Efficient Implementation of IQ_VOQ Based Input Blocks in the Design of Routing Node for Uniform & Bursty Traffic

Bhavana S. Pote[1], Rehan Maroofi[1] and Vilas N. Nitnaware[2]

[1] *Department of Electronics,*
Ramdeobaba College of Engineering and Management, Nagpur, India

[2] *Department of Electronics Design Technology,*
Ramdeobaba College of Engineering and Management Nagpur, India

bhavanaspote@gmail.com, rehanmaroofi@hotmail.com, nitnawarevn@rknec.edu

Abstract

The rise in complexity of SOC (System-on-Chip) architectures has brought into focus the need for better communication models for SOCs. The traditional bus based approach is reaching its limit with the emergence of high core count SOCs. The theory and practices of wired communication networks is being applied to tackle communication issues in complex SOCs. This is referred to as the Network on Chip (NOC) model. Routers are one of the most important elements of an on chip network. The underlying architecture of a router is based on a crossbar switch as it offers higher throughput and lower latency due to point-to-point architecture. Homogenous sizes of the input module in the router may not be efficient as some cores may be underutilized while some of them may be overloaded. A heterogeneous size of the input module is preferred for the predicted traffic but bursty in nature. We proposed, input module of size 64_packet array for the most busy node in the design while the most underutilized input module can also satisfy the need of the packet array of size 16. For moderate traffic, the input module with 32 packet array can be an efficient solution. The Islip based IQ_VOQ is the most practical combination, popular in the CISCO router [12000 series]. The input module proposed here is based on Virtual output queuing while Islip scheduling algorithm is based on unfolding and folding concept. First, the RTL implementation of input module for 3-proposed design has been carried out and later it realized using a standard-cell-based ASIC flow using 90 nm saed-typ technology library of Synopsis Educational Design Kit.

Keywords: *VOQ, CIOQ, CICBW, FIFO, HOL Blocking, ISLIP*

1. Introduction to IQ_VOQ [input queuing_virtual output queuing]

Traditionally, the difference in switches_ design was in the way the queuing function is implemented. There has been extensive research work in this area and many switching architectures have been proposed, such as output-queued (OQ) and input queued (IQ) switches. It is well known that OQ switches are highly desirable for their optimal performance and QOS guarantees [1], [2], [3], [4].

However, the high internal speedup coupled with the limitation in memory access time prohibits the OQ to scale to even a medium sized switch [1]. On the other hand, IQ switches have gained more interest because of their low cost and high scalability. The only drawback with the IQ switches are the head-of-line (HOL) blocking problem which limits the

achievable throughput to approximately 58.6% [1]. By using a simple buffering strategy, HOL blocking can be entirely eliminated [1], [2], [3]. Specifically, if each input maintains a separate queue for each output, HOL blocking is eliminated because a cell cannot be held up by a cell queued ahead of it that is destined for a different output. This scheme is called Virtual Output Queuing. Figure 1 is an example of a 3 × 3 VOQ switch. With the VOQs, a cell at the head of a queue can no more block any cell destined to another output.

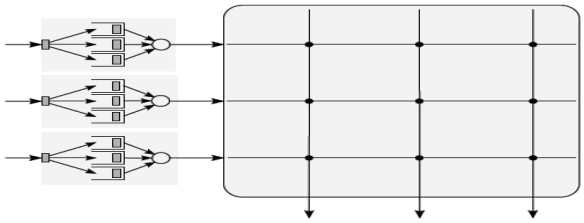

Figure 1. Virtual Output Queuing

Apart from the above discussed queuing techniques, Combined input/output queuing [CIOQ] and Combined Input Crosspoint Buffered Switch [CICBW] techniques are also available. The following table compares the advantages and the disadvantages of all such techniques.

1.1 Summary of the Queuing Techniques

Table 1. Summary of the Queuing Techniques

Architecture	Advantages	Disadvantages
OQ	100 % Throughput for any traffic, QoS guarantees.	Speedup of N required, impractical
IQ-VOQ	Simplicity, No HOL Blocking. 100% throughput for uniform traffic with ISLIP algorithm. More area efficient.	No performance Guarantees for bursty traffic. [Performance could be improved with efficient buffering].
CIOQ	Emulation of OQ behavior	Speedup, complex scheduling algorithms
CICQ	Decoupling of input and output, better performance than unbuffered crossbars.	Compromise between Cross point buffer size and performance
CIXOB	100 % throughput for any traffic, QoS guarantees, simple schedulers.	Switch speedup

A router requires a scheduling algorithm to decide which one of the queues at each input can forward its HOL cell to the destination output.

