ISSN 2348 – 8034
Impact Factor- 3.155

GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

FPGA IMPLEMENTATION OF HARDWARE EFFICIENT SEQUENTIAL DECIMAL FIXED POINT MULTIPLER

O.Vignesh
Teaching Fellow, Department of ECE, Anna University Regional Office Coimbatore, India

ABSTRACT

The hardware realization of the decimal multiplication where a novel algorithm and a corresponding architecture are proposed to reduce the area of decimal multiplication while keeping the latency in a reasonable rate. In the sequential architecture, the partial product generation and selection cycles are reduced to one. Moreover, the selected easy multiples reduce the hardware requirement of the partial products selector. With simpler combinational logics and less registers, our design shows a much better performance on the area cost. The latency of a multiplication in the proposed architecture is shortened by a less product of the clock period and number of iterations. The proposed multiplier implemented in Cyclone II FPGA and simulated in ALTERA QUARTUS II. The simulation results as a power, delay, speed of proposed multiplier is compared with start and art multiplier. The proposed multiplier is adapted to the arithmetic logic unit circuit in general purpose processors.

Keywords: *Decimal multiplication, Latency, PPG, FPGA Implementation.*

I. INTRODUCTION

In this paper the design of a 2-bit sequential multiplier is designed with 8-bit A and B inputs and a 16-bit result. This multiplier has an 8-bit bi-directional I/O for inputting its A and B operands, and outputting its 16-bit output one byte at a time. Multiplication begins with the start pulse, and the data bus will contain operands A and B in two consecutive clock pulses. After accepting these data inputs, the multiplier begins its multiplication process and when it is completed, it starts sending the result out on the data bus. When the least significant byte is placed on data bus, the LSB out output is issued, and for the most-significant byte, MSB out is issued. When both bytes are outputted, done becomes 1, and the multiplier is ready for another set of data. The multiplexed bi-directorial data bus is used to reduce the total number of pins of the multiplier.

The decimal numeral system (also called base ten or occasionally denary) has ten as its base. It is the numerical base most widely used by modern civilizations. Decimal notation often refers to a base-10 positional notation such as the Hindu-Arabic numeral system; however, it can also be used more generally to refer to non-positional systems such as Roman or Chinese numerals which are also based on powers of ten. Decimals also refer to decimal fractions, either separately or in contrast to vulgar fractions. In this context, a decimal is a tenth part, and decimals become a series of nested tenths. There was a notation in use like 'tenth-meter', meaning the tenth decimal of the meter, currently an Angstrom. The contrast here is between decimals and vulgar fractions, and decimal divisions and other divisions of measures, like the inch.

In integer parts or integral part of a decimal number is the part to the left of the decimal separator. The part from the decimal separator to the right is the fractional part. It is usual for a decimal number that consists only of a fractional part (mathematically, a proper fraction) to have a leading zero in its notation (its numeral). This helps disambiguation between a decimal sign and other punctuation, and especially when the negative number sign is indicated, it helps visualize the sign of the numeral as a whole. For most purposes, however, binary values are converted to or from the equivalent decimal values for presentation to or input from humans; computer programs express literals in decimal by default. Both computer hardware and software also use internal representations which are effectively decimal for storing decimal values and doing arithmetic. Often this arithmetic is done on data which are encoded using some variant of binary-coded decimal. Decimal arithmetic is used in computers so that decimal fractional results can be computed exactly, which is not possible using a binary fractional representation. This is often important for financial and other calculations.

II. SEQUENTIAL DECIMAL MULTIPLIER

ISSN 2348 – 8034
Impact Factor- 3.155

Algorithms and architectures for the decimal multiplication are usually classified into two main categories namely parallel and sequential multipliers. The former, suitable for high throughput applications, generates and accumulates partial products at once; hence huge area consumption cannot be avoided. The latter, however, generates partial products gradually (i.e., one per iteration) and accumulates them sequentially. This architecture, despite of a lower throughput, is popularly used whenever cost efficiency is the main intention [8]. Both the parallel and sequential multipliers consist of three main steps 1) partial product generation (PPG), 2) partial product accumulation (PPA), and 3) final carry propagation adder or conversion. The most popular algorithm for decimal PPG is based on the generation of easy-multiples (e.g. X, 2X, 4X, 5X) of the multiplicand X.

