An FPGA based Associative Execution Unit[(*)].

Iacob Petrescu, Quadrilogic s.r.l. (iacobp@quadrilogic.ro)

Abstract.

Field Programmable Gates Array (FPGA) has enable the advent of a new computing paradigm based on direct hardware computational algorithm implementation. The presence of the adequate CAD platforms, doubled by powerful FPGA's hardware resources, facilitates experimentations with various computational architectures. The present papers explores the design of a FPGA associative execution unit, dedicated mainly to solving complex search problems from many application areas including computational geometry, graph theory, list/matrix computations, etc.
The associative execution unit is parameterized as word-length and word-numbers and operates in a bit-parallel/word-serial mode. Architectural details of the multi-comparand/multi-repondent associative execution unit and its functions are also provided. The unit can be attached to a host computer system, in order to obtain instructions and to exchange data, or provided with a control unit for autonomous operation. Features of the utilized FPGA devices are presented and discussed as well as a the results of a search example.

Key words: Associative memory,Associative Execution Unit, FPGA, Multi-comparand, Verilog, Xilinx ISE Design Suite.

Introduction.

Data storing and data searching are among the basic operations in computer science and computer engineering. Searching problems arise in different important areas as a result of the exponentially grows of the stored data. New solutions, based on massively parallel processing, are required in order to speedup search operations. Particularly parallel associative processors are well suited to cope with such tasks. An associative processor comprise, as a basic component, an associative memory in which the words are stored and retrieved based on their contents, not on their address [1], [2], [3]. In order to have the above mentioned features an associative memory must provide at least the following functions: broadcast of search argument/comparand to all locations, comparison of the search argument/comparand with the contents of all locations, identification of matching words and, if necessary, prioritizing multiple matching words. Desirable would be the presence of the facility for a multi-comparand search, with multi-responders, as well as some logic processing with responder arguments. Associative machines belong to a broader category of parallel SIMD (Single Instruction Multiple Data) machines that are well suited for fast parallel search operations. In the last years, an important progress was achieved, concerning the implementation of a new associative machines with:

(*) TheXXXVIII–International Scientific Symposium, Military Equipment and Technologies Research Agency. Bucharest. 29-30, May, 2008.

reconfigurable processing elements interconnection network for embedded applications [4], associative processor for database applications and image processing [5], scalable ASC processor [6], associative search and responder resolution features [7], multi-comparand, multi-search FPGA based associative processors [8], [9], etc. One of main disadvantage of associative memory is the lack of appropriate software. Running existing code on a processor with an associative memory will probably not yield performance improvement. To overcome this problem entirely new algorithms has been proposed.

Various associative machines implement search operation in two main modes: fully associative and less than fully associative.

The *full associative mode* (bit-parallel, word-parallel) has a search time of an order of *O*(1). The comparison logic is included in each memory cell, the masked comparand is checked against to all bits of all data memory words and the result is stored in the respondent word (fig.1).

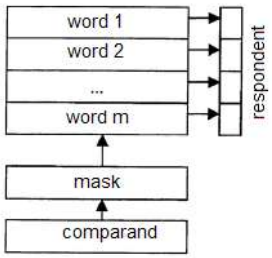

Fig.1. Fully associative search.

Less than fully associative search mode require les hardware but more time to match than fully associative architecture and it can be implemented in three ways: *bit-serial, byte-serial, word-serial*.

In *bit-serial architecture* (fig.2), a bit slice from each data memory word is inspected simultaneously. For example, bit *n* of al data memory words is inspected, bit *n*-1 is inspected, bit *n*-2 is inspected, and so on, until the entire width of the key field is checked. The search time in this architecture is *O*(*n*), where n is the data memory words width in bits. Bit-serial architectures are common because they can be constructed using off-the-shelf conventional RAM chips. The comparison logic is at the output of the memory, not in each word. One bit-slice at a time is loaded into comparison logic and is compared with the appropriate bit from the masked comparand.

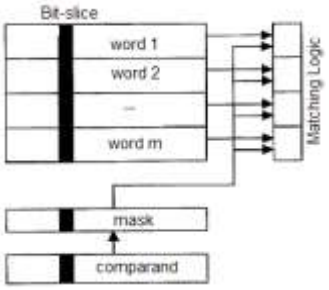

Fig.2. Bit-serial search.

