Handel-C Design of a Packet Processing Device for platform-FPGAs

Reza Nejabati and Martin Fleury
Department of Electronic Systems Engineering,
University of Essex, Colchester, CO3 4SQ, U.K.
Tel.: +44 1206 872904
Fax: +44 1206 872900
E-mail: {rnejab,fleum}@essex.ac.uk

Abstract:
Handel-C is a convenient tool for high-level design for FPGAs. Platform FPGAs, with high-levels of integration are suitable as packet processing devices. An existing ATM packet-switching design has been augmented with a CAM, with a simplified control structure, by the addition of a back-pressure mechanism, and a new shared-memory buffering scheme. The paper gives details of the Handel-C implementation, which demonstrates competitive switching performance for aVirtex FPGA.
Keywords: Handel-C, FPGA, packet processing, ATM

I. Introduction

There are an increasing number of applications for packet processing devices in telecommunications networks. For example, the Tarari Virtex-2 FPGA-based (Field-Programmable Gate Array) layer 3-7 smart switch [1] includes in its applications: encryption, compression, and virus and spam filtering. The Virtex-2 Pro FPGA, which has up to 10 M equivalent system gates, with a range of on-chip microprocessor soft and hard cores, along with the IBM CoreConnect system bus, can now be regarded as supporting System-on-a-Chip (SoC) designs. Rocket I/O serial-bus transceivers provide optical transmission rates. At the SoC level of integration, designs must take a highly modular approach [2], with consequent savings in time-to-market, and hardware component reusability.

This paper presents a Handel-C behavioural level design for an ATM (Asynchronous Transfer Mode) packet-processing device targeted at Virtex class FPGAs [3]. The primary purpose of this device is packet routing and switching, but the same structure could be adapted to smart-switch processing. In particular, header classification is facilitated by the ability of FPGA's to support on-chip Content-Addressable Memory (CAM). Despite a flirtation with 10 Gbps Ethernet by some manufacturers, an ATM-like solution [4] remains a preferred option for telecoms. companies, because of the provision of network management functionality as well as switching control, which

makes for a robust and maintainable network. ATM supports both packet[1] *and* circuit switching.

An earlier `toy' ATM design in Handel-C [5] did not include any control functionality. The Handel-C hardware compiler [6] uses the semantics of the Communicating Sequential Processes (CSP) [7] paradigm. The associated DK design environment from Celoxica Ltd. is employed for construction of the behavioural model. DK includes co-simulation tools, allowing the functionality of the packet processing architecture to be checked. A previous design [8], which the design in this paper builds upon and augments (for example back-pressure is included and a search engine implemented), relied on VHDL (VHSIC Hardware Description Language) and VHDL simulation tools. That design was not specifically targeted at FPGA's, for example not using an on-chip CAM. In its turn, the design in [8] is built apparently upon earlier work by Itoh [9].

Future MAN (Metropolitan Area Network) routers will be required to be packet-processing devices, with the ability to adapt to

---

[1] Notice, that ATM packets are normally called cells, because of their size – 53 bytes, while this paper uses the term `packet' to emphasize the generality of the ideas. An ATM form of routing using labels (MultiProtocol Label Switching or MPLS) but with variable length IP packets is now sought

new tasks such as countering network security threats. In particular, there is a need to upgrade to new routes and route header classifications. Two important necessities for the future packet processors are 1) high speed classification 2) and reconfigureability. Packet processing devices have been implemented by RISC processors or ASIC's (Application Specific Integrated Circuits). Broadly, RISC processors provide flexibility but there are potential problems with the increasing transmission rates of networks. ASIC's can provide high-speed processing but components are sometimes weakly programmable or merely parameterizable. The advantage of FPGA's is the ability to configure parallel and custom processing streams, as well as instruction-level parallelism. With current 0.13 μm feature sizes, clock speeds have increased, and the availability of embedded processors, which are suited to making control decisions, allow all aspects of telecoms. processing to be supported. Encrypted reconfiguration bit streams provide commercial confidentiality.

