Implementing a Scalable ASC Processor

Hong Wang and Robert A. Walker
Computer Science Department
Kent State University
Kent, OH 44242
{honwang, walker}@cs.kent.edu

Abstract

Previous papers [1,2] have described our implementation of a small prototype processor and control unit for associative computing, called the ASC Procesor. That initial prototype was implemented on an Altera education board using an Altera FLEX 10K FPGA, and was limited to an unrealistic 4 Processing Elements (PEs). This paper describes a more complete implementation — a scalable ASC processor that can scale up to 52 PEs on an Altera APEX 20KE board, or further on larger FPGAs. This paper also proposes extensions to support multiple control units and control parallelism.

1. Introduction

At Kent State University (KSU), a major research focus has been to develop algorithms, software and hardware for associative processing [3] – a form of data parallel processing that accesses memory by content rather than address. Associative processing is particularly suitable for massive data searching and manipulation, as might occur in image processing, data mining, graphics, and relational databases.

We have developed an initial prototype of a processor for associative computing, called the ASC processor [1,2]. Our ASC processor is very loosely based on the STARAN computer developed earlier at Goodyear Aerospace Corporation, though updated with modern RISC concepts, FPGA implementation, etc., and on the ASC model of associative computing developed at Goodyear and at Kent State University.

We are currently working to support not only the KSU ASC model of associative computing, which uses a single Instruction Stream Control Unit, but also the MASC model (Multiple ASC), which supports control parallelism through multiple Instruction Stream Control Units. In the MASC model, each Instruction Stream Control Unit (IS) has its own set of instructions to execute, and each IS can broadcast control signals to all PEs. Initially, all PEs listen to one instruction stream, but can switch to other ISs or be idle depending on the data in the PE or the instruction currently being executed.

This paper briefly introduces our earlier initial prototype of the ASC processor, and then describes how it can be scaled up on larger FPGAs. We then propose further extensions to support control parallelism through multiple Instruction Stream Control Units.

2. Implementing a Prototype ASC Processor

The initial prototype of our ASC processor [1,2] is a byte serial associative processor. The processor has a single *Instruction Stream Control Unit* (variously called the *IS Control Unit*, the *Control Unit,* or simply the *IS*) and an *Associative Processing Array* that contains 4 PEs, along with responder resolution circuitry and MAX/MIN circuitry necessary to support associative computing. The IS Control Unit directs the processing array to perform associative searches, constant time search for a maximum or minimum value in a particular field across all processors, as well as other scalar and parallel arithmetic and logic operations.

The IS Control Unit has its own data memory and 32 bit instruction memory as well as scalar arithmetic and logic operation circuitry. The Control Unit fetches and decodes instructions stored in the instruction memory. Scalar operations are executed by the Control Unit directly; for parallel and associative operations, the Control Unit sends control signals to the four PEs directing them to perform the appropriate operation. To perform an associative search, the search key is stored into the Common Register, which is readable by both the Control Unit and the PEs, and instructions are sent to the PEs telling them to compare the data in the Common Register to data in the PE's General Purpose Registers. Since the ASC processor is byte serial, the Control Unit loops as necessary to process multiple-byte data.

