# Design and Implementation of a High-Speed ATM Host Interface Controller

Chan Kim, Jong-Arm Jun, Yeong-Ho Park, Kyu-Ho Lee, Hyup-Jong Kim
Electronics and Telecommuncations Research Insititute, Korea

## Abstract

*The performance increase in computers and ATM networks together with emerging high bandwidth services has resulted in bottleneck at the host-network interfaces. This paper describes the design and implementation of an ATM host interface controller ASIC which relieves the host CPU from processing burdens by doing most of the SAR processing by hardware and also provides high performance. This NIC uses local memory to store control data as well as the received cells and it internally has STM1 framer with clock recovery function. The ASIC is a single-chip solution for the implementation of high performance low-cost ATM network adapters for computers having PCI bus.*

## Introduction

As ATM network penetrates into the traditional local and even residential area network, the cheap, compact implementation of high performance ATM host-network adapter becomes more demanded because we are seeing the increasing performance in both the computers and also in networks along with the emergence of high bandwidth multimedia services. This paper describes the design and implementation of a single chip ATM network interface controller called ASAH-NIC which can be used to implement this low-cost but high performance ATM adapters.

The ATM Subscriber Access Handler - Network Interface Controller (ASAH-NIC) provides the segmentation and reassembly functions for AAL5 and non-AAL5 traffic with internal 155Mbps physical layer functions with clock recovery. The number of connections which can be serviced at the same time is only limited by the size of the local memory to be used attached to the chip. Specifically, ASAH-NIC uses local memory to store control data for segmentation and reassembly of practically non-liminted number of connections and to store received cells before they are reassembled into packets. The segmented cells are stored in internal 10 cell FIFO before sent to the physical layer. This ASIC also has add-drop function for single-ring configuration which will be used in a residential ATM network.

## Architectural Overview

Fig. 1 shows the functional block diagram of ASAH-NIC. It is composed of PCI core, DMA master, DMA slave, segmentation engine, reassembly engine, add-drop control and utopia interface, local memory arbiter and interface and SDH framer. The PCI core provides the capability of direct interface to PCI bus with no glue logic. Since most of the functions are done by segmentation and reassembly engine with DMA read and write capability, the host CPU is much relieved from the chores of protocol processing and data movement than in traditional adapter design.

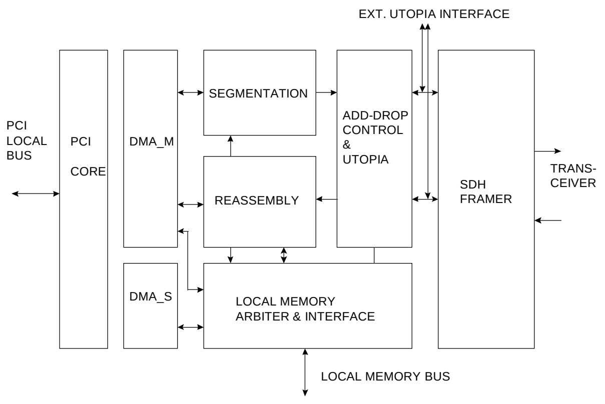


Figure 1. ASAH-NIC functional block diagram

The segmentation and re-assembly block performs all functions related to AAL0, AAL3/4, and AAL5 processing. ASAH-NIC provides complete SAR functions for AAL 5. ASAH-NIC allows pre-formatted OAM cells to be inserted or extracted on an active connection.

### PCI Core

The PCI core is a commercially available one from Virtual Chips Inc. It conforms fully to the PCI specification and has four separate FIFOs for master read, write and slave read, write and provides easy to understand interfaces for the back-end logic.

### DMA Master

The DMA master block is divided into arbiter, DMA read block for segmentation, DMA write block for reassembly. The arbiter arbitrates between DMA segmentation and DMA reassembly. The DMA segmentation block performs DMA read according to the DMA request from the segmentation engine. The segmentation engine requests DMA to the DMA master block providing such information as the start byte location of the DMA read, the number of bytes, and other information needed to form ATM cell like cell header, trailer, partial CRC32, etc. The DMA segmentation block returns the segmented ATM cell to the segmentation engine which is then stored in the segmentation cell FIFO. Because the DMA address and size given from the segmentation engine is in byte unit, the DMA segmenation block converts these byte information to corresponding word information and extracts from the word stream delivered from the PCI bus only the needed bytes to make a new aligned 32 bit word stream. This is a powerful function which makes it completely unnecessary for the host CPU to move or align data after higher layer protocol processing. The read data from the PCI buffer is temporarily stored in the FIFO which is internal to the DMA segmentation block and ATM cells are formed using the data read from the FIFO and other data like ATM cell header, trailer, etc. The DMA reassembly block performs DMA write according to the DMA write request from the reassembly engine. The reassembly engine requests DMA write of the cell payload to the DMA master block providing such information as the host memory address to write the data at, the local memory address to read the cell data from., the partial CRC32 for the CRC calculation, and so on. The DMA reassembly block moves the ATM cell from the local memory to the PCI memory to form the CPCS-PDU packet in the host memory. For AAL5, partial CRC32 value is returned to the segmentation or reassembly engines for later use.