II. Design environment

Handel-C is a programming language designed for porting and compiling programs into hardware netlists, primarily for FPGA's. It includes a small super-set of `C', extended with constructs for nested parallelism, variable data-widths, and hardware data-types. Unlike CSP, a clock synchronous timing model is adopted. CSP channels are a high-level design element, supporting synchronization and data exchange between parallel branches of a design, without the need to consider low-level timing issues. Software-oriented features such as functions encourage code porting and partitioning, as well as software synthesis. The main weakness of Handel-C is that there is no low-level control of logic placement, which may result in high clock latencies (or logic cell usage). However, via the DK-2 environment, VHDL components can be co-simulated alongside Handel-C components, in the manner of co-design. Handel-C can also act as shorthand for VHDL designs, and indeed we have used it to output RTL (Register Transfer Level) VHDL. The facility of the Handel-C environment, especially for high-level design, was the main reason for its use herein.

III. Packet Processing Architecture

The ATM packet-processing device performs packet parsing and packet forwarding. Figure 1 shows the generic system-level model. Packet parsing is performed with a search engine, while a switch fabric performs packet forwarding. The architecture of the switch fabric is based on a self-routing paradigm [4,8,9], whereby the routing tag guides the route through the switch in distributed fashion. On arrival of each incoming packet, the packet parser adds an internal routing label to the packet header. The search engine, through its Lookup Tables (LUT's), also provides an interface mechanism to the control plane module, so that this module can change routing tasks by updating appropriate LUT's. A parser is dedicated to each input line.
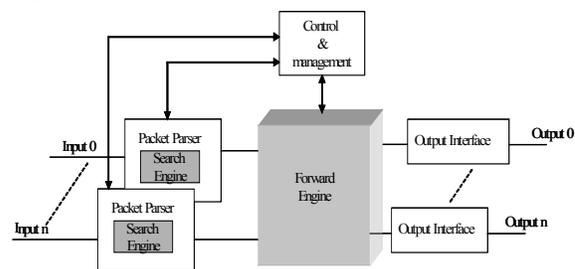

Figure 1: Generic model of the ATM packet processing device.

The switch fabric forwards packets based on the internal routing label to an appropriate output interface, whereupon the internal routing label will be removed from the packet.

IV. Search engine architecture

A routing label to reflect the output destination is attached to each packet by the search engine. The length of the routing label depends on the number of stages in the switch fabric and the parameters supported by the label. In the prototype, a one-byte routing label was designed to support switch fabrics with sizes up to 16×16 switching elements. Figure 2 depicts the contents of the routing label.

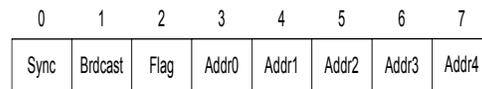| 0	1	2	3	4	5	6	7
Sync	Brdcast	Flag	Addr0	Addr1	Addr2	Addr3	Addr4

Figure 2: Internal routing label

The contents of the routing label are defined as:

- Sync: the arrival of an ATM packet is signalled by a bit one.

- Brdcast: a packet with this bit set is broadcast to all outgoing ports.

- Flag: when set an additional byte with address bits is appended to the label, allowing arbitrarily larger-sized switches.
- Addr0…Addr4: these bits are used to determine: an outgoing link for each packet; and the path taken by a packet within the switch fabric. Figure 3 shows the functionality of the search engine. The routing label and the new header are determined based on the VPI (Virtual Path Indicator) and VCI (Virtual Channel Indicator) fields (hierarchical routing system of ATM) of the incoming packet header.
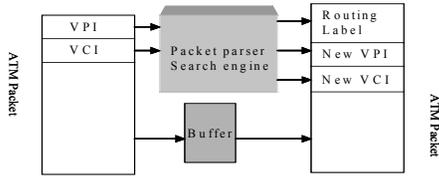

