Implementing Associative Search and Responder Resolution

Meiduo Wu, Robert A. Walker, and Jerry Potter

Computer Science Department
Kent State University
Kent, OH 44242
{mwu, walker, potter}@cs.kent.edu

Abstract

In a paper presented last year at WMPP'01 [Walker01], we described the initial prototype of an associative processor implemented using field-programmable logic devices (FPLDs). That paper presented an overview of the design, and concentrated on the processor's instruction set and its implementation using FPLDs. This paper describes the implementation of the processor's associative operation — associative searching and responder resolution — in more detail.

1 Overview of Prototype ASC Processor

Developed at Kent State, the *ASC* (*ASsociative Computing*) model [Potter92, Potter 94] of associative computing grew out of work on the STARAN and MPP computers at Goodyear Aerospace Corporation. Associative processors store one record of data in each PE, and can search for a key value across all PEs, or find the maximum value in a field across all PEs, in constant time.

The initial prototype of our ASC processor [Walker01] is a byte serial associative processor consisting of a single *IS Control Unit*, and an *Associative Processing Array* that contains 4 PE cells as well as some support circuitry. Under the direction of the IS Control Unit, the Associative Processing Array performs associative searches and other operations. The IS Control Unit is also responsible for processing multi-byte operations.

The Associative Processing Array, shown in Figure 1, consists of an array *of Processing Element (PE) Cells*, along with *MAX/MIN circuitry* and *responder resolution circuitry*. Each PE Cell consists of a PE and a local data memory. The local data memory usually stores one record out of a tabular data structure, where a record consists of several variable-size fields. The MAX/MIN circuitry can find the maximum or minimum value in a particular field across all the data memories in the PE array in constant time, while the responder resolution circuitry can recognize the existence of at least one responder in constant time.

Each PE is composed of an 8-bit ALU, a 1-bit ALU, sixteen 8-bit General-Purpose Registers, sixteen 1-bit Logical Registers, a Responder Register, a Mask Stack, and Step/Find/ResolveFirst circuitry. The 8-bit ALU and the sixteen 8-bit *General-Purpose Registers* ($GR0 to $GR15) is the portion of the CPU that performs the actual arithmetic and logical operations. The 1-bit ALU and the sixteen 1-bit *Logical Registers* ($LR0 to $LR15) are used to perform logical operations.

In conjunction with the responder resolution and selection circuitry (the *Responder Register*, the *Mask Stack*, and the *Step/Find/ResolveFirst* circuitry), the 1-bit ALU and the Logical Registers are also used to support associative processing. The 1-bit Responder Register is used to indicate whether a PE is a responder to a particular associative search or not. The Step/Find/ResolveFirst circuitry is used to iteratively step through multiple responders in various ways as described in Section 3.2.

The Mask Stack can contain at most 16 1-bit logical values, and represents multiple levels of association (discussed later). The top of the Mask Stack always represents the current status of the PE — whether it is masked ('1') or unmasked ('0'). In ASC, there are two types of instructions executed by the Associative Processing Array: masked instructions and unmasked instructions. Masked instructions are only executed by those PEs with a '1' on the top of their Mask Stack, while unmasked instructions are executed by all the PEs regardless of the state of the Mask Stack. If the value in the Responder Register is pushed onto the Mask Stack, then masked instructions can be used to limit further processing to only those PEs with a successful associative search.

The remainder of this paper is organized as follows. Section 2 describes how associative search is implemented — how the ASC processor can search for a particular value across all PEs, or for a maximum or minimum value in a particular field. Section 3 describes various options supported by ASC for processing multiple successful responders.

