An Efficient Heterogeneous Register File Implementation for FPGAs

Hasan Erdem Yantır and 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, yurdakul}@boun.edu.tr

***Abstract*—For the future of computing, wide usage of heterogeneous architecture is indispensable since advances in technology scaling cannot satisfy the expected increase in performance of computational platforms anymore. FPGA is a promising platform for heterogeneous computing due to its configurable structure. Each part of an FPGA can be configured to perform a different task that it is best suited for. Such a heterogeneous system needs a common register file (RF) that can serve different parts of the FPGA with at different characteristics in terms of running frequency, data consumption/production rate, required number of ports, data widths, address spaces and endianness. In this study, we propose a heterogeneous register file (HRF) architecture for FPGA-based heterogeneous systems. The designed register file uses a heterogeneous multi-port base-RF to provide such heterogeneity. For the power and area reduction, the design takes advantage of frequency differences between processing elements and HRF by an efficient multi-pumping system. According to the literature, this is the first study on FPGA-based heterogeneous RFs. For experimentation, HRF is tested in four different heterogeneous architectures with increasing complexity. For all HRF configurations, speed, area and energy are measured. Test results of the HRF on Xilinx Virtex-5 show that our heterogeneous register file outperforms other RF architectures implemented by conventional methods.**

I. Introduction

In the past, performance of computational platforms could increase by advances in technology scaling without any architectural change. However the technology advance is not sufficient anymore and wide usage of heterogeneous architectures is envisaged for the future computing systems. FPGAs are great candidates of heterogeneous platforms for the implementation of such complex applications because of their fully reconfigurable architectures. Hence, each part of an FPGA can be configured to perform a different task that it is best suited for. Nowadays, most embedded applications require partitioning the functionality between computational units with different characteristics. Each computational unit works on a different set of data and sometimes they require processing the same data set.

One of the obstacles in the way of obtaining high performance in heterogeneous computing is the memory-wall [1]. If the processing elements cannot get the data from register file at the processing rate, this causes a bottleneck that adversely affects the overall performance. In order to meet the requirement of proper data usage between the computational units, such a heterogeneous system needs a heterogeneous register file that can meet the requirements of different computing units on the FPGA. These requirements can be specified in terms of execution frequency, data consumption/production rate, data width, address space, number of ports, bandwidth and endianness (order of bytes/bits in memory).

In this study, we propose a heterogeneous register file (HRF) for heterogeneous systems on a single FPGA. In fact, the term *heterogeneous memory* could be used in place of *heterogeneous register file*. However we prefer the heterogeneous register file term since implementation is inside the FPGA. The designed register file exploits multi-porting and multi-pumping to provide such heterogeneity. Multi-pumping also provides significant power and area reduction. Facilitating the RF implementations that cannot be designed by conventional methods is one of the extra advantages of multi-pumping. In the designed system, each processing element can use as many ports as it requires and the register file can be adapted to meet the data requirements of the processing elements. In a heterogeneous system, processing elements can run at different clock domains depending on their architectures. When the frequencies of the processing elements are different from each other, multi-pumping can be exploited to take advantage of these differences. Sometimes, data widths of the processing elements may be different. In that case, port widths and word lengths of block memory devices can be set according to the processing elements. When the port width and number of required ports can be satisfied for a processing element, the bandwidth requirements of that processing element can be met. Endianness is another feature that refers to how bytes of a data word are ordered within memory. In HRF, the endianness of a port can be defined as either big-endian or little-endian for each processing elements.

The rest of the paper is organized as follows: In the following section, we mention about the related studies in the literature. Section III explains the HRF architecture in an orderly fashion. Section III-A explains detailed architecture of base-HRF designed by using dual port BRAMs. The method described in this section is the enhanced version of the study in [2]. Section III-B explains how multi-pumping can be used with base-HRFs so as to obtain smaller register files with increased number of ports. The ultimate architecture of our HRF also becomes clear in this section. Results are discussed in Section IV. The final section concludes the work.

II. Related Work

A heterogeneous system with shared memory needs a multi-port register file (MPo-RF). The multi-port register file implementation in FPGA is one of the most resource consuming parts of such a design. In a heterogeneous system designed as a custom chip, RFs can be implemented as hard-IPs. However, on FPGAs, this is an infeasible solution. Therefore, general purpose HRFs for heterogeneous systems on FPGAs have to be implemented with on-chip resources of the FPGA.

