A Trigonometric Hardware Acceleration in 32-bit RISC-V Microcontroller with Custom Instruction

Khai-Duy Nguyen\textsuperscript{1a),} Dang Tuan Kiet\textsuperscript{1b),} Trong-Thuc Hoang\textsuperscript{1c),} Nguyen Quang Nhu Quynh\textsuperscript{2d),} Xuan-Tu Tran\textsuperscript{2c),} and Cong-Kha Pham\textsuperscript{1f)}

Abstract This work presents a 32-bit Reduced Instruction Set Computer fifth-generation (RISC-V) microprocessor with a COordinate Rotation Digital Computer (CORDIC) accelerator. The accelerator is implemented inside the core and being used by the software via custom instruction. The used microprocessor is the VexRiscv with the Instruction Set Architecture (ISA) of RV32IM, that means 32-bit RISC-V including Integer and Multiplication. The experimental results were collected using Field-Programmable Gate Array (FPGA) on the DE2-115 development kit and Application Specific Integrated Chip (ASIC) synthesizer on 180-nm CMOS process library.

key words: 32-bit microprocessor, accelerator, CORDIC, custom instruction, RISC-V, trigonometric.

Classification: Integrated circuits (logic)

1. Introduction

Reduced Instruction Set Computer fifth-generation (RISC-V) [1] is a free and open Instruction Set Architecture (ISA) developed at the University of California at Berkeley. It possesses a stand-out characteristic that makes it attractive to the open-source communities in both academy and industry areas. Many RISC-V-based processors have been presented recently. Some worth-mentioned works are the highly customizable Rocket-chip coreplex [2], the 32-bit E-core series [3], the 64-bit U-core series [4], the minimal RISC-V V-core for embedded systems [5], and the 32-bit single-cycle RISC-V [6]. Due to its versatility and customizability, the RISC-V ISA has become a suitable target for developing a highly customizable computer system.

Nowadays, computational tasks have become far more complex than the way general-purpose computers can serve them. Thus, processor efficiency requirements are becoming increasingly critical. Accelerators are extensively used for many intensive computational tasks, reducing execution time and energy consumption. Different companies and research groups are developing accelerators in RISC-V for various applications such as digital signal processing [7], artificial intelligence [8, 9], and solving mathematical algorithms [10, 11].

Among heavy computational tasks, the calculation of trigonometric functions is widely used, especially in digital signal processing algorithms of wireless and communication systems such as WiMAX [12], 3GPP-LTE [13], MIMO [14], CDMA [15], OFDM [16] and WLAN [17]. However, the complexity of trigonometric algorithms makes it a problem when computing in the digital realm. If a general-purpose processor executes the algorithms, it will break down the algorithms into multiple simple calculations, reducing its efficiency. To cope with this problem, an accelerator specialized in trigonometry computation is in need.

COordinate Rotation Digital Computer (CORDIC) [18] is a popular algorithm used for trigonometry computation [19–22]. It is a simple and effective algorithm for hyperbolic and trigonometric functions, usually converged with one digit (or bit) per iteration. A CORDIC only uses adders and shifters to calculate the result, with the benefit that it could be done with relatively basic hardware. Besides, CORDIC can calculate multiple functions with the same hardware, so they are ideal for many applications. We aim to develop a trigonometric accelerator applying the CORDIC algorithm to increase efficiency in calculating trigonometric functions and researching the RISC-V processor’s customizability.

In this paper, we designed and implemented a 32-bit RISC-V microprocessor with a CORDIC algorithm accelerator. The implemented core processor is the VexRiscv Central Processing Unit (CPU), a base implementation and multi-division in 32-bit registers (RV32IM) of the RISC-V ISA processor. Within the VexRiscv core, the CORDIC accelerator was connected directly to the Execute stage. The core was placed in Brie System-on-Chip (SoC) and was synthesized on Field Programmable Gate Array (FPGA) and...
Application Specific Integrated Chip (ASIC) level with the cell logic of 180-nm CMOS technology.

The remainder of this paper is organized as follows. Section 2 provides background information for the project. Section 3 describes the architecture of the proposed SoC chip. The implemented results are presented in detail in Section 4. Finally, Section 5 concludes the paper.

