A Programmable and Highly Pipelined PPP Architecture for Gigabit IP over SDH/SONET

Ciaran Toal, Sakir Sezer
School of Electrical and Electronic Engineering
Queen’s University Belfast
Ashby Building, Stranmillis Road
Belfast, BT9 5AH, N. Ireland, U.K.
Ciaran.Toal@ee.qub.ac.uk

Abstract

This paper details the implementation of a highly pipelined 2.5 Gbps Point-to-Point-Protocol Packet Processor (P5) aimed for the latest System-on-a-Programmable-Chip (SoPC) technology. Throughput rates beyond 2.5 Gbps based on FPGA technology could be achieved by designing a new highly pipelined and parallel processing architecture for frames and datagrams. A novel pipelined data sorting mechanism with an extremely low resynchronization buffer and backpressure scheme are introduced to keep the data memory requirements as low as possible for embedded on-chip applications.

1. Introduction

The emergence of the Internet in recent years has seen a marked change in the nature of traffic carried by telecommunication networks. For a century these networks have been devoted, almost exclusively, to the transmission of voice traffic. These networks have evolved slowly, as in recent decades the analogue network has been replaced by a digital infrastructure. However, there has been an explosive increase in the amount of data traffic since the emergence of the Internet in the last decade.

This trend is set to continue as voice and video data is now being transmitted in packets over the Internet for Voice-over-Internet-Protocol (VoIP), WEB-TV, WEB-Cam and internet video conferencing [9]. The nature of businesses has become dependent on the instant information access and exchange enabled by the internet.

Consumer trends have changed. Digital television channels are distributed over satellite and optical fibre to individual consumer set top boxes; these same television channels provide interactive content-on-demand at the push of a button. More and more people have a home web browser, require more than 1 email account and will not go anywhere unless they are carrying the latest wireless mobile communication technology. Next generation wireless technology will not only be handling voice data traffic but will also be transmitting JPEG images and MPEG video streams as 3rd and 4th generation mobile phones hit the market [8].

While the technology behind the scenes becomes more advanced, the products utilising this technology are becoming more accessible to the consumer. Newer generations of network packet processors must be able to cope with this massive amount of information for this trend to continue.

The project presented in this paper investigates a new parallel and pipelined architecture that can be implemented on new generation programmable logic devices. The PPP processor presents an ideal case study for parallel architectures as the required throughput rate of 2.5 Gbps can only be achieved through hardware acceleration parallel processing. Before implementing a 32-bit P5, an 8-bit (625 Mbps) P5 system was studied and implemented. This gave a sound basis from which the 32-bit system could be compared to.

2. The Point-to-Point Protocol

The Point-to-Point Protocol (PPP) [4], [5], [6], [7] provides a method for transmitting datagrams over point-to-point links and is the most efficient layer 2 protocol for
encapsulating IP datagrams [2]. PPP is widely used as a data link layer protocol in dial-up, cable and ADSL modems and also for data over SDH/SONET links. PPP is composed of three parts:

- A framing method for encapsulating datagrams to transmit over physical layer point-to-point links. PPP uses HDLC like framing [5].
- An extensible Link Control Protocol (LCP) to establish, configure, and test the data-link connection. [4].
- A family of Network Control Protocols (NCP) for establishing and configuring different network-layer protocols. PPP is designed to allow the simultaneous use of multiple network-layer protocols [4].

It should be noted that the main focus of this paper is on the data-path implementation of the P^5.

**PPP Frame Format**

The PPP frame format [4] is shown in Figure 1.

Flag field: All PPP frames begin and end with the standard flag value of 0x7E. This value is character stuffed if it occurs within the payload field.

Address field: This field typically defaults to the hexadecimal value 0xFF to indicate that all stations are to accept the frame. However, this implementation allows this field to be programmable so that it is compatible with MAPOS systems [1], [2].

Control field: PPP may be configured via the LCP to use sequence numbers and acknowledgements for reliable data transmission. This is of particular use in noisy environments such as wireless networks, but will be disabled by default. In normal operating conditions the value of this field is 0x03, indicating the use of unnumbered frames [7].

Protocol field: This field provides an indication of the type of packet held within the frame payload. Protocols starting with a 0 bit are network layer protocols such as IP or IPX, those starting with a 1 bit are used to negotiate other protocols including LCP and NCP. The default size of the protocol field is 2 bytes but this may be negotiated down to 1 byte using LCP.