1.3 Overview of iSLIP Scheduling Algorithm

Finding a scheduling algorithm that is simple, fair and efficient is critical in designing a high speed input queued switch. Studies have shown that a suitable scheduling algorithm can increase the throughput of an input-queued switch to 100% when arrivals are independent.

Researchers proposed many practical algorithms such as MUCS [1], iSLIP [1], and RPA [1]. Here, we will look into the iSLIP algorithm proposed by McKeown for input-queued switches [11] above. In an iterative algorithm called iSLIP [11] can achieve 100% throughput

under uniform traffic. When the input traffic is non-uniform, SLIP still has a good performance if the input load is not too much. Moreover, it's fast, fair and easy to implement in many switch fabrics [11]. If the system demands more area efficient scheduler, ISLIP based folded scheduler [1], [2] can also be used to save silicon area by around 31-33% over the existing ISLIP scheduler by Nick McKeon [11].

2. Function of IQ_VOQ based input_module

The input module proposed here consists of packet array, FIFO, Destination Head array, Destination Tail array and the linked list array, Figure 3. The input module receives the packets in two phases from the input IP block, 36 bits in each phase [1]. It stores into packet array at the address given by FIFO. Destination Head maintains the record of the oldest pending packet for that destination while the record of all the newest pending packets are maintained jointly by linked list array and the Destination Tail array. The linked list array provides the next packet for that particular destination [15]. All the arrays in the design are based on DFFs, see Figure 2.

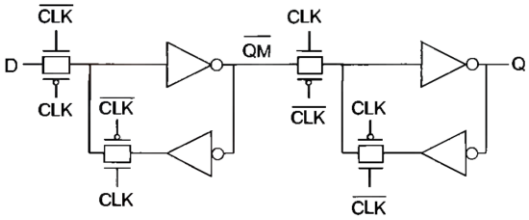

Figure 2. DFF Schematic

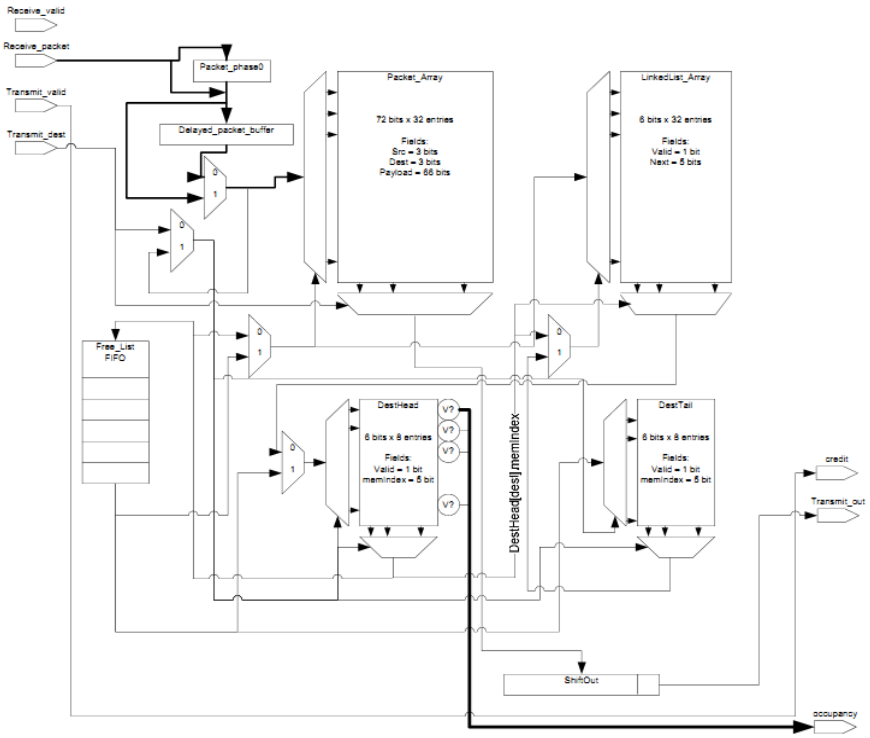

Figure 3. Input Module

The design considers total 72 bits size of the packet, see Table 2, which further consists of two fields control field and payload. Control field consists of 6 bits, each 3 bits is used to describe source and destination address while Payload consists of 66 bits.