2. Background Research

2.1 CORDIC Algorithm

The CORDIC algorithm calculates the trigonometric functions by performing two-dimensional vector rotation in circular coordinate systems. For example, equation 1 represents the iteration equations of the CORDIC radix-2 algorithm in rotation mode of the circular coordinate system:

\[\begin{align*}
x_{i+1} &= x_i - d_i y_i 2^{-i} \\
y_{i+1} &= y_i + d_i x_i 2^{-i} \\
z_{i+1} &= z_i - d_i \alpha_i
\end{align*}\]  

The value of \(d_i\) is expressed in equation 2. The values of \(\alpha_i\) are chosen so that \(\tan(\alpha_i) = 2^{-i}\) and the multiplication of tangential terms are reduced to a simple shift operation. Therefore, we have the value of \(\alpha_i = \tan^{-1}(2^{-i})\).

\[d_i = \begin{cases} 
-1 & \text{if } z_i < 0 \\
1 & \text{otherwise}
\end{cases} \]  

2.2 RISC-V Instruction Set Architecture

RISC-V ISA allows the creation of open-source processors, and, in relation to them, many open-source resources have been developed, such as compiler, debuggers, hardware implementations in different hardware description languages. Thanks to the community’s support, suitable Integrated Development Environment (IDE) and Operating Systems (OS) for many variations of processors are available. Originally, RISC-V ISA was proposed to be a simple specification of a processor. It means that the base integer ISA is an adequate minimal set of instructions [23]. Moreover, RISC-V has been designed to support extensive customization. Therefore, optional instruction-set extensions can be customized into the base integer ISA. The base integer ISA is named “I” prefixed by RV32 and RV64, which provide 32-bit and 64-bit address spaces, respectively. The 64-bit version RV64I is suitable for large and sophisticated systems; on the other hand, 32-bit address spaces of RV32I are adequate for embedded and Internet of Thing applications [24–26].

2.3 VexRiscv: 32bit Microprocessor

VexRiscv [27] is an RV32IM variant of RISC-V ISA processor. VexRiscv is written in a new hardware construction language called SpinalHDL [28], a language based on the Scala programming language [29]. VexRiscv has a modular design, with most of the components of the processor are optional. The extension of VexRiscv includes multiplication and division, instruction and data caches, memory management unit, hazard controller, etc. The advantage of the high-level SpinalHDL maximizes the customization ability to the VexRiscv, which makes VexRiscv the ideal platform for developing hardware accelerators. Briey System-on-Chip, an implementation consists of the VexRiscv core and peripherals, was used as the primary subject in this study.

3. Proposed Implementation

3.1 System Overview

Fig. 2 shows the architecture of the Briey SoC, which includes the VexRiscv core and its peripherals. The core is composed of a 32-bit RV32IM RISC-V CPU, a 4 Kilo-Bytes (KB) instruction cache ($I$), a 4KB data cache ($D$), an Arithmetic Logic Unit (ALU), and a full barrel shifter. Also, static memory translator, branching, debug module
via Joint Test Action Group (JTAG), and CORDIC module is included. The Briey SoC uses an Advanced eXtensible Interface (AXI) [32] Crossbar to connect the RISC-V core with a Synchronous Dynamic Random Access Memory (SDRAM) Controller, Advanced Peripheral Bus 3 (ABP3) Bridge, and a 16-KB on-chip Random Access Memory (RAM). The SoC also includes peripherals such as a General Purpose Input/Output (GPIO), a timer, and a Universal Asynchronous Receiver/Transmitter (UART) controller. These peripherals are all connected to the APB3 Bridge.

3.2 CORDIC Accelerator

Fig. 3 presents the structure of the CORDIC circuit, which serves as the central unit of computation in the accelerator. This design uses the bit-parallel, unrolled structure, which only uses adders, subtractors, and shifters. Each iteration costs three adders/subtractors with two hardwired shift operations. Rotate angles \((\alpha_0, \alpha_1, etc.)\) are also fit for each iteration. With this combination circuit, depending on the design purpose, \(x_0, y_0, z_0\) inputs can be changed to apply different modes, creating design flexibility. Our design focuses on trigonometry; thus, the circular rotation mode is applied. Therefore, the general CORDIC algorithm is directly transformed into equation 3:

\[
x_n = K[x_0 \cos(z_0) - y_0 \sin(z_0)]
\]

\[
y_n = K[y_0 \cos(z_0) + x_0 \sin(z_0)]
\]

\[
z_n = 0
\]

In this mode, commonly used input values are \(x_0 = 1/K, y_0 = 0,\) and \(z_0 = angle\) with the scaling factor \(K \approx 1.646\).

