PARTIAL PRODUCT GENERATION OF RADIX-10 PARALLEL DECIMAL MULTIPLIER IN FPGA

MINOR PROJECT

By:

Ashish Kumar Srivastava (172VL005)

Vishakh V (172VL025)

(M. Tech VLSI Design)

Under the guidance of Dr. Aparna P

Dept. of Electronics & Communication Engineering

National Institute of Technology Karnataka, Surathkal

CONTENTS **Page**

INTRODUCTION

Multipliers are being increasingly used in DSP processors, filters, communications systems etc. With the rising complications of technology, high-speed systems are in great demand. On comparison with other operation in an arithmetic logic unit the multiplier consumes more time and power. Hence the demand to design or implement multipliers with optimal speed, power and area has increased.

This project includes the implementation of decimal multipliers which are arranged in parallel, with the idea of reducing delay. The partial products are generated in parallel by using signed digit radix-10 recoding of the multiplier and a simplified box of multiplicand multiples. Number of partial products are reduced by creating a tree structure of partial products. The same is developed by using a new algorithm known as decimal multi-operand carry save addition. This uses unconventional decimal coded number systems, which largely improves the area and latency of the prior or existing design. It includes optimized digit recorders, decimal carry-save adders (CSA's) combining different decimal-coded operands. The generation of partial products are developed parallel by using signed-digit (SD) radix-10 recordings of the multiplier and a simplified set of multiplicand multiples.

The modules have been designed in Verilog HDL, simulated and synthesized using Xilinx Vivado 2017.3.

FIXED-POINT DECIMAL MULTIPLICATION

The decimal integer of any operand $Z = \sum_{i=0}^{d-1} Z_i 10^i$ can be represented as a positive weighted 4-bit vector as

$$Z_i = \sum_{i=0}^{3} z_{i,j} r_j$$

Where $z_{i,j}$ is the j^{th} bit of the i^{th} digit, $Z \epsilon [0,9]$ is the i^{th} decimal digit and $r_i \geq 1$ is the weight of the j^{th} bit. Table 1 shows the set of coded decimal no. system that consist BCD (with $r_j = 2^j$) the other. The codes are referred by their weight bits as (r_3, r_2, r_1, r_0). The 4-bit vector Z_i in the coded decimal number (r_3, r_2, r_1, r_0) is represented by $Z_i(r_3, r_2, r_1, r_0)$.

The multiplicand $X = \sum_{i=0}^{d-1} X_i 10^i$ and multiplier $Y = \sum_{i=0}^{d-1} Y_i 10^i$ are unsigned decimal integer

d-digital BCD words. Multiplication consists of the three aspects generation of partial product, reduction (addition) of partial product and finally carry propagation addition (conversion to non-redundant 2d digit BCD representation.

$$P = \sum_{i=0}^{2i-1} p_i 10^i$$

RADIX-10 ARCHITECTURE

Figure shows the architecture of d-digit SD-radix multiplier this multiplier consists of three stages generation of partial product [decimal] coded in 4221 (generation of multiplicand multiplies and SD radix-10 encoding of the multiplier), reduction of partial product and a finally carry propagate addition (BCD). The multiplier is encoded into d-SD radix-10 digit and an additional bit to generate d+1 partial products. SD radix-10 digit controls 5:1 mux to select a positive multi operand multiplies 0,X,2X,3X,4X,5X)coded in 4221. The output bits of mux is inverted by XOR gate. When the sign of SD radix-10 is negative, to find the partial product. Next, the d+1 partial product is coded (4221) decimal digit using P:2 CSA trees. The digits to be condensed for each column ranges from p=d+1 to p=2. thus, the two 2d-digit operands S and H are formed from d+1 partial products.

The resultant product is 2d-digit BCD word denoted by P=2H+S. S and H are required to be progressed before being added. S is recoded from (4221) to BCD excess-6 (BCD plus 6). The H x2 multiplication is executed along with the recoding of S. The X2 requires a (4221) to (5421) digit recorder and a 1-bit wired left shift to obtain 2H. Finally, BCD carry – propagate addition is obtained by using quaternary tree (Q-T) adder

