PERFORMANCE ANALYSIS OF DOUBLE DIGIT DECIMAL MULTIPLIER ON VARIOUS FPGA LOGIC FAMILIES

Rekha K. James, K. Poulose Jacob[*]

Department of Computer Science
Cochin University of Science and Technology
Kochi, Kerala, India
email: rekhajames@cusat.ac.in,
kpj@cusat.ac.in

Sreela Sasi[†]

Department of Computer and Information Science
Gannon University
Erie, PA, USA
email: sasi001@gannon.edu

ABSTRACT

Decimal multiplication is an integral part of financial, commercial, and internet-based computations. This paper presents a novel double digit decimal multiplication (DDDM) technique that performs 2 digit multiplications simultaneously in one clock cycle. This design offers low latency and high throughput. When multiplying two n-digit operands to produce a 2n-digit product, the design has a latency of $\lceil (n/2)+1 \rceil$ cycles. The paper presents area and delay comparisons for 7-digit, 16-digit, 34-digit double digit decimal multipliers on different families of Xilinx, Altera, Actel and Quick Logic FPGAs. The multipliers presented can be extended to support decimal floating-point multiplication for IEEE P754 standard.

Keywords: Decimal Multipliers, FPGA, Carry Save Adders

1. INTRODUCTION

The majority of the world's commercial and financial data is stored and manipulated in decimal form. A simple example is a pocket calculator, which is based on some form of decimal arithmetic. Currently, general purpose computers do decimal computations using binary arithmetic. Binary data can be stored efficiently and manipulated very quickly on two-state computers. Additional points favoring binary arithmetic include better error characteristics and less hardware to implement the same function. However, there are compelling reasons to consider decimal arithmetic, particularly for business computations. The reasons include human's natural affinity for decimal arithmetic and the inexact mapping between some decimal and binary values. Binary floating point values can only approximate certain common decimal numbers. For example a value of 0.1 requires an infinitely recurring binary pattern of zeros and ones. When an average user performs a calculation such as addition of 0.1 and 0.9, the expected result is 1.0. The user would find it very confusing to be presented with an answer of 0.999999. In this world of precision, such errors generated by conversion between decimal and binary formats are no more tolerable. In many cases, the law requires that results of financial calculations performed on a computer exactly match those carried out using pencil and paper. That can be achieved only if the calculations are executed in decimal. Recently, support for decimal arithmetic has received increased attention due to this growing importance in financial analysis, banking, tax calculation, currency conversion, insurance, telephone billing and accounting which cannot tolerate such errors.

Due to the increasing significance of decimal arithmetic, standard specifications are recently added to the draft revision of the IEEE 754 Standard for Floating-Point Arithmetic [1]. The new IEEE 754R standard defines a single data type that can be used for integer, fixed-point and floating-point decimal arithmetic. Hardware support for decimal operations, however, has been limited. But the scenario is set to change with the cost of die space continually dropping and the significant speedup achievable in hardware [2]. But till now, there is little in the way of hardware assist for financial applications that perform operations on data stored in decimal form. This is because decimal arithmetic operations are typically more complex, slower and occupy more area leading to more power and less speed when implemented in hardware. Hence, the major consideration while implementing decimal arithmetic is to enhance its speed and reduce area as much as possible.

General-purpose processors, such as those from AMD and Intel, provide the ability to add and subtract values stored in decimal format. More-complex operations like multiplication and division must be constructed from the ground up using shifts, addition and subtraction. To speed up such engineering and scientific calculations, today's computers include high-performance, floating-point coprocessors. Nowadays, Field Programmable Gate Arrays (FPGAs) are frequently used for complex designs that are oriented for functioning as co-processors.

