Power Aware Dividers in FPGA

Gustavo Sutter[1], Jean-Pierre Deschamps[2], Gery Bioul[3], and Eduardo Boemo[1]

[1] School of Engineering, Universidad Autónoma de Madrid, Spain
{gustavo.sutter,eduardo.boemo}@uam.es
[2] Dept. Eng. Electrònica, Elèctrica i Automàtica, Universitat Rovira i Virgili, Tarragona, Spain
[3] Universidad Nacional del Centro, Tandil, Argentina,
gbioul@exa.unicen.edu.ar

Abstract. This paper surveys different implementations of dividers on FPGA technology. A special attention is paid on ATP (area-time-power) trade-offs between restoring, non-restoring, and SRT dividers algorithms for different operand widths, remainder representations, and radices. Main results show that SRT radix-2 present the best ATP figure. In combinational implementation, an important power improvement, - up to 51% - with respect to traditional non-restoring implementations is obtained. Moreover, up to 93% power improvement can be achieved if pipelining is implemented. Finally, the sequential implementation is another important way to reduce the consumption in more than 89 %.

1 Introduction

A widely implemented class of division algorithm is based on digit recurrence. The most common implementation in modern microprocessors is *SRT* (Sweeney, Robertson, and Tocher) division. Digit recurrence, specifically SRT, and other division algorithm surveys can be found in [1-5]. This paper is focused on floating-point dividers, namely with operands in range [1,2).

Many implementations of SRT dividers on FPGA were recently presented. In [6], dedicated Virtex II multipliers are used to implement radix 2, 4, and 8 SRT dividers. Paper [7] presents a minimally redundant radix-8 SRT division scheme, and previous results of a radix-4 SRT are pointed out. In [8], radix 2, 4, and 8 SRT division schemes are iteratively implemented as fully combinational and pipelined circuits. In contrast this paper is based on the implementation reported in [9], where low-level component instantiations in parameterized VHDL code are used in order to keep control over implementation details.

Power consumption, has not been taken into account in these previous works. Thus, this paper tries to contribute to this research area by studying - and optimizing- the power dimension. Section 2 presents the division algorithms; and section 3 briefly discusses the details of Virtex implementation. Finally, section 4 presents some experimental results.

E. Macii et al. (Eds.): PATMOS 2004, LNCS 3254, pp. 574–584, 2004.

2 Algorithms

Given two non-negative real numbers X (the dividend) and D (the divisor), the quotient q and the remainder r are non-negative real numbers defined by the following expression: $X = q.D + r$ with $r < D.ulp$, where ulp is the unit in the least significant position. If X<D and D are the (unsigned) significant of two IEEE-754 floating-point numbers, then they belong to the range [1,2), and q lies in the range [0.5, 1). This result can be normalized by shifting the quotient by one bit to the left, and adjusting the exponent accordingly.

Division generally does not provide finite length result. The accuracy must be defined beforehand by setting the allowed maximum length of the result (p). The number of algorithmic cycles will therefore depend upon the aimed accuracy, not upon the operand length (n).

Restoring and non-restoring algorithm: To divide two integers, the most well known procedures are restoring and non-restoring digit-recurrence algorithms [3],[4]. The corresponding FPGA implementations are straightforward, and the area/time figure is always better for non-restoring. Figure 1 depicts restoring and non-restoring division algorithm. In the latter one, a correction step must be added in order to modify the last remainder whenever negative.

Restoring division algorithm		*Non-restoring division algorithm*	
r(0) := X; *for* i *in* 1 .. p *loop* rest_step(r(i-1), D,q(i), r(i)); *end loop;*	rest_step (a, b, q, r) z := 2*a - b; *if* z < 0 *then* q := 0; r := 2*a; *else* q := 1; r := z; *end if;*	r(0) := X; *for* i *in* 0 .. p-1 *loop* nonr_step(r(i-1), D,q(i), r(i)); *end loop;* q(p):=1; q(0):=1-q(0);	nonr_step (a,b,q,r) *if* a < 0 *then* q := 0; r := 2*a+b; *else* q := 1; r := 2*a-b; *end if;*

Fig. 1. Restoring and non-restoring division algorithms

SRT division: As others digit-recurrence algorithms, SRT generates a fixed number of quotient bits at every iteration. The algorithm can be implemented with the standard radix-r ($r = 2^k$) SRT iteration architecture presented at figure 2.a. The n-bit integer division requires $t = n/k$ iterations. An additional step is required in order to convert the signed-digit quotient representation into a standard radix-2 notation. The division x/d produces k bits of the quotient q per iteration.

