A Programmable Protocol Processor Architecture for High Speed Internet Protocol Processing

Yutai Ma, Axel Jantsch and Hannu Tenhunen
ESD Laboratory, Department of Electronics
Royal Institute of Technology, Sweden
{yutaim, axel, hannu}@ele.kth.se

Abstract- *We present a FSM model for programmable protocol processors. It removes the branch instruction penalty and thus it is suitable for control intensive protocol processing. A case study of high speed IP routing on this model is done, which indicates that two times performance of general purpose superscalar microprocessors can be achieved.*

1 Introduction

Due to the ever-evolving protocols and services used in computer communications, it raises a demand for protocol processors with flexibility so as to make it possible to keep up with the pace of protocol research and development. On the other hand, although general purpose computers have powerful processing capability, because they are not designed particularly for protocol processing the potential high performance can not be utilized efficiently when they are applied to high speed computer communication.

Programmable protocol processors provide an alternative solution to constructing high speed computer communication systems with flexibility by using special architectures which are suitable for protocol processing. The experiment of [2] indicates that at least 3–4 times performance gain can be achieved by using special instructions for protocol processing. [3] developed an architecture model for protocol processing. Their simulation results show that an aggregate gain of approximately 10:1 is obtained compared to general purpose processors. [1, 5] developed a protocol processor for gateways. Their protocol processor gains 12-fold performance of conventional system for transmission and 7-fold performance for reception.

FSM (finite state machine) has been widely used to describe internet protocols. We notice that FSM based protocol processors have many advantages over traditional processors in protocol processing. The main advantages are that the penalty of branch instructions is removed and a branch or jump instruction can be combined with other operations. These provide a possibility for constructing flexible and high performance protocol processing systems.

This paper is devoted to FSM protocol processor architecture with concentration on state memory organization and efficient branch instruction implementation. Although research have been done to compare protocol processors and general-purpose microprocessors, there are no reports about programmable protocol processors and high performance superscalar microprocessors. We use IP routing as an example to investigate application possibility of programmable protocol processors, which indicates that FSM programmable protocol processors can improve the performance of protocol processing by 100% over general-purpose superscalar microprocessors.

2 A FSM Architecture Based on ALU Flags

An obstacle to the performance of SRAM based FSMs is how to generate addresses for different state transitions from a protocol state at a time. We want the address generation to be simple enough so that we can integrate it into address decoding of the state memory. In this section we discuss a new FSM model and an efficient branch target address generation.

2.1 Architecture and Word Format of The State Memory

The word format is shown in Figure 1 and the FSM model is shown in Figure 2. The instruction field controls ALU operations, memory read/write, and I/O operations. The next state field specifies a base address to the state memory for next state memory read operation. We use ALU flags and the control field to capture a state transition or a branch/jump instruction. Since protocol processing is full of control and branch instructions, it deserves combining a branch or jump instruction with other ALU or memory read/write operations. The control field is used to specify which flag join condition evaluation for state transition. The control field works with the branch bits to generate an n-bit address offset for branch target instructions, where n is the bit-width of the branch field. In this illustration, only one bit is used. Therefore, a base address specified by the next state field and a generated branch target address offset form the address to the state memory.

Control Field	1-Bit Branch	Next State	Instructon Field

Figure 1: Word Format of State SRAM.

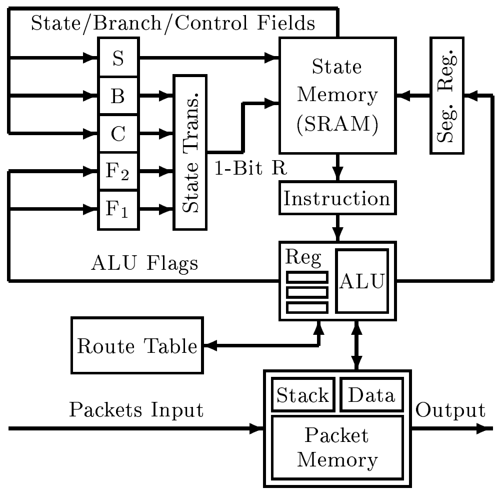

Figure 2: A FSM Model for Programmable Protocol Processors Based on ALU Flags.

