Design of Network Security Devices with Hardware Compilation and FPGA Development System --- Educational Implications

K. C. S. Cheng and M. Fleury
Dept. of Electronic Systems Engineering, University of Essex
{kcsche;fleum}@essex.ac.uk

1. Introduction

Embedded network security devices are on the increase as purely software approaches cannot cope with existing packet throughputs, let alone high-performance LANs such as Gb Ethernet. For example, Sourcefire Inc., well known for Snort --- a rule-based software search engine, have turned to an intrusion detection system (IDS) using up to ten G5 PowerPC processors aimed at fibre-optic line-rates of 2--8 Gbps. SourceFire prefer not to use an ASIC, because these are not adaptable to new exploits. In this paper, we consider designs based on a look-up-table-based (SRAM) FPGA, which is reconfigurable but supports greater throughput than a RISC, as it does not suffer from the fetch-execute bottleneck and can process multiple parallel streams according to need. As a comparison, the AES encryption algorithm runs with throughput at 1.5 Gbps on a Pentium 4 (3.2 GHz), at 12.2 Gbps on a Virtex XCV1000 FPGA (max. clock speed 131 MHz), and 25.6 Gbps on an Amphion ASIC at 200 MHz clock speed.

By way of illustration, the design of a template structure for packet scanning is described in the paper. Each hardware thread/process runs in parallel within the structure and can be replaced (reconfigured) when a new threat becomes imminent. The structure resolves access to a shared buffer through reader and writer processes. The design has been accomplished by means of hardware compilation, which, for security applications, has advantages over traditional hardware description languages (HDLs). A hardware compiler converts a program or more accurately an algorithm directly into hardware. Hence, a hardware compiler exists at a higher level of abstraction than an HDL or a silicon complier, allowing faster production of attack detection routines at a cost in control of the form of the output circuit. Since platform-FPGAs have become available from the two main manufacturers, Xilinx and Altera, gate (or rather slice) usage has become less critical, and certainly is not an issue for packet scanning routines. Ten processes took up less than 11% of a Virtex-II XC2V1000 in the packet scanning application.

We used the Celoxica DK IDE, built around the Handel-C hardware compiler. DK has the "look-and-feel" of MS Visual Studio. It incorporates a clock-accurate simulator. In DK, the user can choose to run simulation without any hardware, or generate the output in standard Electronic Design Interchange Format (EDIF) netlist or RTL VHDL. The ability to automatically re-time designs (introduce registers to decouple logic thus optimising timing) is a significant improvement supported in DK version 3. In some cases, timed logic can replace Handel-C channels. The main feature of DK utilised in the packet scanner design are the device driver libraries. The Platform Developer's Kit (PDK) consists of three elements: the Data Stream Manager (DSM), the Platform Abstraction Layer (PAL) and the Platform Support Library (PSL). Only the latter two are used in the design. PAL is a thin wrapper layer around PSL, and hence we found no speed advantage from using PSL directly. PSL is a device specific layer, which is extensible. The RC200 development board, also from Celoxica, was used to develop the structure. The principal features employed in the design were: Xilinx XC2V1000-4 Virtex-II FPGA; 2 banks of ZBT SRAM providing a total of 4 MB; a CPLD for configuration/reconfiguration; an Ethernet MAC/PHY with 10/100 base-T socket; parallel port for bit-file download; and RS-232 serial port.

Of the many different areas in network security suitable for a hardware solution, Internet Protocol (IP) (the main packet routing protocol) fragmentation was chosen as an example for these reasons: it is a network layer issue, which is simpler to tackle than complex transport protocols; the next generation IP version 6 (IPv6) also supports IP fragmentation; and around 0.5% of the total traffic are fragmented packets. Fragmented packets can form a denial-of-service attack as a PC can be induced to spend all its time waiting for a fragment that never arrives. Alternatively, a fragment can pass through a firewall in a way denied to a fully formed packet.