The quotient digit q_j is represented using a radix-r notation (radix complement or sign-magnitude). The first remainder w_0 is initialized to X. At iteration j, the residual w_j is multiplied by the radix r (shifted by k bits on the left, producing $r.w_j$). Based on a few most significant bits of $r.w_j$ and d (n_r and n_d bits respectively), the next quotient digit q_{j+1} can be inferred using a quotient digit selection table (*Qsel*). Finally, the product $q_{j+1} \times d$ is subtracted to $r.w_j$ to form the next residual w_{j+1}.

Figure 2.b exhibits the general architecture; the last block *cond_adder* is only necessary if a positive remainder has to be calculated (not essential in floating-point implementations). For the hardware implementation of an SRT divider, some important parameters have to be traded-off:

Radix $r = 2^k$: For large values of k, the iteration number t decreases but each step is more complex (larger *Qsel* tables, complex products $qj+1 \times d$).

Residual representation w_j: Traditionally in ASIC implementations, a redundant number system such as carry-save is used for w_j to accelerate the operation $q_{j+1} \times d$ - $r.wj$. It is not necessary in current FPGAs: The dedicated carry logic circuitry makes ripple-carry faster than carry-save additions or subtractions for small bit widths.

Quotient representation: To speed up the subtractive division, redundant digit sets of the form of $\{-\alpha,-\alpha+1,...,0,...,\alpha-1,\alpha\}$ are used. The radix-2^k quotient is represented by a signed-digit redundant number system. It ensures that the next quotient digit determination is possible only based on a few most significant bits of the remainder and divisor (nr and nd respectively). Higher values of α lead to simpler quotient digit selection (smaller values of $nr + nd$ for the address of the $Qsel$ table) but also to more complex products $q_{j+1} \times d$.

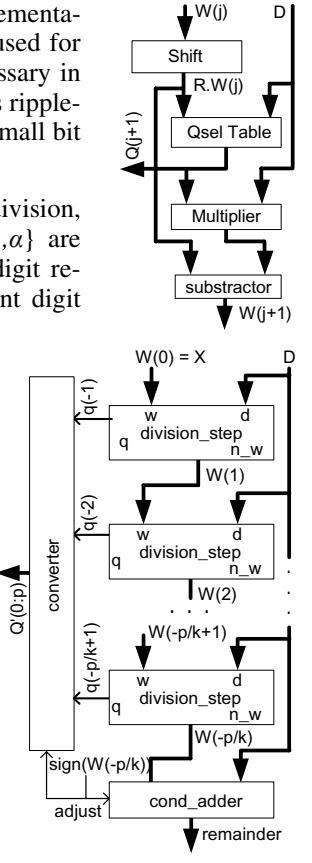


Fig. 2. a. SRT division step b. General SRT array

3 FPGA Implementations

Complete description of the algorithms and area-time tradeoffs in Virtex and Virtex 2 implementations are summarized in [9]. The architectures analyzed are: traditional restoring and non-restoring algorithms, then SRT radix 2, 4, 8, and 16 with 2's complement remainder, and finally, a novel implementation of SRT radix-2 with carry-save remainder representation. These division algorithms are implemented in different versions: fully combinational (minimum latency), pipelined with different logic depth (maximum throughput), and finally, sequential implementation with a choice of granularities (minimum area).

3.1 Radix-2 Restoring and Non-restoring

Integer division is traditionally done with restoring or non-restoring algorithms. The adjustment to fractional operands is trivial. The restoring division algorithm, implemented with the algorithm depicted in figure 1, needs p *rest_step* cells. Each cell uses an $(n+1)$-bit subtractor and an n-bit 2-1 multiplexer. This leads to $(n+1)$ slices. Finally, an n-bit divider with p-bit accuracy use $p.(C_{rest\_step}(n)) = p.(n+1) = p.n + p$ slices.