The easy multiples can be generated as a non-redundant decimal number via a carry-save approach. In this method, one or two of the easy-multiples are selected, based on the value of the multiplier's digit yi, to generate the appropriate partial product Pi = yi × X. Furthermore, the original multiplier's digit can also be converted into a redundant digit-set to reduce the required easy multiples. The performance and cost of the PPG depend mainly on the combination of the encodings of the easy multiples and multiplier's digit. In order to achieve the final product, one needs to generate and sum up the partial products for all digits of the multiplier. This step, as known as partial product accumulation, usually consists of a carry-propagating or carry-free decimal adder. To speed up the accumulation performance, which decides the critical path in most of the cases, the carry save or carry-free method is widely considered. However in the accumulation adder, the algorithm and corresponding encoding raise two problems, 1) it has to accept and wisely process the encoding or representation of the partial product generated from PPG, 2) the representation of the intermediate result generated in every iteration decides the number of required registers.

The last final carry propagation adder to wrap up the carry-save result in the last cycles has a similar structure and principle in most cases. It should be noted that, a final conversion which consists of a carry propagating process is required to translate the result to non-redundant representation, in case of using a carry-free decimal adder in PPA.

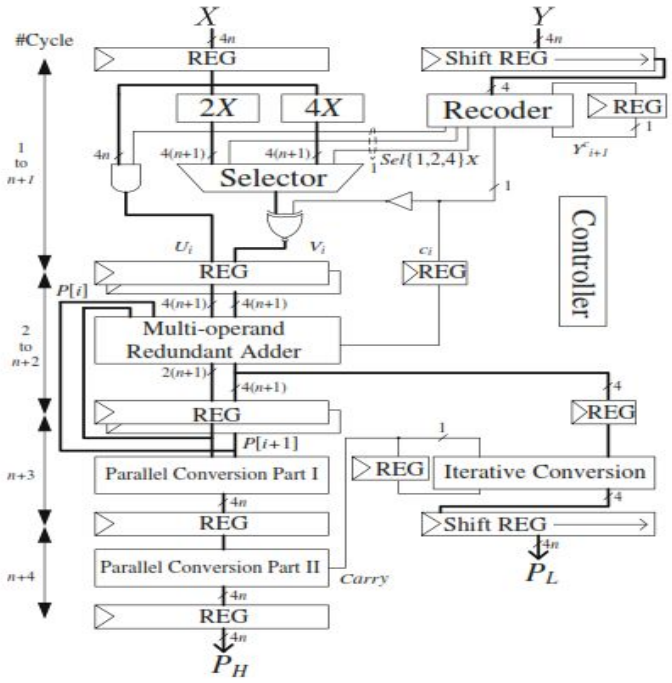


Figure 1: Block diagram of sequential decimal multiplier

III. PARTIAL PRODUCT GENERATION AND ACCUMULATION

The PPG of the proposed multiplier is based on the generation of easy-multiples of the multiplicand; thus, we need to determine the required easy-multiples. The representation of the multiplier Y plays a key role in selecting the appropriate easy-multiples. Consequently, we opt for digit-set [-4, 5] to represent the multiplier Y in order to reduce the number of required easy-multiples and hence decrease the complexity of the PPG. This, however, calls for a recoder to convert the multiplier from digit-set [0, 9] to [-4, 5]. The recoder is implemented based on Eq. 1 where y constitute the ith digit of the multiplier Y in [-4, 5] digit set.