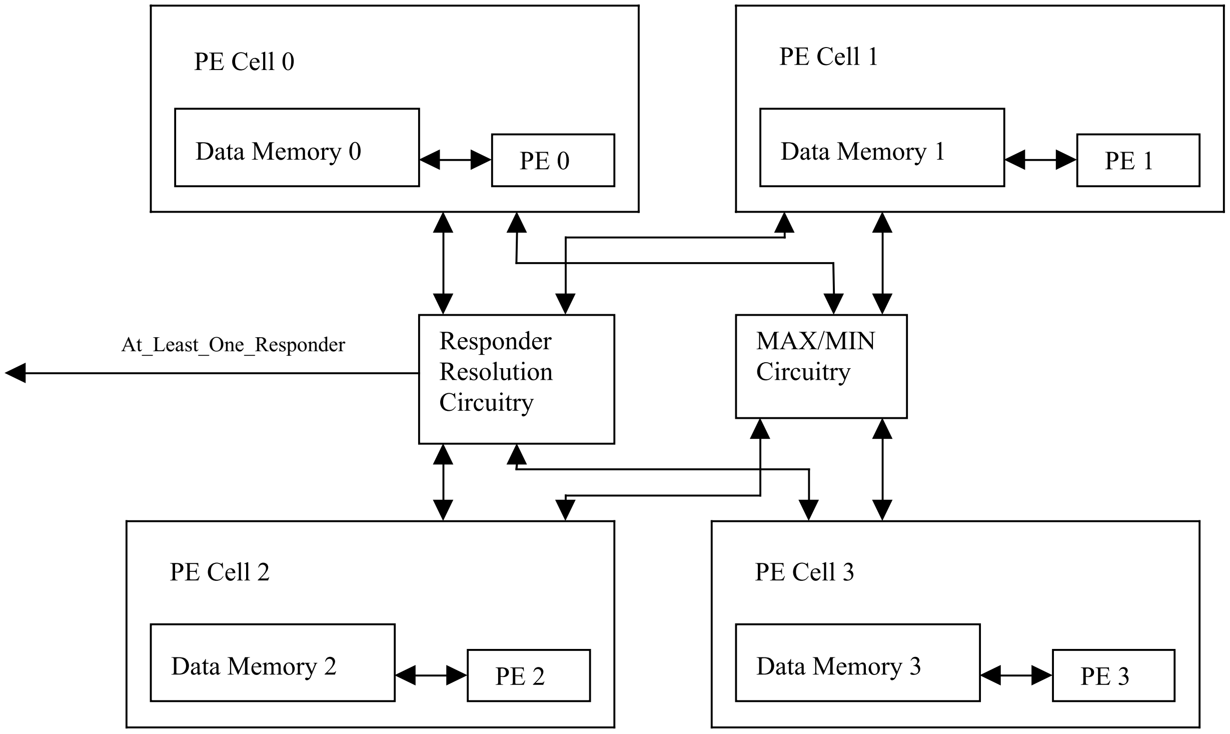

Figure 1 – Associative Processing Array in ASC

2 Associative Search

The fundamental operation of an associative processor such as ASC is *associative searching*. If a tabular data structure is stored in each PE's local data memory, ASC can search for a particular key value in a particular field across all PEs, or find the extreme value in a field across all PEs, in constant time (some of these timing issues were reexamined recently in [Jin01]). This section will describe how associative searching is implemented in our prototype ASC processor.

2.1 Using the Responder Register and Mask Stack in Associative Searching

When performing an associative search for a key value, the *Responder Register* in each PE is used to store the result of the search, and the *Mask Stack* is used to store multiple levels of search results (multiple association groups). To illustrate how the Mask Stack and the Responder Register are used in associative search, assume the data memories in the PE array store information for cars on the lots of various auto dealers. If each data memory stores the information for one car, the Associative Processing Array might contain the values shown in Figure 2.

Given this data, suppose a customer wants to find all Focus cars located on a dealer's lot in Ohio. This search contains two conditions: the model must be "Focus", and the location must be "Ohio". The associative search begins by initializing the top of the Mask Stack to TRUE for all PE cells, namely, pushing '1' onto the initially empty stacks. After the first comparison is performed by all PEs (in parallel), the result, which is either '1' or '0' (TRUE or FALSE), is stored by each PE in one of its logical registers, perhaps $LR1. The result of the second comparison is similarly stored in $LR2. To produce the final result, $LR1 is ANDed with $LR2, and the result is temporarily stored by each PE into its Responder Register.

The top of the Mask Stack, which indicates whether the PE is masked ('1') or unmasked ('0'), must also be updated. Across all PEs, the temporary result in the Responder Register is ANDed with the top of the Mask Stack, and the final result is pushed onto the Mask Stack, leaving the PE masked (responding to following masked instructions) only if it was masked before and the associative search was successful. This final result is also stored back into the Responder Register.