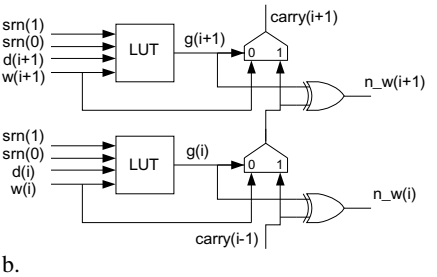
Non-restoring division is more efficient in Virtex FPGA. The *nonr_step* is implemented with p $(n+1)$-bit adder-subtractor. Each cell needs $n/2+1$ slices. If the final remainder is required, an additional conditional adder ($n/2$ slices) is necessary to adjust a possibly negative remainder.

3.2 SRTs Algorithms with 2´s Complement Remainder

SRT dividers for radix-2, 4, 8 and 16 were implemented. In radix-2 the *Qsel* table is trivial (actually, it doesn't exist). The most significant two bits of the remainder are used to settle on the operation to be executed in the next division step (figure 3.a).

W(n:n-1)= srn(1:0)	Remainder Value	Operation	Q(i)
0 0	$0 \le r < 1/2$	Nothing	0
0 1	$1/2 \le r < 1$	Subtract Div	1
1 0	$-1 < r \le -1/2$	Add Divisor	-1
1 1	$-1/2 \le r < 0$	Nothing	0

a.



b.

Fig. 3. SRT radix-2, with 2´s complement remainder. a. Operations to be done in a radix-2 SRT divider cell. b. Configuration of a Virtex slice for *srt_step_r2* cell.

Therefore, *Qsel*, the multiplier (multiplying by –1, 0 or 1 only) and the subtractor can be integrated in a single cell (*srt_step_r2*) that uses $(n+1)/2$ slices only. The slice detail of *srt_step_r2* is shown at figure 3.b. The *carry* (-1) is filled with *srn*(0) (the second most significant bit of the previous remainder). The logic function *g(i)* implemented in each LUT is $s_0\overline{d}\,\overline{w} + \overline{s_0}\,\overline{d}w + \overline{s_1}dw + s_1d\overline{w}$.

The total area of the SRT radix-2 corresponds to a *p srt_step_r2* cell in $p.(n+1)/2$ slices, a conditional adder ($n/2$ slices) if positive remainder is necessary, and a converter cell ($p/2+1$ slices), which is similar to the area of the non-restoring divider. That is, $p.(n+1)/2 + n/2 + p/2+1 = (pn+n+2p)/2+1$ slices. It can be observed in figure 3.a that the operation carried out by *srt_step_r2* is, for half of the time, "doing nothing": this characteristic of algorithm leads to a lower activity and consequently, lower power consumption.

For the higher radices, the *Qsel* table, the multiplier, and the adder-subtractor are necessary. Figure 4.a exhibits a SRT radix-4 division step, and table 1 display the parameters for the different radixes. An additional *converter* translates signed representation of radix-n digits to 2's complement, as is shown in figure 3.b for radix-4.

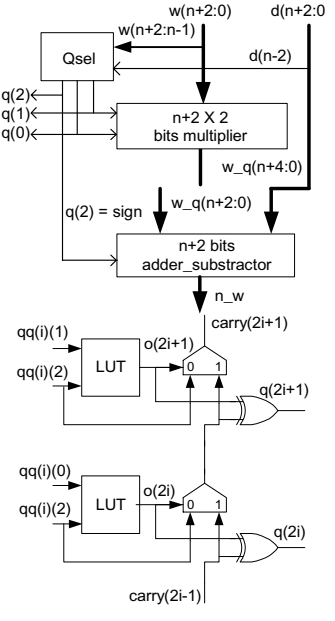


Fig. 4. SRT radix-4 divider details. a. *srt_step_r4* cell. b. Slice detail for SRT radix-4 converter.

3.3 Radix-2 SRT with Carry-Save Remainder

The block diagram of SRT radix-2 carry-save remainder is shown at figure 5.a. The division_step cell is depicted in figure 5.b. The first (leftmost) 3 bits of u and v are required to address the Qsel table where q_pos and q_neg are extracted. It has been empirically established that 3+3 bits from the carry-save representation are good enough to make a proper selection of the quotient digit [13], although 4+4 bits are suggested in [3] and [4]. The *Qsel* cell is implemented using 8 LUTs (4 slices) together with F5mux, F6mux multiplexers. The carry-save adder (CSA) of figure 5.b can be implemented within (n+1) slices using the cell of figure 5.c. Each CSA digit is calculated with one LUT only (together with a *muxcy* and a *xorcy*), but, due to routing limitations, only one CSA digit can be calculated per slice. For that reason, the division cell area is $C_{cell\_cs1} = n + 4$ slices.

