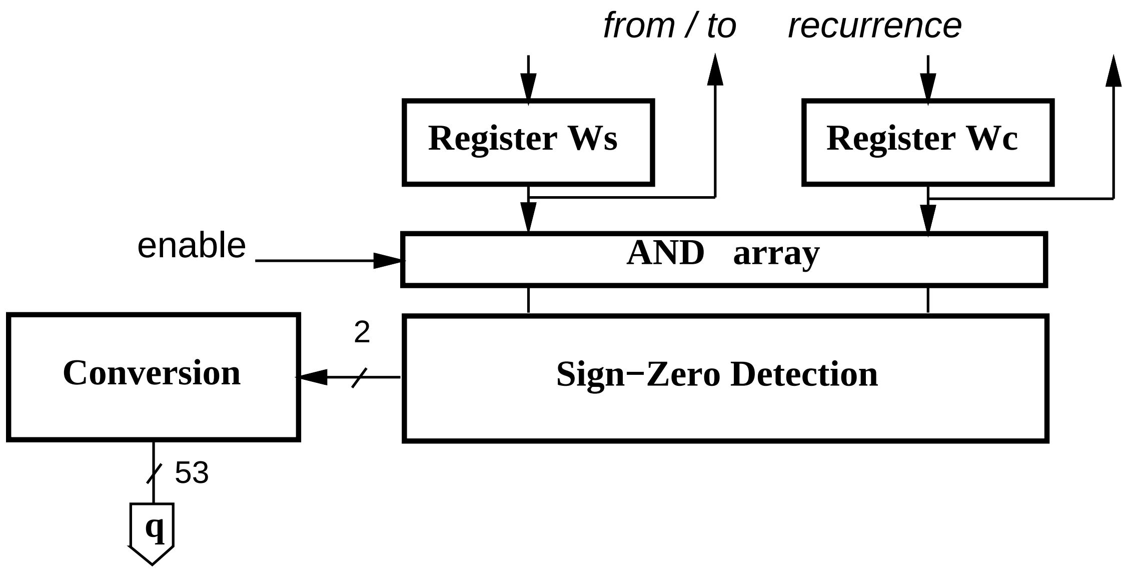


Figure 3.20: Disabling SZD during recurrence iterations.

non-reduced voltage implementation.

3.11 Switching-off Not Active Blocks

The modification consists in switching-off blocks which are not active during several cycles. This is the case for the sign-zero-detection block (SZD), which is only used in the rounding step to determine the sign of the final remainder and if it is zero. The SZD can be switched off by forcing a constant logic value at its inputs during the recurrence steps (Figure 3.20).

3.12 Optimization by Synthesis for Low-Power

Logic synthesis provides the automatic synthesis of gate-level netlists, optimizing the design for various constraints. The solution of an optimization problem is measured in terms of a cost function [36]. The cost measures the extent to which a constraint has been met. If the constraint has been satisfied, the corresponding cost is zero. A different priority is given to the constraints, for example:

1. maximum delay
2. maximum energy dissipation
3. maximum area.

This means that timing constraints will not be violated to save power, but available time slack will be used to reduce it. A transformation is accepted if it decreases one of the cost functions, without increasing higher priority costs.

In order to minimize the energy, or power dissipation, determined either by probabilistic estimation algorithms or by gate-level simulation, circuit transformations that try to reduce one of the main factors contributing to the energy consumption (gate capacitance, net switching activity, net transition times and net capacitive loading) are applied to the design.

In our case, we used Synopsys Power Compiler, which performs synthesis at gate-level with optimization capability for power dissipation. The main features of the tool, derived from [15], are briefly discussed in Appendix B Section B.3. The synthesis is performed on relatively small blocks as explained in Chapter 4.

Chapter 4
Implementations

Introduction

The techniques presented in Chapter 3 are applied to double-precision division/ square root units, which implement the algorithms described in Chapter 2. First, we give an overview of the design flow and the tools and the libraries of standard cells used. Then, we present the implementations of division for radix-4, 8, 16, and 512, and the implementation of a radix-4 combined division and square root unit. For each scheme, we provide the energy consumption for the basic, or standard, and low-power implementations and an estimate of a possible implementation with dual-voltage and by optimizing some blocks with Synopsys Power Compiler. In the presentation of the units, we highlight the differences from the implementation of the radix-4 divider, set as the reference. However, for sake of clarity and completeness, some repetitions of concepts and figures occur. Detail of the implementation of blocks, which are common to many units, is given in Appendix A.

4.1 Design Flow, Tools and Libraries

4.1.1 Design Flow and Tools

The most convenient way of describing the units under investigation is to use a hardware description language, in this case VHDL which allows the description and simulation of the system at different level of abstraction and the use of hierarchy. The design flow we used is depicted in Figure 4.1. The behavioral and RT-level are handled by Synopsys Tools [37]. Synopsys provides a number of tools to generate, maintain and simulate a VHDL description of the circuit. The inter-

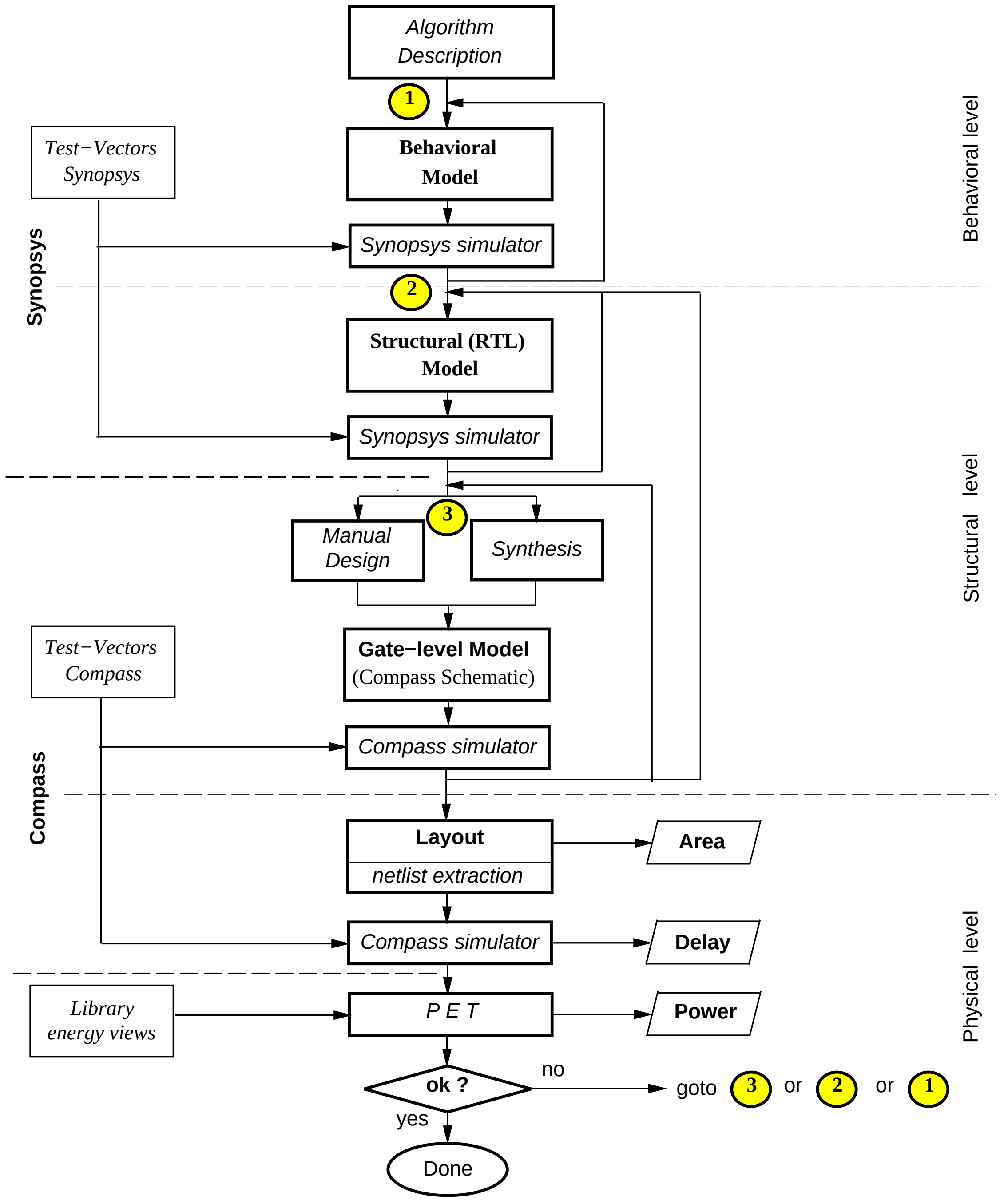


Figure 4.1: Design flow and tools.

face between the RT-level and the physical level is handled by COMPASS Tools [38]. COMPASS provides ASICSynthesizer a logic synthesizer that maps the VHDL behavioral description of a block into gates. However, ASICSynthesizer performs synthesis by optimizing only delay and area. COMPASS also provides an automatic floor-planner for the layout generation and a simulator at gate-level (*Qsim*), for the simulation of pre-layout and layout-extracted netlists. The design can be divided into the following steps (or levels):

Behavioral level A behavioral model of the divider was developed from the algorithm. Using Synopsys, some simulations were carried out on this model to test the functionality and the correctness of the results.

RT-level The unit was manually divided into functional blocks. Each block represents a different functionality of the system. A block could be either a combinational or a sequential circuit, and a controller was introduced in order to have the correct sequencing of the operations. Then, part of these functional blocks were expanded into sub-blocks containing logic functions, adders, multiplexers and registers.

Gate-level The VHDL description of the RTL-model, obtained with Synopsys, was imported into the COMPASS environment for the physical design and the layout generation. The gate netlists of each block were generated either by COMPASS ASICSynthesizer (relatively small and irregular blocks) or by manual design (large and regular blocks).

Physical level The layout was generated (cell placement and routing) in a totally automatic way and the netlist of the whole unit, including the interconnection capacitance, was extracted from the layout.

In addition, synthesis using Synopsys Power Compiler was performed. As explained later in Section 4.2, the results of the synthesis of large blocks are not completely satisfactory. For this reason, we limit the synthesis with Power Compiler to the selection function, which is a small and irregular block. First the design with the shortest delay is synthesized, and then, incrementally, a new compilation is done to optimize the design for power dissipation trying not to increase the delay.

As explained in Section 1.5, in order to compute the energy dissipated in a circuit, information on the capacitance (layout) and on the circuit activity (simulation or statistics) are required. This computation is done by PET: Power Evaluation Tool (Appendix B Section B.1), which computes the energy dissipated in a circuit from the layout-extracted netlist, the standard cell library characteristics, and the results of a logic-level simulation run on a given set of test vectors.

The average energy/power dissipation can be determined by applying random-generated input patterns (test vectors) and monitoring the energy dissipated using a simulator. This approach belongs to the Monte Carlo methods [39]. Monte Carlo simulations give an accurate estimate of the expected value with a limited number of trials (test vectors) [40].

The estimation error, derived from [41], for a normal distribution of the energy values can be written as:

$$\frac{\mid E_{op} - \eta \mid}{\eta} = \frac{t\ s}{\eta \sqrt{N}}\ . \tag{4.1}$$

where E_{op} is the expected value of the average energy dissipation, η and s are measured average and standard deviation of the N random samples of energy, and t is obtained from the t-distribution with $(N-1)$ degrees of freedom [41]. Consequently, the percentage error ϵ, in a given confidence level $(1-\alpha) \times 100\%$, is

$$\epsilon = \frac{t_\alpha s}{\eta \sqrt{N}} \tag{4.2}$$

The same approach to estimate the total average power dissipation on a set of benchmark circuits is presented in [42]. For those benchmark circuits, simulations on about 10 random vectors are sufficient to have an estimation error smaller than 5%. Moreover, according to [42], the validity of expression (4.2) can be extended to any distribution for small values of s.

At the end of the chapter, in Section 4.7 at page 133 we summarize the error obtained for the estimation of the energy dissipated in the units presented in this work.

4.1.2 Standard Cell Libraries

The units were realized using the Passport $0.6\mu m$, 3.3 V, three-metal layers, standard cell library [43] and the layout was obtained by automatic floor-planning. The percent reductions in the energy dissipation indicated below might vary for different technologies and layout styles. The critical path, unless otherwise specified, is computed post-layout and takes into account the RC-effect of interconnections.

The Passport library was designed to operate with $V_{DD} = 3.3\ V$ and COMPASS tools cannot implement more than one supply voltage. In order to evaluate the application of dual voltage, we performed SPICE simulations on a 4-bit carry-ripple adder to determine the dependency of the delay with respect to V_{DD} (Figure 4.2). The delay is normalized to the one for $V_{DD} = 3.3\ V$. The plot shows that for $V_{DD} = 2.0\ V$ the delay is doubled, and that for voltages below 1.7 V the delay increases in excess.

The energy consumption for dual voltage was estimated on a block basis, by using the following expression:

$$E_{d-v} = E_{VDD}\left[\frac{b}{s} + \left(\frac{V_2}{V_{DD}}\right)^2 \left(1 - \frac{b}{s}\right)\right] \tag{4.3}$$

where E_{VDD} is the energy dissipated in the block when the power is supplied by

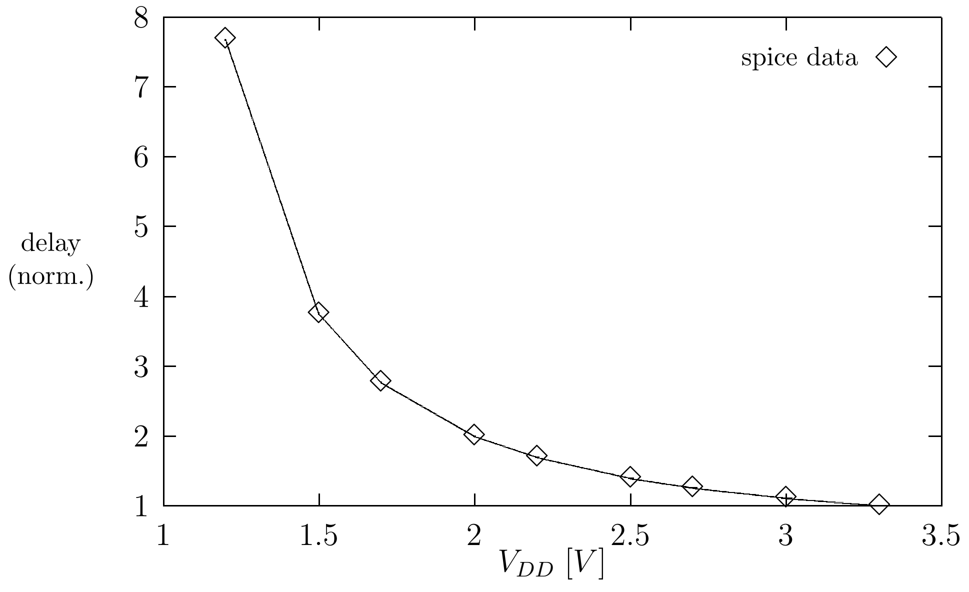


Figure 4.2: Delay (normalized) with different V_{DD}.

V_{DD} only, b are the bits in the block not to be scaled and s is the total number of bits (refer to Figure 3.10 in page 48). Expression (4.3) is based on the following assumptions:

1. the number of transitions are uniformly distributed from the MSB to the LSB,
2. no variations in neither load capacitance nor activity due to the scaling.

The first assumption was verified by counting the actual number of transitions detected by the logic simulator at the input of the blocks in question, while SPICE simulations on a 4-bit slice of the recurrence showed that the second assumption leads to an over-estimation because the value provided by expression (4.3) is about 10% larger than the actual energy dissipation for values of V_2 from 3.3 V to 2.0 V.

The library of standard cells used in Synopsys Power Compiler is different from the one used in COMPASS. This is due to the fact that the Passport library, used in COMPASS, is not characterized, both timing and power, for Synopsys. The library used in Synopsys is the ST CB45000 Standard Cell, 0.35 μm 5 layer metal HCMOS6 process, with power supply voltage of 2.7 V [44].

Databook comparisons and testing on small circuits showed that the CB45000 library at 2.7 V is about 33% faster than the Passport library at 3.3 V.

4.1.3 Presentation of Results

For each of the units below, we present four implementations. The first implementation is the one obtained with the only constraint of minimum delay. This implementation is also indicated as standard and abbreviated *std* in the tables. The second implementation is the low-power implementation obtained by applying the techniques described in Chapter 3. This implementation is indicated as *l-p* in the tables. With our library and tools it is not possible to realize layouts which use dual voltage (Section 3.6). For this reason we can provide just estimates of dual voltage implementations, which are abbreviated *d-v* in the tables. Estimates of the energy dissipation after to optimization with Synopsys Power Compiler are indicated as *syn* in the tables.

4.2 Radix-4 Division

The techniques presented in Chapter 3 are applied to the case of a double-precision radix-4 division unit, which is typical of those found in many floating-point processors.

4.2.1 Algorithm and Basic Implementation

The algorithm and the basic implementation of the radix-4 division has been already presented in Section 3.1.

We indicate with *std* the implementation of the basic radix-4 divider shown in Figure 3.1 at page 38. The critical path, shown in Figure 4.3, is 7.0ns. It

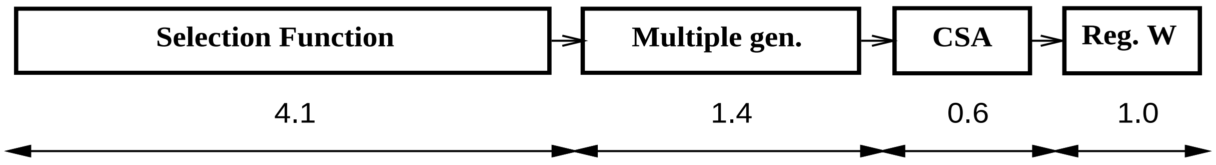


Figure 4.3: Critical path in ns.

is computed post-layout and takes into account the RC-effect of interconnections. This first implementation, optimized for minimum delay, has the energy dissipation characteristics shown for *std* in Table 4.1 in page 80 at the end of this section. The largest part of the energy is consumed in the registers and in the convert-and-round unit.

4.2.2 Low-Power Implementation

Retiming the recurrence

The retiming is done by moving the selection function from the first part of the cycle to the last part of the previous cycle (Figure 4.4). The reduction in the number of transitions in the recurrence for the retimed implementation is 15% with respect to the *std*.

The critical path is now limited to the 8 most-significant bits, so that the 48 least-significant can be redesigned for lower power dissipation by changing the redundant representation of the residual, using low-drive gates and dual voltage. Note that, although only 7 bits are required for the selection function, since the representation is in carry-save form, the eighth bit in the recurrence produces the least-significant carry to go in the selection function.

Furthermore, by eliminating buffering for the 8 most-significant bits in the critical path in MULT, we can reduce the critical path (see Figure 3.5 at page 43). However, the load connected to the output of register q_{j+1} is larger (320%) and the delay in the register is increased by about 30% reducing the benefits of this modi-

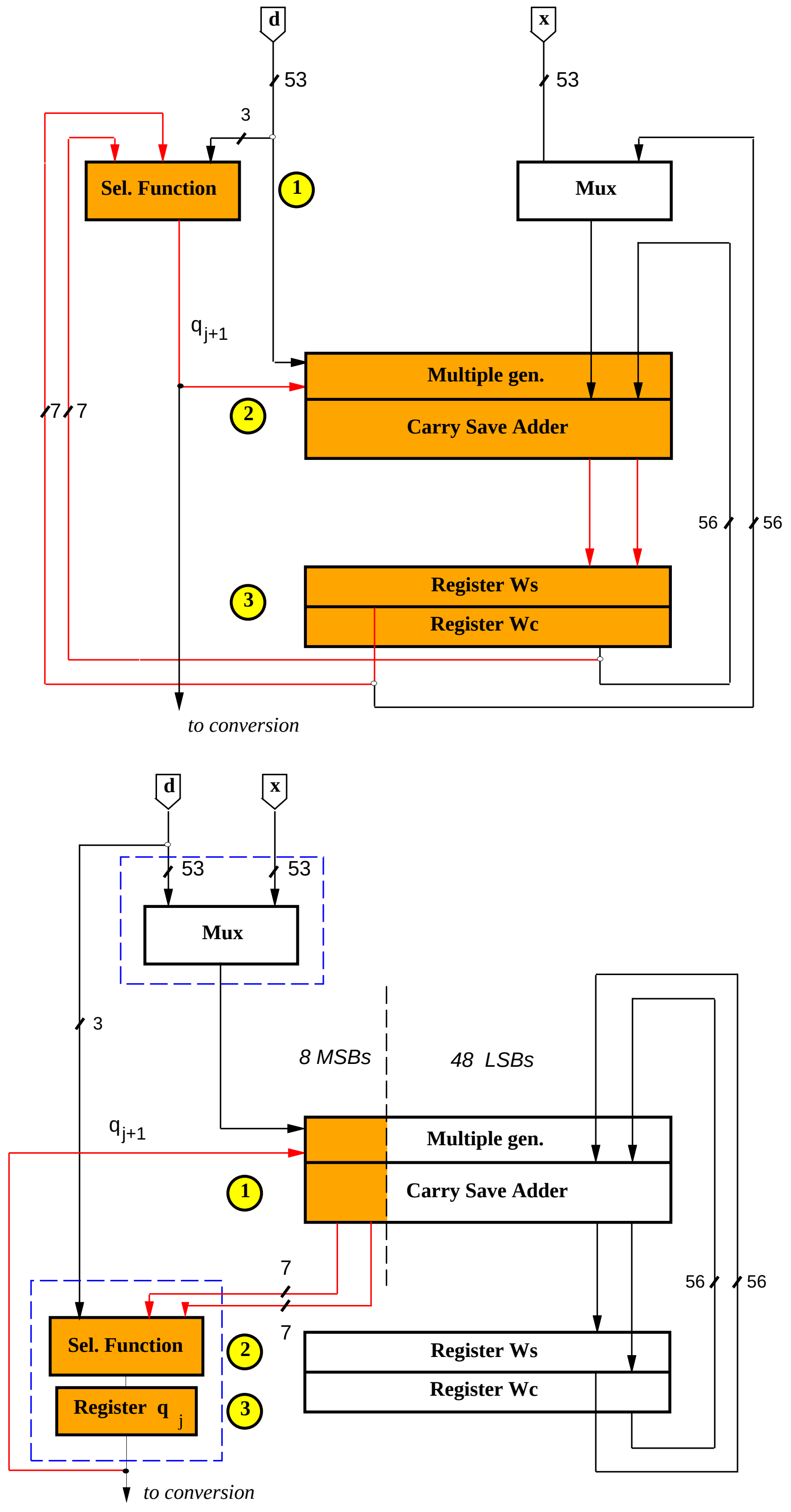


Figure 4.4: Retiming of recurrence.

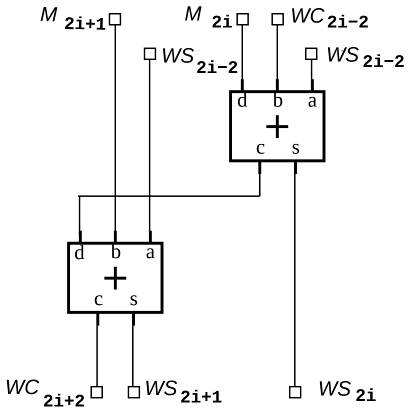


Figure 4.5: Radix-4 implementation in the carry-save adder.

fication. The overall improvement in delay is 0.3 ns corresponding to less than 5% of the critical path.

After the retiming, the multiplexer can be moved out of the recurrence.

Changing the redundant representation to reduce the number of flip-flops

The change in the redundant representation is done using a radix-4 carry-save representation with two sum and one carry flip-flops for each two bits (Figure 4.5). Since this requires a redesign of the carry-save adder to propagate the carry of the even bit-slice to the next bit-slice, in order not to increase the critical path this is done only in the 48 least-significant bits of $w[j]$. This modification results in a reduction of 25% in the number of flip-flops for the bits not in the critical path. Figure 4.6 shows that the 7 MSBs of the carry-save representation of $w[j+1]$ are assimilated in **qds adder**, and by storing the assimilated value for these 7 bits, we can eliminate the corresponding flip-flops in register Wc. The number of flip-flops in register Wc decreases from 56 to 25.

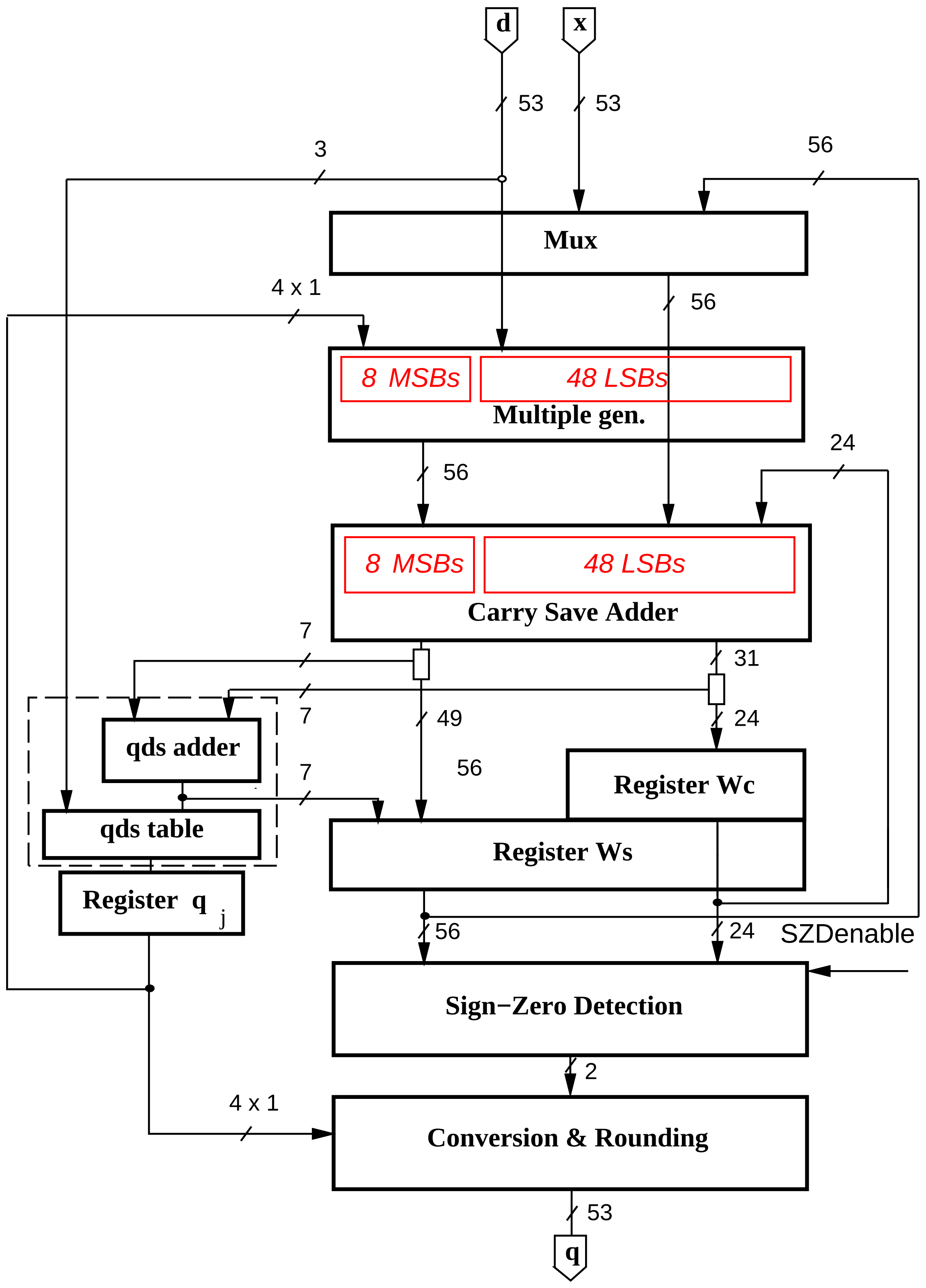


Figure 4.6: Block diagram of *l*-*p* unit.

Using low-drive gates and equalizing the paths

In the retimed recurrence, we can use lower drive capability gates for the 48 least-significant bits (LSBs) of the multiple generator and the carry-save adder.

By equalizing the paths of the input signals of the blocks we reduce the generation of glitches. The equalization is done by delaying the clock to registers Ws and Wc, as previously explained in Figure 3.11 at page 49.

The use of automatic floor-planning in the placement and routing of standard cells limits the control on the interconnection delay, and the difference in the delays generates glitches. Therefore, the reduction of the spurious transitions is quite small, and this reflects on the energy dissipation that is reduced by less than 5%.

The combination of these techniques results in implementation *rec.* The actual reduction in the recurrence is about 20% with respect to *std* (Table 4.1 in page 80).

Reductions in the SZD unit

As mentioned in Section 3.11, the SZD is only used in the rounding step and it can be switched off by forcing a constant logic value at its inputs during the recurrence steps.

