Efficient Implementations of Multi-pumped Multi-port Register Files in FPGAs

Hasan Erdem Yantır, Salih Bayar, Arda Yurdakul
Computer Engineering, Boğaziçi University
P.K. 2 TR-34342 Bebek, Istanbul, TURKEY
Phone: +90 212 359 7780, Fax: +90 212 287 2461

{hasanerdem.yantir, salih.bayar, yurdakul}@boun.edu.tr

***Abstract*—Existing implementation methods of multi-port register files (MPo-RF) in FPGAs are not scalable enough to deal with the increased number of ports due to higher logic area and power. While the usage of dedicated block RAMs (BRAMs) limits the designer to use only single read and single write port, slice based approach causes large resource occupation and degrades design performance significantly. Similarly, the conventional multi-pumping (MPu) approaches are not efficient enough due to increased combinational delay and area of huge multiplexers. In this paper, we propose a new design which exploits the banking and replication of BRAMs with efficient shift register based multi-pumping (SR-MPu) approach. While increased port number causes internal frequency drops in conventional multiplexer based MPu approaches, it does not affect internal operating frequency of our SR-MPu methodology. Test results on Xilinx Virtex-5 XC5VLX110T FPGA show that our 32-bit 12-read & 6-write (12R&6W) RF can operate internally up to 429 Mhz while 64-bit version up to 408 Mhz. The speed of our RF is independent from MPu factor and occupies lower logic resources up to 47% when compared with other design methods. In terms of energy consumption, our RF design saves energy up to 26% according to the Xilinx Power Analyzer (XPA) results.**

I. Introduction

Technology trend shifts towards to the parallel computational platforms on both embedded systems and personal computers. Nowadays, the most of embedded applications require to process more data per unit time. To meet the requirements of such data intensive applications, the usage of parallel platforms are unavoidable. FPGAs are great candidates of parallel platforms for the implementation of such complex applications because of their fully parallel and reconfigurable architectures. FPGAs also have built-in dedicated cores such as hardware multipliers, memory blocks, DSP blocks, Digital Clock Managers (DCMs) and even dual-core processors [1] to achieve more speed-up in data intensive applications. Since FPGAs are almost fully reconfigurable by their nature, different types of digital circuits can be designed and combined at any time. Hence, designing a fast reconfigurable RF with different depth, width, speed and number of ports can be easily done on an FPGA.

The demand to process more data per unit time requires multiple read and write operations at a time, which can be achieved by the usage of MPo-RFs instead of single port RFs (SPo-RF). For example, a four-issue VLIW processor requires at least 8 read and 4 write ports if an instruction consists of 2 read and 1 write operations as it is in [2]. As the issue-width of a VLIW processor increases, to meet the data rate requirements the number of read and write ports on the RF have to increase accordingly.

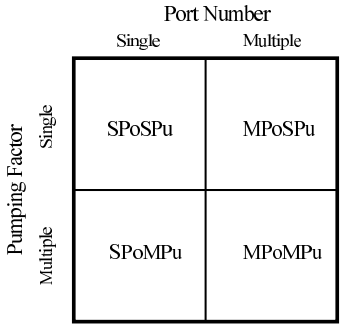


Fig. 1: Register File Taxonomy

There are different ways in designing of MPo-RFs in FPGAs. The first method is utilizing memory capabilities of slices that are available on an FPGA. The study [3] gives an example of a slice-based multi-port memory with one write and three read ports. However, this design is not scalable at all because the combinational and wiring delays of slices dominate as the number of ports increases. The second method relies on the basis of combining on-chip block RAMs (BRAM) that are available in FPGAs. There are various studies based on this approach: [4] [5] [6] [7] [8] [9].

Multi-pumping (MPu) might also be used for designing MPo-RFs. Multi-pumping is setting the RF internal operating frequency by the times of system external operating frequency. So, multiple read and write operations in memory can be carried out in a time-shared manner. The ratio between RF internal operating frequency and system external operating frequency is defined as multi-pumping factor (MPuF). For example, when the MPuF is set to two on a SPo-RF, then the obtained SPo-RF behaves as if it has two physical read and write ports. Here, write and read operations are executed at twice the speed of the external system operating frequency. The conventional multi-pumping approach is based on the utilizing multiplexers and demultiplexers at the inputs and

outputs of an RF as described in [7] and [4]. Yet, this approach is also not scalable because of the increased combinational delay in multiplexers and demultiplexers when the number of read- and write ports increases.

Multi-pumping can be applied not only to SPo-RF but also to MPo-RF. When no multi-pumping performed on SPo-RF and MPo-RF, then they can be named as single-port single-pumped (SPoSPu) and multi-port single-pumped (MPoSPu) respectively. When it is applied to SPo-RF and MPo-RF, the resulting designs are called single-port multi-pumped (SPoMPu) and multi-port multi-pumped (MPoMPu) accordingly. To make it more comprehensible, we may classify RFs with different port sizes and MPuFs by constructing an RF Taxonomy (see Fig. 1) like Flynns' [10], which is dedicated for the classification of computer architectures. Here, note that after applying multi-pumping to SPoSPu (named as SPoMPu), it becomes a multi-ported RF. So, according to our taxonomy, all three MPoSPu, SPoMPu and MPoMPu are multi-ported while SPoSPu remains as a single-ported RF.

