Implementing Hastlayer support for Xilinx SoC Zynq FPGA family

E. Dávid, D. El-Saig, Z. Lehóczky, and G.G. Barnaföldi

***Abstract*—Hastlayer by Lombiq Technologies aims to provide software developers of the .NET platform a tool to accelerate performance-critical parts of their programs with FPGAs. It is primarily intended to be used by software developers with no specific hardware design knowledge, allowing them to utilize the power of FPGAs with their existing skills. To achieve this FPGA agnostic aim, first, a dedicated firmware and software framework has to be developed for the target platform to enable the execution of the Hastlayer-generated core. In collaboration with Wigner GPU Laboratory of the Wigner Research Centre for Physics, we already developed several frameworks for Hastlayer. Here we present the implementation of Hastlayer support for the Xilinx SoC Zynq FPGA family.**

***Index Terms*—Field programmable gate arrays, High level synthesis**

I. Introduction

BEST performance on certain hardware requires specific programming usually at lower levels, on the other hand, this always restricts the flexibility of the applications. To find the golden way between high-level programming and specific hardware is an optimization Saint Graal.

The idea of high-level synthesis (HLS) in the field-programmable gate array (FPGA) firmware development or in more broadly in the RTL (Register Transfer Level) digital designs is quite old. It was first proposed approximately in the 1980s. The first tools appeared in 1994 from Synopsys [1], while the first real application in the industry was reported by Sony in 2001 [2].

On one hand, it is promising for the existing firmware developers working with traditional hardware description languages (HDL) like VHDL (VHSIC Hardware Description Language, where VHDL means Very High-Speed Integrated Circuits) and Verilog to capture the complexities of the design in a high-level way and generate the low-level implementation from that abstraction. It would help to manage the bigger and more complex designs with more ease and in a faster way because a complex design usually starts with some models written in high-level languages (e.g., MATLAB, C++, SystemC) which will later be used as a golden reference.

On the other hand, HLS is also promising for traditional software developers. Because some algorithms are more suited for FPGAs than CPUs (Central Processing Unit) or GPUs (Graphics Processing Unit), and it would be quite efficient to move the algorithm already running on a CPU to an FPGA without any register transfer level (RTL) digital design-specific expertise.

Unfortunately, this goal was never fully achieved. There were proprietary tools like Catapult C from Mentor Graphics (actively developed between 2004 and 2011) [3] or Impulse C (between 2003 and 2009), which was famous for its high-frequency trading application. But they never became as widespread as the standard FPGA vendor tools.

Later from the early 2010s, the two major FPGA vendors Altera (now Intel) and Xilinx (now part of AMD), started to actively develop their HLS solutions closely integrated with their existing tools. Fortunately, this strategy continues, and today we have full FPGA OpenCL support from both vendors. Due to the improvement of technology and market forces, today we have FPGA accelerator cards in the main public cloud data centers just as GPU cards (e.g., Amazon F1, Azure Alveo cards).

Hastlayer from Lombiq Technologies [4] can be considered as an HLS tool that converts some parts of the .NET application to RTL, which can be implemented by FPGAs. But it aims more to fully hide the FPGA and OpenCL details from the targeted .NET software developers. To deliver this high-level feature to the .NET end-users, supporting tools and firmware and software frameworks need to be developed to hide the FPGA and OpenCL details and present a more abstract interface for the upper layers.

Wigner Research Centre for Physics' DAQ and Wigner GPU Laboratory plays key role in the FPGA developments for high-energy physics (HEP) [5-6].

In a joint project of the Wigner GPU Laboratory [7] and Lombiq Technologies [8], have already developed several Hastlayer support platforms for different FPGA cards. The first

The research was supported by the Hungarian National Research, Development and Innovation Office (NKFIH) under the contract numbers OTKA K135515. The support by the Wigner GPU Laboratory is also appreciated.