Decimal multipliers are typically implemented using an iterative approach because of their complexity. Several iterative designs for fixed-point decimal multiplication

9781-4244-3846-4/09/$25.00 ©2009 IEEE

have been proposed [3, 4, 5, 6] in the seventies and eighties. These designs iterate over the digits of the multiplier, and based on the value of the current digit, either successively add the current multiplicand or a multiple of the multiplicand. The multiples are generated using either lookup tables or developed using a subset of previously generated multiples. Usually, the entire multiplicand is multiplied by one multiplier digits to generate a partial product in each cycle. The partial product is added to an intermediate product register that holds the previously accumulated partial products. Several existing designs for decimal multiplication generate and store multiples of the multiplicand before partial product generation. Then the multiplier digits are used to select the appropriate multiple as the partial product [7]. The multiplier presented in [8] makes use of a secondary set of multiples generated using combinational logic. This design uses dedicated hardware operating at high frequencies with relatively low latencies. The multiplier in [9] stores intermediate product digits in a less restrictive, redundant format called the overloaded decimal representation that reduces the delay of the iterative portion of the multiplier. An alternative approach is to generate the partial product as needed. This leads to less wiring and elimination of registers to store multiples of the multiplicand [10]. An integral building block of a decimal digit by digit multiplier is the single digit multiplier. The single digit multiplier in [11] uses a standard 4 × 4 unsigned binary multiplier that generates an 8-bit binary output that needs to be corrected to two decimal digits.

Decimal Floating point has 3 representations – 32 bit format with 7 significand digits, 64 bit format with 16 significand digits and 128 bit format with 34 significand digits. Fixed point multiplication of significands is an integral component of floating point multiplication. This paper presents double digit fixed-point decimal multiplication that offers low latency and high throughput. The proposed multiplier generates multiplicand multiples for 2 digits simultaneously and uses decimal carry-save addition in the iterative portion of the design. When multiplying two n-digit operands to produce a 2n-digit product, the design has an initiation interval of $\lceil (n/2)+1 \rceil$ cycles. The paper presents area and delay comparison for implementations of 7 digit, 16 digit and 34 digit DDDM on different families of Xilinx, Altera, Actel and Quick logic FPGAs.

The organization of the paper is as follows: Initially, the approach for double digit multiplication is discussed. DDDM for 7 digits, 16 digits and 34 digits are synthesized using VHDL. Finally, the paper concludes by tabulating a comparison of area and delay analysis of the proposed design for various lengths on different families of Xilinx, Altera, Actel and Quick logic FPGAs.

2. DECIMAL MULTIPLICATION

A decimal multiplier multiplies an *n*-digit multiplicand, A, by an *n*-digit multiplier, B producing a *2n*-digit product, P. A straightforward approach to decimal multiplication is to iterate over the digits of the multiplier, *B*, and based on the value of the current digit, B_i, successively add multiples of A to a product register [12]. The multiplier is accessed from least significant digit to most significant digit, and the product register is shifted one digit to the right after an iteration corresponding to division by 10. This approach allows an *n*-digit adder to be used to add the multiples of A to the partial product register. The multiples 2A through 9A, called primary multiples, are calculated at the start of the algorithm and stored along with A to reduce delay. The disadvantages of this approach are the enormous area or delay required for generating all the eight multiples, and the eight additional registers needed to store these multiples. An alternative method is to find a reduced set of multiples called secondary multiples. For example, if 2A, 5A, and 8A are computed and stored along with A, all the other multiples can be obtained with, at most, a single addition. This reduced set of multiples is called a secondary set, as no more than two members of the set need to be added to generate a missing multiple. Another reduced set of multiples comprising A, 2A, 4A, and 8A has a one-to-one correspondence with the weighted bits of a BCD digit. The disadvantage is that certain missing multiples can be generated only by the addition of 3 multiples from the reduced set. For example, the generation of 7A requires the addition of three multiples: A, 2A, and 4A. Although the secondary multiple approach reduces the delay or area and register count, it introduces the overhead of potentially one more addition for each iteration. The multiplier design proposed by [8] uses decimal carry-save addition to reduce this overhead. It gives a decimal multiplication algorithm suitable for high-performance with short cycle times. Since the floating point multiplier may need to handle operands up to 34 decimal digits further improvements in latency are suggested in this research using a double digit decimal multiplication technique.

3. DOUBLE DIGIT MULTIPLICATION

The block diagram for the DDDM is shown in Fig. 1. The 'Secondary Multiple Generation Block' generates secondary multiples 2A, 4A and 5A of length (n+1) digits. This is a purely combinational block with a maximum delay of 6 gates [8]. The multiplier input, B is loaded into the 'Multiplier Shift Register' using an asynchronous load input. Suitable secondary multiples are selected by using two pairs of multiplexers for the two digit multiplier shift register output using Table 1.