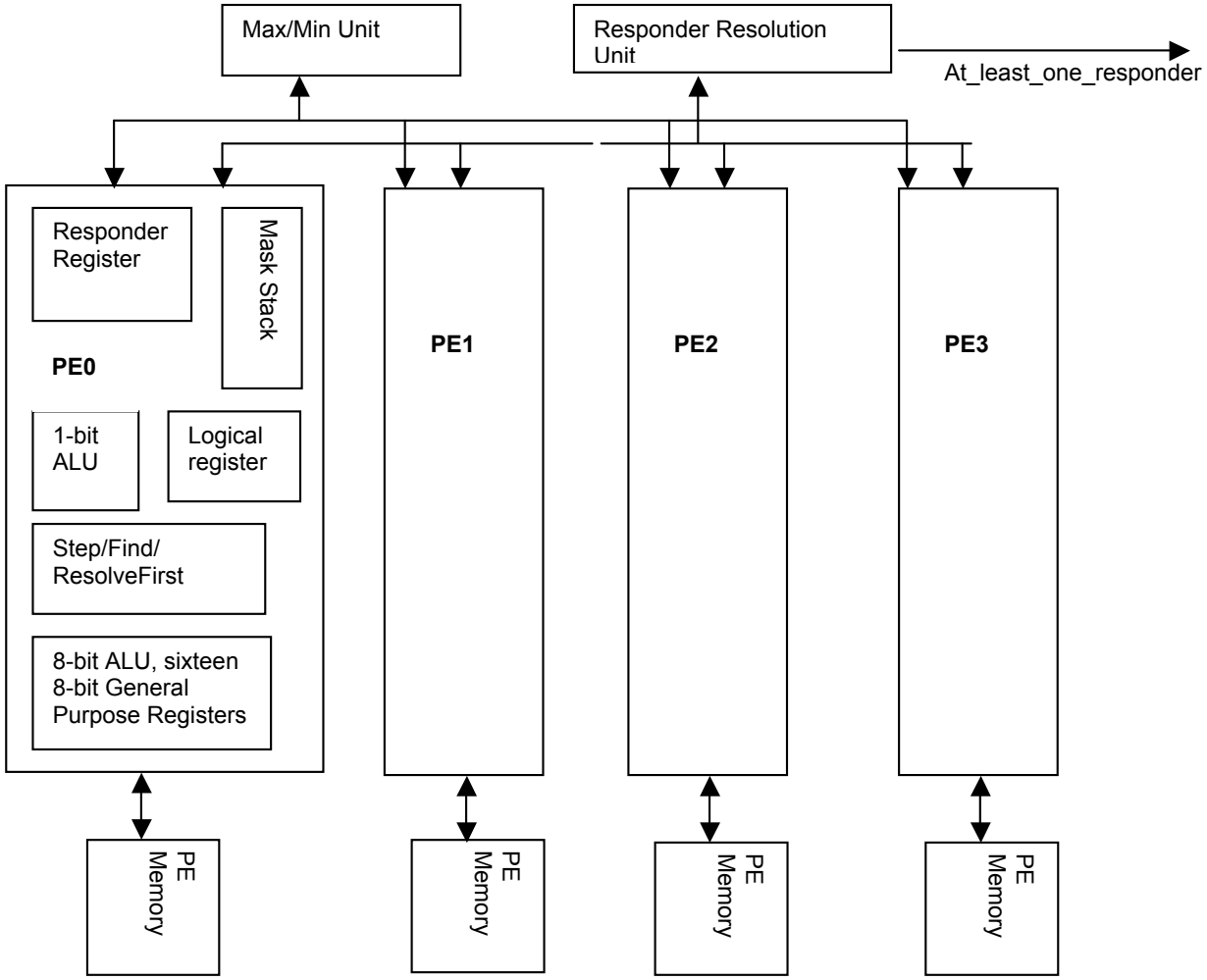

Figure 1 — A 4-PE ASC Processor Array

The Associative Processing Array, shown in Figure 1, is composed of 4 Processing Element (PE) cells, MAX/MIN circuitry and responder resolution circuitry. Each PE cell is composed of a PE and a local data memory. The data memory usually contains data of several variable-size data fields which comes from a record of a tabular format data structure.

Each PE has an 8-bit ALU, a 1-bit ALU, sixteen 8-bit General Purpose Registers, sixteen 1-bit Logical Register, a Responder Register, a Mask Stack and Step/Find/ResolveFirst circuitry. The 8-bit ALU and sixteen 8-bit General Purpose Registers are used for normal arithmetic and logic operation.

The remaining circuitry inside each PE is used to implement associative computing. When an associative search is performed, every PE simultaneously searches a specified General Purpose Register for a particular search key; those PEs that find this key are called responders and are flagged by a '1' in their Responder Register. If this value is then stored in the top of the mask stack, that PE will be selected for further processing by masked instructions, while PEs with a '0' in the top of the mask stack will not be selected. Additional circuitry handles more complex search criteria.