E. Dávid is an FPGA developer of the Wigner DAQ Laboratory, Wigner Research Centre for Physics, 29-33 Konkoly-Thege Miklós Str, H-1121 Budapest Hungary (e-mail: erno.david@wigner.hu).

D. El-Saig is a software developer at Lombiq Technologies Ltd., 14 Zrínyi Str, H-1051 Budapest (e-mail: david.el-saig@lombiq.com).

Z. Lehóczky, is the director of the Lombiq Technologies Ltd., 14 Zrínyi Str, H-1051 Budapest, Hungary (e-mail: zoltan.lehoczky@lombiq.com).

G. G. Barnaföldi is the leader of the Wigner GPU Laboratory, Wigner Research Centre for Physics, 29-33 Konkoly-Thege Miklós Str, H-1121 Budapest Hungary (e-mail: barnafoldi.gergely@wigner.hu).

was the Microsoft Catapult platform based on Altera Stratix V FPGA. The second was the Xilinx Alveo data center cards family based on Virtex UltraScale FPGAs [9]. And this report provides details about the latest phase: supporting the Xilinx Zynq-7000 SoC embedded FPGA family.

In the following sections, we describe the Hastlayer generated core, the targeted test platforms, the Xilinx Vitis OpenCL RTL Kernel acceleration flow, how to scale the kernel clock frequency to the maximum safest value, and at the end, we present our test results.

II. The Hastlayer-generated core

The Hastlayer SDK (Software Development Kit) [10] is the main toolset for Hastlayer development. It covers the whole .NET acceleration flow: transforms the .NET assemblies, builds the FPGA firmware, programs the target platform, and manages the communication.

In the first step, the SDK converts the .NET code into a VHDL module (the Hastlayer-generated core), which behaves the same way as the original code. This core has few control and status lines and a custom 32-bit data bus to communicate with the outside world (Figure 1).

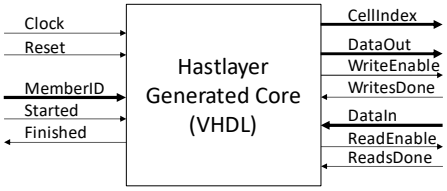

Fig 1. Hastlayer-generated core I/O ports

To enable Hastlayer to use the memory, read and write access in a given .NET class member function, the .NET code should access the data buffer through a specific class called SimpleMemory, which is provided by the Hastlayer SDK.

The following simple example increments the first 32-bit integer value by one in the data buffer:

```
public class SimpleIncrementExample
{
  public virtual void Run(SimpleMemory memory)
 {
    int cellIndex = 0;
    int value = memory.ReadInt32(cellIndex)
    memory.WriteInt32(cellIndex, value + 1);
  }
}
```

To implement a Hastlayer-generated core in an FPGA and to enable communication with the main .NET application the appropriate firmware and software layers have to be developed. On the firmware level, the core has to be extended with a control state machine and with a memory interface to provide access to the main memory (typically external DDR memory). On the software level, there must be an additional API layer that allows core invocation and passing and receiving a data buffer to and from the core. This support layer can be implemented in different ways. It can be completely customized as it is in the case of Nexys card support [11], where the .NET side and the FPGA card is connected with a USB cable. Or it can be based around PCIe bus communication as in the case of the Microsoft Catapult platform and Xilinx Alveo cards.

III. Target platforms

This project phase targeted the embedded FPGA segments typically with more limited FPGA resources than the data center variants and typically with built-in ARM CPU cores. In the previous phase, we just finished the Xilinx data center Alveo card family support, which was natural to continue with another FPGA family from Xilinx.

The Xilinx Zynq-7000 SoC FPGA family is one of the popular FPGAs in the embedded segment (e.g., in the aerospace industry). Their main feature is the built-in dual-core ARM Cortex A9 CPU @667 MHz with an internal DDR memory controller and with the capability to run full-featured 32-bit Linux distributions like Ubuntu or in our case the officially more supported PetaLinux. The Zynq-7000 family is fully supported by the Xilinx Vitis tool acceleration flow.

