From a FPGA Prototyping Platform to a Computing Platform: The MANGO Experience

J. Flich, R. Tornero, D. Rodriguez, D. Russo, J.M. Martínez, C. Hernández
Universitat Politècnica de València
Valencia, Spain
jflich@disca.upv.es

***Abstract*—In this paper we describe the evolution of the FPGA-based prototype deployed in the MANGO project, from a hardware prototyping platform of HPC architectures to a computing platform targeting HPC and AI applications. Our main goal is to reinvest on the MANGO cluster by providing a duality in its use for both large-scale hardware prototyping and high-performance computation. From our experience we can reach several interesting conclusions about the complexities and hurdles that lay below FPGA technologies, and therefore, shedding some light onto the real complexities that difficult the adoption of FPGAs on either large-scale pure HPC systems or on hybrid systems (HPC + BigData/AI).**

***Index Terms*—FPGA, Prototype, MANGO project**

I. Introduction

High-Performance Computing (HPC) is always pushing technology for the quest of higher performance. Power and energy dissipation have pushed HPC systems to diversify the compute devices used and currently there is a plethora of types of systems including manycore accelerators, GPUs, and in some cases FPGAs. Although there is no consensus whether systems made of FPGAs can be called HPC systems or not, what is certain is that with the new emerging applications, mostly in the arena of Artificial Intelligence, heterogeneous systems are being adopted and used massively in large infrastructures.

In the recent past, the EU has funded several projects with the focus on HPC architectures, trying to push European presence in the HPC market. In this paper, we describe one small effort made on several of such projects where FPGAs have been targeted as a key component. The FET-HPC MANGO project [1] ended up in May 2019. This project targeted the deployment of a large prototyping infrastructure made of FPGAs. As a result, the prototype was deployed with the goal to develop and explore future HPC architectures by means of its emulation capabilities. Manycores and GPU-like accelerators, together with application-specific custom accelerators, were deployed and analyzed (via emulation) on the prototype.

Recently, new european projects have emerged where somehow the MANGO prototype is targeted as a computing system. This is the case of FET-HPC RECIPE project [2] and the ICT DeepHealth project [3]. In both projects the MANGO prototype, half of it as it is managed at UPV premises, is being adapted from strict emulation capabilities (for chip prototyping) to strict performance capabilities (for energy-efficient computing exploration). In this paper, we describe such migration effort and highlight the hurdles beneath such effort which demonstrates, somehow, the difficulties of adopting FPGA technologies in the HPC community.

The paper is organized as follows. Section II describes the current MANGO prototype, showing its physical constraints and its performance capabilities. Section III describes the adaptation efforts made to use the MANGO prototype as a compute platform. Section IV shows preliminary results of this adaptation effort, and Section V concludes the paper.

II. The MANGO Prototype

The complete HW infrastructure[1] for the MANGO project consists of a group of components that can be split in two clearly differentiated subsystems. The first subsystem is the General-purpose Node infrastructure (GN), made of four blade servers, each with two XEON E5-2680 v4 processors.

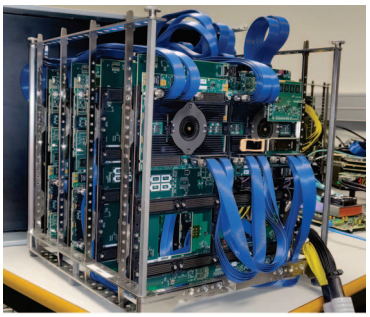
GNs are used to access and manage the FPGA subsystem. GNs run the Resource Manager (RM) based on the Barbeque [4] runtime. With this, applications are launched and resources are dynamically assigned and used. The GN node is also in charge of running the host-side part of applications needing FPGA resources. FPGA-specific management software is deployed on GNs in order to launch bitstreams to the FPGAs.

The second subsystem is the HN infrastructure, made on eight Heterogeneous Node (HN) clusters with 96 high-end FPGAs in total. Figure 1 shows one of such clusters. The HN subsystem is the core infrastructure of the MANGO project. The role of the HN subsystem was to provide the physical infrastructure hosting the different architectures and configurations of the set of accelerators researched in the project.