The MIN/MAX Unit is based on Falkof's algorithm [4], and allows the Processing Array to search for a maximum or minimum value across all PEs. For our initial prototype 4-PE ASC processor, four 8-bit Shift Registers are used to process the 8-bit data serially, and four 1-bit Mask Registers are used to indicate the maxium/minimum value. The data in the Shift Registers is processed from most significant bit to least significant bit, ANDing each bit with the Mask Register bit (which is initially 1 for all PEs). The results of this AND are ORed together to indicate whether or not there is at least 1 responder. If the

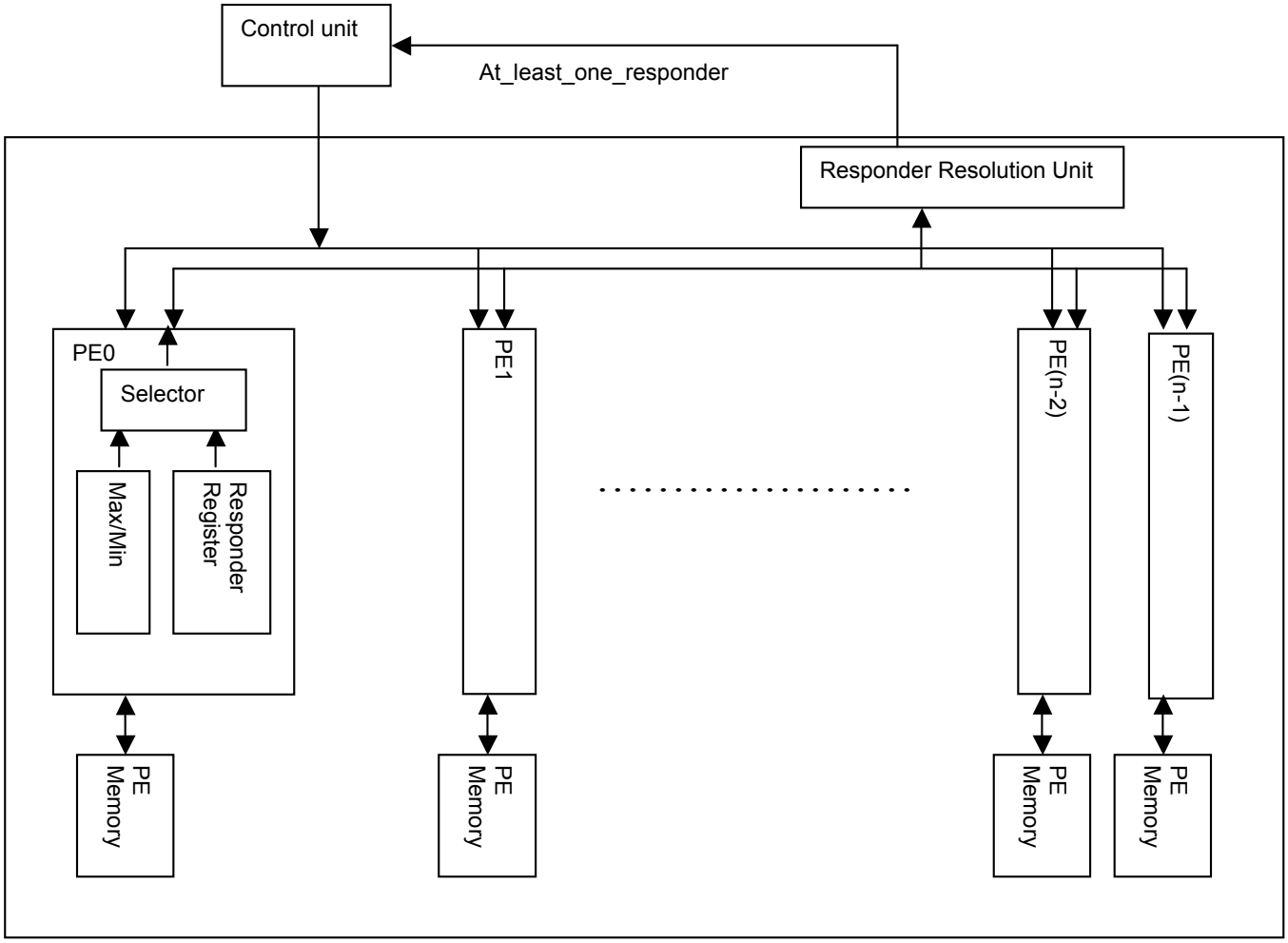

Figure 2 - A Scalable ASC Processor

OR result is '1', the Mask Register is set to the corresponding AND result. Otherwise, there is no responder, and all PEs are tied, so the Mask Register is not updated. The remaining bits are then processed, and when the algorithm terminates all PE(s) that have a '1' in the Mask Register contain the maximum/minimum value.