The Mask Stack is particularly useful if there are multiple associative groups in an application. If a search contains

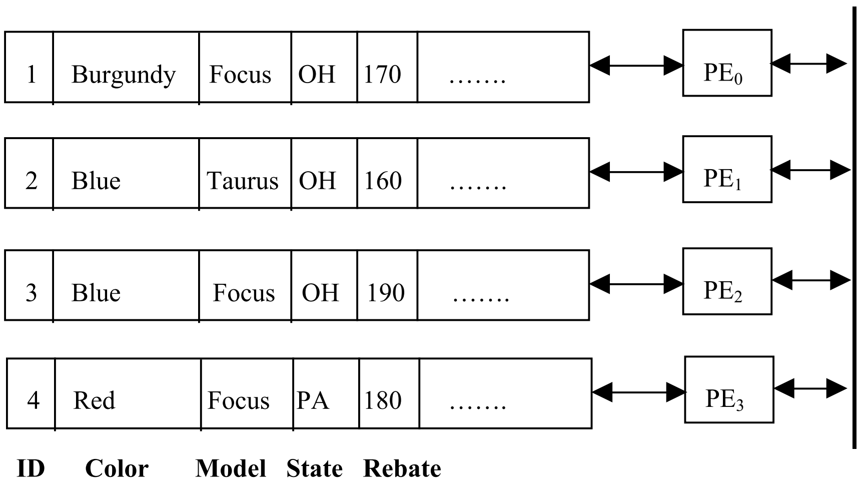

Figure 2 – Auto Information Stored in the PE Cells

multiple levels of conditions, such as an IF-THEN statement with several nested conditions, the result of each condition in this statement will be pushed onto the stack according to the order of nested levels from outside to inside.

2.2 Maximum/Minimum (MAX/MIN) Circuitry

An associative processor not only can search for a particular key value in a particular field across all PEs in constant time, but also can find the extreme (maximum or minimum) value in a particular field across all PEs in constant time. The *Maximum/Minimum (MAX/MIN) circuitry* provides the later capability.

2.2.1 The Algorithm

The general idea for the MIN/MAX circuitry is to use bit slices as masks for the extreme (maximum or minimum) value [Falkoff62]. To search for the maximum value, the following algorithm is performed in parallel across all PEs:

1. Search the bit slices from the most significant bit to the least significant bit. As each bit slice is processed, it is ANDed with the Mask bit (a 1-bit Mask register used to indicate whether or not the data in the specified field is the maximum).
2. Check all the results of the AND to ensure that at least one new maximum value remains (at least one result is 1). If this condition is true, then the Mask bit is updated; if all the results are 0, then all current entries are considered have the same bit value and remain tied (the Mask value is not updated at this time).
3. Continue to process the remaining bit slices as above until all are processed.
4. Once all bit slices have been processed, if only one Mask bit is 1, it marks the largest number; if more than one Mask bit is 1, those cells are tied for the maximum value.

The minimum value can be obtained in a similar way, but the bit slices are complemented first before being ANDed with Mask bits.

2.2.2 The MAX/MIN Circuitry

The MAX/MIN circuitry, shown in Figure 3, uses the above algorithm to search for the extreme value across multiple values in constant time. In this circuitry, there are four 8-bit shift registers (one for each PE cell), four 1-bit Mask registers, four AND gates, and a large OR gate. The shift registers will shift the one-byte data bit-by-bit simultaneously for processing. The Mask registers always keep track of the result of comparison of one bit slice. After all the eight bit slices are processed, the Mask register will contain the result for identifying the extreme value. Since this ASC processor is byte-serial, the MAX/MIN circuitry is also designed to process data byte by byte for consistency.