ISSN 2348 – 8034
Impact Factor- 3.155

$$\begin{cases} y_i^s = y_i;\ y_{i+1}^c = 0 & \text{if} y_i + y_i^c \le 5 \\ y_i^s = y_i - 10;\ y_{i+1}^c = 1 & \text{if} y_i + y_i^c > 5 \end{cases} \quad (1)$$

Given that the recoded multiplier needs to be ready iteratively (i.e. one digit per iteration), the carries in Eq.1 need to be stored in a D flip-flop and used in the next iteration as shown in Fig. 2.

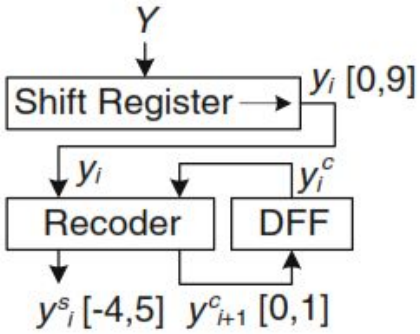


Figure 2: Block diagram of Recoding of the multiplier.

Given the digit-set of the multiplier (i.e., [-4, 5]), computing the easy multiples X, ±2X and ±4X is sufficient for generating a partial product as a sum of two decimal numbers i.e., $P_i = U_i + V$. It should be noted that the addition $U_i + V_i$ is actually performed in the PPA step. Finally, a combinational logic is required to select the appropriate easy-multiples based on the value of the multiplier's digit. Table 3.1 describes the selection rules for generating U_i and V_i.

Table 3.1 Selection of the easy-multiples

$y_i =$	−4	−3	−2	−1	0	1	2	3	4	5
$U_i =$	0	1X	0	1X	0	1X	0	1X	0	1X
$V_i =$	−4X	−4X	−2X	−2X	0	0	2X	2X	4X	4X

$$\begin{cases} t_i^1 = x_{i-1}^3 \\ t_i^0 = x_{i-1}^2 + x_{i-1}^1 \\ 2x_i^3 = \overline{x_i^0} \cdot \left(t_i^1 \cdot x_i^2 \cdot \overline{x_i^1} + x_i^3\right) + \overline{x_i^2} \cdot x_i^1 + \overline{t_i^1} \cdot x_i^3 \\ 2x_i^2 = x_i^0 \cdot \left(t_i^1 \cdot \overline{x_i^3} \cdot \overline{x_i^2} + x_i^1\right) + \overline{x_i^0} \cdot \left(\overline{t_i^1} \cdot x_i^2 \cdot \overline{t_i^1} + x_i^3\right) \\ \qquad + t_i^1 \cdot x_i^1 + \overline{t_i^1} \cdot x_i^3 \\ 2x_i^1 = \overline{x_i^2} \cdot \overline{x_i^1} \cdot \left(t_i^1 \oplus x_i^0\right) + \left(t_i^1 \odot x_i^0\right) \cdot \left(x_i^1 + x_i^2\right) \\ 2x_i^0 = t_i^0 \end{cases} \quad (2)$$

With the intention of reducing the complexity of the PPG, we managed to generate the easy-multiples in the encodings shown in Table 2; thereby simplifying the carry-free addition of the PPA step.

ISSN 2348 – 8034
Impact Factor- 3.155

Table 2 Redundant digit-set conversion of 2X and 4X

X_i	2X		4X	
	t_{i+1}	w_i	t_{i+1}	w_i
0	0	0	0	0
1	0	2	1	−6
2	1	−6	1	−2
3	1	−4	1	2
4	1	−2	2	−4
5	1	0	2	0
6	1	2	3	−6
7	1	4	3	−2
8	2	−4	3	2
9	2	−2	4	−4

Particularly, easy-multiple 1X is kept as BCD and ±2X, ±4X are encoded into digit-set [-6, 6] and represented as a signed-digit 2's complement. In this approach, first, each digit (e.g., ith) is divided into a transfer ti+1 and a sum wi (as shown in Table 2); second, wi +ti generates the converted ith digit. According to Table 2, the generations of the easy multiples 2X and 4X is performed via the logical expressions of Eqs.2 and 3. Regarding the symmetric signed-digit 2's complement representation of 2X and 4X, the-2X and -4X multiples are generated through a simple 2's complement per digit. However, the 2's complement operation is partially deferred until the PPA step.