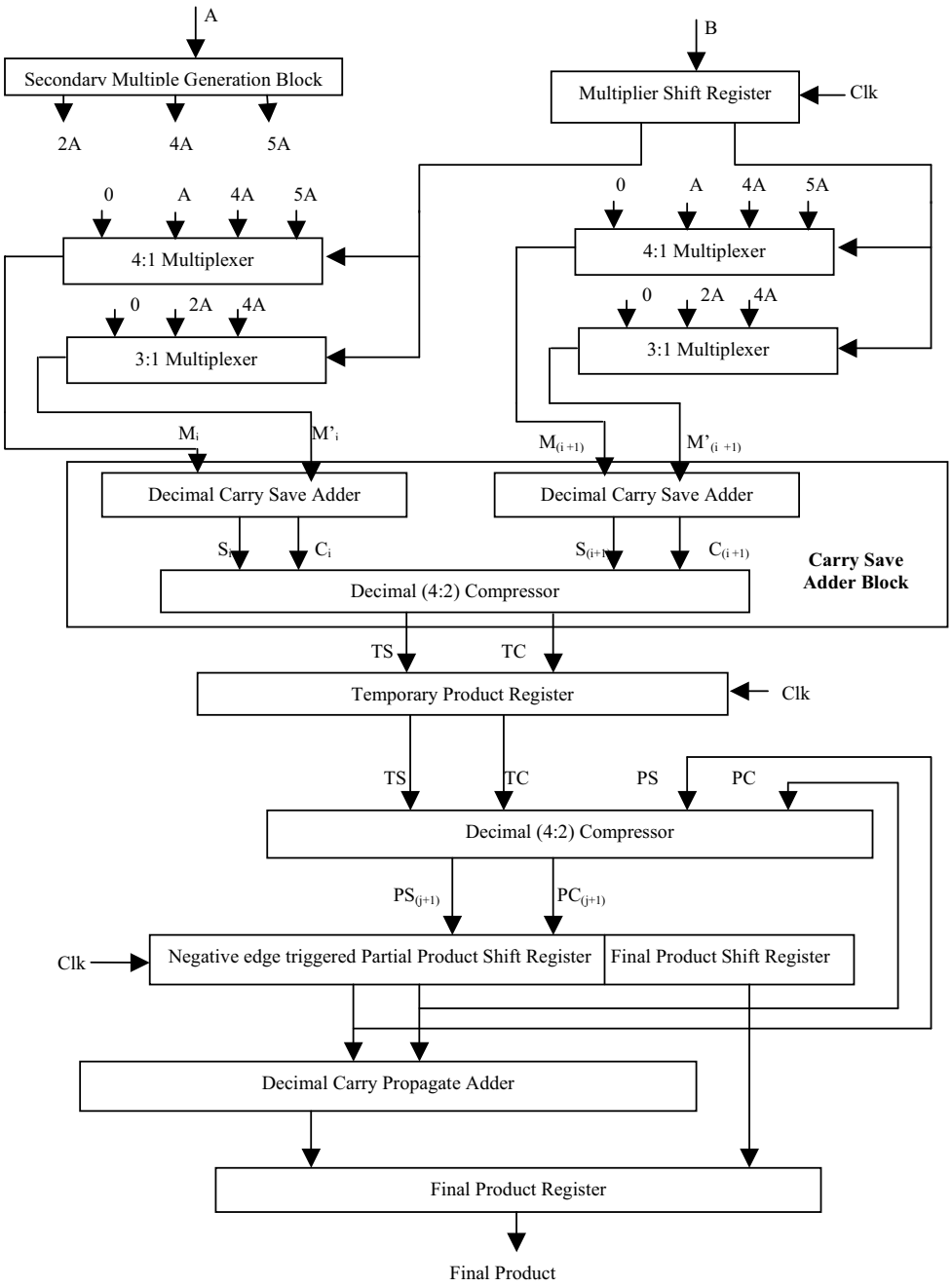

Fig. 1. Block Diagram of the Double Digit Fixed Point Decimal Multiplication

The 'Decimal Carry save Adder Block' adds the two selected secondary multiplies using carry save addition and generates an (n+1) digit sum output (S_i) and an (n+1) bit carry output (C_i). Similar addition is done by the second decimal carry save adder to produce $S_{(i+1)}$ and $C_{(i+1)}$. These four outputs are now added by a 4:2 compressor to give temporary sum (TS) and temporary carry (TC) values, of length (n+1) digits and (n+1) bits respectively.

