A New Parallel VLSI Architecture in Real Time by using Microcontroller

K.V. Vinetha[1], S. Thirumala Devi[2]

Abstract: ***In this paper, we proposed a new architecture of multiplier-and-accumulator (MAC) for high-speed arithmetic. By combining multiplication with accumulation and devising a hybrid type of carry save adder (CSA), the performance was improved. Since the accumulator that has the largest delay in MAC was merged into CSA, the overall performance was elevated. The proposed CSA tree uses 1's-complement-based radix-2 modified Booth's algorithm (MBA) and has the modified array for the sign extension in order to increase the bit density of the operands. The CSA propagates the carries to the least significant bits of the partial products and generates the least significant bits in advance to decrease the number of the input bits of the final adder. Also, the proposed MAC accumulates the intermediate results in the type of sum and carry bits instead of the output of the final adder, which made it possible to optimize the pipeline scheme to improve the performance. The proposed architecture was synthesized with 250, 180 and 130 m, and 90 nm standard CMOS library. Based on the theoretical and experimental estimation, we analyzed the results such as the amount of hardware resources, delay, and pipelining scheme. We used Sakurai's alpha power law for the delay modeling. The proposed MAC showed the superior properties to the standard design in many ways and performance twice as much as the previous research in the similar clock frequency. We expect that the proposed MAC can be adapted to various fields requiring high performance such as the signal processing areas.***

Keywords: multiplier-and-accumulator (MAC), carry save adder (CSA), Booth's algorithm (MBA)

1. Introduction

With the recent rapid advances in multimedia and communication systems, real-time signal processings like audio signal processing, video/image processing, or large-capacity data processing are increasingly being demanded. The multiplier and multiplier-and accumulator (MAC) [1] are the essential elements of the digital signal processing such as filtering, convolution, and inner products. Most digital signal processing methods use nonlinear functions such as discrete cosine transform (DCT) [2] or discrete wavelet transform (DWT) [3]. Because they are basically accomplished by repetitive application of multiplication and addition, the speed of the multiplication and addition arithmetic's determines the execution speed proceedings. and performance of the entire calculation. Because the multiplier requires the longest delay among the basic operational blocks in digital system, the critical path is determined by the multiplier, in general. For high-speed multiplication, the modified radix-4 Booth's algorithm (MBA) [4] is commonly used. However, this cannot completely solve the problem due to the long critical path for multiplication [5] In general, a multiplier uses Booth's algorithm [3] and array of full adders (FAs), or Wallace tree [5] instead of the array of FAs., i.e., this multiplier mainly consists of the three parts: Booth encoder, a tree to compress the partial products such as Wallace tree, and final adder [2], [4]. Because Wallace tree is to add the partial products from encoder as parallel as possible, its operation time is proportional to where is the number of inputs. It uses the fact that counting the number of 1'samong the inputs reduces the number of outputs into. In real implementation, many (3:2) or (7:3) counters are used to reduce the number of outputs in each pipeline step. The mos effective way to increase the speed of a multiplier is to reduce the number of the partial products because multiplication proceeds a series of additions for the partial products. To reduce the number of calculation steps for the partial products, MBA algorithm has been applied mostly where Wallace tree has taken the role of increasing the speed to add the partial products. To increase the speed of the MBA algorithm, many parallel multiplication architectures have been researched [11]– [13]. Among them, the architectures based on the Baugh–Wooley algorithm (BWA) have been developed.

VlSI Architecture

In 19791 anticipated scaling of VLSI technology favored the development of regular machines that exploited concurrency and locality and that were programmable. As shown in columns 2 and 3 of Table 1, twenty years was expected to bring more than a thousand fold increase in the number of grids2 , and hence the number of devices that could be economically fabricated on a chip. Clearly concurrency (parallelism) would need to be exploited to convert this increase in device count to performance. Locality was required because the wire bandwidth at the periphery of a module was scaling only as the square root of the device count, much slower than the 2/3 power required by Rent's rule [LanRus71]. Also, even in 1979 it was apparent that wires, not gates, limited the area, performance, and power of many modules. The issue of design complexity motivated regularity and programmability. Designing an array of identical, simple processing nodes is an easier task than designing a complex multi-million transistor processor. A programmable design was called for so that the mounting design costs could beamortized over large numbers of application. In the twenty years since the first conference many of the hard problems of parallel machine design have been solved. We now understand how to design fast, efficient networks to connect arrays of processors together [Dally92, DYN97]3 . Mechanisms that allow processors to quickly communicate and synchronize over these networks have been developed [LDK+98]. We understand how to implement efficient, coherent shared memory systems [ASHH88]. Several meth-ods of programming parallel machines have been demonstrated. Research machines were constructed to demonstrate the technology, provide a platform for parallel software research, and solve the engineering problems associated with its realization