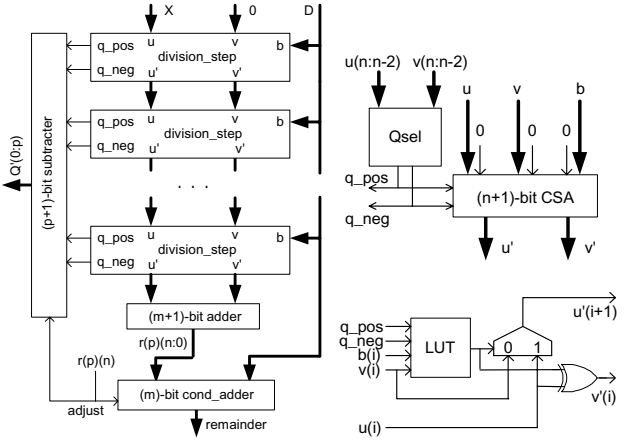


Fig. 5. SRT radix-2 with remainder in carry-save format. a Array implementation. b *division_step*. c. Slice contents in implementation of carry-save adder.

Table 1. Parameters for SRT radix-2, 4, 8, and 16. Qsel table size, remainder and divisor bits utilized, quotient range and bits utilized, remainder width, and number of stages.

Radix	Qsel Table					Quotient		Remainder width	Stages
	Total Bits	Remainder bits	Divisor bits	Sign Calculus	slices	Range	bits		
2	2	W(n-1)	-	W(n)	-	{-1, 0, 1}	2	N+1	P
4	5	W(n+1:n-1)	d(n-2)	W(n+2)	2	{-3,…,0,…,3}	3	N+3	P/2
8	9	W(n+2:n-2)	d(n-2:n-4)	W(n+3)	17	{-7,...,0,...,7}	4	N+4	P/3
16	12	W(n+4:n-2)	d(n-2:n-6)	W(n+5)	141	{-15,...,0,..., 15}	5	N+5	P/4

4 Implementation Results

The circuits are implemented in a Virtex XCV800hq240. They are described in VHDL instantiating low level primitives such as LUTs, muxcy, xorcy [10] when necessary. Xilinx ISE 6.1 tool [11] and XST [12] for synthesis were utilized. A common pin assignment, the preservation of the hierarchy, speed optimization, and timing

constrains were part of the experimental setup. Area and delay figures were extracted from Xilinx tools. On another hand, power consumption was measured using a Xilinx prototype board AFX PQ240-100. It was separated in static, dynamic (data-path and synchronization), and off-chip power as described in [14]. Chip measurements were done using three different sequences: a) random vectors (*avg_tog*); b) a sequence with a high transition probability (*max_tog*) and finally, c) a sequence with low activity (*min_tog*). The test vectors were entered with a pattern generator [15]. All the circuits were implemented and measured under identical conditions. Off-chip power was determined by measuring the average input current corresponding to the pad ring. This component was almost constant for all divider. At the output, each pad supported the load of the logic analyzer, lower than 3 pF [16].

4.1 Results in Array Implementations

Table 2 shows, for Virtex devices, area in slices, and delay expressed in ns. Up to 24 bits, non-restoring and SRT radix-2 (*srt_r2*) shows best results in delays. For greater operand sizes, SRT carry-save remainder (*srt_cs*), SRT radix 16 (*srt_r16*), and 4 (*srt_r4*) are the best options. In terms of area, SRT radix-2 and non-restoring (*nr*) are always the best. On the opposite side, restoring (*rest*) and SRT radix-16 consume more area. Best results in area × delay figure are provided by SRT radix-2 up to 24 bits, and SRT radix-4 for bigger divider sizes.

Table 2. Results for array implementations in Virtex.

