Performance Evaluation of RAM-Based Implementation of Finite State Machines in FPGAs

R. Senhadji-Navarro
Dpto. de Arquitectura y Tecnología de Computadores
Universidad de Sevilla
Email: raouf@us.es

I. García-Vargas
Dpto. de Arquitectura y Tecnología de Computadores
Universidad de Sevilla
Email: iggv@us.es

J.L. Guisado
Dpto. de Arquitectura y Tecnología de Computadores
Universidad de Sevilla
Email: jlguisado@us.es

***Abstract*— This paper presents a study of performance of RAM-based implementations in FPGAs of Finite State Machines (FSMs). The influence of the FSM characteristics on speed and area has been studied, taking into account the particular features of different FPGA families, like the size of LUTs, the size of memory blocks, the number of embedded multiplexer levels and the specific decoding logic for distributed RAM. Our study can be useful for efficiently implementing FPGA-based state machines.**

I. Introduction

Memory-based architectures are receiving considerable attention as an effective means to implement Finite State Machines. This growing interest has been motivated by the inclusion of a large amount of memory in modern Field Programmable Gate Array (FPGA) devices. An example of this interest is the considerable effort expended by the manufacturers in providing tools to map logic into embedded memory blocks [1]. In recent years, new techniques have been developed to efficiently exploit FPGA memory resources to implement state machines, improving the performance in terms of speed, area or power consumption [2], [3], [4], [5], [6]. Examples of such techniques are functional decomposition [2] and FSMs with input multiplexing (FSMIM) [5]. In [6], we showed that FSMIM implementations are faster than conventional LUT-based implementations in the 81% of the used testbenchs.

On the other hand, RAM-based architectures offer a promising way to support run-time modifications of the state machine in order to adapt its behavior to different execution contexts. These run-time modifications are easily carried out by means of write operations because the functionality of the state machine is defined by the data stored in the RAM. The main advantages of this kind of reconfiguration over Dynamic Partial Reconfiguration (DPR) are less reprogramming cost, independence of placement and routing, and possibility to be applied to FPGAs without DPR capabilities [7], [8], [9].

In this paper, a performance evaluation of RAM-based FSM implementations in FPGAs has been presented. We have studied the impact of the characteristics of FSMs on the speed and the area of the resulting RAM-based implementations. In Section 2, we give a background on RAM-based FSM architectures. In Section 3, we study memory implementations in FPGAs. Finally, Section 4 presents experimental results.

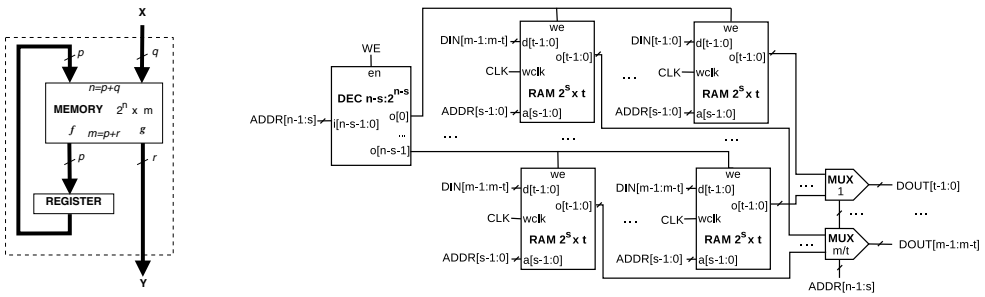
II. RAM-based architectures for implementing state machines

In logic design, a state machine with q inputs and r outputs can be defined as a 4-tuple (S, s_0, f, g) where S is a set of states; $s_0 \in S$, the reset state; $f : S \times \{0,1\}^q \rightarrow S$, the transition function; and g, the output function. In Mealy state machines, the output function depends on the states and the inputs ($g : S \times \{0,1\}^q \rightarrow \{0,1\}^r$). In Moore state machines, the output function only depends on the states ($g : S \rightarrow \{0,1\}^r$).

The implementation of state machines can be done by using different kind of resources. An alternative to standard implementations based on logic cells consists of the use of memory to implement the transition and output functions [5]. In the present study, we assume the more general Mealy machines. Fig. 1(a) shows the block diagram of a memory-based Mealy FSM implementation. The memory generates the output and the next state from the input and the present state. The present state is fed back from the register, and together with the inputs, it composes the memory address signal.

