Reconfigurable Realization of Low-Complexity RNS Multiplier

Reza Nekovei
Associate Professor, EECS Department
Texas A&M University--Kingsville
Kingsville, TX 78363, U.S.A.
(361) 593-2635
rnekovei@tamuk.edu

Sneha Bhagwat
Graduate Student, EECS Department
Texas A&M University--Kingsville
Kingsville, TX 78363, U.S.A.
(361) 593-2004
Kssb002@tamuk.edu

ABSTRACT

Residue Number System (RNS) is a non-weighted system with carry-free and borrow-free arithmetic operations. Addition, subtraction, and multiplication are carried out on each residue digit concurrently and independently. This simplifies supporting parallel high-speed concurrent computation. It is highly desirable to produce RNS based devices for DSP algorithms in telecommunication and multimedia applications due to their advantages in terms of speed, power, and precision. Additionally, availability of recent low cost, low power, high density reconfigurable devices with their fast turn around and flexibility make them the ideal platform in comparison to the custom ASIC's. This paper investigates the applicability of programmable logic devices for realizing a RNS multiplier. Hardware implementation of the multiplier mapped on reconfigurable technologies using the low-complexity combinatorial RNS multiplier is illustrated. A performance study helps to understand the realization of RNS multipliers based on area/delay optimizations, speed effects, and device family architecture. This involves generation, simulation and synthesis of VHDL code using WVO, Altera Maxplus-II, FPGAExpress, and LeonardoSpectrum as design environments. Altera CPLD and FPGA device families are studied for implementation of RNS mod-5 and mod-15 multipliers using the three-stage low-complexity architecture. The $area \times time$ *product demonstrates the superiority of FPGA architecture for this application.*

1. INTRODUCTION

Over the last three decades there has been considerable interest in the implementation of digital computing elements using hardware based on the residue number system. A great deal of research has already been done on the Residue Number System (RNS). RNS is a non-weighted, carry-free system that performs addition, subtraction and multiplication as parallel operations and supports concurrent computations.

RNS systems are particularly efficient for executing algorithms, which contains a significant amount of multiply-accumulate operations (such as DSP algorithms) even when the unavoidable forward and inverse conversion overhead is considered. Combinatorial RNS architectures are of interest, as they become more efficient compared to look-up table architectures with increasing modulus size [1-3,8-11]. RNS multipliers are considered for this purpose. To analyze the performance of the residue multipliers, VHDL models were developed. The VHDL models help us to study, understand, and validate the functionality of the circuits, but it does not provide us with any performance information with respect to the speed or area required by the design. In order to compare the performance of the proposed multipliers, they are simulated using various Altera CPLDs such as Max 7000 Series, Max 9000, and Flex10K FPGA architectures. The study of these device architectures gives us the basis on which the proposed multiplier designs would be implemented using MaxplusII and their performance with respect to speed and area is analyzed by using ViewLogic's FPGAExpress. LeonardoSpectrum synthesis tool was used to determine the percentage of logic cells utilized, the frequency and overall performance of the CPLDs and FPGAs.
Finally, the results were tabulated with the timing and performance analysis in terms of area, utilization, and speed. A comparison of the devices with respect to the performance has been carried out and the findings of the research have been summarized in the conclusions section that suggests the best-fit device for various specifications and the advantages of each chosen device.

2. RNS ARITHMETIC

The Residue Number System can efficiently perform computations in digital signal processing applications such as the computation of the Fast Fourier Transform and digital filtering [4], since it provides carry-free addition and multiplication and borrow-free subtraction. In RNS an integer is encoded as a vector of residues with respect to its moduli. The moduli is a set of selected relatively prime integers $\{m_1, m_2, m_3 ....m_N\}$. This encoding represents an integer with an N-tuple smaller integers (residues), which have the property of carry-free numbering system and highly desirable for parallel processing. In this numbering system, for a given value A when A is nonnegative and smaller than the product of selected moduli (i.e. $0 \le A < \prod_{i=1}^{N} m_i$), there exists a unique RNS representation for A.