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. However, this method is not scalable at all because the combinational and wiring delays of slices dominate as the number of ports increases [3]. The second method relies on using block RAMs that are available in traditional FPGAs [4] [5]. However, they have only two ports and are not sufficient for a heterogeneous architecture that usually needs many ports and shows different characteristics. The third method is combining on-chip block RAMs (BRAM) in conventional methods to form an MPo-RF. In MPo-RF design with BRAMs, two common methods are replication and banking. Multi-read and one-write RFs can be designed by using only replication. To increase the number of write ports, both replication and banking can be exploited as suggested by Sagmir et al. [2]. 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. 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 on the same bank. In [6], this problem is solved by using 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 directed to the output of the MPo-RF. This methodology comes with extra resource usage, decrease in operating frequency and increase in power consumption.

There exist some nonspecific RF architectures implemented in a heterogeneous system. As an example, the work in [7] proposes a register file design that supports multiple accesses from two processors. One of these processors is a high performance core, and the other one is a low power core. In this design, each processing element can use 2R&1W ports and there are some additional mechanisms to provide data consistency between storage units. It uses multiplexer-based port-sharing which has been proven to be inefficient for FPGAs [8]. In the study [9], each processor uses its local memory and a unified external memory is shared between the processors. This system uses a NUMA layout and there is no shared memory that can be used concurrently between the processors. In [10], an ESL synthesis framework for heterogeneous systems has been proposed. In this system, processors are connected to a shared memory by using Xilinx LMBs (local memory buses) so only one processor can access the shared memory at a time. In [11], each processing element uses a dedicated memory with a shared external memory. A heterogeneous register file should service different processing elements that show different characteristics in terms of operating frequency, number of ports, data width, address space and endianness. To the best of our knowledge, none of these studies has touched structurally heterogeneous register files for heterogeneous computing platforms on FPGAs like has been addressed in this study.

III. HRF Architecture

Our heterogeneous register file design is a combination of base heterogeneous MPo-RF (base-HRF) and multi-pump circuits applied to ports of the base-HRF. The resulting HRF can serve different processing elements having different characteristics. Processing elements might be anything that can be implementable in FPGA like hardcore/softcore processors, DSP processors, custom logic circuits, etc. Figure 1 shows a representation of FPGA-based heterogeneous computing system consisting of four processing elements (PEs) and an abstract HRF architecture used commonly by these PEs. In the figure, clocks of each processing elements are generated from DCMs or PLLs existing in contemporary FPGAs. Number of processing elements, address spaces of the processing elements, number of read/write ports, data widths, multi-pumping factors, endianness and clock frequencies (DCM values) in the figure can be changed depending on the application running on heterogeneous system. For example, PE 0 needs three read ports and its access speed is three times slower than that of the base-HRF. It accesses a memory of H_0 depth and word length of each memory is D_0. The following sections will elaborate on the architectural details of our HRF design.

A. Base Heterogeneous Register File

In traditional FPGA architectures, there exist dedicated BRAMs for storage. As an example, FPGA used for this study (Xilinx Virtex-5 XC5VLX110T) includes total 148 BRAMs. 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 ports depending on the BRAM mode. So exclusively BRAMs are not sufficient to implement a heterogeneous MPo-RF.

As pointed in Section II, an MPo-RF can be designed by replication and banking as Sagmir et al. suggested [2]. However, in this method, the RF is divided into equally sized

