Case Studies of Configurable Binary Design Library on FPGA

1st Vega Mario
Engineering Department
University of Houston-Clear Lake
Houston, USA

2nd Mukesh Chowdary Madineni
Engineering Department
University of Houston-Clear Lake
Houston, USA

3rd Benjamin Garrett
Engineering Department
University of Houston-Clear Lake
Houston, USA

4th Xiaokun Yang*
Engineering Department
University of Houston-Clear Lake
Houston, USA

5th Hailu Xu
Department of Computer Engineering & Computer Science
California State University, Long Beach
Houston, USA

2022 International Symposium on Measurement and Control in Robotics (ISMCR) | 978-1-6654-5496-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISMCR56534.2022.9950580

***Abstract*—This paper presents a configurable binary design library including fundamental arithmetic circuits like full-adder, full-subtractor, binary multiplier, shifter, and more. The Chisel Hardware Construction Language (HCL) is employed to build the parameterizable designs with different precision including half-word, word, double-word, and quad-word. Chisel HCL is an open-source embedded domain-specific language that inherits the object-oriented and functional programming aspects of Scala for constructing hardware. Experimental results show the same accuracy achieved by our proposed work compared with the Verilog HDL implementations. The hardware cost in terms of slice count, power consumption, and the maximum clock frequency is further estimated. Compared with traditional design intellectual properties (IPs) provided by IP vendors, our proposed work is configurable and expandable to the other arithmetic implementations and projects.**

***Index Terms*—FPGA, arithmetic circuits, Chisel Hardware Construction Language, Verilog Hardware Description Language**

I. Introduction

The field-programmable gate array (FPGA) vendors like Intel-Altera and AMD-Xilinx allow integrated circuit designers creating complex circuits by instantiating and interconnecting intellectual properties (IPs) from their provided tools like Altera Quartus [16] and Xilinx Vivado [17]. All these IPs are hard cores which can be synthesized and implemented through the FPGA design flow. Additionally, the design netlist can be demonstrated on the selected FPGA board. Though all the IPs are configurable and reusable by FPGA designs, they are not open to the application-specific integrated circuit (ASIC) tools, such as the simulators like Synopsys VCS [18], Cadence NC [20], and Mentor Graphic Modelsim/Questa [22], and synthesis tools like Synopsys Design Compiler [19] and Cadence Genus Synthesis Solution [21].

No doubt that it is a big challenge for academia and industry to start an integrated circuit design without existing IPs. An open-source design library thus is crucial to ASIC designers as well as the FPGA development groups. The other challenge for the IP vendors is that the provided IPs must be configurable and easy-to-use for different design specifications including but not limited to precision needs, speed constraints, chip size considerations, power dissipation, etc.

Under this context, this paper proposes a case study to the configurable designs on a binary library including six basic arithmetic operators. In industry, it is very common to generate some Verilog Hardware Description Language (HDL) designs using HDL generators instead of writing Verilog code from scratch, particularly for the design projects of reusable IPs, standardized interfaces, and bus protocol based wrappers and bridges. Most of such generators were written by script language such as Perl and/or some high-level synthesis tools. The main concerns of such design approaches are: 1) there is no powerful library behind the generators, making the design on the generator very complicated; 2) most of the designs are not parameterizable so that they are not configurable and expandable to the other projects.

To overcome above-mentioned constraints, this paper focuses on exploring the Hardware Construction Language (HCL) generation using Chisel language. Chisel HCL is basically a hardware design language/generator created by Lawrence Berkeley Research Lab that supports advanced hardware design by using highly parameterized generators [10], [11]. It provides circuit generation and reuse of design components for both ASIC and FPGA-based digital circuit designs [1]. Chisel adds hardware construction primitives to Scala embedded programming language, providing designers to construct parameterizable circuit generators that produce synthesizable Verilog HDL.