For initial testing, the ZedBoard Zynq-7000 ARM/FPGA SoC development board was chosen from Digilent as a widely used development kit for Zynq projects (equipped with Zynq 7020 FPGA and 512MB DDR2 RAM). Later we switched to an SoC module from Trenz Electronic (TE0715-04-30-1C equipped with Zynq 7030 FPGA and 1024 MB DDR2 RAM).

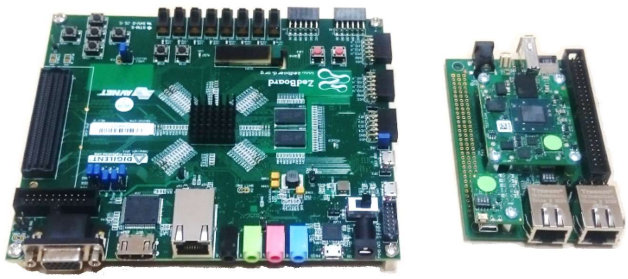

Fig 2. Digilent ZedBoard Zynq-7000 and Trenz TE0715-04-30-1C SoC module

IV. Xilinx Vitis OpenCL RTL kernel

The Xilinx Vitis application acceleration development flow is based on the industry-standard OpenCL environment and this flow is fully supported on the Zynq-7000 family too [12]. This primarily means that kernels written for GPUs in the OpenCL language can be implemented on Xilinx FPGAs with little modifications. In addition to the OpenCL language the Vitis flow also allows providing the kernel in RTL form as VHDL or Verilog source code which allows us to easily embed the Hastlayer-generated VHDL core into the RTL kernel framework and use the OpenCL flow for accelerating the main .NET application.

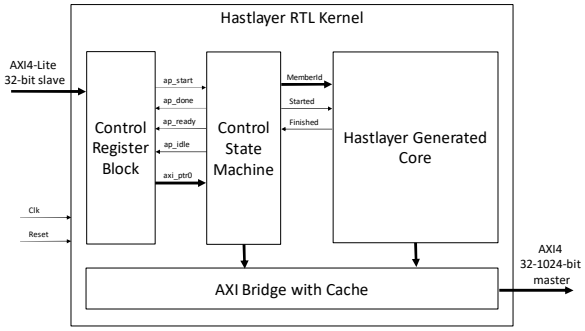

Fig 3. Hastlayer-generated core as RTL kernel block diagram

The main interfaces of the Vitis RTL OpenCL kernel: clock and reset, 32-bit AXI4-lite slave interface accessing the kernel's control and status registers from outside, and at least one built-in AXI4 master interface to access the main memory (between 32 and 1024-bit wide data bus).

The control register block and the control state machine are responsible for handling the incoming data buffer, extract the header information, execute the Hastlayer-generated core, and at the end pack the status information back into the header.

The custom AXI bridge and cache module handle the memory read and write requests from the Hastlayer-generated core and the control state machine. Both modules use the Hastlayer custom 32-bit interface, which has to be adopted to the AMBA AXI4 bus interface. The AXI4 interface typically matched with the DDR memory controller data width and can be between 32 and 1024-bit to match the external DDR memory burst sizes. This allows more optimal DDR bandwidth usage but increases the overhead in 32-bit accessing mode. To mitigate this, the bridge implements a single associative cache line. When the Hastlayer-generated core issues write or read requests, first it is checked against the cache line. If the requested address is in the cache line, the request is handled immediately. If it is missed, then the core has to wait until the data arrives or is written back to the DDR memory. This significantly decreases memory access times in consecutive cases.

By design, the Hastlayer-generated core is free to modify the whole input buffer. There is no separated input and output memory area. At the end of the processing, the CPU side will receive the whole buffer.

