IEEE International Conference
on Telecommunications
Bucharest, Romania, June 04–07, 2001

The Architecture of an IP over ATM Processor

R. Abielmona, S. Abielmona, M. Abou-Gabal,
W. Hermas, V. Groza, E. Petriu
School of Information Technology Engineering
University of Ottawa, Ottawa, ON, K1N 6N5, Canada
Phone: (613) 562-5800 x6315, Email: rabielmo@site.uttawa.ca

***Abstract** – In this paper we present Proknet, which is a play on the following words: network and processor. It represents a module that is capable of receiving incoming variable-sized IP packets, and outputting fixed-sized ATM cells. The applications of this module are numerous, especially in the communications field, where this type of circuit is needed in order to implement the ATM forum specifications. Our design is outlined and briefly discussed, while our final results are presented and analyzed.*

***Keywords** – IP over ATM, Packet Segmentation, Network Processors*

I. INTRODUCTION

The Internet has become one of the most diverse, information-filled libraries this century has ever seen. Its contents range from a variety of different topics, some of which include education, information, leisure and entertainment, as well as research and development. As the Internet becomes more reliable, we are evolving into dependencies for its various tools and services. Our dependency hence increases our awareness of the Internet, which in its turn leads to people demanding more diverse and tailored services that they can map to their everyday lives. This demand drives the emergence of new technologies as well as the refinement of old ones in ways never seen before. Two of the leading technologies that have surfaced are the **Internet Protocol (IP)**, and **Asynchronous Transfer Mode (ATM)**.

The main goal of this research was to provide an IP over ATM segmentation entity, that accepts a variable-sized IP packet, and transforms it into fixed-sized ATM cells. The design goals set out at the start of the research included forum compliance, efficient implementations, digital design adherence, speed of execution and network processor functionality. The research provided a solid foundation for IP over ATM segmentation abilities, and made the design of network processors, a current hot topic within the communications field, simpler to visualize and understand.

The undertaken work was accomplished through the use of various tools, including the *Synopsis* Verilog compiler and simulator, the *Cadence* design analyzer tool for synthesis and the *Cadence* waveform analyzer tool for simulation results analysis. Preliminary work was done on the *Xilinx* Foundation Student Edition design entry tools [1].

The next section gives a little more detail on what IP and ATM are, how they are inter-related, and where they fit in Proknet. The system and detailed architectures, as well as the final results are then shown, finishing with an analysis of the overall research and a few concluding remarks.

II. PRELIMINARIES

The IP communication protocol utilizes connectionless-based datagrams that are routed using hop-by-hop routing algorithms to transfer information, while ATM utilizes connection-based cells that are guided using a pre-planned path between nodes. The main differences between the two is the processing power required by both as well as the reliability offered by both protocols. The total length of the IP packet is variable, ranging from 20 bytes to 65,536 bytes [2]. The datagram approach is best suited for this type of routing methodology because each packet is treated independently, with no reference to packets that are gone before it. Different packets may thus end up selecting different routes depending on the network's current status, and thus may end up arriving at the destination in different orders. Neither error control nor reliability mechanisms are exercised in this protocol, which allows for the transmission of such variable-sized packets, yet routing of every single packet is required which consumes greater processing power. Now, the total length of an ATM cell is fixed at 53 bytes. The cell approach is alternatively best suited for this type of routing methodology because each packet is guided via a pre-planned route, where each cell traverses the dedicated route for the entire length of the data transmission. With this scenario, error control and reliability mechanisms are provided on a connection basis rather than on a cell basis, which reduces the processing power required by each node [3].

Currently though, as a result of different amalgamated services, more and more packet-switching networks have been making use of the more reliable cell-switching networks. The benefit from both technologies hence improved the different service providers' delivered products. This allowed networks around the world to present a service that is as reliable and dedicated as ATM, while utilizing IP's greater transmission capability and reduced overhead in terms of route provisioning [4].

Out of this concept transpired the migration of networks to offer IP over ATM. As described above, this protocol uses both technologies to deliver data, voice and video from one **Customer Premise Equipment (CPE)** to another CPE and ultimately to the user that desires this service. In order to be able to transmit variable-sized IP packets over a fixed-cell ATM network, one has to be able to segment the IP packets into 48-byte cells at the transmitter, and also be able to reassemble these same cells into the original packet at the receiver. The segmentation function is the focus of this research.