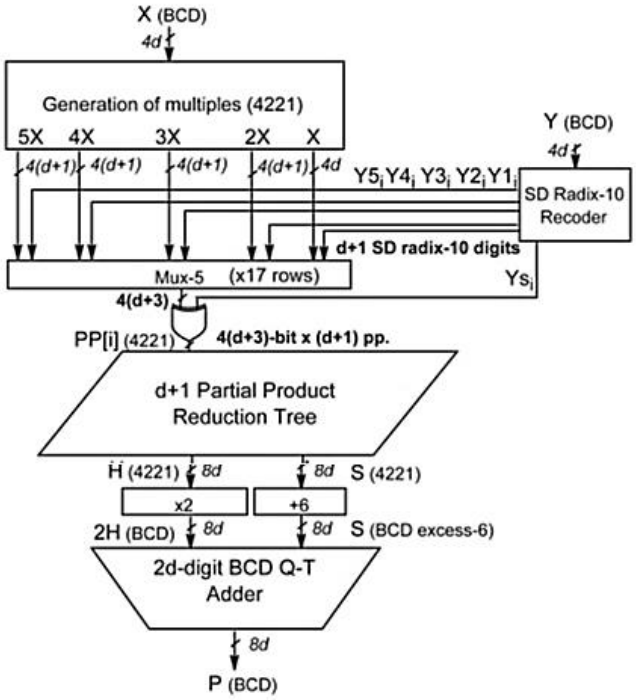


Figure 1 – Radix -10 Parallel Decimal Multiplier

SD RADIX-10 RECODING

The block diagram of generation of partial product by using the SD radix-10 recoding. This recoding transforms a BCD digit Y_i in {0,.....,9} into a SD radix-10 Yb_i in {-5,.....,5} .The value of the recoded digit *Ybi* depends upon the value of *Yi* and on a signal ys_{i-1} (sign signal) this shows if y_{i-1} is greater than or equal to S. Thus, the d-digit BCD multiplier *Y* is recoded into the d+1 digit SD radix-10 multiplier is refer by

$$Yb = \sum_{i=0}^{d} Yb_i 10^i \text{ with } Yb_d = ys_{d-1} \in \{0,1\}.$$

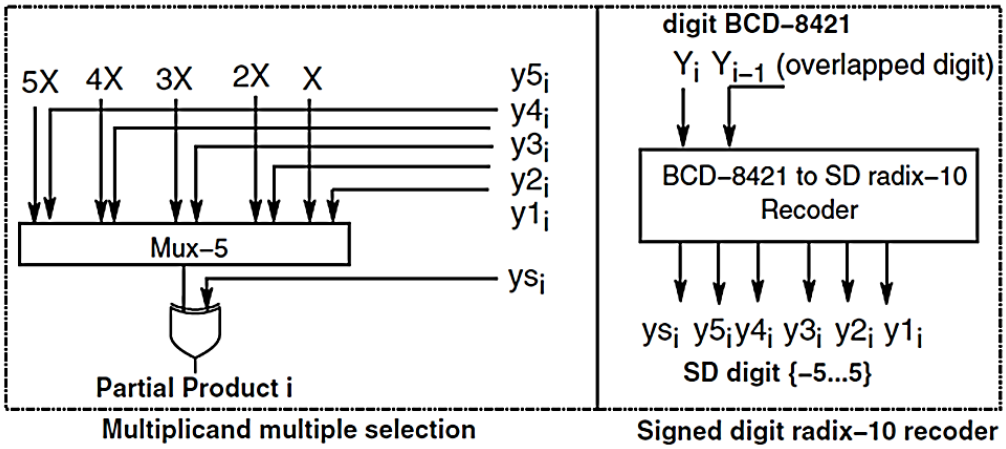


Figure 2 – Partial product generation for SD radix-10

Each digit Yb_i generates a partial product PP[i] selecting the proper multiplicand multiple coded in (4221). This is performed in a similar way to a modified booth recoding. Represented as five "hot one code" signals $\{y1_i, y2_i, y3_i, y4_i, y5_i\}$ and a sign bit ys_i. These signals are attained

directly from the BCD multiplier digits Y_i using the following logical expressions:

$$ys_i = y_{i,3}\ V\ y_{i,2}.(y_{i,1}\ V\ y_{i,0})$$

$$y5_i = y_{i,2}\ V\ \overline{y_{i,1}}.(y_{i,0} \oplus ys_{i-1})$$

$$y4_i = ys_{i-1}.y_{i,0}.(y_{i,2} \oplus y_{i,1})\ V\ \overline{ys_{i-1}}.y_{i,2}.\overline{y_{i,0}}$$