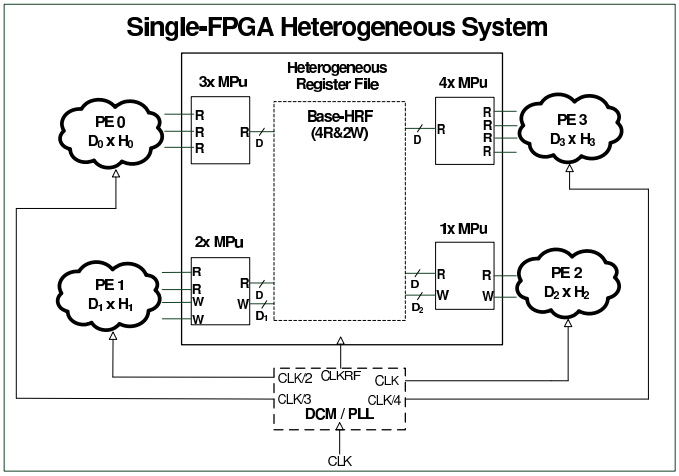


Fig. 1: Single-FPGA Heterogeneous System with HRF

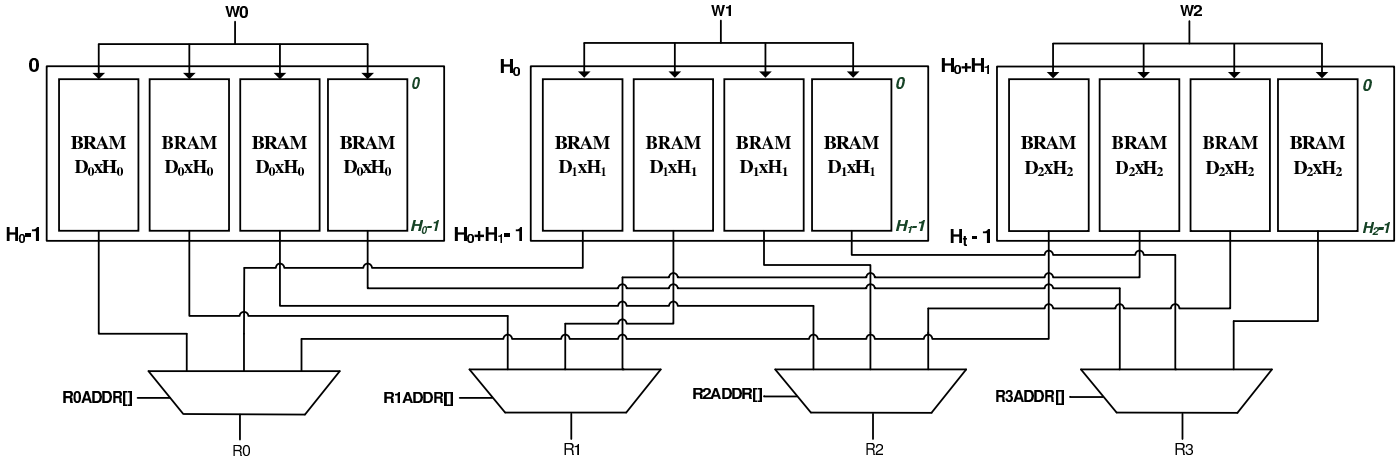


Fig. 2: A base-HRF with four read and three write ports (4R&3W)

address spaces and each BRAM holds full address space although only a portion of this address space is reachable. In addition, data width of the register file is fixed for all banks and cannot be changed. This architecture does not provide a heterogeneous structure since port widths and register file sizes are fixed. When used for heterogeneous systems, this method results in the waste of storage resources in FPGA. In a heterogeneous system, each processing element requires its own address space. Data widths of the processing elements may vary with characteristics of the processing elements. For example, Microblaze soft-core processor consumes 32-bit data and Picoblaze processor consumes 8-bit data. A heterogeneous system may consist of these processors and an encryption system like 128-bit AES core from OpenCores [12] to encrypt the results of their operations instantaneously. Additionally, the address space of a processing element should not be reachable from other processing elements to protect its data from accidental write operations and data corruptions. At the same time, processing elements may use computational results of each other so they can access the address spaces of other processing elements to get data. For this reason, address spaces of processing elements should be separated. Otherwise (if a processing element can write entire address space), this system requires a complex compiler that have to orchestrate all processing elements against data corruptions. Alternatively, this issue can also be handled by hardware mechanisms. In both cases, complexity of the heterogeneous system increases.

In our HRF design, each processing element has an address space and this address space corresponds to a portion/bank of the HRF with varying heights and data widths. Each bank consists of replicated BRAMs configured as simple dual port i.e. they have only one read and one write port. Each register bank is associated with a write port. Inside a bank, all BRAMs hold the same data and represent the same address space range to increase the number of read ports. In other words, each replicated BRAM is associated with a read port. This address space is named as local address space. Union of all banks forms the global address space so the global address space is the sum of all local address spaces of processing elements. A processing element can only write to its own address space and can read data from address spaces of all processing elements. Equation 1 shows the formula of the total address space. At the addressing perspective, width of the read address is wider than write address because read address space is higher. Equations 2 and 3 show the write address width and read address width respectively. In here, write addresses are the trimmed versions of the read addresses.