The overall architecture of the proposed PPG is illustrated in Fig. 3, where ci is stored for the 2's complement operation (per digit) performed in the PPA step. Additionally, the "one hot selector" selects one of input signals by only one exclusive bit "1".

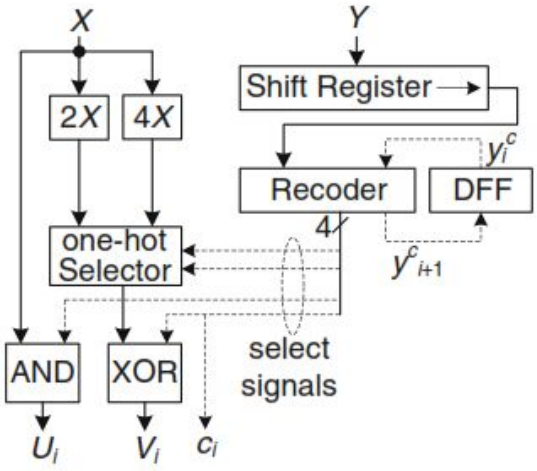


Figure 3: Block diagram of proposed partial product generation

ISSN 2348 – 8034
Impact Factor- 3.155

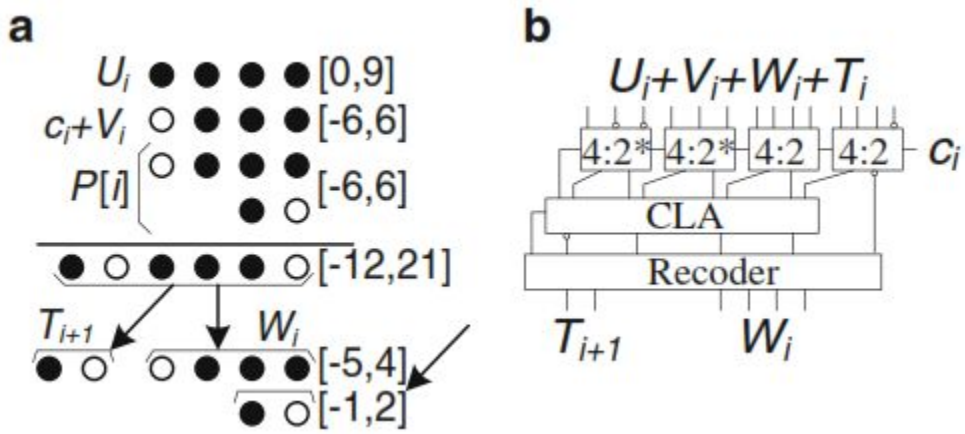
$$\begin{cases} t_i^2 &= x_{i-1}^3 \cdot x_{i-1}^0 \\ t_i^1 &= x_{i-1}^2 + x_{i-1}^3 \cdot \overline{x_{i-1}^0} \\ t_i^0 &= x_{i-1}^1 + x_{i-1}^3 \cdot \overline{x_{i-1}^0} + \overline{x_{i-1}^3} \cdot \overline{x_{i-1}^2} \cdot x_{i-1}^0 \\ 4x_i^3 &= \overline{x_i^3} \cdot \overline{x_i^2} \cdot \overline{x_i^1} \cdot x_i^0 + x_i^2 \cdot x_i^1 \cdot \overline{x_i^0} + \\ & \quad \overline{t_i^2} \cdot \left(\overline{x_i^2} \cdot \overline{x_i^1} \cdot x_i^0 + \overline{t_i^1} \cdot x_i^2 \cdot x_i^1 + \overline{x_i^0} \cdot \left(\overline{t_i^1} \cdot x_i^1 + x_i^2\right)\right) \\ 4x_i^2 &= x_i^0 \cdot \left(t_i^2 \cdot \overline{x_i^3} \cdot \overline{x_i^1} + \overline{t_i^2} \cdot \left(\overline{t_i^1} \cdot x_i^2 \cdot x_i^1 + x_i^3\right)\right. \\ & \quad \left. + \overline{x_i^2} \cdot \left(t_i^2 \cdot \overline{x_i^3} + t_i^1\right)\right) + \overline{x_i^0} \cdot \left(\overline{x_i^2} \cdot \left(\overline{t_i^2} \cdot \overline{t_i^1} \cdot x_i^1\right.\right. \\ & \quad \left.\left. + t_i^2 \cdot \overline{x_i^1}\right) + x_i^2 \cdot \left(\overline{t_i^2} \cdot \overline{x_i^1} + t_i^2 \cdot x_i^1 + t_i^1\right)\right) + t_i^1 \cdot x_i^3 \\ 4x_i^1 &= t_i^1 \cdot \left(\overline{x_i^1} \cdot \left(\overline{x_i^3} \cdot \overline{x_i^0} + x_i^2\right) + x_i^3 \cdot x_i^0\right) + \\ & \quad \overline{t_i^1} \cdot \left(\overline{x_i^3} \cdot \overline{x_i^2} \cdot x_i^0 + x_i^3 \cdot \overline{x_i^0} + x_i^1\right) \\ 4x_i^0 &= t_i^0 \end{cases} \quad (3)$$