Reductions in the convert-and-round unit

The total energy dissipated in the convert-and-round unit is 30% of *rec.*

In the implementation of the modified algorithm (Figure 4.7), described in Section 3.10, we obtained a reduction of the energy dissipation for the convert-and-round unit of about 55%, However, more than 50% of the total energy in the unit was dissipated in the trees to distribute the clock, and the other signals to the array of flip-flops. By implementing gated-trees we obtained a reduction of about 65% in the block.

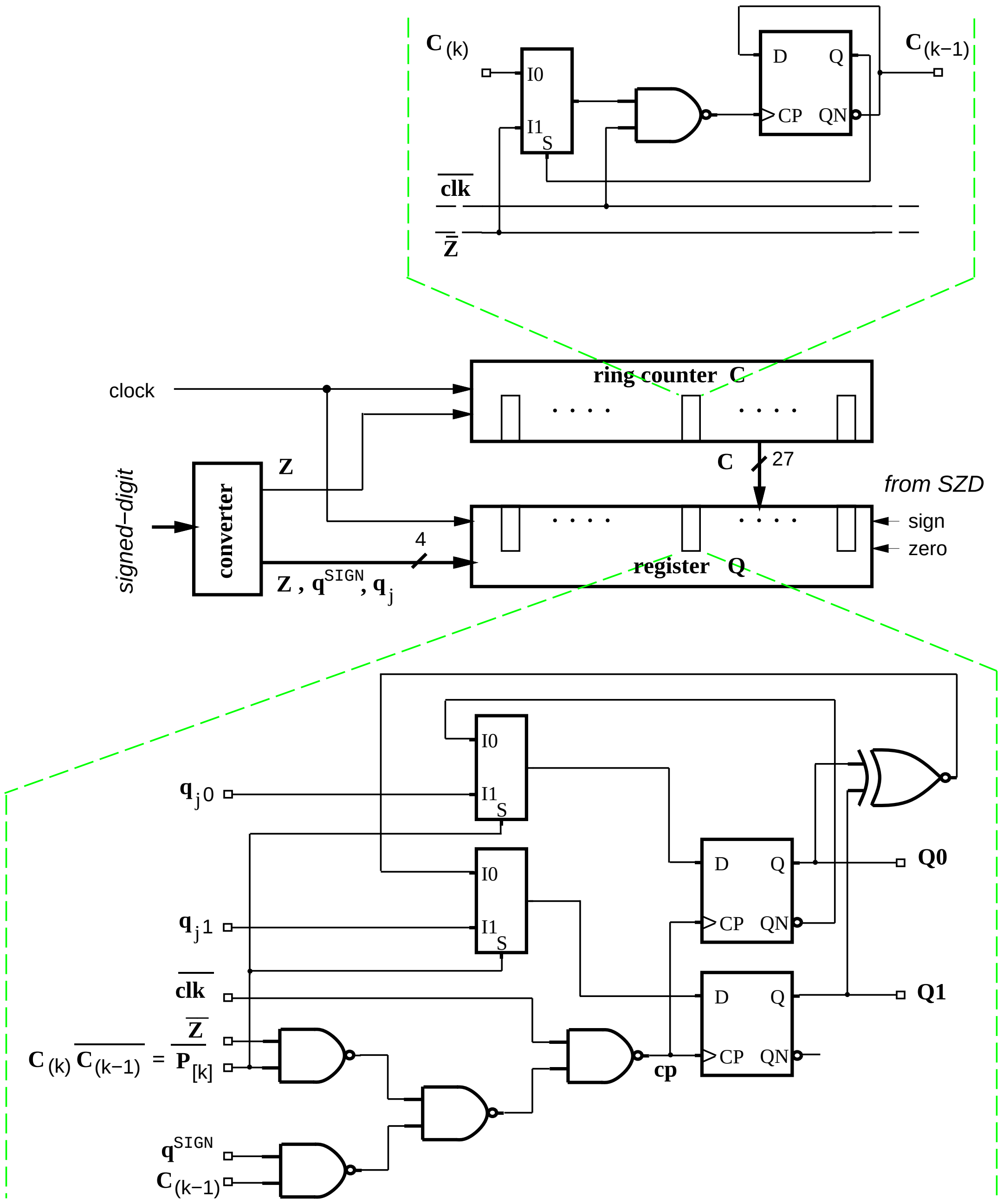


Figure 4.7: Convert-and-round unit for radix-4 divider.

This final implementation of the convert-and-round unit and its integration in the whole divider corresponds to *l-p*. With respect to the basic implementation *std* we reduced the energy dissipation by 40% (Table 4.1 in page 80).

4.2.3 Dual Voltage Implementation

In order to evaluate the possible lower voltage V_2 to be used in a dual voltage implementation we need to determine the time slack available for the LSBs in the recurrence. The delay of the least-significant portion depends on the type of CSA adder used, since the delay of the radix-4 CSA is larger than that of the radix-2 CSA. By implementing the LSBs of the recurrence with radix-2 CSAs, the delay in the LSBs is 3.1 ns, resulting in a time slack of 3.9 ns. In this case $V_2 = 2.0\ V$ can be chosen without affecting the latency of the divider. On the other hand, by opting for the use of radix-4 CSAs, the time slack is reduced to 3.0 ns and, consequently, V_2 can be lowered to 2.2 V. The same estimated values for E_{div} are obtained by applying expression (4.3), so that the radix-4 CSA solution might be preferred because of the smaller area. Only two level-shifters (low to high) are needed (Figure 3.10, page 48).

In the convert-and-round unit, unlike in the case of the recurrence, the number of required level shifters is quite high (53), but each bit can switch at most twice. Furthermore, the additional delay due to the low-voltage cells in the rounding cycle might increase the critical path. However, we roughly estimated that the energy dissipated could be halved with respect to *l-p*. Entry *d-v* in Table 4.1 represents an estimation of a possible implementation with low-voltage gates. The energy reduction with respect to the basic divider is about 60%.

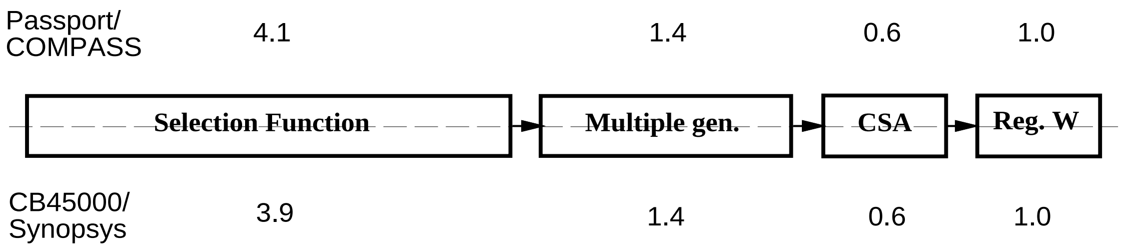


Figure 4.8: Critical path for implementations with Passport/COMPASS and CB45000/Synopsys.

4.2.4 Optimization with Synopsys Power Compiler

Recurrence in radix-4 divider

The first approach was to synthesize with Synopsys Design Compiler the RT-level VHDL description of a fairly complex circuit as the recurrence portion of the radix-4 divider. The timing constraints were set accordingly to the relation between the critical path obtained for the implementation of Section 4.2 with Passport/COMPASS (7.0 ns) and the ratio between the speed of the two libraries (0.67). The resulting timing constraint of 5.0 ns for the critical path was not met (7.0 ns) in the synthesis with Design Compiler. The critical path of the resulting circuit is compared in Figure 4.8 with the one obtained with Passport/COMPASS. Note that the critical path for the implementation with Passport/COMPASS was not obtained by synthesis of the whole RT-level model, but by manual design of the blocks in the recurrence with the exception of the selection function that was synthesized stand-alone using COMPASS ASICSynthesizer.

After having obtained the fastest possible circuit with Synopsys we optimized the power with Power Compiler. Results showed a reduction in the power dissipated of about 7% with a small increase in the critical path (2%).

Then, we synthesized the RT-level VHDL description of the retimed recurrence and we got a better reduction in power dissipation (about 10%) and a shorter critical path (5.9 *ns*), but still the timing constraints were not met.

In conclusion, for larger and fairly complex circuits not only the power is not reduced much, but also the initial design, optimized for smaller delay, is not as good as attainable by manual design. For this reasons, we decided to use Synopsys Power Compiler only to optimize the energy dissipation of small blocks, as described next.

Selection function of radix-4 divider

The second approach was to use the same methodology used for the design with COMPASS: manual design of the large regular blocks and synthesis of selection function and other small irregular blocks.

The synthesis of the selection function stand-alone was more satisfactory and showed a critical path of 3.0 *ns* (critical path for SEL in Passport/COMPASS implementation is 4.0 *ns*). The power reduction, obtained by incremental compilation with power dissipation constraints, was of about 20%, without affecting the delay.

In Table 4.1, the columns labeled *syn* represent an estimate of the units derived from *l-p* and *d-v* in which the selection function was optimized with Power Compiler.

4.2.5 Summary of Results for Radix-4

Table 4.1 summarizes the result obtained in the low-power optimization of the radix-4 divider. Each column represents a different implementation. Values in boldface indicate a variation from the previous value. Entry *std* refers to the standard implementation, optimized for speed, entry *rec* is obtained from *std* by applying low-power techniques to the recurrence portion, and entry *l-p* is *rec* with the low-power conversion and rounding. Entry *d-v* is an estimate of a possible implementation with dual voltage, and entries *syn* indicate the improvements attainable with Synopsys Power Compiler optimization. In columns *syn* only variations in SEL are indicated.

blocks	std nJ	rec nJ	l-p nJ	syn (est.)	d-v (est.)	syn (est.)
control	1.1	1.1	1.1	1.1	1.1	1.1
clk tree	0.9	0.9	0.9	0.9	0.9	0.9
mux	1.1	**0.3**	0.3		0.3	
mul. gen.	3.6	**2.8**	2.8		**1.9**	
CSA	5.9	**4.8**	4.8		**2.2**	
sel. func.	1.3	**1.6**	1.6	**1.2**	1.6	**1.2**
register Ws	6.4	6.4	6.4		***4.0**	
register Wc	6.2	**3.5**	3.5		**2.0**	
register q	-	**0.3**	0.3		0.3	
total recur.	24.5	**19.5**	19.5	**19.0**	**12.0**	**11.5**
SZD	5.7	5.7	**0.6**		0.6	
conv-round unit	13.2	13.2	**3.9**		***1.4**	
total C&R	19.0	19.0	**4.5**	4.5	**2.0**	2.0
Total divider	45.5	**40.5**	**26.0**	**25.5**	**16.0**	**15.5**
Ratio	1.00	**0.90**	**0.60**	**0.55**	**0.35**	**0.33**

Values marked * include level shifters

Table 4.1: Energy consumption per division for radix-4.

The delay of the divider is not changed because the retiming did not increase the critical path and other modification that affected delay were done for parts in the unit not in the critical path. As for the area, we have a reduction of about 20% between *std* and *l-p*. This is mainly due to the change in the redundant representation of $w[j]$ and in the new convert-and-round unit. In both cases we eliminated flip-flops, about 25% of the total. We estimated that an optimization with Synopsys Power Compiler could reduce the energy dissipation by an additional 5%.

Figure 4.9 shows the breakdown, as a percentage of the total, of the energy dissipated in the main blocks composing the unit.

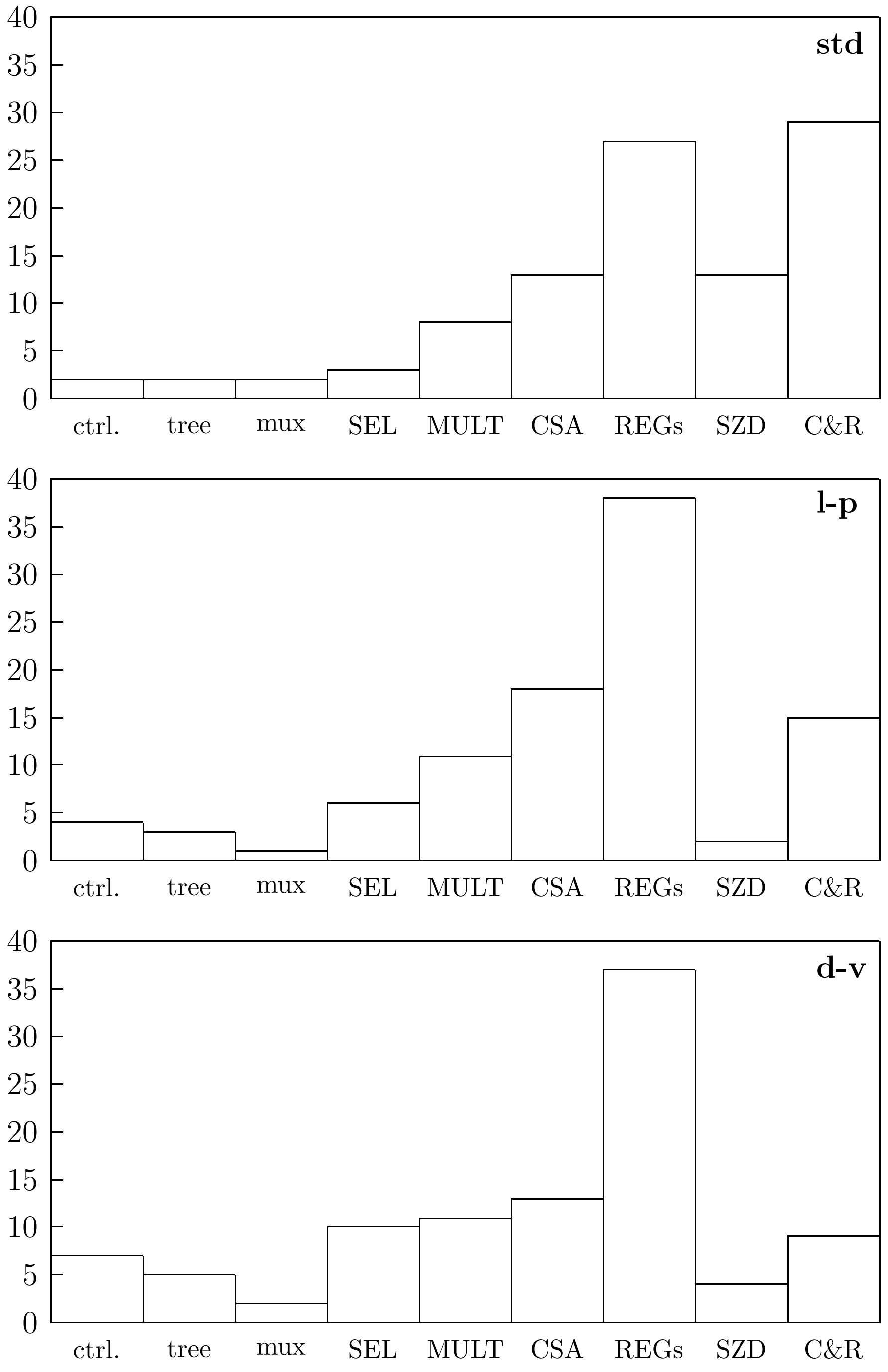


Figure 4.9: Percentage of energy dissipation in radix-4 divider.

4.3 Radix-8 Division

4.3.1 Algorithm and Basic Implementation

The radix-8 division algorithm is implemented by the residual recurrence

$$w[j+1] = 8w[j] - q_{j+1}d \qquad j = 0, 1, \ldots m$$

with initial value $w[0] = x$, where x is the dividend, d the divisor, and q_{j+1} the quotient digit at the j-th iteration. Both d and x are normalized in $[0.5, 1)$. The quotient digit is in signed-digit representation $\{-a, \ldots, -1, 0, 1, \ldots, a\}$ with redundancy factor $\rho = a/7$. The residual $w[j]$ is stored in carry-save representation (w_S and w_C). The quotient digit is determined, at each iteration, by the selection function

$$q_j = SEL(d_\delta, \hat{y})$$

where d_δ is d truncated after the δ-th fractional bit and $\hat{y} = 8w_S + 8w_C$ truncated after t fractional bits.

In order to avoid the implementation of a complicated multiple generator, the quotient digit is split into two parts q_H with weight 4 and q_L with weight 1 ($q_j = 4q_H + q_L$) and the digit set of each part is reduced to $\{-2, -1, 0, 1, 2\}$.

Since the selection function (SEL) is in the critical path, to have the minimum latency we have to minimize its delay. We explored the implementation of three possible values of a: 6, 7, and 10 (the maximum value possible with the above mentioned representation). Table 4.2 shows a summary of the results. The gate-level implementation was obtained by synthesizing the VHDL description of the selection function with COMPASS ASICSynthesizer. This includes both the assimilation of the carry-save representation of $\hat{y}$ and the actual digit-selection function.

From Table 4.2, we can see that SEL for $a = 7$ is as fast as for $a = 6$, but

	bits of		delay $[ns]$		
a	d	$\hat{y}$	q_L	q_H	gates
10	3	6	4.0	3.0	325
7	3	8	3.8	3.0	370
6	5	7	3.8	2.7	420

Table 4.2: Radix-8: summary selection function.

m_h	d_δ							
	8/16	9/16	10/16	11/16	12/16	13/16	14/16	15/16
m_7	27	30	34	36	40	44	48	48
m_6	23	26	28	32	34	36	40	40
m_5	18	20	24	24	28	28	32	32
m_4	14	16	18	20	20	24	24	24
m_3	10	12	12	12	16	16	16	16
m_2	6	8	8	8	8	8	8	8
m_1	0	0	0	0	0	0	0	0
m_0	-4	-4	-4	-4	-4	-4	-4	-4
m_{-1}	-8	-8	-8	-8	-8	-8	-12	-12
m_{-2}	-12	-12	-12	-16	-16	-16	-16	-20
m_{-3}	-16	-16	-20	-20	-24	-24	-24	-28
m_{-4}	-20	-22	-24	-26	-28	-32	-32	-36
m_{-5}	-24	-26	-30	-32	-36	-36	-40	-44
m_{-6}	-28	-31	-34	-38	-40	-44	-48	-52

Values in table are multiplied by 16

Table 4.3: Selection function for radix-8 and $a = 7$.

its area is smaller[1]. Surprisingly, the delay for the over-redundant case $a = 10$ is larger. Therefore, the SEL for $a = 7$ is chosen, which results in a redundancy factor $\rho = 1$. The selection logic function is described in Table 4.3.

A first implementation of the divider is shown in Figure 4.10. The scheme is completed by a controller (not depicted in the figure).

To have the divider compliant with IEEE standard for double-precision while operating with fractional values, 1-bit shifts are performed on the operands. More-

[1]Smaller area implies smaller capacitance and usually reduced energy dissipation.

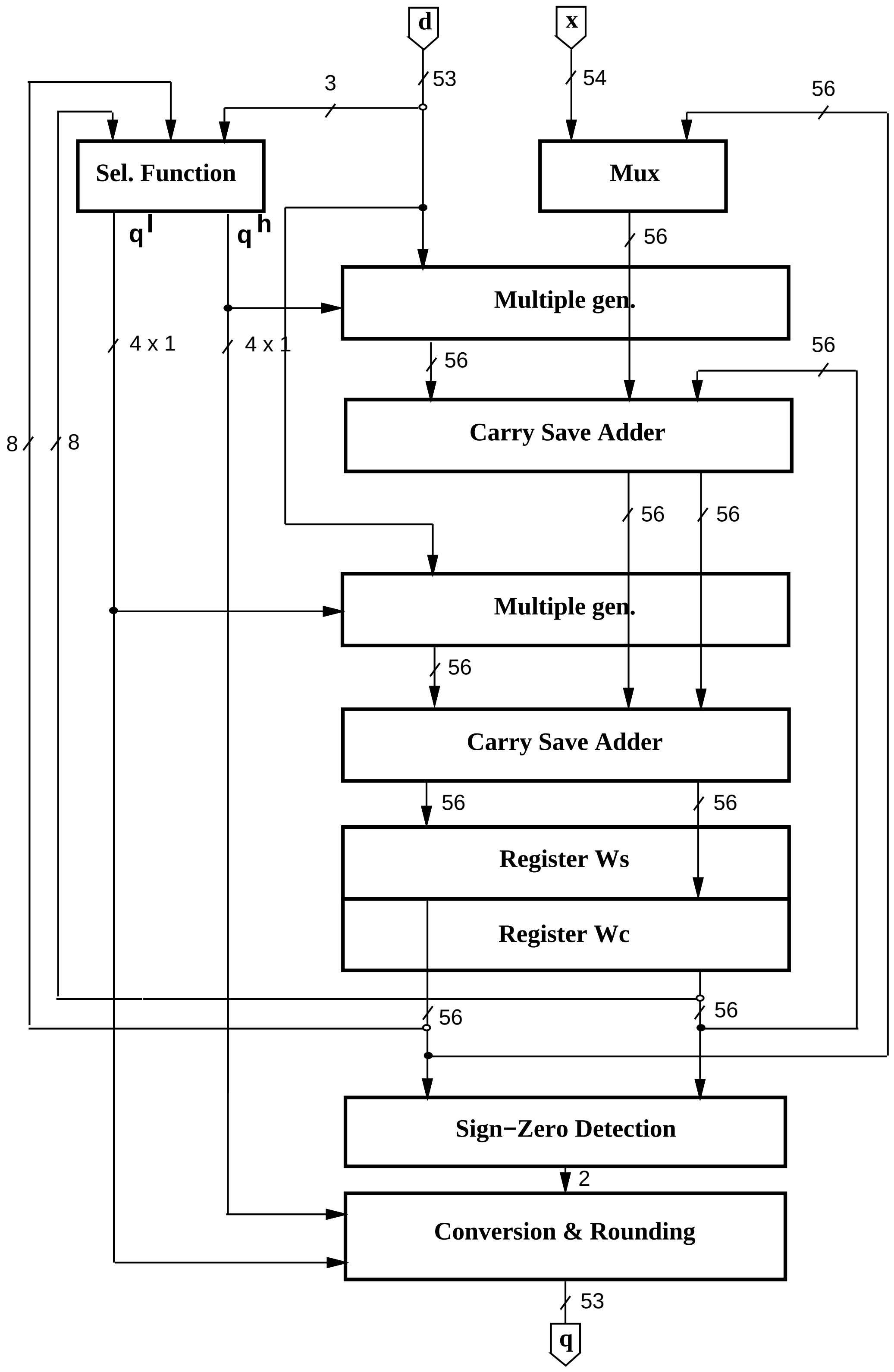


Figure 4.10: Implementation of the radix-8 divider.

over, to have a bound residual in the first iteration ($w[0] = x < d$), when $x \geq d$ we shift x one bit to the right obtaining a fractional quotient. To compute the 53 bits of the quotient and an additional bit to perform rounding, $54/3 = 18$ iterations are required. An additional cycle is required to load the value x as first residual $w[0]$. However, for the proposed architecture and selection function, the simplest way to accomplish this is to do as follows:

- Clear the registers for w (this is done at the end of the previous division). With the selection function we have implemented, this produces a $q_1 = 1$.

- Compute $w[1] = x - d$ using the hardware for the recurrence. This requires a multiplexer, which is not on the critical path.

- For q_1 to be 1, we shift the dividend three bits to the right. As a consequence, it is necessary to shift the final quotient accordingly,

In conclusion, the load cycle is substituted by an extra iteration for a total of 20 iterations: 19 to compute the digits and one for the rounding. Finally, the quotient is normalized in $[1, 2)$ by shifting it four positions to the left. Note that all shifts are done by wiring and do not affect the latency of the operation. In the recurrence ($w[j]$) we need 54 fractional bits and 2 integer bits: one to hold the sign and the other to avoid the overflow in the CS-representation (being $\rho = 1$).

There are two possible critical paths, one going through q_H and the other through q_L. Since the delay of q_H is smaller than that of q_L, but the number of adders to traverse is larger, a good design tries to equalize the delays of both paths. The resulting critical paths are

SEL(q_L)		+ mult	+ HA	+ reg	
4.5		+ 1.4	+ 0.6	+ 1.5	= 8.0 ns.
SEL(q_H)	+ mult	+ HA	+ FA	+ reg	
3.6	+ 1.4	+ 0.6	+ 0.9	+ 1.5	= 8.0 ns.

In conclusion, the post-layout critical path is 8.0 ns. The energy dissipated by this basic implementation is $E_{div} = 47.5\ nJ$ and the contributions of the blocks is shown in Table 4.4 in column "standard".

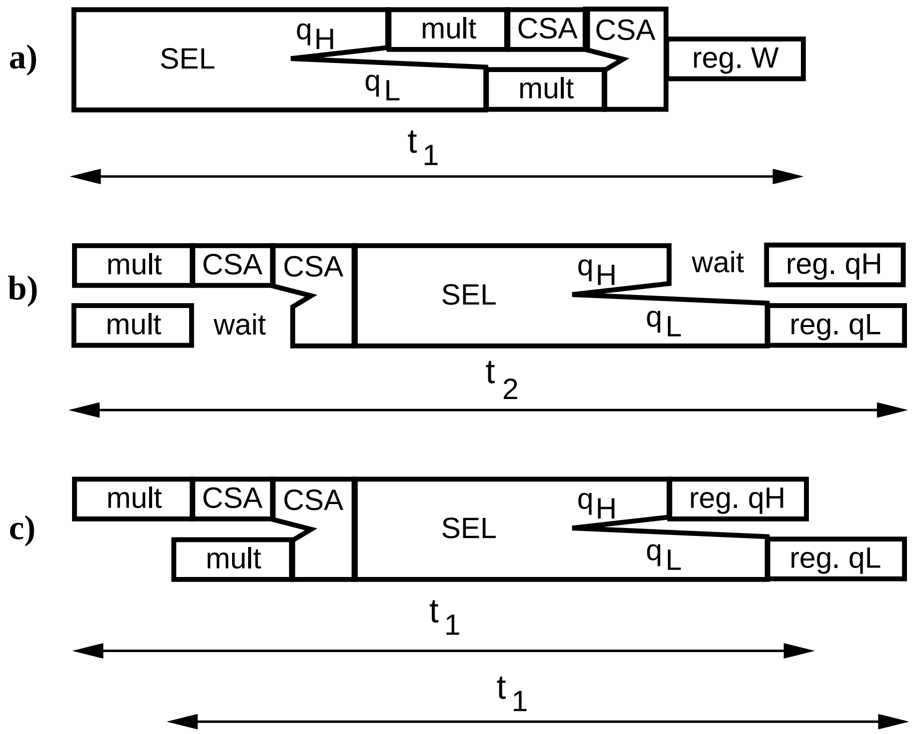
4.3.2 Low-Power Implementation

For the recurrence, the retiming is done by moving the selection function of Figure 4.10 from the first part of the cycle to the last part of the previous cycle (see Figure 4.11.a and Figure 4.11.b). Two new registers (q_H and q_L) are needed to store the quotient digit.

However the retiming alters the critical path because the two paths through q_H and q_L have different delays, and now the delay of q_L is added to the delay of the two CSAs (see Figure 4.11.b). To reduce the critical path to the previous value we skew the clock of register q_L by the delay difference of the two paths, which, in this case, corresponds to the delay of the CSA, as shown in Figure 4.11.c. The clock can be skewed by adding the appropriate delay (e.g. some buffers) in the clock distribution tree.

After the retiming the multiplexer is moved out of the recurrence, as shown in Figure 4.15. Consequently, the operations in the first cycle are modified by resetting registers q_H and q_L to 0 and -1 respectively and by storing x in $w[0] = 0 - (-x)$.

By using a radix-8 carry-save representation as shown in Figure 4.12, we only need to store one carry bit for each digit, instead of three. This can be done for the 50 LSBs that, after the retiming, are not on the critical path. The eight MSBs,



Box size is proportional to the delay.

Figure 4.11: Retiming and critical path. a) before retiming, b) after retiming, c) after retiming and skewing the clock.

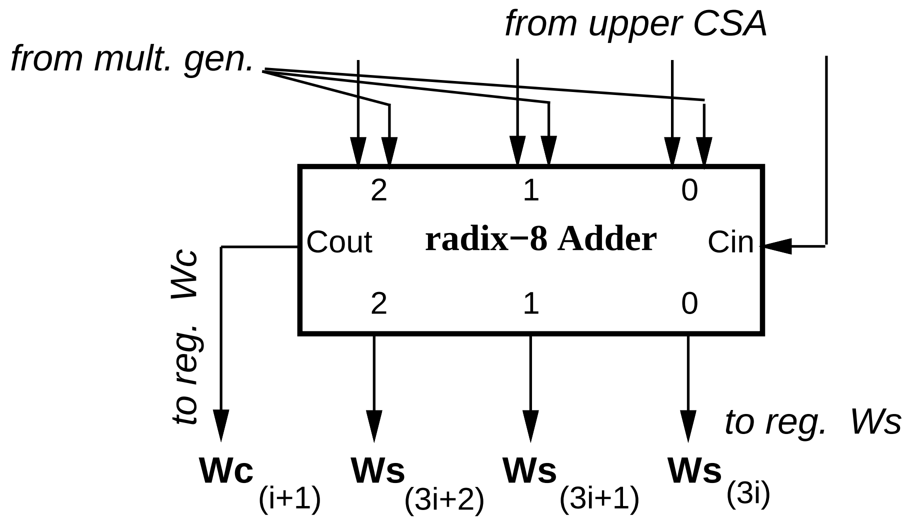


Figure 4.12: Radix-8 carry-save adder (lower).

assimilated in the adder inside the selection function block (Figure 4.15), can be stored in w_S eliminating another eight flip-flops in w_C. By retiming and changing the representation, the reduction in *l-p* with respect to *std* is about 10%.