$$y3_i = y_{i,1}.(y_{i,0} \oplus ys_{i-1})$$

$$y2_i = \overline{ys_{i-1}}.\overline{y_{i,0}}.(y_{i,0}\ V\ \overline{y_{i,2}}.y_{i,1})\quad V\ ys_{i-1}.\overline{y_{i,3}}.y_{i,0}.\overline{y_{i,2} \oplus y_{i,1}}$$

$$y1_i = \overline{y_{i,2} V y_{i,1}}.(y_{i,0} \oplus ys_{i-1})$$

Symbols V, $\cdot$, and $\oplus$ specifies Boolean operators OR, AND, and XOR, respectively. The five "hot one code" signals are used as control signals for 5:1 muxes to select the positive d+1 digit multiples {0, X, 2X, 3X, 4X, 5X}. To find the partial product, the nominated positive multiple is 10's complemented if ys_i bit is one. This is done simply by a bit inversion of the positive (4221) decimal –coded multiple using a row of XOR gates that are controlled by ys_i .The accumulation of one *ulp* (unit in the last place) is done enclosing a tail – encoded bit ys_i (hot one) to the next

significant partial product *PP*[*i*] . To remove a sign extension, and thus, to reduce the difficulty of the partial product reduction tree, the Partial product sign bits ys_i are coded at every leading position into two digits as:

$$(pp[i]_{d+2}, pp[i]_{d+1}) = \begin{cases} (\overline{ys_0}, ys_0\ ys_0\ ys_0\ ys_0), & i=0, \\ (0,111\overline{ys_i}), & 0<i<d-1 \\ (0,0000), & i=d-1. \end{cases}$$

GENERATION OF MULTIPLICAND MULTIPLES

The block diagram for the generation of the positive multiplicand multiples (*X*, 2*X*, 3*X*, 4*X*, 5*X*) for SD radix-10 recoding. All these multiples are encoded (4221). The *X* BCD multiplicand can be easily recoded to (4221) using the logical expressions

$$(w_{i,3}, w_{i,2}, w_{i,1}, w_{i,0}) = (x_{i,3}\ V\ x_{i,2}, x_{i,3}, x_{i,3}\ V\ x_{i,1}, x_{i,0})$$

where $x_{i,j}\ and\ w_{i,j}$ are respectively, the bit of the BCD and (4221) representation of *X* and *W* respectively. The generation of multiples is as follows:

Decimal coding

Z_i	$Z_i(BCD)$	$Z_i(5421)$	$Z_i(4221)$	$Z_i(5211)$	$Z_i(4311)$	$Z_i(3321)$
0	0000	0000	0000	0000	0000	0000
1	0001	0001	0001	0001 0010	0001 0010	0001
2	0010	0010	0100 0010	0100 0011	0011	0010
3	0011	0011	0101 0011	0101 0110	0100	0100 1000 0011
4	0100	0100	0110 1000	0111	1000 0110 0101	1001 0101
5	0101	1000	0111 1001	1000	1001 0111 1010	1010 0110
6	0110	1001	1010 1100	1010 1001	1011	1100 1011 0111
7	0111	1010	1011 1101	1011 1100	1100	1101
8	1000	1011	1110	1110 1101	1110 1101	1110
9	1001	1100	1111	1111	1111	1111

Multiples 2*X:* Table 1 show the BCD digit recoded into (5421) decimal coding. An L1shift is executed to the recoded multiplicand, getting the 2X BCD multiples. Then by using expression the 2X BCD multiples is recoded again.

Multiples 4*X:* It is obtained as 2*X* x2, where the 2*X* multiple is coded in (4221).The second 2*X* operation performed by is *shift L*1 ,to encoded from (4221) to (5211).

Z_i	0	1	2	3	4
Z_i (4221s)	0000	0001	0010	0011	1000
Z_i (5211s)	0000	0001	0100	0101	0111
Z_i	5	6	7	8	9
Z_i (4221s)	1001	1010	1011	1110	1111
Z_i (5211s)	1000	1001	1100	1101	1111

Selected decimal codes for the recoded digits

Multiples 5*X:* Each BCD digit is first recoded to the (5421) decimal coding is shown. It is obtained by using *shift L*3 of the (4221) multiplicand reproducing the code in 5211. Then recoded from (5221) to (4221).