With the intention of reducing the latency of the PPA step, one can use a multi-operand redundant adder as to implement Eq. 4, where P [i] and P [i +1] are represented in a carry-save format. Figure 3 illustrates the dot-notation and the circuitry of the multi-operand redundant addition used in the proposed PPA where (4:2) with the asterisk is a simplified compressor. To efficiently decode the intermediate sum in digit-set [-12, 21], a non-conventional transfer digit on digit-set [-12, 21] which has a negative weight bit in the least significant position is represented. Therefore, a hybrid redundant multi-operands addition is proposed based on the principle.

The partial product accumulation is applied to properly add the partial product (i.e., $U_i + V_i + c_i$,) to the accumulated previous products P[i]. This is resembled in the recurrence Eq. 3.4, where c_i is the word-wide extension of c_i (1-bit c_i per digit).

$$P[i+1] = 0.1P[i] + U_i + V_i + c_i \quad (4)$$

ISSN 2348 – 8034
Impact Factor- 3.155



a. dot-notation *b. circuitry.*

Figure 4: Block diagram of Partial Product Accumulation.

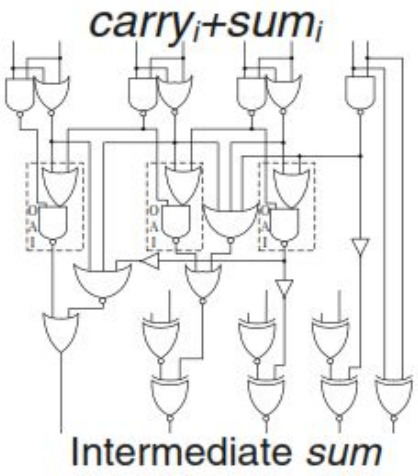


Figure 5: Structure of CLA in PPA.