In conventional applications, the memory used to implement a state machine can be a ROM memory. However, in run-time reconfigurable applications, a RAM memory must be used in order to add reconfiguration capabilities to the FSM. In this kind of applications, the write port of the RAM is used to modify the functionality of the state machine when execution context changes. The width of the read and write ports do not have necessarily to be the same. However, design tools do not infer memories with such asymmetric port widths. In order to overcome that lack, we have proposed in [9] a HDL description of this kind of memories.

In a RAM-based FSM implementation, the amount of memory required grows exponentially with the number of inputs and the number of state encoding bits. To overcome this problem different approaches based on the use of additional logic together with the memory blocks have been proposed. Different contributions have shown that this strategy is a successful way to improve the performance in speed and area over conventional RAM-based FSM implementations [2], [5]. In [6], the experiment results showed that the FSMIM approach reduces by 87% the average number of memory



(a) Architecture (b) Implementation of a memory in FPGA devices

Fig. 1. Memory based-implementation of state machines

blocks using less than 5 additional LUTs on average (in LUT-based implementations, this value up to 220). In addtion, these implementations are faster in 59% of cases.

III. Memory implementations in FPGAs

In FPGA devices, a RAM memory (hereinafter called logical RAM) is implemented by joining together smaller memory resources (hereinafter called memory elements). The interconnection between different memory elements is made by using a decoder and a set of multiplexers. In a write operation, the decoder enables the write signal of the memory elements that contain the word to write. Note that in a logical ROM memory the decoder does not exist. In a read operation, the multiplexer allows the selection of the output data of the memory elements that contain the word to be read. Fig. 1(b) shows a general scheme of a $2^n \times m$ logical memory made by using $2^s \times t$ memory elements. The memory elements are organized as a matrix with 2^{n-s} rows and $\frac{m}{t}$ columns. A $n-s : 2^{n-s}$ decoder and $\frac{m}{t}$ multiplexers of 2^{n-s} inputs and t-bit width are required to interconnect the memory elements. Given a logical RAM, the complexity of the decoder and multiplexers grows with the decrease of depth of the memory elements (i.e., with the decrease of s). As shown by experimental results, the complexity of these elements has a significant influence on the performance of the logical RAM.

Most of the FPGA manufacturers (like Xilinx and Altera) offer two kinds of memory elements in the same device. One of them consists of Look-Up Tables (LUT) configured as little RAM memories (in this case, the logical memory is called distributed RAM). Another kind of resources are RAM blocks integrated on the FPGA device as coarse-grain resources (in this case, the logical memory is called block RAM).

A distributed RAM allows for asynchronous read operations, whereas a block RAM only allows for synchronous read operation. Mealy architecture (shown in Fig. 1(a)) is based on an asynchronous memory. So, in order to use RAM blocks to implement it, FSM implementations with synchronized outputs are considered. In this case, both the outputs and the present state are registered.

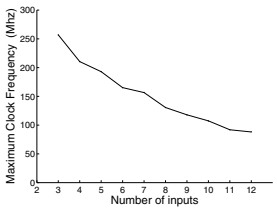
RAM blocks can be configured with different aspect ratios (i.e., the ratio between depth and width). Let us call vertical slicing the mapping of a logical RAM to memory blocks with the maximum depth (without exceeding the logical RAM depth). Let us call horizontal slicing the mapping of a logical RAM to memory blocks with the maximum width (without exceeding the logical RAM width). Vertical slicing allows a reduction of multiplexers complexity in RAM with sizable memory depth, improving speed. However, vertical slicing are power-inefficient because each access to logical RAM activates more memory blocks than in horizontal slicing [10].

The main advantage of block RAMs is higher performance over distributed RAMs in sizable memories. This is due to the fact that large memory elements reduce the amount of logic to interconnect them, providing better performance. However, FPGA manufacturers must find a trade-off between the advantage of large memory blocks and the disadvantage of a reduced number of blocks. Large memory blocks can cause a significant waste of memory that could be required for other purposes (let us call it memory fragmentation). The most recent devices allow to use each memory block as two independent half-size memory blocks in order to reduce the memory fragmentation.