			$x10^2$	$x10^1$	$x10^0$
			4221	4221	4221
X (4221)	35		0000	0001	1001
		x5 ↓			
$L3_{shift}$			5211	5211	5211
$5X$ (5211)	175		0001	1100	1000
		↓			
Digit recoding			4221	4221	4221
$5X$ (4221)	175		0001	1011	1001

Calculation of 5X for decimal operands coded in (4221)

Multiples 3*X:* It is obtained by a carry-propagate addition of BCD multiples *X* and 2*X* in a d- digit BCD adder (implemented as a quaternary- tree decimal adder) by using equation 3.4 the BCD sum digit are recoded into (4221).

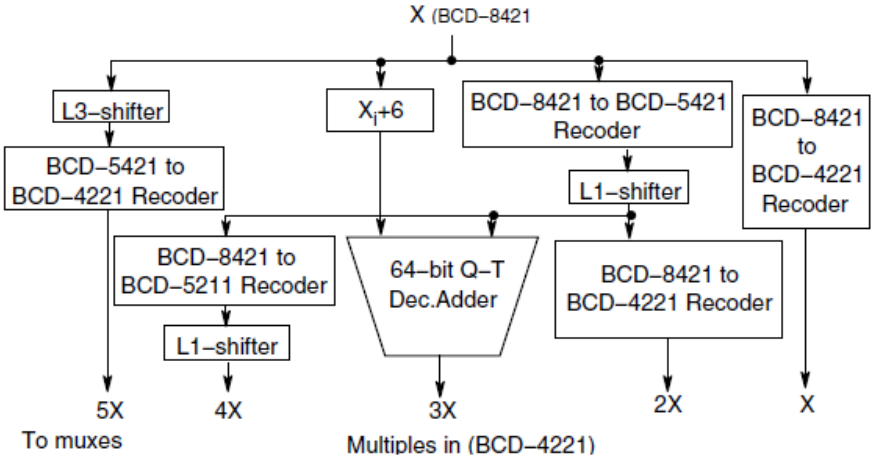


Figure 3 – Multiples for SD radix–10 encoding

IMPLEMENTATION OF DIGIT RECODERS

The outline of proficient digit recoders is a basic issue, due to their high effect on the area and performance of the entire multiplier. Digit recoders are utilized to process the decimal multiplicand multiples and in the decrease of partial products to figure x2n (n > 0) operations.

For logical Execution of digits recoders for BCD, BCD excess 6, and (5421) decimal codes are straight forward, since there is just a mapping of decimal digits to these codes (every decimal digit has a solitary 4-bit representation). On the other hand, because of the redundancy of (4221) and (5211) decimal codes, there are a few decisions for the digit recoding to (4221) or (5211). The sixteen 4-bit vectors of a coding can be mapped (recoded) into distinctive subsets of 4-bit vectors of the other decimal coding depicting to the same decimal digit. These subsets of the (4221) and (5211) codes are additionally decimal coding. Among all of the subsets considered, the non-redundant decimal codes (4221s) and (5211s) (subsets of ten 4-bit vectors), present fascinating properties. Specifically, these codes verify