ISSN 2348 – 8034
Impact Factor- 3.155

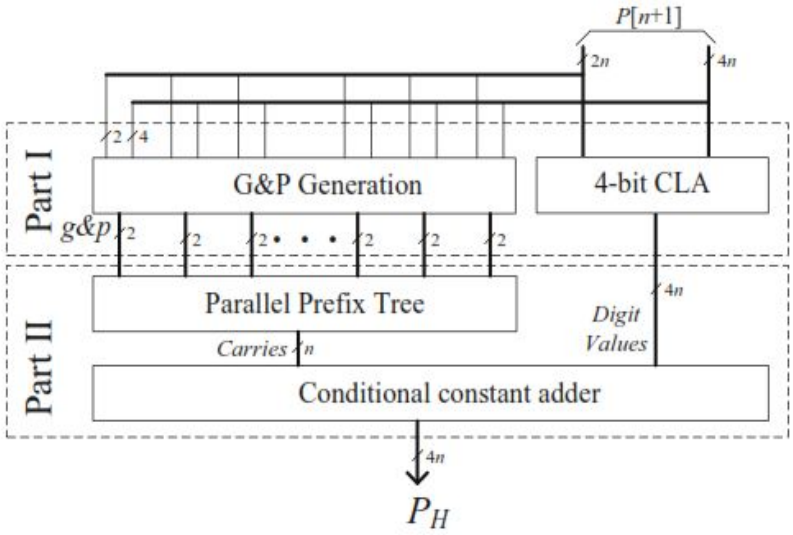


Figure 6 Block diagram of parallel conversion.

The parallel conversion which is depicted in Fig. 3.6 consists of two main parts are ,

Part I: Preparing generate and propagate signals (g and p) for the negative carry (borrow) based on decimal transfer digit in [-1, 2] and residual sum in [-5, 4]. Moreover, A 4-bit carry-look-ahead adder (CLA) is responsible to generate the appropriate digit value.

Part II: A parallel prefix tree computes the negative carry of each digit position; then a conditional constant adder produces the final converted product.

The sequential decimal multipler, consists of three main parts namely PPG, PPA and Conversion each of which consumes 1, n + 1 and 2 cycles, respectively. This concludes that the entire single multiplication can be performed in n + 4 cycles with the initiation interval of n + 1 cycles. The cycle time, thus the clock frequency, determined by the critical delay path of the PPA, is equal to the latency of the multi-operand adder.

The Multiplier Via Decimal Carry Save Adder

The cycle time of the multiplier proposed in this paper depends on the pipelining registers and a BCD (4:2) compressor which is created based on the BCD (3:2)-adder. The traditional method costs more timing delay and area in the PPA stage compared to other methods. In addition, to generate the multiples (i.e. [1X, 9X]), a decimal (3:2)-adder has to be applied in the PPG stage which increases the area cost. Subsequently, the big amount of multiplexors to select the easy-multiples in the design also aggravates the area cost. Due to the architecture, n + 4cyclesare required to perform a multiplication.

The Multiplier Via Overloaded Decimal Adder

This multiplier, using the overloaded decimal representation, calls for a special decimal carry-free adder which brings about a critical delay path of a (2:1) multiplexer, a +6 increment block, a binary full-adder plus registers. The long timing delay of the decimal (4:2)-adder is eliminated in this design by the overloaded decimal adder. However, to speed up the clock speed, the latches which are inserted into the overloaded decimal adder cause more area cost. Additionally, two clean-up blocks with many latches have to be applied to convert the digits back to the BCD format.

The Multiplier Via Svoboda's Signed-Digit Adder

ISSN 2348 – 8034
Impact Factor- 3.155

This multiplier takes advantage of the decimal signed-digitadder, introduced for the iterative portion of the PPA. The cycle time is determined by the latency of Svoboda's adder plus registers . Nevertheless, in this design, the digit-set of the multiplicand is first converted by a recoder, and the partial product is generated by a special unit which calculates the multiples in digit-sets [2−5]×[2−5]. Subsequently, an overlap removal unit has to be applied to finish the multiples generation and selection.To apply the signed digit adder, two steps which implya big area have to be performed in the PPG stage.With these logics, the number of required cycles is $n + 4$ and the area cost is 18550 NAND2 for a 16-digit multiplication.

IV. RESULTS AND DISCUSSION

The VHDL simulations were performed first to ensure the functional verification of the design using ALTERA QUARTUS-II.VHDL simulations have been performed for Proposed PPG, Structure of CLA, Proposed PPA, Proposed Parallel conversion, Proposed sequential decimal mutipler.

Simulation Output Sequential Decimal Multipler

It is an overall stage of the output. It consists of proposed PPG, structure of CLA, Proposed PPA and proposed parallel conversion.