Using the Chisel platform, it is able to make the binary design library open to ASIC simulators and synthesis tools like Synopsys VCS and Design Compiler, and Cadence NC and Genus Synthesis Solution. The binary design library includes the fundamental arithmetic circuits of full-adder, full-subtractor, multiplier, shifter, leading one detector, and two's complement. After demonstrating the validity, the proposed design library can be expanded into many other design modules like the floating-point arithmetic components and network neurons. Below are the main contributions of this work:

- This paper presents a design flow from Chisel HCL description to Verilog HDL design to the final hardware cost evaluation. As case studies, the binary design library is implemented and evaluated in terms of accuracy, design schematics, speed, slice count, and power consumption.
- Experimental results show the validity of the proposed design library which is able to achieve 100% accuracy of the binary design computation. By parameterizing the precision of each design module, different designs with different speed-area-power tradeoff can be obtained.

The organization of this paper is as follows: it first reviews the related works in Section II. The proposed approach with FPGA designs is discussed in Section III. Section IV introduces the implementations of this research work and Section V further demonstrates the experimental results in terms of accuracy, slice count, maximum operational frequency, and power consumption. In the last two sections the conclusion and future works of this paper are summarized.

II. Related Works

The arithmetic logic units are the fundamental building blocks of system-level circuits and design applications. A modern real-time application requires very powerful and complex arithmetic operators in its design, from a basic microprocessor to image/video processing unit to complex neural networks [13]–[15].

Previous works on designing such arithmetic circuits have mainly concentrated on building high-speed and/or power-efficient circuits using HDL or other design methodologies such as high-level synthesis [5], [12]. Based on different design specifications, configurable designs over different projects are needed. For example, Chisel HCL was used as the register-transfer level (RTL) generator to tape-out RISC-V based SoC [3]. And one of the research projects introduced the concept and implementation of a hardware compiler framework that used an open-source hardware intermediate hardware representation named flexible intermediate representation for RTL (FIRRTL) to transform target-independent RTL into technology specific RTL [2].

Further, the comparative study on designing with Chisel HCL against Verilog HDL was carried out in [6]. A N-bit fixed priority arbiter was designed in Chisel HCL and Verilog to compare the timing, power, and area of the designed module. The Chisel implementation required 250 lines of Chisel source code whereas the Verilog implementation required 400 lines. This article also showed that the Chisel implemented design used less hardware resources compared to the Verilog design.

Therefore, this paper presents the binary arithmetic designs using the abstractness and scalability of Chisel HCL. The Chisel based design library demonstrates the ease of programming hardware circuits as compared to Verilog/VHDL and proves that generated Verilog has similar slice count, maximum clock frequency, and on-chip dynamic power consumption.

III. Proposed Work

This section presents a configurable and reusable binary design library that is developed with Chisel HCL. The IntelliJ software from JetBrains is employed as the developing environment with Scala plugins and Chisel HCL libraries. As an example, Fig. 1 shows the design flow using the proposed Chisel library, from the Chisel design and verification, then the generated Verilog HDL. After the HDL generation, the traditional FPGA design procedure including the synthesis, implementation, and evaluation can be conducted.

Specifically, Fig. 1(a) shows the example of a Chisel HCL design on a full adder. In what follows, Fig. 1(b) shows the simulation result of the Chisel HCL design after running the test script file by passing the input data. This result proves the functionality of the HCL design.

After declaring the Chisel design module, the Chisel compiler is called to translate the HCL design module into Verilog HDL. This elaboration process requires passing bit width as a parameter to generate a synthesizable Verilog for the design module. The generated Verilog HDL is written into a new Verilog file which is added into a Vivado project as shown in Fig. 1(c).

In Vivado, the FPGA design flow can be carried out including synthesis and implementation. The synthesis circuit can be shown in Fig. 1(d). Finally, the results of FPGA slice count and power analysis after the implementation is successfully completed as shown in Fig. 1(e).

IV. Implementation of Configurable Module Design

This section presents the implementations of six fundamental designs of binary arithmetic circuits, including full-adder, full-subtractor, multiplier, shifter, 2's complement operator, and leading bit detector modules. The proposed work is scalable to other arithmetic modules implementation. The validity of the proposed work can be extended to many other parameterizable circuit and system generations.