$$Total\ Height\, H_t = \sum_{i=0}^{Bank\#} H_i \tag{1}$$

$$Write\ Address\ Width\ of\ PE_i = log_2(H_i) \tag{2}$$

$$Read\ Address\ Width\ for\ all\ PEs = log_2(H_t) \tag{3}$$

Figure 2 illustrates implementation of 4R&3W base-HRF using BRAMs. In the example, there are three write ports so there should be three banks. Data widths and heights of the banks are determined according to the processing elements and they can be different. Each bank holds a local address space as shown in the figure. Global address correspondence of each bank is shown at the left corners of the banks. For example, the first bank holds the data that corresponds the addresses between 0 and $H_0 - 1$. During a read operation, the data that is pointed by the given read address is directed to output port from BRAM that contains corresponding address space. This data direction is done by multiplexers connected to outputs of BRAMs. The select inputs of the multiplexers are the most significant bits of read address that is able to distinguish the smallest address space. The size of select inputs can be changed depending on the partition (i.e. number of write ports). Here it is worth noting that the height of a bank has to be a power of two, otherwise multiplexers would be very large. During a write operation, the value is written to the corresponding register bank. However it is not possible to realize more than one write operation to the same bank at the same time. After applying multi-pumping (see Section III-B), if the number of write ports is not sufficient for a processing element, local address space of a processing element is fragmented since there are more than one write ports dedicated to this processing element. This problem can be handled by register renaming at compile time [13].

1) Output Port Width: In HRF, read output data width of the base-HRF should be set as the largest data width of the processing elements to guarantee one-cycle access to HRF for all processing elements. The processing elements with smaller data widths should use only a portion of the data. In such a case, processing elements can put data in a buffer and process it in parts.

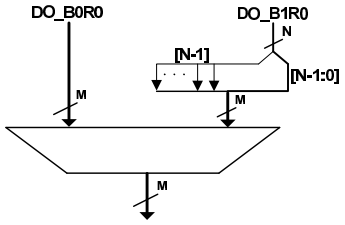


Fig. 3: Sign pass between different width data

2) Signed Number Conversions: In some cases, a higher-bit processing element can require the results of lower-bit processing elements. To handle incompatibility between the number representations, a data read from a bank is fit to the target whilst passing through the multiplexer. A conversion mechanism is required to preserve the data sign. In such a case, remaining bits of the data are complemented with the most significant bits of the read value to preserve the sign. Figure 3 shows this basic conversion.

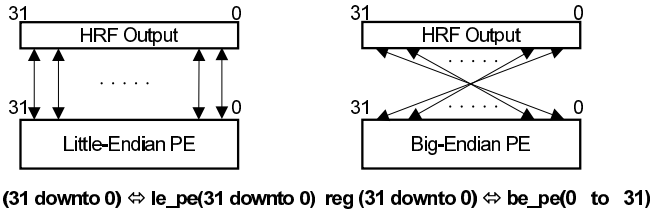


Fig. 4: Connection methods with different endianness

3) Endianness: Endianness differences between the processing elements can be handled automatically in the design phase of our register file. If processing elements have different endianness, then HRF holds all data values in little-endian format. Designer can specify the endianness of a processing element and routing of the HRF can be designed by regarding endianness. In such a case, order of the buses coming to/going from base-HRF can be reversed in order to provide compatibility. Figure 4 shows the little-endian (4a) and big-endian (4b) connection schemes for little-endian and big-endian processing elements respectively in an example 32-bit HRF design. For the byte endianness, also byte orders for each processing element can be configured.

B. Multi-pumping

Multi-pumping (MPu) is a method used in digital circuits. The idea behind multi-pumping is to operate the RF much faster than its input and output ports so that multiple accesses can be done to realize multiple reads and multiple writes in a time-multiplexed manner. This method illustrates one resource as if there exist multiple replications of this resource. In this way, the area of the RF is reduced considerably. The multi-pumping factor (MPuF) is defined as the rate between the operating frequency of the register file and the frequency to read from or write to a port. The formula of MPuF is given in Equation 4.

$$MPuF = \frac{Processing\ Element\ Period}{HRF\ Period} \tag{4}$$

In the literature, multi-pumping is achieved by connecting multiplexers to the input ports and demultiplexers to output ports of the register file. However, in [8], it is shown that this type of implementation is not scalable, because the operation frequency decreases when multi-pumping factor (MPuF) increases. This shift register based design method provides smaller and faster MPo-RFs compared to the traditional multi-pumped implementations.