The Responder Resolution Unit collects the responder bit from each of the 4 PEs. After processing, it sends an At_least_one_responder signal to the Control Unit telling it that PE(s) responded. The Responder Resolution Unit also sends a Responders_before_me signal to each PE so that PEs can be processed sequentially if necessary in the Step/Find/ResolveFirst circuitry.

The initial prototype ASC processor described above has been implemented in an Altera FLEX 10K70 field-programmable gate array (FPGA) as described in [1,2]. Although this prototype has only 4 PEs, the design can be scaled up substantially in bigger FPGAs. Section 3 of this paper will the modifications made to scale the processor up on an Altera APEX 20K1000E chip.

3. Implementing a Scalable ASC model

A FLEX 10K70 FPGA contains only about 70,000 gates, and thus can hold only 1 IS Control Unit and 4 PEs. In order to design a bigger ASC processor, we moved to an APEX 20K1000E FPGA board and updated our Altera design software from Max+PlusII to Quartus2. This APEX 20K1000E device [5] has 1 million gates available, grouped into 160 MegaLAB structures, each MegaLAB containing a group of 24 logic array blocks (LABs), one Embedded System Block (ESB), and a MegaLAB interconnect. Each LAB contains 10 Logic Elements (LE),

and each MegaLAB interconnect routes signals within the MegaLAB structure.

The Embedded System Blocks (ESBs) can implement either logic or memory. When implementing memory, each ESB can be configured in any of the following sizes: 128x16, 256x8, 512x4, 1,024x2, or 2,048x1. ESBs can also be combined to form larger memory blocks — for example, two 128x16 RAM blocks can be combined to form one 128x32 RAM block or one 512x4 RAM block.

On the APEX 20K1000E FPGA, our Control Unit requires 1200 LEs, and each PE requires about 700 LEs. Given the size of the APEX 20K1000E FPGA (38400 LEs), one such FPGA could implement an ASC processor with one Control Unit and up to 52 PEs and support circuitry. Since the Control Unit occupies about 5 MegaLabs, it has 5 ESBs for use as instruction and data memory, while each PE occupies about 3 MegaLABs, giving each of them 3 ESBs as memory (another increase over the limited memory of our prototype ASC processor).

Unfortunately, our initial prototype ASC processor [1,2] is strictly a 4 PE design, whereas a design that more cleanly scales up to more PEs would be desirable. From an algorithmic point of view, some algorithms require a specific number of PE to perform optimally [6]. Later on, when we add more functionality to our PEs, the number of PEs the chip can hold will decrease. Similarly, as we design a multiple Instruction Stream ASC processor, the design will contain more structures, again decreasing the number of PEs that a chip of a specific size can hold. For all these reasons, it is desirable that the associative PE array be easily scalable.

A scalable version of our ASC processor is illustrated in Figure 2. In this design, all PEs listen to a single Instruction Stream bus. Each PE also talks to a central Responder Resolution Unit — PEs send responder signal to this unit, and receive Responders_before_me signals from it.

Unlike the initial prototype of the ASC processor, where the responder circuitry was distributed partially into each PE and partially into the Responder Resolution Unit, that functionality is centralized in the Responder Resolution Unit in this scalable design. The Responder Resolution Unit is responsible for processing the responder signal from the Responder Register in each PE, as well as responder signals from the Maximum/Minimum Unit (described next).

Before going to Responder Resolution Unit from PE, these two sets of signals go through a Selector controlled by the Instruction Stream Control Unit. For example, if the current command is to find a responder among PEs, the Instruction Stream Control Unit will select the responder signal from the Responder Register to send to the Responder Resolution Unit. If the current command is to find a maximum value, the Instruction Stream Control Unit will select the Maximum/Minimum Unit's AND output to send to the Responder Resolution Unit.