In this paper, we focus on efficient implementations of multi-ported RFs with exploting a new multi-pumping approach called shift register based multi-pumping (SR-MPu). Hence, we can summarize our contributions as follows:

- Proposing a new multi-pumping approach for the design of MPoMPu-RFs files in FPGAs. This approach makes the operation frequency of the register file independent from MPuF. For example, even when MPuF is 10, the degradation on the operation frequency of an MPo-RF design with SR-MPu approach is around 2.5%. With the same MPuF, the degradation in the operation frequency of MPo-RF with the Mux-MPu is more than 40%. With our SR-MPu method, it is also possible to design about 20% smaller MPoMPu-RFs than the Mux-MPu adopted in LaForest's studies [7] [4].
- Combination of non-multi-pumped MPo-RF and the new SR-MPu approach to create an "emulated multi-port" register file that can operate at higher frequencies and occupy fewer resources.
- This new MPoMPu-RF design consumes up to 25% less energy when compared with its counterparts.
- The developed multi-pumping methodology can be adapted to other kind of designs to multi-pump them effectively.

The rest of the paper is organized as follows: In the following section, we mention about the studies in the literature. Section III gives methods of designing MPo-RFs by using dual port BRAMs in FPGAs. In Section IV, how multi-pumping can be used with multi-ported register files so as to obtain smaller register files with increased number of ports is explained. The method described in this section is adopted in [7] and [4]. In Section V, we describe our SR-MPu method that can be safely used in aggressively multi-pumped circuits. Results are discussed in Section VI. The final section concludes the work.

II. Related Work

The MPo memory implementation in FPGA is one of the most resource consuming part of a design. The usage of BRAMs inside the FPGAs is a well-known methodology to implement RF. Nios II, MicroBlaze and Picoblaze soft-core processors use BRAMs to implement an RF with 2R&1W by replication. In addition, Xilinx synthesis tool (XST) provides automatic replacement for multi-read and one-write RFs in this manner [11]. When more ports are required, generally the MPo-RF is implemented by using FPGA slices. Sagmir et al. [5] suggest a method to build MPo-RFs by using BRAMs that supports not only multi-read but also multi-write. As mentioned in Section III, BRAMs can be connected to each other by replication and banking. However in this method, an RF is partitioned between register banks. Multiple-read can be done from the same bank, but two or more write operations cannot be done to the same bank. In order to handle this problem, a set of methods were proposed by Anjam et al [6]. In these methods, registers trying to access the same bank at the same time are renamed and directed to different banks. These operations are handled by the compiler and the assembler. In [7], this problem solved by a live value table (LVT) that holds the ID of the most recently updated bank. During a read operation, the most recent value is selected by LVT outputs, and it is directed to the output of the MPo-RF. However, LVT, which also has the same number of RW ports with the RF, is implemented by utilizing slices. Hence, this method comes with additional resource occupation and propagation delay due to LVT. Moreover, as total number of banks and the size of the RF increases, the speed of the design decreases and its resource usage increases.

In [12], a method is suggested by Xilinx for doubling total port number of an RF by multi-pumping. In this design, the ports of a true dual-port memory are multiplexed by two. Hence, there are two write and two read ports. Dedicated registers are used for reading each read port of the multi-ported memory. Having a separate register for each read is not a feasible design choice for higher MPuF due to long wiring delays. In [7] multiplexers are connected to write ports and demultiplexers are connected to read ports. However the size and the propagation delay of the MPo-RF increases as MPuF increases. Hence, this design approach is also not scalable. A recent study by Canis et al. [13] exploits the multi-pumping for resource sharing in high-level synthesis. However, they have used multi-pumping for computational units with at most MPuF two and not conducted on aggressive multi-pumping. In their design, while multiplexers are used to feed the multi-pumped circuit at the input, shift registers are used to get the outputs.

In [7] and [4], write-after-read strategy is adopted in multi-pumped memory locations if simultaneous read and write to the same location occur. Even though this strategy might be beneficial in multi-processor architectures that use the same memory for communication, this is not true for VLIW processors where compilers determine the access order of data-

path operations to the RF. Even the basic compilers handle all kinds of data dependencies so carefully that a read and write to the same location never occur in the same clock cycle [2]. Therefore in our multi-pumped register files, we did not consider to take any precautions to cope with this phenomenon. A similar approach has been taken in [6] for MPo-RFs.

III. Multi-Port RF Designs in FPGA

In traditional FPGA architectures, there exist dedicated BRAMs for storage [14] [15]. As an example, FPGA used for this study (Xilinx Virtex-5 XC5VLX110T) includes total 148 BRAMs. In Virtex-5, each BRAM can store up to 36 Kbits of data and can be configured as either two 18 Kb RAMs or one 36 Kb RAM. However these BRAMs have only two ports which are used either as write or read port depending on the BRAM mode (true or simple dual port). In true dual port mode, the BRAMs have two ports. Each port can change its behavior (read or write) during run time i.e. 2R or 1R&1W or 2W. In this configuration each register can store maximum 32-bit wide data with four bits parity and 1024 locations. In simple dual port mode, a BRAM has exactly one dedicated port for reading and another dedicated port for writing. Moreover, this configuration cannot be changed during run time. In this mode, each register can store 64-bit wide data with eight bits parity and 512 locations.