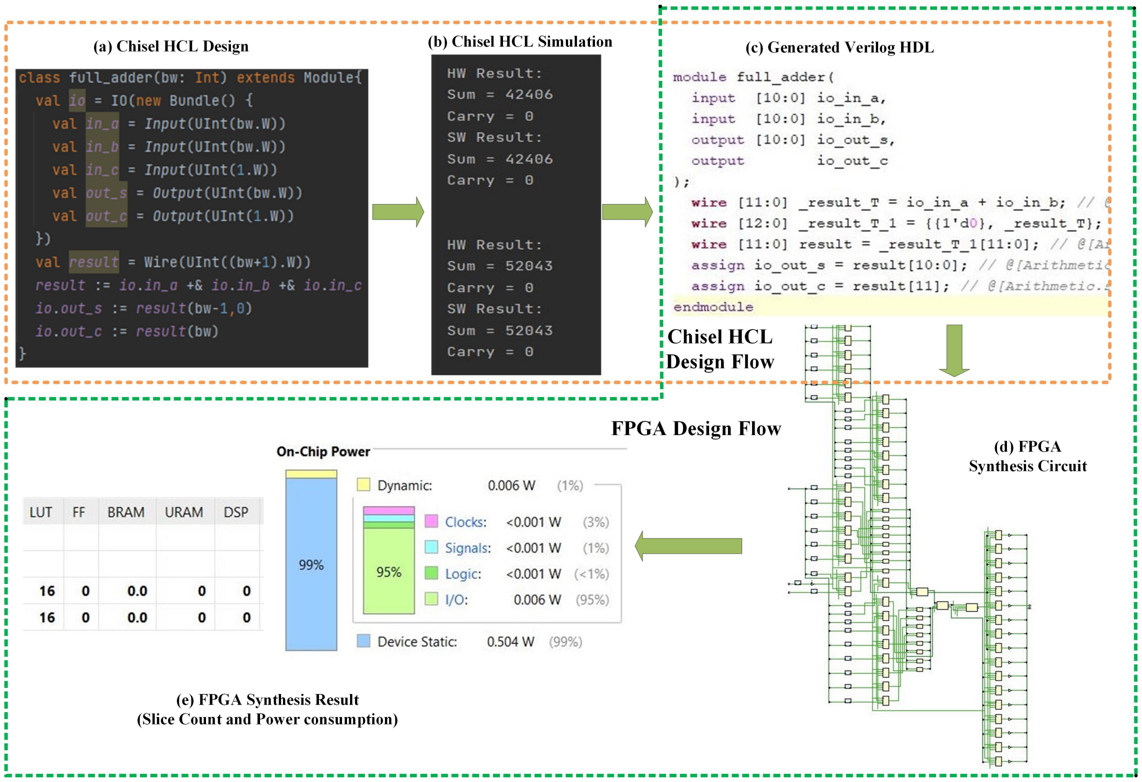
A. Full-adder module design

Full-adder is a basic arithmetic module used in mathematical computation. This module is built on the logic of binary addition. Based on the requirement, the bit width for input numbers and the output sum is selected and the Verilog code is generated. It takes a single clock cycle to compute output sum and carry for the selected bit width. The maximum clock frequency that can be achieved for a 16-bit module is 604.96 MHz and the total on-chip power is 510 mW.

Notice that the evaluation results are based on the specific FPGA platform and configuration. Different clock frequency and power dissipation could be achieved with other selected FPGA parts. More details will be discussed in the next section.

B. Full-subtractor module design

Similar to the full-adder module, the full subtractor is developed on the logic of binary subtraction. This module takes two numbers as input and computes the borrow out and