In general, processing elements in a heterogeneous system runs at different frequencies and each of them belongs to different clock domains. Regarding the different clock domains, each processing element can exploit multi-pumping for area reduction. In HRF, different multi-pumping factors can be applied to different ports of the register file depending on characteristics of processing elements. When applied to the write ports, multi-pumping also diminishes address space fragmentation for a processing element as stated in Section III-A. With respect to variance in MPuFs, each connected processing elements run at most at a fraction of the HRF speed. For example in Figure 1, if MPuF=2 were applied to read port of the base-HRF, the corresponding processing element that is getting service from first read port would run at most half of the HRF clock speed (*CLKRF*). In the designed HRF, each processing element can be dedicated more than one port (both read and write). However it is required that the MPuFs of these ports should be the same and equal to the clock speed ratio of HRF and processing element. If a processing element needs fewer ports than the multi-pumping offers, the processing element does not have to make connections to excess ports and they can be left unconnected.

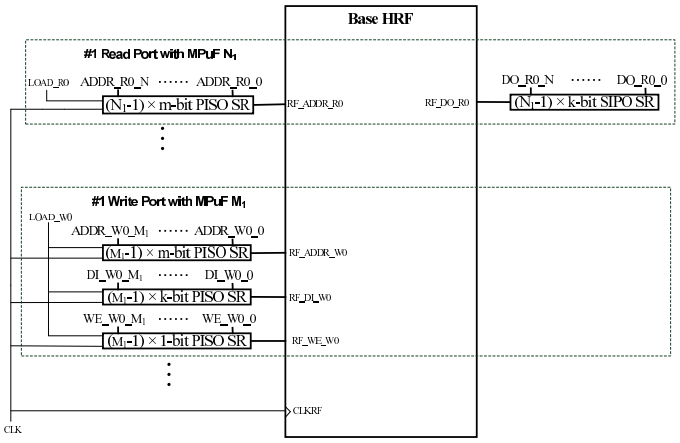
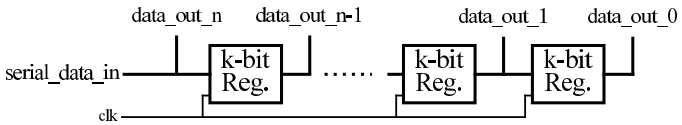
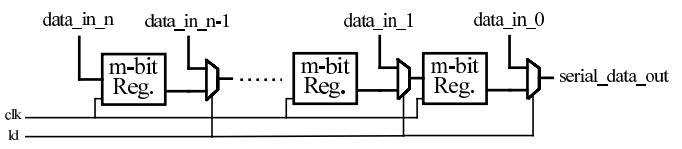


Fig. 5: A Multi-pumped HRF

Figure 5 shows the design of shift register based multi-pumping with base-HRF. In the 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 6a and 6b respectively. In PISO, the two-to-one multiplexers select the signals which come from either the previous register or the processing element. Each processing element has its own load signal (*LOAD_Wy* and *LOAD_Ry*) if corresponding MPuF is greater than 1. Otherwise it is connected directly to HRF without using PISO and SIPO, that is, it can run at the frequency of the HRF. The processing elements control the flow of the data to/from HRF by this signal. All input signals are registered in order to be processed by RF. Keeping in mind that the processing element and the RF have fully synchronized the clocks, one access cycle from the processing element to the RF is realized as follows: Firstly, the processing element sets "LOAD_Rx" or "LOAD_Wx" at the rising edge of its clock, and holds it high for one cycle of the register file. In this way, the input values coming from the processing element *(ADDR_Ry_Rx, ADDR_Wy_Wx, DI_Wy_Wx, WE_Wy_Wx)* are stored in the PISO shift registers. In here, *x* corresponds the to port number that is connected to *xth* input of the shift registers and *y* corresponds to port number of HRF that the related SIPOs or PISOs is connected to. Note that the ones corresponding to x=0 directly access to base-HRF when load signal is logic-1 because they are first input of SIPOs. 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, load signal is set to logic-0 and keeps it at this level till the end of the clock period of the processing element. However, at each rising clock edge of the base-HRF, values in shift registers are moved towards the base-HRF. In this way, the base-HRF processes the next values of read address, write address, write data, write enable one-by-one. Similarly, base-HRF produces one read value at each clock cycle and each read value is shifted through its corresponding SIPO shift register. After n cycles of the base-HRF, all PISO shift registers are empty and all SIPO shift registers are full. This process takes multiple cycles in RF but this period seems like one cycle for the processing element. Hence at the end of the clock cycle of the processing element, the values are read from SIPO shift registers except the last ones, which are directly taken from the HRF.