Byte serial architecture (fig.3) is similar to bit-serial, except that a byte slice from each data memory word is inspected simultaneously. For example, byte 0 of each data memory word is inspected,

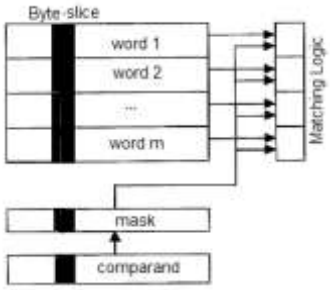

Fig.3. Byte-serial search.

byte 1 is inspected, and so on until the entire key field is checked. The search time in this organization is $O(b)$, where b is the number of bytes in the data memory word.

Word-serial architecture (fig.4) compares one entire data memory word against the masked compared, then checks the next data word and so on, until the end of the data memory word is reached. One data memory access for each check, which is done sequentially. The search time in this

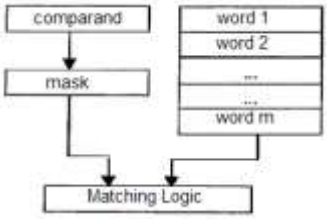

Fig.4. Word-serial search.

architecture is $O(m)$, where m is the number of words in the data memory. This architecture still performs better than a conventional architecture, however, because the comparisons are all done in the neighborhood of the data memory and do not require loading data into the processor registers.

Mass storage systems. Associative access to data from mass storage systems such as disks or tape can be done by adding hardware at the interface between main memory and the mass storage. The extra hardware check for matches on the fly as the data is transferred from mass storage (fig.5). Most

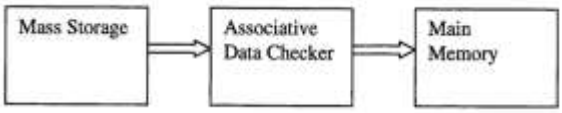

Fig.5. Adding hardware at the interface between main memory and the mass storage.

proposals for this type of access check a large amount of data, such as a sector from a disk, in parallel and are thus quite fast. Usually, only data blocks containing the desired pattern are written into main memory for further processing.

Adding an I/O buffer. Another disadvantage of associative memory is that, even though it is searched in parallel, it must be loaded serially. Therefore unless the data can be loaded into the memory at one time and then used repeatedly, there will be little performance improvement over conventional memory. A solution was proposed to this problem [10], which uses a part of the associative memory as an input/output buffer (fig. 6). This reduces the load and store time, but request additional memory cells. The I/O buffer can be loaded and stored serially while other processing is occurring in

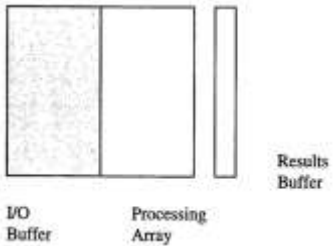

Fig.6. Using a part of the associative memory as an input/output buffer.

the remainder of the memory. Once the I/O buffer is ready, it can be transferred to or from the processing array section of the memory one bit-slice at a time, using the results buffer.

The next section describes a proposed FPGA based associative execution unit, dedicated mainly to solving complex search problems from many application areas including computational geometry, graph theory, list/matrix computations, etc. The associative execution unit is parameterized as word-length and word-numbers and operates in a bit-parallel/word-serial mode. Architectural details of the multi-comparand/multi-repondent associative execution unit and its functions are also provided. The unit can be attached to a host computer system, in order to obtain instructions and to exchange data, or provided with a control unit for autonomous operation. In the final section features of the utilized FPGA devices are presented and discussed as well as a the results of a search example.

1. **The Associative Execution Unit Architecture** (AEU).

In this design, AEU communicates with a host computer in order to receive instructions, send conditions and exchange data, using the dedicated registers: IR, CR and I/O. The instruction received from the host is interpreted by a local control unit (LCU), which sends the appropriate control signals to the command points of the involved hardware resources, as a single micro-operation, a single micro-instruction or as a sequences of micro-instructions. Activating a given command point, belonging to a given hardware resource, will evoke a specific micro-operation associated with that hardware resource. From the execution point of view, it is important to define and to code each micro-operation, at each hardware resource level. Also, it is compulsory to code each register which may act as a data source or/and data destination. These codes will enable the designer to define a set of primitive micro-operations and a set of micro-instructions.