The At_least_one_responder signal is sent to both the Instruction Stream Control Unit and to the PE's Maximum/Minimum Unit to find a maximum or minimum value. It is also sent to Step/Find/ResolveFirst circuitry as in the prototype ASC processor.

In contrast, the Maximum/Minimum Unit is distributed across PEs in our new processor design, as illustrated in Figure 2**.** Each PE now has its own dedicated Maximum/Minimum Unit. In the Maximum/Minimum Unit initially designed by Meido Wu [2], there is a shift register, a 1-bit mask register and 1 AND gate. The shift register holds the data from the field to be searched for a maximum or minimum value. A 1 bit mask register indicates whether or not it is extreme value. The AND gate will AND the current bit slice of the shift register with the current mask register bit and output a responder signal, which is sent to a central Responder Resolution Unit. The Responder Resolution Unit then ORs all the responder signals from all the PEs together to find whether or not there is At_leaset_one_responder, and broadcasts this OR output to all the PEs. A PE updates its mask register with the AND output if it receive a '1', or leaves it the same if it receive a '0'. Compared with the previous design, this new design is easier to scale up and saves on design circuitry outside the PE array.

When scaled to larger numbers of PEs, the timing / performance varies. While the maximum frequency is about 33.47 MHZ when there are only 5 PEs, the maximum frequency is 33.63 MHz for 10 PEs and 26.9 MHZ for 50 PEs. For larger FPGAs, it may be necessary to optimize the current design further to avoid too much degradation.

Overall, making these changes to our initial prototype ASC processor to produce this new scalable ASC processor reduced the circuitry required for the Responder Resolution Unit and the Maximum/Minimum Unit. Moreover, the new architecture is particularly supportive of the MASC mode, where each IS Control Unit will need only one Responder Resolution Unit instead of having both its own Responder Resolution Unit and Maximum/Minimum Unit.

4. Implementing Multiple Instruction Streams in the MASC Model

The Kent State Multiple Associative Computing (MASC) model [3] is illustrated in Figure 3**.** As in the ASC model described earlier in this paper, each PE has its own data memory. Multiple instruction streams can broadcast to all PEs in constant time. A PE cell listens to one Instruction Stream Control Unit (IS) at a time, but can