2.2 Address Offset Generation of Branch Targets

Condition evaluation and branch instruction (block "State Trans." in Figure 2) is illustrated in Figures 3 and its generated address offset of branch target is shown in Table 1.

When a control bit is set to "1", the current state will transfer to a next state only when the corresponding flag is set up, in this case the branch bit should be set to “1”. If a control field is set to “0”, no condition is required for the corresponding flag. If no control bit is set to “1”, then no condition is required for this state transition. In this case the branch bit can be either “0” or “1”. This is useful for unconditional jump instructions and tasks which cannot be completed in one instruction cycle. This insight is illustrated in Table 1.

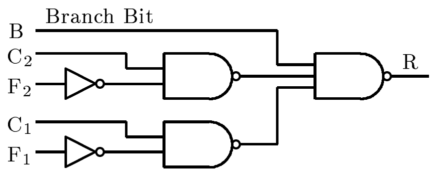

Figure 3: State Transitions Evaluation.

Table 1: Branch Target Address Offset

Control Field	Cond. Eval.	1-Bit Branch	R: Branch Target Address Offset
> 0	True	“1”	“0”
	False	“1”	“1”
$= 0$	–	“0”	“1”
	–	“1”	“0”

It is worth noting that to clear ambiguity among multiple flag pattern matches in a state when the bit-width of address offset is less than that of the control field or flag register (here, the bit-width of the address offset is one), the pattern with more ‘1’s should be matched first. For example, for flag patterns of “11” and “10”, pattern match of “11” should be done first.

2.3 State Memory Capacity

To reduce the state memory complexity, the field length of next state is restricted. For example, with 10-bit representation of the branch bit and next state fields 1k instructions can be accommodated. To accommodate more instructions/programs into the state memory, a virtual memory management unit like the one used on general purpose microprocessors can be used. We can also choose a simplified form by using a segment register to indicate a memory segment as shown in Figure 2. This segment register can be set up by a specific ALU operation. When a program needs to roam into another segment, an instruction is used to update the segment register and a following instruction use its own next state field and the segment register to fetch next instruction in another state memory segment. Since jumping across segments is much less frequent compared to normal operations, this overhead is light.

2.4 An Example

We use a send subroutine of ARQ protocol [4] to illustrate how our model works. To simplify this illustration, we do not focus on these statements implementation but assume that some statements are implemented by one instruction. The send subroutine is described in Figure 4. Assume two flag bits are used to represent the frame type data, nak and ack and they are encoded as “11”, “10” and “01” respectively. A program by using our model is shown in Figure 5, where we assume that this subroutine returns to an instruction in address “011”.

3 Comparison with General Purpose Processors

Protocol processing is full of control operations. Therefore, the execution efficiency of branch and jump instructions has a severe impact on protocol processing performance.

```
s.kind = fk;                /* Instruction 1 */
s.seq = frame_nr;
s.ack = (frame_expected+MAX_SEQ) %
        (MAX_SEQ+1);
stop_ack_timer();           /* End of Instr 1*/
if fk = data then           /* Instruction 2 */
   s.info = buffer[frame_nr % NR_BUFS];
   start_timer(frame_nr % NR_BUFS);
end if;                     /* End of Instr 2 */
if fk = nak then            /* Instruction 3 */
   no_nak = false;
end if;                     /* End of Instr 3 */
to_physical_layer(&s);      /* Instruction 4 */
```

Figure 4: ARQ Send Subroutine [4].

Addr.	Instr.	Control	Branch Bit	Next Sate
000	Instr1	11	1	01
001	Instr2	10	1	10
010	Instr3	00	0	10
101	nop	10	1	10
110	Instr4	00	1	11

Figure 5: A FSM Program for the ARQ Send Subroutine Based on Our FSM Architecture

From Figure 2 and Figure 3 we see that the main advantages of our model over general purpose processors are that the hazard of branch and jump instructions is removed and case statement is supported. On the other hand, in our model branch and jump instructions can be combined with other operations. These advantages lead to a high efficiency of protocol processing.