2.2.3 An Example of Searching for a Maximum Value

Using the car example from Figure 2 again, if the customer wants to find the largest rebate offered among all the cars, the algorithm works as shown in Figure 4. To begin, the MAX/MIN circuitry loads the rebate data from the PE's data memories into the corresponding shift registers, and all Mask registers are initialized to '1' to

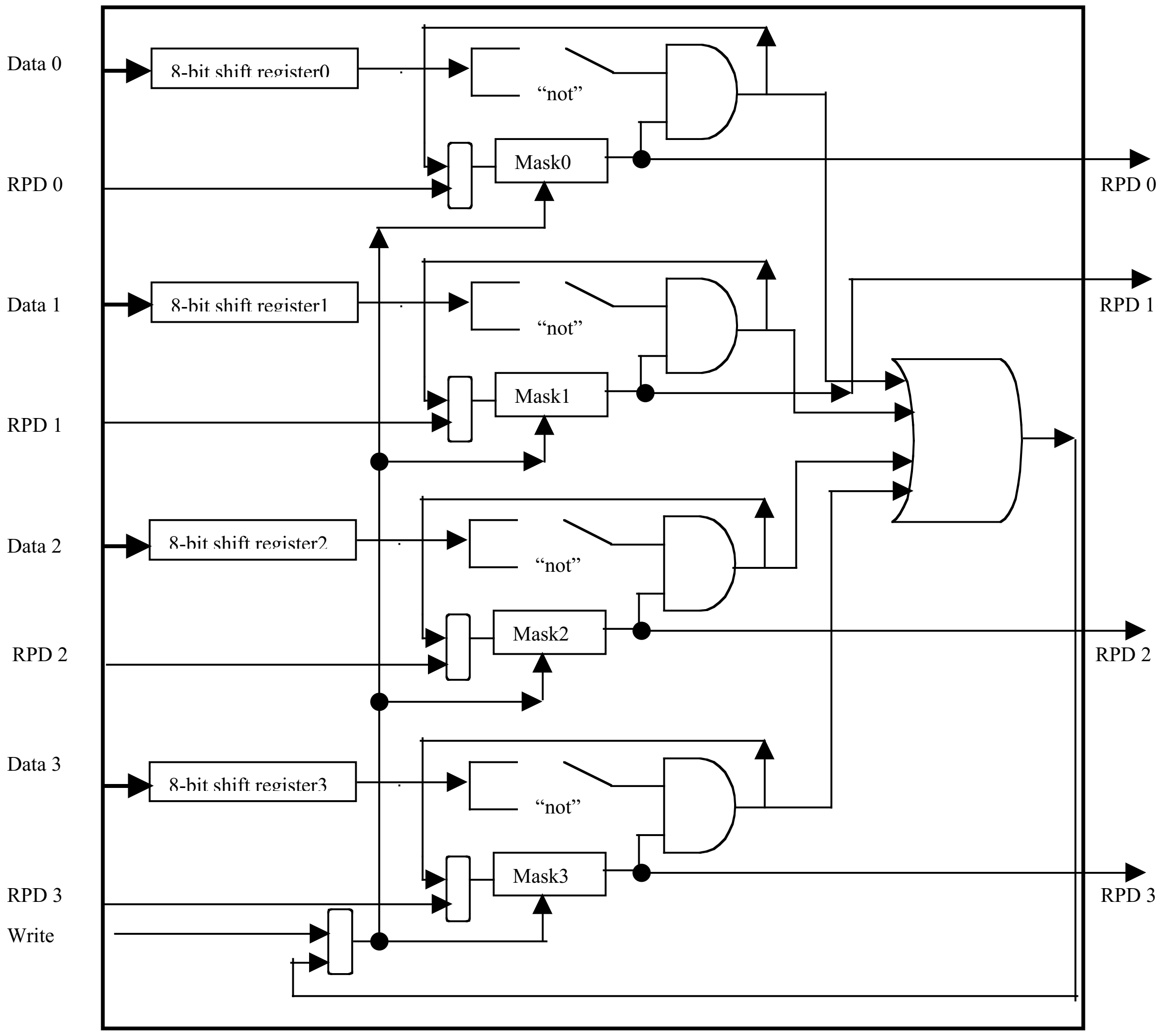

Figure 3 – MAX/MIN Circuitry

indicate that all PEs are active. Then it begins to process the bit slices from left to right. Since all of the seventh bits are '1', and all the results of ANDing each seventh bit with its corresponding Mask bit are '1', the ANDing results update the values in the Mask registers after processing the seventh bit slice.