N P	nonRest		rest		srt_r2		srt_r4		srt_r8		srt_r16		srt_cs	
	Slices	Delay	Slices	Delay	Slices	Delay	Slices	Delay	Slices	Delay	Slices	Delay	Slices	Delay
40	880	251.7	1640	329.1	861	293.2	940	243.7	1112	277.3	2258	245.7	1779	238.6
32	576	180.6	1056	238.3	561	198.1	624	187.8	804	224.6	1666	191.1	1183	176.2
24	336	118.7	600	158.4	325	125.5	372	125.7	487	154.6	1137	138.4	695	141.4
16	160	68.8	272	91.5	153	69.2	184	82.4	243	83.9	676	81.8	335	87.9

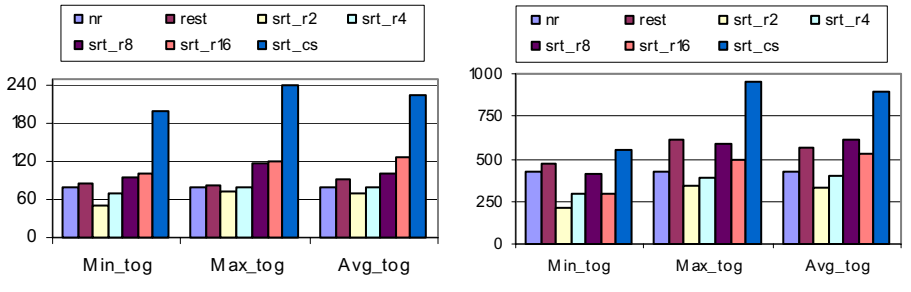


Fig. 6. Dynamic power consumption in mW/MHz. a. 16 bits dividers; b. 32 bits dividers.

Figure 6 depicts power consumption expressed in mW/MHz for the different vector sequences within 16 and 32 bits width. For 16 bits, SRT radix-2 presents the best results, improved in an average of 18.5% with respect to non-restoring divider, and

71% with respect to the SRT with carry-save remainder. For 32-bit representation SRT radix-2 and radix-4 are the best options. SRT radix-2 improves by up to 51,2 % the results of non-restoring division, and up to 78 % the results of SRT with carry-save remainder (the fastest one). Finally, area-time-power (ATP) for the different 32-bit dividers is presented in figure 7. The *srt_r2*, *srt_r4*, and *nr_f* offer best ATP figure. The analysis of area × delay × power relations of merit points to SRT radix-2 as the best choice, followed by SRT radix-4.

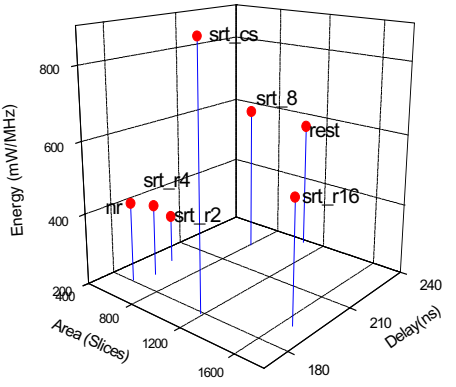


Fig. 7. Area-Time-Power for 32-bit dividers for the *avg_tog* sequence.

Table 3. Area in slices, register utilization, and maximum frequency, for different logic depths and architectures.

LD	C	Non-Restoring			SRT radix 2 SRL				SRT radix 4				SRT carry save			
		slices	FF	MHz	slices	FF	srl	MHz	slices	FF	srl	MHz	slices	FF	srl	MHz
1	33	1968	2705	101.7	1747	2274	88	111.7	-	-	-	-	3356	3298	90	79,1
2	16	1256	1328	55.9	1169	1152	52	53.8	1288	1265	39	62,6	2257	1647	52	48,3
3	11	1066	933	49.5	1012	835	48	49.6	-	-	-	-	1939	1164	48	43,1
4	8	943	688	34.5	915	641	40	34.3	967	632	30	39,4	1745	871	40	33,7
5	7	905	617	33.4	888	583	40	32.5	-	-	-	-	1689	780	40	30,7
6	6	866	538	25.3	858	521	36	26.4	907	508	36	31,3	1629	685	36	28,4
7	5	822	454	22.5	825	455	28	23.9	-	-	-	-	1566	586	28	24,9
8	4	779	368	20.0	786	385	32	20.2	835	372	12	21,7	1508	483	16	20,0
11	3	738	289	15.6	725	321	-	14.7	-	-	-	-	1456	386	-	16,8
12	3	740	292	14.4	728	327	-	13.6	782	315	-	16,2	1462	392	-	15,4
16	2	697	208	10.8	675	224	-	10.2	736	222	-	10,6	1355	256	-	11,2
32	1	656	128	5.6	625	128	-	5.2	752	226	-	5,3	1251	147	-	5,9

4.2 Results for Pipeline Implementations

The effective frequency of each node of a digital circuit can be significantly incremented by the occurrence of glitches. Although glitches do not produce errors in well-designed synchronous systems, they can be responsible for up to 70 % of the circuit activity [17]. The useless consumption associated to glitches can be decreased in two ways: equalizing all circuit paths [18] (almost impossible in FPGA), or inserting intermediate registers or latches to reduce the logic depth [19-23].