The quotient-digit selection is a function of three bits of the divisor and eight of the residual. In the radix-8 case, Figure 4.13 shows the partitioning in eight parts (all the possible values of d_3) for both the higher and lower parts. By partitioning the selection function, we could obtain a reduction of 40% in the energy dissipated in SEL, but at the expense of a larger clock cycle (about 10%), due to additional delay of the demultiplexer and the OR gates, and area. For this reason, the corresponding value is not included in Table 4.4.

In the original formulation of the on-the-fly conversion algorithm, three registers (Q, QM, and QP) are necessary to store the partial quotient for $\rho = 1$. As explained in Section 3.10, the algorithm is modified by eliminating the shifting of the digits previously loaded, the two registers QM and QP, and by recoding. Two additional registers are introduced: the ring counter C to keep track of the digits to update and the temporary register T used in the recoding (Figure 4.14). In this way, we

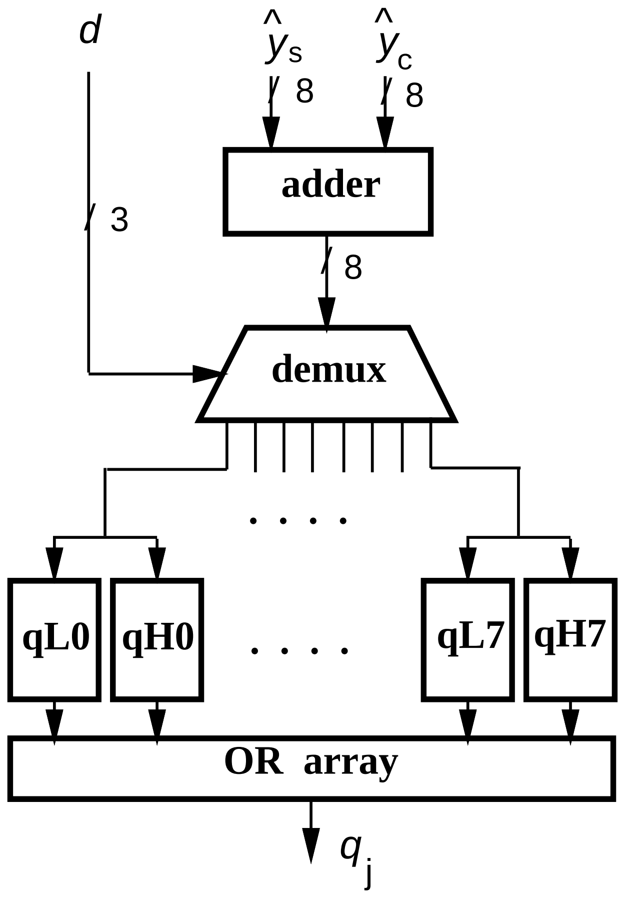


Figure 4.13: Partitioned selection function.

reduce the number of flip-flops in the convert-and-round unit from 171 (registers Q, QM, QP) to 81 (registers Q, C, T).

4.3.3 Dual Voltage Implementation

For the radix-8 divider, the 50 least-significant bits in the recurrence can be redesigned for low voltage. We can apply the dual-voltage technique also to the convert-and-round unit which is not in the critical path.

When applying dual voltage to the recurrence, the two cases with radix-2 and radix-8 CSA must be considered. As explained in Section 3.6, the time slack is longer for the radix-2 CSA implementation and the possible voltage V_2 is lower. In particular for the radix-8 divider (critical path is 8.0 ns), we obtain the following values:

radix-2 CSA $t_{slack} = 3.5\ ns \Rightarrow V_2 = 2.0\ V \Rightarrow E_{div} = 20\ nJ.$

radix-8 CSA $t_{slack} = 1.2\ ns \Rightarrow V_2 = 3.0\ V. \Rightarrow E_{div} = 26\ nJ.$

The values of E_{div} indicated above take into account both the different number of flip-flops in register Wc and the voltage scaling in the convert-and-round unit.

It is clear that the implementation with radix-2 CSA is the most convenient for dual voltage. The reduction in *d-v* is about 30% with respect to *l-p*.

4.3.4 Optimization with Synopsys Power Compiler

Because the synthesis of large circuits does not give results as good as manual design, we synthesized only the selection function using Synopsys Power Compiler. The synthesis of the selection function showed a critical path (through q_L) of 3.0 ns (critical path for SEL in Passport/COMPASS implementation is 4.5 ns), while the path through q_H was of 2.6 ns (3.6 ns for Passport/COMPASS). The energy

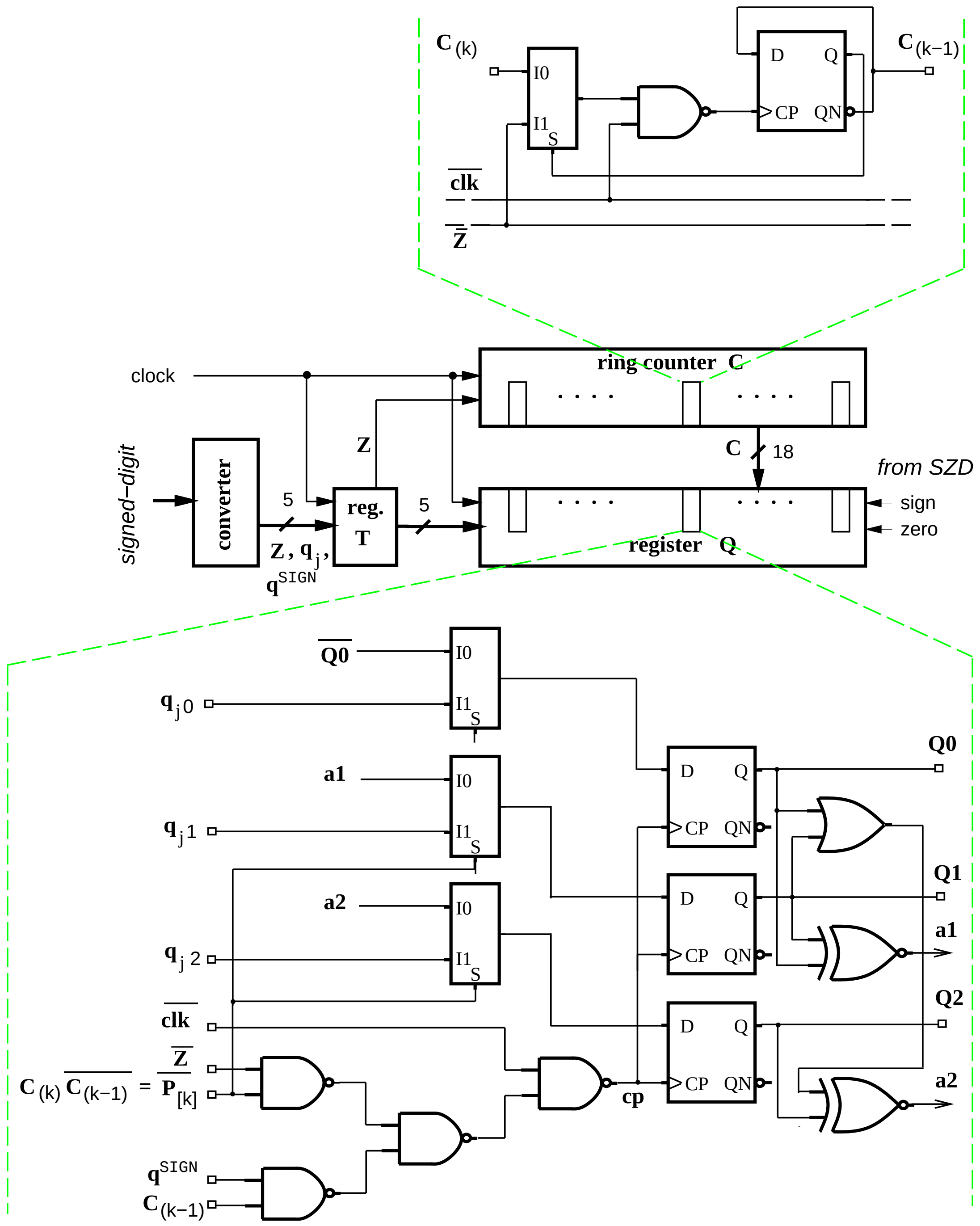


Figure 4.14: Convert-and-round unit for radix-8 divider .

reduction in the selection function, obtained by incremental compilation with power dissipation constraints was very little, about 5%.

4.3.5 Summary of Results for Radix-8

Figure 4.15 shows the implementation of the low-power radix-8 divider and Table 4.4 summarizes the results obtained by applying the low-power techniques described above. We did not include in the table the estimation of a possible implementation with Synopsys Power Compiler because the reduction of energy in the selection function is less than 1% of the total.

Figure 4.16 shows the breakdown, as a percentage of the total, of the energy dissipated in the main blocks composing the unit.

blocks	standard nJ	low-power nJ	dual voltage (est.) nJ
control	0.6	0.6	0.6
clk tree	0.4	0.4	0.4
mux	1.4	0.2	0.2
mul. gen. H	3.1	2.2	1.1
CSA H	4.4	4.4	2.2
mul. gen. L	2.6	2.2	1.1
CSA L	6.0	4.8	2.4
sel. func.	3.6	4.6	4.6
register Ws	4.2	4.0	*2.2
register Wc	4.2	1.2	*2.0
register q_H	-	0.2	0.2
register q_L	-	0.2	0.2
SZD	3.8	0.6	0.6
C&R unit	13.4	2.8	*1.0
Total $[nJ]$	47.5	28.5	19.0
Ratio	1.00	0.60	0.40
Area $[mm^2]$	2.2	1.8	-

Values marked * include level shifters.

Table 4.4: Energy-per-division for radix-8.

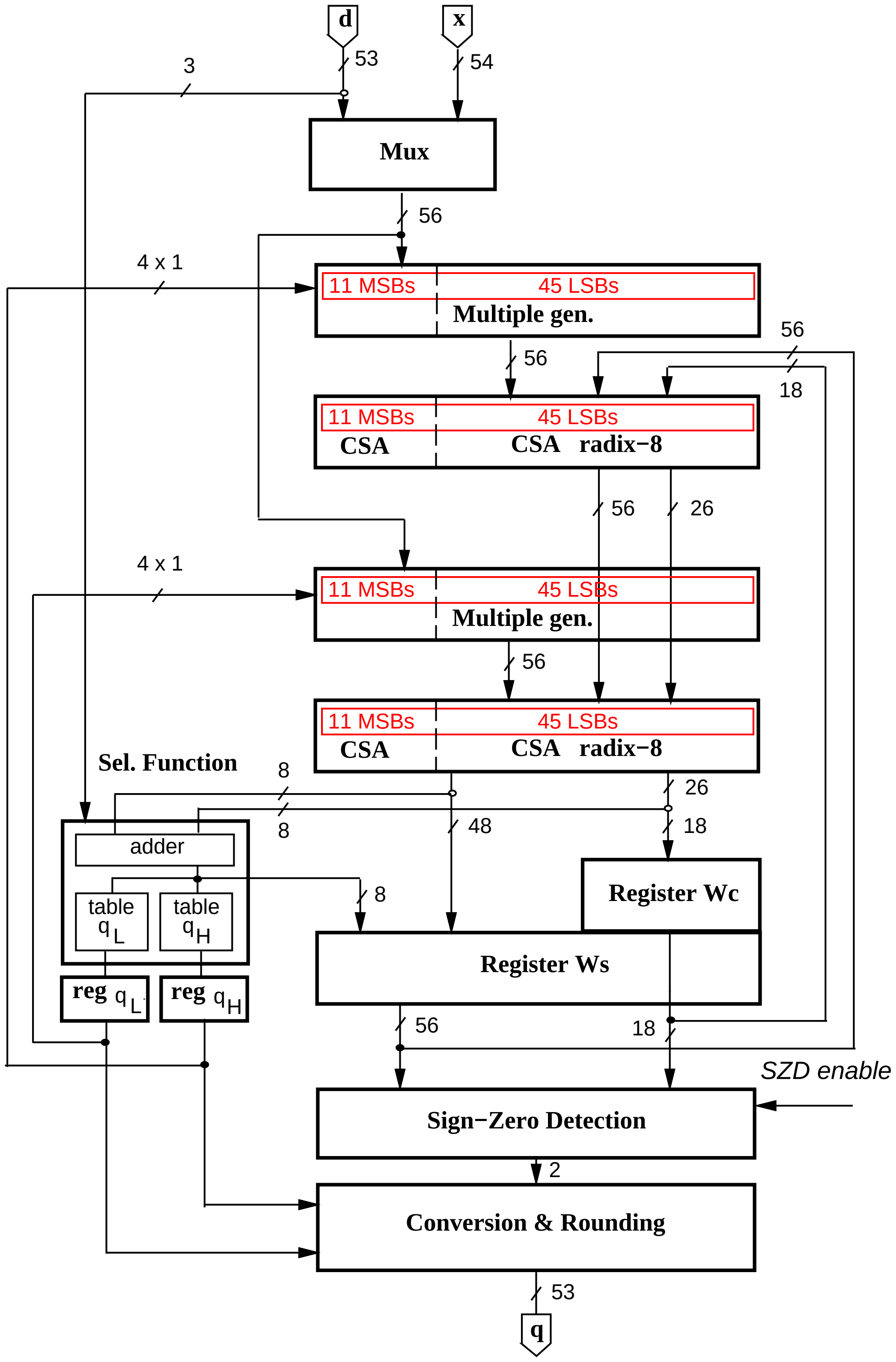


Figure 4.15: Low-power implementation of the radix-8 divider.

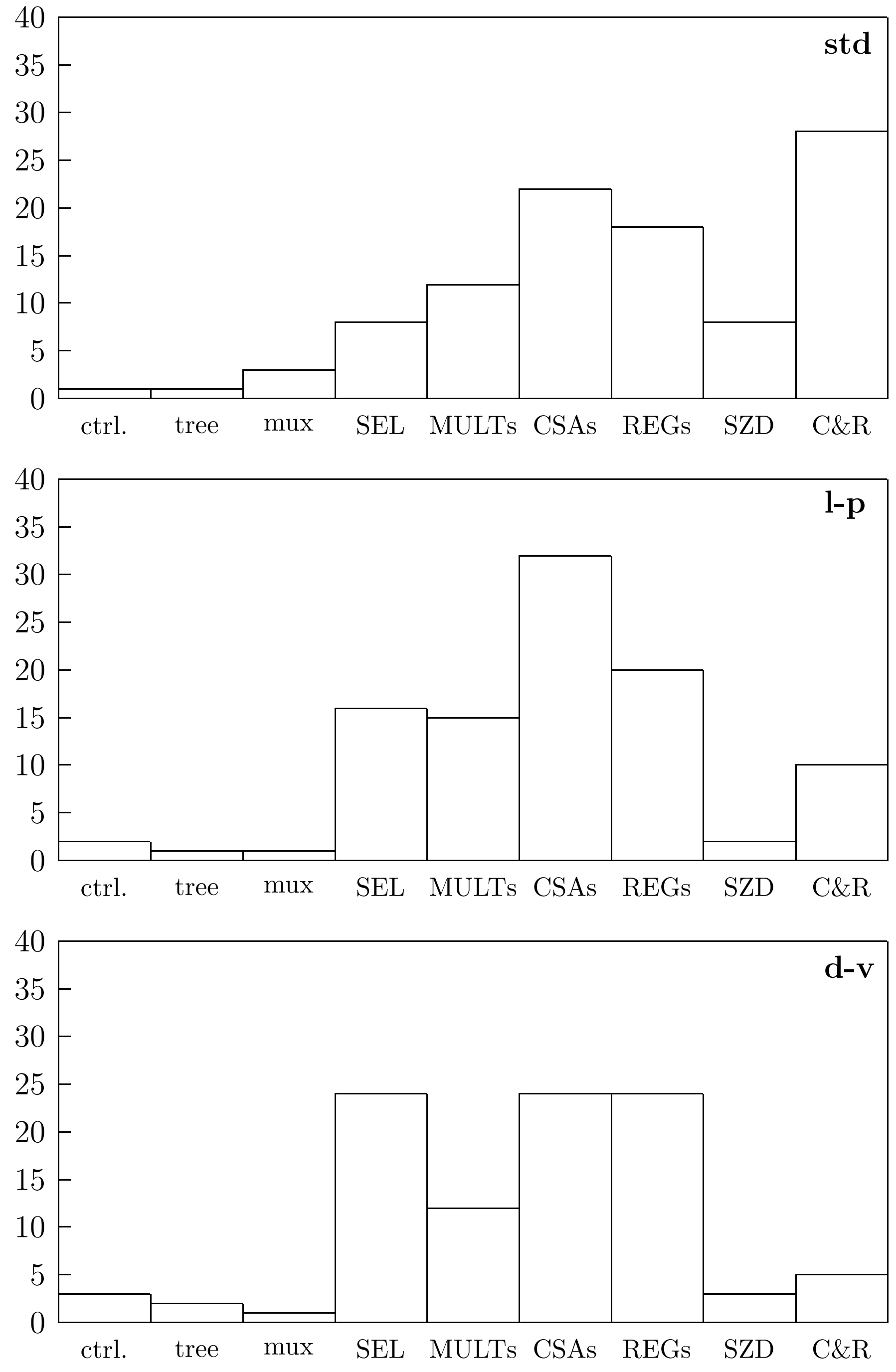


Figure 4.16: Percentage of energy dissipation in radix-8 divider.

In the *l-p* implementation the largest part of the energy is dissipated in the CSAs (more than 30%), while in the *d-v* estimate the largest portion is equally distributed (about 25% of the total for each block) among the two CSAs, the two registers W and the selection function.

4.3.6 Comparison with Scheme with Overlapped Radix-2 Stages

In [34] a radix-8 divider is implemented by overlapping three radix-2 stages and computing the quotient digits in parallel. Moreover, the next partial remainder (w_s and w_c) is calculated speculatively for each possible quotient digit. This scheme, indicated here as *r8overlap*, is implemented in the Sun *UltraSPARC* FP-unit. As described in [34], the critical path is: $1 \times SEL + 2 \times CSA + 3 \times MUX$

In order to compare the *r8overlap* division unit with our radix-8 divider, we made the following assumptions:

- The CSA (or FA) has the same delay (0.8 ns) in both implementations.
- The multiple generator is equivalent to the 3:1 MUX of *r8overlap*.
- We implemented the radix-2 selection function of [34] with our library and obtained a delay of 1.9 ns.
- Buffering of one of the MUXes is required.

With these assumptions, we can reasonably estimate the pre-layout critical path of *r8overlap* as:

SEL	+ 2 CSA	+ 3 MUX	+ buff.	+ reg.
1.9	+ 1.6	+ 1.5	+ 0.7	+ 1.3 = 7.0 ns

This is similar to the critical path (pre-layout) of the radix-8 unit described here:

SEL(q_L)		+ mult	+ HA	+ reg	
3.8		+ 1.2	+ 0.5	+ 1.3	= 6.8 ns.
SEL(q_H)	+ mult	+ HA	+ FA	+ reg	
3.0	+ 1.2	+ 0.5	+ 0.8	+ 1.3	= 6.8 ns.

As for the area, Table 4.5 shows a comparison of the number of the wider (bitwise) blocks. Register QN in *r8overlap* can be eliminated by introducing register C. However it is not possible to change the representation of w_C (e.g. reducing the size of register Wc) without penalizing the performance. Moreover, the resulting selection function of the overlapped implementation is about twice as large as the radix-8 SEL. In conclusion, it is reasonable to assume that the area of our divider is significantly smaller.

	r8overlap	radix-8
no. CSAs	6	2
no. mux/mult	6	2
no. registers	4 (Ws, Wc, Q, QN)	2.7 (Ws, Wc/3, Q, C)

Table 4.5: Area comparison.

We don't have data available on the energy consumption of the *r8overlap* division unit, but considering the larger area (larger current drawn) and roughly the same operation latency, we conclude that the energy dissipation is smaller in our implementation. Furthermore, the energy reduction techniques applied in the radix-8 divider might not be effective in the *r8overlap* scheme.

4.4 Radix-16 Division

4.4.1 Algorithm and Implementation

The radix-16 division algorithm is implemented by the residual recurrence

$$w[j+1] = 16w[j] - q_{j+1}d \qquad j = 0, 1, \ldots 13$$

with initial value $w[0] = x$, and quotient given by

$$q = \sum_{j=1}^{14} q_j 16^{-j} \tag{4.4}$$

Two additional cycles, for initialization and rounding, are required to produce the quotient in conventional representation (53 bits for IEEE double-precision) for a total of 16 cycles. As usual, both d and x are normalized in $[0.5, 1)$ and $x < d$.

The radix-16 division unit is obtained by overlapping the computation of two radix-4 digits [30]. Consequently, the quotient digit is split into two parts q_H and q_L such that $q_j = 4q_H + q_L$ with digit set $\{-2, -1, 0, 1, 2\}$ in each part, resulting in the digit-set $[-10, 10]$ for q_j ($a = 10$). The quotient digit is determined, at each iteration, by the selection function depicted in Figure 4.17. Once the digit q_H is chosen, its value is used to select among all the possible combinations of $q_H d$. The redundancy factor is $\rho = \frac{a}{r-1} = \frac{2}{3}$. The residual $w[j]$ is stored in carry-save representation (w_S and w_C). The signed-digit representation of the quotient is converted to conventional two's complement representation and rounded by the on-the-fly convert-and-round unit.

The implementation of the standard divider, optimized for shortest latency, is shown in Figure 4.18. Table 4.6 shows the delay through the two parts of SEL. Note that the larger delay of SELq_L is compensated by the additional carry-save adder in the recurrence (Figure 4.18) in the path from SELq_H.

The critical path post-layout is 9.2 ns and 16 cycles are required to complete the operation, corresponding to a latency of 150 ns. The energy dissipated by this basic implementation is $E_{div} = 46.0\ nJ$ and the contributions of the blocks is shown in Table 4.9 in column "standard".

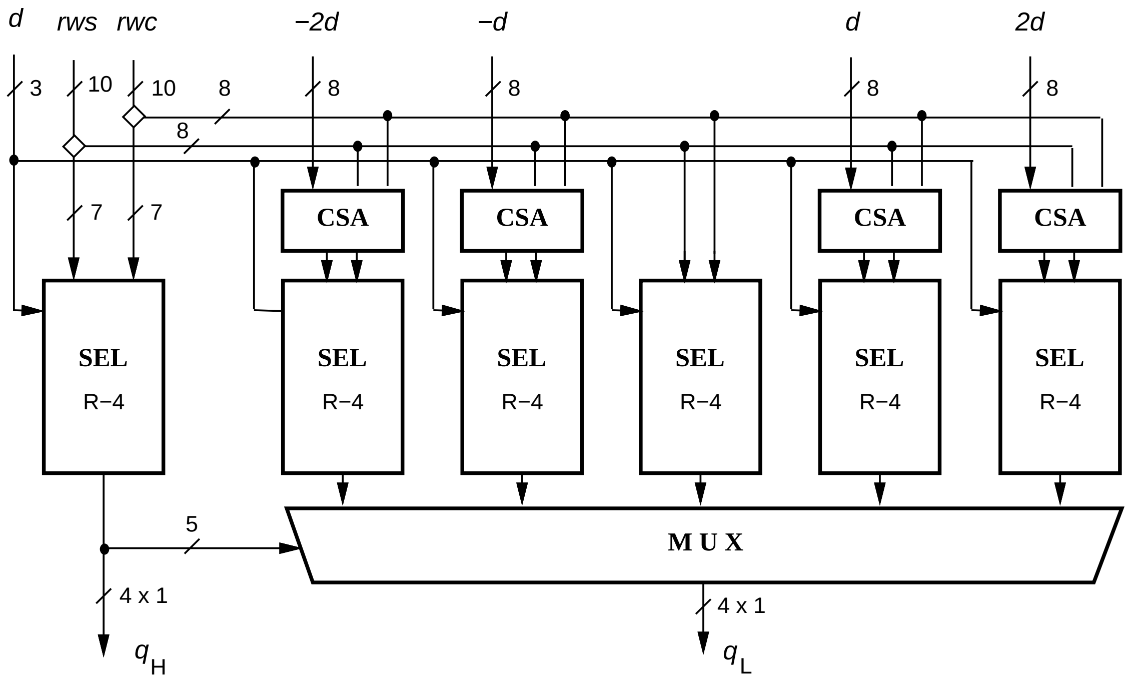


Figure 4.17: Selection function for radix-16.

	path		delay [ns]
q_L	SELq_L - MULT - HA - REG		
	5.7 + 1.4 + 0.6 + 1.5	=	9.2
q_H	SELq_H - MULT - HA - FA - REG		
	4.0 + 1.4 + 0.6 + 1.1 + 1.5	=	8.6

Table 4.6: Critical path through q_L and q_H.

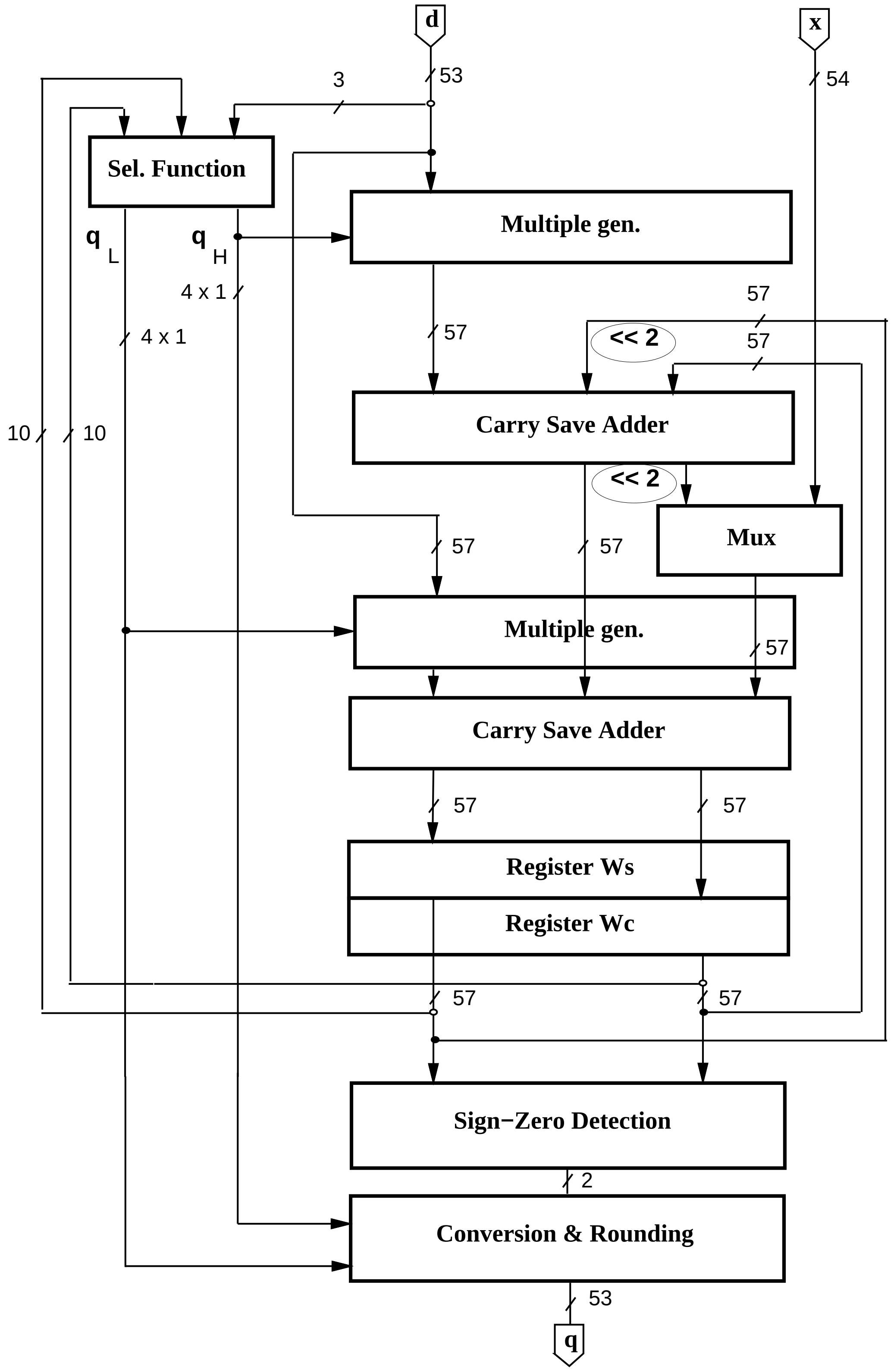


Figure 4.18: Basic implementation radix-16.

