Magnetic RAM Reconfigurable Array

Victor Silva  
INESC-ID  
vicant@prosys.inesc.pt

Jorge R. Fernandes  
INESC-ID/IST/UTL  
jorge.fernandes@inesc-id.pt

Luís B. Oliveira  
INESC-ID/FCT/UNL  
luis.b.oliveira@inesc-id.pt

Mário P. Véstias  
INESC-ID/ISEL/IPL  
mvestias@deetc.isel.ipl.pt

Horácio C. Neto  
INESC-ID/IST/UTL  
hcen@inesc-id.pt

Abstract—This paper presents the on-going research in developing a coarse-grained Reconfigurable Array using Magnetic RAM. The architecture is organized as a one-dimensional array of programmable ALU. The configuration bits are stored in magnetic random-access memories (MRAM). MRAM provide non-volatility with cell areas and access speeds comparable to those of SRAM, and with lower process complexity than flash memory. MRAM can also be efficiently organized as multi-context memories. This type of coarse-grained array, where each reconfigurable element computes on 4-bit or larger input words, is more suitable to execute data-oriented algorithms and is more able to exploit large amounts of operation-level parallelism than common fine-grained architectures. By substantially reducing the overhead for configurability, this coarse-grain architecture is also more apt to efficiently exploit run-time reconfiguration and therefore to take advantage of the multi-context MRAM-based configuration memories.

I. INTRODUCTION

Reconfigurable arrays (RA) try to overcome the disadvantages of Field Programmable Gate Array (FPGA) based computing solutions by providing multiple bit-wide data-paths and complex operators instead of bit-level configurability. The wider data-paths allow for the efficient implementation of complex operators in silicon. Thus, RA offer lower overhead, they exploit better operation level parallelism and they are better suited for data-oriented algorithms.

So far, most RA are SRAM based and thus their configuration data is lost when power goes off and has to be setup in each power up. Magnetic RAM (MRAM) technology offers a solution to the intrinsic volatility of SRAM and at the same time offers a low reading and writing times, virtual limitless re-programmability and almost no area overhead compared with other technologies such as FLASH as depicted on Table 1 extracted from [1].

II. MTJ PRINCIPLES

A Magnetic Tunneling Junction (MTJ) cell is conceptually made of two thin ferromagnetic layers separated by an ultra thin non-magnetic oxide layer [1, 2], as depicted on Figure 1.

<table>
<thead>
<tr>
<th>Memory Technology</th>
<th>DRAM</th>
<th>FLASH</th>
<th>SRAM</th>
<th>MRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell structure</td>
<td>1T/Cap</td>
<td>1 FGT</td>
<td>6T</td>
<td>1T MTR</td>
</tr>
<tr>
<td>Cell size (130nm u²)</td>
<td>0,14</td>
<td>0,2</td>
<td>0,65</td>
<td>0,34</td>
</tr>
<tr>
<td>Write Energy</td>
<td>&lt; 200pl</td>
<td>~200pl</td>
<td>&lt; 100 pl</td>
<td>&lt; 100 pl</td>
</tr>
<tr>
<td>Endurance</td>
<td>∞</td>
<td>1,00E+06</td>
<td>∞</td>
<td>1,00E+14</td>
</tr>
<tr>
<td>Acess Time</td>
<td>~ 50ns</td>
<td>~ 50ns</td>
<td>5 - 10 ns</td>
<td>~ 30ns</td>
</tr>
</tbody>
</table>

Table 1 Comparison of different memory technologies

Figure 1 - Conceptual view of a Magnetic Tunneling Junction

The relative magnetic orientation of these layers defines two different values of equivalent resistance Rp (low resistance), when the free layer and the pinned layer are oriented to the same direction, and Rap (high resistance), when the free layer and the pinned layer are oriented in opposite direction. These different values of equivalent resistance can be employed as state variables to represent the logic states ‘0’ and ‘1’.