In order to reduce the power consumption, pipeline versions of non-restoring algorithm, SRT radix-2, SRT radix-4 and SRT radix 2 carry-save remainder representation were constructed. The circuits utilize SRL (LUTs configured as shift-register) whenever possible. It allows the designer to condense up to 16-bit shift-register (SR) in a single LUT. Table 3 shows area, expressed in slices, register count, and maximum

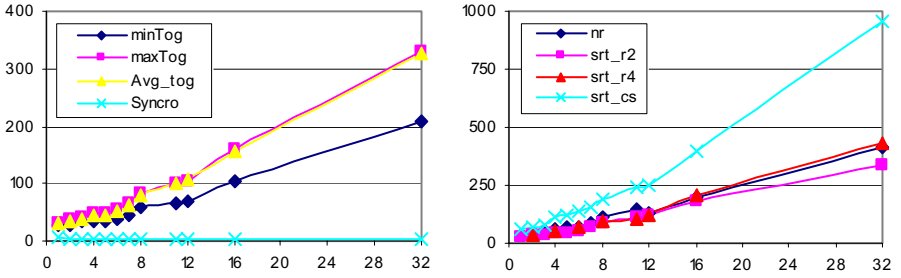


Fig. 8. Power consumption (mW/MHz) vs. logic depth. a. SRT radix-2 pipelining implementation. b. Different pipeline implementations using *avg_tog* sequence.

bandwidth in MHz, for 32-bits divider implementations and different logic depth (LD), defined as the maximum number of division steps between successive register banks. Pipelining in FPGA shows a low impact in area due to the embedded registers distributed into the slices and the SRL characteristics of LUT.

Figure 8.a presents dynamic power consumption versus logic depth for non-restoring implementations. The three different patterns have similar shape: it decreases practically linearly with the reduction of LD. It stands out the low influence of the synchronization power.

As more pipeline stages are added, fewer glitches are produced, and the power is lowered. This reduction in the activity makes less important the architecture selected. Thus, the different dividers have similar consumption. Figure 8.b shows, for the three different architectures, the dynamic power consumption as a function of the logic depth. A maximally pipelined architecture (LD = 1) saves up to 93 % of the dynamic power consumption with respect to the fully combinational architecture (LD=32). That is, combinational architectures consume more than twelve times more than the fully pipelined version.

4.3 Results for Iterative Implementations

To reduce area, using iterative architecture is a common technique. The general architecture adds a state machine that controls the data-path. It is constituted by g consecutive *division_step*'s and the corresponding registers to store intermediate values. The circuit calculates at each clock $g.r$ bits, and an extra cycle is necessary for remainder calculation. Then, a total of $p/g.r$ cycles are used to complete the operation plus an extra cycle if the remainder is needed.

Table 4 shows the results for iterative implementations in Virtex. The amount of bits calculated at a time (G), and the total clock cycles necessary (C), slices and register

Table 4. Results for iterative implementations in Virtex.

	G	C	slices	FF	P(ns)	F(MHz)	L(ns)
non_rest	1	32	113	203	8,9	112,0	285,7
	2	16	124	202	13,7	72,8	219,8
	4	8	155	200	23,8	42,1	190,2
	8	4	219	196	44,9	22,3	179,6
srt_r2	1	32	135	240	8,0	124,6	256,9
	2	16	139	236	13,4	74,5	214,8
	4	8	169	236	24,1	41,5	193,0
	8	4	229	236	47,9	20,9	191,7
srt_r4	2	16	134	221	12,7	78,8	202,9
	4	8	169	221	21,9	45,7	175,2
	8	4	240	222	41,2	24,3	164,6
_r16	4	8	336	255	23,0	43,5	183,9
	8	4	603	294	41,9	23,9	167,5
srt_cs	1	32	179	269	11,9	83,8	382,0
	2	16	210	267	17,7	56,6	282,6
	4	8	282	267	29,9	33,4	239,3
	8	4	426	267	52,5	19,0	210,2

utilization. Finally, minimum period, maximum frequency and latency are presented. As G grows, the latency decreases at the cost of extra area. Best results in terms of latency are provided by SRT radix-4. Minimum value for area × latency figure is obtained for G=2.