The remaining bits are processed similarly. After the sixth bit slice is processed, the Mask registers remain unchanged since all the sixth bits are '0'. All the fifth bits are '1', so the Mask registers all remain '1'. In the fourth bit slice, two bits are '0' and two ANDing results are '0', so the value of two Mask registers become '0' while other two remain '1'. Continuing the algorithm, after processing the third bit slice, only the value of Mask2 register remains '1'. The Mask2 register remains '1' until the end. This result indicates that PE2 has the maximum rebate value.

2.2.4 MAX/MIN Processing of Multi-byte Values

The example above searches to find the maximum value in one-byte data. However, the MIN/MAX circuitry can also be used to search for the extreme value in multiple-byte data. Since the IS Control Unit is responsible for processing multiple-byte fields, it would execute assembly code for searching for the maximum value in 4-byte data as follows:

```
SETMAXIN           // load responder bit to Mask
                   register
LDMAXIN   $GR1     // load the most significant
                   byte to shift register
```

Bit Slice (7..0) of Rebate		Value in Mask Register During Processing		
Process bit from left to right	$\longrightarrow$			After processing each bit $\longrightarrow$
(Bit order)	7 6 5 4 3 2 1 0		Initialize	7 6 5 4 3 2 1 0
Rebate				
(170)	1 0 1 0 1 0 1 0	Mask 0	1	1 1 1 0 0 0 0 0
(160)	1 0 1 0 0 0 0 0	Mask 1	1	1 1 1 0 0 0 0 0
(190)	1 0 1 1 1 1 1 0	Mask 2	1	1 1 1 1 1 1 1 1
(180)	1 0 1 1 0 1 0 0	Mask 3	1	1 1 1 1 0 0 0 0

Figure 4 – Processing Data in the MAX/MIN Circuitry

```
MAX                 // search for the maximum
LDMAXIN   $GR2      // load the second byte to
                    shift register
MAX                 // search for the maximum
LDMAXIN   $GR3      // load the third byte to shift
                    register
MAX                 // search for the maximum
LDMAXIN   $GR4      // load the least significant byte
                    to shift register
MAX                 // search for the maximum
STMAXIN             // store value of Mask register
                    back to responder bit in PE
```

In the code above, the 4-byte data is stored across $GR1, $GR2, $GR3 and $GR4 in each PE. The MAX instruction needs 8 cycles to process 8 bit-slices in parallel from the most significant bit to the least significant bit. As successive bytes are processed, the Mask registers store the result, eventually identifying the maximum value after all bit-slices are processed. Finally, the STMAXIN instruction stores the values of Mask registers back to the Responder Registers.

Searching for the minimum value is similar to searching for the maximum value. Instead of using the MAX instruction, the minimum value searching uses the MIN instruction. The assembly code needed for searching for the minimum value of 4-byte data is as follows:

```
SETMAXIN            // load responder bit to Mask
                    register
LDMAXIN   $GR1      // load the most significant byte
                    to MAX/MIN
MIN                 // search for the minimum
LDMAXIN   $GR2      // load the second byte to shift
                    register
MIN                 // search for the minimum
LDMAXIN   $GR3      // load the third byte to shift
                    register
MIN                 // search for the minimum
LDMAXIN   $GR4      // load the least significant byte
                    to MAX/MIN
MIN                 // search for the minimum
STMAXIN             // store value of Mask register
                    back to responder bit in PE
```

3 Responder Resolution and Selection

As described in Section 2.1, the result of a successful associative search is indicated by a '1' in a PE's Responder Register (such a PE is often referred to informally as a "responder"). Because there may be more than one successful match, *responder resolution circuitry* is necessary to recognize responders, and *responder selection circuitry* is often necessary to select one particular responder for further processing.

3.1 Responder Resolution Circuitry

The *responder resolution circuitry* is used to identify whether or not there exists at least one responding PE. Based on the hardware implementation of the reduction network in the STARAN [Batcher99], the current responder resolution circuitry for our 4-PE Associative Processing Array is shown in Figure 5. This responder resolution circuitry has two important functions: (1) to send an At_Least_One_Responder signal to the IS Control Unit to tell it that a PE has responded, and (2) to send a corresponding Responders_Before_Me signal to the Step/Find/ResolveFirst circuitry in each PE. (The Step/Find/ResolveFirst circuitry selects a particular responder as described later in Section 3.2.)