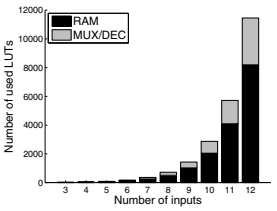
The main advantage of the distributed RAM is the flexibility to define the number of ports and the aspect ratio. An additional advantage of distributed RAM is asynchronous read operation. This capability can be used to implement state machines with unsynchronized outputs required in some practical applications. In some RAM-based FSM approaches, asynchronous read operation is also required like those described in [7]. In the presented study we have used both kinds of memory resources.

IV. Experimental results

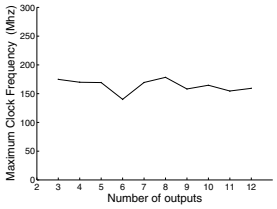
In order to study the influence of the FSM parameters on area and speed, we have implemented in Xilinx FPGAs the RAM-based FSM architecture with different values for their parameters; i.e., number of inputs (q), number of outputs (r), and number of state encoding bits (p). From a qualitative point of view, the behavior of the parameters is the same independently of the used FPGA device. The quantitative differences observed are due to the specific device features such as the size of LUTs, the size of memory blocks, the number of embedded multiplexer levels or the specific decoding logic for distributed RAM, among others. Fig. 2 and Fig. 3 show the implementation results in a xc3s5000-4 FPGA obtained by using distributed RAM and block RAM, respectively. These



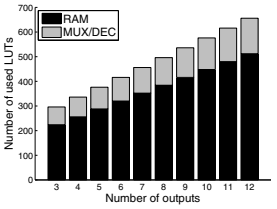
(a) Influence of the number of inputs on speed



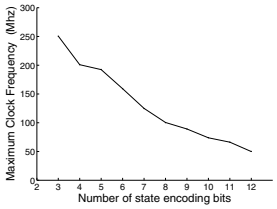
(b) Influence of the number of inputs on area



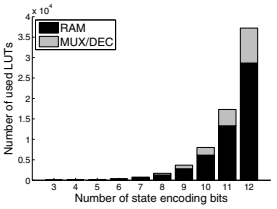
(c) Influence of the number of outputs on speed



(d) Influence of the number of outputs on area



(e) Influence of the number of state encoding bits on speed



(f) Influence of the number of state encoding bits on area

Fig. 2. Speed and area results of distributed RAM

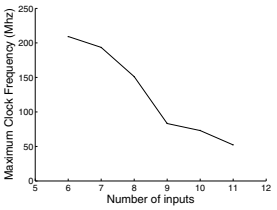
results have been used as reference to study the influence of the parameters. However, we provide additional results in order to characterize RAM-based implementations in different FPGA families. The used FPGA device has 4-LUTs and 18-Kbits RAM blocks. We have used vertical memory slicing for block RAMs. The write port of the RAM used to implement the FSM allows to reconfigure it at transition-level. In all the studied cases, the speed analysis showed that the critical path includes the multiplexer bank that involves read operations. This means that the speed results are also valid for ROM implementations.

A. Distributed RAM results

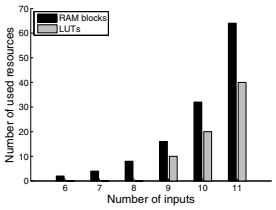
The distributed RAM results show that the speed decreases significantly with the number of inputs (q) (Fig. 2(a)) and the number of state encoding bits (p) (Fig. 2(e)). The reason is that the growth of the logical RAM depth increases the complexity of multiplexers. An increment of one unit in the value of any of these parameters requires to duplicate the number of inputs of multiplexers, adding a new logic-level composed by embedded multiplexers. If it is not available, a new LUT-level must be used instead, but producing a higher speed reduction. For example, in Fig. 2(a) a 5% speed reduction is observed when q changes from 6 to 7. But when q changes from 7 to 8, a new LUT-level is required so that a 17% speed reduction is now observed. In contrast, the number of outputs (Fig. 2(c)) does not have influence on speed because memory depth remains constant (r only has influence on memory width). The little differences observed are due to place-and-routing.