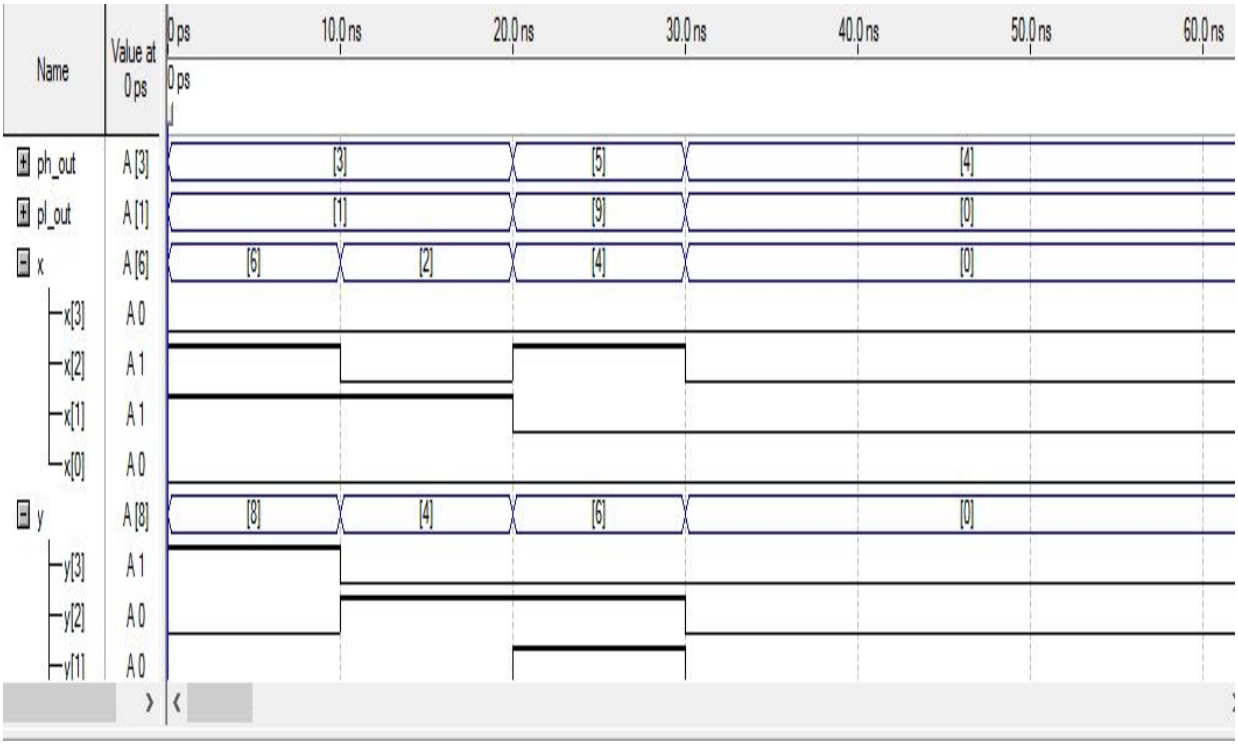


Figure 7: Output Waveform for Sequential decimal multipler

ISSN 2348 – 8034
Impact Factor- 3.155

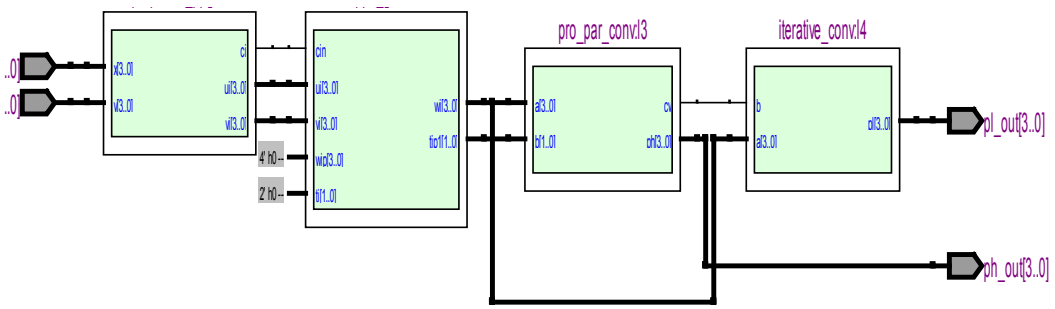


Figure 8: RTL view of sequential decimal multiplier

Performance Analysis

This section presents the evaluation results of the proposed multiplier and the comparison with the previous works, in terms of area and delay. We simulated and analyzed the entire proposed design by ALTERA QUARTUS II , Synopsys Design Compiler, and Power Compiler using the STM 90 nm CMOS standard library for 1.00 V VDD and 25 ◦C temperature in which the FO4 latency is 45 ps and the area of a NAND2 gate is 4.4 um2. The correctness of the design is verified by 50,000 random test vectors.

Table 4.1 Comparison of each stage of the sequential decimal multipler

Stages	Delay(ns)	Power
PPG	9.35ns	325.15mw
Structure of CLA	11.991ns	68.76mw
PPA	13.907ns	325.31mw
parallel conversion	9.900ns	324.57mw
sequential decimal multipler	15.529ns	324.45mw

V. CONCLUSION

In this paper, introduced a new architecture to generate and accumulate the partial products in a sequential decimal multiplier. Additionally, the less multiples in the proposed multiplier also mean a smaller selection unit. A multi-operand signed digit adder is proposed to accumulate the partial products in a reasonable timing delay. With simpler combinational logics and less registers, our design shows a much better performance on the area cost. The latency of a multiplication in the proposed architecture is shortened 6by a less product of the clock period and number of iterations. The proposed multiplier implemented in Cyclone II FPGA and simulated in ALTERA QUARTUS II. The

ISSN 2348 – 8034
Impact Factor- 3.155

