Reconfigurable Hardware Network Packet Scanning Structure

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

Abstract

Denial-of-service attacks are an increasing problem in today's networks. Embedded security systems are required as purely software defences are unable to cope with packet rates on high-speed networks. Reconfigurable logic is well suited to the changing nature of the threat. This paper introduces a packet-scanning structure that can be applied to a range of development boards and target systems. Multiple clock domains allow the network interface to be de-coupled from the security logic. The design has been accomplished through hardware compilation on a platform FPGA, a method appropriate to quickly matching changing threats. Automatic re-timing offers a way to optimize clock widths. Results show that this is a scalable solution.

1. Introduction

Embedded network security systems 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 [1] 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 a look-up-table-based (SRAM) FPGA [2], 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 at 1.5 Gbps on a Pentium 4 (3.2 GHz) [3], at 12.2 Gbps on a Virtex XCV1000 FPGA [4], and 25.6 Gbps on an Amphion at 200 MHz clock speed [5]. Hardware security devices also bring reduced vulnerability to attacks against the device itself (from software afflictions such as virus, worms, and executable content) and can act as an insulating layer around a PC and its data.

The packet scanner considered herein provides passive defence at a home or campus PC. It could also be deployed as a boundary device, though with the growth in campus and corporate VLANs, active intrusion detection is preferred to firewall protection. Packet scanning also has a role in protection of routers in the network core, for example against Border Gateway Protocol (BGP) exploits. Packet scanning also differs from firewall protection [6] in that it can act against the packet payload and that stateful filters can be implemented, *i.e.* filters that are responsive to patterns of activity over time. Providing a timely response to threats is an important feature of an embedded security system, which is not simply a function of which hardware is applied to the task, but also depends on the ability to write/design the appropriate response. In turn, this depends on the design environment and a packet scanner system structure that will enable easy modification.

Packet scanners work in reactive mode, acting to thwart newly discovered exploits. Therefore, a hardware system should also be able to quickly adjust its response. Reconfigurable hardware in the form of a SRAM FPGA is well suited to this task but only if it is also possible to quickly re-program the array. Fortunately, from the point that a netlist of logic components and their interconnections is available, the process of place-and-route is largely automated,

unless optimised designs are required. There are three higher-level ways to arrive at a netlist: 1) by means of a hardware description language (HDL) [7]; 2) through a silicon compiler [8]; or 3) using a hardware compiler [9]. HDLs have the disadvantage that compilation times for large designs are lengthy [10], slowing down design iterations, though the range of associated tools allows low-level optimisations. A silicon compiler gives a succinct circuit description, often in an existing software language, *e.g.* [11]. A hardware compiler, as used in this design, converts a program or more accurately an algorithm into hardware. Hence, a hardware compiler exists at a higher level of abstraction 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 (refer to Section 6).

Threat response time can also be improved if a pre-existing structure exists into which a scanning routine can be slotted. Device driver libraries have become available, such as Celoxica's PAL/PSL library (refer to Section 3.2), to allow an FPGA to interface to Ethernet and PCI busses, as well as standard computer peripherals such as RS-233, PS-2 and VGA. Network device interfaces are clearly of particular importance for a packet scanner. Beyond that a buffering structure is needed that will allow a set of scanning routines to work in parallel on one packet, while allowing arriving packets to be stored in a custom data-structure. Access to the buffer must be regulated in hardware to prevent over-write. This is a different problem to software concurrency control through software semaphores or monitors, as those solutions assume virtual concurrency simulated by an operating system scheduler. In this paper, we consider the design of a generic buffering structure that will meet the needs of a variety of scanning routines.

Suspicious content is identified in a number of ways. Signatures (unique byte sequences) can be matched against a packet's content (either header or payload) or conversely a hash of successive segments of a packet's content is matched against a database of such hashes. The signatures or hashes can be stored either in external SRAM banks, in block RAM on the FPGA [12] as conventional arrays or as Content-Addressable Memory (CAM). We have examined on-chip FPGA CAMs but a memory hierarchy, combining block RAM and SRAM [13], is also possible. Particularly in denial-of-service attacks, it is possible for an exploit to extend over a number of packets, requiring stateful filter structures such as counters and timers. Again, these can be pre-designed components of a generic structure. There are a variety of evaluation and development boards, such as the RC200/300 range from Celoxica Ltd, Tarari's content processor, Xilinx's ML300, Digilent Inc.'s XUP V2P, along with production boards [14], which implies that a structure that will also extend to future boards should be sought, not least because it will allow exchange of blocking routines or filters.