Let the selected moduli set be $\{m_1, m_2, m_3 ....m_N\}$, N-tuples for A and B can be calculated by

$$A_i = A \bmod m_i = \langle A \rangle_{m_i} \quad i = 1,2,...N \qquad (1)$$

$$B_i = B \bmod m_i = \langle B \rangle_{m_i} \quad i = 1,2,...N \qquad (2)$$

The multiplication result for A and B can be performed in parallel to create the N-tuple result C where

$$C = A \times B = \left(\langle A_1 \times B_1 \rangle_{m_1}, \langle A_2 \times B_2 \rangle_{m_2}, ... \langle A_N \times B_N \rangle_{m_N} \right)$$

or

$$C_i = \langle A_i \times B_i \rangle_{m_i} \qquad (3)$$

which illustrates how each C_i can be independently computed from corresponding A_i and B_i with no information needed from other elements. The following example illustrates this concept:

Let integers A=11, B=12, and set {2, 3, 5, 7} be the selected base. The dynamic range for this set is

$$\prod_{i=1}^{N} m_i = 2 \times 3 \times 5 \times 7 = 210$$

$$A \xleftrightarrow{RNS} \{1,2,1,4\}$$

$$B \xleftrightarrow{RNS} \{0,0,2,5\}$$

then

$$C \xleftrightarrow{RNS} \{ \langle 1 \times 0 \rangle_2, \langle 2 \times 0 \rangle_3, \langle 1 \times 2 \rangle_5, \langle 4 \times 5 \rangle_7 \}$$

$$C = 11 \times 12 = 132 \xleftrightarrow{RNS} \{0,0,2,6\}$$

To verify, the RNS tuple can be converted back into decimal or binary using the Chinese Remainder Theorem [4]:

$$n = \left\langle \sum_{i=0}^{N} \hat{M}_i \langle \alpha_i \times x_i \rangle_{m_i} \right\rangle_M \qquad (4)$$

Where, by definition, $M = \prod_{i=0}^{N} m_i$, $\hat{M}_i = M / m_i$, and the multiplicative inverse is $\alpha_i = \langle \hat{M}_i^{-1} \rangle_{m_i}$. For the above example:

$$C = \langle 105 \times \langle 1 \times 0 \rangle_2 + 70 \times \langle 1 \times 0 \rangle_3 + 42 \times \langle 3 \times 2 \rangle_5 + 30 \times \langle 4 \times 6 \rangle_7 \rangle_{210}$$

$$C = 42 + 90 = 132$$

3. VHDL IMPLEMENTATION

3.1 Three Stage Architecture

Several VLSI architectures have been proposed for RNS multiplication. They are implemented, either by memory lookup table techniques or combinatorial logic [1,2,3]. They typically require manually placing hundreds of ROM macrocells [6]. Therefore, the three-stage low complexity multiplier seems most promising for configurable implementation. In this architecture, residue multiplier is divided into three stages, as in previously reported residue arithmetic units [5,7,12]. Each stage is comprised of columns of Full Adders, Half Adders, and OR gates. The hardware is optimized by reducing the number of bits to be added by the ith column of 1-bit adders.

The multiplier computes the residue C when A and B are in binary residue form by

$$C = < A \times B >_{m_i} = \left\langle \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \langle 2^{i+j} \rangle_{m_i} \right\rangle_{m_i} \qquad (5)$$

The above equation is implemented in three stages. The first stage, calculates the nested summations using columns of optimized 1-bit adders to generate the bit products. The second stage transforms them into the appropriate length. Finally, the third stage performs the residual mapping using conditional correction [5]. Figure 1 illustrates the basic VHDL modules for the implementation of RNS multiplier.

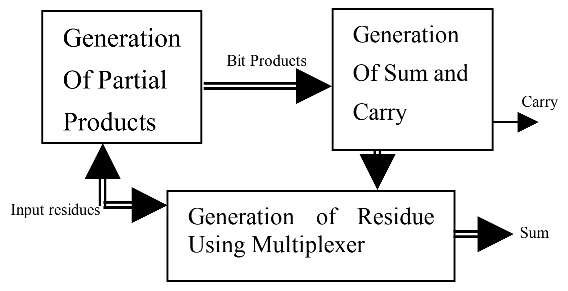
Generation Of Partial Products
Bit Products
Generation Of Sum and Carry
Carry
Input residues
Generation of Residue Using Multiplexer
Sum