Table 2. Packet Size

Bits	71 69	68 66	65 0
Field	Src	Dest	Packet Data

3. Input Block to Handle Uniform and Discrete Traffic

Prioritized SLIP has 100% throughput under uniform traffic, but real network traffic is highly correlated from cell to cell and so in practice, cells tend to arrive in bursts. ***So a good scheduling algorithm should have an acceptable performance under bursty traffic too, but when the input traffic is burst, performance of Prioritized SLIP can decrease.***

In order to handle the non uniform traffic, two more sizes of the input modules are designed. Homogenous sizes of the input module may not be efficient as some cores may be underutilized while some of them may be overloaded. In the section described below, it facilitates the appropriate size of the input module depending upon the traffic imposed by an input IP core. Heterogeneous sizes of the input module is preferred for the predicted traffic but bursty in nature. The input core is feeding more data packets at the input module, logically to handle such traffic and to accommodate more no. of data packet, the size of the packet array has to big. While the input module which occasionally receives the data packet from the input IP core, such IP core do not need to have the big size of input module. We proposed, input module of size 64_packet array for the most busy node in the design while the most underutilized input module can also satisfy the need of the packet array of size 16. For moderate traffic, the input module with 32 packet array can be an efficient solution.

The following section exhibits the size of the input module with packet array 16, 32, 64. With the change in the size of the packet array, other arrays like FIFO, linked list, Destination Head and the Destination Tail also modified, see following tables.

Table 3. 16_Entry Packet Array

Components of Input Block	Array to store packet	FIFO to store free entries in packet array	Array for oldest pending packets	Array for newest pending packets	Array for Linked List	Shift Register
Size:	16 M. Index	16 M Index	8 M. Index	8 M. Index	16 M Index	9 bits per cycle, 8 cycles [Time Efficient Scheduler]
Data width:	72 bits	4 bits	5 bits	5 bits	5 bits	
Address width:	4 bits	4 bits	3 bits	3 bits	4 bits	

Table 4. 32_Entry Packet Array

Components of Input Block	Array to store packet	FIFO to store free entries in packet array	Array for oldest pending packets	Array for newest pending packets	Array for Linked List	Shift Register
Size:	32 M. Index	32 M Index	8 M. Index	8 M. Index	32 M Index	9 bits per cycle, 8 cycles [Time Efficient Scheduler]
Data width:	72 bits	5 bits	6 bits	6 bits	6 bits	
Address width:	5 bits	5 bits	3 bits	3 bits	5 bits	

Table 5. 64_Entry Packet Array

Components of Input Block	**Array to store packet**	**FIFO to store free entries in packet array**	**Array for oldest pending packets**	**Array for newest pending packets**	**Array for Linked List**	**Shift Register**
Size:	64 M. Index	64 M Index	8 M. Index	8 M. Index	64 M Index	9 bits per cycle, 8 cycles [Time Efficient Scheduler]
Data width:	72 bits	6 bits	7 bits	7 bits	7 bits	
Address width:	6 bits	6 bits	3 bits	3 bits	6 bits	