The remainder of this paper is organized as follows. Section 2 is a short review of other embedded systems for network security. Section 3 describes our FPGA design environment. We have used hardware compilation rather than a traditional hardware description language (HDL) such as VHDL or Verilog. Hardware compilation may allow a more direct translation from software algorithm description to hardware circuitry, which is more appropriate when a rapid response to a network threat is required. Section 4 is analysis of the type of denial-of-service attacks that embedded network systems are suited to. Section 5 is an analysis of our prototype design and Section 6 gives some preliminary results. Finally, Section 7 draws some conclusions and considers future research.

2. Related work

One category of software packet scanner, Dragon, Bro [15] and Snort [16], rely on exact string matching to locate offending packets. In an early Snort v. 1.6.3 implementation, Boyer-Moore algorithm was applied sequentially to each string, leading to an inefficient search [17].

(In the Boyer-Moore algorithm, a window is passed over the data, and a match is sought by searching backwards within the window.) In fact, some hackers purportedly sent worst-case data to befuddle search engines. Since then there have been improvements [18] including the use of the Aho-Corasick algorithm to search simultaneously for multiple strings. However, as tests confirm [19], most software IDS are insufficient at Gb Ethernet rates. In fact, Snort was originally developed for "small, lightly-utilized networks" [16], but unlike some other commercial products appears in open source form.

There are at least five approaches to hardware string matching [20]: 1) String matching as in the software approach; 2) non-deterministic finite automata (NFA); 3) comparators; 4) hashing, which involves approximate matching and, hence, can lead to false positives; 5) combinations of the others. The following designs were all implemented on Virtex FPGA. In [21] using approach 1), the well-known Knuth-Morris-Pratt (KMP) algorithm is employed to search a character at a time, matching by comparators. The main advantage of the KMP is said to be the ability to scale more readily, *i.e.* apply multiple rules by repeated application of the KMP, *i.e.* through pipelining. Approach 2) is applied in [22] to regular expressions. In [23], approach 3 is applied. Characters from a single packet are fanned out to a set of comparators, which is similar to the operation of a CAM. a CAM. In [12], approach 4) is applied. A Bloom filter is employed to match multiple strings at the same time. For each string, multiple hash functions are applied, each function outputting one value from a finite set of values. The resulting bit pattern vector acts as the search string. Bloom filter matching only identifies candidate threats, implying that a further exact match is required if false matches are to be avoided. In [12], multiple Bloom filters are applied in parallel to the same packet. This implies that variable length IP packets must be buffered. Thus, higher memory usage is a disadvantage of this approach. A similar comment applies to the packet-wise parallelism applied in [23], in which a dispatcher places an arriving packet in one of four implemented content-scanners (regular expressions). On the other hand, logic usage grows according to the number of characters in character-based approaches, whereas this is not the case for hashing methods.

Choice of search algorithm is to some degree interchangeable, whereas it is less easy to alter the system architecture. In our initial work we have employed a flexible approach, able to adapt to a variety of algorithms and adjust its pace to the packet arrival rate. This implies packet-oriented rather than character-oriented parallelism, as only then can packets be buffered if there is a delay in processing a previous packet. A single packet can be scanned for multiple rules or multiple packets can be processed in parallel for different (or the same) rules.

3. Development environment

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

3.1 Hardware compilation

Handel-C [24] 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 4. 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 [26] 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.

3.2 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 (see Section 3.1) supported in DK version 3. 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) [27]. 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.

3.3 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 [28]; 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 [29], 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 [30] allows selective examination of the reconfiguration bitstream through the JTAG interface. However, Xilinx now provides bitstream encryption [31], 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. An extensive survey of reverse-engineering attacks on FPGAs appears in [32].