The MANGO cluster is a set of motherboards, FPGA modules, DDR modules and I/O modules. FPGAs and motherboards are interconnected through specific data cables (blue cables in the picture). All these components can be physically reconfigured and many different topologies can be exercised. All the components are manufactured and sold by ProDesign GmbH company [5] which was a MANGO partner. The flexibility of the components (its connections) allowed exploring new FPGA-based architectures and served as a good hardware infrastructure system to explore manycore architectures in MANGO. New FPGA components can be added to our current prototype as new FPGA devices emerge in the market (downward compatibility is also guaranteed).

[1]The prototype was split after the MANGO project end between different partners and half of the prototype was assigned to our lab for further research and exploitation. In this paper we describe the prototype at our lab premises.

978-3-9819263-5-4/DATE21/©2021 EDAA



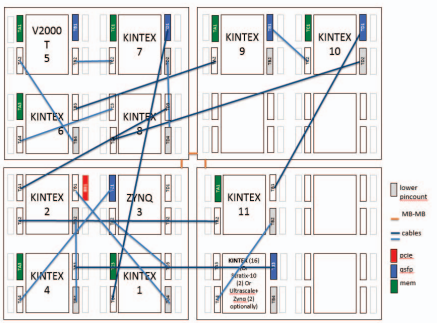


Fig. 1. MANGO cluster made of 12 FPGAs, 8 DDR memories. I/O PCIe and wiring. (a) Picture of the MANGO cluster. (b) Interconnection pattern.

Module	Description
MB-4M-R2	Motherboard to allocate four FPGAs and extension modules. Can be connected to other motherboards in a daisychain manner up to 4400 I/O signals (FPGA-FPGA), onboard OS and host data exchange system
FM-XCKU115-R1	Xilinx Kintex Ultrascale XCKU115 extension module board 75 Mbit internal memory, 7.9 M ASIC gates, 585 I/O pins: up to 1.2 Gbps, 56 high speed transceivers (GTH) at 12.5Gbps, 5500 DSP slices
FM-XC7Z100-R1	Xilinx Zynq™ XC7Z100 extension module board 2.7 M ASIC gates, Dual ARM Cortex-A9 MPCore with CoreSight, 1GB DDR3 memory, 260 I/O pins: up to 1.2 Gbps, 16 high speed transceives (MGT) at 12.5 Gbps 2 individually adjustable voltage regions
FM-XC7V2000T-R2	Xilinx Virtex XC7V2000T extension module board. 2 M ASIC gates capacity, 1084 i/o pins: up to 1.0 Gbps, 16 high speed transceivers (MGT) running at 12.5 Gbps, 8 individually adjustable voltage regions
FM-XCZU19EG	Xilinx Zynq™ UltraScale+ ZU19EG extension module board. 6.2 M ASIC gates, Quad-core ARM® Cortex™-A53 and Dual-core ARM® Cortex™-R5, 531 I/O pins: up to 1.2 Gbps, 16 high speed MGT transceivers at 16 Gbps, 4 individually adjustable voltage regions
MB-ICC	Motherboard interconnect cable.
IC-PDS-CABLE-R1	Extension modules interconnect cable
EB-PDS-DDR3-R2	SDRAM DDR3-1600 extension module board, 2 GByte of capacity. Data rate up to 1600 Mbps
EB-PDS-DDR4-R3	SDRAM DDR4-2400 extension module board, 2.5 GByte of Capacity. Data rate up to 2400 Mbps
EB-PDS-PCIe-Cable-R3	PCIe Gen 3 x 8 lanes extension module board
PCIe-DMBI	PCIe gen1, 4-lane board (part of PCIe DMBI gen1 kit) allows for PCIe communications between HOST and proFPGA system (MB and FPGA modules), data rate up to 3.2Gbps
EB-PDS-QSFP+-R1	QSFP+ interconnection board 2 QSFP+ connectors, fixed clock generator at 100 MHz, programmable clock generator

TABLE I
HN CLUSTER COMPONENTS DESCRIPTION

Different FPGA technologies are embedded on each MANGO cluster. Each cluster has four motherboards (each one hosting four FPGAs) including 10 Kintex XCU115 FPGAs[2] plus one Virtex 7 V2000T FPGA and one Zynq XC7Z100 FPGA. Eight DDR4 memory modules and two PCIe connections are placed on each cluster. Table I shows the components description for one cluster and Table II shows the most relevant specifications for each FPGA device.