Figure 1. Block diagram of VHDL implementation modules

3.2 Hardware Architecture

The low-complexity RNS multiplier architecture is based on a hardware reduction procedure, which minimizes the number of bits required to add in each column and hence simplifying the required hardware. The actual construction of a stage in the implemented architecture is performed after a three-step design procedure is followed [5]. The objective of the procedure is to locate and optimize all possible disjoint couples and triplets. The couples and triplets are meant to identify two or three bit values which can be added without carry. The triplets are defined as

$$y_{k,j_1} + y_{k,j_2} + y_{k,j_3} \le 1 \ \forall \, \mathrm{Y}_k \in \mathrm{I}_k , \qquad (6)$$

and the couples as

$$y_{k,j_1} + y_{k,j_2} \le 1 \ \ \forall \, \mathrm{Y}_k \in \mathrm{I}_k \ . \qquad (7)$$

The bits in a column to be added can be minimized by maximizing the bits in the couples or triplets. The optimization procedure given by [5] groups the input bits into appropriate sets of triplets and couples.

Due to the fact that these additions do not require a carry, a full adder, or half adder can be simplified into a simple XOR gate for generating the sum. However, the XOR gates can also be replaced with a simpler OR gate since only one of its two inputs gets asserted for any residue number. This hardware reduction procedure becomes possible since the residue multiplication operands create an incomplete set and in many cases the bits assigned to a full adder (triplets) or a half adder (couples) do not require all possible values, but rather a limited subset. The hardware is further simplified in the subsequent digits (more significant columns), since the number of carries generated at the *i*th column is reduced and, therefore, fewer bits have to be processed by the (i+1)*th* column. The VHDL multiplier models are implemented using these three stages comprising of full adders, half adders, and OR gates, Figure 2 illustrates the modulo-5 residue multiplier.

4. PERFORMANCE ANALYSIS

Two types of modulo multipliers were implemented and tested on Altera's Max and Flex devices. Analysis for the devices was performed using the Maxplus II, Synopsis FPGAExpress, and Mentor Graphics LeonardoSpectrum software based on optimization for area and speed.

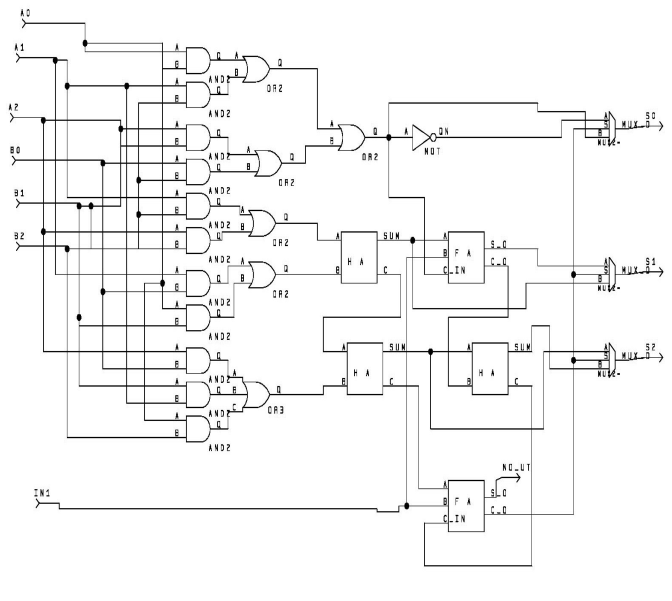

Figure 2. Architecture for the modulo-5 multiplier

4.1 Timing Analysis

The output delays for RNS modulo-5 and modulo-15 multipliers from the VHDL code synthesized for both area and delay optimizations with FPGA Express tool are shown in Tables 1 and 2. The devices have been categorized as Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs) according to the architecture and technologies used.