Table 1: Recoding of Digits of B_i.

Bi	Mi	M'i	Bi	Mi	M'i
0	0	0	5	5A	0
1	A	0	6	4A	2A
2	0	2A	7	5A	2A
3	A	2A	8	4A	4A
4	0	4A	9	5A	4A

The TS and TC values stored in the 'Temporary Product Registers' are added with the shifted output of the previous partial product (PS_i and PC_i) in the 'Partial Product Register' using a 4:2 compressor to get a new partial product. The last two digits of the partial product formed is a part of the final product. The new partial product is stored in the 'Partial Product Shift Register' at the negative edge of the clock in shifted form. For this purpose, the data in the 'Final Product Shift Register' is shifted for 2 digits during the previous positive edge, giving room to store the new 2 digits of the final product during the negative clock edge.

For each iteration cycle, the multiplicand is multiplied by 2 digits of the multiplier. The partial product formed is shifted by 2 digits and the process is repeated for $\lceil (n/2) \rceil$ iterations. After $\lceil (n/2) \rceil$ iterations the final product in the form of 'carry save' sum and carry is available at the output of the 'Partial Product Shift Register'. This is then passed to a 'Decimal Carry Propagate Adder', which is actually a Decimal Incrementer. Decimal Incrementer is shown in Fig. 2. It is a circuit that adds a single bit (C_i) to a decimal digit ($X_{i(3)}$) along with the carry in $C_{o(i-1)}$, and gives the result in decimal with a carry out (C_{oi}).

$$C_{oi} = X_{i(3)} C_i C_{o(i-1)} + X_{i(3)} X_{i(0)} C_i + X_{i(0)} C_{o(i-1)} \quad (1)$$

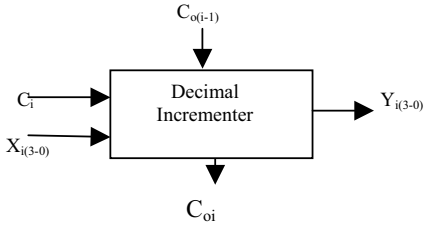

Fig. 2. Decimal Incrementer

The C_{oi} is generated after 2 gate delays for each digit. For 'n' digit multiplication, the ripple delay for C_{oi} at the final Decimal Propagate Adder is '2n' gate delays. The Boolean expressions for a single digit Decimal Incrementer are given in equations 2-5. Total delay of the 'Decimal Carry Propagate Adder' is the delay of one digit Decimal Incrementer and '2n' gate delays. This is much less than the delay of an 'n' digit BCD ripple adder. The adder output is then stored in the 'Final Product Register'. The final product is available after $\lceil (n/2)+1 \rceil$ clock cycles.

4. SYNTHESIS RESULTS

In FPGAs, the choice of the optimum multiplier depends on area and propagation time. These parameters are studied for DDDM for 7-digits, 16-digits, 34-digits and respective implementation results are tabulated. The implementations are written in VHDL to synthesize, place, and route the design. The designs were implemented in different families of Xilinx, Altera, Actel and Quick logic FPGAs and the results that were generated are tabulated in Tables 2, 3 and 4. Estimated operating frequencies and logic resource utilization are compiled using the place and route tool log files. Table 2 shows the implementation results of DDDM for 7 digits on different families of various FPGAs. Table 3 and Table 4 show the implementation results of double digit fixed point multiplier for 16 digits and 34 digits respectively on different families of various FPGAs. The study reveals that efficient mapping of the double digit multiplier is achieved when Xilinx FPGA devices are used.