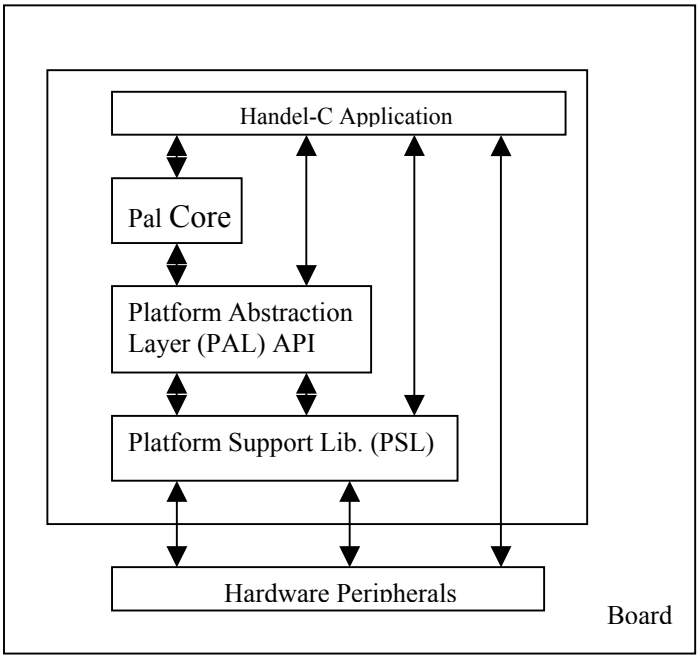

Figure 1: Relationship between application code and device driver libraries (after [27])

4. 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 [33] 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 [34] it is suggested that the disadvantages of IP fragmentation outweigh its advantages, and MTU discovery is an alternative. In [35], 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 4.1) 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 [36]. 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 [37] 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.

4.1 Tiny Overlapping Fragment Attack

The Tiny Overlapping Fragment Attack is a combination of the "Tiny Fragment Attack" and the "Overlapping Fragment Attack" [38]. 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 [39], 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.

4.2 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 [37] 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.

5. 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), as described in Sections 3.1—3.2. A functional block diagram of the structure is shown in Figure 2.

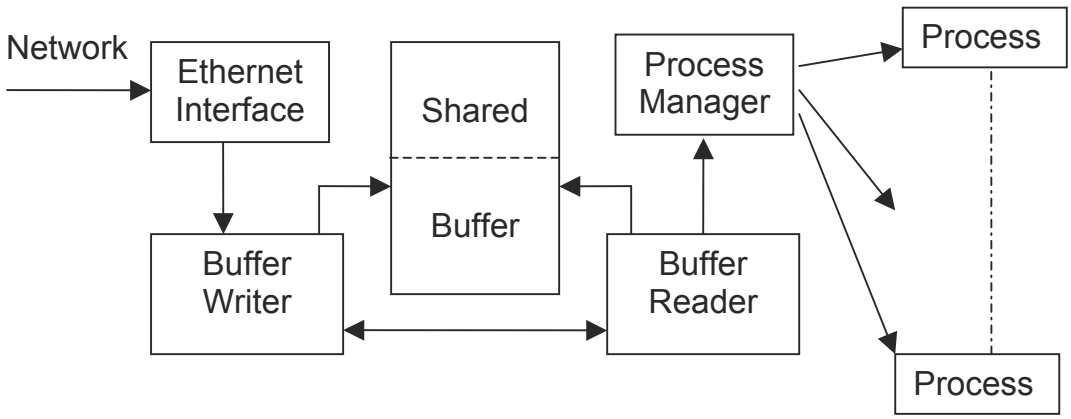

Figure 2: Functional Block Diagram of the Structure

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 2×256 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 (Section 3.1) 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. In [40], a channel is implemented by two handshaking lines, 'ready' and 'transfer'. The 'ready' line is asserted, through an OR gate, whenever any statement is ready to input and similarly the 'transfer' is asserted, through an OR gate, whenever a statement (in another process) is ready to output. The 'ready' and 'transfer' signals are ANDed together to create a 'reg_load' signal, which enables loading of a register to complete the transfer (and resets the channel for communication). The compiler checks that there is only one input and output statement that can communicate over a channel at any one time. Through use of handshaking, a channel is able to communicate across clock domains.