For our IP application, the characteristics are that timing is not required between the source and the destination, the bit rate is variable, and it is a connection-less transfer. This classifies IP as a class D application, which entails the utilization of the type 5 **ATM Adaptation Layer (AAL)** protocol, which is a newly defined protocol. Like all other AALs, the higher layer protocols send a block of data to the **Convergence Sublayer (CS)**, which encapsulates them into **Protocol Data Units (PDU)** [5]. Actually, the CS is referred to as the **Common Part Convergence Sub layer (CPCS)**, making this layer in the CS the one that performs common functions for the higher protocols, while the **Service Specific Convergence Sub layer (SSCS)** performs the service specific operations depending on which AAL is being utilized at the time by the higher layer protocols, based on their application.

III. HIGH-LEVEL ARCHITECTURE

To successfully transition from the design phase into the implementation phase, it was necessary to select a base architecture. This task was not taken lightly as it influenced many future design aspects and decisions. The system architecture had to present the designer with the main components, their interconnections and their operations. That said, the system architecture was not going to present the internal implementation details of any component, leaving it up to the imagination and innovation of the engineer to design a functionally correct system. For this research, the system architecture went through quite a few design cycles before its eventual hardened declaration. The fact that the architecture is presented first in this paper is due to its vitality for a smooth module design transition. The drawbacks of hard-coding the circuit to match the underlying architecture were numerous, and expensive in terms of design cycles. Other pitfalls of a bad system architecture were work-arounds implemented in order to do exactly that: work around a system bug. On the other hand, a well thought-out architecture provided a baseline for all designers to review and synchronize to. The chosen architecture is shown in figure 1.

Proknet is made up of the following (refer to figure 2): a **Total Length Logic (TLL)** module, an **Arithmetic Logic Unit (ALU)** module, a **Trailer** module, a **Cyclic Redundancy Check (CRC)** module, a **Segmentation and Reassembly (SAR)** module and a **Control Unit (CU)** module.

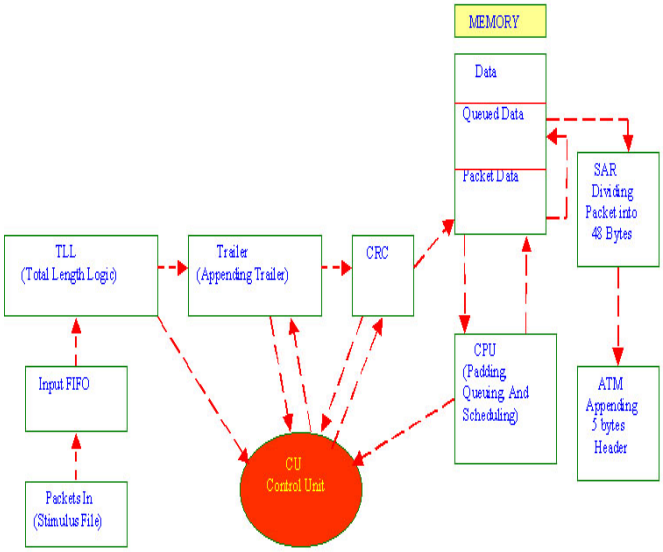

Fig. 1. Final system architecture

A. TLL module

In order to be able to design a fully functional segmentation entity according to our specifications, we required the retrieval of certain information form the IP packet itself. This information was necessary in order for the segmenter to proactively monitor the stream of packets entering its module, as well as perform preliminary setups for the downstream modules, such as padding and memory storage. The most important information contained in the IP packet is the **Total Length (TL)** field, which is stored in the 3^{rd} and 4^{th} byte of the header, making it a 16 bit field, and hence setting the total length of the packet to be between 20 bytes (minimum length of the header) and 65,535 bytes (maximum allowable transferable length).