Figure 3: Functional model of the search engine

The search engine makes use of CAM's. CAM's, both binary and ternary, are a hardware approach for search algorithms. A CAM provides constant-time look-up. With a data field as input, CAM returns the address of the matched data as output. One-clock cycle lookups are possible, and fast algorithms exist for updating a CAM's contents [10]. The search engine prototype included one 512×8 CAM. This size mapped onto a single 4096-bit dual-port synchronous block RAM in the Xilinx Virtex family. The design, Figure 4, comprises update and search ports. The control-plane module updates the CAM's contents by means of the update port. To update, the VPI is applied to the CAM as an address and the corresponding routing label is stored as data. The search port is used to allocate an appropriate routing label to each packet. The incoming VPI is applied as input data to the search port and after one clock cycle the matching data will be the corresponding routing label.
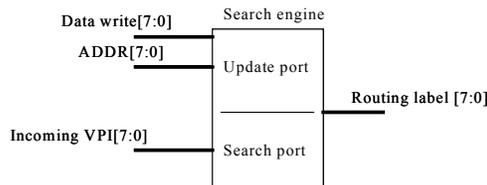

Figure 4: Search engine

Though the prototype allowed only 8 bits of the ATM header VPI field to be stored in each CAM slot, modularity of the design allows both CAM depth and width to be expanded to build a desirable CAM size (*e.g.* to support the VCI field in the ATM packet header or to replace the header of incoming packets with a new header). The width is expandable; without additional logic, a CAM of 16-bit width is constructed by merging two 8-bit width CAM's. The depth of the CAM is expandable with additional tri-state gates (bus-multiplexer). Two 512×8 CAM's allow for 1024 bytes in depth (1024 locations), with 512 locations stored in the first CAM and 512 locations in the second CAM. Figure 5 shows the architecture of a) 512×16 and b) 1024×8 CAM's.
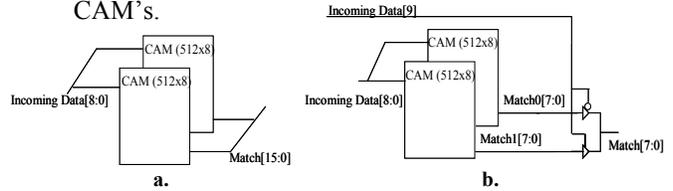

Figure 5: Architecture of a) 16 × 8 and b) 8 × 16 CAM

The following lines of Handel-C code show the implementation of an 512 × 8 CAM structure using Virtex dual-port block RAM's. The code excerpt shows various built-in hardware data-types (*mpram, ram, rom*), and data-widths specified as 8 bits.

```
mpram RAMB4_S8_S8{
        ram unsigned int 8 Write[CAMdepth];
        rom unsigned int 8 Read[CAMdepth];
} CAM with {block=1};
```

The following lines of the code show implementation of the match (search) function:

```
inline void CAM_Search(){
        if (MATCH_ENABLE) MATCH =
CAM.Read[DATA_IN];
        else MATCH=0;
}
```

V. Switch fabric architecture

The switch fabric is a Multi-stage Interconnection Network (MIN) [4] of 2 × 2 Switch Elements (SE). A suitable MIN design offers a compromise between blocking probability and switch complexity, Figure 6 shows the functional block diagrams of a) a 2 × 2 SE and b) for a 4 × 4 MIN. In each SE, a packet is routed to outlet0 if the corresponding address bit is zero or routed to outlet1 if the corresponding address bit is one. By setting

the broadcast bit in the routing label, a packet can be duplicated to both outlets
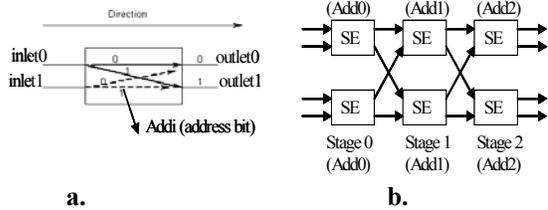