The AEU architecture (fig.7) is centered on an associative data memory array (D) with a capacity of *m* words of *n* bits each. At each bit *j* level, in each word *i*, from D, there is a *comparison logic*, for *equal* and *less* relationships, between corresponding bits *j* of a selected pair: *comparand* Ck - *mask* Mk.

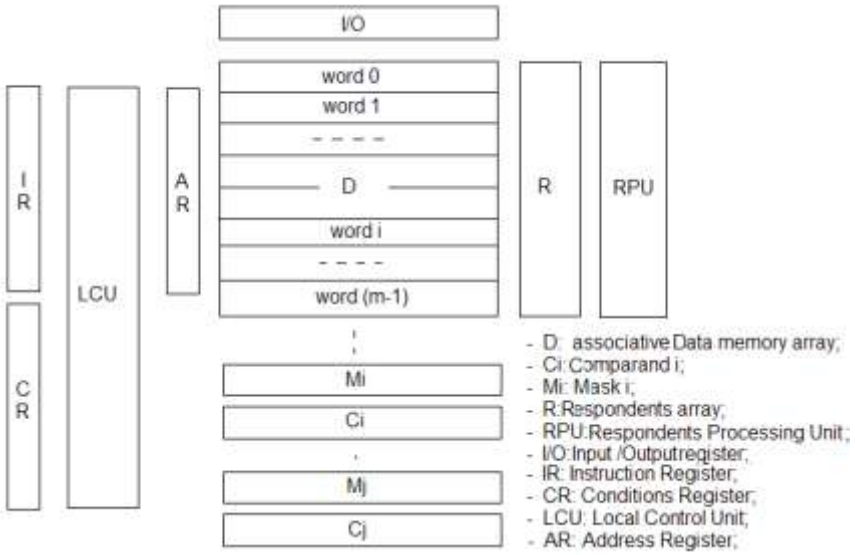

Fig.7. General Associative Execution Unit Architecture.

In order to obtain the logical values for the bit *i* of the respondents Re (*equal*) and Rl (*less*), from the respondents array R, along each word *i*, from D, there are two logic networks for processing the *equal* and *less* results obtained at each bit level of the word *i*. The respondents array comprises a number of *m* bits registers. It is compulsory the presence of the respondents registers *equal* (Re) and *less* (Rl), among these registers. Re and Rl are supplied directly by the logic of the array D, The other registers would store the respondends for: *less or equal* (Rle), *equal or greater* (Reg) and *greater* (Rg), computed by RPU on the following relationships:

$$Rle = Rl \mid Re \quad (1)$$

$$Rg = \sim Rle \quad (2)$$

$$Reg = Rg \mid Re \quad (3)$$

To facilitate multi-comparand associative operations, AEU comprises an array consisting of comparand-mask pairs {Ci, Mi},$1 \leq i \leq p$, where *p* si a given integer.

The Respondents' Processing Unit (RPU) executes unary and binary logical operations on operands selected from respondents' array. The final destination of the RPU results are the registers that implement the R array, whose contents may be transferred to host computer via the I/O register. Generally speaking, the AEU consists of combinational resources as: adders, muxes, ALUs, etc., and resources with memory: registers, memories and so on. Figures 8 and 9 illustrate the diagrams of these resources and the associated control and synchronizing signals.

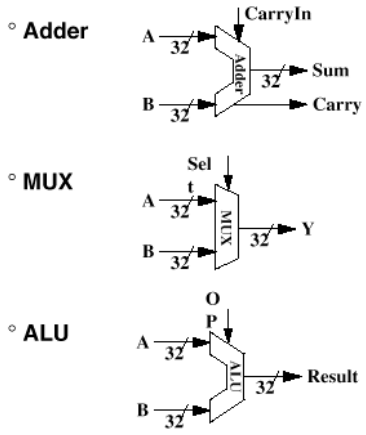

Fig.8. Adder's, Muxe's and ALU's control signals.