The wiring between FPGAs within a cluster is shown in Figure 1, forming a 2D mesh. Pin restrictions and bandwidth unbalances motivated for this specific topology. Nevertheless, wiring can be changed and adapted to new communication requirements between FPGAs. All the clusters can be interconnected with optical cables forming a hypercube (not shown).

Communication between GNs and HNs is achieved via two different interfaces. Programming and management of FPGAs is performed via USB (for simplicity). Data communication is performed through PCIe interface. Transfer rates between GNs and HNs was always lower than 2Gbps, due mainly for the emulation purpose of the prototype. These rates need a boost when the propotype is being targeted as compute platform.

Parameter	V2000T	XCKU115	xC7Z100	XCZU19EG
Logic cells	1,954,560	1,451,100	444,000	1,143,450
CLB Flip-Flops	2,443,200	1,326,720	554,800	1,045,440
CLB LUTs	1,200,000	663,360	277,400	522,720
DSP Slices	2,160	5,520	2,020	1,968
Max Distr. RAM (Kb)	21,550	18,300	3,381	9,800
Total Block RAM (Kb)	46,512	75,900	26,500	34,600
Max. single-ended I/O	1,200	676	260	531
Max. single-ended HP I/O	-	676	250	572
Max. diff. HP I/O pairs	-	312	120	
Max. single-ended HR I/O	-	156	250	
Max. diff. HR I/O pairs	-	72	120	
Max. single-ended HD I/O	-	-	-	96
PCIe gen3	no	6	yes	5
Signal Rate Std. IO (Gbps)	1	1	1.2	1.2
MGTs	16	64	16	16
Signal Rate MGTs (Gbps)	12.5	16.3	12.5	12.5
ARM application processor	-	-	Dual-core Cortex™-A9 @667MHz	Quad-core Cortex™-A53 @1.5GHz
ARM extension/real-time processor	-	-	NEON™	Dual-core Cortex™-R5 @600 MHz

TABLE II
XILINX FPGAS SPECIFICATIONS

In sum, the MANGO prototype at UPV premises contains 96 high-end FPGAs with an aggregated performance of 700 TMACS[3], approximately.

III. MANGO PROTOTYPE ADAPTATION

The prototype served well as an emulation platform where specific software tools were provided for proper cluster management: bit-stream loading, booting the systems, and reconfiguring the FPGAs. Basic I/O communication was also achieved. For an emulation platform maximum achievable frequency is not a concern and thus low frequencies are usually adopted. Additionally, programmability is provided by the software ecosystem of the emulated platform. However, if one plans to use this prototype as a compute platform major issues emerge to enable an efficient utilization. We identified three main issues. The first one is programming support to allow the user easily offload computationg to FPGA devices (e.g through OpenCL). The second one is related to the multi-FPGA configuration of the cluster and the need to support it transparently by the programming perspective, and finally, another important aspect is the communication efficiency between GNs and HNs.

[2]Some clusters replace one XCU115 by one ZU19EG.

[2]FPGAs parameters details have been collected from specific products documentation available on https://www.xilinx.com/support.html

[3]TMACS stands for tera multiply-accumulates per second. This is a term used by marketing to describe the DSP performance in Xilinx FPGAs. Each DSP block accounts for one GMAC when operating at 1 Ghz.