4.4.2 Low-Power Implementation

The retiming of the recurrence is done by moving the selection function from the first part of the cycle to the last part of the previous cycle. Two 4-bit registers are needed to store the quotient digit. After the retiming the critical path is limited to the 10 most-significant bits of the recurrence. As shown in Figure 4.19, in order not to increase the cycle delay in the retimed unit, the clock of register q_L is skewed.

Since in the retimed implementation the selection function is placed after the second CSA, instead of directly after the registers, there is a large increase in the number of glitches, which are responsible for the increased dissipation of the selection function. One way to filter those glitches is to buffer the selection function with multiplexers acting as latches, as described in Figure 3.13 at page 51. The select signal is driven by a different clock (same period, different phase) that enables the muxes to transmit the value from the CSA when it is stable, and hold the current value otherwise. However, in this case the delay of the mux affects the critical path. For radix-16 the energy dissipated in the selection function is halved, but the critical path is increased by about 5%. For this reason, the value is not included in Table 4.9.

For the 44 least-significant bits the radix-2 CSA is replaced by a radix-16 CSA (R16 CSA) that for each digit of the radix stores only one carry bit. Figure 4.20 shows the radix-16 CSA and Table 4.7 explains how the two level of adders are connected to produce the correct residual. The number of flip-flops in register Wc is reduced from 57 to 25.

In order not to increase the cycle time when using radix-16 CSA the two paths shown in Table 4.8 should have the same delay. Therefore, the condition to be satisfied is:

$$t_{R16\ CSA} \leq \frac{t_{HA} + t_{SEL_{q_L}}}{2} = 3.15\ ns$$

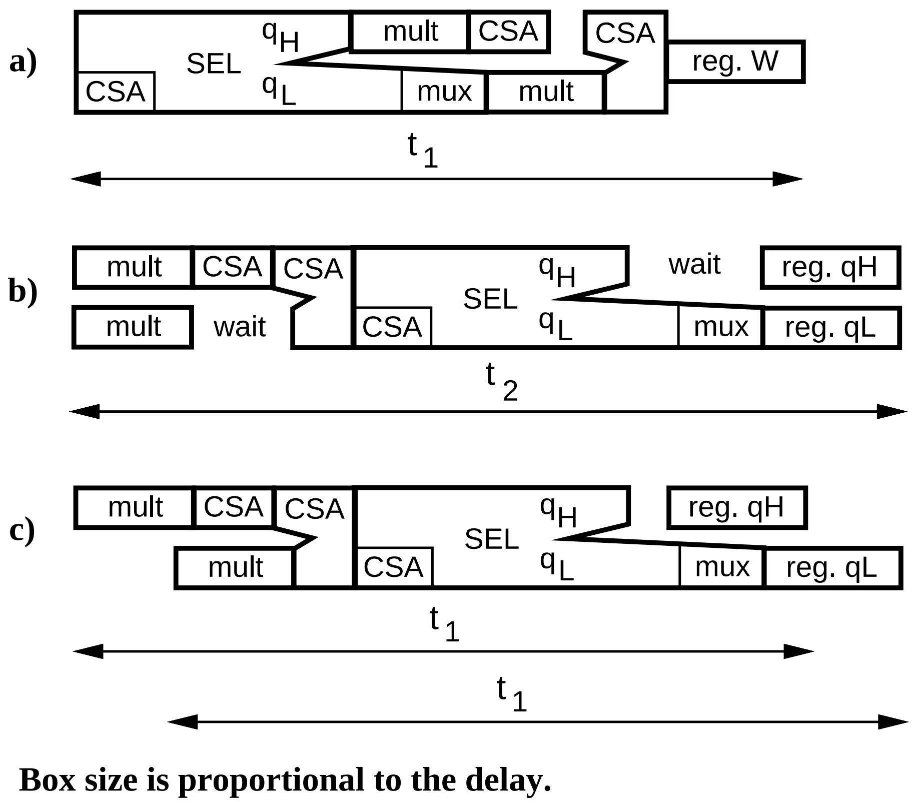


Figure 4.19: Retiming and critical path. a) before retiming, b) after retiming, c) after retiming and skewing the clock.

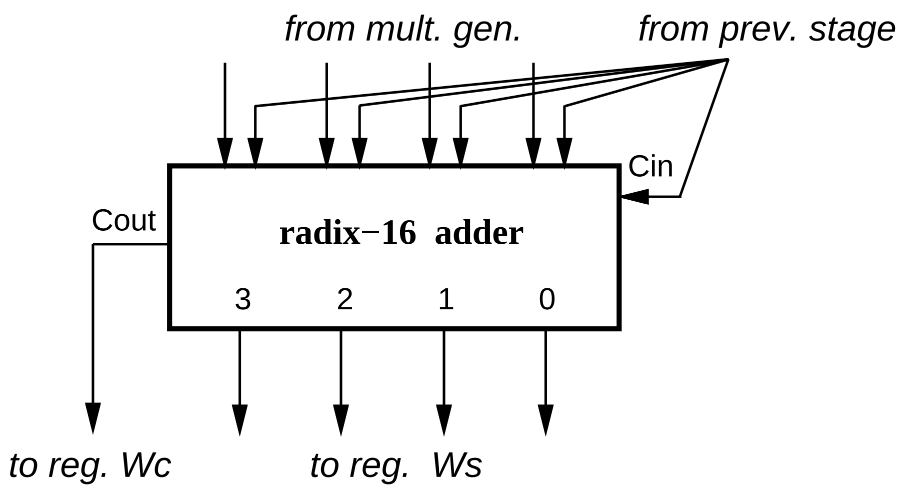


Figure 4.20: Radix-16 CSA.

iter. j		iteration $j+1$									
		first level					second level				
w_{s0}		$0\ m_{aS}$		m_{a0}	S_0		0	m_{bS}	m_{b0}	w_{s0}	
w_{s1}		0		m_{a1}	S_1		0		m_{b1}	w_{s1}	
w_{s2}	w_{c2}	$4w_{s0}$		m_{a2}	S_2		$4S_0$		m_{b2}	w_{s2}	w_{c2}
w_{s3}		$4w_{s1}$		m_{a3}	S_3		$4S_1$		m_{b3}	w_{s3}	
w_{s4}		$4w_{s2}$	$4w_{c2}$	m_{a4}	S_4	C_4	$4S_2$		m_{b4}	w_{s4}	
w_{s5}		$4w_{s3}$		m_{a5}	S_5		$4S_3$		m_{b5}	w_{s5}	
w_{s6}	w_{c6}	$4w_{s4}$		m_{a6}	S_6		$4S_4$	$4C_4$	m_{b6}	w_{s6}	w_{c6}
w_{s7}		$4w_{s5}$		m_{a7}	S_7		$4S_5$		m_{b7}	w_{s7}	
w_{s8}		$4w_{s6}$	$4w_{c6}$	m_{a8}	S_8	C_8	$4S_6$		m_{b8}	w_{s8}	
w_{s9}		$4w_{s7}$		m_{a9}	S_9		$4S_7$		m_{b9}	w_{s9}	
w_{s10}	w_{c10}	$4w_{s8}$		m_{a10}	S_{10}		$4S_8$	$4C_8$	m_{b10}	w_{s10}	w_{c10}
w_{s11}		$4w_{s9}$		m_{a11}	S_{11}		$4S_9$		m_{b11}	w_{s11}	
......		......					......				

Table 4.7: Bit arrangement in two-level adders.

The easiest way to implement this R16 CSA is by using a 4-bit carry-ripple adder. The corresponding delay, in our library, is

$$t_{Sn} = t_{C_1} + 3t_{ripple} + t_{HA} = 0.8 + 3(0.35) + 0.5\ ns \simeq 2.0\ ns\ .$$

MSBs:	MULT	HA	SEL QL	REG
LSBs:	MULT	R16 CSA	R16 CSA	REG

Table 4.8: Paths in MSBs and LSBs in the recurrence.

Furthermore, the 7 most-significant bits assimilated in the selection function could be stored in the register Ws, saving 7 additional flip-flops. However, in the radix-16 case, the assimilated value must be selected among the 5 possible alternatives (see Figure 4.17) and this requires an additional multiplexer driven by q_H that increases the load both on q_H and q_L. For this reason the 7 MSBs bits are stored in carry-save representation.

In the original on-the-fly conversion and rounding algorithm the partial quotient is stored in two registers (Q and QM). By implementing the modified algorithm, with $\rho = \frac{2}{3}$, only register Q (54 bits) and register C (14 bits) are needed. With the implementation of the modified algorithm the number of flip-flops in the convert-and-round unit is reduced from 108 to 69 and the power dissipated from $10.7nJ$ to $2.4nJ$, resulting in a reduction of about 20% in the whole divider.

4.4.3 Dual Voltage Implementation

When applying dual voltage to the recurrence, the two cases with a radix-2 and a radix-16 CSA must be considered. As explained in Section 3.6, the time slack is longer for the radix-2 CSA implementation and the possible voltage V_2 is lower. In particular, for the radix-16 divider (critical path is 9.2 ns), we get the following values:

radix-2 CSA $t_{slack} = 4.6\ ns \Rightarrow V_2 = 2.0\ V \Rightarrow E_{div} = 22\ nJ$.

radix-16 CSA $t_{slack} = 1.8\ ns \Rightarrow V_2 = 2.7\ V. \Rightarrow E_{div} = 27\ nJ$.

The values of E_{div} indicated above take into account both the different number of flip-flops in register Wc and the voltage scaling in the convert-and-round unit.

It is clear that the implementation with radix-2 CSA is the most convenient for dual voltage.

4.4.4 Optimization with Synopsys Power Compiler

The synthesized selection function showed a delay of 4.0 ns through q_L, and 3.0 ns through q_H with delay constraints met. In addition the reduction in energy dissipation, obtained by incremental compilation with power dissipation constraints resulted to be about 25%.

blocks	standard nJ	low-power nJ	synopsys (est.) nJ	dual voltage (est.) nJ	synopsys (est.) nJ
control	0.5	0.5		0.5	
clk tree	0.5	0.5		0.5	
mux	2.6	0.4		0.4	
mul. gen. H	2.5	1.6		0.8	
CSA H	3.3	3.3		1.9	
mul. gen. L	2.7	1.8		1.0	
CSA L	5.0	4.3		2.5	
sel. func.	5.9	8.2	6.1	8.2	6.1
register Ws	4.4	4.3		*2.1	
register Wc	4.2	1.6		*1.9	
register q_H	-	0.2		0.2	
register q_L	-	0.2		0.2	
SZD	3.7	0.6		0.6	
C&R unit	10.7	2.4		*0.9	
E_{div} $[nJ]$	46.0	30.0	28.0	22.0	20.0
Ratio	1.00	0.65	0.60	0.50	0.45
Area $[mm^2]$	2.2	1.8	-	-	-

Values marked * include level shifters.

Table 4.9: Energy-per-division for radix-16.

4.4.5 Summary of Results for Radix-16

Table 4.9 reports the average energy dissipation and area for the standard and the low-power implementation, which is shown in Figure 4.21.

Figure 4.22 shows the breakdown, as a percentage of the total, of the energy dissipated in the main blocks composing the unit.

In the *std* the largest part of energy is dissipated in the convert-and-round unit (about 25%). With the application of the energy reduction techniques, the energy dissipated in C&R unit is reduced to less than 10% of the total for *l-p* and less than 5% for *d-v*. On the other hand the contribution of the selection function to the total energy dissipation increases up to 35% of the total in *d-v*. By optimizing the selection function with Synopsys Power Compiler, a reduction of 25% in the block

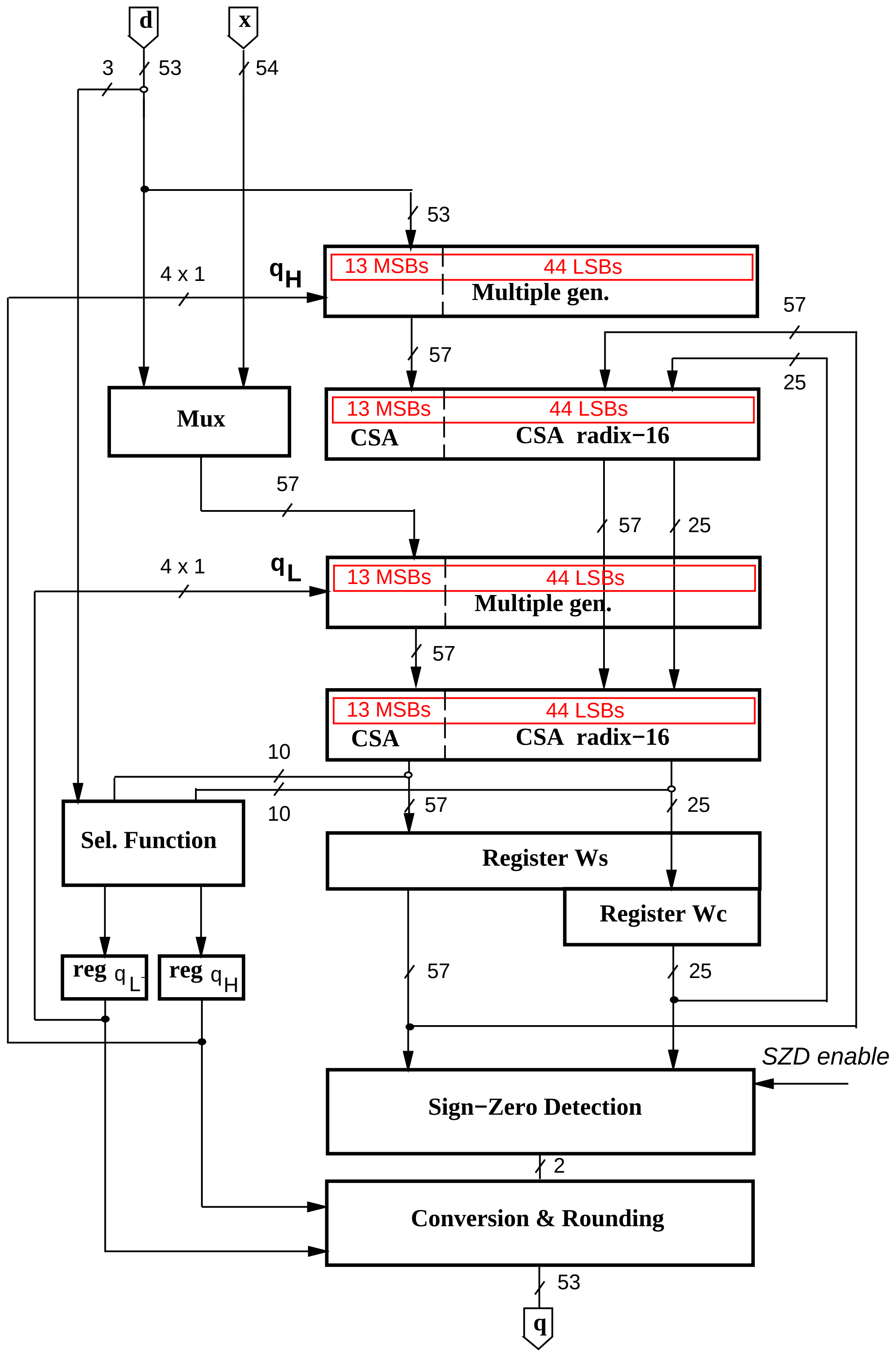


Figure 4.21: Low-power radix-16 divider.

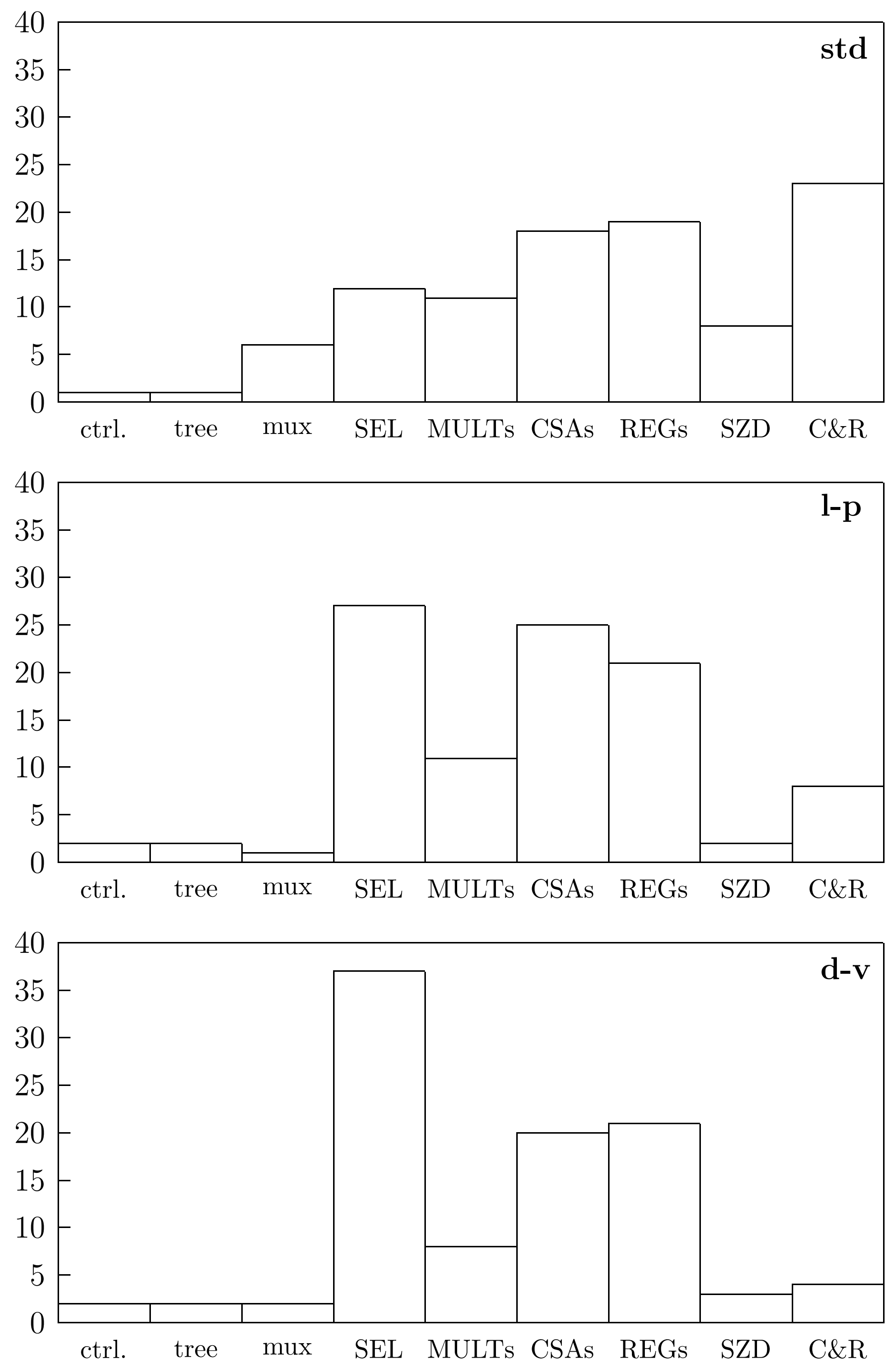


Figure 4.22: Percentage of energy dissipation in radix-16 divider.

would reflect in a contribution of about 30% of the total in *d-v*.

4.5 Radix-512 Division

4.5.1 Algorithm and Basic Implementation

We now refresh some of the expressions of the radix-512 division algorithm presented in Chapter 2, The recurrence is implemented by

$$w[j+1] = 512w[j] - q_{j+1}z \qquad j = 0, 1, \ldots 5 \tag{4.5}$$

with $w[0] = Mx$, $z = Md$, and quotient-digit selection

$$q_{j+1} = \lfloor \hat{y} + 1/2 \rfloor \ .$$

The scaling factor M is determined by

$$M = -\gamma_1 d_{15} + \gamma_2 \ . \tag{4.6}$$

All the details of the implementation of the radix-512 divider are given in [33]. Here we briefly summarize the main features of the unit and determine the energy dissipated.

As for the other radices, d is normalized in $[0.5, 1)$ and $x < d$. According to [32] the scaling factor M is in the range:

$$0 < M < 2$$

with 13 fractional bits. A total of 15 bits is required to store M. Because the scaled operands can be greater than 1, for the z and $w[j]$ representation we need one sign bit, one integer bit and $54 + 13 = 67$ fractional bits for a total of 69 bits. To have the correct recoding, as explained in [33], an extra integer bit is added. In conclusion, the number of bits needed to store the partial remainder $w[j]$ (bits in the recurrence) is 70.

A first implementation of the divider is shown in Figure 2.4 on page 31. Since the operations indicated in expression (4.5) and expression (4.6) are similar, they can be executed in the same unit. In [33] in order to reduce the area, the multiplier-accumulator required for the computation of M is eliminated, and the operation of expression (4.6) is executed in the main multiplier-accumulator. The modified block diagram is shown in Figure 4.23.

Figure 4.24 shows the operation performed in the divider and the values stored in the registers during the different cycles.

Block gamma-table is a logic function which produces the two quantities $-\gamma_1$ and $-\gamma_2$ according to

$$\gamma_1 = \frac{1}{{d_6}^2 + d_6 2^{-6} + 2^{-15}}$$

$$\gamma_2 = \frac{2d_6 + 2^{-6}}{{d_6}^2 + d_6 2^{-6} + 2^{-15}}$$

where d_{15} and d_6 are d truncated to its 15th and 6th fractional bit respectively. The block was synthesized with standard cells using COMPASS ASICSynthesizer.

The recoder is used to recode the multiplier into radix-4 representation with digits in the set $\{-2, -1, 0, 1, 2\}$.

Block MultAdd in Figure 4.23 executes both the multiplication and the addition in the recurrence. The multiple generator produces the partial products $t_0, t_1, \ldots, t_7$ and the adder reduces the number of the partial products to the final carry-save representation (9:2 adder). Summarizing, the MultAdd operations are:

- In M calculation, the recoded multiplier is d_{15} (15 bits) which produces 8 partial products $(t_0, t_1, \ldots, t_7)$. In addition, $-\gamma_2$ must be added.

$$-M = -(-\gamma_1 d_{15} + \gamma_2) = t_0 + t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + (-\gamma_2)$$

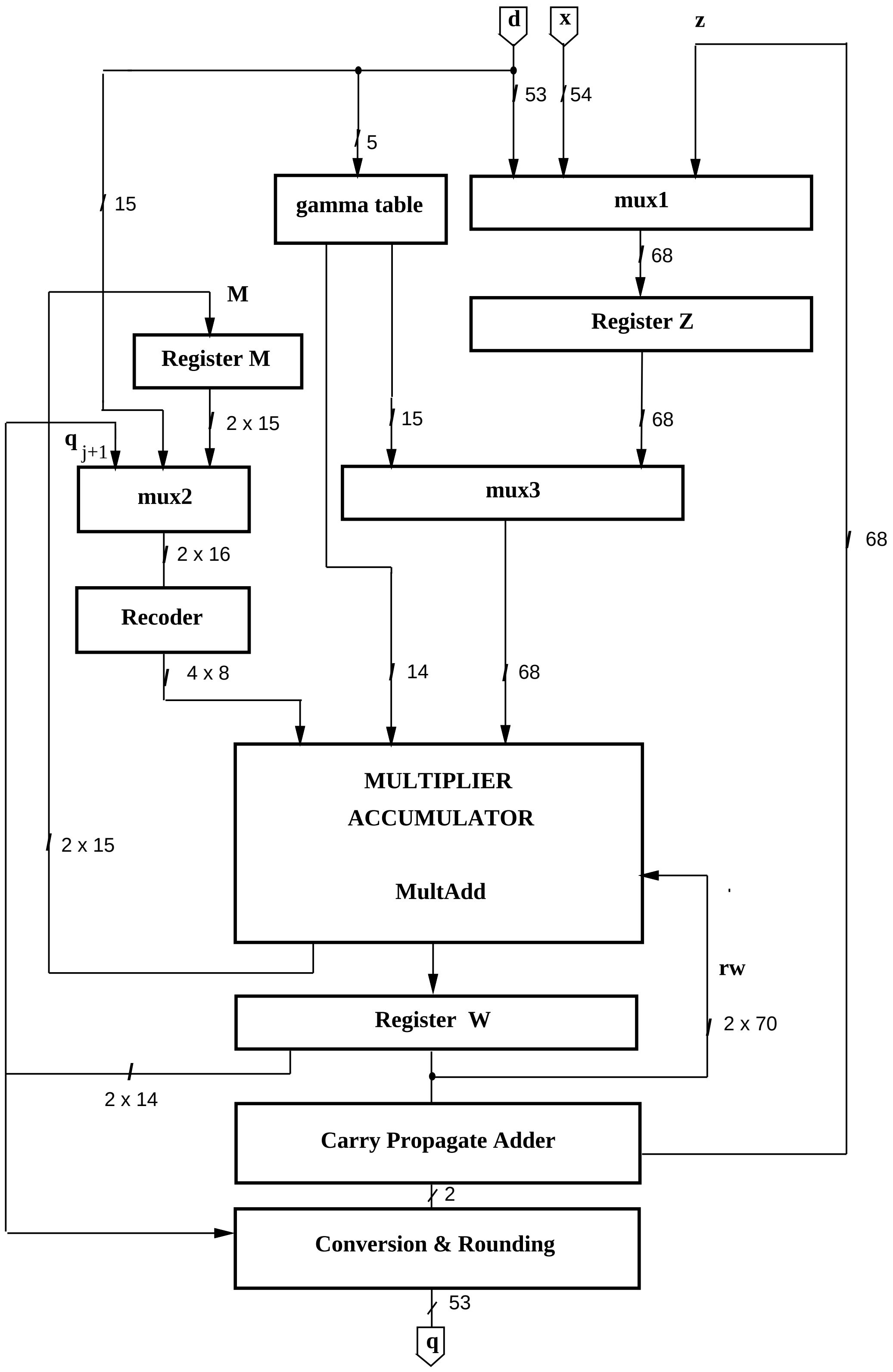


Figure 4.23: Block diagram of modified divider.

cycle	1	2	3	4	5	6	7	8	9	10
Reg. Z		d	x	z						
Reg. M		M								
MultAdd	M	Md	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	
Reg. W			Md	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]

Figure 4.24: Cycles and operations.

- In the scaling, the recoded multiplier is $-M$ (15 bits).

$$Md,\ Mx\ = t_0 + t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7$$

- In the recurrence, the recoded multiplier is q_{j+1} that is 11 bits (10 + 1 for sign) bits. In this case only 6 partial products are generated (t_0-t_5) and t_6 and t_7 are used to add the carry-save represented shifted residual $rw[j]$.

$$w[j+1] = 512\ w[j] - q_{j+1}z = t_0 + t_1 + t_2 + t_3 + t_4 + t_5 + rw_s + rw_c$$

The conversion block performs the conversion from the signed-digit quotient and the rounding. The carry-propagate adder is used to assimilate the carry save representation of $Md = z$ and the final remainder. In addition, hardware to detect if the sum is zero (needed for the rounding) is provided in the carry-propagate adder.

The first implementation of the divider, which corresponds to the scheme of Figure 4.23, has a critical path of 10.5 ns (Figure 4.25) and a total execution time

$$t_{div} = 10.5 \times 10 = 105 \quad [ns]\ .$$

The energy dissipated by this standard implementation is reported in Table 4.11 and indicated in the column "standard". Figure 4.26 shows the percentage of energy dissipated in the blocks for the basic implementation of Figure 4.23.

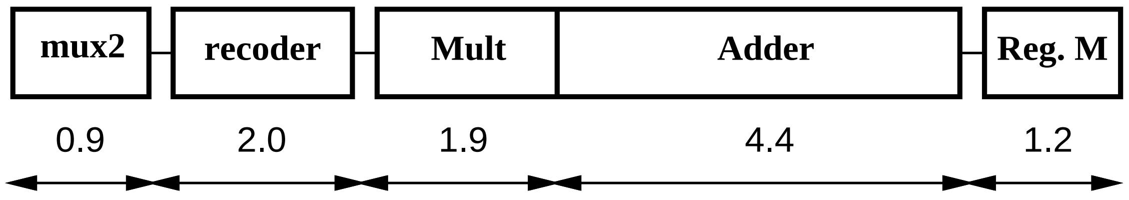


Figure 4.25: Critical path (ns) for basic implementation.