Payload field: The payload field is variable in length up to a negotiated maximum. If the maximum length is not negotiated via LCP during link establishment a default length of 1500 bytes is allocated. This payload may be padded if necessary.

Checksum field: The checksum field contains the CRC value calculated over the entire frame. In general 16-bit and 32-bit CRC checks are used. For accuracy purposes the system will incorporate 32-bit CRC checking.

PPP frames are delimited using a logical 'flag' denoted by the occurrence of the character value of 0x7E. Safeguards are required against the possibility that this character should occur within the frame payload.

The character 0x7D is used to ‘escape’ any instances of 0x7E. Any instances of 0x7E outside of the flag fields are replaced with a two byte sequence consisting of 0x7D and the original character with its sixth bit complimented. This is best clarified using an example:

A sequence of characters destined for storage within the frame payload may be: 0x31, 0x33, 0x7E, 0x96. Post character stuffing, this sequence will be: 0x31, 0x33, 0x7D, 0x5E, 0x96.

3. System Architecture and Design

Today, commercial PPP packet processors are 8-bit systems. Therefore an 8-bit system was studied, designed and simulated before designing the more complex 32-bit system. Although both systems have the same structure, the fact that only 1 byte is processed in each clock cycle makes the overall control and synchronisation of the 8-bit system much easier.

The P^5 is designed with a general purpose microprocessor interface, to make the system programmable, and with a simplified physical layer interface to interlink to the most common optical transmission systems.

Figure 2 shows a simplified block diagram of the P^5 system architecture. The system is composed of 3 main blocks, a Transmitter, a Receiver and a Protocol OAM (Operations, Administration and Maintenance) Control. Data is buffered before transmission and after reception in memory. The Transmitter formats and packetises datagrams before transmitting over a physical link. The receiver unpackets and extracts the encapsulated datagram from the received data. The Protocol OAM Control provides an efficient interface for control and status information to be exchanged between an external microcontroller and the internal Receiver and Transmitter blocks.
The Transmitter/Receiver Control units accommodate the control path for the framing procedure. Control signals generated by the µP, PHY and the Protocol OAM Control are interpreted and executed by a well-defined finite state machine. These Transmitter/Receiver Control units also relay status information of the packet processors back to the Protocol OAM Controller. The exchange of status information between a µP (host commuter) is carried out via interrupts and a status/control register map for further processing.

A highly efficient and optimised parallel CRC core has been developed. The CRC Control unit co-ordinates and synchronises data being fed into the CRC core. The CRC core computes a 32-bit Frame Check Sequence FCS via an 8 x 32-bit parallel matrix (for the 8-bit P5) or via a 32 x 32-bit parallel matrix (for the 32-bit P5) [3].

Before data is transmitted, the Escape Generate module checks for the presence of a flag character in a frame location in which it is not expected. For each flag character detected, the module inserts an escape character and XORs the flag character with the value 0x20 before transmission. The receiver block carries out the reverse of this Escape operation.

Although, the data path for the 32-bit P5 system is the same as that implemented for the 8-bit system, the control and synchronisation of the 32-bit system is much more complex. To illustrate this, the Escape Generate and the Escape Detect blocks will be examined in more detail.

Considering the Escape Generate block for an 8-bit system, if a flag character was present, the system will halt the input data for 1 clock cycle while simple manipulation takes place and an extra byte is inserted. With the 32-bit version however, a flag character could be present on any of four locations. If a flag is present then an Escape character must be inserted and the flag data XOR’d. This means that instead of the system holding 4 bytes to transmit at this moment, there are suddenly 5 bytes to transfer on a 32-bit channel. Therefore 1 byte must be transmitted on the next clock cycle with the first 3 of the next 4 incoming bytes. Figure 5 demonstrates this problem diagrammatically. Complicated structuring and reorganising is required.

Both, the Transmitter and Receiver blocks are essentially mirror images of each other. They each contain a Control unit, a CRC unit and an Escape Generate/Detect unit. A PPP frame propagates at 32 bits per clock cycle through the transmitter or receiver block and is processed in a pipelined fashion at each processing unit.
If all 4 byte locations consisted of flag characters, however unlikely, then there will be 4 bytes of data awaiting transmission. And say for example on the previous cycle that there was already another byte waiting to be transferred, then the complexity of this problem is easily realised especially considering all the processing and reorganising must be carried out in a single clock cycle.