From there, the values of cosine and sine can be gathered at the output ports \(x_n\) and \(y_n\), respectively. Such input values pose a problem in that the value of the outputs will be a real number between -1 and 1. These real numbers cannot be saved directly to the Register File because we are implementing an RV32IM system that only allows integers. To overcome this difficulty, instead of using the usual \(x_0\) value, we use the value \(x_0 = 1024/K\). The output will be stored in a 32-bit register using a 22.10 fixed-point number; 22-bit integer with 10-bit decimal. The value of a sine or cosine is a product of a number multiplied by 1024 and it can be saved directly to the Registry File.

Taking advantage of the VexRiscv modular architecture, a CORDIC module is added as a separate plugin to the system. This module is used through two custom instructions, which use the free RISC-V ISA opcodes. The entire CORDIC module, including the custom instructions, is written in SpinalHDL. Fig. 4 shows the fields used for the two custom instructions, sine and cosine, and their values. The R-type format is used, but only with the opcode and funct3 fields. Because of the simple requirement of the custom instructions, \(rs2\) and \(funct7\) are left blank. The input angle values are entered through \(rs1\) while the trigonometric results are saved to \(rd\).

CORDIC accelerator is added to the Execute stage for computation. When the custom instruction is decoded in the Decode stage, the input angle will be delivered to the CORDIC accelerator instead of the normal Execute unit. The initial angle in the degree unit is represented as an integer. Thus, it will be processed before being fed into the CORDIC circuit. First of all, the angle will be multiplied by \(2^{32}/360\) using Canonical Signed Digit (CSD) algorithm. The use of multiplication serves two purposes; first, it helps keep input angles in the range of 0 and 359 degrees using 32 bits. Values outside this range will lead to overflow when represented in 32 bits, thereby keeping simplicity and ensuring accuracy for handling the angle.

Second, the two most significant bits can be used to determine the quadrant of the angle. Because of the convergence limitation, the CORDIC algorithm can only be used in the 1st and 4th quadrants. Therefore, the angle will be passed through a quadrant mapper to extend the computation to the remaining two quadrants. The quadrant mapper determines the quadrant of the input angle to change the sign of \(x_0, y_0,\) and \(z_0\), thereby ensuring that the CORDIC accelerator can handle angles in the range from \((-2^{31})\) to \((2^{31} - 1)\). After completing these processing steps, the angle is fed into the CORDIC circuit. The trigonometric value from the CORDIC circuit will be passed to the next stage of the pipeline stored in the Register File.
Figure 5 shows the flow of custom instruction in the pipeline of the VexRiscv processor. The horizontal axis represents time in clock cycle units, while the vertical axis represents the instruction order. Since VexRiscv’s pipeline architecture includes five stages (i.e., Fetch, Decode, Execute, Memory, and WriteBack), we assume that a basic instruction will be processed in one clock cycle for each stage as shown with the first and fourth instructions. The second and third instructions represent the pipeline diagram of the two custom instructions sine and cosine in the instruction execution order. Each custom instruction requires eight clock cycles to complete, as they require four clock cycles on the Execute stage. Thus, each custom instruction used increases the latency of the pipeline by three clock cycles.

4. Experimental Result

This section shows the result of the VexRiscv implementation in FPGA and ASIC. First, the implementation results are obtained on the Altera DE2-115 FPGA Development Kit with the chip of EP4CE115F29C. Next, the speed test results of the CORDIC algorithm on hardware and software are compared. We also compare the hardware implementation with two different works. Last, we will present the synthesized ASIC result on CMOS 180-nm technology, including maximum frequency, area, and power consumption.