This resistance can be evaluated by sending current through the junction and measuring the voltages at the nodes of the MTJ. A structure is required to determine whether the junction is on a low resistivity configuration or in a high resistivity configuration.

To write information in a MTJ, a magnetic field is applied through the junction in order to force the change on the magnetic orientation of the free layer.
On the Thermally Assisted Switching (TAS) writing approach a local current will momentarily heat up the junction by Joule effect. This heating will unpin the free layer [3]. Magnetic remanence of the ferromagnetic elements provides non-volatility.

These properties will be the foundation of the storage elements that will be employed in the configuration memory cells.

III. COARSE GRAINED RECONFIGURABLE ARCHITECTURES

Coarse grain reconfigurable arrays (RA) as the one depicted on Figure 2, with path-width greater than 1 offer significant advantages compared with fine-grained solutions such as FPGA. Fine-grained solutions are less efficient due to a huge routing area overhead and poor routability [5, 6, 7].

Since computational data-paths have regular structure, full custom design of reconfigurable data-path units can drastically reduced the required area when compared with the ones obtained by assembling FPGA single bit Configurable Logic Blocks (CLB). Coarse-grained architectures provide operator level Configurable Function Blocks (CFB), word level data-path and powerful and very area-efficient data-path routing switches [5, 6].

They also offer a massive reduction of configuration memory and thus of configuration time, as well as a drastic reduction on the complexity of the placement and routing (P&R) problem [5, 6, 7].

Finally, their operators data-path and functionality are nearer to average target applications and thus the mapping of an application dataflow and control flow is easier than with fine-grained architecture. Coarse-grained architectures offer a real opportunity of relieving the average application designer of having a deep understating of VLSI design issues, namely the know-how required for the efficient generation of computational structures employing hardware design language such as VHDL.

Since the 1990s, several architectures were proposed and implemented, as depicted on Table 2 extracted from [5, 6]. These architectures can be classified according to their basic interconnect structure, granularity and reconfiguration model [5, 6, 7].

The basic interconnect structure is defined by the global arrangement of the processing elements. This arrangement is set either by the target application architectural requirements and/or by implementation constraints. As depicted on table 2, the predominant architectural arrangement is a 2-D array or mesh.

On mesh architectures their processing elements (PE) are arranged in a rectangular 2-D array with horizontal and vertical connections which support rich communications resources that encourage nearest neighbor links between adjacent PE (4NN or 8NN) and with longer lines added width different lengths for connections over distances larger than 1

In other architectures, the processing elements are arranged as one or more linear arrays. Finally, the remaining architectures employ a crossbar switch to connect processing elements.

The width of the data-path varies from 2 bits to 128 bits.