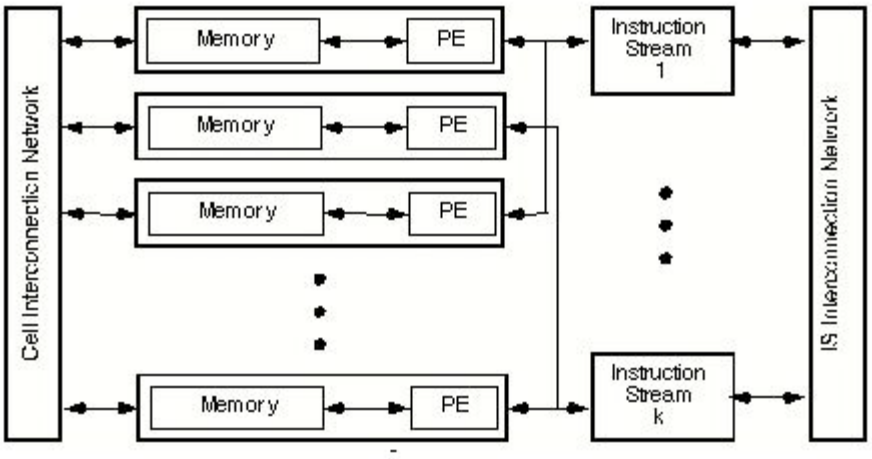

Figure 3 – MASC Architecture

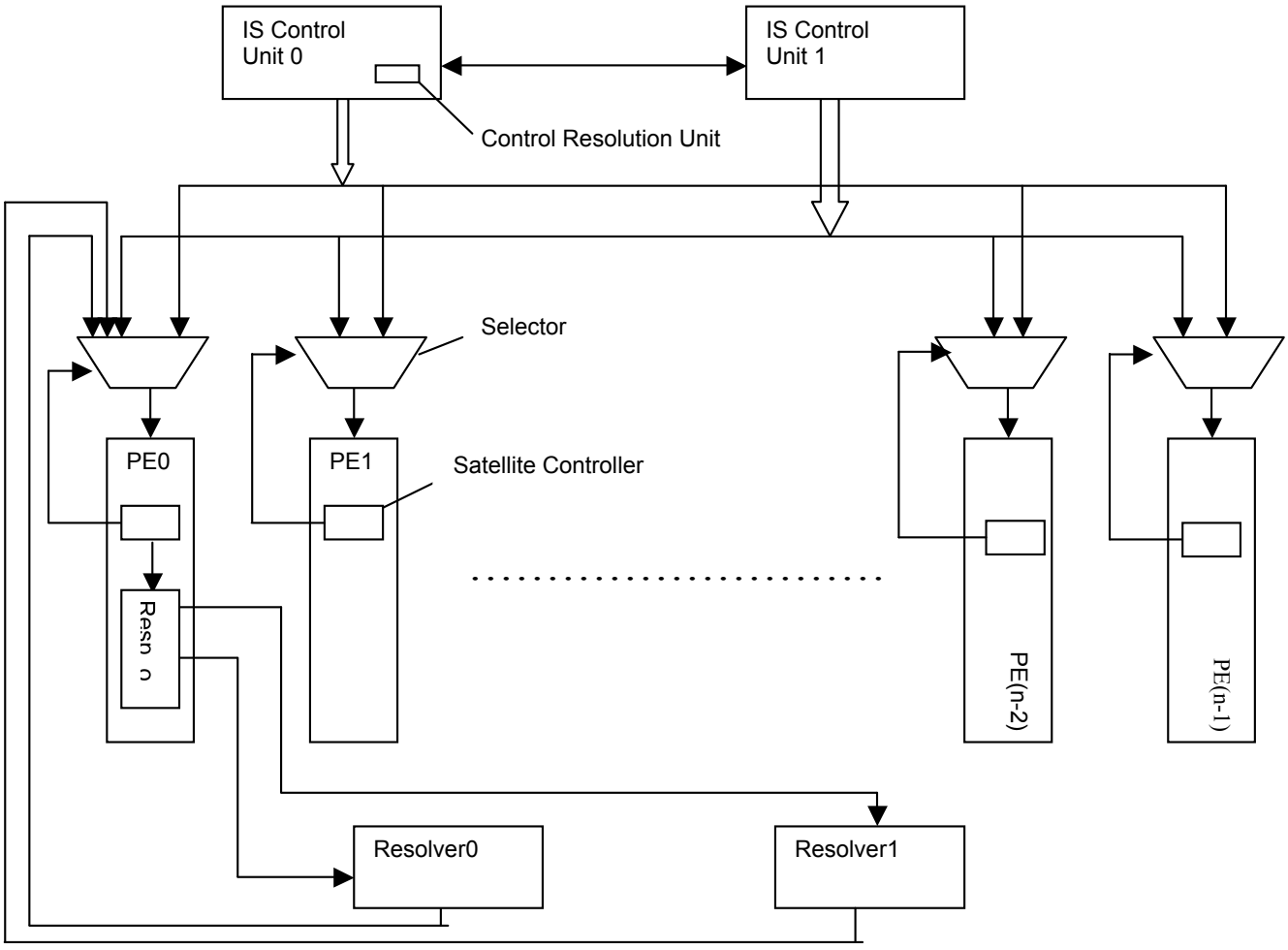

Figure 4 – Implementing the MASC Architecture

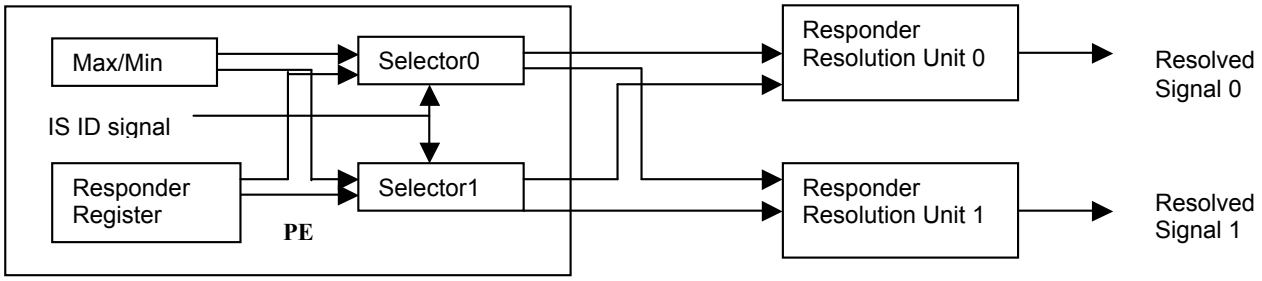

Figure 5 – Responder Resolution Circuitry

be dynamically assigned to any Instruction Stream as necessary. Once assigned to an IS, a PE actively executes instruction from that IS when its Mask bit is '1' and stays idle when the Mask bit is '0'.

We are proposing to adapt our scalable ASC processor to implement the MASC model, as described in this section. To simplify the explanation, we will describe only a two-IS MASC model in this paper (Figure 4), though designs with more ISs are a simple extension. In Figure 4, only the leftmost PE shows how the responder signal from to execute a block of instructions, it sends the starting address in the shared instruction memory to IS1, and IS1 executes that block of instructions, while IS0 executes its own block of instructions at the same time.

To ensure that each PE listens to the correct IS, a Satellite Control Unit is placed in each PE. When IS0 sends SWITCH conditions to a PE's Satellite Control Unit, that Satellite Control Unit will generate an ID signal to select control signals from the correct IS. Similarly, this ID signal is used to control outputs to the Responder Resolution Array and to select the resolved signal from the correct Responder Resolution Unit. For example, suppose a program must execute separate blocks of instructions on two groups of PEs based on the Brand field of the PE's memory. If this field contains TOYOTA, the PE should listen to IS0, otherwise it should listen to IS1.

PE's Satellite Control Unit compares TOYOTA with its local data and generates IS0's ID signal if the comparison is true, or IS1's ID signal if false. This ID signal then lets the control signals entering this PE from the appropriate IS go through.