5. CONCLUSION

This paper proposed double digit decimal fixed point multipliers that can be used in floating point multiplier circuits. This design leads to more regular VLSI implementation, and does not require special registers for storing easy multiples. The design was validated using lengths of 7 digits, 16 digits, and 34 digits multipliers that are required for all the three formats of floating point decimal multiplication. The latency for the multiplication of two n-digit BCD operands is $\lceil (n/2)+1 \rceil$ cycles, and a new multiplication can begin every $\lceil (n/2) \rceil$ cycle.

$$Y_{i(3)} = X_{i(3)}\overline{C_i}\,\overline{C_{0(i-1)}} + (X_{i(2)}X_{i(1)}X_{i(0)} + X_{i(3)}\overline{X_{i(0)}})(C_i \oplus C_{0(i-1)}) + \overline{X_{i(3)}}X_{i(2)}X_{i(1)}C_iC_{0(i-1)} \quad (2)$$

$$Y_{i(2)} = X_{i(2)}\overline{C_i}\,\overline{C_{0(i-1)}} + (X_{i(1)}X_{i(0)} + \overline{X_{i(3)}}X_{i(2)})(C_i \oplus C_{0(i-1)}) + (\overline{X_{i(3)}}X_{i(2)}\overline{X_{i(1)}} + \overline{X_{i(2)}}X_{i(1)})C_iC_{0(i-1).)} \quad (3)$$

$$Y_{i(1)} = X_{i(3)}\overline{C_i}\,\overline{C_{0(i-1)}} + (X_{i(1)}\overline{X_{i(0)}} + \overline{X_{i(3)}}\,\overline{X_{i(1)}}X_{i(0)})(C_i \oplus C_{0(i-1)}) + \overline{X_{i(3)}}\,\overline{X_{i(1)}}C_iC_{0(i-1)} \quad (4)$$

$$Y_{i(0)} = X_{i(0)} \oplus C_i \oplus C_{0(i-1))} \quad (5)$$

Table 2 : Implementation results of Double Digit Decimal Multipliers (7 Digit) on FPGAs

FPGA	Family	Speed Grade	Utilized Area (Logic Cells)	Maximum Frequency(M Hz)	Delay (ns)
ALTERA	APEX 20 KE	-3	1109	22	47.04
	APEX 20 K	-3	1110	20.9	49.66
	ACEX 1 K	-3	1072	32.9	29.46
	FLEX 10 KE	-3	1072	25.2	37.54
ACTEL	Family	Speed Grade	Utilized Area (Core Cells*/Modules)	Maximum Frequency(M Hz)	Delay (ns)
	3200DX	-3	1544*	10.1	98.08
	A500K	STD	2071*	20.7	47.6
	54SXA	-3	1526	15.6	63.78
	RT54SX	-1	1819	15.1	65.86
XILINX	Family	Speed Grade	Utilized Area (*Gates/ CLBs)	Maximum Frequency(M Hz)	Delay (ns)
	Cool Runner	-6	*6804	0.5	2145.80
	Spartan	-4	504	18.7	54.64
	Spartan 2	-5	525	31.5	33.35
	Spartan XL	-4	504	18.7	54.64
	Virtex	-4	525	30.4	34.11
	Virtex E	-6	525	45.2	25.23
	XC9500XV	-7	4425	28.6	35.00
Quick Logic	Family	Speed Grade	Utilized Area (Logic Cells)	Maximum Frequency(M Hz)	Delay (ns)
	pASIC3	-1	1139	8.5	132.46

Table 3: Implementation results of Double Digit Decimal Multipliers (16 Digit) on FPGAs.

FPGA	Family	Speed Grade	Utilized Area (Logic Cells)	Maximum Frequency(M Hz)	Delay (ns)
ALTERA	APEX 20 KE	-3	2446	22	107.38
	APEX 20 K	-3	2447	20.9	113.60
	ACEX 1 K	-3	2405	32.9	34.95
	FLEX 10 KE	-3	2405	25.1	85.07
ACTEL	Family	Speed Grade	Utilized Area (Core Cells*/Modules)	Maximum Frequency(M Hz)	Delay (ns)
	3200DX	-3	3348	10.1	206.39
	A500K	STD	4469*	20.1	83.14
	54SXA	-3	3357	15.0	127.20
	RT54SX	-1	3937	14.6	129.88
XILINX	Family	Speed Grade	Utilized Area (*Gates/ CLBs)	Maximum Frequency(M Hz)	Delay (ns)
	Cool Runner	-15	*15588	0.5	2129.30
	Spartan 2	-6	1152	36.0	59.08
	Virtex	-6	1152	40.1	51.92
	Virtex E	-8	1152	58.6	36.75
Quick Logic	Family	Speed Grade	Utilized Area (Logic Cells)	Maximum Frequency(M Hz)	Delay (ns)
	pASIC3	-1	2500	8.6	297.14