<table>
<thead>
<tr>
<th>Project</th>
<th>First Pub</th>
<th>Architecture</th>
<th>Granularity</th>
<th>Fabrics</th>
<th>Intended target application</th>
</tr>
</thead>
<tbody>
<tr>
<td>PADDI</td>
<td>1990</td>
<td>Croosbar</td>
<td>16 bit</td>
<td>Central crossbar</td>
<td>DSP</td>
</tr>
<tr>
<td>PADDI-2</td>
<td>1992</td>
<td>Croosbar</td>
<td>16 bit</td>
<td>Multiple crossbar</td>
<td>DSP and others</td>
</tr>
<tr>
<td>DP-FPGA</td>
<td>1994</td>
<td>2-D array</td>
<td>1 &amp; 4 bit, Multi granular</td>
<td>Inhomogeneous routing channels Regular datapath</td>
<td></td>
</tr>
<tr>
<td>KressArray</td>
<td>1995</td>
<td>2-D mesh</td>
<td>Family: Select pathwidth Multiple NN &amp; bus segments (Adaptable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colt</td>
<td>1996</td>
<td>2-D array</td>
<td>1 &amp; 16 bit inhomogenous (Sophisticated) Highly dynamic recon.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RaPID</td>
<td>1996</td>
<td>1-D array</td>
<td>16 bit</td>
<td>Segmented bases Pipelining</td>
<td></td>
</tr>
<tr>
<td>Matrix</td>
<td>1996</td>
<td>2-D mesh</td>
<td>8 bit, Multi-granular NN, length 4 &amp; Global lines General purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAW</td>
<td>1997</td>
<td>2-D mesh</td>
<td>8 bit, Multi-granular 8NN switched connections Experimental</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Garp</td>
<td>1997</td>
<td>2-D mesh</td>
<td>2 bit</td>
<td>Global and semi global lines Loop acceleration</td>
<td></td>
</tr>
<tr>
<td>Pleiades</td>
<td>1997</td>
<td>Mesh/Cross</td>
<td>Multi-granular Multiple segmented crossbar Multimedia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PipeKensh</td>
<td>1998</td>
<td>1-D array</td>
<td>128 bit</td>
<td>Sophisticated Pipelining</td>
<td></td>
</tr>
<tr>
<td>REMARC</td>
<td>1998</td>
<td>2-D mesh</td>
<td>16 bit</td>
<td>NN &amp; full length buses Multimedia</td>
<td></td>
</tr>
<tr>
<td>MorphicSys</td>
<td>1999</td>
<td>2-D mesh</td>
<td>16 bit</td>
<td>NN, length 2 &amp; 3 global lines Not disclosed</td>
<td></td>
</tr>
<tr>
<td>CHESS</td>
<td>1999</td>
<td>Hexagon mesh</td>
<td>4 bit, Multi-granular 8NN and buses Multimedia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DReAM</td>
<td>2000</td>
<td>2-D array</td>
<td>8 &amp; 16 bit</td>
<td>NN, segmented buses New generation wireless</td>
<td></td>
</tr>
<tr>
<td>Chameleom</td>
<td>2000</td>
<td>2-D array</td>
<td>32 bit</td>
<td>Not disclosed tele &amp; data communication</td>
<td></td>
</tr>
<tr>
<td>MorphICs</td>
<td>2000</td>
<td>2-D array</td>
<td>not disclosed Not disclosed Next generation wireless</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 – Coarse grained architectures
Some architectures support a single size, while others support different data-path widths and others permit bundling several processing elements to compose a larger width data-path operators.

As always, the selection of the data-path width is a trade-off between flexibility and efficiency. While a larger width data-path is more efficient for long words it can also be area inefficient and lead to a processing overhead due to the need of managing bit segments on the data-path if most MSB bits are unused. Smaller width data-paths can lead to routing overhead due to the need of composing a larger data-path operator from several small data-path operators.

The reconfiguration model determines when a new configuration can be loaded into architecture. For architectures only supporting static configuration, the configuration is loaded at the beginning of the execution and stays the same during the computation phase. When a new configuration has to be loaded, the execution must stop. With dynamic reconfiguration, a new reconfiguration can be loaded while the application is running. This model includes the case in which the execution relies on the steady configuration of the processing elements.

IV. RECONFIGURABLE ARRAY ARCHITECTURE

The architecture currently under development is organized as one linear array of up to 16 reconfigurable ALU (rALU), as depicted on Figure 3. Each ALU is capable of supporting up to 16 different operations, thus a four MRAM bit configuration memory is associated to each ALU. It is the information stored on the configuration memory that selects which operation will be carried out.

For computation, there are four global data buses, two input data buses and two output data buses. The A bus and the B bus feed the rALU with its operands. The S bus and the Carry out bus deliver the result of the logical-arithmetic operation carried and its eventual carry out.

The exclusive access to the global data buses for each rALU is ensured by 16 selector signals, Select(k) for k = 0 till 15, one for each rALU.

For configuration, there is one input global bus, as depicted on Figure 4. The Conf bus carries the data that will be stored on the MTJ to be used later on for configuration, thus supporting shallow dynamic re-configurability.

Sixteen selector signals, EnableWrite(k) for k = 0 till 15, determine which rALU(k) configuration memory will be written.