Based on this working mode the RTL kernel function receives the starting address of the data buffer, which holds a small header section at the beginning, following with the actual data content. This way the number of OpenCL memory buffer transactions is reduced to one. This allows faster kernel execution times. On a PCIe-based machine, it requires one host-to-device transfer at the beginning and one device-to-host transfer at the end of the kernel execution. On Zynq-based cards, since the DDR memory is shared between the processor system and the FPGA fabric, there is no actual data movement. The OpenCL buffer is allocated in the physically contiguous memory area and from the Hastlayer core accessible as a single memory area.

This manifests as:

The OpenCL kernel function definition:
void hastip(unsigned int *buffer)

```
Input:
offset = buffer[0]
memberId = buffer[1]
dataArray = buffer[offset + 0 : offset + datasize - 1]

Output:
buffer[2] = hardwareExecutionTimeLo (lower 32-bit)
buffer[3] = hardwareExecutionTimeHi (upper 32-bit)
```

The overall system architecture is presented in Figure 4. The main .NET application runs in the PetaLinux on the embedded ARM CPU side. The Microsoft .NET Core SDK v3.1 fully supports Zynq's 32-bit ARM Cortex A9 processor systems. The OpenCL RTL kernel with the Hastlayer-generated core resides in the programmable FPGA fabric section and communicates with the CPU side over an AXI interface. The main AXI interconnect resides in the processor block and connects together the CPU and the kernel with the built-in DDR memory controller.

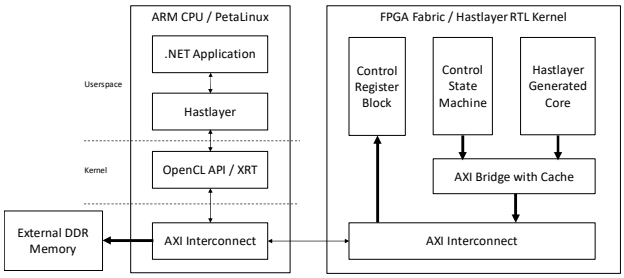

Fig 4. Overall system architecture

The OpenCL RTL kernel with the Hastlayer-generated core invocation sequence is presented in Figure 5.

1) In the first step the .NET layer prepares the OpenCL kernel buffer and fills the required header information and the user data for processing.
2) Through standard OpenCL kernel invocation function calls the API and the XRT triggers the control and state machine (with register writes) to start the Hastlayer core launch.
3) First, the header information is parsed, and based on this information the appropriate member function is started.
4) During the execution, only the core has access to the data buffer.
5) When the core is done, the control state machine flushes the cache and fills the header with the execution status details.
6) After executing the kernel, the buffer is returned to the .NET application and it sees exactly the same content as if executed by the original .NET code.

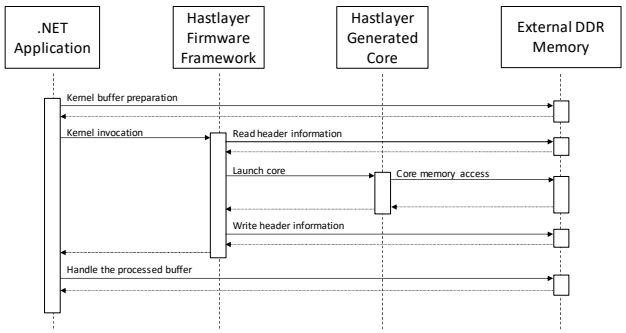

Fig 5. Hastlayer kernel invocation sequence

V. Implementing Kernel Clock Frequency Auto Scaling

One of the difficulties when the FPGA firmware source code is generated from a higher-level language like .NET is that the resulting combinational logic complexity in the digital design can vary between wide ranges depending on the complexity of the initial .NET code, influenced by the chosen parallelism and other implementation details. This can degrade the maximum kernel frequency, especially when the FPGA resource usage becomes denser.

In a handwritten FPGA firmware, this is easier to manage. It is typically designed to use a fixed clock frequency and the design verified during the STA (Static Timing Analysis) to meet the required setup and hold timing rules. And when the timing fails, the developer has to fine-tune the design to meet the timing.