Figure 6: a) SE block diagram b) MIN functional block diagram

The SE architecture is based on a shared-buffer architecture [4,8], which is dominant in the field of commercial packet switches. Based on switch functionality and design requirements (modularity and scalability), a high-level functional model for the SE is demonstrated in Figure 7.a.
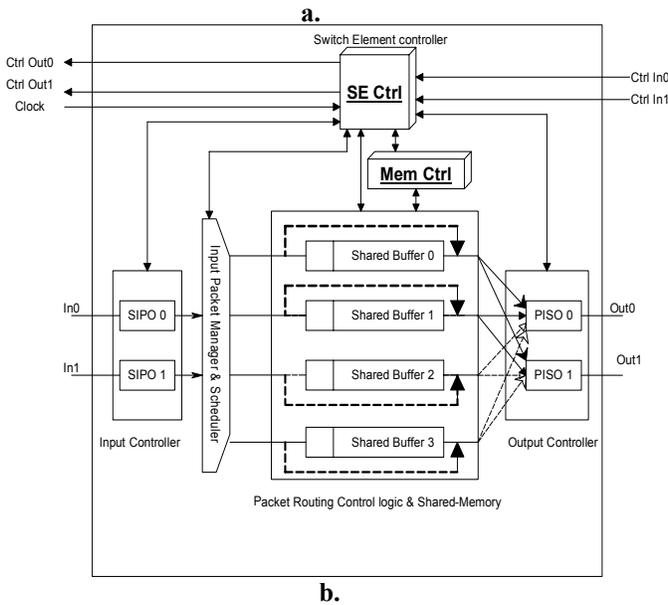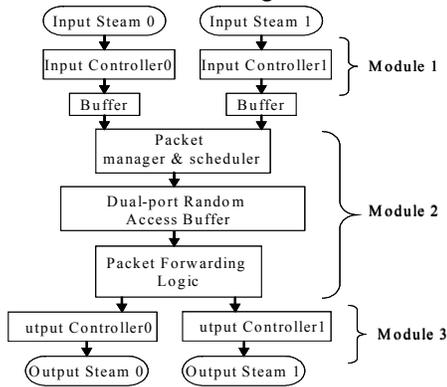

a.

b.

Figure 7: SE a) functional and b) generic architecture

The SE has three main modules, which work in parallel. Shared registers act as an interface between modules. Based on the high-level functional model, generic hardware architecture for the SE has been designed. Figure 7.b shows this architecture, which is a simplification on the design presented in [8], with buffering adapted for FPGA's. All of the SE units in this architecture work concurrently. The SE comprises the following parts:

- Input controller consists of two serial-to-parallel converters (serial input parallel output (SIPO)). Serial-to-parallel conversion is necessary in order to reduce the internal working speed of the SE.

The following lines of Handel-C code show the SIPO implementation. The *par* construct represents each of the statements in the block in time parallel fashion. All statements must complete before further processing can proceed.

```
par{
    in0=link0.inputLink0; //Read one bit from input link0
    in1=link1.inputLink1; //Read one bit from input link1
    dataIn0=(dataIn0 | (0@ in0) )<<(0@!(bitIndex[2] &
bitIndex[1] & bitIndex[0])); // SIPO for link0
    dataIn1=(dataIn1 | (0@ in1) )<<(0@!(bitIndex[2] &
bitIndex[1] & bitIndex[0])); // SIPO for link1
}
```

- Output controller consists of two parallel-to-serial converters (parallel input serial output (PISO)). The output controller converts byte streams from "packet forwarding logic" or "packet manager & scheduler" sections to serial outgoing bit streams.

- Packet Manager & Scheduler unit concurrently scan incoming bytes for a header. On receiving the first byte of a valid packet, it starts to accept the remaining bytes of that packet. Based on the appropriate address bit in the control tag the unit checks the destination (output controller0 and/or output controller1) and the shared-memory state to decide whether: the incoming packet has to be discarded; stored in the shared-memory; or directly routed to the proper output controller.

- Shared-Memory comprises one memory control unit and four shared-buffers each with one packet capacity.

All shared-buffers are dual-port RAM, so that input and output units can access them

concurrently. Dual-port RAM also allow pipelined reading from and writing to buffers. As shared-buffers will give better performance in terms of buffer usage than dedicated buffers, four shared-buffers are used in the buffering technique. The memory control unit controls these buffers. This unit performs two major control tasks. One task is performed during the write cycle and the other is performed during the read cycle. The memory control unit attaches a priority number [4,8] to a packet while it is written to a shared-memory and reads packet bytes from a shared-memory based on its priority number.