[Seitz85, NWD93, SBSS93]. The results of this research resulted in numerous commercial machines [Scott96] that form the core of the high-end computer industry today4. Just as important, we learned what didn't work: MIMD machines are preferable to SIMD machines even for data-parallel applications. Similarly, general-purpose MIMD machines are preferable to systolic arrays, even for regular computations with local communication. Bit-serial processors loose more in efficiency than they gain in density5. A good general-purpose network (like a 3-D torus) usually outperforms a network with a topology matched to the problem of interest (like a tree for divide and conquer problems). It is better to provide a general-purpose set of mechanisms than to specialize a machine for a single model of computation. While successful at the high end, parallel VLSI architectures have had little impact on the mainstream computer industry. Most desktop machines are uniprocessors and even departmental servers contain at most a few 10s of processors. Today's mainstream microprocessor chips are dense enough to hold 1000 of the 8086s or 68000s of 1979, yet we use all of this area to implement a single processor. What went wrong? Why isn't the average PC a fine-grain parallel machine with 10s of processors on integrated processor-DRAM chips in the spirit of Mosaic or the J-Machine? By many objective measures this would clearly be a more efficient architecture. There are three main reasons for this course of events:

1) There was considerable opportunity to apply additional grids to improve the performance of sequential processors.
2) Software compatibility favored sequential machines.
3) High-overhead mechanisms used in early parallel machines motivated a coarse granularity of both hardware and software.

In 1979 there was more than a factor of 100 difference in performance between the best microprocessors (0.5MIPS, 0.001MFLOPS) and a high-end CPU such as used in the Cray 1 (70MIPS, 250MFLOPS [Russel78]) or IBM 370. Only a small part of this difference, about a factor of 3, was due to the difference in gate delay between bipolar and MOS technology. Most of the difference was due to increased gate count that was used to aggressively pipeline execution and to exploit parallelism. Between 1979 and 1999 microprocessors closed this gap by incorporating most of the advanced features pioneered in mainframes and supercomputers in the 1960s and 70s as well as a few new tricks. On-chip caches, on-chip memory management units, pipelined multipliers and floating-point units, multiple instruction issue, and even out-of-order instruction issue were added to processors during this period. The addition of these features, along with quadrupling the word width from 16-bits to 64-bits created a sufficient appetite for grids without resorting to explicit parallelism. During the past 20 years, the performance of a high-end microprocessor increased from 0.5MIPS to 500MIPS, about a factor of 1000. From the data in Table 1, we see that clock frequency increased by a factor of 80: a factor of 20 is due to gate delay and a factor of 4 is due to reducing the number of gates per clock. The remaining factor of 12.5 reflects a reduction in clocks per instruction (CPI) from about 10 for the unpipelined microprocessors of 1979 to just under 1 for today's 3- and 4-way multiple-issue superscalar processors. Software evolved considerably during the last 20 years: from text-based applications running on proprietary operating systems (like VMS and MVS) to graphics-based applications running on third-party operating systems (like Windows and Unix). What remained constant, however, was the sequential nature of this software. Manufacturers wanting to sell machines that would run existing software needed to build fast sequential machines. Commercial parallel machines shut themselves out of the mainstream by taking a path that emphasized capability (running very large problems) rather than economy (solving the most problems per dollar ´ second). These machines were coarse-grained both in the amount of memory per node and in the size of individually scheduled tasks. Early machines were forced by high-overhead mechanisms to run programs with large tasks sizes. To ensure software compatibility, later machines were forced to follow this same route, often because of macro packages (like PVM and MPI) that hid the improved mechanisms behind high-overhead software. A coarse-grain parallel computer node is largely indistinguishable from a conventional workstation or PC with one exception: it is considerably more expensive. While one can equalize the expense by constructing coarse-grain parallel computers from networks of workstations [ACP95], achieving an economy that is better than serial machines requires fine-grain nodes [FKD+95]. In summary, for most of the 80s and 90s software compatibility motivated building sequential machines; there was little economic advantage to coarse-grain parallel machines; and there were many obvious ways to use more grids to make a sequential CPU faster. Given this environment, it is no surprise that industry responded by making sequential CPUs faster and only building coarse-grain parallel machines. The next 20 years The next 20 years promise to be exciting ones in the area of VLSI architecture with a major revolution in the architecture of mainstream processors. Continued scaling of technology (columns 3 and 4 of Table 16) will give us yet another thousandfold increase in chip density. As in 1979 it is natural to think of developing architectures that are programmable and exploit concurrency and locality to exploit this increased density. Unlike 1979, however, there are three reasons why a revolution is likely now: First, sequential processors are out of steam. While clever architects will undoubtedly continue to develop new methods to squeak a few percentage points more performance from sequential processors, we are clearly well past the point of diminishing returns7 . Large amounts of chip area are spent on complex instruction issue logic and branch prediction hardware while yielding small improvements in performance. To continue improving performance geometrically each year, there is no alternative except to exploit explicit parallelism.