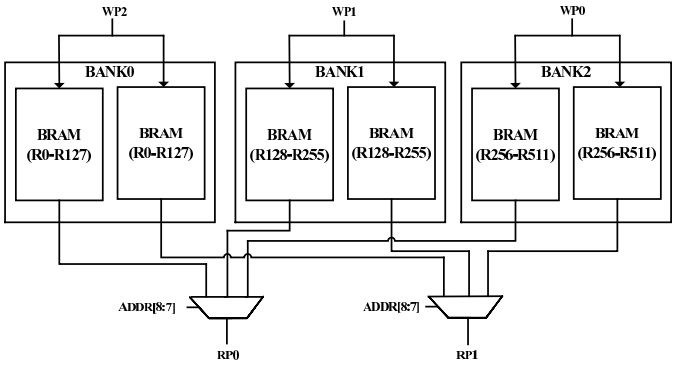


Fig. 2: MPo-RF with three write and two read ports (2R&3W)

In MPo-RF design with BRAMs, two common methods are replication and banking. XST supports generation of multi-read and one-write RFs by using only replication [11]. To increase the number of write ports, both replication and banking can be exploited as Sagmir et al. suggested [5]. In this approach, BRAMs are grouped by replication and form a bank. In a bank, all of the BRAMs contain the same data, i.e., the same data are written to all BRAMs inside a bank. Number of read ports can be increased by adding extra BRAMs to each bank. Number of write ports can be increased by adding extra banks. Figure 2 illustrates how six such BRAMs can be used to implement 512×64-bit RF with 2R&3W ports. In the example, there are three write ports so there should be three banks. The first bank holds the data between 0 and 127 and the second bank holds 128-255 and the third bank 256-512. However, though each BRAM has 512 locations, only the first 128 locations are used in the first and second banks, and 256 locations are used in the third bank. For the third bank, the range between 0-255 correspond to the range between 256-512 in the designed MPo-RF. During a write operation, the value is written to the corresponding register bank. However it is not possible to realize more than one write operations to the same bank at the same time. All write operations should be mutually exclusive with respect to the associated bank. This problem can be handled by software or hardware mechanisms as explained in Section II. In our approach, we assume that this issue is resolved by the compiler.

IV. Mult-port Multi-pump RF Designs in FPGAs

A. About Multi-pumping

MPu is a method used in digital circuits. This method means illustrating one resource as if there exist multiple replications of this resource. In a multi-pumped design, circuits behave as if they have multiple replicas of themselves, while the internal multi-pumped circuit frequency is by the times of external operating system frequency. Thus, a multi-pumped circuit is able to make multiple operations in one clock cycle time of the outer circuit. DDR2 and DDR3 synchronous dynamic random access memories also exploit the MPu to increase the data rate [16].

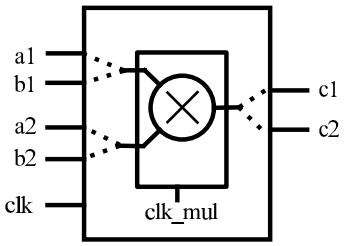


Fig. 3: A Basic Multi-pumped Circuit

Figure 3 illustrates the most fundamental working principle of MPu approach. The inner circuit in the figure is a multiplier driven by clock clk_mul. Outer circuit is a computation unit that takes two pair of arguments and gives multiplication result of each pair. Outer circuit uses inner multiplication circuit for multiplication operation and clk is the clock of the outer circuit. The inner multiplication unit works two times faster than outer circuit that uses this inner multiplication unit and computes the data in order. At the first clock cycle $a1 \times b1$ operation is done and in the second cycle $a2 \times b2$ operation is done and overall time that takes to make this computations lasts one clock cycle for outer circuit. For this reason, the outer circuit that uses this multiplication unit supposes there are two units although there exist only one multiplication unit. In order to provide input from outer circuit to inner circuit, some phase difference is required between two clocks. Shorty, in MPu approach internal ports run at faster frequencies than the external ports and the ratio of these two frequencies gives MPuF. In the figure, the MPuF of the design is two (i.e. $f_{clk\_mul} = 2 * f_{clk}$).

Here it is worth to mention about the reason why we utilized MPu in FPGAs. Almost all FPGAs include some coarse-grain configurable units implemented as ASIC such as BRAMs, hardware multipliers, DSP blocks, etc. It is a rule that these parts of FPGA are faster than the fine-grain reconfigurable parts which are used for implementing functionality [17]. Reconfigurable area is for general purpose and includes combinational logics, connection paths and routing matrices. However ASICs are designed for application specific purposes and are not suitable for generic usage. In other words, ASIC outperforms the FPGA because of its custom design for the dedicated purpose. Most likely BRAMs should run faster than any implemented design on reconfigurable area. For this reason, we aimed to gain advantage of speed difference between BRAMs and logic by utilizing an efficient MPu to MPoSPu-RF. The resulting design is MPoMPu-RF with much more ports than MPoSPu-RF.