Table 1. Delay values for modulo-5

CPLD				
Devices	Area Optimization		Speed Optimization	
	LEs	Delay, ns	LEs	Delay, ns
EPM7032LC44	22	49	34	38
EPM7064QC100	24	49	33	37
EPM7128SLC84	22	49.5	31	38
EPM7192SQC160	22	50	32	53.5
EPM9320LC84	23	50	33	40
FPGA				
Devices	Area Optimization		Speed Optimization	
	LEs	Delay, ns	LEs	Delay, ns
EPF10K50RC240	14	33.0	21	26.4
EPF10K70RC240	14	33.5	21	26.4
EPF10K50VRC240	14	26.4	21	21.12
EPF10K100ARC240	14	26.4	21	21.12

Table 2. Delay values for modulo-15

CPLD				
Devices	Area Optimization		Speed Optimization	
	LEs	Delay, ns	LEs	Delay, ns
EPM7032LC44	60	75.5	90	75.0
EPM7064QC100	90	74.0	107	75.0
EPM7128SLC84	60	75.5	110	75.0
EPM7192SQC160	55	74.5	64	69.5
FPGA				
Devices	Area Optimization		Speed Optimization	
	LEs	Delay, ns	LEs	Delay, ns
EPF10K10QC208-3	24	48.6	29.3	38
EPF10K20RC208-3	24	47	28	37
EPF10K30RC240-3	24	46.89	27.1	35
EPF10K50RC240-3	24	46.20	26	33
EPF10K70RC240-3	24	35.64	26	30
EPF10K100ARC240-1	22	34.96	24	26.4

The MAX family of CPLDs uses product-term based macro-cells whereas Flex family FPGAs use the SRAM based Look-Up Tables (LUTs). The LUT is a functional generator, which can quickly compute any function of four variables, used as memory or shift registers.

The multipliers with moduli 5,8,7 have been represented using the modulo-5 multiplier and the moduli 11, 13 and 15

have been represented using the modulo-15 multiplier. These two RNS multipliers have been synthesized for both CPLD and FPGA architectures.
We notice from the above tables that when the larger circuit were optimized for speed, they use more Look-Up Tables (LUTs), thus increasing the area whereas the delay reduction is only marginal when compared to the devices optimized for area. This is especially obvious for the FPGA devices.

4.2 Performance Analysis for CPLDs

A better understanding of the performance is obtained when we consider the throughput performance or the data rate of the devices and effective size of the design with respect to the device. The larger circuit (modulo-15) is used for performance analysis in this section. Tables 3 and 4 explain the data rate and required resources.

Table 3. Effective size using modulo-15

Family	Device	Available Les	Used LEs	% Of LCs used
Max 7000	EPM7032LC44	32	20	62.5%
Max 7000	EPM7064QC100	64	25	39.06%
Max 7000S	EPM7128SLC84	128	22	17.19%
Max 7000S	EPM7128EQC160	128	27	21%
Max 7000S	EPM7192SQC160	192	27	14%
Max 9000	EPM9320LC84-15	320	29	9%

Table 4. Throughput Analysis of modulo-15

Max Devices	IOs Available	IOs Used	Maximum Delay (ns)	Throughput Rate (Mbps)
EPM7032LC44	32	14	35.7	141.6
EPM7064QC100	64	14	44.64	178.56
EPM7128SLC84	64	14	39.29	157.16
EPM7192QC160	120	14	32.14	128.56

A comparison between the delay analysis values from Maxplus II software and the delays obtained previously from FPGAExpress for area and speed optimizations clearly indicates the dependency to the synthesis algorithms.
However, the throughput gives a clear estimation of the device performance, as it is the output of the device based on the frequency and number of bits per second that emphasizes the overall performance of the device. It can be seen that EPM7064QC100 is the fastest. Even that as the size of the device increases, the number of IOs and LEs increase, the device performs slows down, a data rate of only 128 Mbps for the large EPM7192QC160 device. This is due to the overhead for unnecessarily oversized device selection.

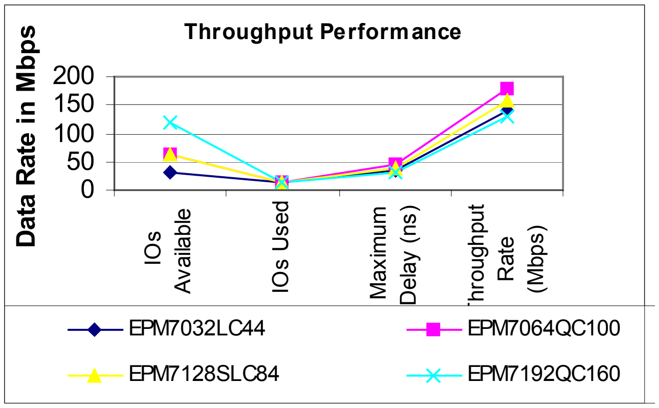

Figure 3. Overall performance of CPLDs

Thus among all the Max device families tested EPM7064QC100 can be chosen as the best fit device for designing the low complexity modulo multipliers of modulo 11, 13 and 15 having a word length of n=4. EPM7192QC160 was the next most efficient device using less LCs and reduced delays but may be used for larger moduli such as mod 25, 27, 29, 31 or other multipliers with word length n=5. In order to obtain maximum speed-performance, it is often necessary to have intimate knowledge of the structure of the PLD, and to investigate many design alternatives before establishing one device as most suitable for the implementation.