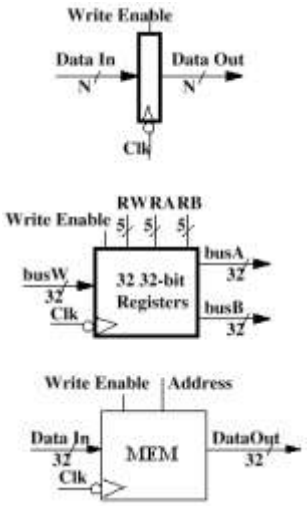

Fig.9. Register's, General Registers'and main Memory's control and synchronizing signals.

In order to solve a given problem one must design a suitable AEU structure, based on the above described general AEU organization, and to express it in a Hardware Description Language (HDL) terms. Once the description is done the next two steps follow: simulation and implementation.

2. Simulation.

With the rapid advances in GPS technologies, it is now become feasible for spatio-temporal database systems to keep track of continuously moving objects accurately. For example, the mobile phone service provider may wish to know how many users are currently present in a specific area. It is

believable that the same problem arises in connection with planes flaying in the neighborhood of an airport.

Considering a 2D situation the problem statement is the following. Let $S = \{p_1, p_2, ..., p_n\}$ be a set of moving points in $\mathbb{R}^2$. For any time t, let $p_i(t)$ be the position of p_i at time t, and $S(t) = \{ p_1(t), p_2(t), ..., p_n(t)\}$ be the configuration of S at time t. Given an axis-aligned rectangle $R \subseteq \mathbb{R}^2$ and a time stamp t_q, report $S(t_q) \cap R$, i.e., all points of S that lie inside R at a time t_q.

Lets have a set *S* of 8 points stored as bytes in a file CMASOCI.txt. : 9A 8B 0C 0D FF A0 A3 86. Considering a 2D problem, each byte consists of an left nybble (*y* coordinate) and a right nybble (*x* coordinate). One may be interested to know what points are placed in a rectangle defined on *x* axis by coordinates 05 and 0C and on *y* axis by coordinates 70 and E0.

Inspecting the simulation results, of the following Verilog problem , it may be concluded that inside the mentioned area there are 3 points: 9A, 8B and 86, witch correspond to bits: 1, 2, 8, having "1" values in the resulting binary vector Rz.

```
//  Multi-Comparand Associative Execution Unit (MASMC) parameterized at word length "n" and
//Associative Memory ,""D", capacity  "m" levels. Based on Comparands "Cj" and Masks "Mj" the
//AEU generates //the m bits responders Rje[i] and  Rjl[i] , which express the "=", "<" relationships
//between binary  //vectors"Mj&Cj" si D[i]&Mj"

    module MASMC;
        parameter n=8;
        parameter m=8;
        integer i;
        reg [n:1] D[1:m];
        reg [1:m] R1g,R1l,R2g,R2l,Rz;
        reg [n:1] M1,C1,M2,C2,C3,C4;
    initial begin
        i=1;
        R1g='h00;
        R1l='h00;
        R2g='h00;
        R2l='h00;
        Rz='h00;
        M1='h0F;
        C1='h05;
           M2='hF0;
           C2='h0C;
           C3='h70;
           C4='hE0;
```

```
            $readmemh("CMASOCI.txt",D);
             end

        always
                begin
                  while(i<m+1)
                   begin
                     R1g[i]=((M1&C1)<(D[i]&M1));
                     R1l[i]=((M1&C2)>(D[i]&M1));
                     R2g[i]=((M2&C3)<(D[i]&M2));
                     R2l[i]=((M2&C4)>(D[i]&M2));
                     Rz[i] = R1g[i]&R1l[i]&R2g[i]&R2l[i];

$display($time,, "M1=%h C1=%h M2=%h C2=%h C3=%h C4=%h D[i]=%h R1g[i]=%b R1l[i]=%b
R2g[i]=%b R2l[i]=%b Rz[i]=%b i=%0d",
M1,C1,M2,C2,C3,C4,D[i],R1g[i],R1l[i],R2g[i],R2l[i],Rz[i],i);
       i=i+1;
    end
   $stop;
end
endmodule
```

```
Entering Phase I...
Compiling source file : MAMC.V
The size of this model is [3%, 2%] of the capacity of the free version

Entering Phase II...
Entering Phase III...
No errors in compilation
Top-level modules:
   MASMC

C1> .
0 M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 D[i]=9a R1g[i]=1 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=1 i=1
0 M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 D[i]=8b R1g[i]=1 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=1 i=2
0 M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 D[i]=0c R1g[i]=1 R1l[i]=0 R2g[i]=0 R2l[i]=1 Rz[i]=0 i=3
0 M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 D[i]=0d R1g[i]=1 R1l[i]=0 R2g[i]=0 R2l[i]=1 Rz[i]=0 i=4
0 M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 D[i]=ff R1g[i]=1 R1l[i]=0 R2g[i]=1 R2l[i]=0 Rz[i]=0 i=5
0 M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 D[i]=a0 R1g[i]=0 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=0 i=6
0 M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 D[i]=a3 R1g[i]=0 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=0 i=7
0 M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 D[i]=86 R1g[i]=1 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=1 i=8
Stop
```