In our MPoMPu-RF design, MPu circuit behaves as an interface between outer circuit (e.g. a processor or a set of computational units) and MPo-RF as mentioned in Section III. The term processors will be used to refer the outer circuit from now on. Here multi-pumped means driving the register file with a clock speed which is multiple of the processor speed. In the operation, the processor sends all of the read and write requests at the same time and waits for its one clock period. At the same time memory takes the requests and process them in order at a speed of its internal frequency. After the processor period ends, all of the operations inside the memory are handled, they are ready at the output and all changes are made in the RF. This process takes multiple cycles. The formula of MPuF is given in Equation 1.

$$MPuF = \frac{Processor\ \ Period}{RF\ \ Period} \tag{1}$$

B. Multiplexer based Multi-pumping

MPo-RF implementation with multi-pumping is done with multiplexers to direct the signals of RF to BRAM and demultiplexers to take the outputs as shown in Figure 4. In the figure, base MPo-RF consists of k read and m write ports. To increase the number of ports, a multiplexer of n ports is connected to each of the "read_address" (*addr_RF_Rx*), "write_address" (*addr_RF_Wx*) and "read_data" (*data_out_RF_Rx*) ports. Similarly a demultiplexer of n ports is connected to each of the "write_data" (*data_in_RF_Wx*) ports. In this way, we obtain a MPoMPu-RF with $k \times n$ read and $m \times n$ write ports. In this design, S is the common select input for all multiplexers with the size of $\lceil log\_2(n) \rceil$ where n is MPuF. Hence, in one clock cycle of the outer circuit, one can carry out $k \times n$ read and $m \times n$ write operations by incrementing S at each clock cycle of the inner circuit. In each cycle of the inner circuit, k read and m write operations are carried out through the ports selected by the S input of the RF. If internal operating frequency of multi-pumped RF is f Mhz and MPuF = n, then the speed of the outer circuit must be at most f/n. However, the architecture in Figure 4 is not scalable with MPuF. In order to implement a multiplexer, built-in multiplexers inside the carry chain and look up tables are used in FPGAs. As the number of inputs in multiplexers increases, the propagation and wiring delays due to multiplexers and demultiplexer get so long that operating frequency of the inner circuit degrades dramatically. Hence, as a designer tries to increase the port size by solely increasing MPuF, f decreases significantly and this results in a drastic degradation in the speed of the outer circuit. This fact is also pointed out in LaForest's work [7].

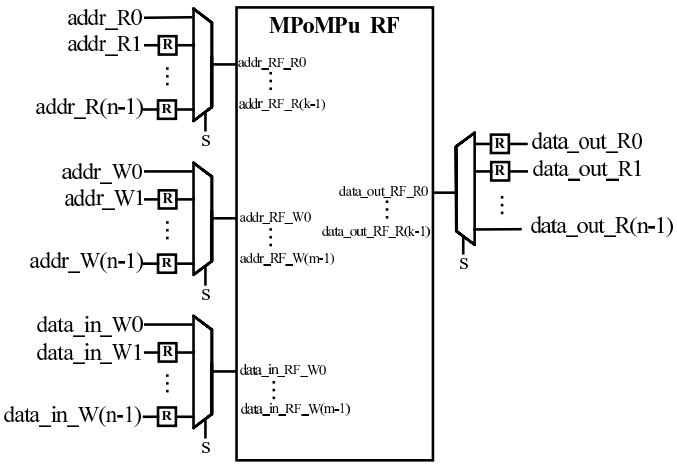


Fig. 4: Multiplexer Based MPoMPu-RF Design

V. Shift Register Based Multi-pumping

In order to eliminate the disadvantages of multi-pumping method explained in prior sections, we propose a new multi-pumping method for FPGAs. Figure 5 shows the design of shift register based multi-pumping with MPo-RF. SR-MPu approach uses the shift registers instead of multiplexers for the same functionality. In the proposed design, all of the input and output signals are connected to Parallel In Serial Out (PISO) and Single In Parallel Out (SIPO) shift registers as shown in Figures 6 and 7 respectively. In PISO, the two-to-one multiplexers select the signals which come from either the previous register or the outer circuit. All input signals are registered in order to be processed by RF. Keeping in mind that the outer circuit and the inner circuit have fully synchronized clocks, one access cycle from the outer circuit to the memory is realized as follows: Firstly, the outer circuit sets "load" (*ld*) at the rising edge of the outer circuit's clock, and holds it high for one cycle of the inner circuit. In this way, the input values coming from the outer circuit *(addr_Rx, addr_Wx, data_in_Wx, we_Wx)* are stored in the PISO shift registers. Note that the ones corresponding to $x = 0$ directly access to RF when $ld = 1$. In this way, the first read values are also loaded into the related SIPO shift registers. At the beginning of the next cycle of the inner clock, the outer circuit resets ld signal to logic-0 and keeps it at this level till the end of the clock period of the outer circuit. However, at each rising clock edge of the inner circuit, values in shift registers are moved towards the RF. In this way, the RF processes the next values

of read address, write address, write data, write enable one-by-one. Similarly, RF produces one read value at each clock cycle and each read value is shifted through its corresponding SIPO shift regiter. After n cycles of the inner circuit, all PISO shift registers are empty and all SIPO shift registers are full. Hence at the end of the clock cycle of the outer circuit, the values are read from SIPO shift registers except the last ones, which are directly taken from the memory.