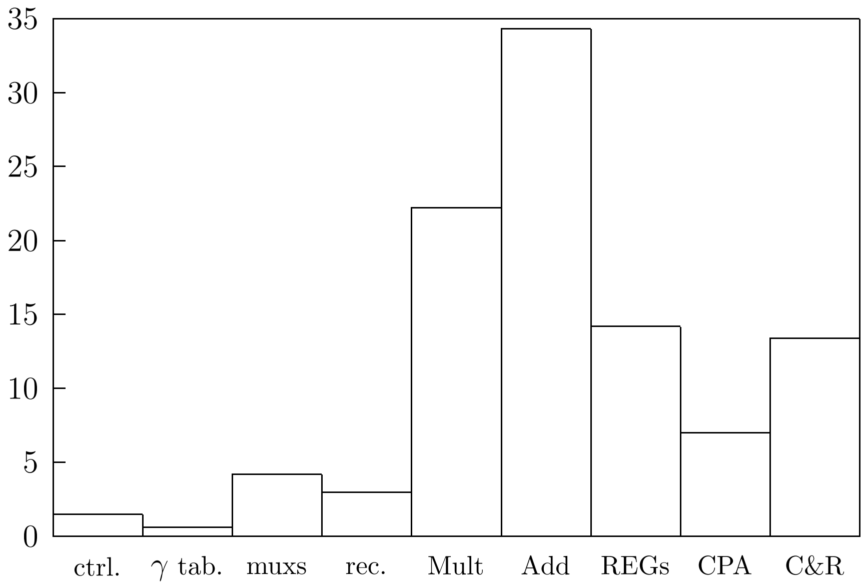


Figure 4.26: Percentage of energy dissipation in basic radix-512 divider.

4.5.2 Low-Power Implementation

For energy reduction in radix-512, we used a different approach than for the lower radices. From Figure 4.26, it is clear that most of the energy (about 60% of the total) is dissipated in the MultAdd block. This is not unexpected because MultAdd is the largest block and consists of several levels of CSAs. As a consequence, the distribution of the energy consumption is quite different from the dissipation in lower radices where, for example, the energy dissipation in the corresponding blocks (MULT and CSA) for radix-4 is less than 25% of the total. For this reason, many of the techniques presented in Chapter 3, which were developed for lower radices, are not very effective for the radix-512 divider.

Retiming by itself only reduces glitches at the input of MULT for lower radices, and in case of radix-512 retiming by itself is not much beneficial for MultAdd because the several levels of the tree of adders produce many glitches anyway.

Changing the redundant representation is a technique designed to reduce the energy dissipation in the registers by eliminating some flip-flops. Because it requires the propagation of the carry within a digit, this technique increase the energy dissipation in the adder, that however, for lower radices, does not offset the reductions obtained in the register. For radix-512, there is not sufficient time to propagate the carry in a $\log_2 512 = 9$ bit adder without increasing the critical path. The use of a radix-8 CSA (a radix-512 CSA could be decomposed in 3 radix-8 CSAs) might reduce the energy dissipated in the registers by about 2% of the whole energy consumption. But this reduction in the registers will be offset by the increase of glitches for the propagation of the carry in MultAdd.

Techniques such as equalizing the paths and using low-drive cells are not very effective for lower radices and are impractical for radix-512.

The techniques that can reduce significantly the energy dissipation in the radix-512 are the modification in the convert-and-round unit, disabling the CPA when not used, and using dual voltage in the recurrence to reduce the energy dissipated in MultAdd.

In addition to the techniques presented in Chapter 3, some work has been done in [45] to reduce the power dissipation in trees of adders. In [45], by using the redundancy in a 4:2 CSA (compressor), different configurations of the compressors are used to reduce the probability of transitions in the tree. However, experimental results in [45], showed that for a large tree of adders such as the one used in a 54×54 multiplier [7], the power savings are about 5%.

Disabling the clock in registers

The first modification applied to the radix-512 divider is to disable the clock in flip-flops that do not change. This is particularly advantageous in register M and Z which change once and three times respectively, per division (Figure 4.24). The reduction in the energy dissipated is about 2.0 nJ corresponding to 3% of the total divider.

Reductions in the CPA unit

The carry-propagate adder (CPA) is used twice during the division. A first time to assimilate the value of z in the third cycle, and a second time in the last cycle to determine the sign of the remainder (and if it is zero). The CPA is switched off by forcing a constant logic value at its inputs when it is not used. The reduction in energy with respect to the basic implementation is about 5%.

Reductions in the convert-and-round unit

In the basic implementation of the radix-512 divider, three registers (Q, QM, and QP) are necessary to store the partial quotient ($\rho = 1$). As explained in Section 3.10, the algorithm is modified as for lower radices, by eliminating the shifting of the digits previously loaded, the two registers QM and QP, and by recoding. Two additional registers are introduced: the 6-bit ring counter C to keep track of the digits to update and the temporary register T (10 bits) used in the recoding. The digit-decrementer, implemented both for the digits of Q and for register T, is a 9-bit ripple decrement-by-one circuit and its delay is about 4.0 ns. Note that the delay of the decrementer does not affects the critical path.

By implementing the modified algorithm in the convert-and-round unit, we reduce the number of flip-flops in the unit from 162 (registers Q, QM, QP) to 70 (registers Q, C, T). The energy consumption in the divider is reduced by about 10%.

4.5.3 Dual Voltage Implementation

Retiming the recurrence

As mentioned above, for radix-512 the purpose of retiming is to limit the critical path to a few bits and use dual voltage. In order to do so, we have to move the operations done on the MSBs (selection in mux2 and recoding) from the beginning of the cycle to the end of the cycle. This can be done as sketched in Figure 4.27.b by introducing an extra register to store the carry-save representation of the recoded operand.

However, the scheme of Figure 4.27 requires an additional initialization cycle to store in register R the recoded value of d_{15} (d_{REC}). An alternative to the addition of the extra cycle is to take advantage of the fact that in cycles 2 and 3 the value to be recoded is the same ($-M$). In this case we can use a multiplexer to divert to MultAdd either the output or register R or directly the output of the recoder, as shown in Figure 4.28. This multiplexer (Mux-R) is controlled by a new signal *DIVERT*, set by the controller, that routes the MultAdd input signals. Table 4.10 shows the values of the signals in the unit in the first 4 cycles. Note that blocks Mux2 and Rec are replicated in the table for clarity.

The new multiplexer Mux-R is now on the critical path and the clock cycle must be lengthed to accommodate the additional 0.5 ns of its delay. The new critical path is shown in Figure 4.29. However, the solution with Mux-R is still advantageous over the solution with an extra cycle. In fact, the number of cycles for the radix-512 division (10) is quite small and the longer clock cycle increases the execution time by about 5 ns that is still a shorter time than one extra clock cycle required for the first solution.

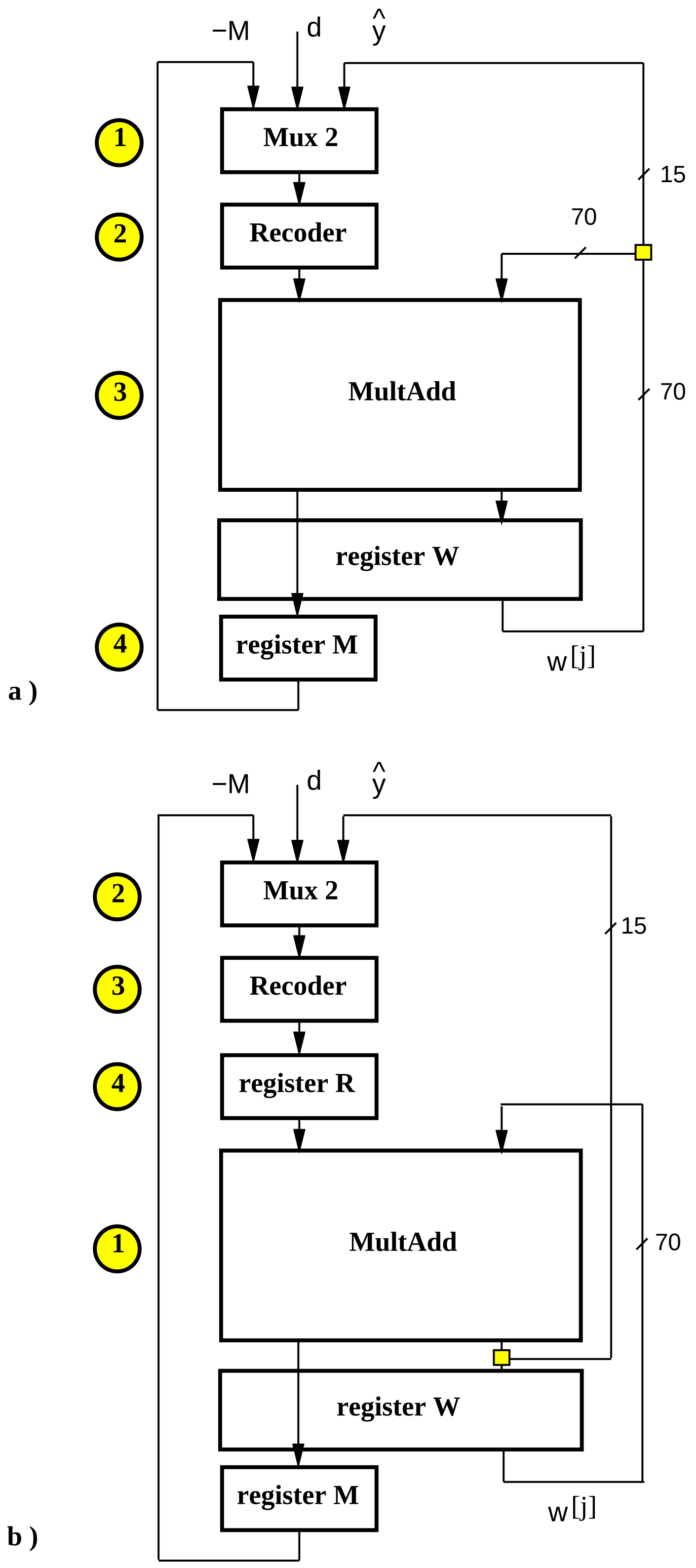


Figure 4.27: Retiming of the recurrence.

	cycles			
	1	2	3	4
REG M	-	$-M$	$-M$	$-M$
REG W	-	-	Md	$w[0]$
Mux2	d	$-M$		
Rec	d_{REC}	$-M_{REC}$		
REG R	-	-	$-M_{REC}$	$\hat{y}[1]_{REC}$
Mux-R	d_{REC}	$-M_{REC}$	$-M_{REC}$	$\hat{y}[1]_{REC}$
MultAdd	$-M$	Md	$w[0] = Mx$	$w[1]$
Mux2			$\hat{y}[1]$	$\hat{y}[2]$
Rec			$\hat{y}[1]_{REC}$	$\hat{y}[2]_{REC}$
DIVERT	0	0	1	1

Table 4.10: Operations and signal values in retimed unit.

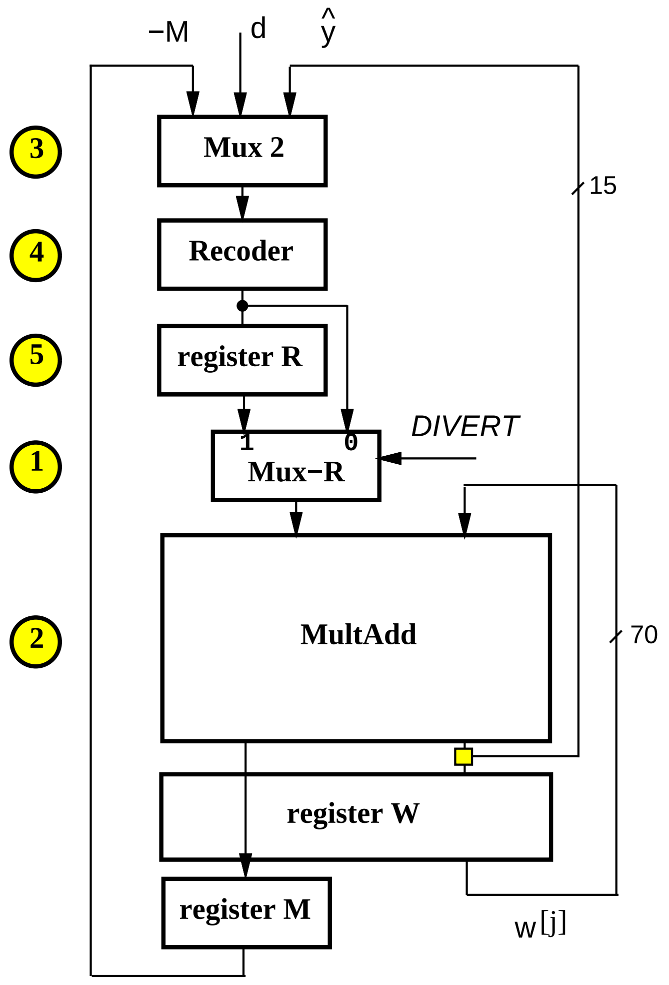


Figure 4.28: Retimed recurrence with Mux-R.

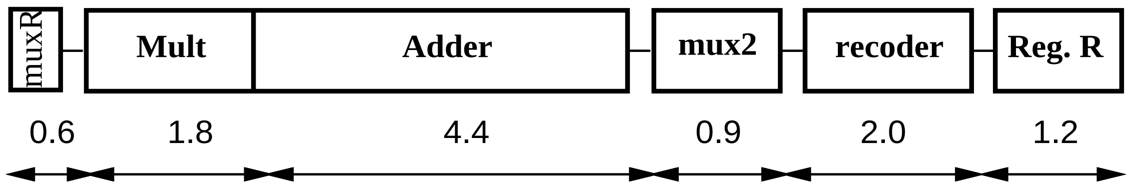


Figure 4.29: Critical path (*ns*) after retiming.

Dual Voltage

After the retiming the 52 LSBs of MultAdd and register W can be redesigned for dual voltage. The time slack for those 52 LSBs is 4.3 *ns* that allows a minimum dual voltage $V_2 = 2.3\ V$.

Furthermore, voltage can be also scaled in the convert-and-round unit resulting in reduction of energy dissipated of 40% with respect to the basic implementation of the divider.

4.5.4 Summary of Results for Radix-512

For the radix-512 divider, the retiming increases the critical path by about 5%. Because we want to reduce the energy without penalizing the performance, in this case, there is a tradeoff between smaller energy and longer delay. Since, as previously discussed, the only technique which reduces significantly the energy consumption in the recurrence is the use of dual voltage, we decided not to apply the other techniques (change redundant representation, using low-drive cells, etc.) and give up performance for small energy reductions. Table 4.11 reports the energy values for the three implementations:

1. *standard* is the basic implementation of Figure 4.23.

2. *low-power* is the implementation with reduced energy dissipation, but same delay as the basic. It is obtained by applying the low-power techniques mentioned above with the exception of retiming.

3. *dual-voltage* is the estimation of the implementation with retiming and dual voltage, but longer execution time.

Optimization with Synopsys Power Compiler was not performed because in the radix-512 divider the selection of the quotient digit is done by rounding.

Figure 4.30 shows the breakdown, as a percentage of the total, of the energy dissipated in the main blocks composing the unit.

blocks	standard nJ	low-power nJ	dual voltage (est.) nJ
control	1.0	1.0	1.0
clk tree	0.5	0.5	0.5
γ table	0.4	0.4	0.4
mux 1	1.1	1.1	1.1
mux 2	0.6	0.6	0.8
mux 3	1.1	1.1	1.1
recoder	2.0	2.0	3.8
Mult	14.5	14.5	8.0
Add	22.5	22.5	12.5
registers W	6.6	5.5	*3.6
register M	0.5	0.3	0.3
register Z	2.3	1.5	1.5
reg. R + mux-R	-	-	1.1
CPA	4.5	1.5	1.5
C&R unit	8.7	2.7	*1.3
Total $[nJ]$	66.5	55.0	38.5
Ratio	1.00	0.85	0.60
Area $[mm^2]$	6.0	6.4	-
T_{cycle} $[ns]$	11.0	11.0	11.5
t_{div} $[ns]$	110	110	115

Values marked * include level shifters.

Table 4.11: Energy-per-division for radix-512.

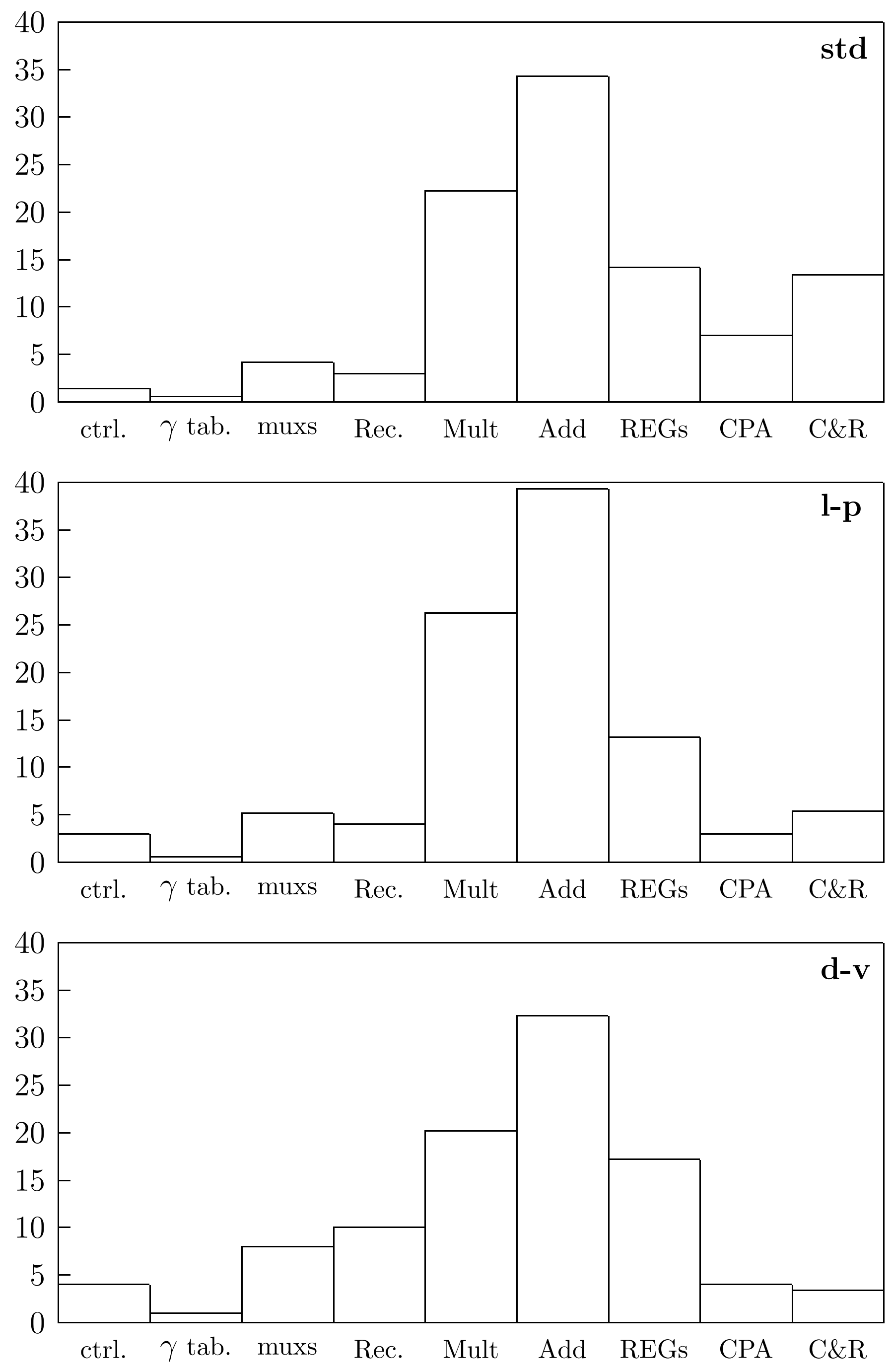


Figure 4.30: Percentage of energy dissipation in radix-512 divider.

4.6 Radix-4 Combined Division and Square Root

4.6.1 Algorithm and Implementation

For radix-4, expression (2.6) and expression (2.7) are rewritten as

$$w[j+1] = 4w[j] + F[j] \qquad j = 0, 2, \dots 26 \tag{4.7}$$

and

$$F[j] = \begin{cases} -q_{j+1}d & \text{(division)} \\ -(S[j]s_{j+1} + \frac{1}{2}\,4^{-(j+1)}s_{j+1}^2) & \text{(square root)} \end{cases} \tag{4.8}$$

The selection function is

$$\begin{aligned} q_{j+1} &= SEL(d_\delta, \hat{y}) \qquad \text{(division)} \\ s_{j+1} &= SEL(\hat{S}[j], \hat{y}) \quad \text{(square root)} \end{aligned}$$

where d and $S[j]$ are truncated after 4 fractional bits, and $\hat{y} = 4w_S + 4w_C$ is truncated after 4 fractional bits. The result digit is in signed-digit representation $\{-2, -1, 0, 1, 2\}$ with redundancy factor $\rho = \frac{2}{3}$ and the residual $w[j]$ is stored in carry-save representation (w_S and w_C) to reduce the iteration time. The value $S[j]$ is obtained by the on-the-fly conversion algorithm. In the on-the-fly conversion, two variables A and B are required. They are updated, in every iteration, as follows:

$$A[j] = S[j] \qquad \text{and} \qquad B[j] = S[j] - r^{-j}$$

The number of bits required in the recurrence is 55 fractional plus 1 integer for a total of 56 bits. To execute the operation 28 cycles are required for the iterations plus one additional cycle for the rounding, for a total of 29 clock cycles. The block diagram of the basic implementation is shown in Figure 4.31.

Implementation of block DSMUX

Block DSMUX selects the inputs to block FGEN according to the operation (division or square root) under execution. If the operation is a division, the value

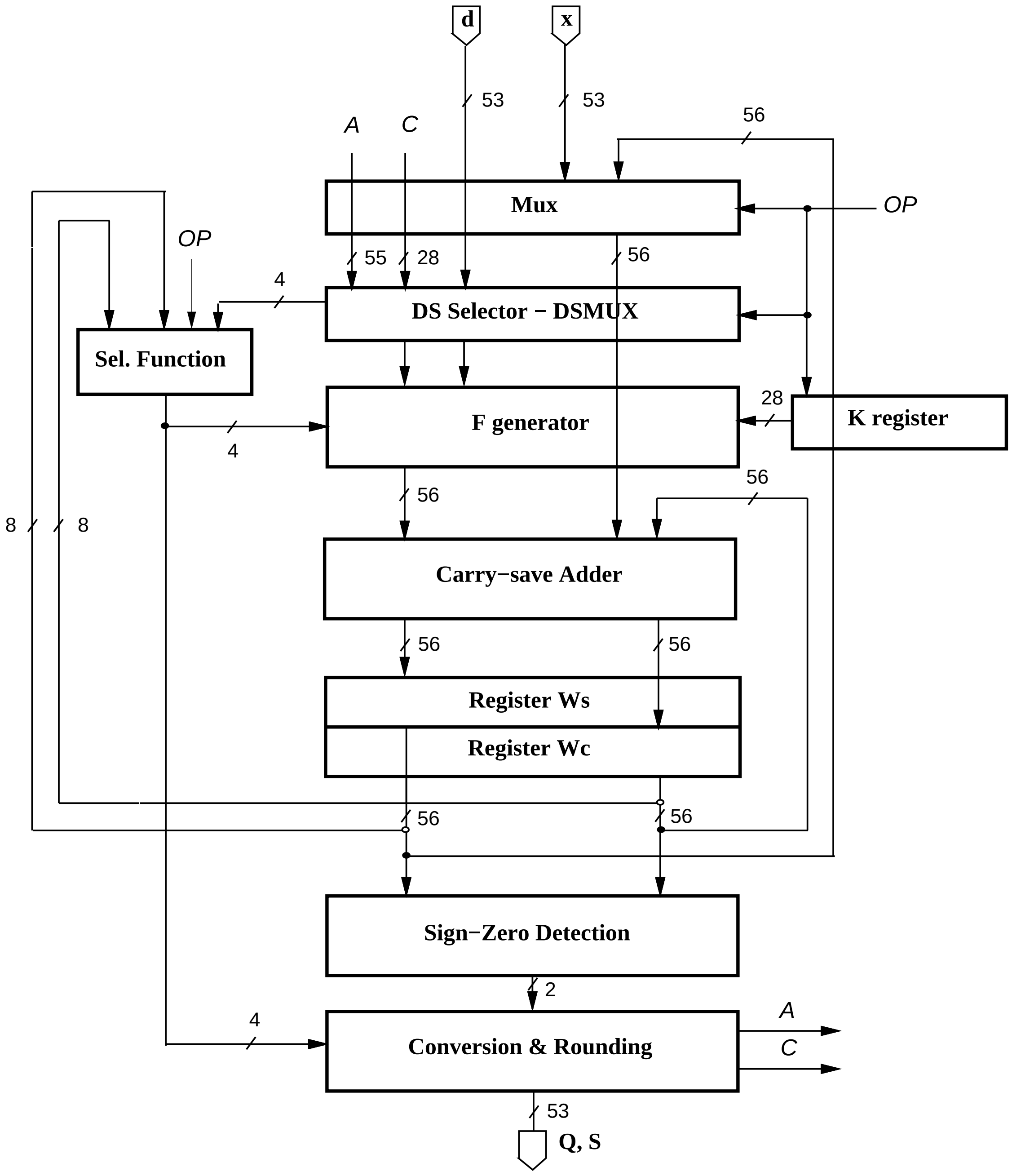


Figure 4.31: Radix-4 combined division/square root unit.

operation	A$_{OUT}$	B	
division	d	d	
square root	A	$\overline{C_i}A_{2i+1} + C_i(A_{2i+1} \oplus A_{2i})$	(odd bits)
		$\overline{C_i}A_{2i} + C_i\overline{A_{2i}}$	(even bits)
		$i = m, \dots, 1, 0$	

Z_i refers to bit in position i in vector Z, being $i = 0$ the LSB.

Table 4.12: DSMUX operations.

to provide to FGEN is d, while for square root the value to provide to FGEN is the partial result $S[j]$, which coincides with $A[j]$. If in the conversion block we implement the algorithm described in Chapter 3 Section 3.10, the value of B can be derived from A and the state in register C. In conclusion, in block DSMUX we utilize the inputs (d, A, and C) to obtain the desired outputs (A$_{OUT}$, B) according to Table 4.12.

Implementation of Selection Function SEL

The selection function can be divided into two parts: an adder and a logic function. The adder is an 8-bit carry-propagate adder whose addends are the 8 MSBs of the carry-save representation of W. The 7 MSBs of the sum are used to generate the result digit at each iteration, along with 3 bits of either d or A chosen as follows:

- if the operation is division the 3 bits of d are those with weight 2^{-2}, 2^{-3}, and 2^{-4}.
- If the operation is square root, the 3 bits are chosen from A, as explained in [10], according to Table 4.13.

The selection logic function is described in Table 4.14. The digit h selected is the one satisfying the expression $m_h \leq \hat{y} < m_{h+1}$. The result digit is in the set $\{-2, -1, 0, 1, 2\}$. This representation makes the F-generator block (FGEN) simpler.

1	0	-	first iteration ($j = 0$)
1	1	1	if ($A_{<0>} = 1$) and ($j > 0$)
$A_{<-2>}$	$A_{<-3>}$	$A_{<-4>}$	if ($A_{<0>} = 0$) and ($j > 0$)

$A_{<-k>}$ refers to bit in A with weight 2^{-k}. $A_{<-k>} = A_{55-k}$

Table 4.13: Bits of A used in SEL.

m_h	d_δ, $\hat{S}[j]$							
	8/16	9/16	10/16	11/16	12/16	13/16	14/16	15/16
m_2	12	14	15	16	18	20	20	22
m_1	4	4	4	4	6	6	8	8
m_0	-4	-5	-6	-6	-6	-8	-8	-8
m_{-1}	-13	-15	-16	-17	-18	-20	-22	-23

Values in table are multiplied by 16

Table 4.14: Selection function for radix-4 combined division/square root.

Implementation of block FGEN

Block FGEN generates the signal $F[j]$ as described in expression (2.7):

$$F[j] = \begin{cases} -q_{j+1}d & \text{(division)} \\ -(S[j]s_{j+1} + \frac{1}{2}\, r^{-(j+1)}s_{j+1}^2) & \text{(square root)} \end{cases}$$

For square root the generation of F is quite complicated [10]. Table 4.15, where *a...aa* and *b...bb* represent bits of $A[j]$ and $B[j]$ respectively, shows the values of the bit-string for the different result digits. To keep track of the position, a 28-bit register (K) is used. The register K is initially loaded with 1 in the MSB and 0 in the remaining positions. At each iteration the 1 is replicated one position to the right according to the following expression.

$$K_i[j+1] = K_{i+1}[j], \qquad K_{27}[j] = 1 \qquad\qquad i = 0, 1, \ldots, 27$$

At the end of the square root all bits of K are 1. To simplify the implementation of F[j], the bit string of Table 4.15 is rearranged as shown in Table 4.16. As can be seen, three zones are defined by the variables

$$K1_i = K_i K_{i-1}\ , \qquad K2_i = K_i \overline{K_{i-1}}\ , \qquad K3_i = K_{i+1}\overline{K_i}$$

and the relation between i and the bit h is $i = \lfloor h/2 \rfloor$. In terms of these variables we obtain

$$\begin{aligned} F_h(\text{odd } h) = \quad & K1_i(P2\overline{A_{h-1}} + P1\overline{A_h} + M1B_h + M2B_{h-1}) \\ & + K2_i(P2 + P1\overline{A_h} + M1B_h + M2)\ +\ K3_i(P1 + M1) \end{aligned}$$

$$\begin{aligned} F_h(\text{even } h) = \quad & K1_i(P2\overline{A_{h-1}} + P1\overline{A_h} + M1B_h + M2B_{h-1}) \\ & + K2_i(P2 + P1 + M1 + M2)\ +\ K3_i(P1 + M1) \end{aligned}$$

For division by setting all the bits of register K to 1 (all $K1_i = 1$ and $K2_i = K3_i = 0$), $F[j]$ can be generated by the same expression as for square root. This corresponds to

$$F_h = P2\overline{A_{h-1}} + P1\overline{A_h} + M1B_h + M2B_{h-1} \qquad h = 0, 1, \ldots, 55$$

with $d = A = B$. Note that for division, when q_{j+1} is positive (subtraction), the carry-in in the adder must be set to 1.

s_{j+1}	$F[j]$	bit-string							
		0		j-1	j	j+1	j+2		27
0	0	00	...	00	00	00	00	...	00
1	$-A[j] - 2 \times 4^{-(j+2)}$	$\overline{aa}$	...	$\overline{aa}$	$\overline{aa}$	11	10	...	00
2	$-2A[j] - 8 \times 4^{-(j+2)}$	$\overline{aa}$	...	$\overline{aa}$	$\overline{a}1$	10	00	...	00
-1	$B[j] + 14 \times 4^{-(j+2)}$	bb	...	bb	bb	11	10	...	00
-2	$2B[j] + 24 \times 4^{-(j+2)}$	bb	...	bb	$b1$	10	00	...	00

Table 4.15: Generation of $F[j]$.