The Responder Resolution Array is composed of two Responder Resolution Units corresponding to the two IS Control Units. All PEs can send their Responder output to both units, but a PE should only send its output to the Responder Resolution Unit of the IS to which it is currently listening. This selection is done as follows, and is illustrated in Figure 5: a PE sends signals to both Responder Resolution Units separately through two the Responder Register and the Maximum / Minimum Unit is connected with Responder Resolution Unit and how the resolved signal goes back to the PE, but the other PEs are implemented the same way. The Control Unit Array in the two IS MASC model has two Instruction Stream Control Units (ISs). Each IS has its own sequential supporting circuitry, and data memory, but the ISs share an instruction memory. Each IS can broadcast control signals to all the PEs in the Associative Processing Array. Initially all the PEs listen to IS zero (IS0), while no PEs listen to IS one (IS1). If IS0 needs IS1 selectors, each representing an IS Control Unit. Under the control of the Satellite Control Unit's ID signal, the correct selector sends output bits to the corresponding Responder Resolution Unit. Other selectors only send '0', because '0' does not contribute to the output of the Resolution Unit.

To coordinate the IS Control Units, IS0 also contains an IS Resolution Unit. IS1 sends an I_am_done signal with a value of '1' to IS0's IS Resolution Unit when its control parallel block is finished. IS0 also sends an I_am_done bit to the IS Resolution Unit when it finishes. The IS resolution unit ANDs these I_am_done signals together, and if the result is '1' then every control unit is finished so the Satellite Contol Units of both PEs will be set to listen to IS0. The program then starts from the next instruction after the control parallel blocks on IS0.