4. Functional Simulation of the Input Module_32_Packet Array

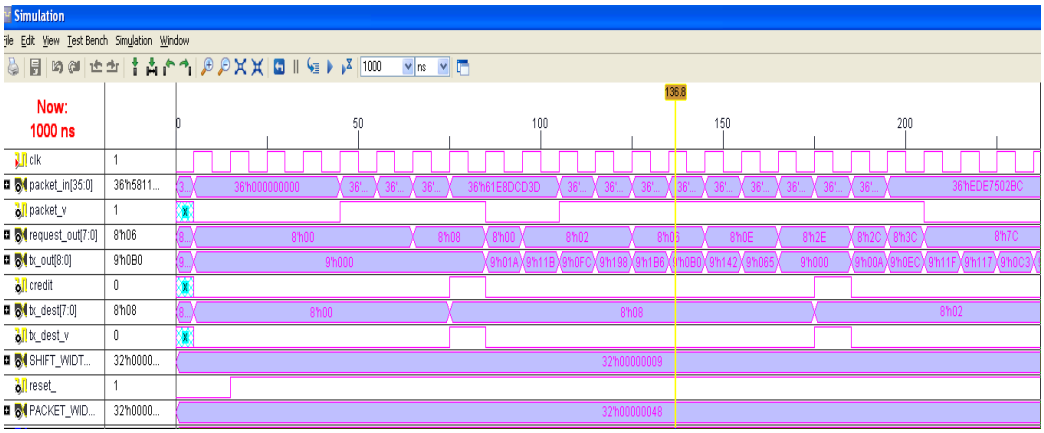

Figure 4. Functional Simulation of the Input Module_32_Packet Array

5. ASIC [FRONT END] Implementation of Input Module

This section intends to implement the RTL Synthesis Flow using Design Compiler of SYNOPSYS [1] to obtain the result of area, timing and power of the input block.

To optimize and analyze the design, two different kinds of constraints are applied. First set of constraints cover all the basic parameters including Environmental attributes, see table 5.1, while second set covers only 3-parameters, see **section 7.** In the first phase of analysis, the design has been synthesized using simple compile command, while in the second phase of analysis it was partitioned and then compiled. For best synthesis results, it is recommended to apply maximum constraints to the design. However, more precise results are obtained by applying partition and then compile method [16].

The tcl commands are written in the script file to set the constraints for the design of the input module, see Table 6.

5.1 Constraints in the Input Block With Packet Array Size 16, 32, 64.

Table 6. Constraints in the Input Block With Packet Array Size 16, 32, 64

16 Packet_Array_script	32 Packet_Array_script	64 Packet_Array_script
# Create user defined variables set CLK_PORT [get_ports clk] set CLK_PERIOD 8.00 set CLK_SKEW 0.01 set INPUT_DELAY 3.2 set OUTPUT_DELAY 3.2 set MAX_AREA 100000 #INPUT_TRANSITION D-max 0.01 set_input_transition 0.01 [remove_from_collection [all_inputs] $CLK_PORT]#set_driving_cell -lib_cell FD1 -pin Qn [get_ports req] #OUTPUT LOAD C-MAX 30fF set_load [expr 30.0/1000] [all_outputs]#set_load [load_of my_lib/AN2/A] [get_ports gnt]	# Create user defined variables set CLK_PORT [get_ports clk] set CLK_PERIOD 10.00 set CLK_SKEW 0.01 set INPUT_DELAY 4 set OUTPUT_DELAY 4 set MAX_AREA 140000 #INPUT_TRANSITION D-max 0.01 set_input_transition 0.01 [remove_from_collection [all_inputs] $CLK_PORT]#set_driving_cell -lib_cell FD1 -pin Qn [get_ports req] #OUTPUT LOAD C-MAX 30fF set_load [expr 30.0/1000] [all_outputs]#set_load [load_of my_lib/AN2/A] [get_ports gnt]	# Create user defined variables set CLK_PORT [get_ports clk] set CLK_PERIOD 16.00 set CLK_SKEW 0.01 set INPUT_DELAY 4.8 set OUTPUT_DELAY 4.8 set MAX_AREA 200000 #INPUT_TRANSITION D-max 0.01 set_input_transition 0.01 [remove_from_collection [all_inputs] $CLK_PORT]#set_driving_cell -lib_cell FD1 -pin Qn [get_ports req] #OUTPUT LOAD C-MAX 30fF set_load [expr 30.0/1000] [all_outputs]#set_load [load_of my_lib/AN2/A] [get_ports gnt]

5.2 The Following Tables Exhibit the Results Obtained After Synthesis

Table 7. Input_Module[16_32_64]_Cell Area

16 Packet_Array	32 Packet_Array	64 Packet_Array
(Number of ports: 58 Number of nets: 743 Number of cells: 384 Number of references: 30 Combinational area: 36423.175781 Noncombinational area: 52813.875000 Net Interconnect area: 7392.208008 **Total cell area:** **89237.328125** **Total area:** **96629.539062**	Number of ports: 58 Number of nets: 745 Number of cells: 380 Number of references: 30 Combinational area: 67469.007812 Noncombinational area: 44666.210938Net Interconnect area: 17875.134766 **Total cell area:** **112136.859375** **Total area:** **130011.992188**	Number of ports: 58 Number of nets: 742 Number of cells: 373 Number of references: 28 Combinational area: 125016.710938 Noncombinational area: 73424.179688Net Interconnect area: 49086.523438 **Total cell area:** **198437.968750** **Total area:** **247524.500000**