(a) Parallel Input Single Output (PISO) shift register.



(b) Single Input Parallel Output (SIPO) shift register.

Fig. 6: SIPO and PISO shift registers.

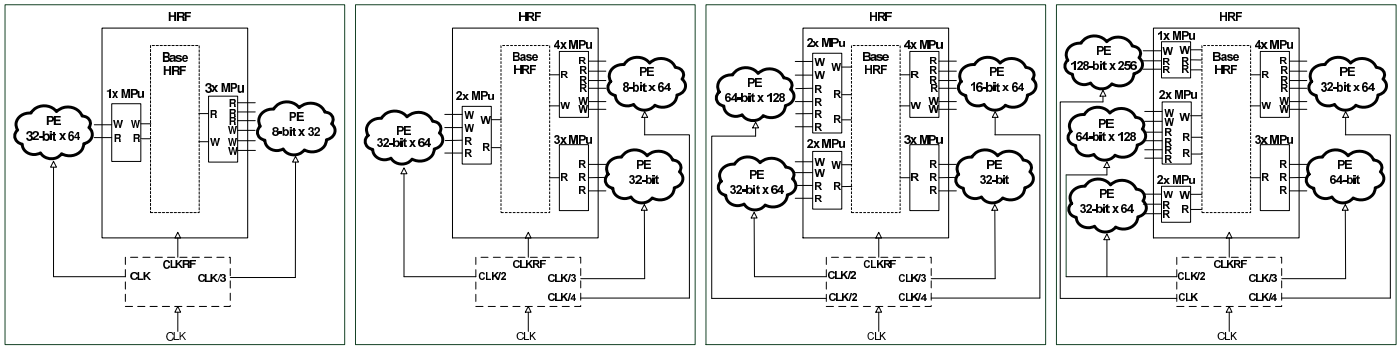
IV. Experimental Results

A set of experiments have been conducted to measure the performance and area of the designed HRFs. In the tests, Xilinx Virtex-5 XC5VLX110T FPGA is used. All HDL files that implement the corresponding HRFs are automatically synthesized with proper features (data widths, address spaces, bank heights, endianness, port numbers, etc.). All timing results are obtained by constraints forcing until the constraints fail to find the maximum speed. For the energy measurements, Xilinx Power Analyzer (XPA) is used by introducing an activity file as input. The activity file utilizes the register file with 100% load. To generate the clocks of the processing elements DCM is used. DCM can generate different clock frequencies with zero clock skew. All clock domain crossing operations is handled by XST. 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.

For the experiments, we have prepared four different FPGA-based heterogeneous systems composing of different processing elements that show different characteristics as illustrated in Figure 7. As the system number increases, complexity of the system increases too. Running frequencies of the processing elements vary and clocks are generated by DCM. Depending on the ratio between HRF speed and processing element speed, MPuFs are set.

Table I shows the resource occupation of each HRF in terms of LUT-FF pairs and 18kB BRAM blocks. In terms of LUT-FF pairs, distributed approach is worst hence it is implemented by using slices. For System 0 and 1, HRF is slightly worse than MPo because HRF uses PISO and SIPO shift registers at the input and output ports. Nevertheless they are comparable. In BRAM usage, HRF consumes the lowest area because it exploits multi-pumping and world length of BRAM blocks can be variable. However in MPo approach, this structure is not suitable for different length blocks. It is also interesting that System 0 and System 1 had to use 4 and 6 18k BRAMs respectively according to our base-HRF design. During synthesis, XST uses slice-based memory resources like RAM32M for small blocks and number of occupied BRAMs decreases. This is also another reason to why LUT-FF pair usage is higher in HRF for System 0 and System 1. When the system complexity increases, our HRF outperforms MPo in both LUT-FF pairs and BRAMs (see System 2). In addition, our competitors become to be eliminated when the complexity increases. For example, distributed RF could not be implemented for System 2 and 3 because total chip area is not sufficient to route all RF signals in an allowable time so its results are not available (NA) as shown in table. For System 3, MPo RF is non implementable too, because there is no enough BRAMs. So, facilitating the multi-ported RF implementations that cannot be designed by pure MPo-RF is one of the extra advantages of multi-pumping.