Minimum delay implementation

The minimum delay implementation, also referred as *standard*, is the one obtained with the constraint of minimum delay. The post-layout critical path is 7.3 ns.

s_{j+1}	$F[j]$		bit-string							
			27		$i+1$	i	$i-1$	$i-2$		0
0	0		00	...	00	00	00	00	...	00
1	$-A[j] - 2 \times 4^{-(j+2)}$		$\overline{aa}$	...	$\overline{aa}$	$\overline{a}1$	11	00	...	00
2	$-2A[j] - 8 \times 4^{-(j+2)}$		$\overline{aa}$	...	$\overline{aa}$	11	00	00	...	00
-1	$B[j] + 14 \times 4^{-(j+2)}$		bb	...	bb	$b1$	11	00	...	00
-2	$2B[j] + 24 \times 4^{-(j+2)}$		bb	...	bb	11	00	00	...	00
		K	1	...	1	1	0	0	...	0
		K1	1	...	1	0	0	0	...	0
		K2	0	...	0	1	0	0	...	0
		K3	0	...	0	0	1	0	...	0

Table 4.16: Generation of $F[j]$ with rearranged bit-string.

The critical path for the combined unit is 5% longer than the critical path for the division only unit of Section 4.2. This is mainly due to the more complicated FGEN block. The energy dissipated by this basic implementation is shown in Table 4.19 in column "standard".

4.6.2 Low Power Implementation

In this section we apply the techniques described in Chapter 3 to the standard implementation. Because of the different conversion algorithm required for square root, in the combined unit a low-power conversion and rounding unit, as the one described in Section 4.2, is already in place in the standard implementation. In addition to the techniques of Chapter 3, we reduce the energy dissipation in register K by gating the clock in the flip-flops.

Retiming the recurrence

The retiming is done by moving the selection function of Figure 4.31 from the first part of the cycle to the last part of the previous cycle (see Figure 4.32). The new 4-bit register is initialized to $q_0 = 1$ for division and $s_0 = 1$ for square root.

This retiming causes a problem for the square root operation. The result digit

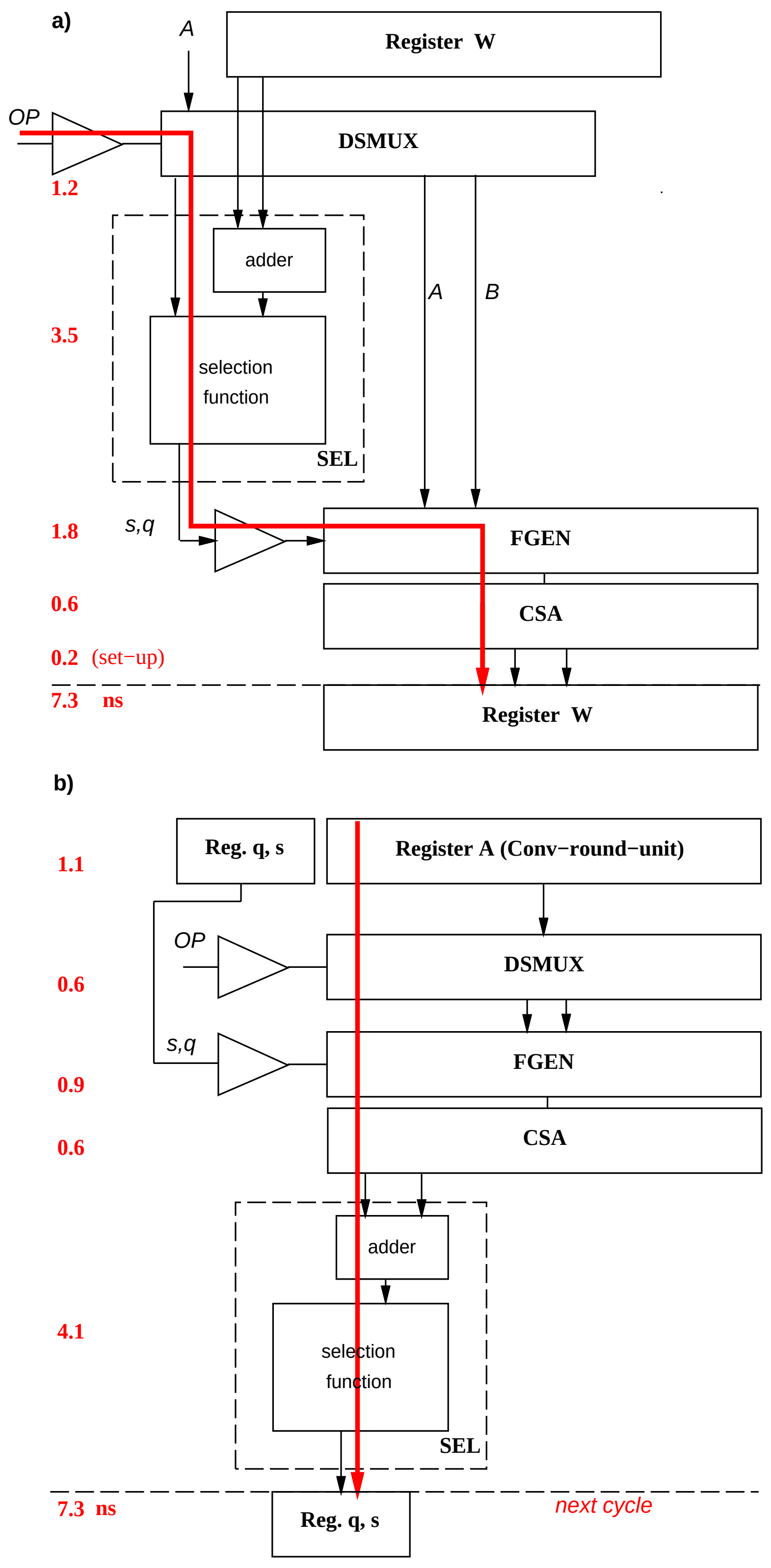


Figure 4.32: Retiming of the recurrence. **a)** before retiming. **b)** after retiming.

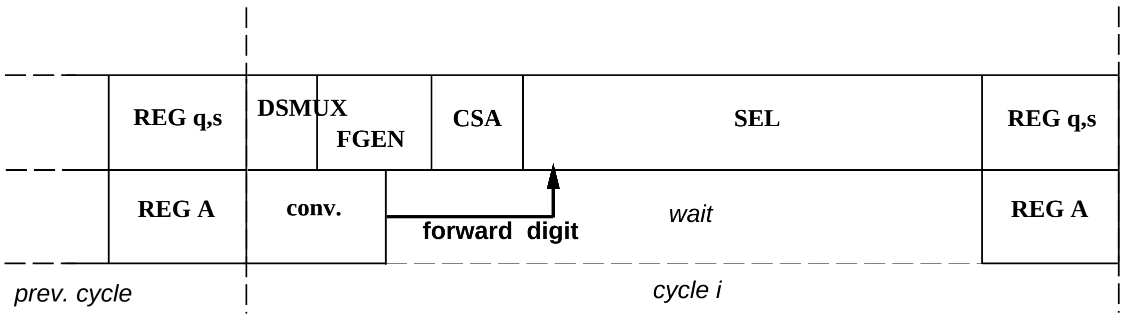


Figure 4.33: Digit forwarding.

j	$\hat{S}[j]$		
1,2	$A_{<0>}$	$A_{<-3>}$	$A_{<0>}$
3	$s_{<0>}$	0	0
4	$A_{<-2>} \oplus s_{SIGN}$	$s_{<1>}$	$s_{<0>}$
5	$A_{<-2>}\overline{(\overline{A_{<-3>}}\ \overline{A_{<-4>}}s_{SIGN})}$	f_1	$A_{<-4>} \oplus s_{SIGN}$
others	$A_{<-2>}$	$A_{<-3>}$	$A_{<-4>}$

$$f_1 = A_{<-3>}\overline{s_{SIGN}} + \overline{(A_{<-3>} \oplus A_{<-4>})}s_{SIGN}$$

Table 4.17: Bits of A used in SEL (retimed).

s_{j+1} is converted in the next clock cycle and, as a consequence, in the first few iterations the value $\hat{S}[j]$ is not available for the selection function (Figure 4.33). However, because the digit-selection is done in the last part of the cycle and the conversion of the previous digit is a short delay operation, we can forward the value of the converted digit from the digit-converter to the the selection function and determine the correct value for $\hat{S}[j]$. By indicating with $s_{<1>}s_{<0>}$ the converted digit and with s_{SIGN} its sign, Table 4.17 shows the modifications in the selection function.

After the retiming, we change the representation of the residual to reduce the number or flip-flops and use low-drive gates in the non critical portion of the recurrence as explained for the radix-4 divider in Section 4.2.

Reduction in register K

Register K is used to generate F[j]. In division the register is initialized to 1 in each bit, and the configuration is not changed for the whole operation. In square root the register is initialized to 1 in the MSB and to 0 in the remaining bits. Every iteration the 1 is propagated to the next bit, for a total of two transitions per flip-flop (one to set the bit to 1, one to reset at the end of the operation). It is convenient to disable the clock for those flip-flops that do not need to be changed in a specific cycle. This is the same modification done for registers A and C in the convert-and-round unit implemented by gating the clock of the flip-flops not used (Section 3.10.2). The enabling function (for the i-th flip-flop) is

$$f_i = OP\ K_{i+1}\ \overline{K_i} \qquad\qquad i = 27, \ldots, 0$$

where $OP = 1$ for square root and $OP = 0$ for division. By implementing this technique the energy dissipation in K is reduced virtually to zero for division and to about one third for square root.

MSBs:	DSMUX	FGEN	CSA	SEL	REG		
	0.6	0.9	0.6	4.1	1.1	=	7.3 ns
LSBs:	MUX	FGEN	CSA		REG		
	1.2	0.9	0.6		1.1	=	3.8 ns

Table 4.18: Paths for MSBs and LSBs in retimed recurrence.

4.6.3 Dual Voltage Implementation

The critical path in the retimed implementation is 7.3 ns. By implementing the LSBs of the recurrence with radix-2 CSAs, the delay in the LSBs is 3.8 ns, resulting in a time slack of 3.5 ns. In this case $V_2 = 2.0\ V$ can be chosen without affecting the latency of the unit. On the other hand, by opting for the use of radix-4 CSAs, the time slack is reduced to 2.6 ns and, consequently, V_2 can be lowered to 2.5 V.

Our estimation showed that the lowest energy is obtained by implementing radix-2 CSAs and $V_2 = 2.0\ V$ for the dual voltage implementation.

4.6.4 Optimization with Synopsys Power Compiler

The synthesized selection function met the set delay constraints (3.0 ns). the reduction in energy dissipation, obtained by incremental compilation with energy dissipation constraints resulted to be about 25%, without increasing the delay.

4.6.5 Summary of Results for Combined Unit

The implementation that consumes a reduced amount of energy is shown in Figure 4.34. Table 4.19 reports the average energy dissipation and area for the standard and low-power implementation. In the table, entry *std* refers to the standard implementation, optimized for speed, entry *l-p* is the low-power implementation with the same delay, entry *d-v* is an estimate of a possible implementation with dual voltage, and entry *sym* is an estimate of *l-p* with selection function optimized by Synopsys Power Compiler.

The energy dissipation for the square root operation is about 10% lower than for the division, on average. This is due to the fact that for square root in every iteration we add $F[j]$, which initially contains many zeros, to the residual $w[j]$.

Some of the low-power techniques used, such as the changed redundant representation, reduce the number of flip-flops in the registers and, consequently, the area.

Figure 4.35 shows the breakdown, as a percentage of the total, of the energy dissipated in the main blocks composing the unit.

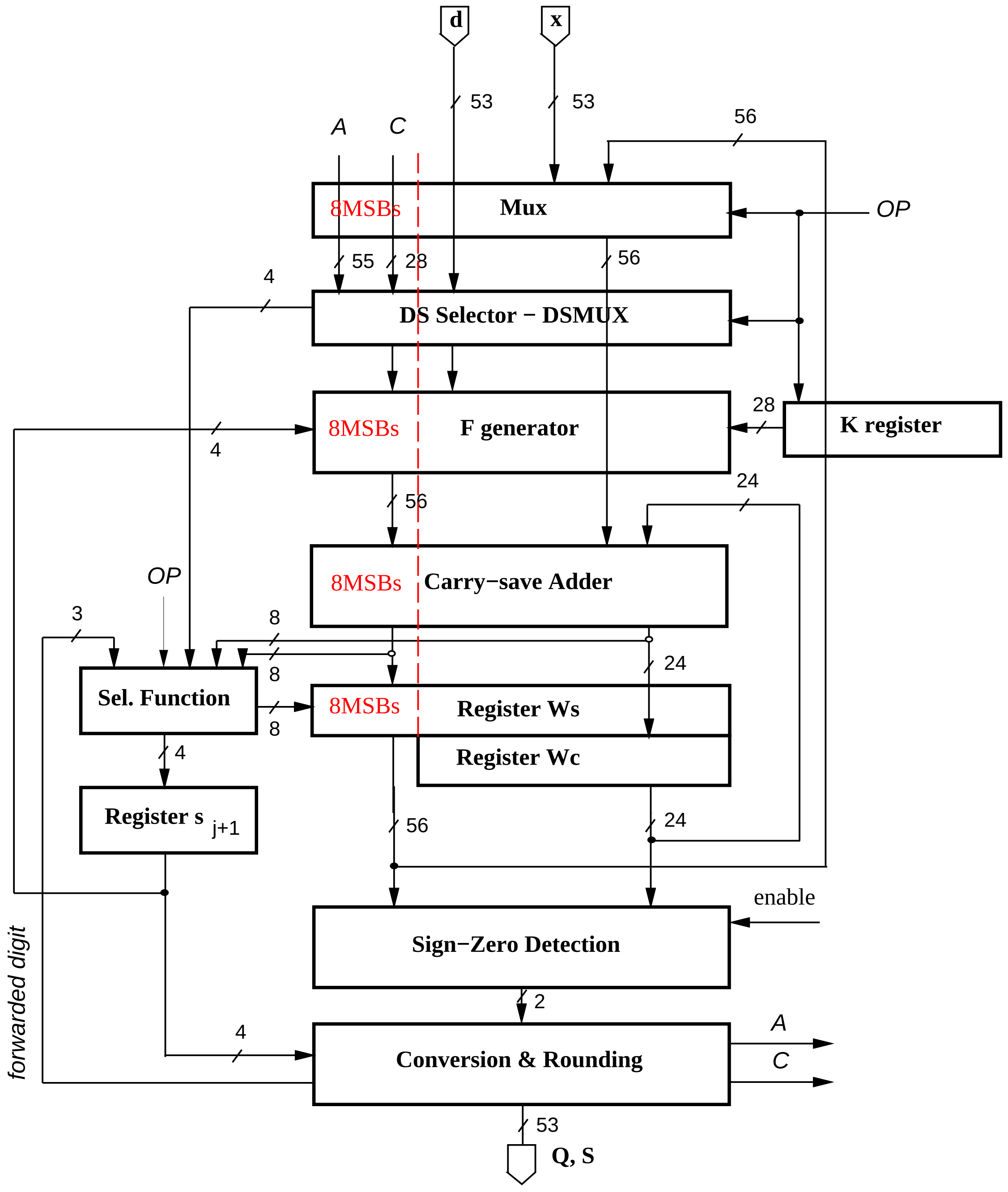


Figure 4.34: Low-power combined division/square root unit.

	division				sqr. root			
blocks	std	l-p	syn	d-v	std	l-p	syn	d-v
	nJ	nJ	nJ	nJ	nJ	nJ	nJ	nJ
control	0.9	0.9		0.9	0.9	0.9		0.9
clk tree	0.8	0.8		0.8	0.8	0.8		0.8
mux	1.9	1.9		0.9	1.7	1.7		0.8
DSMUX	0.1	0.1		0.1	0.3	0.3		0.3
FGEN	7.3	4.9		2.4	5.8	4.3		2.0
CSA	8.8	5.2		3.8	4.7	3.9		2.3
sel. func.	1.5	2.0	1.5	2.0	1.2	1.8	1.4	1.8
register Ws	6.7	6.4		*3.7	6.3	6.1		*3.6
register Wc	6.7	2.7		*3.1	5.1	2.2		*2.5
register q	-	0.3		0.3	-	0.3		0.3
register K	1.3	0.0		0.0	1.6	0.5		0.2
SZD	5.8	0.6		0.6	4.6	0.6		0.6
C&R unit	3.7	3.7		*1.4	3.7	3.7		*1.4
E_{op} $[nJ]$	46.0	29.5	29.0	20.0	37.0	27.0	26.5	17.5
Ratio to std	1.00	0.65	0.63	0.45	1.00	0.75	0.70	0.50
E_{sqrt}/E_{div}	-	-	-	-	0.80	0.90	0.90	0.90
Area $[mm^2]$	1.9	1.8	-	-	1.9	1.8	-	-

Values marked * include level shifters.

Table 4.19: Summary of reductions for division and square root operations.

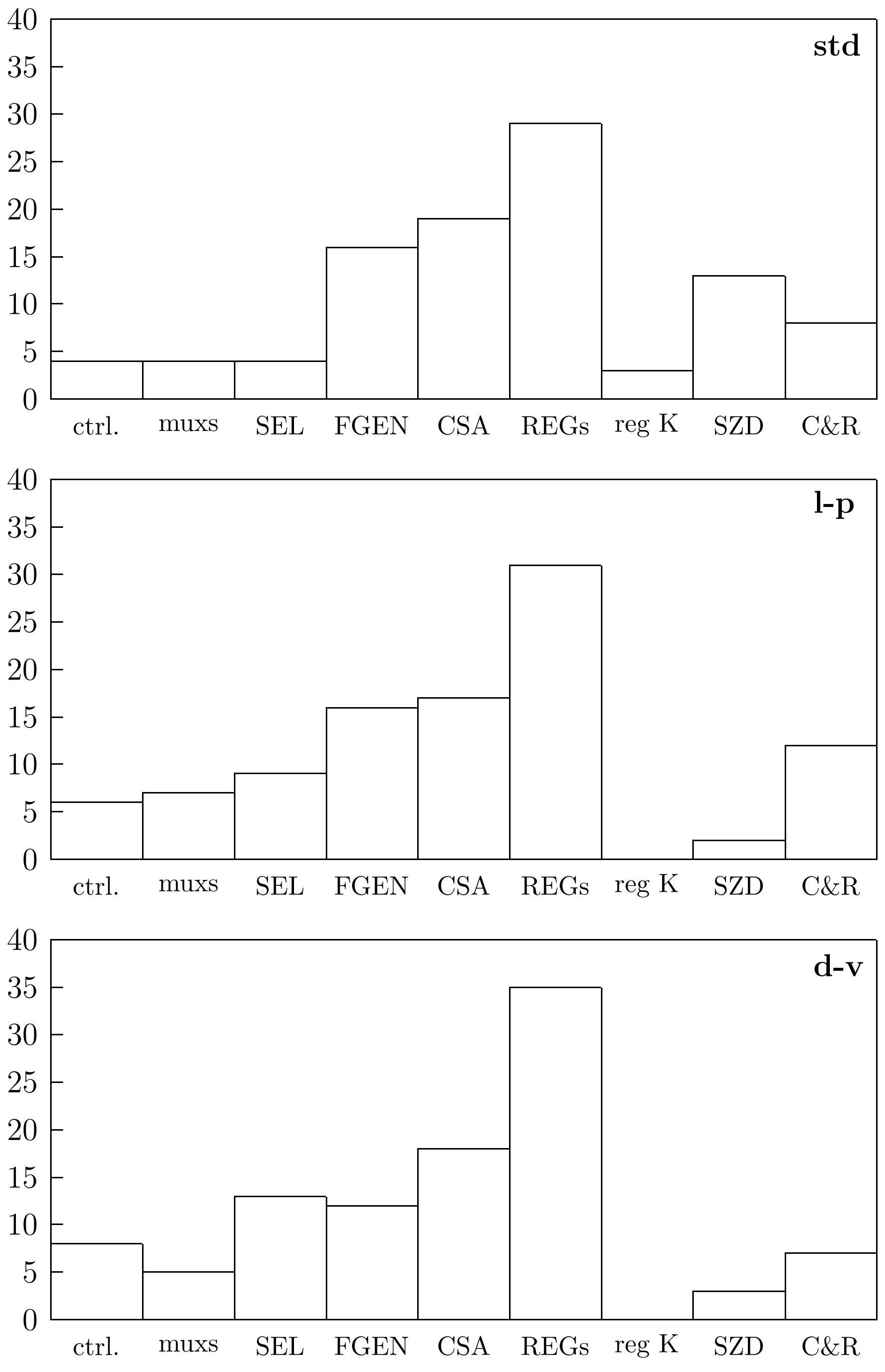


Figure 4.35: Percentage of energy dissipation in radix-4 combined unit.

4.6.6 Energy Comparison with Radix-4 Divider

In this section we compare the results obtained for the radix-4 combined division and square root with those obtained for a radix-4 divider.

Table 4.20 summarizes the results for the implementation *l-p* when performing division. Similar blocks are put in the same row. The combined unit dissipates 15% more than the divider, on average. The largest differences are for the blocks FGEN and "mux". The implementation of FGEN is considerably more complicated than the corresponding unit in the divider and gates with large number of inputs (8-input NAND) have been used to keep the number of levels of logic (and the delay) low. As for the multiplexer, in the retimed implementation of the divider, it is moved out of the recurrence. However, in the divider an extra cycle is required because the first iteration is only used for initialization and no quotient-digit is produced. In the combined implementation in the first iteration we perform the subtraction $x - d$ using two inputs of the CSA and produce the first digit of the result.

4.7 Summary of Estimation Error

The simulations to determine the energy dissipation were carried out on the set of 10 random-generated test vectors shown in Table 4.21. In Table 4.22 we report the percentage errors for the energy estimation of the units presented above and the number of transistors for each implementation. The error is computed by expression (4.2) with confidence level of 99%. Table 4.22 also shows that the accuracy of the estimation is independent of the size of the circuit (number of transistors). This confirms the dimension independence property of this approach which is a common feature of the Monte Carlo methods.

blocks	divider	combined
control	1.1	0.9
clk tree	0.9	0.8
mux	0.3	1.9
DSMUX	-	0.1
FGEN	2.8	4.9
CSA	4.8	5.2
sel. func.	1.6	2.0
register Ws	6.4	6.3
register Wc	3.5	2.7
register q	0.3	0.3
register K	-	0.0
SZD	0.6	0.6
C&R unit	3.9	3.7
E_{div} $[nJ]$	26.0	29.5
Ratio	1.00	1.15
Area $[mm^2]$	1.2	1.8
Ratio	1.0	1.5

Table 4.20: Comparison radix-4 divider/combined unit.

n.	vectors	
1	x	10100101111110100101011011000111010010111111010010110
	d	10100001100111000110110101001001010000110011100011100
2	x	10011001010001101111110010110111001100101000110111111
	d	10001010110110011011011011111110000101011011001101110
3	x	10111110100010110000101111111101011111010001011000011
	d	10011110011101001110011101011011001111001110100111010
4	x	11000101100101010100101101000000100010110010101010011
	d	11000011101101000011110101110110100001110110100001111
5	x	11011001101010000101011001010110101100110101000010110
	d	10110001100101111010011101100100011000110010111101010
6	x	11110110100110110111010101011001111011010011011011101
	d	11000000001110110111011001000100100000000111011011110
7	x	11100011111001001000111010100011110001111100100100100
	d	11111010101100011011001110011100111101010110001101101
8	x	10110111101100001101010101000010011011110110000110110
	d	10100110101011000011111000000111010011010101100010000
9	x	10000110010001100001001111011101000011001000110000101
	d	10001010101011001001001111000010000101010101100100101
10	x	10101111010011101111101010110110010111101001110111110
	d	10110010100000011000011001000111011001010000001100010

Table 4.21: The 10 random vectors.

	standard			low-power		
unit	E_{op} $[nJ]$	ϵ %	FETs	E_{op} $[nJ]$	ϵ %	FETs
radix-4	45.5	2.1	21000	26.0	1.7	16400
radix-8	47.5	4.1	31000	28.5	4.2	24900
radix-16	46.0	2.8	30400	30.0	3.3	25600
radix-512	66.5	2.1	83600	55.0	2.4	68300
comb. div/sqrt	46.0	3.6	25100	29.5	2.7	23800

Table 4.22: Percentage error in energy estimation.

Chapter 5
Evaluation of the Designs

Introduction

In this chapter we provide an overview of the implementations presented in Chapter 4 and comment on the results obtained and on the effectiveness of the techniques.

5.1 Impact of the Energy Reduction Techniques

The impact of the techniques used in the design of low-power division and square root units is summarized in Table 5.1, where they are evaluated in terms of costs and benefits on the three main design constraints: delay, energy and area. For the delay, the cost represents an increase in the critical path and the benefit a reduction in it. For the area the cost and benefits are increase and reduction in the area, whereas for the energy, Table 5.1 lists only the benefits: reduced energy dissipation. The symbol "-" in table means that the corresponding cost/benefit is not affected by that technique. In addition to the traditional design constraints, Table 5.1 also reports the cost in terms of "man-power", which is a measure of the design time needed to implement the technique in question.

It is worth reminding the reader that the results presented in this work are derived from experience in the design of arithmetic units using static CMOS standard cell libraries and automatic floor-planning. By implementing the units in question with different technologies (dynamic CMOS, GaAs, etc.) or using full-custom layout styles, results may be different.

A description of the tradeoffs for each of the techniques presented in Chapter 3 follows.

technique	delay		area		man-power	energy
	cost	benefit	cost	benefit	cost	benefit
retiming	-	low	low	-	high	low
red. in mux	med.	-	-	-	low	high
change repr.	-	-	-	high	med.	high
low-drive gates	-	-	-	low	low	med.
dual voltage	-	-	high	-	med.	high
paths equaliz.	-	-	-	-	high	low
SEL partition	high	-	med.	-	med.	high
glitch filter	high	-	med.	-	med.	med.
C&R algo mod.	-	-	-	high	high	high
gated clock	-	-	med.	-	high	med.
gated tree	-	-	low	-	med.	med.
disable blocks	-	-	high	-	low	high

Table 5.1: Costs and benefits in the application of reduction techniques.

Retiming the Recurrence

The retiming the recurrence is probably the most important and effective technique. Although the benefits of the retiming in itself are moderate, especially for high radices when the increased glitches in the selection function offset the reductions in the multiple generator and carry-save adder, the retiming allow the "decoupling" of the most-significant bits which are on the critical path from the rest of the bits that can be redesigned for low power by applying the other techniques.

The design effort is quite high especially for high radices (radix 8, 16 and 512) in which the retiming alters the critical path.

Reducing the Transitions in the Multiplexer

This modification is relatively easy to implement and gives good reductions in the multiplexer, although it has a smaller impact on the whole unit. However, additional work has to be done by skewing the *select* signal to avoid that the delay of the multiplexer becomes a part of the critical path.

Changing the Redundant Representation

Changing the redundant representation has a high impact on both the energy dissipated and the area. The higher the radix, the higher is the benefit. The tradeoff is that propagating the carry inside the digit increases the number of transitions in the CSA. However, if registers are implemented with edge-triggered flip-flops the extra transitions in the CSA do not offset the reductions in the registers. The critical path is not affected by this techniques unless the delay of the radix-r CSA is too long (e.g. for radix-512).

Using Gates with Lower Drive Capability

Replacing gates not in the critical path with gates which consume less power is relatively easy and can achieve high reductions in the overall energy dissipation. Unfortunately the application of this technique depends highly on the library used. In our library (Passport) the cells with low-drive capability were very limited and the use of this technique not very effective.