2. MIMO System Model

In MIMO communication systems, more than one antenna is used at the transmitter to transmit symbols and more than one antenna is used at the receiver to receive them. In the diagram of Figure 1, spatial multiplexing is used and M transmit antennas transmit M symbols simultaneously while each symbol is received by the N receive antennas. Each symbol transmitted is received by all the receiving antennas thus making multiple channel paths. These paths, if combined, make a matrix of channel elements. Each symbol makes N channel paths and is received by N receive antennas. Since there are M symbols transmitted simultaneously, the channel becomes a NxM matrix.

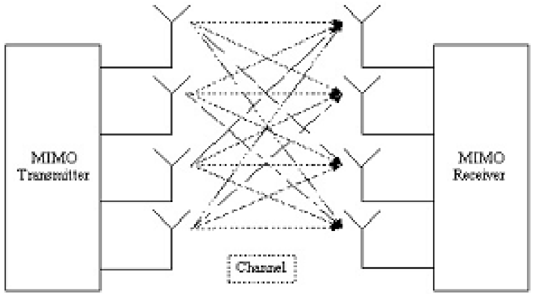


Figure 1: MIMO System Model

If s = (s1,s2,…….sM)T denotes the symbol vector transmitted and a vector r = $(r1,r2,\ldots\ldots.rM)T$for received signal, H denotes the NxM channel matrix between the receive and transmit antenna array, and v denotes the AWGN independent and identically distributed noise vector, then the corresponding receive vector r at the input of the MIMO receiver is given by: r = Hs + v (2)

2.1 Square Root Algorithm for V-BLAST

In VBLAST, successive nulling and cancellation is used to detect the transmitted symbols. The channel matrix is first inverted and then reordered to detect that symbol first which has the highest post detection Signal to Noise ratio (SNR). This corresponds to the row of the inverted channel matrix having minimum Euclidean distance. The symbol after detection is subtracted from the received symbol vector. The corresponding column of the H matrix is zeroed down and the process is repeated with the deflated channel matrix until all the symbols are detected. In this research, MMSE is used for channel inversion. The pseudo inverse of a generic matrix H is given by $H+=(H*H)-1H*=R-1Q*$ (3) The pseudo inverse can be computed using either singular value decomposition (SVD) or QR decomposition. The square root algorithm [3] is developed for MMSE-VBLAST and computes the QR decomposition of the augmented channel matrix. $HNxM\ \alpha\ IMxM = QR = QaNxMx\ RMxM$ (4) Here x denotes the entries that are not relevant. The algorithm first decomposes the channel matrix into QR $ar+ja1$ and then computes $P1/2=R-1$. Once Qa and $P1/2$ are computed, the repeated pseudo inverse can be avoided. The algorithm is described below:

1) Compute Qa and $P1/2$ using equation (5):
$B\varepsilon i = X$ (5) $B = 1HiP|\ i-11/20Mx1P|\ i-11/2-eiNx1Bi-1\ X = x01xMxP|\ i-11/2xBi$ Here i represent iterations and i = 1… N. B is the prearray matrix and has dimension of *(1+M+N) x (1+M)* and $P|01/2=1\ \alpha I,B0=\ 0NxMeiNx1$ is the i-th unit vector of dimension N and θi is any unitary transformation (Jacobi rotation) that block lower triangularizes the pre-array denoted by M. After N iterations, $P01/2=\ P|N1/2and\ Qa=\ BN$ (6) Equations (5) and (6) are used in pseudo inverse computation. For the rest of the algorithm, the reader is referred to [3].

2.2 CORDIC

In hardware, an efficient way of accomplishing a Givens rotation is using a CORDIC. CORDIC implements the rotation equations: $x'=\cos\theta\ x-y\tan\theta\ y'=\cos\theta\ y+x\tan\theta$ (7) When angles are selected such that: $\tan\theta=2-i$ (8) In this case, multiplication by simply becomes a right shift. When several of these CORDIC processing elements are used together, one can rotate by an arbitrary angle by rotating by a combination of allowed angles: $\theta=tan-12-i$ (9) For a rotation using a fixed number of iterations the terms turn out to be a constant. The constant scaling value can be seen in [6] for up to 15 iterations. For our design we need the CORDIC to first rotate a vector to the nulling axis and then remember the angle rotated to following vectors can be rotated to the same angle. These two modes of operation are known as vectoring and rotation, respectively. The design of our CORDIC implemented the rotation equations (7) using the constraint on angles in (9) such that our final result nullsy'. We also needed to design a CORDIC that operates in vectoring and rotation mode. In order to implement the equations, we used shifters and adders to do the bulk of the work along with simple decision logic. Each processing element receives two input vectors and finds their sign. It must now decide based on their signs whether to rotate up or down

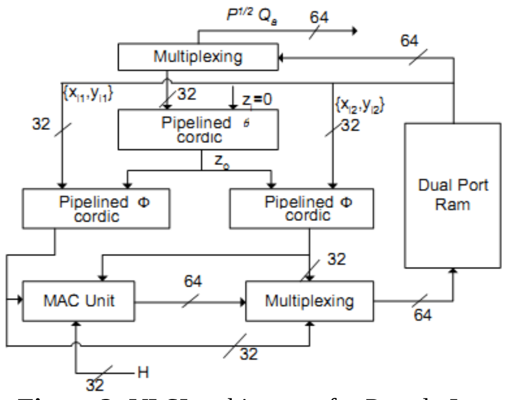


Figure 2: VLSI architecture for Pseudo Inverse

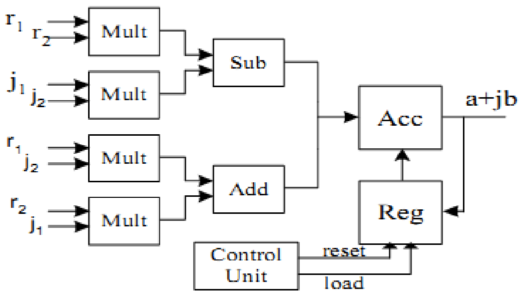


Figure 3: Conventional MAC Module The main design

platform used during this project was SIMULINK. It is a tool-flow that enables the use and creation of high level block diagrams which can be used for simulation, emulation, and hardware description. The blocks used in this design were from the Xilinx block set. Based on Xilinx block set the architecture of Pseudo Inverse is designed and can be seen as follows Xilinx block set. Based on Xilinx block set the architecture of Pseudo Inverse is