(a) Chisel HCL Design
(b) Chisel HCL Simulation
(c) Generated Verilog HDL
Chisel HCL Design Flow
FPGA Design Flow
(d) FPGA Synthesis Circuit
On-Chip Power
99%
95%
(e) FPGA Synthesis Result
(Slice Count and Power consumption)

```
class full_adder(bw: Int) extends Module{
  val io = IO(new Bundle() {
    val in_a = Input(UInt(bw.W))
    val in_b = Input(UInt(bw.W))
    val in_c = Input(UInt(1.W))
    val out_s = Output(UInt(bw.W))
    val out_c = Output(UInt(1.W))
  })
  val result = Wire(UInt((bw+1).W))
  result := io.in_a +& io.in_b +& io.in_c
  io.out_s := result(bw-1,0)
  io.out_c := result(bw)
}
```

```
HW Result:
Sum = 42406
Carry = 0
SW Result:
Sum = 42406
Carry = 0


HW Result:
Sum = 52043
Carry = 0
SW Result:
Sum = 52043
Carry = 0
```

```
module full_adder(
  input  [10:0] io_in_a,
  input  [10:0] io_in_b,
  output [10:0] io_out_s,
  output        io_out_c
);
  wire [11:0] _result_T = io_in_a + io_in_b; // @
  wire [12:0] _result_T_1 = {{1'd0}, _result_T};
  wire [11:0] result = _result_T_1[11:0]; // @[Ar
  assign io_out_s = result[10:0]; // @[Arithmetic
  assign io_out_c = result[11]; // @[Arithmetic.s
endmodule
```

LUT	FF	BRAM	URAM	DSP
16	0	0.0	0	0
16	0	0.0	0	0

Dynamic:	0.006 W	(1%)
Clocks:	<0.001 W	(3%)
Signals:	<0.001 W	(1%)
Logic:	<0.001 W	(<1%)
I/O:	0.006 W	(95%)
Device Static:	0.504 W	(99%)

Fig. 1. A Design Example between Chisel HCL to Verilog HDL Design Flow.

difference as the output. This module takes a single clock cycle to compute the output. The bit width for the module can be selected based on the requirement. The maximum clock frequency that can be obtained from the 16-bit module is 868.8 MHz and the total on-chip power is 510 mW.

C. Multiplier module design

Similar to the multiplication of two decimal numbers, the binary multiplier follows the same method for computing a product result of the two binary numbers. The bit width for the product result is double the size of the input numbers. The product of two binary numbers is computed in a single clock cycle. The advantage of using a fixed-point multiplier is that it can be built using look-up-tables, saving DSP resources on the FPGA. The maximum clock frequency that can be obtained from the 16-bit module is 148.65 MHz and the total on-chip power is 514 mW.

D. Shifter module design

The shifter module shifts the input number by a specified number of bit positions to the right or left. The input to this module is the number to be shifted, the number of bit positions, and, the shift left or right. The output bit width is considered to have the same bit width of the input number that is being shifted. The maximum clock frequency that can be obtained from the 16-bit module is 393.54 MHz and the total on-chip power is 508 mW.

E. Two's complement module design

Two's complement of a number is used to store the negative value of a number. This module involves functions such as bit flipping and bitwise addition. The maximum clock frequency that can be obtained from a 16-bit module is 708.71 MHz and the total on-chip power is 510 mW.

F. Leading one detector module design

The leading one bit detector is designed to return the bit position of the most significant bit in the number. It is designed to be used in a floating-point arithmetic operation. The specified bit widths are 11, 24, 53, and 113. The maximum clock frequency that can be obtained from the 11-bit module is 896.86 MHz and the total on-chip power is 505 mW.

V. Experimental Results

In this section, the hardware cost and speed is estimated using AMD-Xilinx Vivado 2019.2 with the target device Kintex UltraScale xcku035-ffva1156-3-e.

A. Accuracy

Unlike conventional HDL descriptions such as Verilog and VHDL designs, Chisel is used to generate synthesizable Verilog using pre-built libraries. Since the implementation does not assume any approximation in computing the result, the module computes the result with 100% accuracy in a single clock cycle.

TABLE I
MAXIMUM OPERATIONAL FREQUENCY (MHZ)

Designs	**16/11-bit Design**	**32/24-bit Design**	**64/53-bit Design**	**128/113-bit Design**
Full-adder	604.96	586.85	390.93	376.64
Full-subtractor	868.8	513.87	460.19	374.39
Multiplier	148.65	135.35	90.37	87.58
Shifter	393.54	342.81	240.21	239.06
Two's compliment	708.71	900	564.65	546.14
MSB Detector	896.86	693.96	401.28	262.53

B. RTL Analysis Results

All the design modules with different bit width can be synthesized into RTL analysis results. As examples, Fig. 2(a)-(e) show the schematics of the 16-bit full adder, subtractor, multiplier, shifter, and two's complement implementations. And in Fig. 2(f), it shows the result of the most significant bit (MSB) detector with the 11-bit design.