By foreshadowing the TL, we were able to determine several other necessities that were to be used by the pipeline at its various stages. Thus, the need arised for a method that allowed us to read the TL field from the incoming packet, store it for further use as the packet traverses the pipeline, as well as determine the necessities early in the pipeline in order to ensure a synchronized transfer of the packet from module to module. Some of these necessities were calculating the number of padding bits required by the packet as well as the storage of that value in a manner that is easily accessible to the control unit and hence the modules along the pipeline. The latter entity was to operate assuming the follow conditions:

1. Externally fed clock oscillating at 25 MHz. Thus our processing power allows to output data at a rate of 200 Mbps. This is obtained by the following: 8 bits/cycle (width of

bus) * 25 MHz (cycles/sec) = 200 Mbps

2. Our pipeline is to be fed one byte at a time, clocking at a speed of 1/25 MHz = 40 ns, which is the period of one clock cycle.

B. ALU module

In this section, we introduce the design of the ALU. It is a purely combinational circuit which represents the workhorse of the main processor, and thus must be carefully designed, in order to ensure that it does not pose any bottleneck issues [6]. Since the ALU aids in the instruction execution, it resides on the datapath of the main processor, and feeds a register (the **accumulator**) in order to hold the value that it just calculated. The ALU also aids us in minimizing logic gates and routing paths, since it can also be used to handle memory access instructions, which need calculations to be performed in order to be executed. Different algorithms can be implemented within the ALU to speed up the operations, while for even faster execution, multiple ALUs can be instantiated within the circuit to distribute the processing load. The ALU that was designed and implemented in this case consisted of a simple 16-bit adder-subtractor unit, which performed its operations on two inputs, and directly stored its output into the accumulator. Its detailed design is not presented below as a traditional ALU design was undertaken.

C. Trailer module

The trailer is an 8-byte field that was appended to the end of the packet. The trailer contains 4 fields, which are described below:

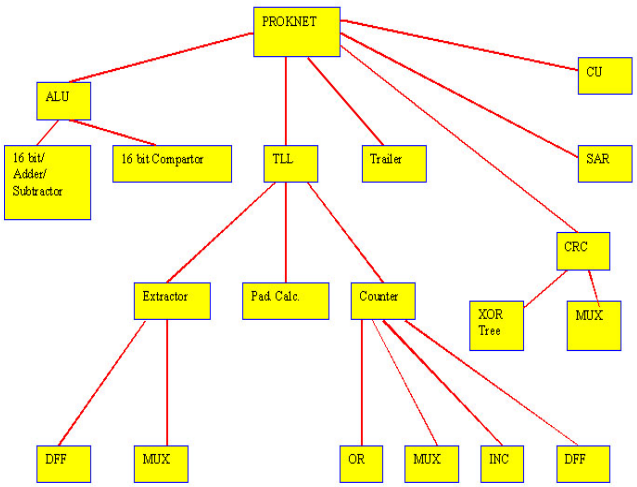

Fig. 2. System breakdown

UU (User to User) The field contains one octet of information, which is transferred transparently between users of the AAL5 CPCS users

CPI (Common Part Indicator) The field is used to interpret subsequent fields for the CPCS functions in the CPCS header and trailer

Packet Length This field contains the size of the incoming packet.

CRC-32 This field is filled with the value of a CRC calculation which is performed over the entire content of the packet. It is filled with zeros before going to the CRC circuitry. Once passed through the CRC-32 circuitry, it is filled with the correct values and then transmitted.

Please refer to figure 3 for a diagram of the following explanation:

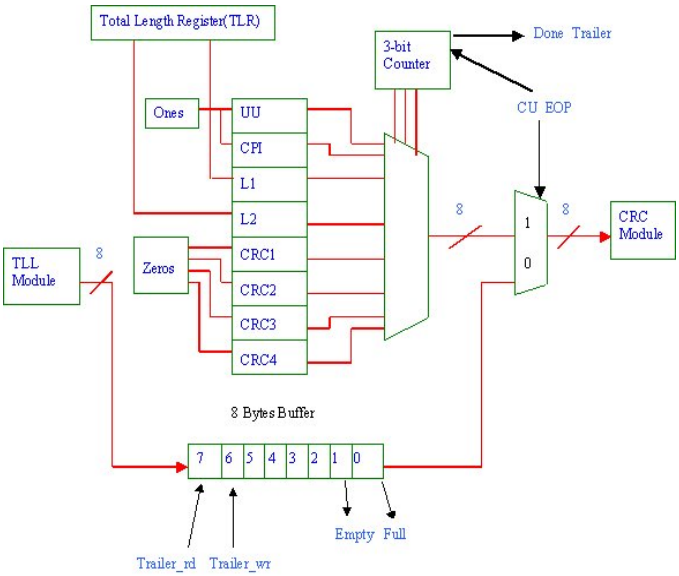

Fig. 3. Trailer module architecture

1. When a byte of a packet is received from the TLL module, the control unit writes into the 8-byte **First-In First-Out (FIFO)** memory
2. On the next clock cycle, the byte that was written in step 1, gets read and at the same time another byte is written into the FIFO
3. Step 1 and 2 are repeated until the *CU_EOP* signal is asserted by the Control Unit. When *CU_EOP* is asserted, two events occur. First, the 2-1 input multiplexer switches up and secondly, the 3-bit counter starts selecting the select signals for the 8-1 input mux
4. The counter keeps counting until all Trailer fields are sent out and have been placed on the pipeline, then the counter sends a *Done_Trailer* signal to the Control Unit, informing the latter that the trailer has been appended to the packet.

D. CRC module

The **cyclic redundancy check (CRC)** is a number derived from, and stored or transmitted with, a block of data in order to detect corruption. By recalculating the CRC and comparing it to the value originally transmitted, the receiver can detect some types of transmission errors. A CRC is more complicated than a checksum. It is calculated using division either using shifts and exclusive ORs or table lookup (modulo 256 or 65536). The CRC is ”redundant” in that it adds no information. A single corrupted bit in the data will result in a one-bit change in the calculated CRC but multiple corrupted bits may cancel each other out. Most CRC implementations operate 8 bits at a time by building a table of 256 entries, representing all 256 possible 8-bit byte combinations, and determining the effect that each byte will have. CRCs are then computed using an input byte to select a 16- or 32-bit value from the table. This value is then used to update the CRC [4]

Since we want to calculate the CRC eight bits at a time, we need an algorithm that will produce the same CRC value as would occur after eight shifts of a bit-wise CRC calculation. There is already a well-established algorithm developed to do the byte-wise CRC. Simply, the contents of the CRC register after eight shifts are a function (exclusive-OR) of various combinations of the input data byte and the previous contents of the CRC register. Following the byte-wise CRC algorithm [3], we managed to implement the functionality.

As a design note, a FIFO buffer was used in the CRC for two reasons. First, the Trailer might be transmitting faster than the CRC can handle (therefore congestion must be avoided); second, the packet must pass through and the last 4 bytes of trailer (which are zeros) must be held in the FIFO, while the 4 *good* bytes of CRC module are placed on the pipeline.

Refer to figure 4 for a diagrammatical view of the following explanation:

1. The Trailer module sends the first byte of packet, thus the Control Unit writes it to the 4 byte FIFO and this is done by asserting *Crc_write*
2. On the next clock cycle, a second packet comes in, the Control Unit will write it into the FIFO, and at the same time it will read the first byte. These actions are done by asserting *Crc_write* and *Crc_Read* simultaneously
3. Steps 1 and 2 are repeated until *CU_Done_Trailer* is asserted, and, when this occurs, two events are triggered: first, the 2 to 1 input multiplexer and the 2-bit counter are enabled, which basically means it is time to output the 4 bytes of CRC (known as predetermined divisor). Therefore, *CU_Done_Trailer* will stay high for 4 clock cycles to allow the 4 bytes to be placed on the pipeline.
4. Once all 4 bytes are placed on the pipeline, the 2-bit counter asserts *Done_crc* signal (meaning that the CRC module has completed its task).