Continuing the previous example, when processing the data illustrated in Figure 6, suppose the price of all Toyotas must be decreased by $500 and the price of all Hondas must decrease by 5%. To perform this operation in MASC, a program has two separate instruction blocks for IS0 and IS1. First IS0 need to broadcast the conditions to both PE's Satellite Control Unit: Toyota and Honda in BRAND field. After the PEs have chosen the correct IS to listen according to the method described above, IS0 starts executing instruction block 0 to deduct $500 from the original price of the Toyota cars. IS0 also gives IS1 the address of instruction block 1, so that it can begin executing the code to decrease the price of the Honda cars

Table 1 - Additional Machine Instructions for MASC

Instruction	Example	Meaning
Load PE data to register in Satellite Control Unit (SCR)	SCLOAD 0X2F, SCR0	Load content of (0X2F) to SCR0
Broadcast conditions for choosing IS to PEs	PSENDC $CR0, $CR1	Send data in Common Registers to all PE's Satellite Control Unit
Send instruction blocks to ISs	PSENDI 0X5F, 0X6F	Send Instruction Memory address (0X5F) to IS0, (0X6F) to IS1 and enable instruction streams
End of the instruction thread	ENDIS	Send 1 to control resolution unit, Meaning "I am finished"

by 5%. When each PE finishes its price updates, IS0 and IS1 set their I_am_done signal to IS Resolution Unit to '1'. When IS Resolution Unit's output signal becomes 1, IS1 sends instruction to set all PEs that is listening to it to listen to IS0. IS1 then becomes idle and IS0 starts the code after the two control parallel blocks.

As another example, the maximum price for all Hondas can be found at the same time as the maximum price for all Toyotas. As in the previous example, IS0 and IS1 both have their own instructions to execute, those for finding maximum price. PEs listening to either IS broadcast their signal from the Maximum/Minimum Unit to the Responder Resolution Units representing IS0 and IS1. However, PEs listening to IS0 will only send 0 to Responder Resolution Unit 1 no matter what its output is; they send their actual output to Responder Resolution Unit 0. PEs listening to IS1 do the inverse. When resolved signal goes back to PEs, only the correct signal is chosen to pass through Satellite control unit.

5. Machine Instructions for MASC

To implement the MASC architecture, we must add the instructions shown in Table 1 to the existing ASC instruction set. This instruction set is designed for the two IS architecture. If there are more than two control parallel conditions, we have to issue multiple MASC instruction. As example, if there are 4 control parallel conditions, After SCLOAD, the program issue PSEND, PSENDI and ENDIS for the first two conditions, and then it issues another PSEND, PSENDI and ENDIS once again for the other two conditions.

6. Conclusion and Future Work

This paper has described the extension of our initial prototype ASC processor into a scalable ASC processor on APEX 20K1000E device. This design has been completed, and a scalable ASC processor with one Instruction Stream and 50 PEs is currently running on this device, where the functionality will be improved and existing ASC algorithms will be tested. This paper has also described our proposed MASC architecture and its implementation that is currently under way.

References:

[1] R. Walker, J. Potter, Y. Wang, and M. Wu, "Implementing Associative Processing: Rethinking Earlier Architectural Decisions", in *Proc. of the 15th International Parallel and Distributed Computing Symposium (Workshop on Massively Parallel Processing),* abstract on p. 195, full text on accompanying CDROM. IEEE, San Francisco, California, April 2001.

[2] "Implementing Associative Search and Responder Resolution"", Meiduo Wu, Robert A. Walker, and Jerry Potter, in *Proc. of the 16th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing)*, abstract on page 246, full text on CDROM, April 2002.

[3] J.L. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, and C. Asthagiri, "ASC: An Associative Computing Paradigm," *IEEE Computer*, November 1994, pp. 19-26.

[4] A. Falkoff, "Algorithms for Parallel Search Memories", *Journal of Associative Computing*. March 9 1962, pp. 488-511.

[5] ALTERA DATA SHEET
http://www.altera.com/literature/ds/apex.pdf

[6] M. Jin, J. Baker, and K. Batcher, "Timings for Associative Operations on the MASC Model", in *Proc. of the 15th International Parallel and Distributed Computing Symposium (Workshop on Massively Parallel Processing),* abstract on p. 193, full text on accompanying CDROM. IEEE, San Francisco, California, April 2001.