A. OpenCL support

Traditionally, FPGA programming has been a complex task for application software developers, as it requires deep knowledge of the device as well as mastering hardware design tools and Hardware Description Languages (HDLs). Nevertheless, in order to push FPGAs into the HPC market, FPGA vendors (or third party related companies) develop FPGA-based acceleration platforms with the aim of simplifying the programming complexity in a large extend, while sustaining the high performance capabilities. Figure 2 illustrates the main components for one of these platforms. They are normally composed of a compiler, a software package that describes the hardware platform, an accelerator board and an offloading runtime engine. The most commonly used programming models in these platforms are OpenCL and High Level Synthesis (HLS). Both allow the developer to code an application kernel in a C-like manner. Thus, they are relieved from low level details of the device. The C source code is then translated (by means of a compiler) into hardware functions (CU blocks in Figure 2). These functions are used to program the reconfigurable partition of the hardware platform implemented in the FPGA. This increases the flexibility of FPGAs, as they can adapt its internal logic blocks to perform different hardware functions at different time. In order to reconfigure an FPGA and move data between the host and the device memory the FPGA provides a set of hardware blocks that implement communication interfaces with the host, peripherals and the kernel logic. These blocks are part of the hardware platform and they are placed in a static partition of the FPGA, which is known as *shell*. With this infrastructure, the FPGA can be accessed from the host through the OpenCL runtime and the drivers. To generate the hardware function for an application kernel the compiler uses a piece of software that describes all the components in the hardware platform, together with its configuration. The compiler uses that information to connect the application kernel logic to the rest of components in the hardware platform. It also has the capacity to suppress those interface components that are not strictly needed to implement an application kernel allowing more hardware resources for the application itself. Different vendors refer to this piece with different names: Design Support Archive (DSA) by Xilinx and Board Support Package (BSP) by Intel. From now on, we use Xilinx terminology and use DSA or hardware platform to refer to the same concept through the rest of the text.

The MANGO cluster does not come with most of elements to compose an acceleration platform (as its main target was prototyping). Thus, our purpose is to transform the MANGO prototype into an actual compute platform. Next, we describe the adaptation of a Xilinx reference hardware platform model, together with its corresponding software libraries and drivers to our system.

The Xilinx reference design has been modified to fit the specification of our physical FPGA infrastructure, which consists of a Kintex UltraScale 115 (XCKU115) FPGA with one 2GB DDR memory bank working at 1GHz and one PCIe Gen3 with 8 lanes interface to communicate with the host offering a theoretical maximum throughput of 8GB/s. The

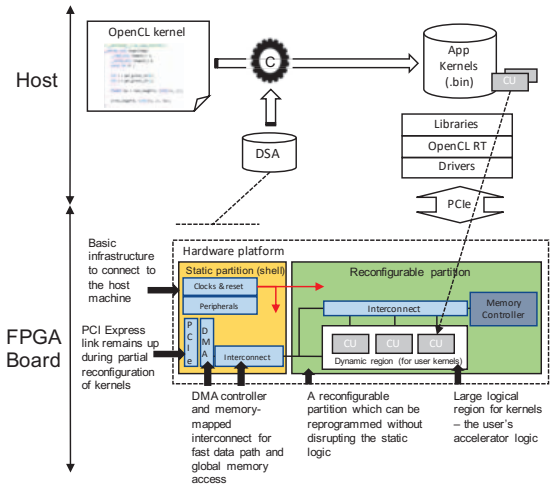


Fig. 2. FPGA-based acceleration platform diagram.

changes in the reference design took into account the Stacked Silicon Interconnect (SSI) technology of the target FPGA and in particular all the constraints related to it, i.e. the distribution of IP cores in the two Super Logic Regions (SLRs) and the addition of SmartConnect IPs that allow communication between IP cores belonging to different SLRs. The main consequence of this lays on the modification of the physical pin constraints for interfacing with the FPGA and the re-definition of the floorplan of the reconfigurable partition. A high level overview of the resulting hardware platform is also shown at the bottom of Figure 2. On the one hand, it contains in the static partition of the FPGA: I) clock and reset logic for the rest of elements in the platform, II) a PCIe controller that allows communication with the host, III) a DMA controller for efficient data transfers and IV) an Interconnect to connect the different elements in the design. On the other hand, it contains in the reconfigurable partition: I) a DDR controller to access the 2GB of memory available in the FPGA board, II) several Interconnects to connect the different elements in that partition with the elements in the static partition and III) a placeholder or dynamic region for the user application kernels.