Table I. Hardware utilization summary in FPGA implementation.

| Component       | LUTs  | FFs   | BRAM     |
|-----------------|-------|-------|----------|
| Briey SoC       | 5,377 | 3,271 | 206,592  |
| VexRiscv Core   | 3,999 | 1,639 | 75,264   |
| Data Cache      | 168   | 150   | 35,584   |
| Instruction Cache | 37   | 97    | 33,584   |
| CORDIC module   | 800   | 186   | 0        |
| JTAG            | 106   | 138   | 0        |
| On-chip RAM     | 55    | 27    | 131,072  |
| SDRAM Controller| 384   | 308   | 0        |
| GPIO            | 15    | 144   | 0        |
| UART            | 184   | 110   | 256      |
| Timer           | 251   | 222   | 0        |

4.1 FPGA Implementation

Table I shows the FPGA resource utilization. The maximum frequency achieved 54.67 Mega-Hertz (MHz). The whole system used a total of 5,377 Look-Up Tables (LUTs) and 3,271 Flip-Flops (FFs), consuming 5% and 3% utilization of the FPGA, respectively. The part that uses the most resources is the VexRiscv core with 3,699 LUTs and 1,639 FFs, respectively. There are 206,592 Block Random Access Memory (BRAM) blocks already in use, accounting for 5% of the total BRAM blocks on DE2-115. The component that used the most BRAM blocks was 16-KB on-chip RAM with 131,072 BRAM blocks. The SoC without CORDIC accelerator included 4,297 LUTs and 3,017 FFs. Therefore, attaching the CORDIC accelerator to the VexRiscv increased system overhead by 25.13% and 8.41% for LUTs and FFs, respectively. The BRAM number is not changed because the CORDIC module does not use memory.

Table II shows the comparison in speed of various implementations. In terms of software, a test program written in C calculates trigonometric values of an angle. On the hardware side, we use the inline assembly to call the CORDIC custom instruction. Both test programs use ten iterations and are compiled using the standard GNU Compiler Collection (GCC) RISC-V toolchain. The results in the table show the number of clock cycles, which are held by the mcycle register, used for the calculation for each angle. When the CORDIC algorithm is implemented in software, the program will be divided into multiple addition, subtraction, shift, and memory access instructions. Each instruction takes several clock cycles to execute, resulting in the CORDIC algorithms taking 3,314 clock cycles to compute a trigonometric operation. In contrast, the CORDIC accelerator computes trigonometric functions directly. By eliminating a large amount of instruction, the processing speed of trigonometric functions is significantly increased. It can be seen that, with similar calculation results, using the hardware perform at least 414 times faster than using the software. In turn, the implementation of the CORDIC accelerator also reduces the maximum frequency by about 25%.
Table II. Comparison of the speeds of different implementations.

| Implementation       | FPGA                | Simulation          | FPGA                |
|----------------------|---------------------|---------------------|---------------------|
| Approach             | Accelerator         | Channel Accelerator | Accelerator         |
| Speed (clock cycles) | 8                   | 35                  | 24                  |
| Maximum Frequency(MHz)| 54.07               | N/A                 | 50                  |
| Processor            | VexRiscv            | RocketChip          | Nios II             |
| CORDIC Structure     | Bit-parallel unrolled| N/A                 | Bit-parallel iterative| N/A |

Table III. Utilization summary in CMOS 180-nm implementation.

| Area                     | Gate (NAND2) |
|--------------------------|--------------|
| Briey SoC                | 2,981,434    |
| VexRiscv Core            | 1,769,722    |
| Data Cache               | 585,506      |
| Instruction Cache        | 579,081      |
| CORDIC module            | 92,120       |
| JTAG                     | 11,754       |
| On-chip RAM              | 987,507      |
| SDRAM Controller         | 28,376       |
| GPIO                     | 27,624       |
| UART                     | 40,646       |
| Timer                    | 30,992       |

In [30], they implemented the CORDIC algorithm for the RockerChip in two forms. In the first form, CORDIC was linked as a coprocessor outside of the SoC via RoCC interface, which cost 24 clock cycles per trigonometric calculation. The other was implementing CORDIC directly inside the SoC as a channel accelerator and was linked to the CPU via the inner system bus, which takes 35 clock cycles per operation. It should be noticed that the CORDIC algorithm design of [30] is the bit-parallel, iterative structure with only one iteration unit. The designer used a loop to reuse the iteration unit, making this design takes many clock cycles. Therefore, our designs are approximately 3 and 4.4 times more efficient than the two compared designs in speed. Also, the implementation in [31] is compared. Altera Nios II-based platform was used to implement the CORDIC algorithm using custom instructions. At the same frequency, the speed of our system proved outstanding with about 45 times faster.