The area results (Fig. 2(b), Fig. 2(d) and Fig. 2(f)) show the number of LUTs used as memory elements and as logic to interconnect them (i.e., decoder and multiplexers). The q parameter causes an exponential growth because each new input implies to duplicate the number of LUTs used as memory, the number of inputs of the multiplexers and the number of outputs of the decoder. The r parameter do not affect to the complexity of multiplexers. Each new FSM output only supposes a new LUT per each row of memory elements and a new multiplexer. Thus, the number of LUTs has a linear increase. In the case of the p parameter, both effects are overlapped because this parameter has influence on both the depth and the width of the logical memory. This explains the quantitative differences between Fig. 2(b) and Fig. 2(f).

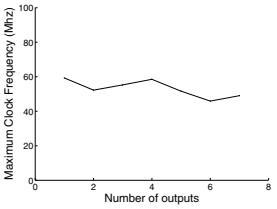
Comparatively, Virtex-5 requires a significantly less number of LUTs than the previous devices mainly due to the availability of LUTs with 6 inputs instead of 4 (this strategy has also been followed by other manufacturers such as Altera). For example, in Virtex-5 the number of LUTs required on average is a 80% less than in Spartan-3. Despite the fact that the architecture of Virtex-2 is similar to that one of Spartan-3, Virtex-2 requires on average a 75% less of LUTs to implementent the interconnection logic. The difference lies



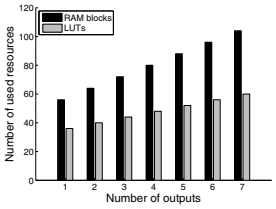
(a) Influence of the number of inputs on speed



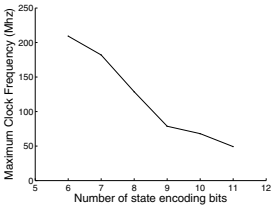
(b) Influence of the number of inputs on area



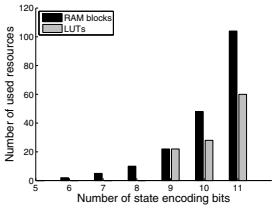
(c) Influence of the number of outputs on speed



(d) Influence of the number of outputs on area



(e) Influence of the number of state encoding bits on speed



(f) Influence of the number of state encoding bits on area

Fig. 3. Speed and area results of block RAM

in the fact than Virtex-2 allows to configure all LUTs of a Configurable Logic Block (CLB) as memory; however, Spartan-3 (and the other Xilinx FPGA devices) allows to configure only the half number of LUTs. Thus, Virtex-2 offer specific logic to decode 3 address lines (instead of 2), reducing the amount of LUTs required to implement the decoder.

B. Block RAM results

Block RAM results show a significant speed reduction with q and p. The used vertical memory slicing has allowed to reduce the complexity of the multiplexers in logical RAM, improving the speed. However, in the cases in which the logical memory depth is greater than the maximum block depth, LUTs are required to implement multiplexers. For example, the cases with $q < 9$ do not require LUTs (as we can shows in Fig. 3(b)). As can be observed, even if LUTs has not been used, the speed is reduced due to the routing overhead. The first case in which LUTs are required ($q = 9$) presents the worse speed degradation with a speed reduction ratio of 45%. In low-power design, the threshold value when LUTs are required is lower because horizontal slicing must be used. In the case of r, Fig. 3(c) shows a negligible reduction of the speed (in all FPGAs, the slope of regression line is less than 2 MHz per output). This is due exclusively to the routing since the complexity of the multiplexers is not affected by r.

The area results (Fig. 3(b), Fig. 3(d) and Fig. 3(f)) show the number of used memory blocks and LUTs. Both resources have an exponential growth with q and p parameters; and a linear behavior with r. Comparatively, Virtex-5 (and more recent families) use less resources mainly due to the availability of 36-Kbits (instead 18-Kbits) RAM blocks.

V. Conclusion

We have presented a performance evaluation of RAM-based FSM implementation in FPGAs. The influence of the FSM parameters on speed and area has been studied, taking into account the specific characteristics of different FPGA families, like size of LUTs, size of memory blocks, number of embedded multiplexer levels and specific decoding logic for distributed RAM.