V. CONFIGURATION MEMORY

The configuration memory contains the information that defines the ALU functionality.

As it is shown on Figure 5, the configuration memory is made of an array of identical cells.

Each Magnetic SRAM (MSRAM) is made of [1, 2, 3, 4]:

- Two MTJ cells (MTJ1 and MTJ2)
- Two unidirectional current sources implemented by the NMOS transistors NM4 and NM5. These current sources will deliver a current when a Write signal is on in both NMOS transistors. These two current sources are responsible for the Joule effect.
- A write line that is employed to propagated the magnetic field in both directions. This line is common to the whole array(s) of MSRAM.
- Two PMOS isolation transistors [4].

Figure 3 – General Computational Architecture

Figure 4 – Configuration Architecture

A. Unbalanced Flip Flop

The original Unbalanced Flip Flop [1, 2, 3] (UFF) is a simple, small and stable structure that is ideal as a sense amplifier. This structure is made of a net of five transistors, two PMOS (MP0-MP1) and three NMOS (MN0-MN2).

The original structure is enhanced with two extra PMOS transistors (MP2 and MP3) that isolate the
Unbalanced Flip Flop from the MTJ when the read cycle has finished [4].

During the read phase, the MN2 NMOS transistor acts as a short circuit, thus, the two crossed couple of inverters are pulled to a meta-stable operating state. At the same time, the PMOS transistors MP2 and MP3 allow different voltage values at each end of the crossed couple of inverters that will move away the operating point from one of the stable states and bring it closer to the other. Each of the aforementioned voltage values depends on the resistance that each MTJ offers.

So, when the Read signal is released, the structure will move to the closest stable state. Afterwards, new information can be stored into the MTJ without altering the value stored in the UFF.

The selector signals above control the multiplexers that define the logical-arithmetic behavior of the ALU. For the less significant bit of the ALU, the logical equations Sel4 and Sel5 are replaced by equations Sel4lsb and Sel5lsb respectively. This difference is due to the need to handle correctly the propagation of the carry bit.

The data-path unit is implemented in a bit slice fashion. Since this module is implemented in a custom mode, a bit slice style makes it easier to change the available data-path width.

The 4 bit ALU is made by the concatenation of 4 identical data-path slices as the one depicted on Figure 6. The less significant bit carry in is set to ‘0’ while the remaining carry in are connected to their direct predecessor lower order bit slice carry out.

On Table 3 are depicted the ALU available functions.

A. ALU control Unit

As it is shown on Figure 6, the function computed by the ALU depends on the values of the six multiplexers selector values (Sel0-Sel5). These values depend on the data stored in the memory configuration bits and their values are given by the following equations:

- \( Sel0 = \neg(Q1) \land \neg(Q2) \land \neg(Q3) \)
- \( Sel1 = Q2 \land \neg(Q3) \)
- \( Sel2 = (Q1 \land Q2) \lor (Q3 \land \neg(Sel4)) \)
- \( Sel3 = (Q0 \land \neg(Sel5)) \lor (Q3 \land \neg(Sel4)) \)

- \( Sel4lsb = \text{logic 0} \)
- \( Sel4 = \neg(Q1) \land \neg(Q2) \land \neg(Sel0) \)
- \( Sel5lsb = (Q0 \land \neg(Q1) \land \neg(Q2)) \lor Sel5 \)
- \( Sel5 = Q0 \land \neg(Q3) \land \neg(Sel1) \)