But in high-level synthesis cases like the Hastlayer, this approach is less available to the .NET developers. In the case of the data center Alveo cards (which is the main target for the Xilinx OpenCL tools), this is solved by the kernel clock auto-scaling feature. At the end of the fitting phase when the static timing analysis reports are available, the maximum kernel frequency is recalculated based on the worst negative slack. This value is saved together with the XCLBIN file, and later during the execution, this information is used to configure the kernel clock frequency. This is a very useful feature because the user always gets a runnable kernel on the first try without any technical intervention. Unfortunately, this feature is not supported during the Vitis embedded design flow. Only fixed clock frequency modes are supported. If the routing fails at the targeted frequency, then it has to be repeated again with a lower clock frequency. This is a time-consuming process and the existing users found the auto-scaling feature more convenient.

To achieve a similar feature on a Zynq-7000 family FPGA three main steps had been done:

1) Instead of an MMCM/PLL in the fabric region configured by Vitis, the kernel clock is provided by the PS (Processor System) through one of the PS to PL configurable clock source (FCLK0). This PLL is reconfigurable during a normal operation between 1 and 250 MHz.

2) At the end of the Vivado routing phase the new kernel clock frequency has to be recalculated based on the STA timing report. Each failing path has to be evaluated according to the timing requirements with multi-cycle exceptions and the kernel clock period must be increased by the largest negative slack. At the end of this process, this clock information has to be updated in the XCLBIN file.

3) During the execution on the Zynq device the kernel clock frequency information has to be extracted from the XCLBIN file and the FCLK0 clock should be reconfigured according to this. To achieve this, the PetaLinux kernel must be built with the "Xilinx PL clock enabler (xlnx, fclk)" driver enabled.

VI. Results

The Hastlayer SDK contains seventeen sample algorithms covering different use case scenarios, beginning from a simple memory read and write example to more complex ones e.g. image processing or prime number calculations, and floating point computations made with the posit number format [13] (for the source code details see the GitHub repository) [14]. During the development, we used these built-in samples to test the functionality and benchmark the results.

Table I summarizes the overall performance results of the samples: the execution time of the pure .NET version running only on the CPU, the achieved kernel clock frequency, the degree of the level of parallelization in the algorithm, the time spent in the kernel (net FPGA) and the total execution time of the accelerated .NET code (total FPGA), and the calculated speedup value.

TABLE I
Overall speedup test results

Algorithm	CPU [ms]	Kernel freq. [MHz]	Paral-lelism	Net FPGA [ms]	Total FPGA [ms]	Speed up
FSharpParallelAlgorithm	22137	143	280	209	249	**88,9**
Fix64Calculator	4416	143	10	769	801	**5,5**
GenomeMatcher	0	143	-	0	3	**-**
ImageProcessingAlgorithms	2839	111	25	29	63	**45,1**
Loopback	0	143	-	0	3	**-**
MemoryTest	0	143	-	0	3	**-**
MonteCarloPiEstimator	3151	125	77	31	40	**78,8**
ObjectOrientedShowcase	17	143	-	0	413	**-**
ParallelAlgorithm	20552	143	260	210	240	**85,6**
Posit32AdvancedCalculator	1	111	-	0	5	**-**
Posit32Calculator	133	42	2	555	572	**0,2**
Posit32FusedCalculator	86	143	-	44	430	**0,2**
PositCalculator	46233	143	-	6875	7274	**6,4**
PrimeCalculator	391	111	30	50	106	**3,7**
RecursiveAlgorithms	6	143	-	0	470	**-**
SimdCalculator	2	125	20	0	3	**-**
UnumCalculator	135	143	-	5	409	**0,3**

In most cases, the Hastlayer-generated VHDL core (net FPGA) outperformed the pure software version (CPU), except in the case of PrimeCalculator. But when we look at the total execution time of accelerated version (total FPGA) then the results are less favorable. Depending on the samples there is an additional execution overhead between 3 and 400 milliseconds

(kernel invocation, communication, etc.) which degrades the overall performance. Also in some cases, the software example executed so fast that we were unable to measure the execution time reliably in PetaLinux to calculate correctly the speedup value.