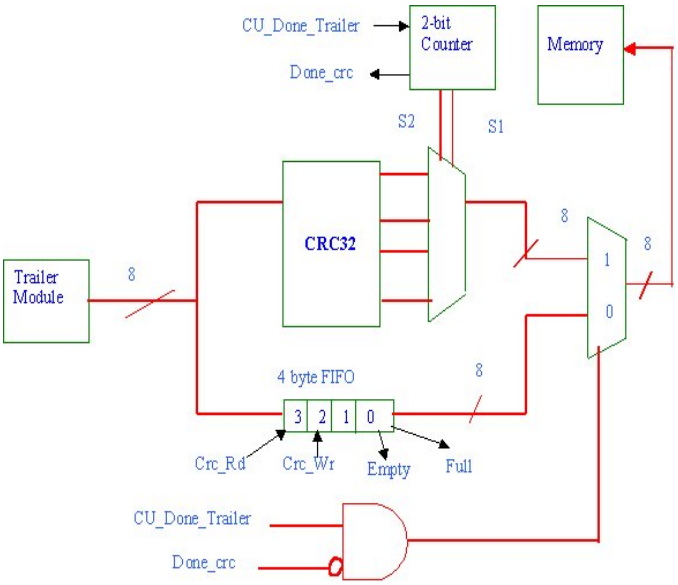

Fig. 4. CRC module architecture

E. SAR module

After the design and implementation of the preliminary modules, we reach the core module of our research labeled the SAR module. Although we intended to implement a segmenter as well as an assembler, the immense time consumption required for this research limited us to the design and implementation of only a segmenter, yet we will continue to refer to this module as the SAR module.

The main purpose of the SAR module is to partition incoming packets into cells. The SAR, residing in the AAL, receives CPCS PDUs that are tagged with a trailer. The SAR then, upon reception of the PDU, is to segment the variable-length PDU into fixed-length cells, each representing now a 48-byte cell, and send them to the ATM layer to be tagged with the appropriate header and transmitted to the physical interface. This is the high-level description of the SAR functionality, yet we have adopted a new method of performing this operation. We have chosen not to transmit the completed packet (CPCS PDU + trailer) to the SAR module at once and allow the SAR to section off 48-byte cells from the PDU for transmission to the ATM layer, until it has exhausted the entire PDU. This method presented two disadvantages from Proknet’s point of view. First, this would impose a strain on the pipeline design that we have chosen for this project. The pipeline is not to be halted for the processing of any type of operation. The above-proposed procedure of the SAR would force the pipeline to delay its operations in the SAR module, hence creating an unnecessary bottleneck for the processing of a PDU of constantly changing length. Second, we have chosen to adopt an 8-bit bus that clocks 8 bits at any one time on the rising edge of our clock. Thus, for example, in order to transfer a 100-byte packet, we require either a faster clock, or a wider data bus able

to accommodate a larger packet, neither of which is available to us. Thus, we slightly modified the operation of the SAR in order to fulfill our design requirements and conditions. The SAR will receive packets from the packet memory only when there is an entire completed packet residing in memory. When this occurs, the packet will not be transferred to the SAR module in its entirety. Instead, it will be transferred from memory to the SAR one byte at a time, where no processing or delaying of that byte is to occur. The byte will simply pass through the SAR module seamlessly, and onto the ATM layer. All the meanwhile, as soon as the first byte of the packet is passed through the SAR, a counter is initiated to count up to '48', signal the ATM layer that we have just reached 48 bytes, reinitialize to '0' and begin counting again. Once the last cell of the packet has been received, a notification is sent, via a control signal, to the ATM layer informing it of that fact, thus allowing the ATM layer to write a bit in its header indicating that this is the last cell pertaining to the current packet.

F. CU module

The Control Unit was renamed to the microprocessor module. Initially, this module was thought to be one of the few first modules to be designed. Time proved otherwise, as we realized that our microprocessor is a tailored one, in that it provides typical microprocessor functionalities, but also extends to supply network processor functionalities. The initial design started with a look-back towards our pipeline design. The latter included modules, that were later scrapped, such as **Queuing and Scheduling (QAS)** and its associated Queue Memory, the Input Buffer Control Unit and the pipeline buffer. These modules were taken into consideration when the initial design was laid out.

The microprocessor chosen for implementation in this project, was an 8-bit microprocessor. All instruction opcodes were limited to 8 bits. The memory was addressed with a 12-bit address bus, while the data path (according to our pipeline structure) was 8 bits wide. The clock period was originally defined to be 40 ns, thus the clock frequency was 25 Mhz. This microprocessor falls under the RISC family of processors, as its instruction is quite reduced [7].