4.2 ASIC Implementation

Table III shows the synthesized result of the Briey SoC for 180-nm CMOS technology. The system components used are similar to the FPGA implementation outlined above. The ASIC implementation area is about 3 millimeters square (mm²), and the gate count value is about 308k logical gates. The VexRiscv core occupies the majority of the area with 59.35%, followed by on-chip RAM with 33.12%. Fig. 6 shows the final implemented result of the Briey SoC, which was placed and routed in 180-nm CMOS technology. The system can operate at a maximum frequency of 107-MHz and power consumption of 649-mW at 100-MHz.

5. Conclusion

The 32bit RISC-V microcontroller based on RV32IM VexRiscv CPU with a trigonometric functions accelerator using CORDIC algorithms was presented in this paper. The complete System-on-Chip was built and tested on DE2-115 FPGA Development Kit, and then synthesized at ASIC level with the CMOS 180-nm technology. With the implementation of FPGA, the proposed architecture used a total of...
5,377 LUTs and 3,271 FFs. Most resources are allocated for VexRiscv core with 3,999 LUTs and 1,639 FFs. The total number of BRAMs in use is 206,592 with 131,072 for 16-KB on-chip RAM. The maximum frequency the system can reach on FPGA is 54.67-MHz. The area of ASIC implementation after synthesized with 180-nm CMOS technology is about 3-mm². The total gate count is approximately 308K, in which, VexRiscv and on-chip RAM accounted for 59.35% and 33.12% of the whole chip, respectively. The maximum frequency of operation is 107-MHz, and the power consumption is 649-mW at 100-MHz.

Acknowledgement

This work was supported through the activities of VDEC, The University of Tokyo, in collaboration with Cadence Design Systems and Mentor Graphics.

References

[1] A. Waterman, et al.: “The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0,” UC Berkeley UCB/EECS-2014-54 (2014) https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.pdf.

[2] K. Asanovic, et al.: “The Rocket Chip Generator,” UC Berkeley UCB/EECS-2016-17 (2014) https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.pdf.

[3] SiFive, Inc.: “SiFive E31 Manual,” (2019) https://sifive.cn/design/sifive/sifive%2Fc89f6e5a-cf9e-44c3-a3db-04420702d2c1_sifive_e31_manual_v1v9.08.pdf.

[4] SiFive, Inc.: “SiFive E54 Manual,” (2019) https://sifive.cn/design/sifive/sifive%2F66e95f812-e9cd-4553-bf6f-c7e71508555_sifive_coreip_U54MC_AXI4_rtl_v1v9.08p2p0_release_manual.pdf.

[5] M. Johns, et al.: “A Minimal RISC-V Vector Processor for Embedded Systems,” Forum for Spec. and Design Lang. (2020) 20085586 (DOI: 10.1109/FDLS.2018.9023940).

[6] D. Dennis, et al.: “Single cycle RISC-V micro architecture processor and its FPGA prototype,” ISED (2017) 17613890 (DOI: 10.1109/ISED.2017.8303926).

[7] L. Calicchia, et al.: “Digital Signal Processing Accelerator for RISC-V,” IICECS (2019) (DOI: 10.1109/IICECS.2019.8964670).

[8] G. Zhang, et al.: “A RISC-V based hardware accelerator designed for Yolo object detection system,” ICIASE (2019) 19556203 (DOI: 10.1109/ICIASE45644.2019.9074051).

[9] R. Portier, et al.: “Extending a Soft-Core RISC-V Processor to Accelerate CNN Inference,” “CSCI (2019) 19553575 (DOI: 10.1109/CSCI.2019.8393700).

[10] J. Wu, et al.: “eRSA: An energy-efficient Reconfigurable Sorting Accelerator,” IETCE (2019) 19631540 (DOI: 10.1109/IETICTE74263.2019.9094818).

[11] R. Singh, et al.: “32 Bit re-configurable RISC processor design and implementation for BETA ISA with an built matrix multiplier,” ISED (2016) 17029245 (DOI: 10.1109/ISED.2016.7977065).

[12] J. G. Andrews, et al.: “Fundamentals of WiMAX: Understanding Broadband Wireless Networking,” Prentice Hall (2007).