Table II and Figure 6 summarizes the impact of different AXI bus data widths and kernel clock frequency scenarios in a synthetic memory test sample (MemoryTest algorithm) where the average cell increment time was measured.

TABLE II
AVERAGE MEMORY CELL INCREMENT TIMES IN DIFFERENT AXI BUS DATA WIDTH AND KERNEL FREQUENCY SCENARIOS (MEMORYTEST ALGORITHM)

AXI bus data width	50 MHz	100 MHz	150 MHz	200 MHz	250 MHz
32-bit	632 ns	374 ns	299 ns	271 ns	245 ns
64-bit	354 ns	207 ns	163 ns	145 ns	132 ns
128-bit	260 ns	150 ns	109 ns	94 ns	82 ns
256-bit	202 ns	109 ns	78 ns	66 ns	57 ns
512-bit	170 ns	89 ns	63 ns	51 ns	43 ns
1024-bit	156 ns	81 ns	56 ns	43 ns	36 ns

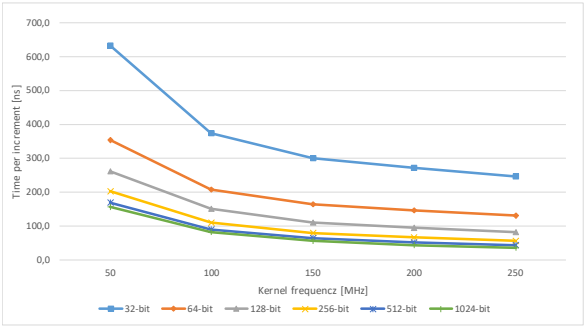

Fig 6. Average memory cell increment times in different AXI bus data width and kernel frequency scenarios (MemoryTest algorithm)

The results show that below certain kernel frequency and data width the performance is suboptimal but above certain limits, the result is saturated. If we include the results from Table V then we can see that there are tradeoffs between the AXI bus data width and the achievable kernel clock frequencies.

VII. CONCLUSION

This report described the main key concepts implementing Hastlayer support for the Xilinx SoC Zynq FPGA family and presented some test results. The project achieved its main goals. Now it is possible to run hardware-accelerated .NET applications on the Zynq devices. The test results show that moving the parallel computations into the FPGA fabric can produce significant speedup in the execution time compared to executing the whole algorithm on the built-in ARM CPU cores. What remains is to further optimize the resource usage of the Hastlayer-generated core and the supporting firmware framework to achieve better resource utilization and higher kernel clock frequency.

APPENDIX

Tables III, IV, and V summarizes the achievable final kernel clock frequencies at different target frequencies. At 100 MHz, almost all samples are routable except the Posit32AdvancedCalculator, Posit32Calculator, PrimeCalculator, and SimdCalculator sample, which contains somewhat larger combinational logic. At 250 MHz, the result is much more diverse. None of the samples are routable at that high frequency in a Zynq FPGA, and the different final frequencies after the auto-scaling procedure represent the overall complexities of a given sample.

TABLE III
SCALED KERNEL CLOCK FREQUENCIES TARGETING 100 MHZ

Algorithm	**AXI bus data width [bit]**					
	32	**64**	**128**	**256**	**512**	**1024**
FSharpParallelAlgorithm	100	100	100	100	100	100
Fix64Calculator	100	100	100	100	100	100
GenomeMatcher	100	100	100	100	100	100
ImageProcessingAlgorithms	100	100	100	100	100	100
Loopback	100	100	100	100	100	100
MemoryTest	100	100	100	100	100	100
MonteCarloPiEstimator	100	100	100	100	100	100
ObjectOrientedShowcase	100	100	100	100	100	100
ParallelAlgorithm	100	100	100	100	100	100
Posit32AdvancedCalculator	94	94	94	94	94	94
Posit32Calculator	58	56	61	59	56	54
Posit32FusedCalculator	100	100	100	100	100	100
PositCalculator	100	100	100	100	100	100
PrimeCalculator	78	78	78	77	77	81
RecursiveAlgorithms	100	100	100	100	100	100
SimdCalculator	99	99	97	96	98	98
UnumCalculator	100	100	100	100	100	100
Average	**96**	**96**	**96**	**96**	**96**	**96**