The SIPO at the output consists of only flip flops and does not include any additional combinational logic. In Mux-MPu RF design, width of the multiplexers increases with MPuF. The maximum width of the multiplexers in PISO shift register design is two and these are placed in front of the flip flops inside a slice so, this design style is in coherence with the FPGA architecture as an FPGA slice consists of LUT, a multiplexer and finally flip flops in sequence. However in the Mux-MPu, this order is altered: a flip flop is followed by a multiplexer and this occupies twice more slices than SR-MPu approach.

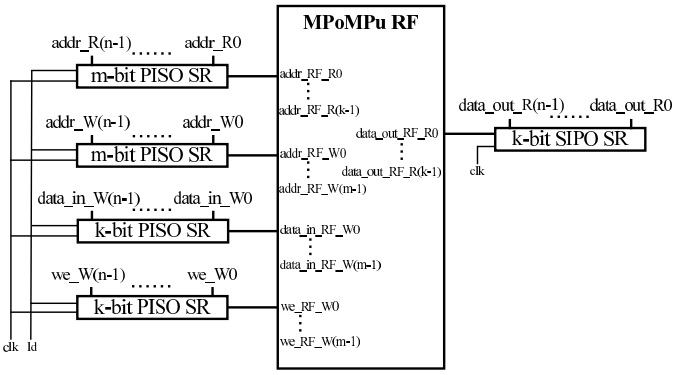


Fig. 5: Shift register based MPoMPu-RF design

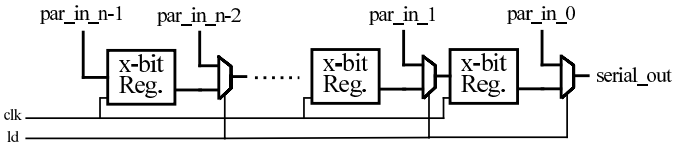


Fig. 6: Parallel Input Single Output (PISO) Shift Register

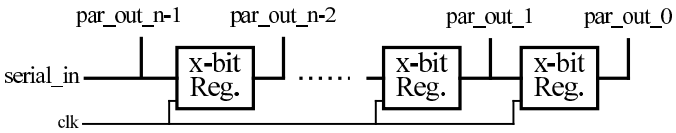


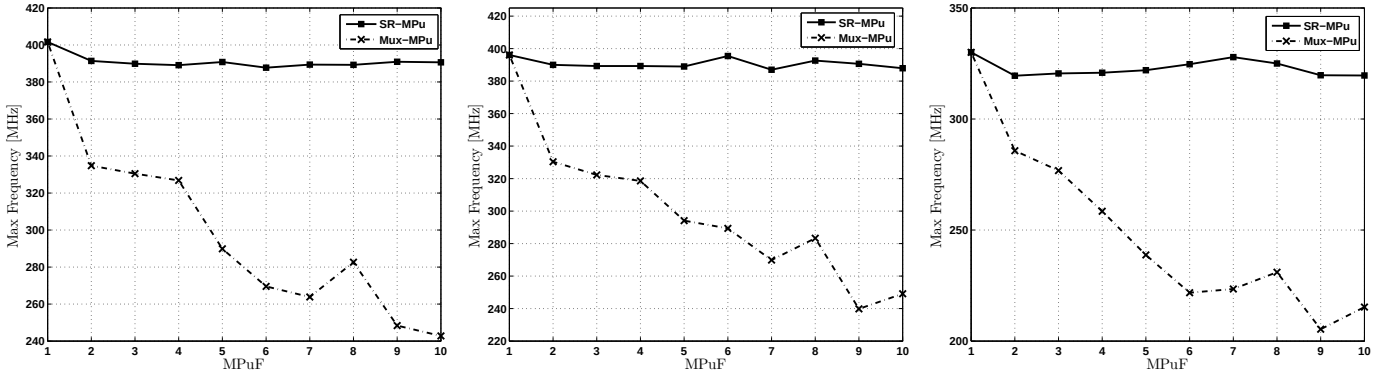
Fig. 7: Single Input Parallel Output (SIPO) Shift Register

VI. Experimental Results

A set of experiments have been conducted to measure the performance, area and energy consumption of the designed MPoMPu-RFs. In the tests, Xilinx Virtex-5 XC5VLX110T FPGA are used. All HDL files that implement the corresponding RFs are automatically synthesized. All of the signals that a pure RF does not need however processor systems possibly need (write enable, RF enable, reset...) are connected for the fair measurements although they affect speed and area results adversely. In [7], these type of signals were left unconnected. For the proper timing measurement, all external ports of the RF are registered and RF is considered as a part of bigger design so as to avoid I/O registers that cause long routing paths. All timing results are obtained from place and route report by constraints forcing until the constraints fail. In the experimental results, frequency is given in terms of MHz. Area results are given in terms of slices and BRAM separately because as far as we know, there is no conversion method from occupied BRAMs to occupied slices. For the power measurements, Xilinx Power Analyzer (XPA) is used by introducing an activity file as input. The activity file utilizes the register file with 100% load. Instead of power consumption, we specified the energy dissipation in terms of power delay product (nJ) because propagation delay and power dissipation generally form a design trade off, that is, improving one of them degrades the other one.