The current platform is presented to the host as a memory-mapped device. This is achieved by means of the Advanced eXtensible Interface (AXI) memory-mapped protocol [6], which describes an interface between a single AXI master and a single AXI slave, that exchange information with each other based on a target address. It means that the access to every possible component in the DSA is carried out through a memory address, which for our particular case is depicted in Table III. Our platform supports two variants of this interface: I) AXI4-MM for high performance data transfers, and II) AXI4-Lite for simple and low-throughput memory-mapped communication (for example, to and from control and status registers). In total we support two different address spaces and every component is configured in a different offset inside of its address space and owns a range of it.

AXI Interface	Slave IP Core	Offset Address	Range	High Address
AXI4-MM Data Interface	DDR4 Controller	0x0000_0000_ 0000_0000	2G	0x0000_0000_ 7FFF_FFFF
	Trace offload FIFO for Application profiling	0x0000_0020_ 0000_0000	2G	0x0000_0020_ 7FFF_FFFF
AXI4-Lite Control Interface	Controlador DDR4	0x0006_0000	128K	0x0007_FFFF
	Clock Wizard Kernel clock source	0x0005_1000	4K	0x0005_1FFF
	Clock Wizard Kernel 2 clock source	0x0005_0000	4K	0x0005_0FFF
	DDR4 calibration status	0x0003_2000	4K	0x0003_2FFF
	GPIO for PR Isolation	0x0003_0000	4K	0x0003_0FFF
	GPIO for Feature ID	0x0003_1000	4K	0x0003_1FFF
	OpenCL Region	0x0000_0000	128K	0x0001_FFFF
	AXI Performance Monitor	0x0010_0000	64K	0x0010_FFFF
	Trace offload FIFO for application profiling	0x0011_0000	4K	0x0011_0FFF

TABLE III
DEVICE ADDRESS SPACE.

When developing our platform we took into account the capabilities of Partial Reconfiguration (PR) in FPGAs. Indeed, we chose the Expanded Partial Reconfiguration (XPR) type of PR, where most of the logic and DDR controllers are included in the reconfigurable partition. The benefit of using XPR lays in the minimization of the resources used by the static region, allowing to the compiler more flexibility for the implementation and optimization of application kernels.

In order to recognize the adapted hardware platform from the host, both the Xilinx OpenCL Hardware Abstraction Layer library (XCL HAL) and Xilinx OpenCL DMA (XCLDMA) driver were modified accordingly to our hardware platform specifications. We removed from the driver some specific features that were present in the reference model (but not needed in our design). We changed the Device ID and Subsystem ID of our PCIe device for the driver to detect our ad-hoc hardware platform correctly. Moreover, modifications to the XCL HAL meant adding the platform name, the relative new PCIe numbering and the new DDR size.

B. HN communication infrastructure

As previously mentioned, the MANGO prototype consists of 12 FPGAs interconnected within each HN cluster. FPGA-FPGA connections in the same cluster use the I/O pins available within each device. However, FPGA-FPGA connections at different clusters are also possible, using the so-called Multi-Gigabit transceivers. In order to take advantage of connectivity within and between clusters, the DSA must support such communication. Indeed, these communication channels must support DMA communication from the server with all the devices in an HN cluster, as well as to offload and control application kernels via OpenCL running on different FPGAs. At the same time, HN devices may communicate each other. To accomplish these goals Xilinx provides an IP core named Chip2Chip [7]. This IP core works like a bridge connecting two devices over an AXI-MM interface in compliance with AXI protocol specifications. The protocols supported are: SelectIO and Aurora.

SelectIO provides minimum latency between devices through FPGA I/O pins. Current prototype deployment uses the so-called High-Performance (HP) pins to connect the devices within a cluster. These pins are designed to meet the performance requirements of high-speed memory and other chip-to-chip interfaces. Chip2Chip (C2C) supports two versions of SelectIO: SelectIO SDR and SelectIO DDR. SelectIO DDR doubles the bandwidth without affecting the latency and the amount of resources needed.