Table 8. Input_Module[16_32_64]_Data Arrival Time

16 Packet_Array_timing	32 Packet_Array_timing	64 Packet_Array_timing
data required time 7.78 **data arrival time -7.77** --- **slack (MET) 0.01**	**data required time 9.69** **data arrival time -9.62** --- **slack (MET) 0.07**	**data required time 15.82** **data arrival time -15.37** --- **slack (MET) 0.45**

Table 9. Input_Module[16_32_64]_Dynamic_Cleakage Power

16 Packet_Array_power	32 Packet_Array_power	64 Packet_Array_power
Cell Internal Power = 10.1695 mW (98%) Net Switching Power = 182.6510 uW (2%) **Total Dynamic Power = 10.3521 mW (100%)** **Cell Leakage Power = 260.6909 uW**	Cell Internal Power = 15.3948 mW (98%) Net Switching Power = 267.9962 uW (2%) **Total Dynamic Power = 15.6628 mW (100%)** **Cell Leakage Power = 637.3826 uW**	Cell Internal Power = 22.1269 mW (99%) Net Switching Power = 315.2994 uW (1%) **Total Dynamic Power = 22.4422 mW (100%)** **Cell Leakage Power = 1.0510 mW**

6. Summary of the Synthesis Results of Input Module

Table 10. Summary of the Synthesis Results of Input Module

	AREA			*TIMING*			*Dynamic POWER*		
	A16	A32	A64	A16	A32	A64	A16	A32	A64
Results after applying constraints	96629	130011	247524	8ns S. met-0.01	10ns S. met-0.07	16ns S. met-0.45	10.35mw	15.66mw	22.44mw
Partition & then compile	91235	129378	240883	SLCAK met	SLCAK-0.35	SLACK-0.15	10.01 mw	14.91mw	21.70mw

7. ASIC Implementation of the input_module [16_32_64] with Second Set of Constraints

To analyze the synthesis results, in this category only 3-basic constraints were uniformly applied to all 3-designs of input module: Clock period – 20 ns, Input delay – 4 ns, Output delay – 4 ns.

Table 11. Summary of the Synthesis Results of Input Module after Second Set of Constraints

Cell Area		**16_Packet Array**	**32_Packet Array**	**64_Packet Array**
		75360.55	136437	264191
Power	**Dynamic**	1.6649 mw	2.9849 mw	1.4401 mw
	Cell Leakage	179.0064 uw	315.9265 uw	598 uw

8. Four times Faster Scheduler than IQ_VOQ based_Input Block [Error! Bookmark not defined.]

Table 12. Comparative Table

Clock Period	5 ns	10 ns	15 ns	20 ns
Input module	SLACK VIOLATED	SLACK VIOLATED	SLACK VIOLATED	SLACK MET
Scheduler	SLACK MET			

Scheduler can run at 5ns while input module run at 20ns without violating slack. This makes scheduler faster 4 times compared to input module.

9. Conclusion

To handle uniform and bursty traffic at each input module, effective use of the size of the packet array, FIFO and the linked array could help. The above comparison of the input module for various sizes of the packet array facilitates to select the appropriate size depending upon the input traffic by IP block. Right selection could also help to make design more area, time and power efficient. IQ_VOQ based input module designed for the packet array of size 32. This input module accept packets in two phases each of 36 bits. To handle non uniform traffic, two more sizes of the input module designed, mainly 16 and 64. It facilitated the design engineer to select the appropriate size of the input module depending upon the input rate of feeding. The scheduler is four times faster than the input block for the same kind of parameters set as constraints.