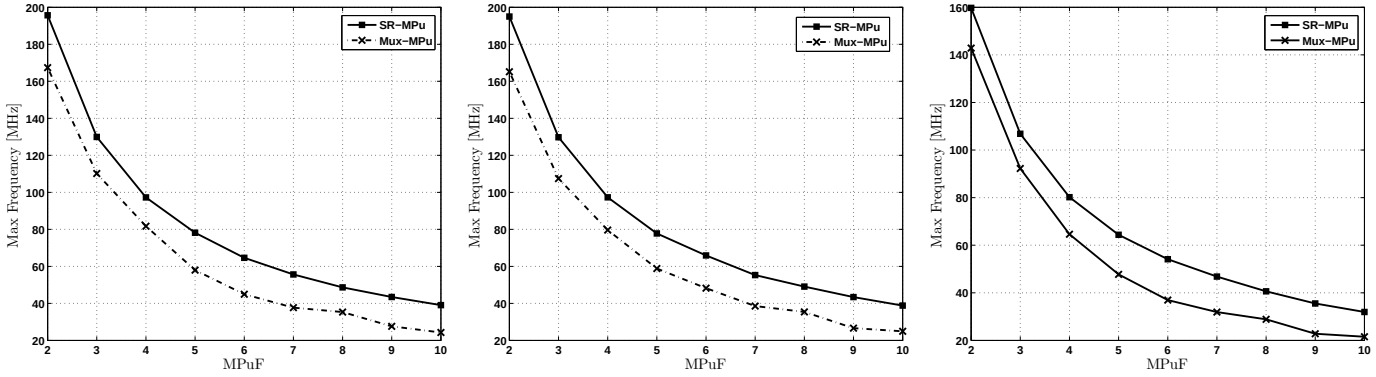
In Figure 8, Figure 9 and Figure 10, the detailed comparison between the Mux-MPu and SR-MPu is presented in terms of maximum internal frequency, maximum external frequency and area respectively. From the figure, it can be inferred that SR based approach outperforms the Mux based type. Moreover, the internal operation frequency of the shift register based MPo-RF is nearly constant at all multi-pumping factors, even when MPuF=10. Also, as the number of RW ports in the base MPo-RF circuit increases, the internal frequency decreases in both designs. This is due to the increased multiplexer sizes in the base MPo-RFs.

These results are not the unique solutions because some advanced constraints and custom placement rules might affect the results. However the variations should be small and these results are sufficient to give an intuition. In our experiments, we tried to obtain the design under delay constraints. When we inspect the results for 3R&2W RFs with MPuF=2, the speed of the SR based approach is about 390 Mhz while in Mux-MPu approach it is about 334 Mhz. As MPuF increases, maximum speed of the Mux-MPu RF design decreases as expected and shift-register based approach remains same at about 390 Mhz. This pattern is also valid for other RF combinations i.e. 4R&2W and 8R&4W. At some implementations, mapping and place&route tools gives the better results for a higher MPuF. This is due to the fact that at these points the designs are more compatible with FPGA architeture and the tools can make better placement. In Virtex 5, the smallest multiplexer that occupies one-pass combinational logic is 4:1 multiplexer. When wider multiplexers required, the multiplexers inside the LUTs are connected to each other by the carry chain and this causes an increase in the longest path. For this reason, in Mux-MPu approach the maximum internal frequency undergoes a break off at every 4:1 multiplexer insertion. For example to obtain 9R&6W RF, we can use 3R&2W with MPuF=3. The internal and external operating frequencies for SR-MPu approach are 390 MHz and 130 MHz respectively. The same



(a) (b) (c)

Fig. 8: Maximum internal operating frequency of the register with respect to MPuF. Base MPo-RF file comes with (a) 3R&2W, (b) 4R&2W, (c) 8R&4W ports



(a) (b) (c)

Fig. 9: Maximum external operating frequency of the register file with respect to MPuF. Base MPo-RF file comes with (a) 3R&2W, (b) 4R&2W, (c) 8R&4W ports

parameters correspond to 330 MHz and 110 MHz for Mux-MPu approach. In terms of area, SR approach always uses fewer resources then Mux-MPu counterpart. To summarize, SR based approach is robust to changes in MPuF and this situation enables us to model RF in a synthesis tool easily. Moreover, SR based approach is faster and less resource occupying compared to Mux-MPu.