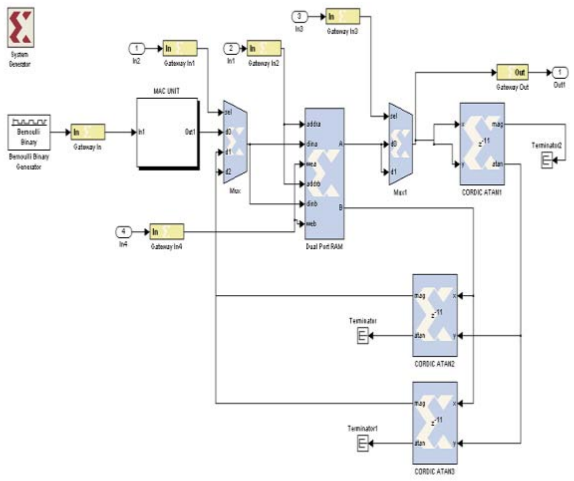
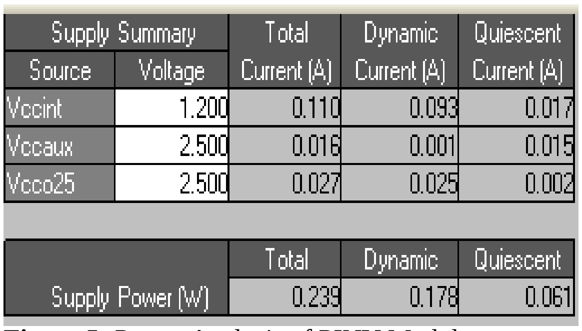


Figure 4: VLSI architecture for Pseudo Inverse designed by Xilinx Blocksets

3. Results

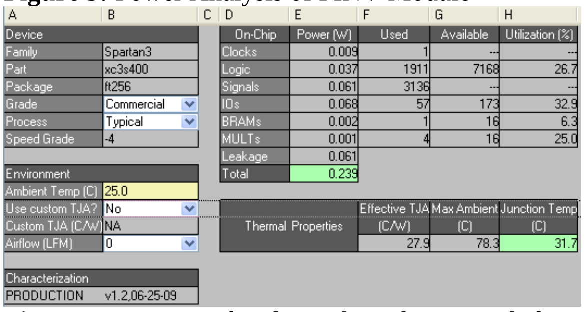
Every block was tested extensively in simulation. Testing was done by performing the algorithm for that block in MATLAB, Xilinx System Generator and Xilinx ISE to obtain the expected values given certain data. The blocks were then given the same inputs as the algorithm was given in MATLAB, simulation was run, and the outputs of the blocks were reviewed. All of the blocks performed as desired, given several known test inputs of a wide range. The total power consumed by Pseudo Inverse module is obtained as 239mW.



Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.110	0.093	0.017
Vccaux	2.500	0.016	0.001	0.015
Vcco25	2.500	0.027	0.025	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.239	0.178	0.061

Figure 5: Power Analysis of PINV Module

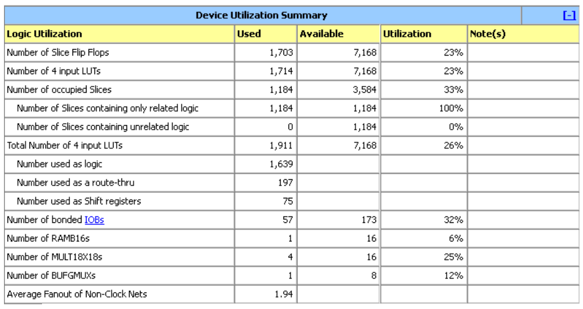


A	B
Device	
Family	Spartan3
Part	xc3s400
Package	ft256
Grade	Commercial
Process	Typical
Speed Grade	-4
Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0
Characterization	
PRODUCTION	v1.2,06-25-09

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.009	1	---	---
Logic	0.037	1911	7168	26.7
Signals	0.061	3136	---	---
IOs	0.068	57	173	32.9
BRAMs	0.002	1	16	6.3
MULTs	0.001	4	16	25.0
Leakage	0.061			
Total	0.239			

	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
Thermal Properties	27.9	78.3	31.7

Figure 6: Device family and package used for Pseudo inverse Module It is clear from Fig. 7 (b) that in Pseudo Inverse Module, CORDIC has used two third of the chip area. However area use by Flip-Flops, Look Up table etc. can be viewed by the figure provided below in



Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	1,703	7,168	23%	
Number of 4 input LUTs	1,714	7,168	23%	
Number of occupied Slices	1,184	3,584	33%	
Number of Slices containing only related logic	1,184	1,184	100%	
Number of Slices containing unrelated logic	0	1,184	0%	
Total Number of 4 input LUTs	1,911	7,168	26%	
Number used as logic	1,639			
Number used as a route-thru	197			
Number used as Shift registers	75			
Number of bonded IOBs	57	173	32%	
Number of RAMB16s	1	16	6%	
Number of MULT18X18s	4	16	25%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	1.94			

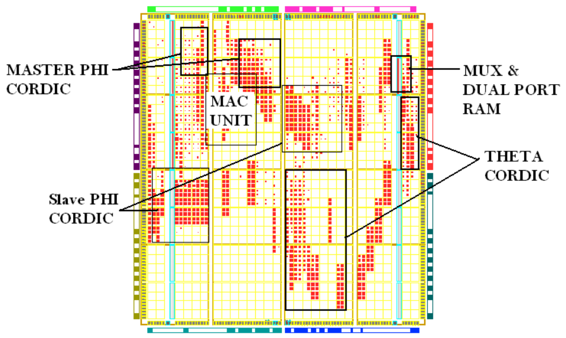


Figure 7: (a) Area utilization summary for Pseudo inverse Module (b) layout of Pseudo Inverse module

4. Conclusion

Instead of QR triangular array that employs large number processors, single processor based VLSI architecture is proposed for V-BLAST detection. The quantization scheme of the square root algorithm for V-BLAST detection is presented considering the tradeoff between the hardware complexity and the performance. The proposed architecture is implemented in SIMULINK used by special sets of XILINX block sets. While the full Square Root algorithm was not designed, the major computationally complex parts were. Finding $p1/2$ enables one to easily perform SIC and subsequently decode information streams in V-BLAST architecture. The future work will be addressed to design and implement other module of square root algorithm like SORT and NULL for power analysis and area utilization.

References

Proceedings Papers

[1] G. J. Foshini: Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas. Bell Labs technical Journal, pages 41-57, Autumn 1996
[2] R.Andraka: A Survey of CORDIC algorithm for FPGAs, FPGA'98. Proceeding of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, Feb. 22-24, 1998, Monterey, CA. pp191-2000
[3] Babak Hassibi: An Efficient Square-Root Algorithm for BLAST", http://mars.bell-labs.com/
[4] Z.Guo and P. Nilson: A VLSI implementation of MIMO detection for future wireless communications, in Proc. IEEE PIMRC'03, vol. 3, 2003, pp. 2852-2856
[5] M.Pedram,"Power Minimisation in IC Design: Principles and Applications", ACM Transactions on Design Automation of Electronic Systems, vol. 1, no. 1, pp. 3-56, January 1996.
[6] X. Wu, M. Pedram, L. Wang, "Multi code state assignment for low power design", Circuits, Devices and

Systems, IEEE Proceedings vol. 147, Issue 5, Oct. 2000 Page(s):271 - 275

[7] Area & Power Efficient VLSI Architecture for Computing Pseudo Inverse of Channel Matrix in a MIMO Wireless System, Zahid Khan, Tughrul Arslan, John S. Thompson, Ahmet T. Erdogan, Proceedings of the 19th International Conference on VLSI Design

Author Profile

Ms. K.V. Vinetha was born on 14th March 1989 in Andhrapradesh, India. She completed her Btech in ECE from khader memorial college of Engineering and technology, nalgonda district (JNTUH) in the year 2010. She received her MTech in Embedded systems from Gudlavalleru Engineering College, gudlavalleru (JNTUK) in the year 2013. She is working as an Asst. professor in the department of ECE in KKR & KSR Institute of Technology and Science affiliated to JNTUK. She is having 3.5 years of experience.

Mrs. S. Thirumala Devi was born on 30 Aug 1987. She received her B.Tech degree in Electronics and Communication Engineering from Sri vishnu Engineering College for women bhimavaram (JNTUH) in 2008, and M.Tech in VLSI design from Sri vishnu Engineering College for women (JNTUK) in 2013. She is working as an Asst professor in the department of ECE in KKR&KSR Institute of Technology and Sciences .she is having 6 years of experience