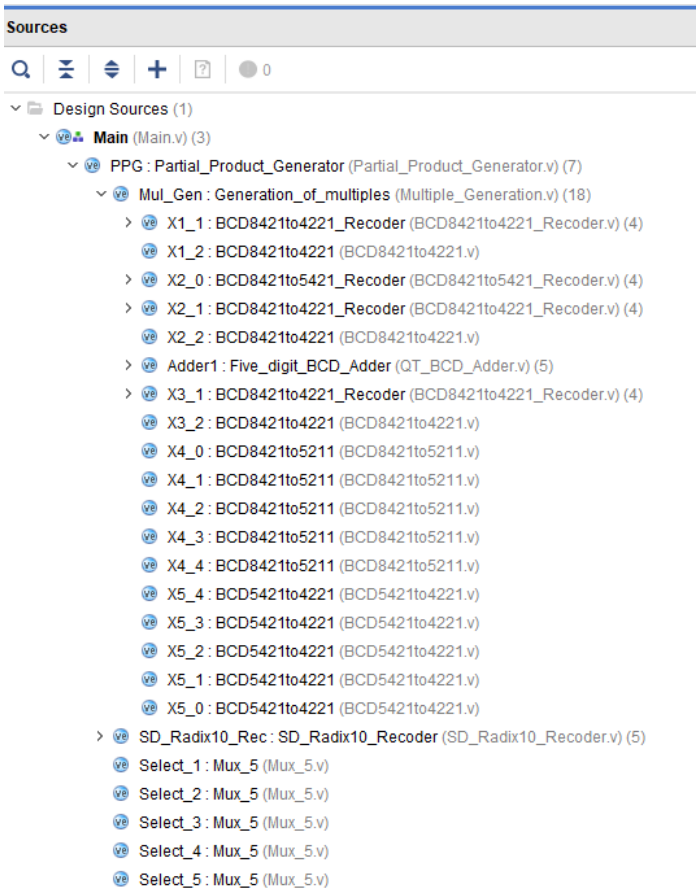
$$2Z(4221s) = L1_{shift}[Z(5211s)]$$

That is, subsequent to moving 1 bit to one side an operand Z depicted in (5211s), the resultant bit-vector represents to the decimal estimation of 2Z coded in (4221s). This rearranges the usage of x2n operations for n > 1. Specifically, for a decimal operand, Z_i(4221) , Z x 2n is implemented by a first level of Z_i (4221*s*) to Z_i (5211*s*) to digit recoders followed by n - 1 levels

of Z_i (4221*s*) to Z_i (5211*s*) digit recoders. The yield of every level of digit recoders is moved 1 bit to one side such that the most noteworthy bit of each (5211s) digit (weight 5) is moved out to the following decimal position (weight 10).

IMPLEMENTATION IN FPGA

Source Files:

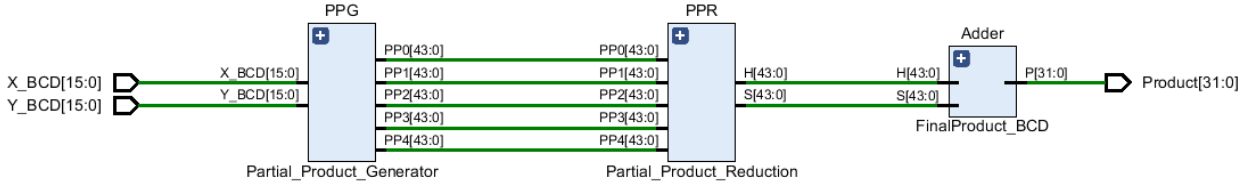


Specifications of the Software used:

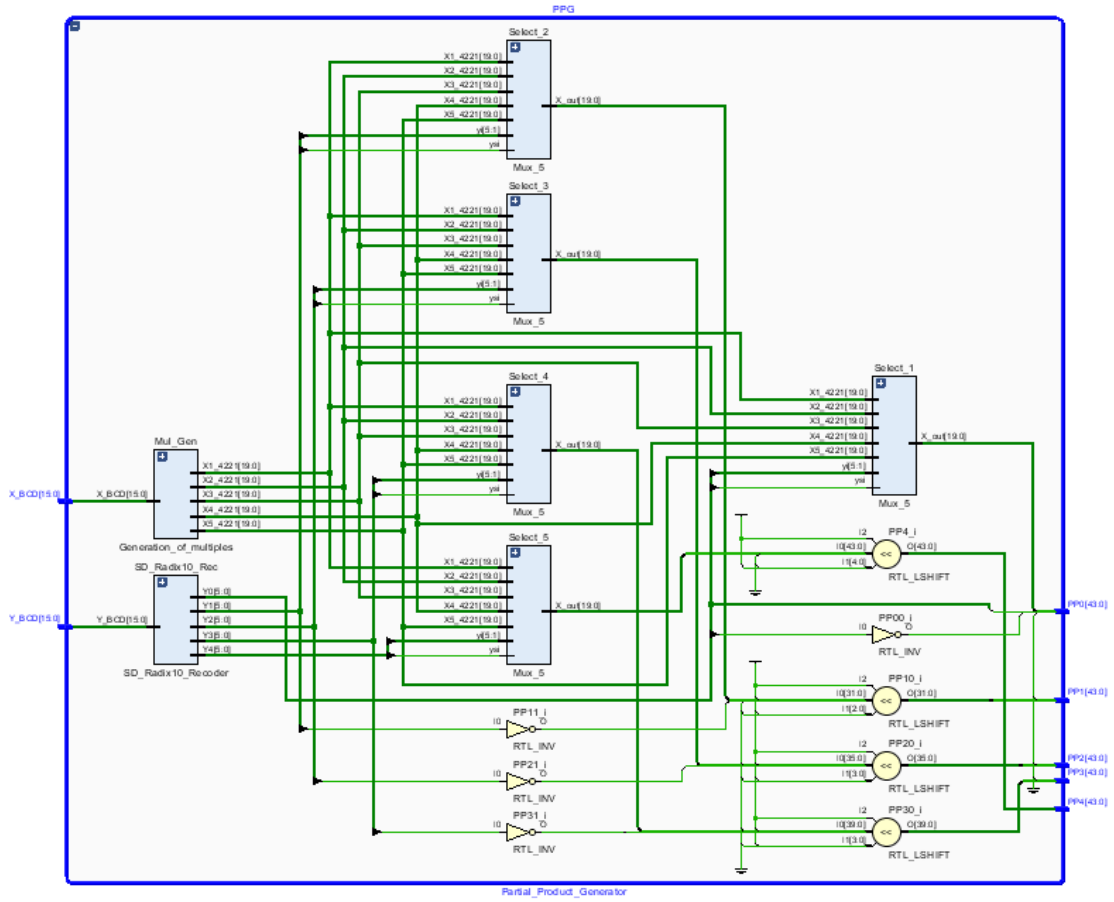
Software - Xilinx Vivado v2017.3

FPGA board
Family - Artix-7
Package - csg324
Part - XC7A100TCSG324-1

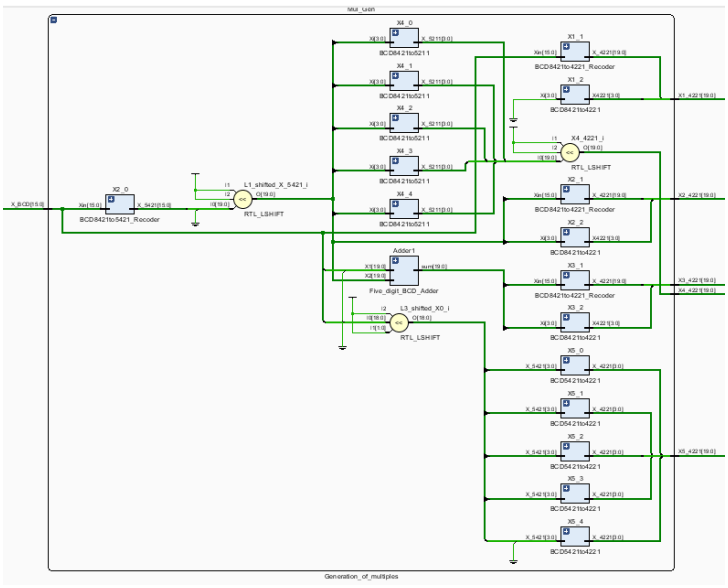
RTL SCHEMATIC



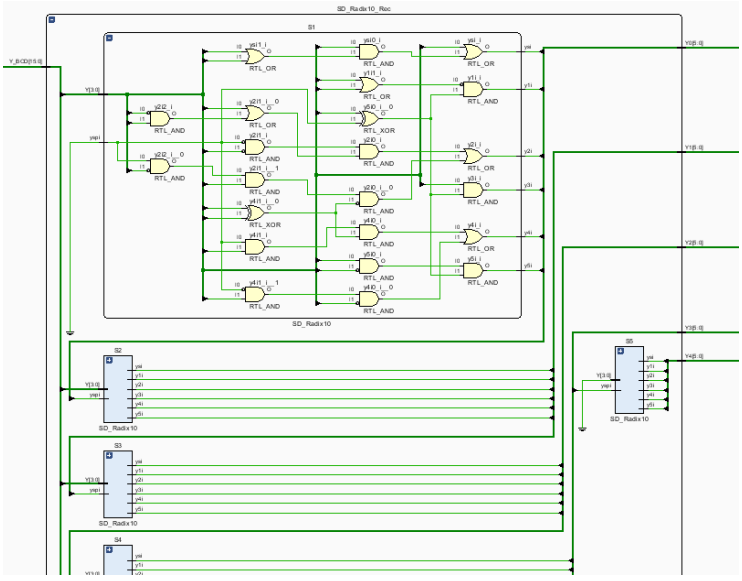
Overall System



Partial Product Generator Block

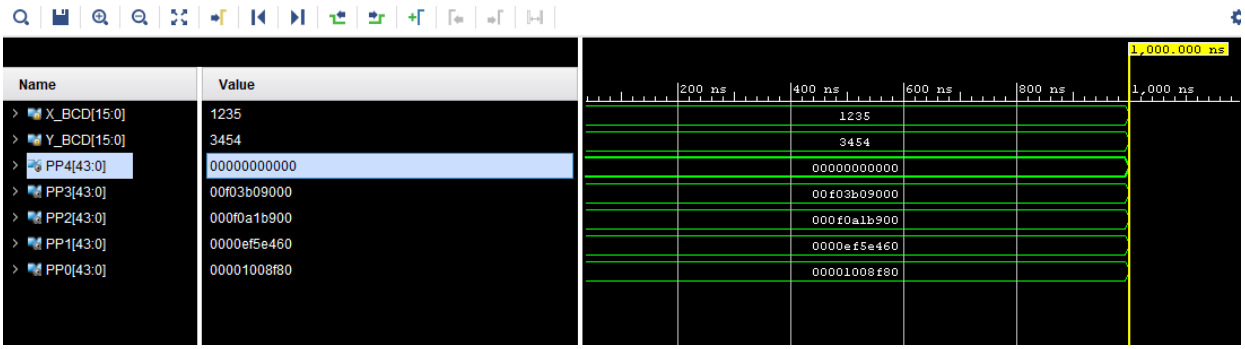


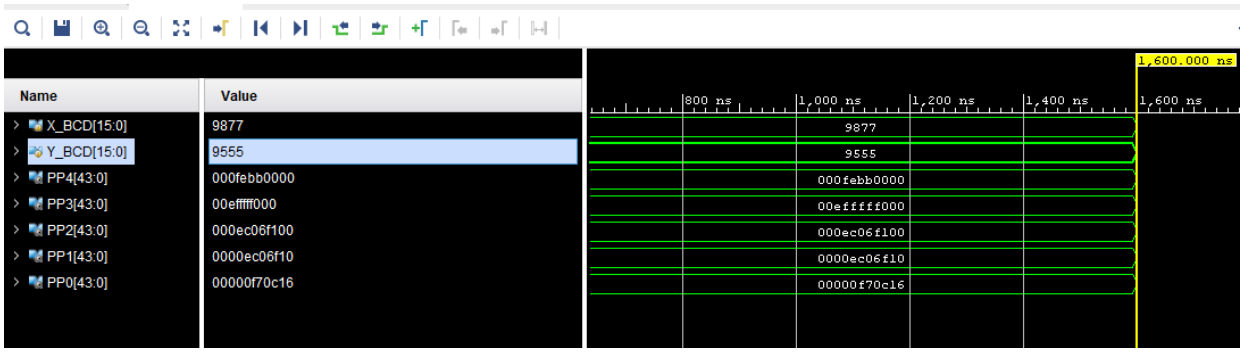
Multiple Generator



Signed Digit Radix 10 Recoder

SIMULATION RESULTS





Resource	Estimation	Available	Utilization %
LUT	484	63400	0.76
IO	64	210	30.48

RESULTS

The partial product generation of radix-10 parallel decimal has been successfully implemented. We can conclude that with negligible trade-off for the area and power we can achieve higher performance.

The present work on Radix-10 parallel decimal multiplier can be extending in various directions. Some of the suggestions as given below:

1. Different encoding method can be analysed to optimize the speed and area.
2. In place of carry save adder, other adders such as carry select adder and carry look ahead adder can be used to increase the performance.

3. Different compressors can be used for accumulation of partial products so that delay can be further reduced.
4. In order to enhance the performance higher order compressors like 17:2 can be used to accumulate the partial products.

REFERENCE

1. Robert D. Kenney and Michael J. Schulte, Senior Member, IEEE, "High-Speed Multi operand Decimal Adders", 2005
2. Mark A. Erle, Eric M. Schwarz, Michael J. Schulte, "Decimal Multiplication With Efficient Partial Product Generation", 17th IEEE Symposium on Computer Arithmetic.
3. Xiaoping Cui , Earl E. Swartzlander, "High Performance Parallel Decimal Multipliers using Hybrid BCD Codes", 2017
4. Vazquez, A., Antelo, E., and Montuschi, P: 'Improved design of high performance parallel decimal multipliers', IEEE Trans. Comput., 2010,59, (5), pp. 679–693
5. Alvaro V´azquez, Elisardo Antelo, Paolo Montuschi, "A New Family of High–Performance Parallel Decimal Multipliers", 2007