The other protocol supported by C2C is Aurora. Aurora is a lightweight, serial communication protocol for multi-gigabit links. There are two versions of the protocol: Aurora 8B/10B and Aurora 64B/66B. They differ in the encoding used and throughput. While the throughput of Aurora 8B/10B ranges from 480Mb/s to 84.48Gb/s, on Aurora 64B/66B the throughput ranges from 500Mb/s to over 400Gb/s. Compared to SelectIO Aurora provides higher bandwidth while using less resources. For this reason, Aurora has been used for the connections between clusters.

C2C can be configured to work in master or slave mode. To connect two FPGAs with C2C one device has to be configured as master and the other device as slave. In Figure 3 a diagram of an example design with two FPGAs connected with C2C is shown. On this example the host has access to the memory on the slave FPGA through the C2C instances. Equally, provided the required logic is available, a kernel executing on the master FPGA can access the memory of the slave FPGA. As a result, this communication interface enables the host to offload and control kernels not only in the master FPGA, but also in the slave one.

As can be seen in Figure 3 the instances of C2C are located on the static region of the DSA. The reason to place the instances in this region is to prevent the link between devices to get affected after each reconfiguration process. During a partial reconfiguration process, the reconfigurable region is in reset mode while the static region is not. This is because the logic that manages the link between the host and the devices is placed on the static region. Placing the instances of C2C in the static region avoids the link to be reestablished after each reconfiguration process. However, the static region has a limited amount of resources thus, since additional logic has been added, the region has to be resized to include more resources.

To resize the static region floorplan design described in Section III-A has been modified. Out of all the resources available in an FPGA, clock resources and HP pins have been prioritised when modifying the floorplan. Since the FPGA can be connected with one or more FPGAs, the static region must have enough HP pins to satisfy the connectivity requirements. At the same time, several instances of C2C have to coexist thus clock resources are of critical importance. Increasing the size of the static region means that there will be less resources to implement OpenCL kernels. Therefore, floorplanning has to be carefully modified ensuring the static region has enough resources without affecting the reconfigurable region. Additionally, the AXI connectivity on the DSA has to be modified so that the C2C instances are connected with the appropriated cores on the design to achieve the functionality described before.

C. Efficient GN-HN data transfers

One of the key aspects to success in the use of heterogeneous devices for HPC, such as GPUs and FPGAs, lies in the setup

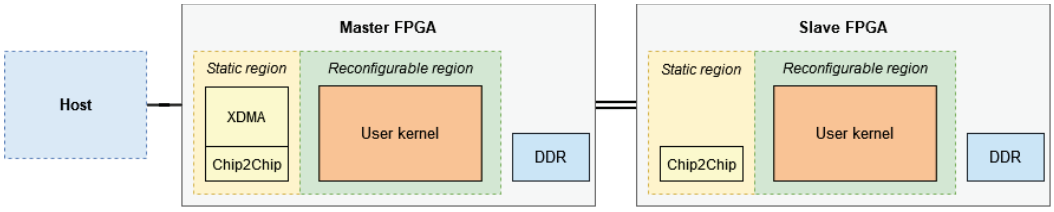


Fig. 3. Example design of two FPGAs connected with Chip2Chip.

of a high bandwidth and low latency efficient mechanism for data movement among the different computational components of the HPC system. In order to keep high performance affordable, communications should minimize the need for the CPU intervention when moving data between acceleration devices.

In our prototype, we are addressing this aspect from GN and HN perspectives, by means of DMA and Remote DMA (RDMA) mainly. For such a purpose, we have integrated a DMA controller (XDMA) in the master FPGA of every HN cluster, as it can be seen in in Figure 3, where a simplified high level design view of an HN is shown. In addition, we have added the MANGO prototype with RDMA support, a mechanism to support direct access to remote memory banks, without intervention of the CPU, via RDMA over Convergent Ethernet (RoCE) network.