Dual Voltage

The use of dual voltage gives probably the highest reduction in the energy consumption because by reducing the voltage the energy decreases quadratically. However, each library is guaranteed to work properly in a given range of power supply voltage (for example library ST CB45000 can operate with voltage between $3.6 - 2.7\ V$) and sometimes the optimal lower voltage V_2 cannot be implemented. Dual voltage requires level-shifters to interface the lower voltage parts with the portions of the circuit at higher voltage. Moreover, in a dual voltage unit the power grid must accommodate three different voltage levels (V_{DD}, V_2 and V_{SS}) and this might complicate the layout of the chip.

Equalizing the Paths to Reduce Glitches

This technique was only adopted in the implementation of the radix-4 divider. It was abandoned in the realization of the other units because the design effort was too high in relation to the benefits. We used automatic floor-planning for the layout to have a fast turn-around time in the realization of many versions, incrementally improved, of the same unit. With automatic floor-planning the cells are placed randomly and the delay due to interconnections is different for each layout. As a consequence, it is impossible to really equalize the paths and the glitches cannot be completely eliminated.

Partitioning and Disabling Selection Function

As already mentioned in Section 3.8, the partitioning of the selection function affects the critical path. However, if the clock period is long enough to accommodate the additional time required, the energy reduction is quite significant especially for high radices.

Glitch Filtering and Suppression

This modification affects the critical path if filtering is positioned at the input of the selection function. This is done for high radices in the retimed implementation. The filtering devices (multiplexers) always increase the area and an extra signal to enable the filter (select input in the multiplexer) has to be generated. Moreover, the technique can be applied to any part of the circuit not in the critical path, where a large number of glitches have to be suppressed, without any penalty on the latency on the unit. However, many select signals require a fine-tuning of the timing of the circuit that could result very hard to implement.

On-the-fly Conversion Algorithm Modification

The modification in the on-the-fly conversion and rounding algorithm brought significant reductions in energy in the convert-and-round unit. The latency of the unit increases with the radix because a digit might be decremented and this is done with a carry-propagate decrementer within a digit. But because the convert-and-round unit is not in the critical path, the modified algorithm can be applied to all the radices (4 through 512) without affecting the performance of the division or square root unit.

Disabling the Clock

This technique is used in the convert-and-round unit not only to reduce the energy dissipated in the flip-flops, but also to allow the loading of the digit in the correct position without the use of a multiplexer. In general, the addition of one or more gates to the clock pin of a flip-flop increases the latency of the circuit. However, in our designs this is only done for registers not in the critical path.

Gating the Trees

For this technique apply the same considerations done for the clock-gating: if the tree is on the critical path, adding a gate increases the latency of the unit. This is not the case of the trees to distribute the signals in the convert-and-round unit, where a significant reduction of the energy dissipated in the unit is achieved.

Switching-off Not Active Blocks

Switching off a block not used for several cycles is probably the easiest modification to implement. However, the block has to be disabled by introducing additional logic gates which increase the area and affect the delay of the unit if the block is on the critical path. The reductions in the energy dissipated are higher for units in which the ratio

$$\frac{\text{cycles block is enabled}}{\text{total cycles per operation}}$$

is smaller. For the SZD block, the ratio is smaller for lower radices.

Synthesis for Low-Power

The experimental results presented in [15] claim that synthesis with Synopsys Power Compiler reduces the power dissipated by about 11% on the average (peak of 66%) for some industrial benchmarks and all the delay constraints are met.

In our small experiment the results obtained are good for relatively small circuits (case of selection functions), while for larger and more complex circuits (radix-4 divider recurrence) not only the power is not reduced much, but also the initial design, optimized for smaller delay, is not as good as attainable by manual design.

For these reasons, we conclude that the use of Synopsys Power Compiler is helpful in solving optimization problems of small functional blocks, but not very effective in reducing delay and power in larger and more complex blocks, such as a divider.

Conclusions

Table 5.1 shows that the modifications done at an higher level of abstraction, such as algorithm modification or change of the encoding, have a larger impact on the energy dissipated than techniques applied a lower level, such as path equalization or glitch filtering. Furthermore, modifications done at higher level of abstraction are more independent of the technology and tools used.

5.2 Results and Comparisons among Radices

Table 5.2 summarizes the results obtained for energy-per-division, area and execution time ($t_{div} = T_{cycle} \times$ cycles) for the implementations of Chapter 4. Note that

		E_{div} $[nJ]$			Area $[mm^2]$		T_{cycle}	cycles	t_{div}
		std	l-p	d-v	std	l-p	$[ns]$		$[ns]$
radix-4		45.5	26.0	16.0	1.4	1.2	7.0	30	210
	ratio	1.00	0.60	0.35				speed-up 1.0	
combined		46.0	29.5	20.0	1.9	1.8	7.3	29	210
radix-4	ratio	1.00	0.65	0.45					
radix-8		47.5	28.5	19.0	2.2	1.8	8.0	20	160
	ratio	1.00	0.60	0.40				speed-up 1.3	
radix-16		46.0	30.0	22.0	2.2	1.8	9.2	16	150
	ratio	1.00	0.65	0.45				speed-up 1.4	
radix-512		66.5	55.0	38.5	6.0	6.4	10.5	10	105
	ratio	1.00	0.85	0.60				speed-up 2.0	

Table 5.2: Energy-per-division, area, execution time and speed-up.

for the combined division/square root unit the number of cycles is one less than for the division only unit. This is due to the different initialization cycle in the two implementations. However, it is possible to change the initialization in the radix-4 divider and reduce the number of cycles to 29. For the implementations of Table 5.2, as the radix increases the cycle time T_{cycle} is longer, but the number of cycles is reduced, and the resulting execution time is shorter. The speed-up, relative to the radix-4 implementation, is the ratio of the execution times

$$\text{speed-up} = \frac{t_{r4}}{t_{div}} \ .$$

The radix-512 divider is the fastest unit and it is about twice as fast as the radix-4 divider.

The main goal of this research work is to reduce the energy consumption in division and square root units without penalizing the performance. Figure 5.1 shows, for each radix, the reductions in the energy dissipation with respect to the "standard" (*std*; symbol $\diamond$ in figure). Label c4 in tables indicates values obtained for the radix-4 combined division and square root unit. For all the radices, with the exception of radix-512, the reduction in energy is around the 60% level for the

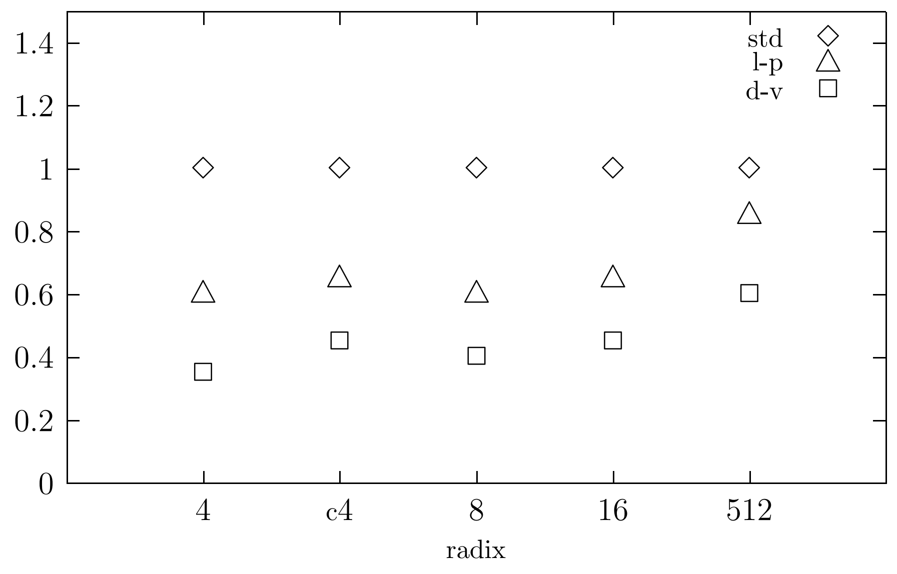


Figure 5.1: Reduction in E_{div}. Ratio to *std* implementation.

low-power implementation (*l-p*; symbol △ in figure), and about 40% for a possible implementation with dual voltage (*d-v*; symbol □ in figure). However, also for the radix-512 divider there is a reduction, although it is smaller.

We now briefly comment on the percentage of energy dissipated in the blocks composing the units, which were presented in Chapter 4. In blocks such as control unit (ctrl) and clock distribution tree (tree), in which energy is not reduced going from the *std* to the *d-v* implementation, although the values of energy in nJ are not changed, the percent contribution to the overall energy dissipation increases. For all radices and schemes, the reductions obtained in the convert-and-round (C&R) unit and by disabling the sign-and-zero detection (SZD) block are quite evident. Blocks in the critical path tend not to reduce their percent contribution to the overall dissipation. In the case of the selection function (SEL), because no techniques are effective to reduce energy without penalizing the critical path, for all the radices there is a percent increase going from the *std* to the *d-v* implementation. This is particularly evident for radix-16 (Figure 4.22 at page 106) where the same energy value for SEL contributes to the 27% of the total of *l-p* and to the 37% of *d-v*. Moreover, for the selection function, due to the increased complexity of the function, the percent contribution to the total grows with the radix: from 11% for *d-v* radix-4 to 37% for *d-v* radix-16. As the radix increases the larger contribution migrates from the registers to the selection function and the hardware to perform the addition (CSAs for radix-8 and 16, Mult and Add for radix-512).

Figure 5.2 and Figure 5.3 show the values of energy-per-division (E_{div}) and energy-per-cycle (E_{pc}), respectively, expressed in nJ. It is interesting to note that, with the exception of radix-512, the units dissipate roughly the same energy to perform a division (Figure 5.2). On the other hand, Figure 5.3 shows that the energy-per-cycle increases with the radix. As it happens for the execution time, the smaller number of cycles for higher radices compensates the higher E_{pc} in $E_{div} = E_{pc} \times$ cycles. However, while for the latency there is a speed-up for higher radices, for energy dissipation there is no improvement. Dividing the values of E_{pc} by T_{cycle} (see expression (1.1)) we obtain the average power dissipation

$$P = \frac{E_{pc}}{T_{cycle}} = V_{DD} I_{ave} \qquad [W].$$

Because T_{cycle} is larger for higher radices, the average power dissipation increases at a slower rate than E_{pc} with the radix (Figure 5.4).

If for a processor low energy is the priority, like for portable electronics where the life time of batteries depends on E_{div}, a high-radix divider with a lower power supply voltage (V_{DD}) and a reduced speed can be used in place of a lower radix divider with same latency. For example, using the data of Table 5.2, a divider with latency of 210 ns can be implemented either with a radix-4 ($E_{div} = 26\ nJ$), or with a radix-16 powered at $V_{DD} = 2.5\ V$ which dissipates about $E_{div} = 18\ nJ$, reducing by one third the energy consumption.

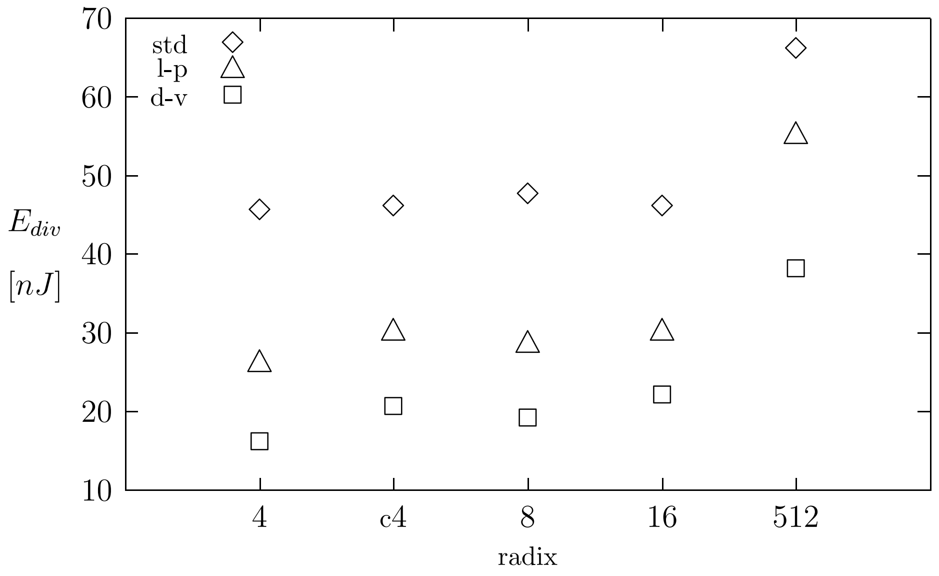


Figure 5.2: Energy-per-division: summary.

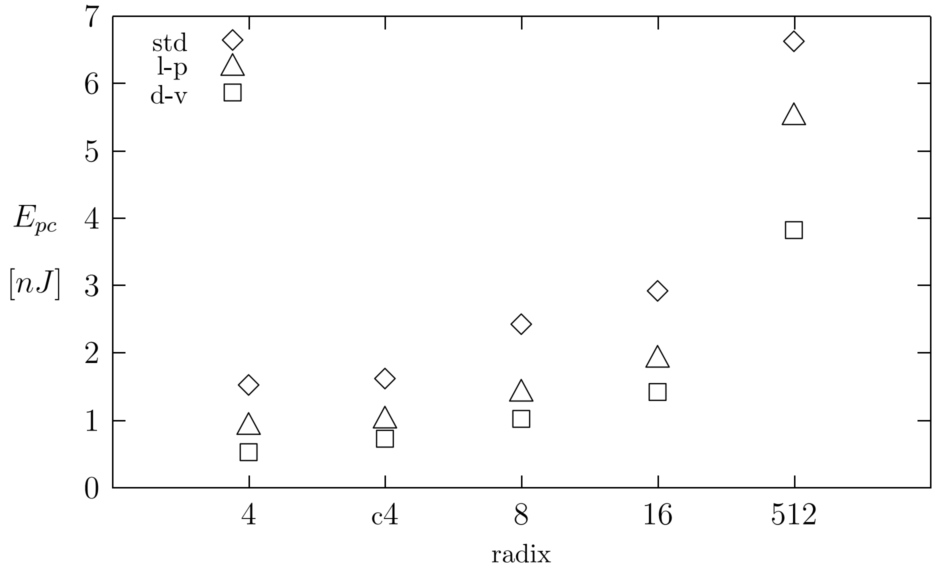


Figure 5.3: Energy-per-cycle: summary.

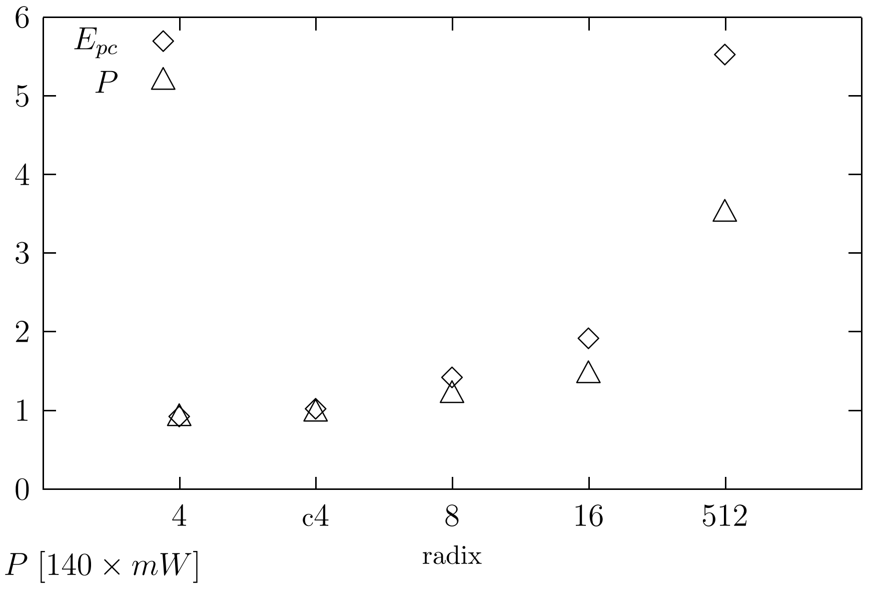


Figure 5.4: Energy-per-cycle and scaled average power for *l*-*p* implementations.

Chapter 6
Conclusions

This work investigated the implementation of low-power double-precision floating-point division and square root units. Although division and square root are not very frequent operations ignoring their implementations can result in system performance degradation. In addition, although division is less frequent than addition and multiplication, because of its longer latency, it dissipates a not negligible portion of the total energy consumed in floating-point units.

Our main objective was to reduce the energy consumption without increasing the execution time and to study the relationship between the radix of the algorithm and the energy consumption. The energy dissipated in CMOS cells can be reduced by applying a number of techniques at different level of abstraction. We both applied already known techniques to the specific case of division and square root, and developed some algorithm-specific modifications that reduce the energy dissipation in the units.

To evaluate the effectiveness of these techniques, we presented the implementation of four different schemes of division and one combined division and square root unit. All the units were implemented with a static CMOS standard cell library. We obtained, for all the radices except radix-512, an overall energy reduction of 40% and estimated that if gates for dual voltage were available in our library we could have reached a reduction of about 60%. Moreover, the energy per operation is roughly the same for radix-4, 8 and 16, and the energy per cycle increases with radix. Because the average power is proportional to the energy per cycle, also the average power dissipation increases with the radix, but to a smaller extent because the cycle time is longer for higher radices. The use of dual voltage is more effective for simple datapaths in which the time slack between the delay of different portions of the circuit is larger.

The results obtained showed that the most effective techniques to reduce the energy dissipation are those applied at a higher level of design abstraction, such as modification in the conversion and rounding algorithm, disabling not active blocks, and the use of dual voltage.

Bibliography

[1] J. Frenkil. A multi-level approach to low-power IC design. *IEEE Spectrum magazine*, pages 54–60, Feb. 1998.

[2] ANSI/IEEE Std 754-1985. IEEE standard for binary floating-point arithmetic, 1985.

[3] S. Oberman and M. Flynn. Design issues in division and other floating-point operations. *IEEE Transactions on Computers*, pages 154–161, February 1997.

[4] J. M. Rabaey, M. Pedram, et al. *Low Power Design Methodologies.* Kluwer Academic Publishers, 1996.

[5] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, et al. Leading-zero anticipatory logic for high-speed floating point addition. *IEEE Journal of Solid-State Circuits*, pages 1157–1164, Aug. 1996.

[6] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, et al. A 286 MHz 64-b floating point multiplier with enhanced CG operation. *IEEE Journal of Solid-State Circuits*, pages 504–513, Apr. 1996.

[7] N. Okhubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nagakome. A 4.4 ns CMOS 54 $\times$ 54 Multiplier using Pass-Transistor Multiplexer. *IEEE Journal of Solid-State Circuits*, pages 251–257, Mar. 1995.

[8] J. L. Hennessy and D. A. Patterson. *Computer Architecture: a Quantitative Approach.* Morgan Kaufmann Publishers Inc., 2nd edition, 1995.

[9] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design.* Addison-Wesley Publishing Company, 2nd edition, 1993.

[10] M.D. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations.* Kluwer Academic Publisher, 1994.

[11] W. Nebel and J. Mermet editors. *Low Power Design in Deep Submicron Electronics.* Kluwer Academic Publishers, 1997.

[12] J. M. Chang and M. Pedram. Energy minimization using multiple supply voltages. *Proc. of International Symposium on Low Power Electronics and Design*, pages 157–162, Aug. 1996.

[13] E. Macii, M. Pedram, and F. Somenzi. High-level power modeling, estimation and optimization. *Proc. of 34th Design Automation Conference*, pages 504–511, June 1997.

[14] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design.* Kluwer Academic Publishers, 1995.

[15] B. Chen and I. Nedelchev. Power compiler: A gate-level power optimization and synthesys system. *Proc. of International Conference on Computer Design (ICCD)*, pages 74–78, Oct. 1997.

[16] A. P. Chandrakasan and R. W. Brodersen. Minimizing power consumption in digital CMOS circuits. *Proceeding of IEEE*, pages 498–523, Apr. 1995.

[17] V. Tiwari, S. Malik, and P. Ashar. Guarded evaluation: pushing power management to logic synthesis/design. *Proc. of International Symposium on Low Power Design*, pages 221–226, Apr. 1995.

[18] L. Benini, P. Siegel, and G. De Micheli. Automatic synthesis of gated clocks for power reduction in sequential circuits. *IEEE Design and Test of Computers*, pages 32–40, Dec. 1994.

[19] T. Lang, E. Musoll, and J. Cortadella. Individual flip-flops with gated clocks for low-power datapaths. *IEEE Transactions on Circuits and Systems*, June 1997.

[20] J. Monteiro, S. Devadas, and A. Ghosh. Retiming sequential circuits for low power. *Proc. of 1993 International Conference on Computer-Aided Design (ICCAD)*, pages 398–402, Nov. 1993.

[21] G. Hachtel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi. Re-encoding sequential circuits to reduce power dissipation. *Proc. of 1994 International Conference on Computer-Aided Design (ICCAD)*, pages 70–73, Nov. 1994.

[22] T. E. Williams and M. A. Horowitz. A zero-overhead self-timed 160-ns 54-b CMOS divider. *IEEE Journal of Solid-State Circuits*, pages 1651–1661, Nov. 1991.

[23] G. Matsubara, N. Ide, H. Tago, S. Suzuki, and N. Goto. 30-ns 55-b shared radix 2 division and square root using a self-timed circuit. *Proc. of 12th Symposium on Computer Arithmetic*, pages 98–105, 1995.

[24] F. Najm. A survey of power estimation techniques in VLSI circuits. *IEEE Transactions on VLSI Systems*, pages 446–455, Dec. 1994.

[25] O. Coudert, R. Haddad, and K. Keutzer. What is the state of the art in commercial EDA tools for low power? *Proc. of International Symposium on Low Power Electronics and Design*, pages 181–187, Aug. 1996.

[26] Synopsys Inc. Power Compiler. http://www.synopsys.com/products/power/.

[27] Sente Inc. WattWatcher/Architect. http://www.powereda.com/.

[28] Israel Koren. *Computer Arithmetic Algorithms.* Prentice-Hall, Inc. , 1993.

[29] S. Oberman and M. Flynn. Division algorithms and implementations. *IEEE Transactions on Computers*, pages 833–854, August 1997.

[30] G.S. Taylor. Radix-16 SRT dividers with overlapped quotient selection stages. *Proc. of 7th Symposium on Computer Arithmetic*, pages 64–71, 1985.

[31] J. Fandrianto. Algorithm for high-speed shared radix-8 division and radix-8 square root. *Proc. of 9th Symposium on Computer Arithmetic*, pages 68–75, Sept. 1989.

[32] M.D. Ercegovac, T. Lang, and P. Montuschi. Very-high radix division with prescaling and selection by rounding. *IEEE Transactions on Computers*, pages 909–918, August 1994.

[33] A. Nannarelli. Implementation of a radix-512 divider. Master's thesis, Univ. of California, Irvine, June 1995.

[34] A. Prabhu and G. Zyner. 167 MHz radix-8 divide and square root using overlapped radix-2 stages. *Proc. of 12th Symposium on Computer Arithmetic*, pages 155–162, July 1995.

[35] K. Usami and M. Horowitz. Clustered voltage scaling technique for low-power design. *Proc. of International Symposium on Low Power Design*, pages 3–8, Apr. 1995.

[36] G. De Micheli. *Synthesis and optimization of digital circuits.* McGraw-Hill, Inc., 1994.

[37] Synopsys. *Synopsys User's Manual.* Synopsys Inc., 1992.

[38] Compass Design Automation. *User Manuals for COMPASS VLSI.* Compass Design Automation, Inc., 1992.

[39] R. Y. Rubinstein. *Simulation and the Monte Carlo method.* John Wiley & Sons, 1981.

[40] C. Z. Mooney. *Monte Carlo simulation.* Sage Publications, 1997.

[41] I. Miller, J. E. Freund, and R. Johnson. *Probability and Statistics for Engineers.* Prentice Hall, 1990.

[42] R. Burch, F. Najm, P. Yang, and T. Trick. A Monte Carlo approach for power estimation. *IEEE Transactions on VLSI Systems*, pages 63–71, Mar. 1993.

[43] Compass Design Automation. *Passport - 0.6-Micron, 3-Volt, High-Performance Standard Cell Library.* Compass Design Automation, Inc., 1994.

[44] ST Microelectronics. *CB45000 series standard cells - databook*. ST Microelectronics, 1997.

[45] P. Larsson and C. Nicol. Transition reduction in carry-save adder trees. *Proc. of International Symposium on Low Power Electronics and Design*, pages 85–88, Aug. 1996.

[46] A. Nannarelli. Report on Error of PET vs. SPICE. *Technical Report*, Oct. 1997. Available at http://www.eng.uci.edu/~alberto/pscripts/an_tech9710.ps.Z.

[47] A. Nannarelli. Short-circuit current modeling for CMOS standard cells energy consumption estimation. *Technical Report*, Feb. 1997. Available at http://www.eng.uci.edu/numlab/archive/pub/nl97p-01/.

[48] K. Anshumali. ACC: automatic cell characterization. *Proc. of Euro ASIC '91*, pages 204–209, May 1991.

[49] A. Nannarelli. Short-Circuit Current Modeling for CMOS Standard Cells Power Characterization. *Technical Report*, Dec 1996. Available on the WWW at http://www.eng.uci.edu/~alberto/pscripts/an_tech9612.ps.Z.

[50] A. Nannarelli. ACC: Automatic cell characterization. Web pages at URL http://www.eng.uci.edu/numlab/ACC/.

Appendix A

Implementation of Blocks Common to Most Radices

Introduction

The functional blocks described in this appendix are those blocks common to most of the implementations presented in this work.

A.1 Register

All the registers are implemented by using arrays of flip-flops. The flip-flops are D-type edge-triggered on the rising edge and include either SET pin, or RESET pin, or both.

A.2 Carry-Save Adder

The radix-2 carry-save adder is implemented as an array of full-adders. Each full-adder (FA) is implemented as depicted in Figure A.1 and it can be decomposed into two half-adders (HA). Its maximum delay is the delay of the two XOR gates, or half-adders ($t_{FA} = t_{HA} + t_{HA}$).

A.3 Selection Function

The selection function (SEL), except for radix-512, is usually composed by a small carry-propagate adder, because of the carry-save representation of the residual, and by a function implemented with logic gates as depicted in Figure A.2. The implementations of SEL are obtained by synthesis of the VHDL description of the

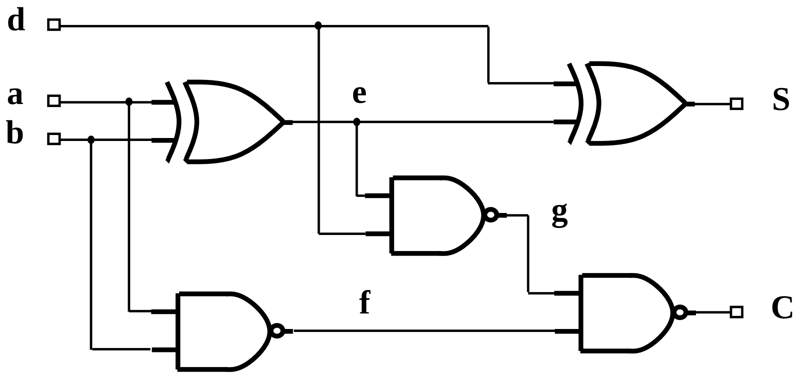


Figure A.1: Implementation of full-adder.

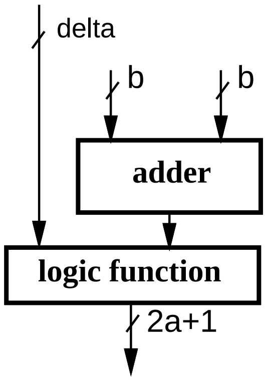


Figure A.2: Selection function.

selection function. SEL includes both the assimilation of the carry-save representation of $\hat{y}$ and the actual digit-selection function.

A.4 Multiple Generator

The multiple generator (MULT) perform the following operation for division:

$$-q_{j+1}\, d\ .$$

In order to avoid the implementation of a complicated multiple generator, the quotient digit is represented in a 1-out-of-h code. In this work, most of the result-digits are represented as signed-digit numbers with values in the set $\{-2, -1, 0, 1, 2\}$. Four signals ($h = 4$) are used to represent these five values with the code given in Ta-

digit	M2	M1	P1	P2
-2	1	0	0	0
-1	0	1	0	0
0	0	0	0	0
1	0	0	1	0
2	0	0	0	1

Table A.1: Result digit encoding.

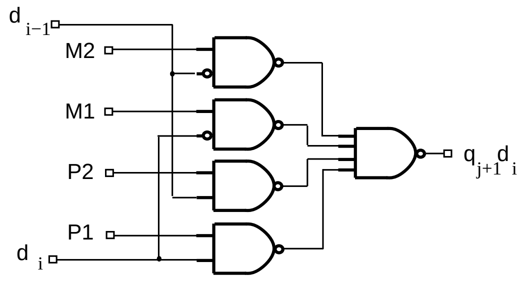


Figure A.3: One bit of the multiple generator.

ble A.1. This representation makes the multiple generator simple, as shown in Figure A.3.

