electronics

Review

Asynchronous Floating-Point Adders and Communication Protocols: A Survey

Pallavi Srivastava [1,*], Edwin Chung [1] and Stepan Ozana [2]

1 School of Computer Science and Engineering, Taylor's University, 1 Jalan Taylor's, 47500 Subang Jaya, Malaysia; chinyau.edwinchung@taylors.edu.my
2 Faculty of Electrical Engineering and Computer Science, Department of Cybernetics and Biomedical Engineering, VSB-Technical University of Ostrava, 708 00 Ostrava, Czech Republic; stepan.ozana@vsb.cz
* Correspondence: tu.pallavi@gmail.com

Received: 18 September 2020; Accepted: 9 October 2020; Published: 15 October 2020

Abstract: Addition is the key operation in digital systems, and floating-point adder (FPA) is frequently used for real number addition because floating-point representation provides a large dynamic range. Most of the existing FPA designs are synchronous and their activities are coordinated by clock signal(s). However, technology scaling has imposed several challenges like clock skew, clock distribution, etc., on synchronous design due to presence of clock signal(s). Asynchronous design is an alternate approach to eliminate these challenges imposed by the clock, as it replaces the global clock with handshaking signals and utilizes a communication protocol to indicate the completion of activities. Bundled data and dual-rail coding are the most common communication protocols used in asynchronous design. All existing asynchronous floating-point adder (AFPA) designs utilize dual-rail coding for completion detection, as it allows the circuit to acknowledge as soon as the computation is done; while bundled data and synchronous designs utilizing single-rail encoding will have to wait for the worst-case delay irrespective of the actual completion time. This paper reviews all the existing AFPA designs and examines the effects of the selected communication protocol on its performance. It also discusses the probable outcome of AFPA designed using protocols other than dual-rail coding.

Keywords: asynchronous circuit design; floating-point adder; dual-rail protocol; bundled data scheme; completion detection

1. Introduction

The computational complexity of scientific and engineering applications has increased in recent years, and it is difficult to envision a modern scientific infrastructure without numerical computing. Several scientific and engineering applications utilize floating-point representation for the computation of real numbers, as it provides a large dynamic range. The use of floating-point in computing goes all the way back to the world's first operating computing machine, the Z3, designed by Konrad Zuse, where it includes a binary floating-point number for computation [1].

Goldberg [2] demonstrates that negligence in floating-point designs can result in erroneous outcome, and hence, thorough research is required for floating-point numbers. One such example of design failure due to inaccurate calculations for floating-point representation is the Intel Pentium Processor failure [3]. Professor Thomas R. Nicely was working on the sum of the reciprocal of twin prime numbers. Being a mathematician, he developed several algorithms and evaluated those algorithms on different types of processors. In 1994, when he included a machine based on the Intel Pentium Processor, he noticed that for division algorithm, the processor might produce an incorrect floating-point result. At first, Intel denied such kind of possibility, but several other researchers reported similar challenges for different applications. Later, it was proved by engineer Tim Coe that for a few cases, floating-point computations of double-precision numbers produced an error larger than that for single-precision floating-point. The worst-case error was produced when double-precision floating point representation is used to calculate the ratio of two numbers 4,195,835 and 3,145,727. The correct result was 1.33382044 . . . and the result computed on Pentium was 1.33373906, which was accurate only up to the 14th bits, and the error produced in this case was larger than its single-precision equivalent.

In February 1991 [3], the inaccurate representation of floating-point numbers might have led to an accident that killed 28 soldiers. An American Patriot Missile battery miscalculated and failed to predict the range of an Iraqi Scud Missile. The time was expressed as an integer number, but it was measured in 1/10 by the internal clock of the system, which is a non-terminating floating-point number in binary number representation. All the computations were done in a 24-bit fixed-point register; therefore, the numbers after 24 bits were truncated, resulting in a negligible truncation error. This small truncation error computed in the Patriot battery became a significant one as the battery was utilized continuously for more than 100 h. As a result, the accumulation of multiple small numbers resulted in a larger error.

Another similar incident happened in June 1996 [3], when a crewless Ariane rocket exploded 30 s after its lift-off due to the inaccurate conversion of floating-point numbers. The project including the rocket and its cargo cost approximately $500 million, but fortunately this time no lives were lost. The computation system was unable to find the accurate conversion of the rocket velocity represented by a 64-bit floating-point number into a 16-bit signed integer, as the resulting velocity (in 16-bit signed integer representation) was greater than 32,767, the largest number represented by a 16-bit signed integer.

These floating-point computation failures indicate that floating-point operations are very crucial for various applications. For instance, applications like computer graphics, mechatronics, aerospace vehicle, drone, etc., need to perform very precise and accurate computations in order to provide the desired outcome [4]. This paper focuses on reviewing the implementation of floating-point addition operation, as almost half of the floating-point operations are dominated by addition and subtraction operations [5], as shown in Figure 1. For instance, 60% of signal processing algorithms require addition operation [6].