4.3 Performance Analysis for FPGAs

The FPGAs are large devices; we only implemented the modulo-15 RNS multiplier for this analysis, so that a reasonable percentage of area is utilized on the programmable device.
When the selected FPGA devices are optimized for speed more LUTs are used but there is a drastic reduction in the delays. There is a difference of almost 10 ns in each device when optimized for area. Hence, unlike Max devices, a delay/speed optimization is preferred for the FPGA devices. Table 2 shows that a speed optimization using FPGAExpress synthesis tools give better results in terms of delays and area when FPGA devices are considered. Here, all the devices with the same speed grade of –3 have been taken into account.
The performance of the RNS multipliers with respect to the number of logic elements used is also an important issue to be analyzed. The Maxplus II software generates a report file for a compiled design, which gives us information regarding the number of Logic Elements (LEs) used by the design. Table 5 gives us the percentage of LEs used by the design for the LEs present in each of the FPGA devices.

Table 5. Device Utilization based on LEs Used

ALTER DEVICES	LEs Available	LEs Used	% LEs used
EPF10K10LC84-3	576	29	5.034%
EPF10K20RC208-3	1152	32	2.34%
EPF10K30RC208-3	1728	29	1.678%
EPF10K50RC240-3	2880	29	1.007%
EPF10K70RC240-2	3744	27	0.73%
EPF10K100ARC240-1	4992	27	0.54%
Flex10KA100ARC240-1	4992	27	0.54%

As the device size increases, obviously the percentage utilization of the device reduces for a given circuit.

Table 6. Speed Grade Effect on the Performance of Implemented Design

Altera device	LEs Available	LEs used	Maximal Frequency (MHz)	Data rate (Mbits/s)
EPF10K30BC356-4	576	29	11.7	46.8
EPF10K30BC356-3	576	28	14.8	59.2
EPF10K30RC484-3	576	26	21.1	84.4
EPF10K30EFC484-2	576	23	29.9	119.6
EPF10K30EFC484-1	576	23	37.3	149.2

For a given device architecture the change of speed grade should have a direct effect on the performance. This is illustrated in the data of Table 7 and Figure 4.

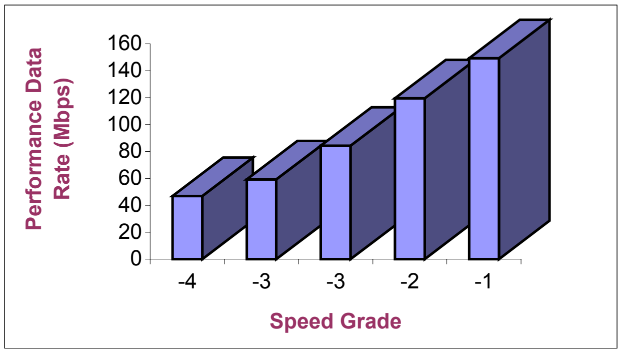

Figure 4. Speed grade effect on Flex 10K device family

Similar to the CPLD device analysis, the FPGA devices can be analyzed for their overall performance based on the throughput data rate in megabits per second (Mbps). This is obtained by considered the throughput frequency for the critical path delays of each FPGA device. When the speed grade of all the devices is maintained at a fix value (–3), then throughput performance of the devices based on the individual critical path delays is measured.
The results have been tabulated and graphically represented in Figure 5.