Handel-C, DK and the RC200/300 series of development boards are also well suited to educational usage, as they bring both a simplified and integrated environment. The range of external connections especially the Ethernet and VGA units allow experiments to be constructed. Recently Celoxica have produced the low-cost (approx. £130) RC10 board, with Spartan FPGA but otherwise similar to the network scanner's RC200, suitable for teaching digital design for year one undergraduates. The main weakness appears to be a lack of direct network connection, though there are many other interfaces including USB. We are considering its use for our existing course at Essex, already based on Handel-C. This course was presented at an earlier EEUG workshop and the first author, who implemented the courseware, is able to relate his experience of this course. Network security applications fit neatly into two Essex degree schemes: BEng in Computer & Network Security and proposed Consumer Electronic Systems degree.

2. Development environment

As outlined in Section 1, we seek a design environment that can enable a rapid response to new threats.

2.1 Hardware compilation

Handel-C is a hardware compiler, which attempts to model a programming language in hardware, and outputs a netlist compatible with FPGA place-and-route tools. Software approaches to hardware [25] allow software to be readily ported to hardware or software to be synthesized from existing hardware designs. Based on ANSI-C, Handel-C adds extra features required for hardware development. These include flexible data widths, parallel processing and communication by channels between parallel threads. In Handel-C:

- Within a single clock domain, execution is clock synchronous.
- Assignment statements each require 1 clock cycle.
- Expression evaluation takes zero clock cycles, but results in propagation delay through corresponding combinational logic. Complex expressions will lead to long propagation delay, lowering the maximum clock speed. This will reduce the overall speed of the circuit.

Though the Handel-C model is clock synchronous, the channel primitive allows synchronized communication between parallel processes by means of a rendezvous. The channel allows the designer to neglect detailed timing issues when first preparing a design.

By means of multiple "main" blocks with associated clock statements, Handel-C supports multiple clock domains in the Xilinx Virtex series (from Virtex II). Communication across domains is through a shared buffer, which feature is built in to the design of Section 3. The channel is the only way for processes to communicate between two clock domains. In some designs, the application logic is slower than memory access. Therefore, data are assembled in several cycles while an application completes. The clock speed of the I/O libraries in this design is restricted to 100 MHz and below, while the simpler application logic can run at faster speeds given a decoupled design.

Refinement of Handel-C programs is normally based on trial and error, as, though heuristics exist, there appears to be no direct relationship between making a change and a resulting improvement to the clock rate. As placement takes place automatically, propagation delays are not visible. However, as hardware compilation is software oriented it allows a more direct translation of algorithms into hardware. Given that network threats can arise quickly, this programmatic way of design may allow quicker responses, provided there is a generic structure available.

2.2 Integrated Development Environment (IDE)

The Celoxica DK IDE, built around Handel-C, has the "look-and-feel" of MS Visual Studio. It incorporates a clock-accurate simulator. In DK, the user can choose to run simulation without any hardware, or generate the output in a standard Electronic Design Interchange Format (EDIF) netlist. (RTL VHDL output is also possible.) The ability to automatically re-time designs (introduce registers to decouple logic thus optimising timing) is an interesting improvement supported in DK version 3 (and recently released version 4). In some cases, timed logic can replace channels. We experimented with re-timing in our implementation.

The main feature utilised in the packet scanner design are the device driver libraries. The Platform Developer's Kit (PDK) consists of three elements: the Data Stream Manager (DSM), the Platform Abstraction Layer (PAL) and the Platform Support Library (PSL). Only the latter two are used in the design. PAL is a thin wrapper layer around PSL, and hence we found no speed advantage from using PSL directly. PSL is a device specific layer, which is extensible. Figure 1 shows the relationship between the libraries.