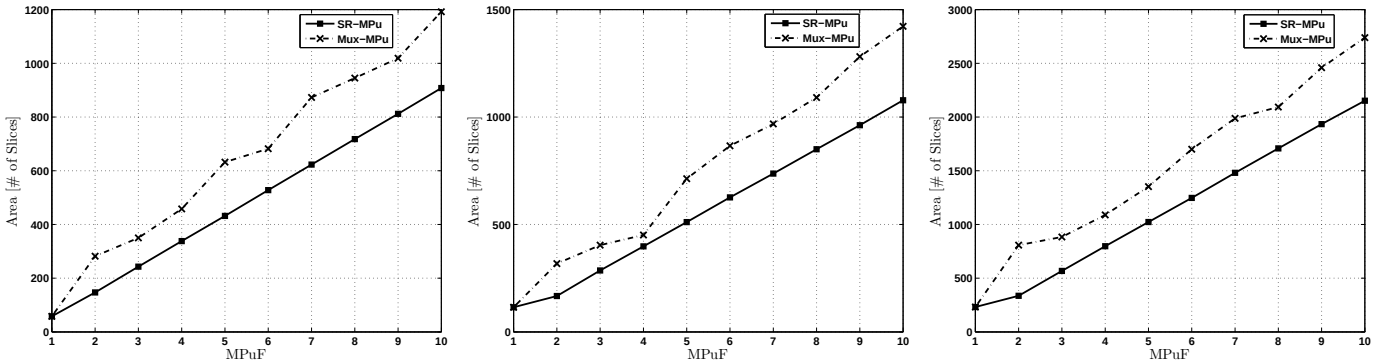
TABLE I: Frequency and area comparison for 4R&2W RF with different wordlengths (MPuF=2)

		SR-MPu		Mux-MPu		Improvement (%)	
RF Width	BRAM Usage	Frequency	Area	Frequency	Area	Frequency	Area
32	4	408.330	104	344.116	196	18.66	46.94
64	8	390.016	167	330.360	318	18.06	47.48
128	16	393.701	297	329.489	617	19.49	51.86
256	32	335.909	552	299.401	1358	12.19	59.35

Table I shows the comparison results for 4R&2W RFs with increasing width from 32-bit to 256-bits. In both both SR-MPu and Mux-MPu approaches, the maximum internal frequency decreases when width increases. However, when we compare their respective internal operation frequencies, SR-MPu approach provides nearly 20% improvement over Mux-MPu approach. The area utilization of SR-MPu approach is nearly 50% better than the Mux-MPu approach. Hence SR-MPu approach provides smaller and faster design than Mux-MPu RF designs.

Using BRAMs with replication and banking is a more effective way of building RF unless it is small and has a few ports when compared with pure slice implementation. If it has many ports, then the usage of implementation with BRAMs makes more sense.

Table II depicts the resource utilization, maximum frequency and power delay product ($P \times D$) for 32-bit width and 64 deep RF implementations with 12R&6W. For a MPoMPu-RF implementation, we always prefer SR-MPu approach however we present the results of Mux-MPu RF implementations for more detailed comparison. As inferred from the table, using SR-MPu approach always outperforms the Mux-MPu approach. The distributed implementation (using slices) is the worst method because its frequency is the lowest and



(a) (b) (c)

Fig. 10: Occupied area of the multi-pumped RF with respect to MPuF. Base MPo-RF file comes with (a) 3R&2W, (b) 4R&2W, (c) 8R&4W ports

TABLE II: Different Methods to design a 32-bit/64 deep RF with 12R&6W Ports

		SR-MPoMPu				Mux-MPoMPu				Improvement (%)		
RF Type	BRAM Usage	Internal Freq	External Freq	Area	PxD	Internal Freq	External Freq	Area	PxD	Freq	Area	PxD
Distributed(12R&6W)	0	187.793	187.793	5364	6.055	187.793	187.793	5364	6.055	0	0	0
MPo(12R&6W)	72	302.115	302.115	312	3.267	302.115	302.115	312	3.267	0	0	0
MPo(6R&3W) & MPuF=2	18	386.250	193.125	136	0.753	320.821	160.411	257	0.823	20.39	47.08	8.44
MPo(4R&2W) & MPuF=3	8	408.831	136.277	155	0.367	344.947	114.982	213	0.444	18.52	27.23	17.28
MPo(2R&1W) & MPuF=6	2	429.923	71.654	165	0.160	377.644	62.941	176	0.214	13.84	6.25	25.17

TABLE III: Different Methods to design a 64-bit/512 deep RF with 18R&12W Ports

		SR-MPoMPu				Mux-MPoMPu				Improvement (%)		
RF Type	BRAM Usage	Internal Freq	External Freq	Area	PxD	Internal Freq	External Freq	Area	PxD	Freq	Area	PxD
MPo(9R&6W) & MPuF=2	54	289.101	144.550	776	4.448	246.488	123.244	1177	4.828	17.29	34.07	7.86
MPo(6R&4W) & MPuF=3	24	349.650	116.550	486	1.862	276.472	92.157	707	2.029	26.47	31.26	8.24
MPo(3R&2W) & MPuF=6	6	387.747	64.625	528	0.619	269.542	44.924	682	0.735	43.85	22.58	15.74