For all the hardware results from Fig. 2, it can be observed that both the input and output signals are from the output of registers. Between the registers, the look-up-tables are insert to implement the functionality of the designs. In such a way the critical path between the two-stage registers can be evaluated, which is used to estimate the maximum operational frequency (MOF).

C. Speed Estimation

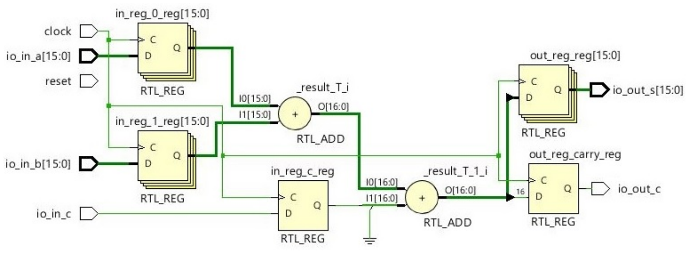
To estimate the MOF, the worst negative slack (WNS) is needed in the implementation report. WNS is basically the minimum timing slacks of all timing endpoints, which can be represented as $WNS = \min(slack(\tau))$, where τ is the set of the timing endpoints. The longest path delay determines the maximum frequency at which the design can operate. The MOF thus can be roughly estimated using the equation $MOF = \frac{1}{T-WNS}$, where the difference between the clock period (denoted as T) and the WNS shows the critical path of the combinational circuit between the registers [23].

Using the design approach, Table. I summarizes the MOF for all the design modules generated by the design library. The results show that the lower complexity the faster the operating clock speed achieved. Specifically, the highest speed of the full-adder implementation is 604.96 MHz which is achieved by the 16-bit design. And the 128-bit design on the full-adder obtains the MOF of 376.64 MHz which is the lowest clock frequency compared with the other implementations of the full-adder.

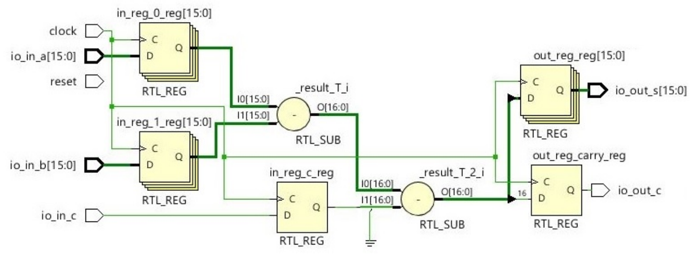
Notice that an unexpected result occurs in the 2's complement designs in the sixth row, where the 32-bit design obtains higher MOF than the 16-bit implementation. Though the 16-bit design spends more IOs and slice count, the critical path could be higher than or similar to that of the 32-bit design, depending on the placement & route of the synthesis results.

D. Area Estimation

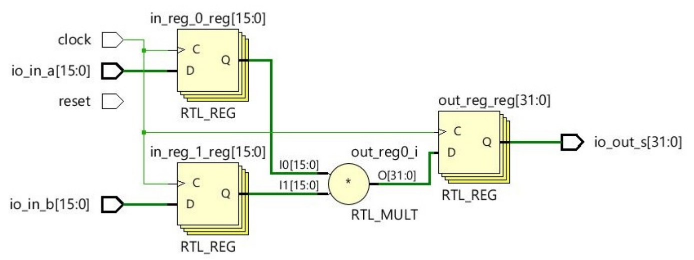
As the practical result summarizes in Table II, it can be observed that the hardware resource usage is increased with