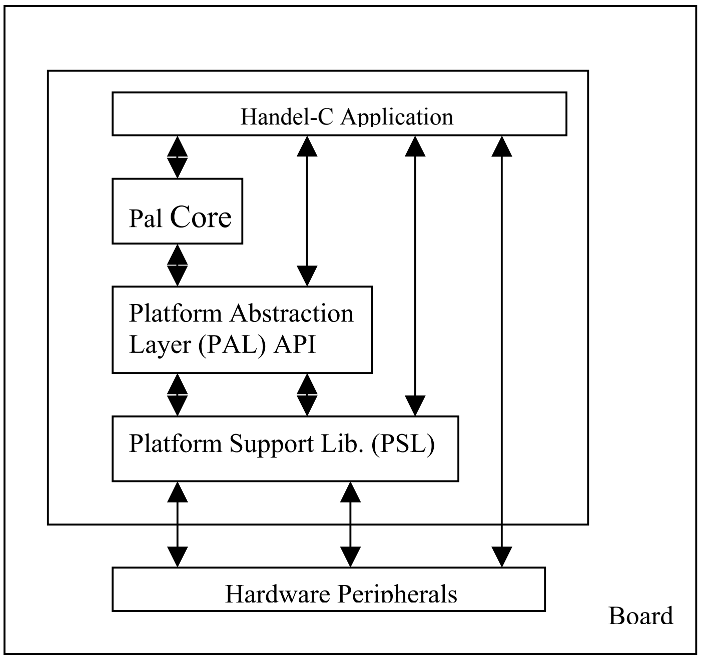
Handel-C Application
Pal Core
Platform Abstraction Layer (PAL) API
Platform Support Lib. (PSL)
Hardware Peripherals
Board

Figure 1: Relationship between application code and device driver libraries.

2.3 Development board

The RC200 development board from Celoxica Ltd. was used to develop the structure. The principal features employed in the design were: Xilinx XC2V1000-4 Virtex-II FPGA; 2 banks of ZBT SRAM providing a total of 4 MB; a CPLD for configuration/reconfiguration; an Ethernet MAC/PHY with 10/100 base-T socket; parallel port for bit-file download; and RS-232 serial port.

The RC300 board, which was released after this work commenced, provides two Gb Ethernet interfaces, making it more suitable for applications of this sort, as the packet stream can be released onto the Ethernet segment, rather than unrealistically outputting to a terminal window on the PC via the RS-232 interface. The Tarari Content Processor, incorporating three Virtex II FPGAs, is custom-designed for packet content scanning. It uses a PCI bus rather than an Ethernet interface and dedicated FPGAs for input and output, allowing high throughput. However, it requires a double-width 64-bit, 66 MHz PCI bus, normally only present on multiprocessor PC servers.

The Virtex II FPGA, if used without modification, may not be ideal for security applications as the Xilinx's Jbits software tool allows selective examination of the reconfiguration bitstream through the JTAG interface. However, Xilinx now provides bitstream encryption, though reconfiguration latency is increased. A triple-DES algorithm is applied, while two keys are stored in a small, battery-powered portion of on-chip memory.

However, readback of the bitstream is not as potent a threat to packet scanners as it is to FPGA cryptographic devices for which keys or possibly algorithms are at risk. The main threat to scanners is probably a commercial one, as an FPGA is a standard part.

3. Fragmentation attack

Of the many different areas in network security suitable for a hardware solution, Internet Protocol (IP) (the main packet routing protocol) fragmentation was chosen as an example for these reasons: it is a network layer issue, which is simpler to tackle than complex transport protocols; the next generation IP version 6 (IPv6) also supports IP fragmentation; and in [1] the authors discovered that around 0.5% of the total traffic are fragmented packets. Although the relative volume of fragmented traffic is not high (though in absolute terms is considerable), it is quite common to have fragmented packets flowing around the networks.

As the maximum transport unit (MTU) can vary across a network path due to buffer sizes or data-link layer protocols, fragmentation allows a packet to be broken up into packets that fit within an MTU. The IP fragmentation mechanism is recursively applied at routers, with packet reassembly normally taking place at the end node. In [2] it is suggested that the disadvantages of IP fragmentation outweigh its advantages, and MTU discovery is an alternative. In [3], the authors offers some alternative remedies, as in some circumstances fragmentation can improve network performance. In security terms, IP fragmentation seems to offer no advantages and only acts as a complication to other packet filtering. As transport-layer protocol headers are only contained in the first fragment (except in the aberration discussed in Section 3.1) packet filters may only process the first fragment and route the rest (assuming that as they cannot be reassembled they will do no harm). Other packet filters cache recent first fragments and the decision applied and re-apply the decision to succeeding fragments. This diversity and complication offers a threat to the successful application of a security policy. However, as fragmentation is widely deployed fragmentation attacks remain a threat.