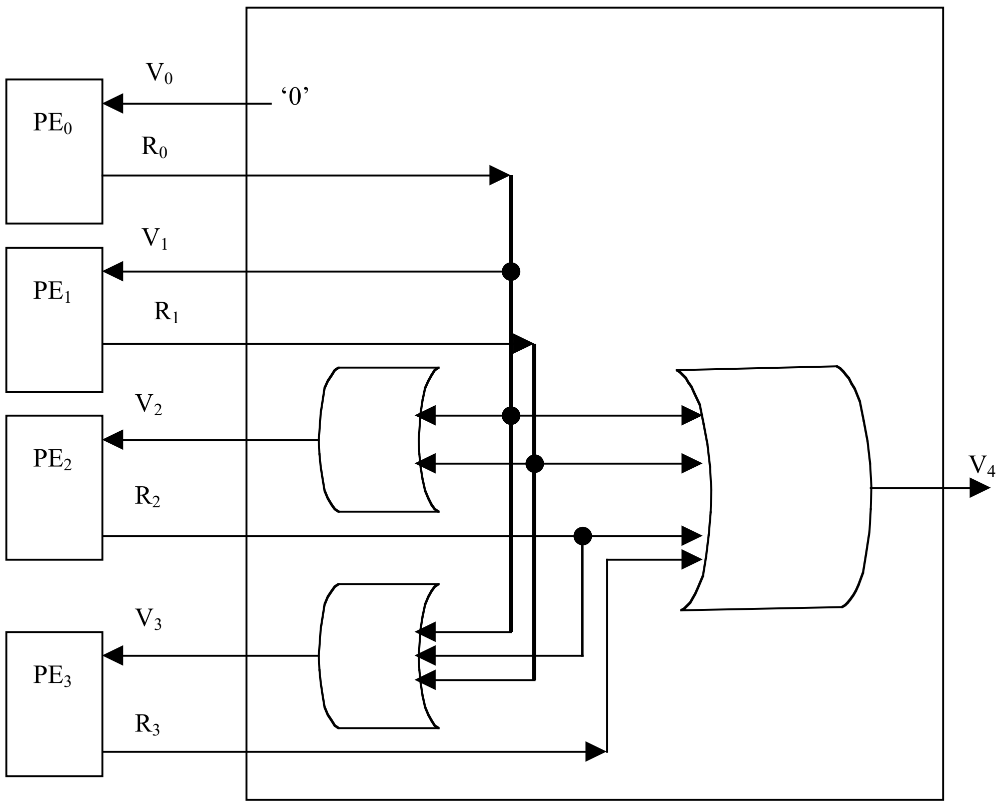

Figure 5 – Implementation of Responder Resolution Circuitry for 4 PEs

As shown in Figure 6, each PE has a responder bit (only PE_0 is shown for simplicity), which is used to indicate whether or not a PE is a responder after an associative search operation as described in Section 2.1. The four inputs to the responder resolution circuitry are the four responder bits ($R_0 \sim R_3$), and $V_0 \sim V_4$ are the five 1-bit outputs of this circuitry. We call signals $V_0 \sim V_3$ Responders_Before_Me signals, and V_4 the At_Least_One_Responder signal.

This circuitry recognizes a responder in unit time as follows. Assume we select a responder with the smallest PE ID number if there are several responders. Since PE_0 has the smallest PE number among these four PEs, we give V_0, which is sent to PE_0, a constant value '0'. This tells PE_0 that no responder whose PE ID number is smaller than '0' exists.

The formula for other V_j outputs is: $V_j = R_0 \lor R_1 \lor \ldots R_{j-1}$ (j = 1, 2, 3, 4). By sending V_i (i = 0, 1, 2, 3) to PE_i, the resolution circuitry tells each PE whether or not there exists a responder whose PE ID number is smaller than that of the PE itself.

When a Responders_Before_Me signal is '0', the PE receiving this signal is the selected responder if the value of its own responder bit is '1'. In this way, at most one PE will recognize itself as a responder at a time. Meanwhile, the resolution circuitry also sends signal V_4 to the IS Control Unit to tell it whether or not there exists at least one responder. V_4 is the result of ORing all responder bits, so it can identify the existence of any responder in the Associative Processing Array.