We focus on branch and jump instructions issue here, however our model also shares other properties of general purpose processors. For example, virtual memory management and interrupt handle mechanisim. This means that our model can provide flexiable and powerful processing capability as general purpose processors while suitable for protocol processing.

4 A Case Study of IP Packet Forward

IP packet forward is to use destination address of an incoming packet to search for the next network address. To make efficient use of internet address space and to attenuate routing entries growth, classless interdomain routing (CIDR) protocol was deployed, by which a longest prefix matching is performed each time. Usually a prefix tree is used to express forwarding tables. The idea is that each prefix is represented by a leaf in a tree structure and prefix value corresponds to a path from root to the leaf of the tree.

Forwarding IP packets has been modeled on our architecture. Compact forwarding tables are used to forward IP packets. In this data structure prefixes are categorized into prefixes with length 1–16, prefixes with length 17–24 and prefixes with length 25–32 groups. With such hierarchical structure, the tables need 68–269 KBytes memory for the real-world IP routing tables.

The model consists of 66 instructions totally. Assume all of the data structures are stored in a secondary cache and a memory access consumes 6 instruction cycles. It takes 15–37 instruction cycles to complete a packet forward action. It is at least two times processing speed of general-purpose superscalar microprocessors. Excluding the memory access expense, it takes only 13 instruction cycles to perform pattern match operations on IP address and routing prefixes. Especially, the binary search in the forwarding operations is performed efficiently on this architecture

It is possible to achieve a wired-speed transmission rate by constructing a parallel processing system. With the penalty-free state transition control, performance degradation due to many compare and branch operations in this application is avoided.

Many efficient IP routing algorithms are emerging every year. Meanwhile, current IP address space is expected to be full after ten years and IPv6 is now on the verge of deployment on the internet. There will be a long time to make IP routing algorithms matured when IPv6 is widely used. Therefore, high performance programmable protocol processors will not only survive but also win definitely a large space on the internet market.

5 Further Works

We will investigate more internet protocols and algorithms and apply them on our architecture, such as multimedia information real-time transfer on the internet, high-speed Ethernet, high-speed ATM network. Attention is focused on architectural support for protocol processing. The profile of the architecture will be more clear with the progress of our work in the future.

We will inspect applications of programmable protocol processors as a standby accelerator for a specific application such as a router/switch, a general-purpose processing system such as a web server and investigate possibility of constructing powerful parallel/pipelined protocol processing systems.

On the other hand, we need to develop an assembler and a compiler to transform high level language programs such as a C language program or a SDL-FSM program into an instruction sequence on this FSM programmable protocol processor architecture.

6 Conclusion

FSM is an effective model for protocol description and protocol processing. In this paper we have proposed an FSM architecture for programmable protocol processors. Compared to general purpose processors, our model removes branch instruction hazard. The case study of forwarding IP packets shows that our programmable protocol processor model is effective for control-intensive internet protocol processing and two times performance is achieved over general purpose superscalar microprocessors. Our FSM model can be used for constructing complex protocol processing systems by using a virtual state memory management unit or by constructing a parallel protocol processor.

References

[1] Tetsuhiko Hirata, Susumu Matsui and Tatsuya Yokoyama, "A high speed protocol processor to boost gateway performance", *Proceedings of Globecom'1990.*

[2] Axel Jantsch, Johnny Oberg and Ahmed Hemani, "Is there a nich for a general protocol processor core?", *Proceedings of Norchip'1998.*

[3] Baiju D. Mandalia, Mohammad Ilyas and Eduardo B. Fernandez, "Performance evaluation of the communications protocol processor", *Proceedings of IEEE ICC'1990.*

[4] Andrew S. Tanenbaum, "*Computer Networks*", pp.216–217, Third Edition, Prentice-Hall, 1996.

[5] Matsuaki Terada, Tatsuya Yokoyama, "A high speed protocol processor to execute OSI", *Proceedings of INFOCOM'91.*