The basic rules for IP fragmentation are:

1. All fragments must use the identification number of the original packet.
2. Each fragment must specify its offset in the original un-fragmented packet.
3. Each fragment must carry the length of the data carried in the fragment (minimum 8 B).
4. Each fragment must know whether there are more fragments after it.

A router accomplishes rules 1—3 by setting fields in the IP header. Rule 4 is accomplished by setting (or unsetting) a 'more fragments' flag within the flags field. The setting of individual flags is not reported by the `libpcap` library, which on Linux systems underlies some IDS, preventing setting rules to detect this type of exploit.

There are various kinds of IP fragmentation exploits; for a list, refer to [4]. However, many of these exploits first appeared some time ago and most operating systems and firewall software have addressed them with patches and upgrades. However, as fragmentation is widely deployed fragmentation attacks remain a threat, with the Rose attack [5] emerging in early 2004. Furthermore, the choice of IP fragmentation prevention is only as an example to demonstrate the idea of using reconfigurable hardware for network security.

3.1 Tiny Overlapping Fragment Attack

The Tiny Overlapping Fragment Attack is a combination of the "Tiny Fragment Attack" and the "Overlapping Fragment Attack" [6]. The target of these attacks is mainly Internet firewalls and the aim is to bypass firewall filtering. In the Tiny Fragment Attack, the first fragment contains only the first eight bytes of the IP payload. In the case of TCP, this is actually the source and destination port numbers. The rest of the TCP header (most importantly the TCP flags field) will be stored in the second fragment. As a result, the firewall will not be able to test the TCP flags, and a harmful Telnet session could be established.

The Overlapping Fragment Attack exploits flaws in the reassembly algorithms. Two fragments are generated with overlapping offsets. The first one is legitimate, while the second one contains malicious information. Since the firewall only checks the first fragment, the two fragments will arrive at the destination. The first one will, however, be overwritten by the second one during reassemble to produce a malicious packet.

The Tiny Overlapping Fragment Attack is an enhanced version of the two attacks mentioned above. It consists of sending three fragments. However, the attack has a simple countermeasure in the form of two rules catching fragment offsets of zero or one [7], which is easily implemented in hardware. Strangely, this implies that one point where a hardware device would be placed would be to protect a firewall.

3.2 The Rose Attack

The Rose Attack's idea is very simple, the first fragment and the last fragment of a very large packet (64 KB) are sent, but not the middle fragments. The fragment buffer in the IP stack is held open for a certain period of time (That time for Microsoft Windows XP is 1 minute and Debian Linux is 30 s). If there are enough fragment pairs to fill the fragment buffer, no more fragmented packets are accepted. The attack is made effective by sending SYN packets, used to make an initial connection. This fragmentation exploit is one of a number which are denial of service attack, as they work to disable the re-assembler by causing it to reserve too much memory or computation time in the expectation of further fragments that never arrive. As some firewalls reassemble fragments there is again a threat to firewalls.

Alarmingly the author of this attack [5] describes how to set up a number of machines to generate a stream of fragments, spoofing addresses to prevent disablement of the offenders. The following effects of this exploit are stated:

- It causes the CPU to spike, thus exhausting processor resources.
- Legitimate fragmented packets are dropped intermittently.
- The machine under attack no longer accepts legitimate fragmented packets (un-fragmented packets do not experience adverse effects) until the fragmentation 'time exceeded' timers expire.
- Buffer overflow can occur on intermediate routers, *i.e.* packets are dropped at high packet rates if there are not sufficient buffers allocated.