Table 7. Effect of throughput rate on the performance

ALTERA DEVICE	LEs Used	Max. Freq.	Data Rate (Mbps)
EPF10K10QC208-3	34	13.8	55.2
EPF10K20RC208-3	32	21.6	86.4
EPF10K30BC356-3	29	17.36	69.44
EPF10K50EFC484-3	29	15.89	63.2
EPF10K70RC240-3	27	14.8	59.2

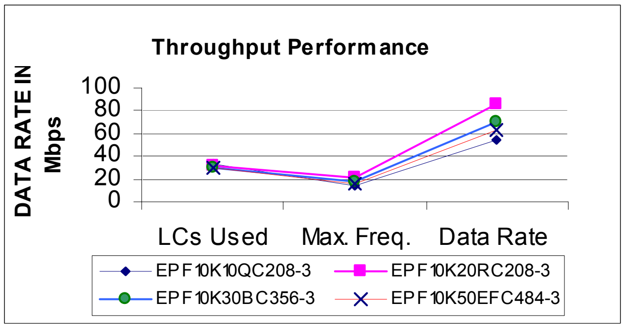

Figure 5. Overall performance of FPGAs

5. Conclusion

The reconfigurable implemented RNS modulo multiplier is fast and area efficient compared to other schemes. Using hardware description language (VHDL) the synthesis of basic elements and modules for residue number systems becomes simple, yet flexible.

1. In order to obtain maximum speed-performance, it is often necessary to have an intimate knowledge of the structure of the PLD, and to investigate many design alternatives. For small RNS applications, speed-performance in CPLDs is higher than in FPGAs.
2. For Flex 10K devices, speed optimization gives better performance, while for CPLDs such as Max7000 and 9000 families, an area optimization resulted in better performance.
3. As expected an unnecessarily oversize chip results in slower performance.
4. FPGA performance often depends more upon how CAD tools map circuits into the chip than is the case for CPLDs.
5. Max CPLDs are not as efficient for adder and multiplier circuits due to potential waste of the logic array when compared to FPGAs that are superior for arithmetic circuits and logic implementation.
8. The amount of routing resources on the chip is crucial for large RNS implementation.

6. References:

[1] G. A. Jullien, "Implementation of multiplication, modulo a prime number," with applications to number theoretic transforms, IEEE Transactions on computers, Oct' 80.

[2] S. H. Leung, "Applications of residue number systems to complex digital filters", in Proc. 15th Asilmar Conf. Circuits, Systems and computers, pp. 70–74, 1981.

[3] D. Radhakrishnan and Y.Yuan, "Novel approaches to the design of VLSI RNS multipliers," IEEE Transactions on Circuits and Systems-II, Jan 1992.

[4] N. Szabo and R. Tanakar, "Residue Arithmetic and its applications to Computer Technology," McGraw-Hill 1967.

[5] V. Paliouras, K. Karaginni, and T. Stouraitis "A Low_complex Combinatorial RNS Multiplier," IEEE Transaction on Circuits and Systems-II, Vol. 48, No. 7, 2001.

[6] R. Klaasen, D. Birreck, S. Wolter, R. Laur, "VLSI Architecture for a convolution-based DCT in residue arithmetic," IEEE proceeding of International Symposium on Circuits and Systems, May 1992.

[7] V. Paliourus and T. Stouraitis, "Multifunction architectures for RNS processors," IEEE Transactions on Circuits and Systems-II, August 1990.

[8] R. Zimmermann, "Efficient VLSI Implementation of Modulo ($2^n \pm 1$) Addition and Multiplication," Proc .14th IEEE Symposium on Computer Arithmetic, pp.158-167, Apr' 99.

[9] E. D. DiClaudio, F. Piazza, and G. Orlandi, "Fast combinatorial RNS processors for DSP applications," IEEE Transactions on Computers, May 1995.

[10] M. A. Bayoumi, G.A.Jullien, and W.C.Miller, "A VLSI implementation of residue adders," IEEE Transactions on Circuits and Systems, vol. CAS-34, pp284-288, March 1987.

[11] T. Stouraitis, S. W. Kim and A. Skavantzos, "Full adder based arithmetic units for infinite rings," IEEE Transactions on Circuits Systems-II, Nov. 1993.

[12] D, J, Soudris, V. Paliouras, T. Stouraitis and C. Goutis, "A VLSI design methodology for RNS full adder-based inner product architecture," IEEE Transactions on Circuits and Systems-II, vol. 46, pp. 1041-1054, Aug. 1999.

[13] Altera Corporation, San Jose, California, "Altera Literature Data Sheet: Flex 10k", *Data Handbook*, 2000.