The final memory interface included a dissection of the previously designed interface. The subroutine calls and the interrupt mechanism are no longer needed, hence eliminating with them the need for a stack pointer. One aspect that was modified from previous designs is the break up of the address registers into two: the **write address register (WAR)** and the **read address register (RAR)**. The memory itself is a **ternary-port memory**. Traditionally, when one needs to perform two operations upon the same memory, in one cycle, *dual-port memory* is utilized. In this project, we need to perform *three* distinct operations upon the memory, all in one cycle. The operations are the fetching of an instruction to be executed, the write out

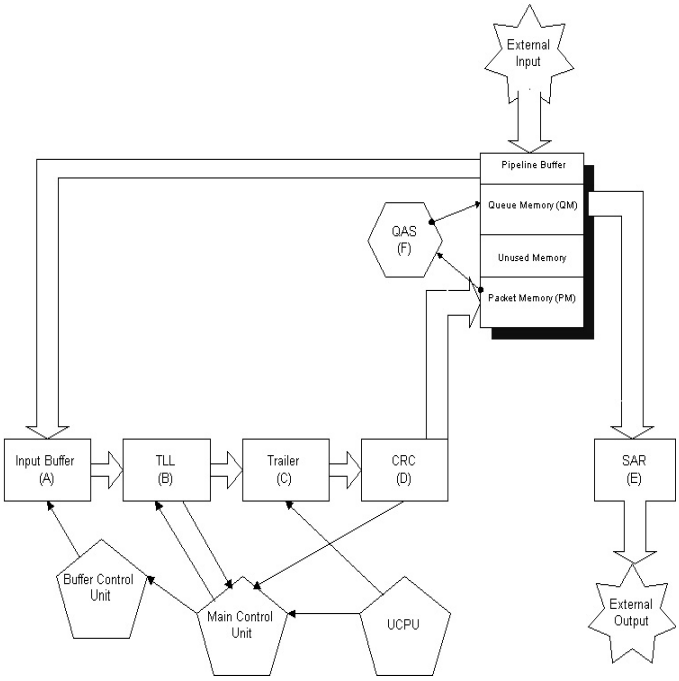

Fig. 5. Initial pipeline design

to memory a byte of packet information from the CRC module, and in some cases, the read out to the SAR module a byte of packet information from the memory. A solution to attempt to reduce the amount of ports on the memory (which are a disadvantage, as each will require a data bus), is to multiplex at least two of the addressing registers together, in order to alleviate the number of data buses to be interfaced to external memory, but this particular solution was not adopted because of two major reasons: first, the operations are independent and do not have any relationships other than the RAR and WAR addressing the packet memory (queue memory was taken out because of the removal of the QAS module); the second reason for not adopting multiplexed addressing schemes is that a decision was made to design the memory internally, and thus avoid the external interface mechanism. The data buses were thus found within the chip, and did not cost the designers any more than their routing specifications. The overall final memory interface is shown in figure 6. The **Program Counter (PC)** is associated with the dual-port address bus, while the **Instruction Register (IR)** is associated with the dual-port data bus. The main address bus is connected to the WAR, while the value to write at the WAR's designated address, is received from the CRC module. Finally, the RAR is associated with the ternary-port address bus, while the SAR module is associated with the ternary-port data bus. All data buses were 8 bits wide, while the address buses were 12 bits wide. That said, not all 12 bits were used in address decoding, as will be seen later on.

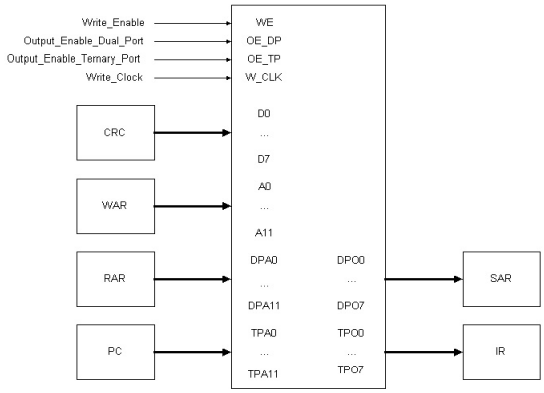

Fig. 6. Final memory interface

IV. RESULTS

Proknet was implemented using the Verilog hardware description language [8]. It has been functionally simulated, synthesized and timing simulated. The space restrictions do not allow us to present all of the results, but the top-level simulation diagram (see figure 7) is shown for clarity.