TABLE IV
SCALED KERNEL CLOCK FREQUENCIES TARGETING 150 MHZ

Algorithm	**AXI bus data width [bit]**					
	32	**64**	**128**	**256**	**512**	**1024**
FSharpParallelAlgorithm	142	142	142	142	142	142
Fix64Calculator	142	142	142	142	142	142
GenomeMatcher	142	142	142	142	142	142
ImageProcessingAlgorithms	115	112	113	115	115	113
Loopback	142	142	142	142	142	142
MemoryTest	142	142	142	142	142	142
MonteCarloPiEstimator	132	131	129	133	131	133

ObjectOrientedShowcase	142	142	142	142	142	142
ParallelAlgorithm	142	142	142	142	142	142
Posit32AdvancedCalculator	126	126	125	127	121	112
Posit32Calculator	42	44	43	41	41	43
Posit32FusedCalculator	142	142	142	142	142	142
PositCalculator	142	142	142	142	142	142
PrimeCalculator	126	125	123	123	123	122
RecursiveAlgorithms	142	142	142	142	142	142
SimdCalculator	142	142	142	142	142	141
UnumCalculator	142	142	142	142	142	142
Average	**132**	**132**	**132**	**132**	**131**	**131**

TABLE V
SCALED KERNEL CLOCK FREQUENCIES TARGETING 250 MHZ

Algorithm	**AXI bus data width [bit]**					
	32	**64**	**128**	**256**	**512**	**1024**
FSharpParallelAlgorithm	149	167	152	158	140	137
Fix64Calculator	189	192	189	179	156	150
GenomeMatcher	187	176	182	166	160	146
ImageProcessingAlgorithms	113	111	115	112	112	110
Loopback	221	221	208	187	175	175
MemoryTest	189	193	187	180	160	156
MonteCarloPiEstimator	129	130	127	125	126	127
ObjectOrientedShowcase	217	221	207	186	180	170
ParallelAlgorithm	174	166	164	160	156	150
Posit32AdvancedCalculator	95	94	94	84	73	43
Posit32Calculator	55	57	60	58	58	55
Posit32FusedCalculator	160	166	162	161	157	144
PositCalculator	225	225	210	187	160	162
PrimeCalculator	64	68	66	51	53	39
RecursiveAlgorithms	177	172	152	165	168	155
SimdCalculator	78	67	80	58	49	48
UnumCalculator	193	197	176	179	162	153
Average	**154**	**154**	**149**	**141**	**132**	**125**

REFERENCES

[1] Architectural Design of DSP ASICs: Tools and Techniques, Proceedings of the 1995 DSPx Exhibition and Symposium, April 1995.

[2] G. Martin and G. Smith, High-Level Synthesis: Past, Present, and Future, *IEEE Design & Test of Computers* vol. 26, no 4, pp- 18-25 July-Aug. 2009. DOI: 10.1109/MDT.2009.83

[3] Calypto acquires Mentor's Catapult C, 2011, Accessed on: June 15, 2021, [Online] Available: https://www.eetimes.com/calypto-acquires-mentors-catapult-c

[4] Z. Lehoczky et al., "Turning Software into Hardware — Hastlayer", 2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS), 16622630, January 2017 DOI: 10.1109/iNIS.2016.051

[5] AP. Buncic *et al.* [ALICE Collaboration], "Technical Design Report for the Upgrade of the Online-Offline Computing System", CERN-LHCC-2015-006, ALICE-TDR-019, 2019. Accessed on: June 15, 2021, [Online] Available: https://cds.cern.ch/record/2011297/files/ALICE-TDR-019.pdf

[6] M. Arrigui *et al.* [ALICE Collaboration], "The ALICE DAQ: current status and future challenges", *Computer Physics Communications,* vol. **140**, no 1–2, pp. 117-129, 2001. DOI: https://doi.org/10.1016/S0010-4655(01)00262-4.