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 );
        }
     }

     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. The code executes in one clock cycle, as the 'if' statements are all in parallel (and the nested par statements also all operate in parallel).

```
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 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 two clock domains. The block diagram is shown in Figure 3. The reason for having two 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'. By splitting the FPGA into two clock domains, throughput is further increased.

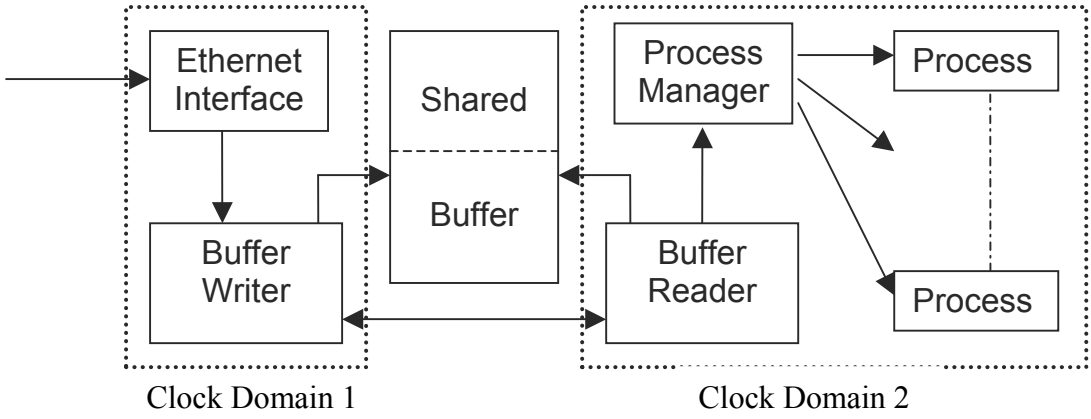

Figure 3: Structure divided into two clock domains

In the event of dual Ethernet interfaces, three clock domains are more 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 4.

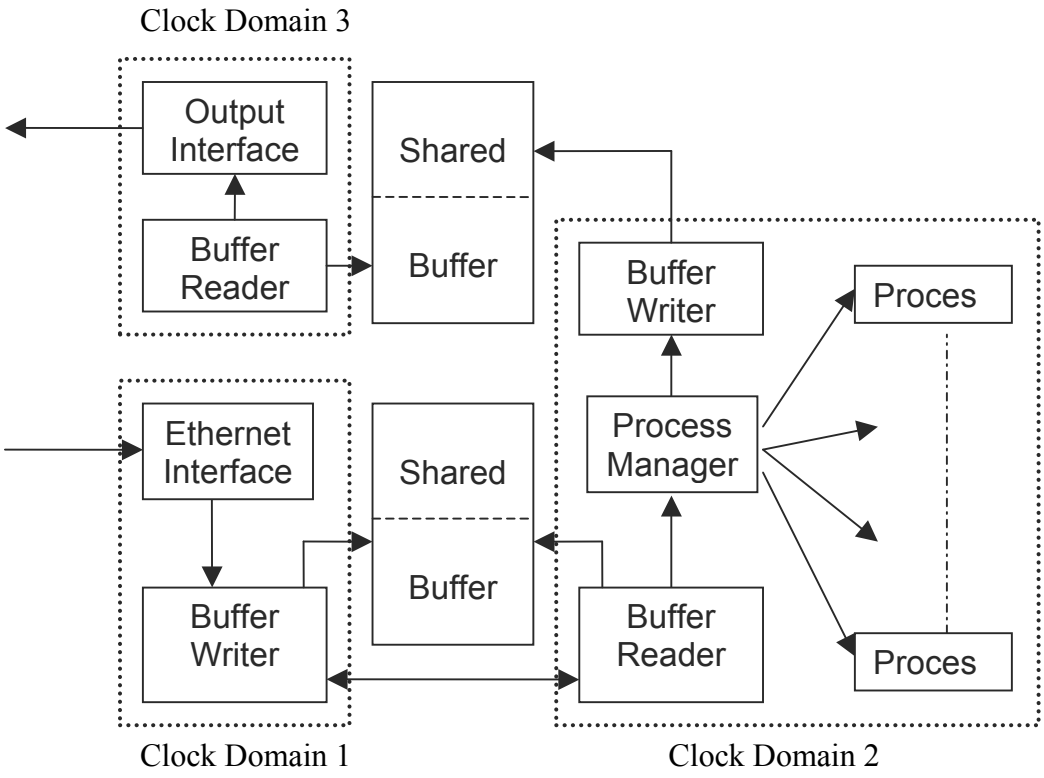

Figure 4: Structure divided into three clock domains

6. 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 5.

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 4.1) 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 5). 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 3.2) 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

6.1 Discussion

Partition of the design into separate clock designs did bring gains in relative clock speed in this implementation, as clock domain 2 now is approximately 100 MHz, whereas the same code ran at 50 MHz in the one clock domain structure. Partitioning is a useful strategy in terms of generic designs with differing I/O interfaces and interface code. Currently, though packet throughput would be around 100 M packet/s with a single process, the input interface reduces this by a considerable amount. Referring to the packet read code in Section 5, there is a minimum of 70 cycles to begin packet processing, while each byte read takes 2 cycles, with a further cycle to store in the shared buffer. Two of the bytes are from the Ethernet frame,

while the remaining 20 transferred to the buffer form the IP header. Finally, to complete takes 7 cycles. In total, this is 141 cycles at 50 MHz or 2,820 ns. All routers support at least 576 byte packets (which is between 42 byte TCP/IP minimal packets and 1500 maximal Ethernet frames). Thus the maximum throughput is 1.634 Gbit/s, assuming a continuous flow of packets, which easily copes with up-and-coming Gb Ethernet Metropolitan network backbones.

7. 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.

The paper represents preliminary results. Currently, the shared buffer has been deployed against packet headers, which are fixed in width. A weakness of shared buffer approaches (see Section 2) in general is that if the data payload is searched then the buffer slots must be of variable width. Dividing the payload into equal-sized chunks may solve this problem but the approach has still to be verified. Clock timings will be the main determinant of relative performance in comparison with the Snort-like software approach described in Section 1. A testing structure has already been constructed so