## DMA Slave

The DMA slave block responds to the PCI slave access request coming from the PCI core with the PCI command and start address. If the request is a memory read, DMA block requests predefined number of words to device register block or memory arbiter block according to the called addressed range. As the data is delivered from device register block or memory block, the data is directly passed and written into PCI slave read buffer until stop_write signal is asserted. The data coming after stop_write is not written to the PCI buffer. The number of prefetch words are programmable. If the request is memory write, the DMA slave block reads the PCI core's slave write buffer and write the data to internal FIFO monitoring PCI buffer's empty flag until last_out signal is asserted or the DMA slave's internal FIFO is full. After moving the data from the PCI slave write buffer to the internal FIFO, it requests to write the stored number of words to the device register block or memory block according to the called address range. If reading the PCI slave write buffer was ended due to the internal FIFO's going full, the slave write controller returns to reading PCI slave write buffer for the remaining words.

## Local Memory Arbiter / Interface

Local memory arbiter and interface block's function is two fold. First, it arbitrates the access request for local memory among five processing blocks. The arbitration is done for read and write each and at the most front end, the read and write is arbitrated. For arbitration, the utopia receive block's request for hashing and writing received cells is given the top priority. Though arbiter jumps to the utopia receive block service(if pending) after finishing a low priority request, when returning from the higher priority service, the cyclic priority assignment among the lower priority pending requests before the jump is remembered and resumes in unaltered manner.
The second important function of the memory arbiter/interface block is the provision of programmable timing interface for various local memory speeds. For this speed matching, it has internally two FIFOs for write and read operation and access times are extended according to the programmable memory speed. To increase the throughput of the memory access, the read and write controller adopts some degree of pipelining concept. That is, for the writing, the write to the local memory is started right after the first word is written in the FIFO because the local memory is at least not faster than internal clock operation And for the reading, after the data delivery is started for a request, the next requst is checked and passed to the front read controller.

## Segmentation Engine

The segmentation block is composed of main segmentation manager, buffer link manager, free buffer descriptor manager, UBR pointer manager, and status report manager. When the host has data to send, it forms the buffer descriptor and sends it to the segmentation engine through the buffer descriptor queue. The buffer link manager reads buffer descriptors from the BDQ and copies it in a free location of the buffer descriptor table and links it to corresponding VC table using pointer fields in the VC table. Independently of this linking actions, the main segmentation manager scans the schedule table which has as the entry the pointers to VC table to be serviced. Fig. 2 shows some of this relationship.

When a VC table is serviced, the main segmentation manager loads the VC table and if exists, loads the first buffer descriptor linked to that VC table. It then calculates the host memory address and number of bytes to read to make cell from the current buffer. The number of bytes read from the buffer is kept in the VC table so that next time the segmentation can start from the last read position of the buffer.

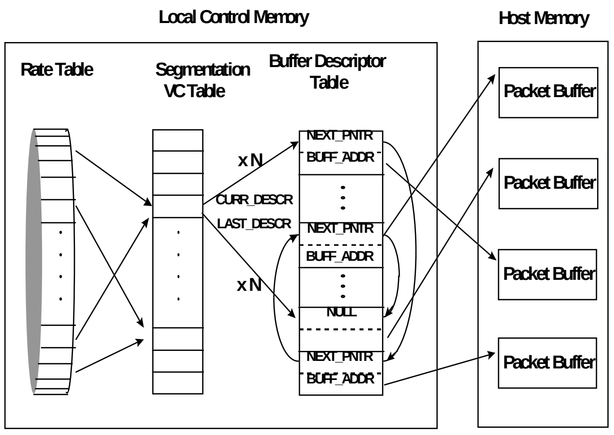
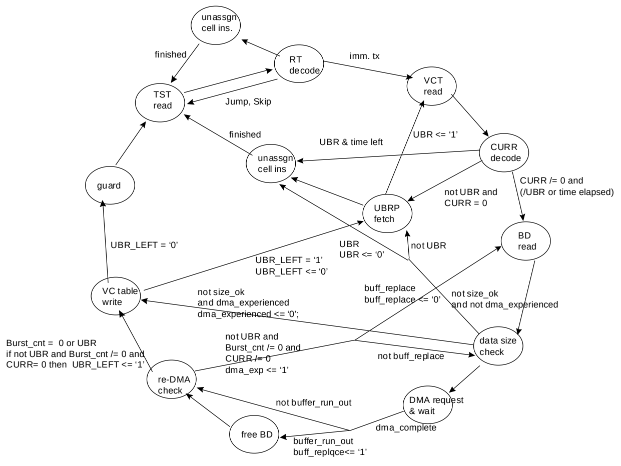