4. Packet Scanner Framework

This aim of the packet scanner structure is to simplify the development of packet inspection processes on an FPGA. It is achieved by the separation of the network interface from the packet operations. The packet scanner has been implemented using Handel-C on the RC200 development board (Xilinx Virtex-II XC2V1000 FPGA). A functional block diagram of the structure is shown in Figure 2.

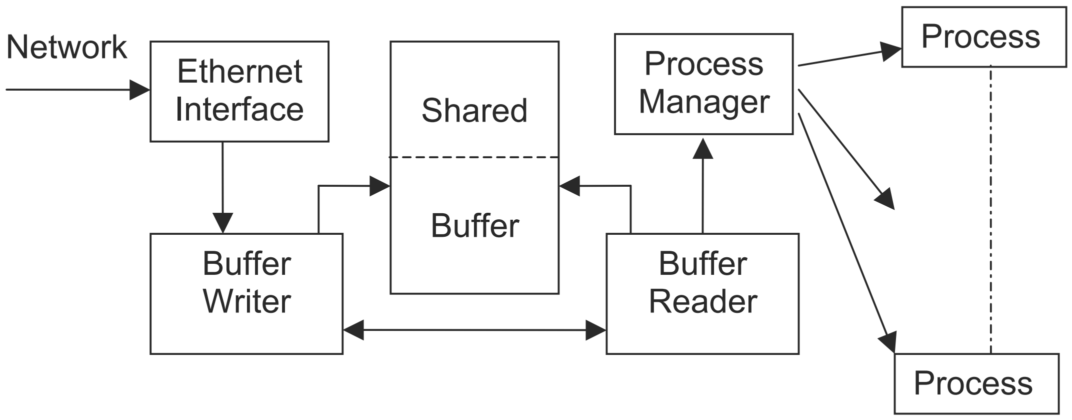

Figure 2: Functional Block Diagram of the Structure

A received packet comes in from the left. The Ethernet Interface is implemented by means of the PSL library (Section 3.2). Packets are read from the network and passed to the "Writer", which then writes the packet (IP header in our example) into the 1 × 256 bit shared buffer.

The shared buffer is implemented using Virtex-II dual-port block RAMs. The following code shows the structure of the shared buffer.

```
// Structure of the Multi-ported RAM
mpram SharedBuffer
{
        rom unsigned 256 Read[1];      // Read Only Port
        wom unsigned 8 Write[32];      // Write Only Port
};

mpram SharedBuffer Queue[MAX_QUEUE_SIZE] with {block = "BlockRAM"};
```

On the other side, the "Reader" reads the IP header from the shared buffer and passes it to the "Process Manager". A Handel-C channel is used together with the buffer to implement a safe FIFO queue. Since the "Reader" needs only one clock cycle to copy the data from the buffer, there will be not any mutual exclusion problem in the design. (*i.e.* the "Writer" will not write into the location the "Reader" is currently reading.) By using a channel, exclusion problems are essentially packaged in the channel construct. The alternative to a channel, Handel-C's semaphore, does not work across different clock domains. The implementation of the semaphore, unlike the channel, described in earlier research papers [40], also appears opaque, perhaps via Handel-C's priority alternation construct or through a priority buffer or through a semaphore management process. Reading a packet is illustrated in the following code. The code illustrates data width control, use of pointers in macro-calls, and the `par` construct, allowing nested parallelism. With the `par` constructs removed porting from 'C' and vice versa is a simple matter. `EthernetRead` is a macro written by us to call PSL library functions. Apart from macros each assignment takes one clock cycle to execute (though this may be in parallel) and all other statements take zero clock cycles. Notice the `delay` statements after each `if`. If omitted these may reduce the clock speed dramatically, as the logic depth increases. (This is reminiscent of the parallel language `occam`, from which Handel-C semantics are derived and which required a skip statement in place of the delay statement in Handel-C.)

```
void ReadPacket()
{
  unsigned 1  Error, Done;
  unsigned 16 Type;
  unsigned 48 Dest, Src;
  unsigned 11 Length;
  unsigned 5  Counter;
  unsigned 8  Data;
  unsigned 8  Temp[20];
  par
  {
     // Attempt to Read Packet
     EthernetReadBegin(&Type, &Dest, &Src, &Length, &Error);
     Counter = 0;
     Done = 0;
  }

   // If read was successful, read the packet data (IP Header)
  if ( Error == 0 )
  {
     // Process Ethernet Type 0x0800 only
     EthernetRead(&Data, &Error);
     if ( Data == 0x08)
     {
        EthernetRead(&Data, &Error);
        if ( Data == 0x00 )
        {
           do
           {
              EthernetRead(&Data, &Error);

              par
              {
                 // Store the 20 Byte IP Header
                 WriteBuffer1(Data, Counter);
                 Done = ( Counter == 19 );
                 Counter++;
              }
           } while ( !Done );
        }
        else
        {
          delay;
        {
     }

     par
     {
        EthernetReadEnd( &Error );
        SignalReader1();
     }
   }
   else
   {
     delay;
    }
}
```