simulation results as a power, speed of proposed multiplier is compared with start and art multiplier. The proposed multiplier is adapted to the arithmetic logic unit circuit in general purpose processors.

REFERENCES

1. Cornea M., et al.," A software implementation of the IEEE 754R decimal floating-point arithmetic using the binary encoding format" IEEE Transactions on Computers, 2009, Vol 58,No.2, pp.148–162.

2. Cowlishaw M.F.,"Decimal floating-point algorism for computers" in Proc. of the 16th IEEE symposium on computer arithmetic, 2011, Vol 34,No.4, pp.345-350.

3. Erle M.A., & Schulte, M.J.," Decimal multiplication via carry save addition" in Proc. IEEE Int. Conf. Application-Specific Systems, Architectures, Processors , 2003, Vol 43,No.5 pp. 337–347.

4. Erle M.A., Schwarz, E.M., Schulte, M.J. " Decimal multiplication with efficient partial product generation" in Proc. 17th IEEE symp. on computer arithmetic, 2005, Vol 23,No.8, pp. 21–28.

5. Han L., & Ko., S." High speed parallel decimal multiplication with redundant internal encodings" IEEE Transactions on Computers, 2013Vol 62,No.5, pp.956–968.

6. Jaberipur G., & KaivaniA.,. "Improving the speed of parallel decimal multiplication" IEEE Transactions on Computers, 2009Vol 58,No.11, pp.1539–1552.

7. Jaberipur G., & Parhami B.," Constant-time addition with hybrid-redundant numbers: theory and implementations" Integration, the VLSI Journal, 2008, Vol 41,No.6, pp.49–64.

8. Kenney R.D., Schulte M.J., Erle M.A., " A high-frequency decimal multiplier" in Proc. IEEE int. conf. comput. des.: VLSI in comput. and processors , 2004Vol 65,No.5,pp. 26–29.

9. Svoboda A., "Decimal adder with signed digit arithmetic" IEEE Transaction on Computers, 1969, Vol 78,No. 4, pp.212–215.

10. Vazquez A., Antelo E.,Montuschi P., ," Improved design of high-performance parallel decimal multipliers" IEEE Transactions on Computers, 2010, Vol 59,No.5, pp.679–693.