Figure 9.a shows the average energy for an operation in n Joules for 32-bit width divider. The synchronization and data-path components are also displayed. The synchronization power decreases as G grows, mainly because of smaller cycles. In the opposite, the data-path consumption grows with G, mainly because the glitches increases. Optimum G value seems to be 4, apart from SRT carry-save remainder (*srt_cs*).

Non-restoring division with G=4 (*nr_g4*) shows lowest power consumption, while SRT radix-2 with G=4 (*srt_r2_g8*) and SRT radix-4 with G=4 (*srt_r4_g4*), have similar energy consumption. An important point is that the value of G, better that a particular algorithms, defines the power figure. In SRT radix-2, G=4 save 51% energy with respect to G=1. The energy savings with respect to the fully combinational implementations are: 85 % as regards SRT radix-2, 89 % as regards non-restoring division, and 94% as regards the SRT carry-save remainder.

4.4 Architectural Comparisons

Figure 9.b shows ATP figure for some 32-bit circuits. The array implementations have the lowest latency, but as the cost of a great area and excessive power dissipation. Pipeline offers the best throughput, with a relatively low increment in area with respect to array implementations and a good power figure, but the initial latency could be prohibitive for some applications. Finally, sequential implementations have the smaller area, a delay less than twice the one of arrays, but have a good power figure.

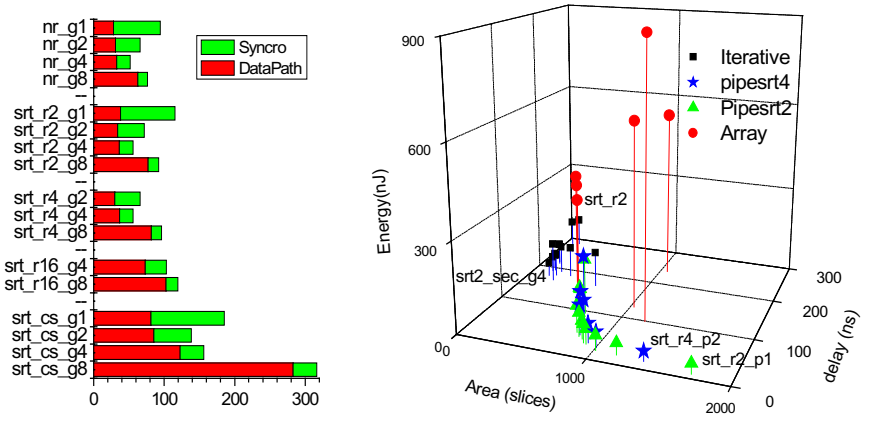


Fig. 9. a. Dynamic power consumption for different sequential divider implementations. b. Area-Time-Power for sequential, array, and pipeline implementations.

5 Conclusions

This paper has presented a power analysis for improved architectures and implementations of SRT division on mantissa operations (floating-point numbers).

The circuits were implemented in VHDL, instantiating low-level primitives when necessary. Array implementations, pipelined with different logic depths, and sequential implementations were constructed and the power was measured.

For array implementations SRT radix-2 has the best ATP figure, reducing power consumption up to 51 % with respect to traditional non-restoring division and 93 % with respect to SRT radix-2 with carry-save remainder representation.

Pipeline architectures offer an important way to reduce the power consumption. The measurements show reductions of up to 93 % of dynamic power in a fully pipelined divider with respect to an entirely combinational architecture. Such improvement is obtained with a relative low impact in area.

The sequential implementations use lower area resources, with a relative low impact in delay, but with an important power reduction, -up to 89 %- with respect to fully combinational implementation. An important criterion in sequential implementations is the amount bits calculated at a time (G). G=4 show the best power consumption and ATP figure, while G = 2 has the best Area × Latency figure. Non-restoring and SRT radix-2 exhibit the best results.

More researches are needed to explore further alternatives such as larger operand sizes, or ad-hoc disabling architectures to reduce glitch propagations. The dividers features are currently analyzed for Virtex 2. An optimized in ATP fully IEEE compliant floating-point unit is another key research interest.