Figure 8a and 8b show the maximum operating frequencies and energy consumptions of the designed RFs respectively. From the figure, it can be inferred that HRF consumes much less energy than others because it uses least number of BRAMs. In terms of energy, it is obvious that using BRAM and exploiting multi-pumping as much as possible decrease the energy consumption. For operating frequency, MPo is better

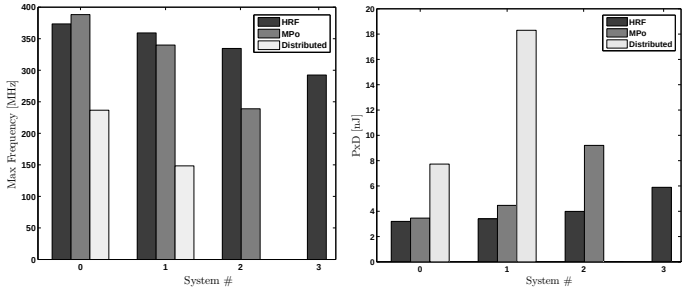


(a) System 0 (b) System 1 (c) System 2 (d) System 3

Fig. 7: Single-FPGA Heterogeneous Systems with increasing complexity.

TABLE I: Resource evaluation results

System	HRF		MPo [2]		Distributed	
	LUT-FF Pairs	BRAMs	LUT-FF Pairs	BRAMs	LUT-FF Pairs	BRAMs
System 0	146	2	132	16	10910	0
System 1	401	3	292	36	21397	0
System 2	1058	15	1670	156	NA	NA
System 3	2247	56	NA	NA	NA	NA



(a) (b)

Fig. 8: Maximum operating frequency and energy consumption results

in System 0 because shift registers in HRF dominate however for System 1 and System 2, HRF outperforms. The distributed implementation (using slices) is the worst method because its frequency is the lowest and it occupies the largest area.

V. Conclusions

In this paper, we proposed an architecture for the design and implementation of a heterogeneous multi-port register file utilizing BRAMs by exploiting our multi-pumping methodology. This heterogeneous register file can be used safely in single-FPGA heterogeneous systems and outperforms other RF architectures. Our design occupies less area and consumes significantly lower energy than its alternatives.

Acknowledgment

This work is fully supported by The Scientific and Technological Research Council of Turkey, TUBITAK under BIDEB 2211 Program.

References

[1] T. J. Ham, B. K. Chelepalli, N. Xue, and B. C. Lee, "Disintegrated control for energy-efficient and heterogeneous memory systems," in *High Performance Computer Architecture (HPCA2013), IEEE 19th International Symposium on*, 2013, pp. 424–435.

[2] 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.

[3] L. Lie-wen, G. Wei-hua, and H. Xiao-long, "Research on low power design methodology of register files based on fpga," in *Electric Information and Control Engineering (ICEICE), 2011 International Conference on*, 2011, pp. 673–676.

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

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

[6] 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.

[7] Y. Wu, P. Zhu, H. SUN, and E. Guidetti, "Register file organization to share process context for heterogeneous multiple processors or joint processor," US Patent US20 130 173 865 A1, Jul 4, 2013.

[8] H. E. Yantir, S. Bayar, and A. Yurdakul, "Efficient implementations of multi-pumped multi-port register files in fpgas," in *Digital System Design (DSD), Euromicro Conference on*, 2013, pp. 185–192.

[9] L. T. Rusten and G. I. Sortland, "Implementing a heterogeneous multi-core prototype in an fpga," Master's thesis, Norwegian University of Science and Technology, Department of Computer and Information Science, 2012.

[10] Y. Corre, J.-P. Diguet, L. Lagadec, D. Heller, and D. Blouin, "Fast template-based heterogeneous mpsoc synthesis on fpga," in *Proceedings of the 9th International Conference on Reconfigurable Computing: Architectures, Tools, and Applications*, ser. ARC'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 154–166. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36812-7_15

[11] W.-T. Zhang, L.-F. Geng, D. li Zhang, G.-M. Du, M.-L. Gao, W. Zhang, N. Hou, and Y.-H. Tang, "Design of heterogeneous mpsoc on fpga," in *ASIC, ASICON '07. 7th International Conference on*, 2007, pp. 102–105.

[12] T. Ruschival, "Aes core project." [Online]. Available: http://opencores.org/project,tiny_aes

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