TABLE IV: Different Methods to design a 64-bit/512 deep RF with 32R&24W Ports

		SR-MPoMPu				Mux-MPoMPu				Improvement (%)		
RF Type	BRAM Usage	Internal Freq	External Freq	Area	PxD	Internal Freq	External Freq	Area	PxD	Freq	Area	PxD
MPo(8R&6W) & MPuF=4	48	301.205	75.301	1224	2.643	224.467	56.117	1436	3.58	34.19	14.76	26.13
MPo(4R&3W) & MPuF=8	12	376.790	47.099	1009	1.101	268.097	33.512	1243	1.24	40.54	18.83	11.06

it occupies the largest area. Another disadvantage of using slices is long synthesis and implementation time. We have implemented the same RF by only using MPo method as mentioned in Section III. The highest external frequency is given in this design. However the RF is divided into six parts. This situation dictates that compiler should provide effort on write conflicts avoidance i.e. no two or more write should direct to the same bank. When RF with 6R&3W and MPuF two is used, internal speed of the RF is higher than previous design (about 384 Mhz) however due to the multi-pumping this design has to be driven at 192 Mhz by the outer circuit. If the processor using this RF works at a frequency below 192 Mhz, this design is more advantageous because it occupies only nine BRAMs while the previous design uses 36 BRAMs. In addition to that, it is worth to mention that since RF is divided into three banks, the effort of the compiler is less than the previous case. Increasing MPu decreases the external speed. However the number of banks and the number of used BRAMs are decreased. So depending on the operation speed of the outer circuit and compiler effort, one can select the most suitable design from the listed MPo-RFs. In terms of energy, using BRAM and exploiting multi-pumping as much as possible decrease the energy consumption.

Table III shows different methods to implement a 64-bit/512 deep RF with 18R&12W ports. In these configuration, RF could not be implemented as pure MPo or distributed. Because implementing this RF requires 216 BRAMs (18×12) and our FPGA has only 148 BRAMs. This situation also occurs in 32R&24W MPoMPu-RF implementations when MPuF is two as presented in Table IV. So, facilitating the multi-ported RF implementations that cannot be designed by pure MPoSPu-RF is one of the extra advantages of multi-pumping.

As inferred from the table results, using SR-based MPoMPu-RF provides improvement up to 43% in internal frequency, 47% in area and 26% in energy consumption.

VII. Conclusions

In this paper, we presented the design and implementation of a MPoMPu-RF utilizing BRAMs by exploiting the MPu methodology. Compared to the slice register implementation, this method provides a considerable resource, speed and energy gain. We point out that the multi-pumping techniques used in the literature are not compliant with the FPGA architecture. Hence, we propose SR-MPu approach which enables design of MPoMPu-RFs with aggressive multi-pumping. This is due to the fact that SR-MPu approach makes the internal frequency of the MPo-RF nearly independent from MPuF. Our MPoMPu-RF design technique can exploit the MPu as much as possible and results a reasonably profit in terms of speed and resource utilization. In addition to the RF design, the SR-MPoMPu method can be used for any resource which can work at faster internal frequencies compared to its external running frequency.

Acknowledgment

This work is fully supported by The Scientific and Technological Research Council of Turkey, TUBITAK under BIDEB 2210 Program and State Planning Organization of Turkey, (DPT) under the TAM Project, Grant No. 2007K120610.

References

[1] Xilinx, *Zynq-7000 All Programmable SoC Technical Reference Manual*. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2012.

[3] Xilinx, *Xilinx UG384 Spartan-6 FPGA Configurable Logic Block User Guide*. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

[4] C. E. LaForest, M. G. Liu, E. R. Rapati, and J. G. Steffan, "Multi-ported memories for fpgas via xor," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 209–218.

[5] M. A. R. Saghir and R. Naous, "A configurable multi-ported register file architecture for soft processor cores," in *Proceedings of the 3rd international conference on Reconfigurable computing: architectures, tools and applications*, ser. ARC'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 14–25. [Online]. Available: http://dl.acm.org/citation.cfm?id=1764631.1764634

[6] F. Anjam, S. Wong, and F. Nadeem, "A multiported register file with register renaming for configurable softcore vliw processors," in *Field-Programmable Technology (FPT), 2010 International Conference on*, Dec., pp. 403–408.

[7] C. E. LaForest and J. G. Steffan, "Efficient multi-ported memories for fpgas," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '10. New York, NY, USA: ACM, 2010, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/1723112.1723122

[8] R. D. Jolly, "A 9-ns, 1.4-gigabyte/s, 17-ported cmos register file," *Solid-State Circuits, IEEE Journal of*, vol. 26, no. 10, pp. 1407–1412, 1991.

[9] D. Garde, "Multi-port register file with flow-through of data," U.S. Patent 4 811 296, 1989.

[10] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Trans. Comput.*, vol. 21, no. 9, pp. 948–960, Sep. 1972. [Online]. Available: http://dx.doi.org/10.1109/TC.1972.5009071

[11] Xilinx, *XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices*. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/xst.pdf

[12] N. Sawyer and M. Defossez, *Quad-Port Memories in Virtex Devices*. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp228.pdf

[13] A. Canis, S. D. Brown, and J. H. Anderson, "Multi-pumping for resource reduction in fpga high-level synthesis," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013.

[14] Xilinx, *IP Processor Block RAM (BRAM) Block*. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/bram_block.pdf

[15] Altera, *Internal Memory (RAM and ROM) User Guide*. [Online]. Available: http://www.altera.com/literature/ug/ug_ram_rom.pdf

[16] N. Instruments, "Synchronous communications and timing configurations in digital devices." [Online]. Available: http://www.ni.com/white-paper/6552/en

[17] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 203–215, Feb.