Figure 2. Segmentation Operation Flow

Burst cells can be generated with one such loading as directed in the VC table. The VC table is updated at every completion of a DMA transaction and after servicing a connection, the main segmentation manager writes the VC table back to the local memory and begins scanning the schedule table. Meanwhile, when a buffer's data is completely segmented out, the buffer descriptor is returned to the free buffer descriptor list. When there is not enough data to transmit for CVBR, UBR connection is serviced.  Fig. 3 shows the simplified state diagram of the main segmentation engine.



unassgn cell ins.
finished
RT decode
imm. tx
VCT read
TST read
Jump, Skip
finished
UBR & time left
UBR <= '1'
CURR decode
unassgn cell ins
guard
UBRP fetch
not UBR and CURR = 0
CURR /= 0 and (/UBR or time elapsed)
BD read
UBR_LEFT = '0'
UBR UBR <= '0'
not UBR
UBR_LEFT = '1' UBR_LEFT <= '0'
VC table write
not size_ok and dma_experienced dma_experienced <= '0';
buff_replace buff_replace <= '0'
not size_ok and not dma_experienced
Burst_cnt = 0 or UBR if not UBR and Burst_cnt /= 0 and CURR= 0 then UBR_LEFT <= '1'
not UBR and Burst_cnt /= 0 and CURR /= 0 dma_exp <= '1'
not buff_replace
data size check
re-DMA check
not buffer_run_out
DMA request & wait
dma_complete
free BD
buffer_run_out buff_replqce<= '1'

Figure 3. Simplified State diagram of the main segmentation manager

This segmentation engine can handle some special cases of the data to be mapped into a cell's payload being located in separate buffer sent from the host.  In this case, though the remaining bytes are not enough for the cell generation, if the following buffer is already linked to the VC table, it assumes the data will be enough with that of the following buffer and requests first DMA read for the data in the current buffer and then second DMA read for the remaining data in the following buffer.  Fig. 4 shows this split DMA case for  various buffer sizes of the following buffer.

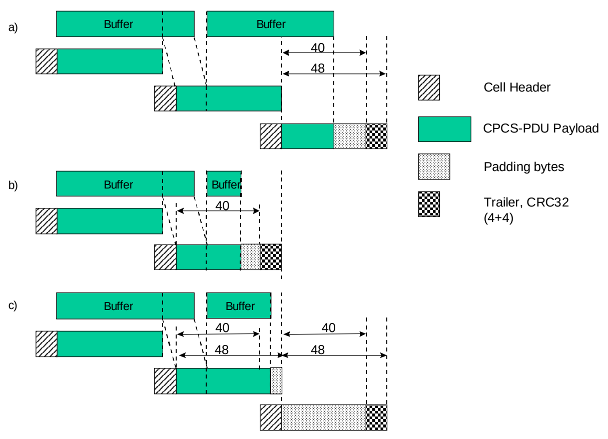


Figure 4. Segmentation Examples - Split DMA

The free buffer descriptor manager manages the unused descriptors of the buffer descriptor table in linked list form and gives the free descriptor in the head to the buffer link manager when requested for new free one and appends the used descriptor to the tail of the free list when it is returned by the main segmentation manager.

The VC table corresponding to the UBR is linked to circular list and serviced one by one (only those who has buffer linked to them) when selected CVBR connection does not have enough data to send.  When selected UBR data is not enough at the time either, null cell is inserted. The UBR pointer manager searches this UBR ring to find the VC table which has buffer and at stand-by mode returns the address when requested by the main segmentation manager.