The following lines of Handel-C code show an implementation of shared-buffer structure with four dual-ported RAM modules:

```
mpram buffer{
        wom byte Write[shareBufferSize];
        rom byte Read[shareBufferSize];
      }buffer0,buffer1,buffer2,buffer3;
if (ram0_Busy & direction0_1 &
(writeCounter[0]==readCounter1 |
writeCounter[0]==(readCounter1-1)))
par{
        dataOut1=buffer0.Read[outByteCount1];
        outCtrl1=ram0;
        readCounter1++;
}
```

• Packet Routing Control Logic monitors two output controllers and the shared-memory. It directs packets from the shared-buffers to the destination output. It can serve two output controllers simultaneously, if there are packets for them in the shared-memory.

• SE Control: all aforementioned units of the SE communicate with each other through pipelined registers. They can write to and read from these registers simultaneously. The SE controller ensures all units are synchronized and are scheduled efficiently, depending on the routing algorithms.

The SE control in each SE also receives control signals from another SE in a later stage, thus implementing a backpressure strategy [11]. In a backpressure strategy, later SE's in a MIN propagate tokens back to indicate whether they are blocked or not. Based on the state of an SE and the incoming control signals, outgoing control signals are set to the correct values for the next stage (ctrl signals in Figure 7.b).

Handel-C code for the SE control structure is as follows:

```
struct control {
        byte dataIn0,dataIn1;
        byte dataOut0,dataOut1;
        unsigned int 6 byteIndex0;
```

```
        unsigned int 6 byteIndex0;
        unsigned int 6 byteIndex1;
        unsigned int 6 outByteCount0;
        unsigned int 6 outByteCount1;
        bit ramCtrl[4];
        mpram buffer{
                wom byte Write[shareBufferSize];
                rom byte Read[shareBufferSize];
        }buffer0,buffer1,buffer2,buffer3;

}switch_00;
```

VI. On-chip communication and interconnection

The switch fabric design is based on the "Baseline" network. One extra stage is added to a "Baseline" network to provide a redundant path for packets through the switch fabric [8,9]. A 4×4 prototype switch fabric comprises six switch elements. Because of modularity, all six SEs are simply connected to each other. To reduce hardware complexity and increase the internal speed of the switch fabric, the input and output controller SE components in the middle stage are removed. Likewise, the input and output controllers of SE's respectively in the first and last stages are removed. Figure 7 shows this architecture, in which inter-component communication was modeled by Handel-C channels. Interface registers are used for interconnection of SE's. The interface registers can be written into during one stage, while being read by another stage in the same clock cycle.
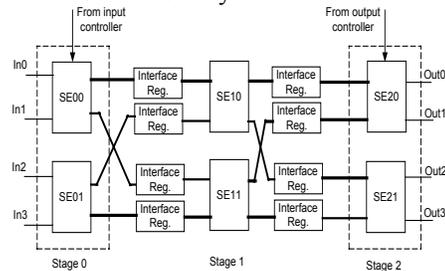

Figure 8: 4×4 Switch Fabric based on 2×2 SE's.

The switch fabric architecture has been implemented in Handel-C. By declaring new switch elements and interface registers an arbitrary switch fabric can be modeled. The following lines of Handel-C code show the declarations for the current switch:

```
struct control {
        byte dataIn0,dataIn1;
        byte dataOut0,dataOut1;
        unsigned int 6 byteIndex0;
        unsigned int 6 byteIndex1;
        unsigned int 6 outByteCount0;
```

```
unsigned int 6 outByteCount1;
bit ramCtrl[4];
mpram buffer{
        wom byte Write[shareBufferSize];
        rom byte Read[shareBufferSize];
    }buffer0,buffer1,buffer2,buffer3;
}SE00,SE01,SE10,SE11,SE20,SE22;
byte SE00To10_0To0,SE00To10_1To0;
byte SE10To20_0To0,SE10To21_1To0;
byte SE01To10_0To1,SE01To11_1To1;
byte SE11To20_0To1,SE11To21_1To1;
```

VII. Results

Packet loss rates of 1×10^9 are specified for ATM. The DK2 co-simulator was used to measure the packet loss probability for a 4×4 switch fabric. Two types of traffic pattern were generated, Table 1, using the on-off model of network traffic.