[7] Wigner GPU Laboratory Accessed on: June 15, 2021, [Online] Available: http://gpu.wigner.hu

[8] Lombiq Technologies, Accessed on: June 15, 2021, [Online] Available: http://lombiq.com

[9] GPU Day 2020, Hastlayer, Implementing on Xilinx Alveo Accelerator Cards, https://gpuday.com/#schedule

[10] Hastlayer-SDK, Accessed on: June 15, 2021, [Online] Available: https://github.com/Lombiq/Hastlayer-SDK

[11] Hastlayer website, Accessed on: June 15, 2021, [Online] Available: https://hastlayer.com/

[12] Xilinx, Vitis Platform,Accessed on: June 15, 2021, [Online] Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[13] Z. Lehoczky et al., "High-level .NET software implementations of unum type I and posit with simultaneous FPGA implementation using Hastlayer", CoNGA '18: Proceedings of the Conference for Next Generation Arithmetic, No. 4 pp 1–7 March 2018 DOI: 10.1145/3190339.3190343

[14] Hastlayer SDK GitHub page: Accessed on: June 15, 2021, [Online] Available: https://github.com/Lombiq/Hastlayer-SDK

E. Dávid received the B.S. degree in engineering informatics from Kandó Kálmán Polytechnic, Budapest, Hungary in 1997 and the M.S. degree in computer engineering from Budapest University of Technology and Economics, Budapest, Hungary in 2011.
He becomes a junior research fellow in Wigner Research Centre for Physics in 2014 and joined the CERN LHC ALICE Collaboration, where he participating in different projects as an FPGA firmware developer.

D. El-Saig was born in Budapest, Hungary in 1990. Since 2018 he has been a software engineer at Lombiq Technologies Ltd, Budapest, Hungary. He was a software engineer at ALGY Mining Ltd between 2013 and 2020.
He presented in the IEEE Symposium on Computation Intelligence and Informatics of 2018 as an associate of Antal Bejczy Center for Intelligent Robotics, Óbuda University, Budapest, Hungary.

Z. Lehóczky co-founder and managing director of Lombiq Technologies (https://lombiq.com), originator of the Hastlayer project. His main expertise is in .NET software development and software architecture design. Active in open-source, core contributor of the ASP.NET (Core) CMS Orchard, guest lecturer at Óbuda University, John von Neumann Faculty of Informatics.

G.G. Barnaföldi (PhD) received the M.S. degree in physics and astronomy at the Eötvös Loránd Univerisity, Budapest Hungary in 2001, he did the Ph.D. degree in 2006 at the same place.

He was an assistant professor at the Information Technology Laboratory between 2000 and 2008. In parallel, starting from 2001 he was employed as a young research fellow by the KFKI RMKI of the Hungarian Academy of Sciences. In 2008 he moved to the Kent State University, Kent, Ohio, USA for a year as a PostDoc. Since 2009 he is at the Wigner Research Centre for Physics, leading the Wigner GPU Laboratory, the Hungarian ALICE Group, and the Heavy-ion Research group at the Department for Theoretical Physics.

He joined the CERN LHC ALICE Collaboration in 2005, where he is the national representative since 2013.

Dr. Barnaföldi's awards and honors include the Györgyi Géza prize, the Bürgen Scholar by the Academia Europea, the Physics Prize of the Hungarian Academy of Sciences, and the Best Talk prize of the 2016 Beijing Science and Technology Fair.