Table 3- ALU logical and arithmetic functions

<table>
<thead>
<tr>
<th>Configuration Memory Q3Q2Q1Q0</th>
<th>ALU Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>A and B</td>
</tr>
<tr>
<td>0001</td>
<td>A or B</td>
</tr>
<tr>
<td>0010</td>
<td>A xor B</td>
</tr>
<tr>
<td>0011</td>
<td>A xnor B</td>
</tr>
<tr>
<td>0100</td>
<td>A</td>
</tr>
<tr>
<td>0101</td>
<td>not A</td>
</tr>
<tr>
<td>0110</td>
<td>B</td>
</tr>
<tr>
<td>0111</td>
<td>not B</td>
</tr>
<tr>
<td>1000</td>
<td>A + B</td>
</tr>
<tr>
<td>1001</td>
<td>B - A</td>
</tr>
<tr>
<td>1010</td>
<td>Fault</td>
</tr>
<tr>
<td>1011</td>
<td>Fault</td>
</tr>
<tr>
<td>1100</td>
<td>Fault</td>
</tr>
<tr>
<td>1110</td>
<td>Fault</td>
</tr>
<tr>
<td>1111</td>
<td>Fault</td>
</tr>
</tbody>
</table>

Figure 6 – Data path bit slice structure

Figure 5 - Configuration memory structure
VII. SUPPORT CIRCUITRY

A. Configuration Memory Writing Control

Due to the number of arithmetic-logic functions available on each ALU, a cluster of four memory bits is assigned to each individual ALU.

As mentioned in section 5, a locally Joule effect current must run through each of the two MTJ of each memory cell. For this purpose, a control signal will turn on or off the NMOS transistors, MN4 and MN5 in Figure 5, depending on the direction of the current that flows through the write line and the information that needs to be written in a memory cell.

The writing cycle is done in two steps depending on the flow of current on the write line. In the first step the current flows in a direction that will allow writing a logical ‘1’ on a given memory cell. In the second step, the current that flows in the write line will reverse its direction in order to allow writing a ‘0’ in another memory cell.

Since the cluster size is small, the four control signals required are generated at the same writing cycle. The control signals for memory i and cluster j are given by the following equations:

\[ \text{Trig}(i) = ((\text{Write1} \text{ and Conf}(i)) \text{ or } (\text{Write0} \text{ and not}(\text{Conf}(i)))) \]

\[ \text{Write}(i)(j) = \text{Trig}(i) \text{ and } \text{EnableWrite}(j) \]

Write1 and Write0 are two external signals that flag the steps one and two respectively of the writing process. Conf(i) contains the value to be written on memory cell i, EnableWrite(j) - flags the intention to change the contents of the memory cells of cluster j.

Since, there are 16 rALU and associated memory clusters, the EnableWrite(j) signal is given by a 4x16 bits decoder with an enable signal. This decoder is fed by a 5 bits input port.

B. ALU I/O and I/O control

For the purpose of this prototype, every ALU will share four data buses: two input buses and two output buses. The A bus and the B bus feed each rALU with its operands while the S bus transfers the result of the logic-arithmetic operation performed on each rALU and the Carry out bus transfers the eventual carry out generated.

For the purpose of buses sharing there are transmission gates placed at each rALU data inputs and data outputs. Thus, there are 16 selector signals, Sel(k) for k = 0 till 15, one for each ALU. These signals are responsible for handling the I/O buses sharing among the 16 available rALU. The selector signals, Sel(k), are generated by a 4x16 bit decoder with an enable signal. This decoder is fed by a 5 bit input port.

C. Multi-context switch control

For multi-context switching, the four MRAM of each rALU configuration memory are switched simultaneously. Since the prototype will support up to 16 rALU, there are 16 different control signals, Read(k) for k = 0 till 15, to trigger the multi-context switching (reading cycle) for every rALU. These 16 control signals are generated by a 4x16 bit decoder with an enable signal. This decoder is fed by a 5 bit input port.

VIII. CONCLUSIONS

A proof of concept coarse-grained reconfigurable array using MRAM is under development. The reconfigurable coarse-grained array offers significant advantages compared with fine-grained solutions such as FPGA.

In order to fulfill our goal several architectural and circuits solutions are being studied, implemented and evaluated. Some of the core blocks have already been designed and tested pre and post-layout.

IX. ACKNOWLEDGMENTS

This work is being developed with a research grant from Fundação para Ciência e Tecnologia (FCT) with the reference PTDC/EEA-ELC/72933/2006.

REFERENCES