that a stream of packets can be directed towards Snort and towards a packet scanner. Of course, this is on an isolated network. In fact, testing is not simple, as the software access to the raw socket interface is required to modify packet headers. Regulating access to a buffer by limiting the reader to a single clock cycle has a potential to increase the clock width if the reader is augmented in any way. Automatic re-timing offers some help in that respect. Minimizing the application clock domain clock width and the running speed is required for long running processes, so as to match packet arrival times. Hence, design of embedded systems for network security is a challenging task, which brings with it a range of new problems.

References

[1] D. Storey, 'Sourcefire 3D Suite Introducing IS5800', presented at Sourcefire Seminar, London , April 2005.

[2] S. D. Brown, R. J. Francis, J. Rose and Z. G. Vranesic, 'Field-Programmable Gate Arrays', Kluwer, Boston, MA, 1992.

[3] H. Lipmaa, 'Fast Software Implementation of AES', 2002, `http://www.tcs.hut.fi/~helger/aes/rijndael.html`

[4] K. Gaj and P. Chodowiec, 'Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard using Field Programmable Gate Arrays', CT-RSA 2001, N. Naccache (ed.), pp. 84-99, LNCS #2020.

[5] Amphion, 'High Performance AES Cores', product description from `http://www.amphion.com/cs5240.html`

[6] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, 'Implementation of a Content-Scanning Module for an Internet Firewall', IEEE Workshop for Custom Computing Machines, 2003

[7] K. C. Chang, Digital Systems Design with VHDL and synthesis, IEEE Computer Society, Piscataway, NJ, 1999

[8] B. M. Pangrle and D. D. Gajski, 'Design Tools for Intelligent Silicon Compilation', IEEE Transactions on Computer-Aided Design, 6(6):1098-1112, 1987

[9] H. Styles and W. Luk: Exploiting Program Branch Probabilities in Hardware Compilation. IEEE Trans. Computers, 53(11): 1408-1419, 2004

[10] J. McCormack *et al.*, 'Implementing the Neon: A 256-bit Graphics Accelerator', IEEE Micro, 19(2):58-69, 1999

[11] P. Hilfinger, 'A High-level Language and Silicon Compiler for Digital Signal Processing',

IEEE Custom Integrated Circuit Conference, pp. 213-216, 1985
[12] S. Dharmapurikar, P. Krishnamruthy, T. S. Sproull, and J. W. Lockwood, 'Deep Packet Inspection using Parallel Bloom Filters', IEEE Micro, 24(1):52-61, 2004
[13] D. V. Schuehler, J. Moscola, and J. W. Lockwood, 'Architecture for a Hardware-based, TCP/IP Content-Processing System', IEEE Micro, 24(1):62-69, 2004
[14] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, 'Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)', ACM Int. Symposium on Field Programmable Gate Arrays, 87-93, 2001
[15] V. Paxson, 'Bro: A System for Detecting Network Intruders in Real-Time, 7th USENIX Security Symposium, 1998
[16] M. Roesch, 'Snort---A Lightweight Intrusion Detection for Networks', 13th Systems Administration Conference, (LISA 99), pp. 229-238, 1999
[17] C. J. Coit, S. Staniford, and J. McAlerney, 'Towards Faster String Matching for Intrusion Detection', 2nd DARPA Information Survivability Conference, pp. 367-373, 2001.