Figure 4 illustrates the different possibilities for transferring data among different memories in the MANGO prototype. It shows in green the options already supported, while it shows in orange the options we plan to integrate as future work. In total we will support five type of data transfers, depending on the source and destination of them. The first one, consists of programming (via OpenCL functions) the DMA controller of an HN to move data from GN system memory to a HN device memory bank or vice versa. In this case, the transfer is limited to the GN and the HN the former is connected to via PCIe. The second one will consist in the movement of data between two whatever memory banks located inside of an HN. In this case, the transfers will be handled by the DMA controller of the target HN and they will be commanded (via OpenCL functions) by both the host or kernel applications. Both the third and fourth data transfers types use RDMA and are similar to each other. The unique difference lies in the location of the target memory banks. Therefore, we have a bunch of possibilities here to perform data transfers: between memory banks in different HNs (labelled as 3 in the figure), between a GN memory and a remote HN memory bank (labelled as 4 in the figure) or even between the memories of two different GNs (this communication is not shown in the figure). The use of RDMA as de facto standard for remote memory access in the MANGO prototype allow us the possibility to extend it with GPUs and put the different type of heterogeneous computing units to work cooperatively in an efficient manner. Finally, the fifth type of data transfer we are working on presents as targets memory banks in two different HNs, but in this case the data transfer will be not performed using RDMA, but ad-hoc OpenCL channels that the kernel application will

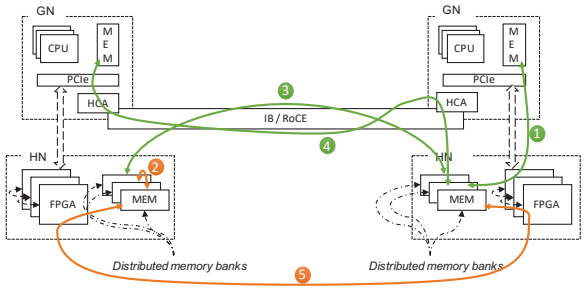


Fig. 4. Possible data transfers in the MANGO prototype.

be able to exercise. This mechanism couples computation and communication reducing to a large extend the communication overheads for specific applications. Therefore, it will be a very interesting option to speed up those application that fit.

IV. Results

The new hardware platform design for the FPGA devices has been synthesized and implemented using Vivado Design Suite 2017.4 [8]. Our basic *shell* occupies less than 8% of the total FPGA resources of a XCKU115 . Therefore, the majority of the resources are available for application acceleration. Figure 7 shows the floorplan of the design. The image on the left corresponds to the floorplanning of the design without communication between FPGAs whereas the image on the right corresponds to the design with support for communication between FPGAs. The area highlighted in yellow concerns to the static region and the remaining area accounts for the reconfigurable region. As mentioned in Section III-B, the static region needs to be resized to enable communication between FPGAs. Thus, the static region on the second design is more than twice the area of the static region in the first design. This way the region has more resources available for all the logic that has been added. Out of all the resources added to the static region, it is important to highlight the HP I/O banks 44 and 45 used for SelectIO.

Regarding the support for communication between FPGAs and following with the example design shown in Figure 3, we have measured the maximum bandwidth based on the data size when performing read and write operations on the local and remote FPGAs. The setup to perform these tests includes a PCie Gen3 with 4 lanes link with a maximum theoretical bandwidth of 4GB/s. The Chip2Chip instances on both devices have been configured to use SelectIO DDR with a data size of 64 bits

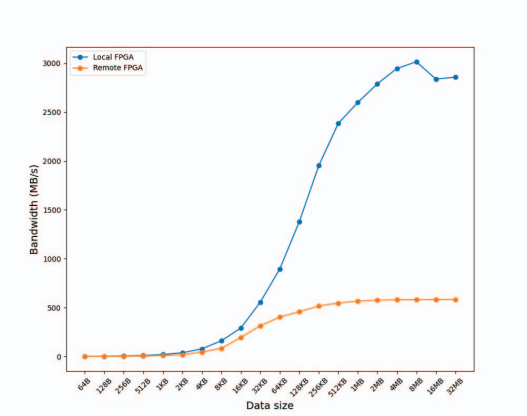


Fig. 5. Read bandwidth.

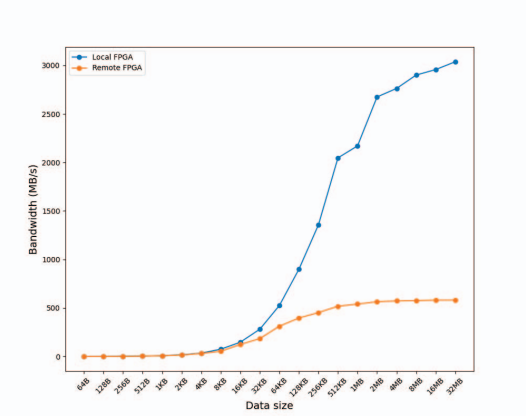


Fig. 6. Write bandwidth.