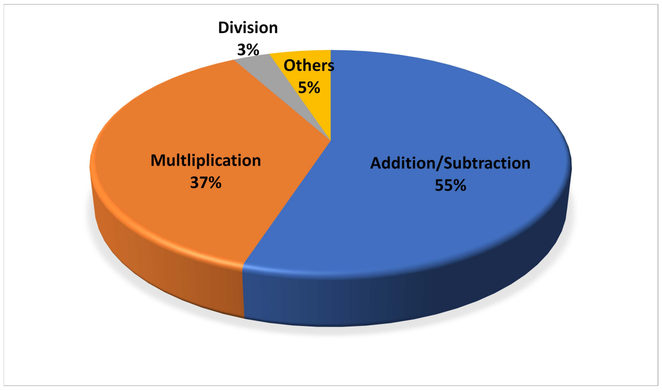


Figure 1. Distribution of floating-point instructions [5].

Several patents deal with the design and implementation of Floating-Point Adder (FPA) [7–12]. FPA has various operations of variable latencies and the latency of FPA should be be optimized [13,14]; either by determining the smaller exponent to be subtracted from the larger exponent [15], or by using dual path algorithm [16–22], or considering subtraction operation separately [23]. Various design techniques are proposed to improve the performance of FPA to increase its speed [24–28], to reduce its silicon area [29], dealing with denormalized numbers [30] and rounding logic [31] and implementing FPA using field programmable gate array (FPGA) [32,33]. A Leading Zero Anticipator (LZA) is also designed for normalization process in dual path algorithm to improve the speed of FPA [34–38], and extensive research articles are available for synchronous FPA [39–45].

The process completion time between two synchronous logic blocks is evaluated based on their critical paths, and the clock period for the design must be larger than the worst of these critical paths, hence limiting the scope of speed improvement. Clock skew and hence clock delay balancing are difficult to manage due to technology scaling, as the clock signal needs to arrive at the same time at all storage elements [46–49]. Moreover, synchronous circuits invest 40% and more of its power in clock distribution [50,51], and as the design grows in complexity, additional delay units are required to tune the delay from the clock source to the flip-flops/latches to overcome clock skew [52–54]. This implies that the presence of a global clock signal leads to more latency and power consumption. Asynchronous circuits provide an alternate solution for these challenges that arise due to the presence of a global clock [49,55], as the clock signal is replaced by the handshaking (request *REQ* and acknowledge *ACK*) signals in such circuits. The datapath becomes active upon the reception of a request signal *REQ*, and it goes back to the inactive state after it has completed its operation and issued an acknowledge signal *ACK*.

The asynchronous design approach is not new; in fact, circuits built right after the invention of the transistor were asynchronous. The ORDVAC at the University of Illinois and the IAS machine at Princeton [56] were examples of these earlier designs. Later, it was realized that the presence of a global clock would help to build smaller and faster circuits, and today it is the preferred design method supported by a vast Electronic Design Automation (EDA) industry. The involvement of technology in our day to day life, however, creates the need for even faster and more compact electronic devices with lower power consumption. The technology scaling imposes various limitations on synchronous circuits [49,57–59], and the asynchronous design approach is being reconsidered by various researchers to overcome the limitations due to the presence of the clock signal. The demand for portable electronic devices with minimal power consumption [60] without compromising the processing speed and silicon area is another major concern [61,62]. Various asynchronous digital circuits are designed and commercialized by the leading companies such as IBM, Intel, Philips Semiconductors, Sun Microsystems (now Oracle), etc., [63–67] over the last two decades with considerable cost benefits. Several successful industrial experiments have also been performed to support the asynchronous circuit design such as Intel RAPPID [68], IBM FIR filter [69,70], optimizing continuous-time digital signal processors [71,72], developing ultra-low-energy devices [73–76], system design to handle extreme temperature [77] and finally, developing alternative computing paradigms [78–80]; however, these experiments were not commercialized. One of the primary reasons for the absence of commercial asynchronous circuits is the absence of sufficiently mature asynchronous EDA tools [51,81]. Fortunately, several languages and design tools are being developed for asynchronous approach such as UNCLE (Unified NULL Convention Logic Environment) [82], Tangram [65,83–88], CHP (communicating hardware processes) [89–95], BALSA [96], asynchronous circuit compiler [97–100], Petrify [101–103] and various other tools [104–113], as the asynchronous circuit design promises to overcome the limitations posed by the synchronous logic due to technology scaling [114].

An FPA requires various operations of variable latencies, and the asynchronous design approach can utilize this feature of FPA to optimize its speed and energy consumption. The asynchronous approach is capable of indicating the process completion as soon as the computation is done, therefore, the AFPA does not have to wait for the worst-case delay if the current computation has finished earlier than that; and the power rail would go to ideal state if there is no ongoing activity. However, limited research work is available on implementing an FPA using asynchronous approach, and all of them have utilised dual-rail coding for completion detection. This paper aims to review the existing designs of asynchronous floating-point adder (AFPA). The organization of the rest of the manuscript is as follows: Section 2 discusses the basic algorithm of floating-point adders. Various communication protocols used to control the datapath of asynchronous circuits are discussed in Section 3. Section 4 reviews the existing asynchronous floating-point adder designs. A summarised comparison of various

AFPA design is provided in Section 5 along with a discussion of the scope of implementing an AFPA using bundled data protocol, followed by the conclusion in Section 6.