References

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm", In SIGCOMM '89: Symposium proceedings on Communications architectures & protocols **(1989)** pp. 1–12.
[2] Abhay K. Parekh and Robert G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," IEEE/ACM Trans. Netw., Vol. 1, No. 3, pp. 344–357 **(1993)**
[3] Abhay K. Parekh and Robert G. Gallagher, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," IEEE/ACM Trans. Netw., Vol. 2, No. 2, pp. 137–150 **(1994)**
[4] M. Shreedhar and George Varghese, "Efficient fair queueing using deficit round robin," In SIGCOMM '95: Proceedings of the conference on Appli- cations, technologies, architectures, and protocols for computer communi-cation, **(1995)** pp. 231–242
[5] S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output qeueueing with a combined input/output-queued switch," IEEE J. Selected Area in Commun., vol. 17, no. 6, pp. 1030–1039, **(1999)** June.
[6] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space division switch," IEEE Trans. Commun., vol. 35, pp. 1347–1356, **(1987)** December.
[7] Y. Tamir and G. L. Frazier, "High-performance multi-queue buffers for VLSI communications switches," SIGARCH Comput. Archit. News, vol. 16, no. 2, pp. 343–354, 1988. [12] H. Obara, "Optimum architecture for input queuing ATM switches," Elec-tronics Letters, vol. 27, no. 7, pp. 555–557, **(1991)** March.
[8] H. Obara and Y. Hamazumi, "Parallel contention resolution control for input queuing ATM switches," Electron. Lett., vol. 28, no. 9, pp. 838–839, **(1992)** April.
[9] Thomas E. Anderson, Susan S. Owii, James B. Saxe, and Charles P. Thacker, "High-speed switch scheduling for local-area networks," ACM Trans. Comput. Syst., vol. 11, no. 4, pp. 319–352, **(1993)**
[10] H. Duan, J. Lockwood, and S. Kang, "Matrix unit cell scheduler (MUCS) for input buffered ATM switches," IEEE Commun. Lett., Vol. 2, No. 1, pp. 20–23 **(1998)**
[11] Nick McKeown, "The iSLIP scheduling algorithm for input-queued switches," IEEE/ACM Trans. Netw., vol. 7, no. 2, pp. 188–201, **(1999)**
[12] M.A. Marsan, A. Bianco, E. Leonardi, and L. Milia, "RPA: A flexible scheduling algorithm for input buffered switches," IEEE Trans. Commun., vol. 47, no. 12, pp. 1921–1933 **(1999)**
[13] Vilas Nitnaware: A paper titled "An on-chip Interconnect Embodying I-slip algorithm" published in the International Conference, Recent Advances in Signal Processing and Automation, in the University of Cambridge, UK, **(2009)** February 21–23.
[14] Rehan Maroofi,1 V. N. Nitnaware,2 and Dr. S. S. Limaye3: "AREA- EFFICIENT DESIGN OF SCHEDULER FOR ROUTING NODE OF NETWORK-ON-CHIP" in International Journal of VLSI design & Communication Systems (VLSICS) Vol.2, No.3 **(2011)** September. [DOI : 10.5121/vlsic.2011.2309 111]
[15] Adnan Aziz, Amit Prakash, and Vijaya Ramachandra, "A near optimal scheduler for switch-memory-switch routers," in Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, SPAA '03, **(2003)** pp. 343–352.

[16] SSYNOPSYS Manual for Design Vision/Design Compiler available at SOLVNET {https://solvent.synopsys.com/}.

Authors

Bhavana S. Pote is a post-graduate student in the Department of Electronics at Ramdeobaba Kamla Nehru College of Engineering in Nagpur, India, pursuing her M. Tech in VLSI. She has a B.E. in Electronic Engineering from Nagpur University.

Rehan Maroofi is a post-graduate student in the Department of Electronics at Ramdeobaba Kamla Nehru College of Engineering in Nagpur, India, pursuing his M. Tech in VLSI. He has a B.E. in Electronic Design Technology from Nagpur University.

Prof. Vilas Nitnaware is pursuing PhD from R.T.M.Nagpur University, India, in the field of "SOC Interconnect". He is an Assistant Professor and working as a Coordinator for the Department of Electronic Design Technology for the last 10 years. He has been associated with microprocessors and microcontrollers related subjects right from the start of his career. His Masters was in the field of VLSI. He has undergone training on Synopsys "Design Compiler". He has developed some embedded system products and also delivered training on microcontrollers and their applications at various Institutes.