3. Implementation.

The implementation used the tools provided by Xilinx ISE Design Suite 9.203i. The target device was an FPGA chip: xc2s200e-6pq208, with 200 K gates. In order to input new comparands and masks, and to visualize the results, two interfaces were designed and implemented for a PS2 keyboard and an VGA monitor (Fig.10).

Fig. 10. The experimental stand.

The device report summary provided by Xilinx ISE Design Suite 9.203i, the Associative Execution Unit implementation used a total of 150 equivalent gate count. Taking into account as well as PS2 keyboard and VGA monitor interfaces the used equivalent gate count increased to 19, 961.

4. Conclusions.

In order to implement successfuly new algoritms in FPGA, the user must consider the ways in wich data I/O operations and commands/status will be performed: PC host assisted or by means of a dedicated hardware.

The first case is tipical for the development platforms, offering facilities as: versatility, friendly user interface, relatively simple changes of the design, etc. The second case is recommended for stable, embeded solutions in which high data transfer rates are required.

The FPGA based digital systems implementation means: no physical layout process, no mask making, no IC manufacturing , medium costs, tremendous flexibility, etc. Relative to ASICs, lowers NREs(Non Recurrent Expenses), shortens TTM (Time To Market).

References:

1. *** Associative Memory Study: Architectures and Technology. AFRL_IF_RS_TR_2001_264. Final Technical Report, January 2002, University of Pittsburgh.
2. C.C., Foster. Content Addressable Parallel Processors. Van Nostrand Reinhold, New York 1976.
3. I. Petrescu, C. Nitu. Realizarea unei memorii adresabile dupa continut. Revista Romana de Informatica si Automatica, vol.17, nr.4, 2007.
4. H., Wang, R.A.,Walker. A Scalable Pipelined Associative SIMD Array with Reconfigurable PE Interconnection Network for Embedded Applications**",** Proc. of the 17th International Conference on Parallel and Distributed Computing and Systems, pp. 667-673. Phoenix, Arizona, November 2005.
5. H. Wang, L. Xie, M. Wu, R. A. Walker. A Scalable Associative Processor with Applications in Database and Image Processing, Proc. of the 18th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing), abstract on page 259, full text on CDROM. Santa Fe, New Mexico, April 2004.
6. R. A. Walker, H. Wang. Implementi ng a Scalable ASC Processor**.** Proc. of the 17th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing). April 2003.
7. M.Wu, R. A. Walker, J. Potter. Implementing Associative Search and Responder Resolution**.** Proc. of the 16th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing). April 2002.
8. Z. Kokosinski, W. Sikora.An FPGA implementation of a multi-comparand multi-search associative processor. Proc. 12th Int. Conference "Field Programmable Logic and Applications" FPL'2002, Montpellier, France. Lecture Notes in Computer Science, Vol. 2438, pp. 826-835 (2002)
9. Z. Kokosinski.An associative processor for multi-comparand parallel searching and its selected applications. Proceedings of the {IASTED} International Conference on Parallel and Distributed Systems Euro.'97, Barcelona, Spain, June 9-11, 1997. ACTA PRESS.

10. D. Sima, T. Fountain, P. Kacsuk. Advanced Computer Architectures: A Design Space Approach, Addison Wesley, Harlow England, 1977, Chapter 12, pp 455-483.