A.5 Sign-and-Zero Detection Unit (SZD)

To perform the rounding, it is necessary to detect the sign of the residual from its redundant representation and to determine if the residual is zero. In [10], a network to detect the two conditions: sign of residual, and residual is zero, is described. We now summarize its implementation. Let w_S and w_c be the values of the (h+1)-bit carry-save representation of the last residual. We introduce two quantities a_S and a_C such that

$$a_S + a_C = w_S + w_C - 2^{-h}$$

and consequently, the condition $w_S + w_C = 0$ results in

$$a_S + a_C = 2^{-h}$$

Therefore, the final residual is zero when:

$$zero = \prod_{i=0}^{h} P_i = \prod_{i=0}^{h} a_{Si} \oplus a_{Ci} \tag{A.1}$$

where a_{Si} and a_{Ci}, which assume either value 1 or 0, represent the bits in position i in the carry-save representation. The sign can also be detected by using a_{Si} and a_{Ci} by observing that:

$$a_S + a_C \geq 0 \quad \Rightarrow \quad w_S + w_C > 0$$

and

$$a_S + a_C < 0 \quad \Rightarrow \quad w_S + w_C \leq 0$$

Therefore:

$$sign = (a_{S0} \oplus a_{C0} \oplus c_{MSB})\ \overline{zero}$$

where c_{MSB} is the carry into the most-significant bit.

The subtraction of 2^{-h} to the carry-save representation of w is done by adding a (h+1)-bit vector of 1s. The resulting expression for the bits of a_S and a_c are

$$a_{Si} = \overline{(w_{Si} \oplus w_{Ci})} \text{ and } a_{Ci+1} = w_{Si} + w_{Ci} \tag{A.2}$$

The P_is of expression (A.2) are generated in a hierarchical way using a carry-look-ahead structure. For example, for a 64-bit sign-and-zero detection unit using groups of 4 bits we have the scheme of Table A.2. And the two corresponding expressions for zero and sign are:

$$zero = P\overline{G}$$

and

$$sign = (G \oplus p_{63})\overline{P}\ .$$

Level 0		
	$g_i = a_{Si}a_{Ci}$ and $p_i = a_{Si} + a_{Ci}$	$i = 0, 1, \ldots, 63$
Level 1		
	for each $j = \lfloor \frac{i}{4} \rfloor$ and corresponding g_k, p_k with	$k = i \pmod 4$
	$G_j = g_3 + g_2p_3 + g_1p_2p_3 + g_0p_1p_2p_3$	$j = 0, 1, \ldots, 15$
	$P_j = p_0p_1p_2p_3$	
Level 2		
	for each $l = \lfloor \frac{j}{4} \rfloor$ and corresponding G_k, P_k with	$k = j \pmod 4$
	$G^*_l = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$	$l = 0, 1, 2, 3$
	$P^*_l = P_0P_1P_2P_3$	
Level 3		
	$G = G^*_3 + G^*_2P^*_3 + G^*_1P^*_2P^*_3 + G^*_0P^*_1P^*_2P^*_3$	
	$P = P^*_0P^*_1P^*_2P^*_3$	

Table A.2: Carry-look-ahead tree for 64-bit SZD.

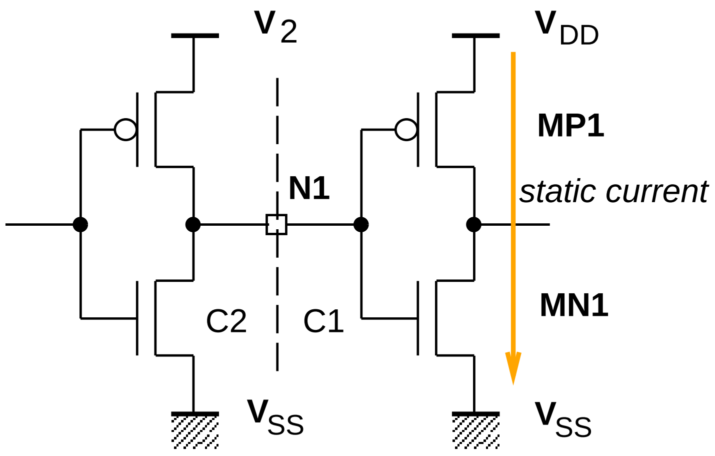


Figure A.4: Dual voltage: C1 is not cut-off.

A.6 Voltage Level Shifter

In this section we describe the voltage level shifter presented in [35]. Voltage level shifters are needed in circuits that operate with dual voltage (V_{DD} regular supply voltage and V_2 reduced supply voltage). Level shifters are necessary when a portion of the circuit at voltage V_2 is connected to a portion at voltage V_{DD}. As shown in Figure A.4, if the output of a circuit operating at V_2 (C2) is connected directly to the input of a circuit operating at V_{DD} (C1), static current flows in C1 at the input level "high". Since the voltage of node N1 is not raised higher than V_2, the p-transistor MP1 cannot be cut-off if $V_2 < V_{DD} - V_{threshold,p}$. Therefore, static current flows from V_{DD} to V_{SS} through MP1 and MN1. In order to block this static current a voltage level shifter is inserted at node N1. No level shifting is necessary when, in the reversed case, the output of a V_{DD} operated circuit is connected to the input of a V_2 circuit. The voltage level shifter is realized as depicted in Figure A.5. Table A.3 indicates the input-output delays and energy consumption for a level shifter operating at $V_{DD} = 3.3\ V$ and $V_2 = 2.0\ V$, and its comparison with an inverter of the Passport library. The values in Table A.3 were obtained by SPICE simulation.

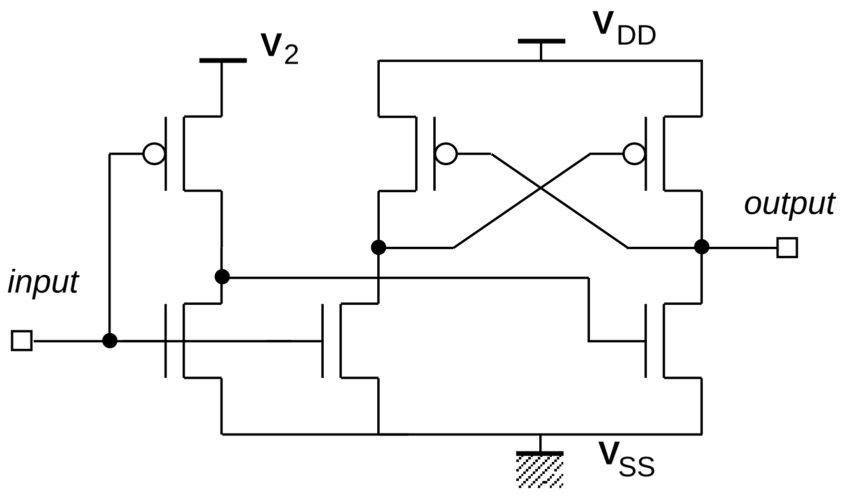


Figure A.5: Voltage level shifter.

	level shifter					inverter				
	delay $[ns]$				E_{tran}	delay $[ns]$				E_{tran}
	t_{LH}	t_{rise}	t_{HL}	t_{fall}	$[nJ]$	t_{LH}	t_{rise}	t_{HL}	t_{fall}	$[nJ]$
SL1	0.144	0.13	0.042	0.11	0.7	0.097	0.20	0.094	0.16	0.3
SL4	0.245	0.17	0.087	0.22	1.2	0.164	0.32	0.163	0.27	0.8
SL16	0.670	0.45	0.271	0.69	3.4	0.459	0.98	0.476	0.86	2.1

SL = standard load = 22 fF for Passport library

Table A.3: Delay and energy comparison between level shifter and inverter.

Appendix B
CAD Tools

Introduction

In this appendix we describe the features of some CAD tools used in the realization of this work. A brief description of COMPASS tools is given in Chapter 4. First, the two tools developed in our laboratory (PET and ACC) are presented. Then, the main features of the commercial tool Synopsys Power Compiler are summarized.

B.1 PET: Power Evaluation Tool

PET belongs to the category of power estimators loosely-coupled with the simulator. It is coupled with COMPASS *Qsim* and it was developed internally for two main reasons:

- to have a flexible tool which could be tailored for specific issues.
- because when the project started there were no commercial tools adaptable to COMPASS without a considerable effort.

PET computes the energy and power dissipation by reading the *energy views* for the cells in the library, the layout-extracted netlist and the trace file generated by *Qsim*. The *energy views* are computed once for a given library, by characterization using ACC (Section B.2), and then stored in a database.

B.1.1 PET Energy and Power Models

As discussed in Section 1.2, the energy consumption in a cell is proportional to the output load, the supply voltage, the number of output transitions in a given time window and the energy dissipated internally. This is summarized by expression (1.5), which is rewritten below

$$E_i = (\ \frac{1}{2} V_{DD}^2 C_L + E^{int} \ ) \ n_i$$

where:

V_{DD} is power supply voltage.

C_L is the total load applied to the output.

E^{int} is the internal energy dissipated in the cell during one transition.

n_i is the number of transitions at the output of the i-cell in the time window.

The term between parenthesis

$$E_{tran} = \frac{1}{2} V_{DD}^2 C_L + E^{int} \qquad [J]$$

represents the energy per transition. The average power dissipated in a cell can be computed from the energy, by introducing the following quantities:

f_0 is the circuit main frequency (clock frequency),

a_i is the activity factor:

$$a_i = \frac{\text{nr. of output transitions (in time window)}}{\text{nr. of clock cycles (in time window)}} = \frac{n_i}{n_T}$$

as

$$P_i = (\ \frac{1}{2} V_{DD}^2 C_L + E^{int} \ ) \ a_i f_0 = E_i \frac{f_0}{n_T} \qquad [W]$$

In a sequential cell also the internal switching, not affecting the cell's output, dissipates energy. To take into account this contribution we can write the energy and power expressions in the following way:

$$E_i = (\ \frac{1}{2} V_{DD}^2 C_L + E^{int} \ ) \ n_i + E^{cl} {n_i}^{cl} \qquad [J]$$

$$P_i = (\ \frac{1}{2}V_{DD}^2 C_L + E^{int}\ )\ a_i f_0 + E^{cl} f_i^{\ cl} \qquad [W]$$

where :

E^{cl} is the energy dissipated internally per transition due to clock switching.

$f_i^{\ cl} = \frac{n_i^{\ cl}}{n_T} f_0$ is the frequency of the transitions of the cell's clock[1].

Now we consider a large circuit containing N cells, N_S of which are sequential. The total energy consumption in the time window is given by:

$$E_{total} = \sum_{i=1}^{N} (\frac{1}{2}V_{DD}^2 C_{Li} + E_i^{\ int}) n_i + \sum_{i=1}^{N_S} E_i^{\ cl} n_i^{\ cl} \qquad [J] \qquad \text{(B.1)}$$

Summarizing, in order to calculate the energy dissipated, given by expression (B.1) we need to determine the value of the following parameters:

- V_{DD} is the power supply voltage.
- C_{Li} is the load at the output of the i-cell.
- $E_i^{\ int}$ is the energy per transition dissipated inside the i-cell.
- n_i is the number of transitions seen at the output of the i-cell.
- $E_i^{\ cl}$ is the energy dissipated internally in the sequential i-cell due to clock switching.
- $n_i^{\ cl}$ is the number of the clock transitions seen at the input of the sequential i-cell.

To compute the power dissipation

$$P_{total} = \frac{f_0}{n_T} \sum_{i=1}^{N} (\frac{1}{2}V_{DD}^2 C_{Li} + E_i^{\ int}) n_i + \frac{f_0}{n_T} \sum_{i=1}^{N_S} E_i^{\ cl} n_i^{\ cl} = \frac{f_0}{n_T}\ E_{total} \qquad [W]. \quad \text{(B.2)}$$

we need the two additional values

[1]There are 2 transitions per clock period. Therefore, $f_i^{\ cl}$ is twice the frequency of the cell's clock.

- f_0: the clock frequency.

- n_T: the number of clock cycles in the time window we are considering.

The quantities V_{DD}, E^{int} and E^{cl} depend on the library that we are using. C_L depends on the design and layout (type of cell connected and wire capacitance) and the number of transitions depends on the design and on the set of input vectors used.

The procedure to determine the energy and power dissipation is the following:

1. For the chosen library determine the quantities E^{int} and E^{cl} for each cell. These values can be provided directly by the silicon vendors or obtained by cell characterization.

2. From the layout, extract the capacitance (output load plus interconnection capacitance) at each node and associate them as output load (C_L) to each cell.

3. Run a simulation on a set of random chosen test vectors using a tool that is able to detect transitions (i.e. a logical level simulator).

4. Calculate energy and power using expression (B.1) and expression (B.2).

B.1.2 PET Implementation

The procedure described above was implemented in PET. It consists of three C routines (**analyze**, **ttgen** and **calpot**) and the use of two COMPASS tools: *Qsim* (logic-level simulator) and *extract* (COMPASS Interconnect layout to netlist extractor) [38]. The latter is used to determine the capacitance (including wires) at each node of the circuit while *Qsim* is used to determine the logic values of the nodes

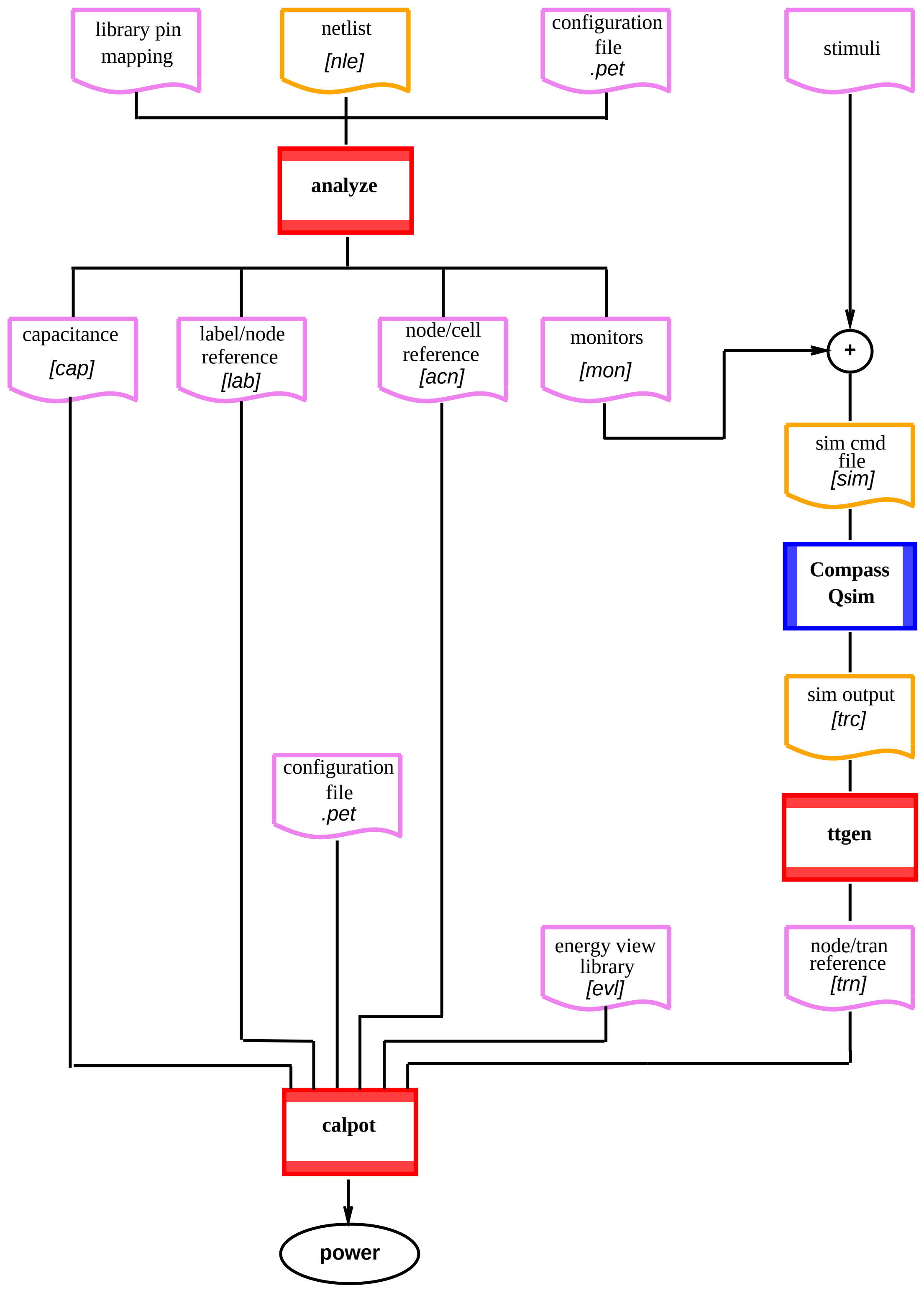


Figure B.1: Structure of PET.

used later to determine the number of transitions. PET is structured as depicted in Figure B.1.

analyze reads the extracted netlist and determines the output load for each cell of the circuit. It also provides to *Qsim* the labels of the nodes to monitor. The files read are:

- a configuration file containing general parameters such as: power supply voltage (V_{DD}), clock frequency, time window of the simulation.
- the netlist [*nle*] extracted by *extract*.
- a file containing the mapping of cell's pins for the library. It is needed to associate the capacitance of node x to the output of cell i.

The files produced are:

- a list of the labels (file [*mon*]) corresponding to the nodes to be monitored by *Qsim*.
- the reference capacitance-node (file [*cap*]).
- the reference cell's output-node (file [*acn*]).
- the reference label-node (file [*lab*]).

All these references are resolved later by **calpot**. The [*mon*] file is incorporated with the input stimuli in the simulation file [*sim*] to be used along with the netlist [*nle*] in the simulator.

ttgen (transitions table generator) reads the simulation output file [*trc*] and creates a transitions table [*trn*]. In this table each label/node is associated with the number of transitions occurred at that node during the simulation.

Finally, **calpot** calculates energy and power dissipation according to expression (B.1) and expression (B.2). The files read are:

- the three files generated by **analyze**: [*cap*], [*acn*], [*lab*].
- the transitions table file [*trn*] produced by **ttgen**.
- the configuration file containing library parameters.
- a file containing the values E^{int} and E^{cl} (*energy views*) for each cell of the library.

B.1.3 PET Testing

PET was tested on a limited set of benchmarks comparing the results with those obtained using SPICE and calculating the power as the product of the voltage and the average current over a time window of the same size of that used for PET [46]. The error was never greater than 10% (the largest benchmark circuit contained about 3,000 transistors).

The main drawback of PET is that it accounts for a fixed amount of short-circuit current for each cell, determined independently of the transition time. This can lead to a lack of accuracy in some situations, for example the power dissipation of blocks not in the critical path where signals could have slow ramps. An approach to include a more accurate evaluation of the short-circuit current is described in [47]. However the improvement in the results obtained is not good enough to justify a significantly greater modeling effort.

B.2 ACC: Automatic Cell Characterization

As an increasing number of transistors is packed in a single chip, the design tools (CAD tools) have to handle larger circuits. Because it is unrealistic to simulate the behavior of a complete system with an electrical-level simulator, such as SPICE, design tools are shifting toward higher levels of abstraction. These levels of abstraction are organized in a hierarchical structure with circuit/electrical level at the bottom of the hierarchy. Circuit characterization is necessary to provide information of the electrical properties of small functional parts of the system to higher hierarchical levels. In general, cell characterization provides capacitance, timing and power values for all the cells in the library to CAD tools operating at gate-level. In our specific case, we characterize the standard cell library to extract the *energy views* necessary for PET.

B.2.1 ACC Energy Views

ACC (Automatic Cell Characterization) is a tool that performs library characterization by automatically running several SPICE simulations on all the cells of the library. It is derived from the tool presented in [48], and can characterize cells for timing, capacitance and energy. However, in this appendix, we only focus on characterization for energy.

As described in Section B.1, the PET energy model for a single cell is

$$E = \left(\frac{1}{2} V_{DD}^2 C_L + E^{int} \right) n_i + E^{cl} n_i^{cl}$$

where:

V_{DD} is power supply voltage.

C_L is the total load applied to the output.

E^{int} is the internal energy dissipated in the cell during one transition.

n_i is the number of output transitions in the time window.

E^{cl} is the energy dissipated internally due to clock switching.

n_i^{cl} is the number of clock transitions, if the cell is sequential.

Of all the quantities indicated in the above expression, the ones obtained by characterization are E^{int} and E^{cl} (*energy views*).

It is convenient to characterize a cell over a period of time in which two output transitions occur (one low-to-high and one high-to-low). The value of energy is computed as the product of V_{DD} and the value obtained by numerical integration of the current $i(t)$ over a time window $[t_1, t_2]$ in which two transitions occur:

$$E_{cy} = \int_{t_1}^{t_2} v(t)\ i(t)\ dt \simeq V_{DD} \sum_{k=0}^{N} i(t_1 + k\Delta t) \qquad \text{with } \Delta t = \frac{t_2 - t_1}{N}$$

The graph of the current $i(t_1{+}k\Delta t)$ is obtained by SPICE simulation with resolution step Δt. By simulating the cell with different loads we determine different values of E_{cy}. The value E^{int} can be obtained, as follows:

1. By linear curve fitting of the values of C_L and E_{cy}, we obtain the two coefficients x_1 and x_0

$$E_{cy} = x_1 C_L + x_0\ .$$

2. From expression (1.5), we get:

$$E_{cy} = (\ \frac{1}{2} V_{DD}^2 C_L + E^{int}\ )\ n_i = 2\ (\ \frac{1}{2} V_{DD}^2 C_L + E^{int}\ )$$

3. By combining the two expressions above:

$$V_{DD}^2 C_L + 2E^{int} = x_1 C_L + x_0$$

   we obtain:

$$V_{DD}^2 = x_1 \qquad \text{and} \qquad E^{int} = \frac{x_0}{2}$$

   Note that the value of x_1 could be used to evaluate the accuracy of the linear curve fitting, being the actual value of V_{DD} known.

For sequential cells, the contribute due to the clock switching E^{cl} is measured, independently of the output load, by applying an input pattern that causes no output transitions (i.e. $n_i = 0$).

Note that the internal energy includes the energy due to short-circuit current which depends on the slope of the transitions. In our characterization for PET, we assumed the input slope to be constant for the library and chosen as the response time af a gate with drive strength of one [43], [49]. This assumption leads to accurate energy values when the circuit is optimized for timing. In fact, longer transition times reflect on longer delays. More detailed information on the characterization of energy due to the short-circuit current is provided in [47].

B.2.2 ACC Implementation

The structure of ACC is shown in Figure B.2. ACC reads three databases containing the SPICE netlists of the cells in the library, a set of loads (CapLib), and different waveforms to be applied as input stimuli (WaveLib). In addition, ACC reads three files containing the simulation specifications, the global paramenters for SPICE, and the SPICE models for the transistors.

ACC was implemented by routines written in C and scripts in UNIX C-shell, for further details see [50]. The flow of ACC is described in Table B.1

B.3 Synopsys Power Compiler

We summarize below the main features of Synopsys Power Compiler. In particular we discuss the power model, the cost function and some techniques used to reduce the power dissipation. Most of the information and data are derived from those presented in [15].

Power Compiler is built on the synthesis environment of Design Compiler and

```
Source configuration file containing library paths and global parameters.
For each cell in library
{
    Create a working directory $CELLNAME.
    Copy in $CELLNAME the simulation specifications (sim.specs).
    Copy in $CELLNAME the SPICE subcircuit ($CELLNAME.sub).
    For each line in sim.specs (e.g. each specification)
    {
        Create SPICE netlist ($CELLNAME.spi).
        Write file containing simulation variables (var).
        For each capacitance value $C_L$ in var
        {
            For each input stimuli set specified in var
            {
                Run SPICE.
                Extract value (e.g. $E_{tran}$) specified in var.
            }
        }
        Elaborate results (polynomial fitting).
    }
    Write energy view.
}
```

Table B.1: ACC working flow.

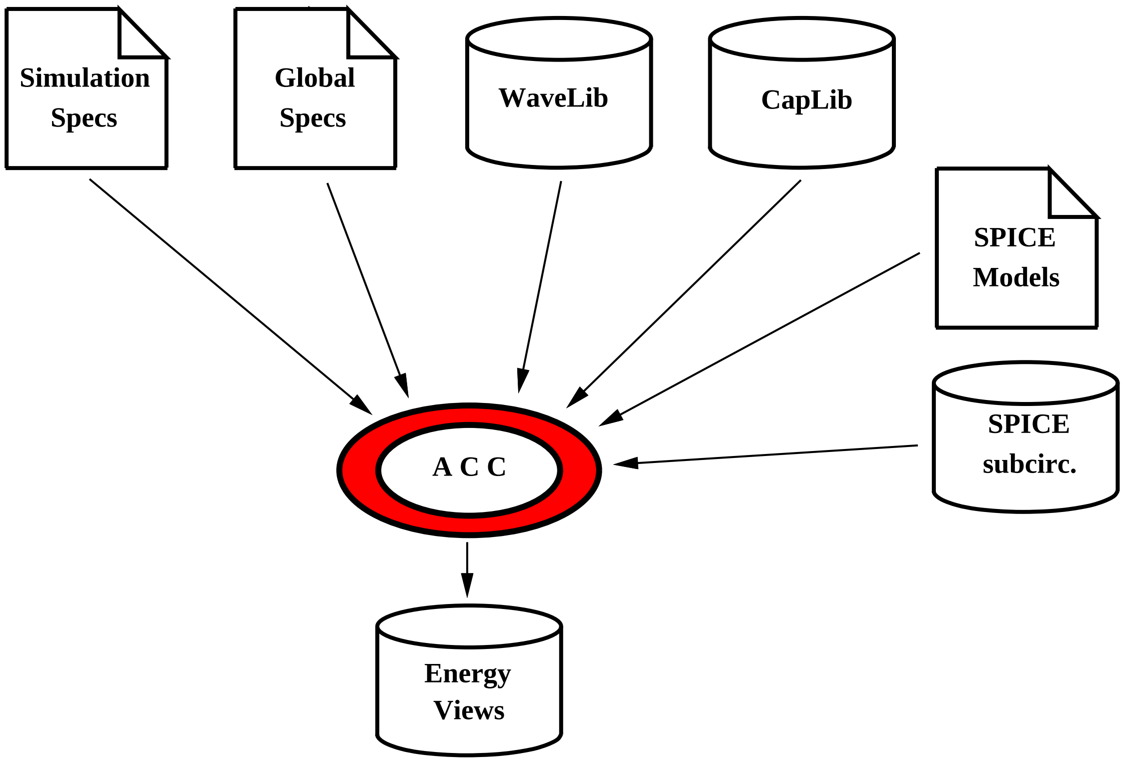


Figure B.2: Structure of ACC.

allows power optimization to be performed with delay and area optimization. Power Compiler obtains its power estimates from Design Power. The power dissipated is divided into 3 contributes:

Switching power: $\frac{1}{2}CV^2f$ depends on pin and wire capacitance, which values are available in the synthesis technology libraries, and transition count information described by toggle rates obtained either from Design Power's probabilistic estimation algorithm or from gate-level simulation.

Internal power: power consumed internally to the gate. The internal power model is not linear and provided by ASIC vendors as a look-up table derived from SPICE characterization. This energy table is indexed by the cell's input edge rates (slopes) and output loads to produce an energy value that is then multiplied by the toggle rate of the output.

Static or leakage power: This is a single constant value for the cell specified by the ASIC vendor.

In Power Compiler the cost function is prioritized as follows:

1. maximum delay
2. minimum delay
3. maximum dynamic power
4. maximum leakage power
5. maximum area.

This means that timing constraints will not be violated to save power, but available time slack will be used to reduce it. A transformation is accepted if decreases one of the cost functions, without increasing higher priority costs.

The circuit transformations that try to reduce one of the main factors contributing to the power dissipation: gate transistor dimensions, net switching activity, net transition times and net capacitive loading are described next.

B.3.1 Gate transistor dimensions

The dimensions of the transistors that compose a CMOS gate can influence a number of factors that determine the power consumption of a design. Sizing of a cell is done by choosing different implementations of the same logic function. These implementations might differ in their parasitic capacitance and internal power.

B.3.2 Composition

In order to reduce the switching power, Power Compiler merges or composes sets of cells into a more complex one. The switching power of the enclosed net is completely eliminated, however the internal power of the new cell is higher because of the increased gate size.

B.3.3 Pin swapping

Some cells can have input pins that are symmetric with respect to the logic function (for example, in a 2-input NAND gate the two input pins are symmetric), but have different capacitance values. Power can be reduced by assigning a higher switching rate net to a lower capacitance pin.

B.3.4 Sizing and buffering

The power due to the net transition time can be reduced by decreasing the transition times at the inputs. Power Compiler substitutes the driver of a net with a higher driver to sharpen the edge of the transition. In alternative the use of buffers can also reduce the transition time. The drawback is that the added capacitance (larger transistors in the driver, or extra gates to implement buffers) might offset the reductions obtained.