The “Process Manager” controls the different processes working on the header. Here is a code segment of an example process which checks for fragmentation threats:

```
void FragmentProcess()
{
  par
  {
    // Rule 1
    if ( (header.ip_p == 6) && (header.ip_off == 0) && (header.ip_len < 40) )
      Pro1 = 1;                         // Rule 1 is matched
     else
       delay;

    // Rule 2
    if ( (header.ip_p == 6) && (header.ip_off == 1) )
      Pro2 = 1;                         // Rule 2 is matched
     else
       delay;

    // Rule 3
    if ( (header.ip_off != 0) || header.ip_flg )
    {
      par
      {
        Count[sec]++;                   // Update Number of hits in this second
        TotalHit++;                     // Update Total hits
        Pro3 = 1;                        // Rule 3 is matched
        if (TotalHit > MAX_ALLOWED)      // Drop Packet if exceeds limit
          Drop++;
        else
          delay;
      }
    }
else
      delay;
  }
}
```

The code executes in one clock cycle, as the ‘`if`’ statements are all in parallel (and the nested `par`’s statements also all operate in parallel.

The advantages of this design are as follows. Firstly, parallelism can be achieved between processes and within a process, and therefore throughput of the system is increased. Secondly, process design and implementation is independent of the network interface and buffer management, which can be different on every development board. Moreover, thirdly, the use of a shared buffer not only allows pipelined reading from and writing to buffers, but it also provides another option to increase the throughput of the system by dividing the FPGA into several clock domains. The reason for having multiple clock domains is because the network interface circuitry restricts the clock speed of the network interface. However, processes do not have such restriction since they are ‘pure logic’. In the event of dual Ethernet interfaces, three clock domains are appropriate. As previously remarked, on the RC300 board it was necessary to output test results elsewhere, in our case via an RS232 serial link to the PC system. This also requires three clock domains and the revised design shown in Figure 3.