Figure 6 – Escape Detect Data Organisation

The 32-bit Escape Detect unit must solve similar issues as the 32-bit Escape Generate unit. With the 8-bit version, if an escape character was present, there is no input data for 1 clock cycle as the system deletes the escape character and then XOR’s the next byte of data. With the 32-bit version however, an escape character could be present on any of four locations. If an escape character is present then it must be deleted and the next data byte XOR’d. This means that instead of the system holding 4 bytes to process at this moment, there are suddenly only 3 bytes and there is effectively a bubble appearing on the channel. Therefore 1 byte of the next set of incoming bytes must be inserted into this bubble. Figure 6 explains this problem diagrammatically. Again pattern reorganising and complex restructuring is required.

This problem has been solved by devising a data reordering mechanism and by further pipelining the unit. In the 8-bit implementation, this process typically takes 1 clock cycle. However for the 32-bit system, the process is divided up into 4 pipelined stages with buffering and decisional mechanisms implemented. The first data transmitted is therefore delayed by 4 clock cycles, approximately 50ns. Subsequent data flow is continuous and efficient.

4. Synthesis and Circuit Study

The P^5 was synthesised and targeted to Xilinx Virtex, and Virtex II devices using Synplicity and Xilinx Foundation tools. Speed and area performance is examined. The pre-layout and post-layout synthesis results are included in Tables 1, 2 and 3.

One of the first points to note is that the 32-bit version of the system is not 4 times bigger than the 8-bit version as one might predict, but is approximately 11 times bigger. This is mainly due to the fact that the 32-bit system contains byte sorter mechanisms built with large decision-making combinational logic.

<table>
<thead>
<tr>
<th>Table 1 – P^5 8-bit Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8-Bit System</strong></td>
</tr>
<tr>
<td>Pre-layout Synthesis</td>
</tr>
<tr>
<td>XCV50 -4 95.3 MHz</td>
</tr>
<tr>
<td>XCV2V40 -6 128.4 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2 – P^5 32-bit Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>32-Bit System</strong></td>
</tr>
<tr>
<td>Pre-layout Synthesis</td>
</tr>
<tr>
<td>XCV600 -4 73 MHz</td>
</tr>
<tr>
<td>XCV2V1000 -6 125.9 MHz</td>
</tr>
</tbody>
</table>

To confirm this point, extra synthesis testing was carried out. The 32-bit Escape Generate module and the 8-bit Escape Generate module were synthesised to a Xilinx XC2V40 FPGA separately. The synthesis results from this test are shown in Table 3. The 32-bit Escape Generate requires 492 LUTs (96% available) and 168 flip-flops (32% of the availability). Most of the combinational logic available on this device is utilised, however less than one third of the available flip-flops are utilised. The 8-bit Escape Generate module requires only 22 LUTs and 6 flip-flops. The 32-bit module requires 25 times more combinational logic and 28 times as many flip-flops as the 8-bit module.

Although the complete 32-bit system is 11 times larger, this size increase is mainly due to the decisional logic required in the Escape modules and partly due to extra decisional logic involved in the CRC and data reordering mechanisms.

<table>
<thead>
<tr>
<th>Table 3 – Escape Generator Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Escape Generate Module</strong></td>
</tr>
<tr>
<td>32-Bit Implementation</td>
</tr>
<tr>
<td>XCV40 -6 492 LUTs (96%)</td>
</tr>
</tbody>
</table>
critical path is the same for each device and in each case passes through 6 LUTs. The delay at each LUT is slightly greater with Virtex technology. This indicates that this speed-up is not achieved by a more efficient placement and routing process but to the technological advantage Virtex II offers over Virtex.

5. Conclusions

In this paper the implementation of a 32-bit P^5 system on an FPGA has been presented. Making use of a 32-bit bus, the system had to operate at a frequency of at least 78.125 MHz. It is imperative that at this speed the system is able to process 32 bits every clock cycle. A parallel pipelined PPP architecture has been developed.

The 8-bit and 32-bit systems' speed requirements are met with Virtex II technology. The circuit requires approximately 25% of the resources of a XC2V-1000 device leaving more than sufficient room to incorporate a Xilinx Microblaze microprocessor core or any other general purpose embedded microcontroller, enabling system programmability.

The 32-bit P^5 is approximately 11 times larger than the 8-bit system. It has been discovered that this size increase is mainly due to the byte sorter and buffering mechanisms included in the 32-bit design, which are heavy in combinational logic.

6. References