[18] M. Norton and D. Roelker, 'SNORT 2.0: High Performance Multi-rule Inspection Engine', White Paper, Sourcefier Inc., Columbia, MD, April, 2004.
[19] B. Yocum, R. Birdsall, and D. Poletti-Metzel, 'Gigabit Intrusion Detection Systems', Network World, 4 Nov. 2002, http://www.nwfusion.com/reviews/2002/1104rev.html
[20] G. Navarro and M. Raffinot, 'Flexible Pattern Matching in Strings', Cambridge University Press, Cambridge, UK, 2002
[21] Z. K. Baker and V. K. Prasanna, 'Time and Area Efficient Pattern Matching on FPGAs', ACM Int. Symposium on Field Programmable Gate Arrays, pp. 223-252, 2003
[22] R. Sidhu and V. K. Prasanna, 'Fast Regular Expression Matching using FPGAs', IEEE Symposium on Field-Programmable Custom Computing Machines, 2001
[23] I. Soannis and D. Pnevmatikatos, 'Fast, Large-Scale String Match for a 10 Gbps FPGA-based Network Intrusion Detection System, 13th Int. Conf. On Field Programmable Logic and Applications, 2003.
[24] M. Bowen, 'Handel-C Language Reference Manual', Celoxica Ltd, Abingdon, UK, 1998
[25] M. Fleury, R. P. Self, and A. C. Downton, 'Hardware Compilation for Software Engineers; an ATM Example', IEE Proc. on Software, 148(1):31-42, 2001
[26] M. Fleury, R. P. Self, and A. C. Downton, 'Multi-spectral Image Processing on a Platform FPGA Engine', Military and Aeronautics Programmable Device (MAPLD), 2005.
[27] PAL (Platform Abstraction Layer User) Manual, Celoxica Ltd., Abingdon, UK, 2002
[28] Virtex-II Platform FPGA Handbook, Xilinx Inc., San Jose, CA, 2001
[29] Tarari . 1098 Technology Place, San Diego, CA, 'Accelerating Content Processing', White paper, 14 pages, 2002
[30] S. A. Guccione, D. Levi, and P. Sundarajan, 'Jbits: A Java-based Interface to FPGA Hardware', 2nd MAPLD Conf. 1999, http://www.io.com/~guccione/Papers/Papers.html
[31] R. C. Pang *et al.*, 'Nonvolatile/Battery-backed Key in PLD', US Patent No. 6366117
[32] T. Wollinger, J. Guajardo and C Paar, 'Security on FPGAs: State-of-the-Art Implementations and Attacks', ACM Transactions on Embedded Computing Systems, 3(3):534-574, 2004
[33] C. Shannon, D. Moore, K. C. Claffy, 'Beyond Folklore: Observations on Fragmented Traffic', IEEE/ACM Transactions on Networks, 10(6): 709-720, 2002.
[34] C. A. Kent, J. C. Mogul, 'Fragmentation Considered Harmful', SIGCOMM '87, 17(5) 1987
[35] P. Chandranmenon and G. Varghese, 'Reconsidering Fragmentation and Reassembly', ACM Symp. On Principles of Distributed Computing, pp. 21-29, 1998
[36] J. Anderson, 'An Analysis of Fragmentation Attacks', March 2001, http://www.ouah.org/fragma.html
[37] W. K. Hollis, 'IPv4 fragmentation --> The Rose Attack', 30 March 2004, http://seclists.org/lists/bugtraq/2004/Mar/0351.html
[38] G. Ziemba, D. Reed and P. Traina, 'RFC 1858 – Security Considerations for IP Fragment Filtering', October 1995
[39] I. Miller, 'Protection Against a Variant of the Tiny Fragment Attack', RFC 3128, June 2001
[40] I. Page, 'Constructing Hardware-Software from a Single Description', Journal of VLSI Signal Processing, 12:87-107, 1996