Table 4: Implementation results of Double Digit Decimal Multipliers (34 Digit) on FPGAs

FPGA	Family	Speed Grade	Utilized Area (Logic Cells)	Maximum Frequency(M Hz)	Delay (ns)
ALTERA	APEX 20 KE	-3	12279	22	306.82
	APEX 20 K	-3	12283	20.9	325.57
	ACEX 1 K	-3	12191	32.9	112.54
	FLEX 10 KE	-3	12191	32.7	194.33
ACTEL	Family	Speed Grade	Utilized Area (Core Cells* /Modules)	Maximum Frequency(M Hz)	Delay (ns)
	3200DX	-3	17134	9.5	616.70
	A500K	STD	22048*	19.6	255.36
	54SXA	-3	16213	15.2	343.90
	RT54SX	-1	18759	14.7	346.58
XILINX	Family	Speed Grade	Utilized Area (*Gates/ CLBs)	Maximum Frequency (M Hz)	Delay (ns)
	Cool Runner	-15	*77218	0.5	2147.43
	Virtex E	-8	5765	59.9	99.49
Quick Logic	Family	Speed Grade	Utilized Area (Logic Cells)	Maximum Frequency(M Hz)	Delay (ns)
	pASIC3	-1	12427	8.6	870.73

The paper presents area and delay analysis on different families of Xilinx, Altera, Actel and Quick logic FPGAs for double digit decimal multiplier implementations. The study reveals that efficient mapping of the double digit multiplier is achieved when Xilinx FPGA devices are used. This design can be developed into an IP core for FPGA. Using this IP core, and development tools, designers can effectively create multipliers to meet their individual requirements. Future research focuses on implementing floating point multipliers using the proposed fixed point multiplier design.

6. REFERENCES

[1] IEEE Standards Committee, "IEEE Standard for Floating-Point Arithmetic." http : // 754 r.ucbtest.org / drafts/754r.pdf, February 2003.

[2] M. A. Erle, J. M. Linebarger, and M. J. Schulte, "Potential Speedup Using Decimal Floating-Point Hardware." 36th Asilomar Conference on Signals, Systems and Computers, Nov 2002.

[3] V R. H. Larson, "High Speed Multiply Using Four Input Carry Save Adder," IBM Technical Disclosure Bulletin, pp. 2053–2054, December 1973.

[4] T. Ohtsuki, Y. Oshima, S. Ishikawa, K. Yabe, and M. Fukuta, "Apparatus for Decimal Multiplication," U.S. Patent, Jun 1987. #4,677,583.

[5] R. L. Hoffman and T. L. Schardt, "Packed Decimal Multiply Algorithm," IBM Technical Disclosure Bulletin, vol. 18, pp. 1562–1563, October 1975.

[6] J. J. Bradley, B. L. Stoffers, T. R. S. Jr., and M. A. Widen, "Simplified Decimal Multiplication by Stripping Leading Zeros," U.S. Patent, Jun 1986. #4,615,016.

[7] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlough, "The IBM Z900 Decimal Arithmetic Unit," in Asilomar Conference on Signals, Systems, and Computers, vol. 2, pp. 1335– 1339, November 2001

[8] M. A. Erle and M. J. Schulte, "Decimal Multiplication Via Carry-Save Addition," IEEE 14th International Conference on Application-specific Systems, Architectures and Processors, pp. 348-358, June 2003

[9] R. D. Kennedy, M. J. Schulte and M. A. Erle, "A High-Frequency Decimal Multiplier," IEEE 14th International IEEE international conference on Computer Design (ICCD'04), pp. 22-29, Oct 2004

[10] Erle, M.A. Schwarz, E.M. Schulte, M.J, "Decimal multiplication with efficient partial product generation**",** 17th IEEE Symposium on Computer Arithmetic, 2005. ARITH-17 2005. pp. 21- 28, June 2005

[11] Jaberipur, G.; Kaivani, A, "Binary-coded decimal digit multipliers", Computers & Digital Techniques, IET Volume 1, Issue 4, July 2007 pp. 377 – 381

[12] R. K. Richards, Arithmetic Operations in Digital Computers. New Jersey: D. Van Nostrand Company, Inc., 1955.