3.2 Responder Selection (Step/Find/ResolveFirst) Circuitry

In an ASC program, it may often occur that an associative search results in multiple responding PEs. In this case, those responding PEs can be processed, using masked instructions as described in Section 1, either in parallel or sequentially. If the responding PEs are processed in parallel, all the responding PEs execute the same set of instructions under the direction of the IS Control Unit. However, if the responding PEs must be processed sequentially, there must exist some mechanism to select a particular responder each time.

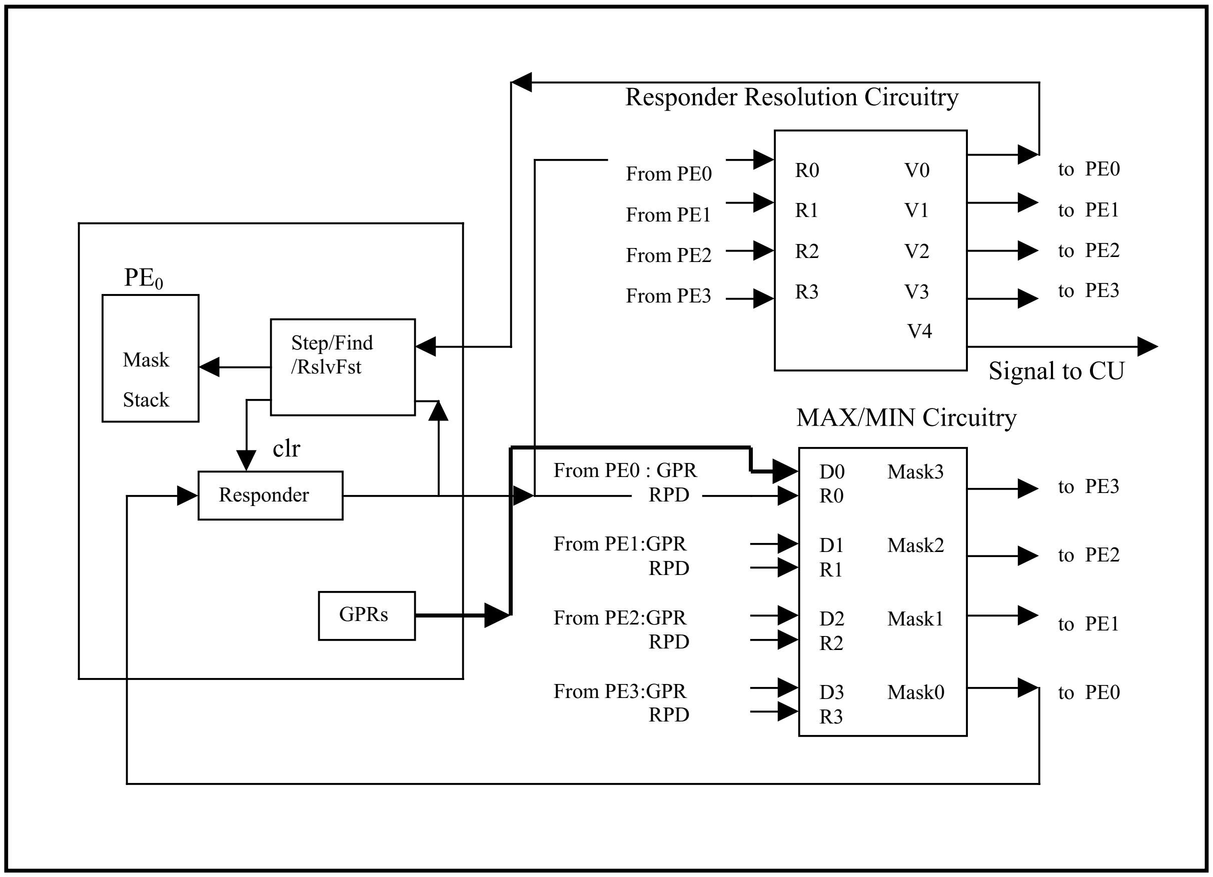

Figure 6 – Responder Circuitry and MAX/MIN Circuitry With PE$_0$

In the ASC language, special responder iteration instructions are used to process multiple responding PEs in the PE array:

- The STEP instruction is repetitively used to pick one responding PE each time for further processing, until all the responding PEs have been visited. It can be thought of as a "for" loop that processes all responders in turn. For the example in Figure 2, after the associative search has found all the Focus cars in Ohio, a STEP instruction could be used to step through the responding PEs to see what color choices are available.
- The FIND instruction is used when only one responding PE must be processed at the moment, but later other operations might be applied to all the responding PEs. FIND must select a responding PE, while still keeping all responders identifiable. It can be thought of as a "while" loop that processes all responders. For the example in Figure 2, a FIND instruction could be used to select one of the Ohio Focus cars, so that Ohio related expenses such as sales tax, emission inspection fees, etc, can be determined for that one car, and then applied to all the Ohio Focus cars in a subsequent step.
- The RESOLVEFIRST instruction is used when only one responding PE must be processed. RESOLVEFIRST must select a responding PE and clear all Responder Registers. For example, after searching for an extreme value, e.g., the maximum rebate, it is possible that multiple PEs might have that same extreme value. However, if all we care about is getting that value, RESOLVEFIRST can be used here to select a responding PE from multiple responding PEs and keep only that one responder identifiable.

Implemented at the assembly language level, Step, Find and ResolveFirst are the three associative selection operations that are used when responders must be processed individually. These three instructions might be issued and sent to the Associative Processing Array by the IS Control Unit when at least one responder exists in the PE array (i.e., when the At-Least-One-Responder signal from the Associative Processing Array is true). In our ASC processor, dedicated *responder selection (Step/Find/ResolveFirst) circuitry* is used in each PE to process these three instructions. As mentioned before, the responder resolution circuitry sends each PE a Responders_Before_Me signal. The *responder selection circuitry* utilizes this signal to make its own decision.

The Step instruction is useful when the responding PEs must be stepped through sequentially. It clears the Responder Register of the selected PE (so that it will not be selected again in the future), yet leaves other PE responders' Responder Registers unchanged. The Step operation in PEi works as follows:

```
IF ( Responders_Before_Me = "0") AND ( the
        responder bit of PEi  is "1" )
THEN
        { push "1" into MaskStack;      //  mask this PE
              to respond to masked instructions
        clear Responder Register;       //  clear the
              responder bit of this PE
        }
ELSE
        {  push "0" into MaskStack;     //  unmask this
              PE
        }
```

The Find instruction is useful when one responding PEs must be selected, but it does not matter which one. However, it leaves **all** Responder Registers unchanged. The Find operation in PEi works as follows:

```
IF ( Responders_Before_Me = "0" ) AND ( the
        responder bit of PEi  is "1" )
THEN
        { push "1" into MaskStack;      // mask this PE to
              respond to masked instructions
        }
ELSE
        {  push "0" into MaskStack;     //  unmask this
              PE
        }
```

The ResolveFirst instruction is useful when one responding PEs must be selected, but it does not matter which one. It not only selects this PE, but also clears the Responder Register of the other PEs (so that they will not be selected in the future). The ResolveFirst operation in PEi works as follows:

```
IF ( Responders_Before_Me = "0" ) AND ( the
        responder bit of PEi  is "1" )
THEN
        { push "1" into MaskStack;      // mask this PE to
              respond to masked instructions
        }
ELSE
        {  push "0" into MaskStack;     //  unmask this
              PE
        clear Responder Register;       //  clear the
              responder bit of this PE
        }
```

4 Conclusions and Future Work

This paper has described the implementation of associative searching in our prototype ASC processor. Although we have described a design with only 4 PEs, the design can be scaled up substantially. Using Altera FLEX 10K chips, 4 PEs and the support circuitry described here require about 70,000 gates, so a chip with a million gates could accommodate approximately 60 PEs. We are currently exploring the option of using larger chips versus many smaller chips.

5 Acknowledgements

The authors thank Dr. Ken Batcher, Yanping Wang and other members of the Parallel Processing Group for many helpful discussions.

6 References

[Batcher99] K. Batcher, "The Resolver Network", seminar, Kent State University, October 1999.

[Falkoff62] A. Falkoff, "Algorithms for Parallel Search Memories", Journal of Associative Computing. March 9 1962, pp. 488-511.

[Jin01] 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.

[Potter92] J.L. Potter, *Associative Computing*, Plenum Publishing, New York, 1992.

[Potter94] 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.

[Walker01] 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.