Traffic pattern	Mean burst length in packets	Mean packet arrival rate
1	3	21 Mbit/s
2	30	1 Mbit/s

Table 1: Simulation traffic patterns

120,000 packets were generated at each input link. To the extent of the simulation, type 1 traffic was shown, Table 2, to be sustainable.

Traffic pattern	Packet loss probability
1	0 %
2	2 %

Table 2: Simulation results

The chip usage was as follows (two Virtex slices to a Configurable Logic Block (CLB)):

Number of Slices: 4,486 out of 12,288, 36%
Total number of 4 input LUTs: 8,183 out of 24,576 33%
Number of Input/Output Blocks: 85 out of 404, 21%
Number of Block RAMs: 2 out of 32, 6%

This indicates that an 8×8 switch is possible on a Virtex with 1 M gates, and that larger switches are possible on the Virtex-2 family. Larger CAM's than the single CAM herein are possible.

For optical transmission, ATM fits within an STM-4 SDH frame at 622.08 Mbit/s. The prototype design clocked at 68.4 MHz, with a potential throughput of 547.2 Mbit/s, of a similar magnitude. Again, the Virtex-2 family has a maximum clock speed of 300 MHz, not the 100 MHz of the earlier Virtex FPGA's.

VIII. Conclusions

This paper has established the feasibility of designing a realistic packet-processing device by means of a hardware compiler. Compared with an HDL, a hardware compiler is more compatible with existing software, allowing future partitioning between programmable and hardware components of an SoC. The design is fully concurrent in a way that is not possible on a RISC processor. Processing is distributed, with control policy determined via LUT's instantiated in CAM's. This form of control is necessary if the design is to remain concurrent. The control policy, as presented, is elementary, but one can envisage more elaborate decisions being made by an embedded processor on the FPGA. Features of this design which distinguish it from previous designs include: an elaboration of the control mechanism via the use of CAM's; introduction of a back-pressure mechanism to regulate cell flow; and a new shared-memory buffering scheme for an FPGA. Work in progress is the control unit of a hybrid electronic/optical switch, based on the synergy respectively between MPLS and GMPLS (Generalized MPLS). Like the approach herein, MPLS places a `shim' label before the header as a network routing mechanism.

References:

[1] Tarari Inc., 10908 Technology. Place, San Diego, CA, http://www.tarari.com.

[2] R. Rajsuman, "System-on-a-Chip Design and Test", Artech, London, 2000.

[3] Virtex data-sheets from Xilinx Inc. San Jose, CA, http://www.xilinx.com.

[4] M. de Prycker, "Asynchronous Transfer Mode Solution for Broadband ISDN", Prentice-Hall, 1995.

[5] M. Fleury, R. P. Self, and A. C. Downton, "Hardware Compilation for Software Engineers: an ATM Example", IEE Proceedings in Software, 148(1):31-42, 2001.

[6] M. Bowen, "Handel-C Language Reference Manual", Celoxica Ltd., 1998.

[7] C. A. R. Hoare, "Communicating Sequential Processes", Prentice-Hall, 1985.

[8] K. Wang and F. Lin, "Design and Implementation of a Fault-tolerant ATM Switch", J. of Info. Science & Engineering., 15:521-541, 1999.

[9] A. Itoh, "A Fault-tolerant Switching Network for B-ISDN", IEEE J. of Selected Areas in Comms., 9(8): 1218-1226, 1991.

[10] M. Peyravian, G. Davis and J. Calvignac, "Search Engine Implications for Network Processor Efficiency", IEEE Network, 17(4):12-20, 2003.

[11] A. Pattavina, Switching Theory, Wiley, Chichester, UK, 1998.