[13] E. Dahlman: “3G Evolution: HSPA and LTE for Mobile Broadband,” Academic (2008).

[14] A. F. Molisch and X. Zhang: “FFT-Based Hybrid Antenna Selection Schemes for Spatially Correlated MIMO Channels,” IEEE Comm. Letters (2004) 7909449 (DOI: 10.1109/LCOMM.2003.822512).

[15] Y. Tang and B. Vucetic: “The FFT-based multiuser detection for DS-CDMA ultra-wideband communication systems,” IEEE Cat. No.04EX812 (2004) 8065919 (DOI: 10.1109/UBWST.2004.1320946).

[16] V. Arunachalam and A. Raj: “Efficient VLSI implementation of FFT for orthogonal frequency division multiplexing application,” IEI Circ. Devices & Syst. (2014) 10.1049/iet-cds.2013.0457.

[17] C. Lin, et al.: “A low-power 64-point FFT/IFFT design for IEEE 802.11a WLAN application,” IEEE Int. Symp. on Circ. and Syst. (2006) 9086615 (DOI: 10.1109/ISCAS.2006.1693653).

[18] P. Meher, et al.: “50 Years of CORDIC: Algorithms, Architectures, and Applications,” IEEE Trans. on Circ. and Syst. 1 (2009) 10847395 (DOI: 10.1109/TCAS.2009.2025803).

[19] J. Zhou, et al.: “Dynamic Configurable Floating-Point FFT Pipelines and Hybrid-Mode CORDIC on FPGA,” Int. Conf. on Embedded Software and Syst. (2018) 10143628 (DOI: 10.1109/ICCSS.2018.95).

[20] X. Xiao, et al.: “Reduced memory architecture for CORDIC-based FFT,” IEEE Int. Symp. on Circ. and Syst. (2010) 11462900 (DOI: 10.1109/ISCAS.2010.5537045).

[21] C. Lin and A. Wu: “Mixed-scaling-rotation CORDIC (MSR-CORDIC) algorithm and architecture for high-performance vector rotational DSP applications,” IEEE Trans. on Circ. and Syst. 1 (2005) 8650018 (DOI: 10.1109/TCAS.2005.853908).

[22] P. Rajagopalan, et al.: “ORDIC Based Phase Calibration Algorithm for Phase Coherent DSP Applications,” SPIN (2020) 19536102 (DOI: 10.1109/SPIN.2020.0070821).

[23] A. Waterman, et al.: “The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213,” ed. A Waterman and K. Asanovic (2019) https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf.

[24] H. Amor and C. Bernier: “Software-Hardware Co-Design of Multi-Standard Digital Baseband Processor for IoT,” DATE (2019) 18673521 (DOI: 10.23919/DATE.2019.8714963).

[25] H. Amor, et al.: “A RISC-V ISA Extension for Ultra-Low Power IoT Wireless Signal Processing,” IEEE Trans. on Computers (2021) (DOI: 10.1109/TC.2021.3063027).

[26] E. Tehrani, et al.: “RISC-V Extension for Lightweight Cryptography,” DSD (2020) 20035388 (DOI: 10.1109/DSD51259.2020.00045).

[27] SpinalHDL: A FPGA friendly 32 bit RISC-V CPU implementation (2020) https://github.com/SpinalHDL/VexRiscv

[28] Scala: The Scala Programming Language (2004) https://www.scala-lang.org/

[29] SpinalHDL: A FPGA friendly 32 bit RISC-V CPU implementation (2020) https://github.com/SpinalHDL/VexRiscv

[30] ARM: AMBA AXI and ACE Protocol Specification 20191213, ed. A Waterman and K. Asanovic (2019) https://www.arm.com/wp-content/uploads/2019/12/arm-axil Philip_20191213.pdf.

[31] ARM: AMBA AXI and ACE Protocol Specification 20191213, ed. A Waterman and K. Asanovic (2019) https://www.arm.com/wp-content/uploads/2019/12/arm-axil Philip_20191213.pdf.

[32] ARM: AMBA AXI and ACE Protocol Specification 20191213, ed. A Waterman and K. Asanovic (2019) https://www.arm.com/wp-content/uploads/2019/12/arm-axil Philip_20191213.pdf.