Acknowledgement. This work is supported in part by Project TIC2001-2688-C03-03 of the Spanish Ministry of Education and Science, and in part by Project 07T/0052/2003-3 of the *Consejería de Educación de la Comunidad de Madrid*, Spain

References

1. S.F. Oberman and M.J. Flynn. “Division algorithms and implementations”. *IEEE Transactions on Computers*, 46(8):833–854, August 1997.
2. M.D. Ercegovac and T. Lang. *Division and Square-Root Algorithms: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic, 1994.
3. B. Parhami. *Computer Arithmetic: Algorithms and Hardware Design*. Oxford University Press, 2000.
4. M.Ercegovac and T.Lang. *Digital arithmetic* San Francisco: Morgan Kaufmann, cop. 2004
5. P.Soderquist and M.Leeser. *Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementation*s, ACM Computing Surveys, Vol. 28, No. 3, September 1996.
6. J-L.Beauchat and A.Tisserand, “Small Multiplier-Based Multiplication and Divison Opera-tors”, *12th Conference on Field Programmable Logic and Applications*, pp. 513-522. 2002.
7. B.R. Lee and N. Burgess, “Improved Small Multiplier Based Multiplication, Squaring and Division” *11thIEEE symposium on Field-Programmable Custom Computing Machines.2003.*
8. X. Wang and B.E. Nelson, “Tradeoffs of Designing Floating Point Division and Square Root on Virtex FPGAs” *11th IEEE symposium on FCCM*. 2003.

9. G.Sutter, G.Bioul, and J-P.Deschamps, “Comparative Study of SRT-Dividers in FPGA”, Conf.on Field Programmable Logic and Applic. (FPL’04), Antwerp, Belgium, Sept 2004.
10. Xilinx Inc, Libraries Guide for ISE 6.1 available at www.xilinx.com, 2003.
11. Xilinx Inc, Xilinx ISE 6 Software Manuals, available at www.xilinx.com, 2003.
12. Xilinx Inc, XST User Guide 4.0, available at www.xilinx.com, June 2003.
13. G.Sutter, “FPGA implementation and comparison of SRT dividers”, UAM, Technical Report, December 2003.
14. E. Todorovich, G. Sutter, N. Acosta, E. Boemo and S. López-Buedo, "End-user low-power alternatives at topological and physical levels. Some examples on FPGAs", Proc. DCIS'2000, Montpellier, France, Nov. 2000.
15. Tektronix inc, TLA7PG2 Pattern Generator Module User Manual. www.tektronix.com.
16. Tektronix inc, TLA 700 Series Logic Analyzer User Manual. www.tektronix.com.
17. A. Shen, A. Gosh, S. Devadas and K. Keutzer, "On average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks", *Proc. ICCAD-92 Conf*, pp.402-407, IEEE Press, 1992.
18. M. Pedram, "Power Minimization in IC Design: Principles and Applications", *ACM Trans. On Design Automation of Electronic Systems"*, vol.1, nº1, pp.3-56, January 1996.
19. A. Chandrakasan, S. Sheng and R. Brodersen, "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 4, pp. 473-484. April 1992
20. E. Mussol and J. Cortadella, “Low-Power Array Multiplier with Transition-Retaining Barriers”, *Proc. PATMOS´95, Fifth Int. Workshop*, pp. 227-235, Oldenburg, October 1995.
21. J. Leiten, J. van Meerbegen and J. Jess, Analysis and Reduction of Glitches in Synchronous Networks”, *Proc. 1995 ED&TC*, pp.1461-1464. New York: IEEE Press, 1995
22. E. Boemo, G. Gonzalez de Rivera, S.Lopez-Buedo and J. Meneses, "Some Notes on Power Management on FPGAs", *LNCS,* No.975, pp.149-157. Berlin: Springer-Verlag 1995.
23. E. Boemo, S. Lopez-Buedo, C. Santos, J. Jauregui and J. Meneses, "Logic Depth and Power Consumption: A Comparative Study Between Standard Cells and FPGAs", *Proc. XIII DCIS Conference (Design of Circuit and Integrated Systems)*, Madrid, Univ. Carlos III: Nov 1998.
24. Sutter G, Todorovich E, Lopez-Buedo S, and Boemo E. “Logic Depth, Power, and Pipeline Granularity: Updated Results on XC4K and Virtex FPGAs” III Workshop on Reconfigurable Computing and Applications JCRA03, Madrid Spain, Sept 2003.