Our experimental results show that the number of used resources increases exponentially with the number of inputs and the number of state encoding bits of an FSM. Accordingly, the maximum clock frequency suffers a significant degradation with these two parameters. For block RAMs, the maximum reduction of speed is obtained in those cases in which the interconnection between different RAM blocks requires the use of LUTs. In contrast, the number of used resources increases linearly with respect to the number of outputs, and the speed is not affected. We have also shown that the specific characteristics of different FPGA families have a significant influence on the performance. For example, on average the implementation of a distributed RAM in 6-LUT FPGAs requires an 80% less LUTs than in a 4-LUT FPGAs.

In the FPGA market a growth trend in the size of embedded memory blocks is currently observed. In addition, mechanisms to combine two adjacent blocks without using LUTs have been provided. Both aspects help to increase the performance of RAM-based FSM implementations, because they contribute to reduce the number of LUTs needed to interconnect memory blocks. However, even if LUTs are not required, the routing for interconnecting the memory blocks can degrade speed. For example, in Virtex-4 (with 18-Kbits memory blocks including the mechanism to combine two adjacent blocks), a speed reduction of 49% has been observed when the number of FSM inputs is doubled, due exclusively to the routing (without using any LUTs). Therefore, despite of remarkable efforts of the manufacturers to increase the performance of memory resources, complex state machines require suitable approaches to reduce the amount of used memory and to improve the operation speed, like FSMIMs or functional decomposition. This is specially important in low-power applications, which require horizontal memory slicing. A different approach to improve the performance of the RAM-based FSM implementation has been recently proposed by the authors [11]. A Finite Virtual State Machines (FVSM) consists of a memory hierarchy that allows enhancing the speed of FSMs with large number of states. The results showed that the FVSM implementations are at least a 33% faster than conventional RAM-based FSM implementations in almost half of the cases, including cases with speed improvement ratio above 80%.

The results presented in this work can help the designers to improve the performance of FPGA-based implementations of state machines and to select the most convenient target device.

References

[1] G. Chiu *et al.*, "Mapping arbitrary logic functions into synchronous embedded memories for area reduction on fpgas," in *IEEE/ACM Int. Conference on Computer-Aided Design*, 2006, pp. 135 –142.

[2] H. Selvaraj *et al.*, "Fsm implementation in embedded memory blocks of programmable logic devices using functional decomposition," in *Information Technology: Coding and Computing, International Conference on*, april 2002, pp. 355 – 360.

[3] M. Rawski *et al.*, "Logic synthesis method of digital circuits designed for implementation with embedded memory blocks of fpgas," in *Design of Digital Systems and Devices. Lecture Notes in Electrical Engineering*, vol. 79, 2011, pp. 121–144.

[4] A. Janarthanan *et al.*, "Power-efficient fsm mapping in fpgas through semb dormancy control," in *Circuits and Systems, 2007. MWSCAS 2007. 50th Midwest Symposium on*, Aug. 2007, pp. 502–505.

[5] R. Senhadji-Navarro *et al.*, "Rom-based fsm implementation using input multiplexing in fpga devices," *Electronics Letters*, vol. 40, no. 20, pp. 1249–1251, Sept. 2004.

[6] I. Garcia-Vargas *et al.*, "Rom-based finite state machine implementation in low cost fpgas," in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, June 2007, pp. 2342–2347.

[7] V. Sklyarov, "Reconfigurable models of finite state machines and their implementation in fpgas," *J. Syst. Archit.*, vol. 47, pp. 1043–1064, 2002.

[8] M. Koster and J. Teich, "(self-)reconfigurable finite state machines: theory and implementation," in *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, 2002, pp. 559 – 566.

[9] R. Senhadji-Navarro *et al.*, "Fpga-based implementation of ram with asymmetric port widths for run-time reconfiguration," in *Electronics, Circuits and Systems, IEEE Int. Conf. on*, Dec. 2007, pp. 178 –181.

[10] R. Tessier *et al.*, "Power-efficient ram mapping algorithms for fpga embedded memory blocks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 278 –290, 2007.

[11] R. Senhadji-Navarro *et al.*, "Finite virtual state machines," *IEICE Transactions on Information and Systems*, vol. E95-D, no. 10, Oct. 2012.