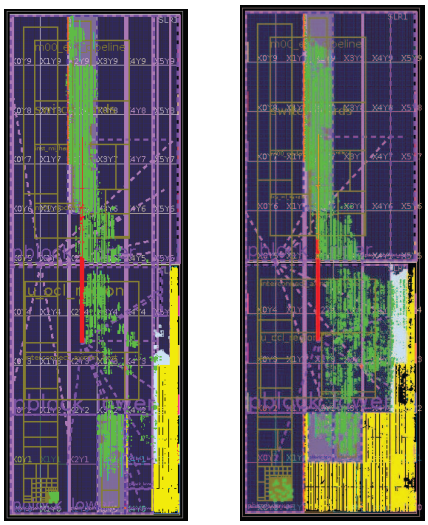


Fig. 7. Design floorplanning.

and the link working at a frequency of 150MHz. As driver we have used *xcldma* provided by Xilinx. To perform the tests we use the utilities that come with the driver (*dma_to_device* and *dma_from_device*).

In Figures 5 and 6 the results of these tests are shown. For each data size the tests have been executed 20 times so the values shown in these figures are the result of averaging out these executions time results. Differences between read and write operations are negligible. The maximum bandwidth obtained on the local FPGA is around 3000MB/s. This value is lower than the maximum theoretical bandwidth provided by PCIe. The reason for this mismatch could be related with the driver configuration used to carry out the tests.

Regarding the remote FPGA, the maximum bandwidth is around 580MB/s which is considerably lower than the bandwidth obtained on the local FPGA. This occurs because the link between FPGAs is limiting the bandwidth. There are two options to increase the bandwidth of the link and remove the current bottleneck: increase the number of pins used to connect the devices or increase the frequency the link is working on. The first approach is not viable on our current design since the static region has a limited number of resources and increasing the size of this region would lower the resources available for the kernels. Thus, increasing the frequency of the link is the most interesting option. C2C supports a maximum link frequency of 400MHz so it would be interesting to see the correlation between the frequency the link is working on and the effective bandwidth our communications achieve.

V. Conclusions and Future Work

We have described the adaptations being performed to evolve the MANGO emulation prototype into a compute platform. We have described the different hurdles that need to be nailed down and the complexities this effort posses. Although our work is not completed yet we have achieved a viable and functional solution which is currently being optimized and adapted to the full cluster configuration. As ongoing work, we plan to optimize the communication efficiency between FPGAs and between GN and HN. We also expect to run use cases from RECIPE and DeepHealth project in the prototype.

Acknowledgment

This work is supported by the European Commission through RECIPE and DeepHealth projects, under the Horizon 2020 program, grant number 801137 and 825111, respectively.

References

[1] (2017) The mango project website. [Online]. Available: http://www.mango-project.eu/
[2] (2018) The recipe project website. [Online]. Available: http://www.recipe-project.eu/
[3] (2019) The deephealth project website. [Online]. Available: https://deephealth-project.eu/
[4] W. F. P. Bellasi, G. Massari, "Effective runtime resource management using linux control groups with the barbequertrm framework," *ACM transactions on embedded computing systems*, vol. 2, no. 39, pp. 1–17, 2015. [Online]. Available: https://dl.acm.org/doi/10.1145/2658990
[5] "Prodesign electronic gmbh." [Online]. Available: https://www.prodesign-europe.com/
[6] Xilinx. (2017, Jul.) Vivado axi reference guide. ug1037-vivado-axi-reference-guide.pdf. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/
[7] ——, "Axi chip2chip logicore ip," Apr 2018. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_chip2chip/v5_0/pg067-axi-chip2chip.pdf
[8] Xilinx, "Vivado Design Suite," https://www.xilinx.com/products/design-tools/vivado.html, online; Last accessed on September 2020.