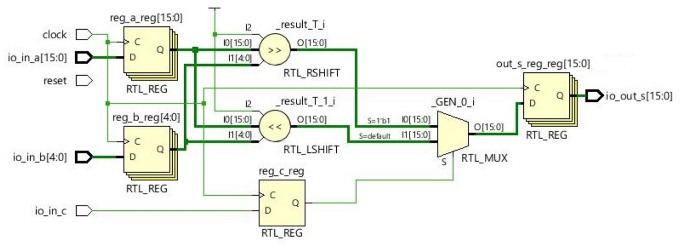
(a) 16-bit full-adder synthesis result



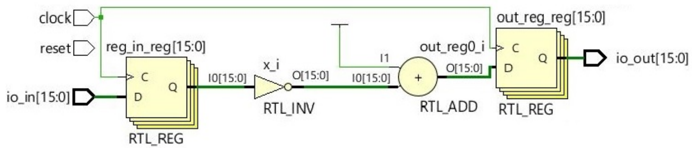
(b) 16-bit full-subtractor synthesis result



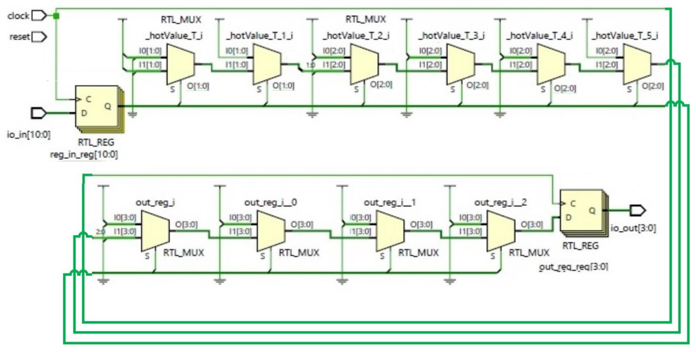
(c) 16-bit multiplier synthesis result



(d) 16-bit shifter synthesis result



(e) 16-bit two's complement synthesis result



(f) 11-bit Leading One Detector

Fig. 2: 16bit Synthesis Results.

TABLE II
LUT COUNT OF DIFFERENT DESIGNS

Designs	16/11-bit Design	32/24-bit Design	64/53-bit Design	128/113-bit Design
Full-adder	16	32	64	128
Full-subtractor	17	33	65	129
Multiplier	279	1,208	5,020	21,175
Shifter	66	158	397	953
Two's compliment	11	31	63	127
MSB Detector	6	20	55	129

TABLE III
DYNAMIC POWER CONSUMPTION OF DIFFERENT DESIGNS (MW)

Designs	16/11-bit Design	32/24-bit Design	64/53-bit Design	128/113-bit Design
Full-adder	6	8	35	49
Full-subtractor	6	8	15	70
Multiplier	10	22	90	180
Shifter	4	8	12	49
Two's compliment	3	6	14	37
MSB Detector	1	3	6	6

increase in bit width and complexity. For example, the 16-bit full-adder takes 16 LUTs and the 128-bit design spends 128 LUTs. Notice that the binary multiplier spends a large amount of LUTs compared with the other binary designs, showing a potential to improve the hardware efficiency to the approximate designs on multipliers.

E. Power Consumption

After the implementation, the power analysis is conducted as the static and dynamic on-chip power consumption on the FPGA board. Table III summarizes the dynamic on-chip power usage while static power remains constant around 504 mW. As the number of slice counts or hardware resources are more, the FPGA draws more power during operation. The switching activity can also lead to higher usage of an FPGA board.

VI. CONCLUSION

In this paper, Chisel HCL is employed to design the parameterizable binary modules including full-adder, full-subtractor, multiplier, shifter, leading one detector, and two's complement. The implementations of the proposed work proves that Chisel is a potential hardware circuit generator that can be used to design a circuit with more complex arithmetic implementation. Since Chisel HCL inherits object-oriented programming and functional programming concepts from Scala, it reduces the complexity of RTL designs on particular arithmetic circuits. The availability to expand on the Chisel Application Program Interface (API) using the constructs and syntax of Scala embedded makes Chisel the perfect choice for performing complex and scalable hardware design.

VII. FUTURE WORKS

Parameterizablity is one of the main advantages and motivations of using the Chisel based design library. In this paper, the binary designs on the fundamental arithmetic modules are conducted to demonstrate the validity of the research work. The future works will be focusing on the complicated designs to extend the design library such as the floating-point operators and network neurons.