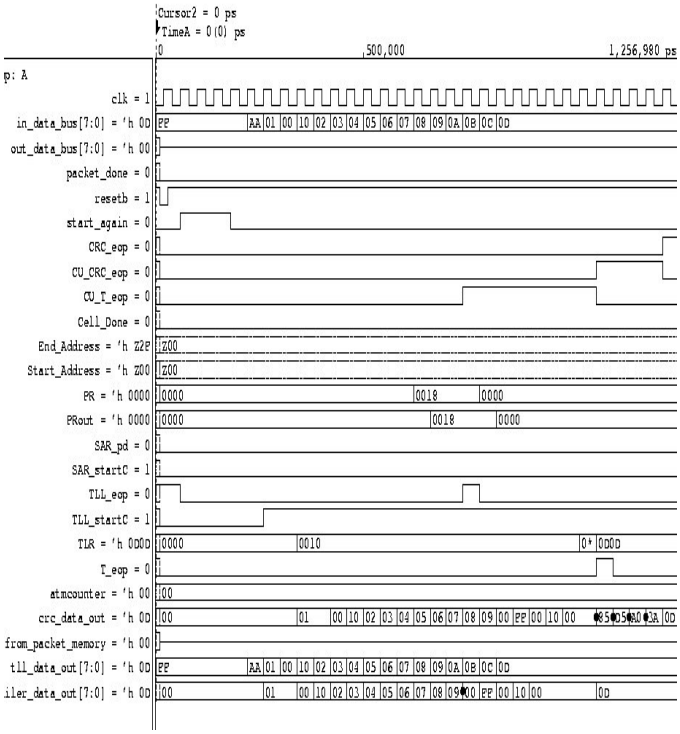

Fig. 7. Top level simulation

V. LIMITATIONS AND FUTURE WORK

A major limitation of Proknet is that it cannot handle more than 2 packets on the pipeline. It can on the other hand handle 1 packet (the size being smaller or greater than 48 bytes), or 2 packets. The limitation is that when more than 2 packets are present on the pipeline, the TLR and **Pad Register (PR)** (the latter holds the number of bytes to pad), as well as the Start_Address and End_Address lines (see figure 7) all latch or store values of the next packet, while the previous one is being processed by the SAR module.

An ATM specification change was made during the design, and that is to only perform the byte-wise CRC calculations on the packet and the trailer, excluding the padding bytes. The reason for that is simplicity of design. In order to make up for this specification change, an assumption about the reassembly side was made: the reassembler will strip the padded bytes first, and then calculate its own CRC to see if it matches the packet's appended CRC information.

Proknet could also benefit from a parallel architecture, where multiple pipelines could be executing on various packets. This system would have to be a massively-parallel system, but would increase the throughput by leaps and bounds. This idea could be beneficial in the re-engineering of today's SARs on the market, as they operate on a queuing and scheduling algorithm, but do not include multiple pipelines.

VI. CONCLUSIONS

This research started off as a migration from traditional microprocessor design to a more innovative network processor design. The idea was a raw one, with only a few references to go by. The design work accomplished proved that this technology is of great benefit to the networking world. Today's major communications field players use SARs as commonly as they use resistors on their printed circuit boards.

The design of an IP over ATM processor provides great insight into the interworkings of a network processor. The work can be easily extended to other technologies merging together, in order to perform the same functionalities, such as ATM over frame relay. Also, given the points mentioned in section V, as well as the implementation of a reassembly-side twin, Proknet will stand as a stand-alone SAR processor.

References

[1] Dave Van den Bout, *The practical Xilinx Designer Lab Book*, Xilinx Inc., 1998.

[2] Martin De Prycker, *Asynchronous Transfer Mode Solution for Broadband ISDN*, Prentice Hall International (UK) Limited, 1995.

[3] Oliver C. Ibe, *Essentials of ATM Networks and Services*, Addison Wesley Longman Inc., 1997.

[4] Fred Halasall, *Data Communications, Computer Networks and Open Systems*, Addison-Wesley Publishers Limited, 1992.

[5] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall PTR, 1996.

[6] Sunggu Lee, *Design of Computers and Other Complex Digital Devices*, Prentice Hall, 2000.

[7] Alan Clements, *Microprocessor Systems Design: 68000 Hardware, Software, and Interfacing*, PWS Publishing Company, 20 Park Plaza, Boston, Massachusetts 02116, 2nd edition, 1992.

[8] Philip R. Moorby David E. Thomas, *The Verilog Hardware Description Language*, Kluwer Academic Publishers, Reading, Massachussetts, 1995.