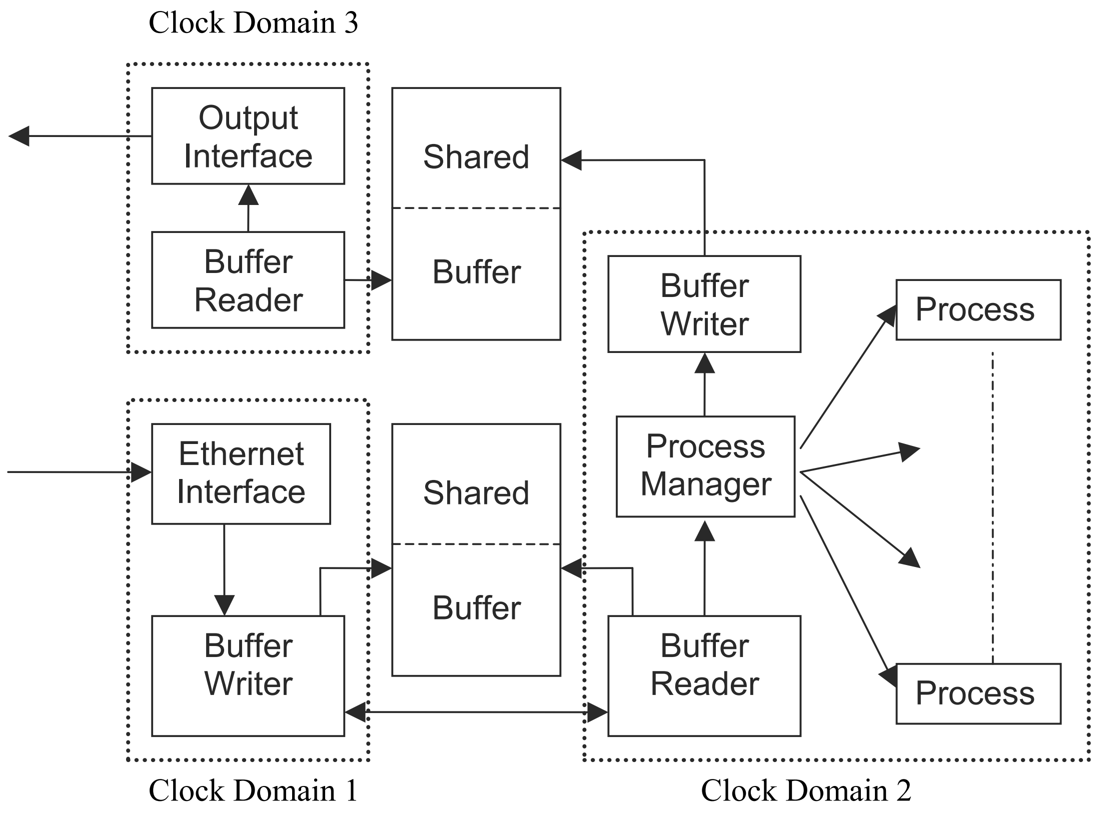

Figure 3: Structure divided into three clock domains

5. Results

In order to test the structure of Figure 2, with a single clock domain, an example of a process was needed. In this case, a single process tackling the IP fragmentation threat was implemented. It checks for certain patterns inside the header and counts the number of occurrences of fragmented packets in a period of time. This process takes one clock cycle to finish its operation.

Since the whole structure is implemented using the Handel-C PSL library, there are some constraints that must be matched. The fastest achievable clock rate for the Ethernet interface library code is 100 MHz. However, the maximum frequency achievable on RC200 is 300 MHz. The "Ethernet Interface" is potentially the bottleneck of the system, not only because there is a restriction of clock frequency but also because the number of clock cycles is also proportional to the size of the packet.
The resource usage of the structure is listed in Table 1. It runs at approximately 50 MHz, as buffering and calling the library code reduces the speed. All the processes used are identical, which is the *FragmentProcess()* of Section 4.

Number of Virtex-II slices		
No Process	1 Process	10 Processes
501 out of 5,120 (9%)	564 out of 5,120 (11%)	581 out of 5,120 (11%)

Table 1: Resource usage of the one clock domain structure

As only 11% of the Virtex-II device was needed for 10 processes, the indication is that it is possible to implement many more processes on the FPGA, which is obviously desirable in order to reduce the cost of deployment.