REFERENCES

[1] J. Bachrach et al., "Chisel: Constructing hardware in a Scala embedded language," IEEE/ACM Design Automation Conference, PP. 1212-1221, 2012. doi: 10.1145/2228360.2228584.

[2] A. Izraelevitz et al., "Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations," 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2017), PP. 209-216, 2017. doi: 10.1109/ICCAD.2017.8203780.

[3] Khan, Muhammad Hadir, et. al., "IBTIDA: Fully open-source ASIC implementation of Chisel-generated System on a Chip," TechRxiv, 2021, doi: 10.36227/techrxiv.16663738.v1

[4] M. L. Petrovic and V. M. Milovanovic, "A Chisel Generator of Parameterizable and Runtime Reconfigurable Linear Insertion Streaming Sorters," 2021 IEEE 32nd International Conference on Microelectronics (MIEL), PP. 251-254, 2021. doi: 10.1109/MIEL52794.2021.9569153.

[5] D. Zoni, A. Galimberti and W. Fornaciari, "Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials," in IEEE Access, Vol. 8, PP. 75809-75821, 2020. doi: 10.1109/ACCESS.2020.2989423.

[6] P. Lennon and R. Gahan, "A Comparative Study of Chisel for FPGA Design," 2018 29th Irish Signals and Systems Conference (ISSC 2018), PP. 1-6, 2018. doi: 10.1109/ISSC.2018.8585292.

[7] J. Bachrach, et al., "Chisel: Constructing hardware in a Scala embedded language", 49th ACM/IEEE Design Automation Conference (DAC 2012), PP. 1212-1221, Jun 2012.

[8] F. Heilmann et al., "Investigate the high-level HDL Chisel" in Germany:Kaiserslautern, 2013.

[9] V. M. Milovanovic and M. L. Petrovic, "A Highly Parametrizable Chisel HCL Generator of Single-Path Delay Feedback FFT Processors," 2019 IEEE 31st International Conference on Microelectronics (MIEL 2019), PP. 247-250, 2019. doi: 10.1109/MIEL.2019.8889581.

[10] "Chisel/FIRRTL Hardware Compiler Framework", 2019. [Online]. Available: https://www.chisel-lang.org/

[11] "Chisel Github", 2019. [Online]. Available: https://github.com/chipsalliance/chisel3

[12] X. Chen, et al. "ThunderGP: Resource-Efficient Graph Processing Framework on FPGAs with HLS," ACM Transactions on Reconfigurable Technology and Systems (TRETS), In press, 2022.

[13] A. L. Reed and X. Yang, "Lightweight Neural Network Architectures for Resource-Limited Devices," 23rd IEEE Intl. Symposium on Quality Electronic Design (ISQED 2022), Accepted, 2022.

[14] I. Westby, X. Yang, T. Liu, and H. Xu, "Exploring FPGA Acceleration on a Multi-Layer Perceptron Neural Network for Digit Recognition," The Journal of Supercomputing, PP. 1-18, May 13, 2021. DOI: 10.1007/S11227-021-03849-7

[15] X. Yang, Y. Zhang, and L. Wu, "A Scalable Image/Video Processing Platform with Open Source Design and Verification Environment," 20th Intl. Symposium on Quality Electronic Design (ISQED 2019) , PP. 110-116, Santa Clara, CA, USA, 2019.

[16] "Intel-Altera Quartus". [Online]. Available: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/download.html

[17] "AMD-Xilinx Vivado". [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[18] "Synopsys VCS Simulator". [Online]. Available: https://www.synopsys.com/verification/simulation/vcs.html

[19] "Synopsys Design Compiler". [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html

[20] "Cadence Software". [Online]. Available: https://www.cadence.com/en_US/home/support/software-downloads.html

[21] "Cadence Genus Synthesis Solution". [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html

[22] "Siemens-Mentor Graphic Modelsim". [Online]. Available: https://eda.sw.siemens.com/en-US/ic/modelsim/

[23] "Vivado Timing - Where can I find the Fmax in the timing report?". [Online]. Available: https://support.xilinx.com/s/article/57304