The status manager writes status information when requested by the main segmentation manager at the point of DMA completion by latching the status information and writing that to the status queue in the local memory and generating interrupt.

## Reassembly Engine

The cell buffer is a block of local memory space which is divided into cell space and operated cyclically like

FIFO memory. The received cell is first stored in cell buffer and taken out for reassembly one by one. The reassembly engine is divided into hash table manager, main reassembly manager, free buffer descriptor manager, and status queue manager. The hash table manager reads two words from the cell buffer(hash index and cell header) and searches the hash bucket chain for the VC table matching the header. When matching VC table is found it gives this address of the VC table to the main reassembly manager for further processing. The main reassembly manager reads the VC table and after interpreting it, requests DMA write to the DMA master block supplying the host memory address to write the cell payload data at and the local cell buffer address to read the cell from. After the DMA write is completed the VC table is updated. The number of bytes written to the buffer is kept in the VC table to start from the last write position next time. When the host buffer to be used is going to be full by current cell, the main reassembly manager prefetches the next free buffer address from the free buffer descriptor manager and divides the DMA request into two DMAs of which the first one is made with link request and link address (new free buffer address) which is to be used to link the received buffers into chain of a packet in the host memory. Likewise, the main reassembly manager fetches a new free buffer address when a new buffer is needed. After finishing the reassembly for a CPCS-PDU packet, the main reassembly manager requests status report to the status report manager. The status report manager latches the status information and writes them into the status queue and generates interrupt. A free buffer descriptor manager manages the unused host buffer descriptors in linked list form like that of segmentation block. The free descriptor is dispensed to the main reassembly manager when a new buffer is needed and it is returned when the host has consumed the received data buffer. (see Fig. 5)

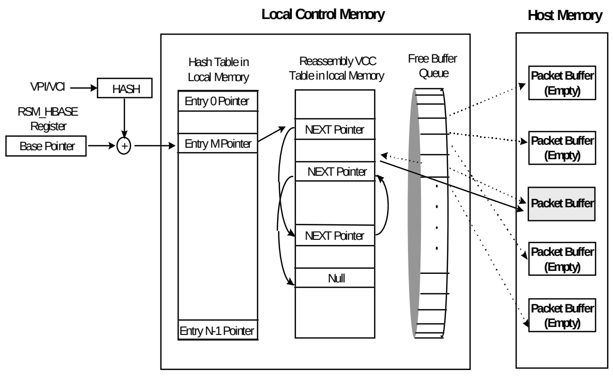


Figure 5. Reassembly Operation Flow

## Add-Drop and UTOPIA Interface Block

When a cell is arrived from physical layer through UTOPIA interface, the add-drop controller first generates hash index using the cell header and reads that address of local memory to get add-drop information and start address of the hash bucket chain for the header. If the returned value show that the cell should be received(or dropped), the cell is written to the local cell buffer with one word of hash chain address pre-pended to the cell. If the read value shows that the cell is to be bypassed, than the cell is written into the bypass FIFO for transmission.

The master node on single ring manages the bandwidth allocation per node on the ring and informs the assigned bandwidth values to each slave node systems. The ring access controller of each node generates token for transmit according to the assigned bandwidth. A transmit cell can be transferred on single ring when there is a token.

## Conclusion

We have described the architecture of a high-sped ATM network interface controller. It was designed to reduce the performance degradation problem at the host - network interface, and provide an efficient shared medium access method for residential ATM network with single ring topology. The segmentation and reassembly operation mechanisms are addressed. And the single ring access mechanism and SDH framer architecture are also considered. It is ideally suited for ATM host adapters, ATM hubs, bridges with PCI bus. Functional level design has been verified using the front-end VHDL simulation tool(Visual HDL and Synopsys) and the gate-level timing simulation and test design was done using C-MDE of LSI Logic Corporation. The chip is now at its final check stage for sign-off at this moment of writing.

## Acknowledgments

We would like to thank Dr. Jae-geun Kim, Dr. Mun-kee Choi, Sun Kang, Yeong-won Hwang, Young-wook Cha, Ik-kyun Kim for their valuable advices and encouragement

## References

[1] Traw, C.B.S., and Smith, J.M., “ Hardware/Software Organization of a High Performance ATM Host Interface,” IEE JSAC Vol.11, No.2, Feb.1993.
[2] ITU-T Rec. I.432. I.361, I.363, I.371
[3] PCI Local Bus Specification, Rev 2.1. June. 1995