We also implemented the three-clock domain of Figure 4. To implement multiple clock domains, each domain is assigned a clock and a "main" function. The following code is the top-level control structure for the three domains:

```
// Clock Domain 1
#define RC200_TARGET_CLOCK_RATE 50000000            // 50 MHz
void main( void )
{
    // Run Ethernet in parallel with other code
    par
    {
        // Runs the device management tasks for the Ethernet interface
        EthernetRun( ClockRate1, 0x12345678dead );
        InitBuffer1();        // Initialize Shared Bufer 1

        seq
        {
            // Specifies initialization settings for Ethernet interface.
            EthernetEnable( RC200EthernetModeDefault );
            // Run the ReadPacket macro forever
            while ( 1 )
                ReadPacket();
        }
    }
}
// Clock Domain 2
#define RC200_TARGET_CLOCK_RATE 100000000           // 100 MHz
void main( void )
{
    par
    {
        RealTimeClock();      // Run the Real Time Clock
        InitBuffer2();        // Initialize Shared Buffer 2
        seq
        {
            InitCounter();
            // Run the ReadBuffer1 macro forever
            while ( 1 )
                        ReadBuffer1();
        }
    }
}
// Clock Domain 3
```

```
#define RC200_TARGET_CLOCK_RATE 50000000            // 50 MHz
void main( void )
{
    // Run the RS232 in parallel with other code
    par
    {
        // Run RS232 controller
        RS232Init(RC200RS232_115200Baud, RC200RS232ParityNone,
                                RC200RS232FlowControlNone, ClockRate3);
        seq
        {
            // Run the ReadPacket macro forever
            while ( 1 )
                ReadBuffer2();
        }
    }
}
```

In the implementation, clock domain 1 and 3 easily run at about 50 MHz. However, the 'Buffer Writer' in clock domain 1 takes at least forty clock cycles (number of bytes extracted for the IP header, 2 cycles per-byte) to load the buffer, whereas the 'Buffer Reader' takes one cycle in domain 2. Only one process has been implemented, which has two functions: 1) check for a "Tiny-Overlapping Fragment Attack" (Section 3.1) and 2) Count the number of fragmented packet received in the last 60 seconds. This process takes one cycle to operate (as mentioned in Section 4). The place-and-route algorithm depends on an initial estimate of the clock width (as specified within the Handel-C code) to reach its actual clock width. Automatic retiming (Section 2.2) was found to offer a small reduction in the clock width for clock domain 2, traded-off against a small increase in slice usage. Table 2 presents the results, showing an example of retiming. Note that Table 2 is a snapshot, as optimizations to the code may result in small variations in clock speed. When retiming was applied (after compilation but before output of the final netlist and place-and-route), the clock speed increased from 89.0 MHz to 91.4 MHz. However, DK's technology mapping (at the same stage in processing) also produced an improvement from 79.3 MHz to 89.1 MHz. Technology mapping can be enabled once the specific device is input.

Technology Mapper	**Retiming**	**Virtex –II slices**	**Min. Clock width** *(domain 2 only)*
No	No	692 out of 5,120 (13%)	12.610 ns
Yes	No	698 out of 5,120 (13%)	11.236 ns
Yes	Yes	739 out of 5,120 (14%)	10.936 ns

Table 2: Resource usage of the three-clock domain structure

5. Conclusion

This paper has identified an area, network security, in which embedded systems will increasingly be deployed. The paper has also described appropriate design methods intended for rapid development of a design. If a pre-existing structure exists then hardware compilation is a way of quickly creating a process that will respond to a new threat from tainted packets. The structure designed in this paper has three clock domains, de-coupling the network interface from the application logic, and the input from output channels.

References

[1]	C. Shannon, D. Moore, K. C. Claffy, 'Beyond Folklore: Observations on Fragmented Traffic', IEEE/ACM Transactions on Networks, 10(6): 709-720, 2002.
[2]	C. A. Kent, J. C. Mogul, 'Fragmentation Considered Harmful', SIGCOMM '87, vol. 17, No. 5, October 1987.
[3]	P. Chandranmenon and G. Varghese, 'Reconsidering Fragmentation and Reassembly', ACM Symp. On Principles of Distributed Computing, pp. 21-29, 1998.
[4]	J. Anderson, 'An Analysis of Fragmentation Attacks', March 2001, `http://www.ouah.org/fragma.html`
[5]	W. K. Hollis, 'IPv4 fragmentation --> The Rose Attack', 30 March 2004, `http://seclists.org/lists/bugtraq/2004/Mar/0351.html`
[6]	G. Ziemba, D. Reed and P. Traina, 'RFC 1858 – Security Considerations for IP Fragment Filtering', October 1995
[7]	I. Miller, 'Protection Against a Variant of the Tiny Fragment Attack', RFC 3128, June 2001