2. Basic Operation of Floating-Point Adder

The implementation of a floating-point adder is more complicated than an integer adder, as floating-point numbers cannot perform addition/subtraction without a few essential preliminary steps, which consists of at least six operations with different computation time [115]. Floating-point numbers are represented using the IEEE754 format developed by the Technical Committees of the IEEE Societies [116,117], and it is widely accepted by academics and industries.

A standard floating-point representation is shown in Figure 2 with sign bit S, characteristics E, and significand. The mantissa M is represented as 'h.significand'; '*h*' is the hidden bit that is not stored in the memory, but it is used for computation, and the value of the hidden bit is always 1.

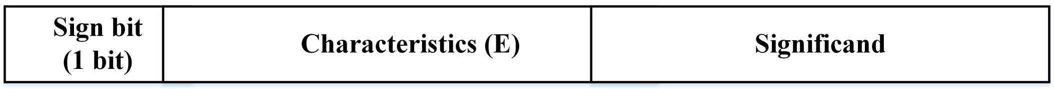


Figure 2. Floating-point representation.

There are two primary floating-point representations defined in the IEEE754 format: Single-Precision and Double Precision, as shown in Figure 3. A single-precision floating-point number is 32-bit long: the MSB is the sign bit, the next eight bits the exponent, and the final 23-bits the significand. A double-precision floating-point number is 64-bits. The MSB represents the sign bit, followed by an 11-bit exponent and a 52-bit significand. The hidden bit '1' is virtually present in both representations with the significand to provide the mantissa.

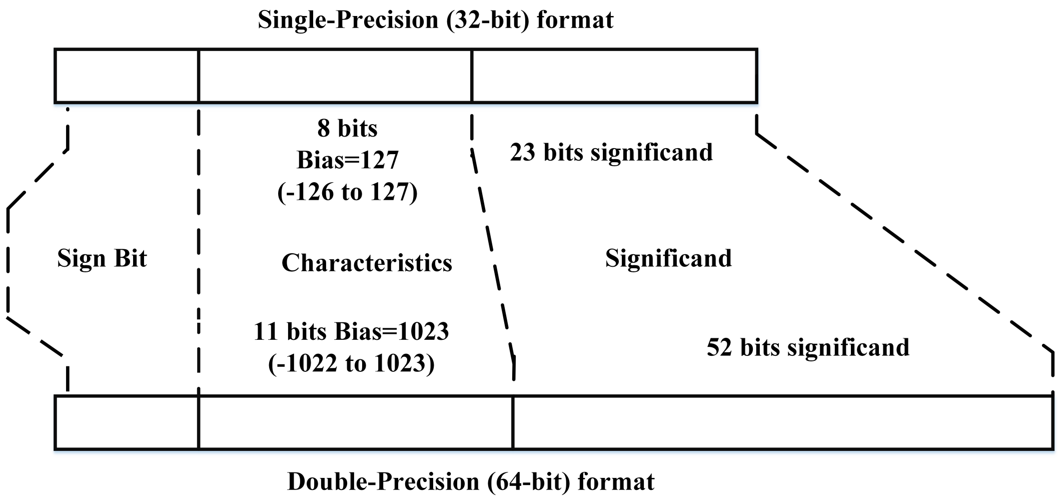


Figure 3. Floating-point representation format.

Figure 4 shows the basic algorithm to add/subtract two floating-point numbers A and B, as explained in the following steps, where operands A and B are represented in the IEEE format as S_A E_A M_A and S_B E_B M_B, respectively.

Step 1: Calculate the exponent difference: $d = |E_A - E_B|$

Step 2: Alignment: Monitor the carry C_{out} to identify the smaller operand. The smaller operand needs to be shifted by amount d before performing addition and the larger operand is directly fed to the adder.

Step 3: The shifter output is fed to the XOR gates with a Control signal such that:

If Control = 0, effective operation is addition. The shifted mantissa from Step 2 would be transferred as it is to the adder with C_{in} = 0, as performing XOR operation with zero bit would provide the same variable, i.e., $x \oplus 0 = x$

If Control = 1, effective operation is subtraction. 2′s complement of the shifted mantissa from Step 2 would be transferred to the adder, as performing XOR operation with one bit would provide the complement of the variable, i.e., $x \oplus 1 = x'$ and C_{in} = 1 would provide the additional 1 for 2′s complement. The sign of the result would be calculated by the sign computation block.

Step 4: Add the shifted mantissa with the mantissa of the other number. The summation output is sent to the normalization unit to provide the final outcome in the IEEE format. The normalization unit performs the following operations:

- Convert the resultant mantissa into a signed magnitude format, find the 2′s complement of the result if it is negative.
- Leading One Detector (LOD): Detect the leading one in the result after subtraction operation, to find the amount required for left shifting.
- For addition operation, either no shifting or maximum 1-bit right shifting is required.
- Convert the result into the standard IEEE format.
- Rounding: Round off the bits shifted out due to normalization.

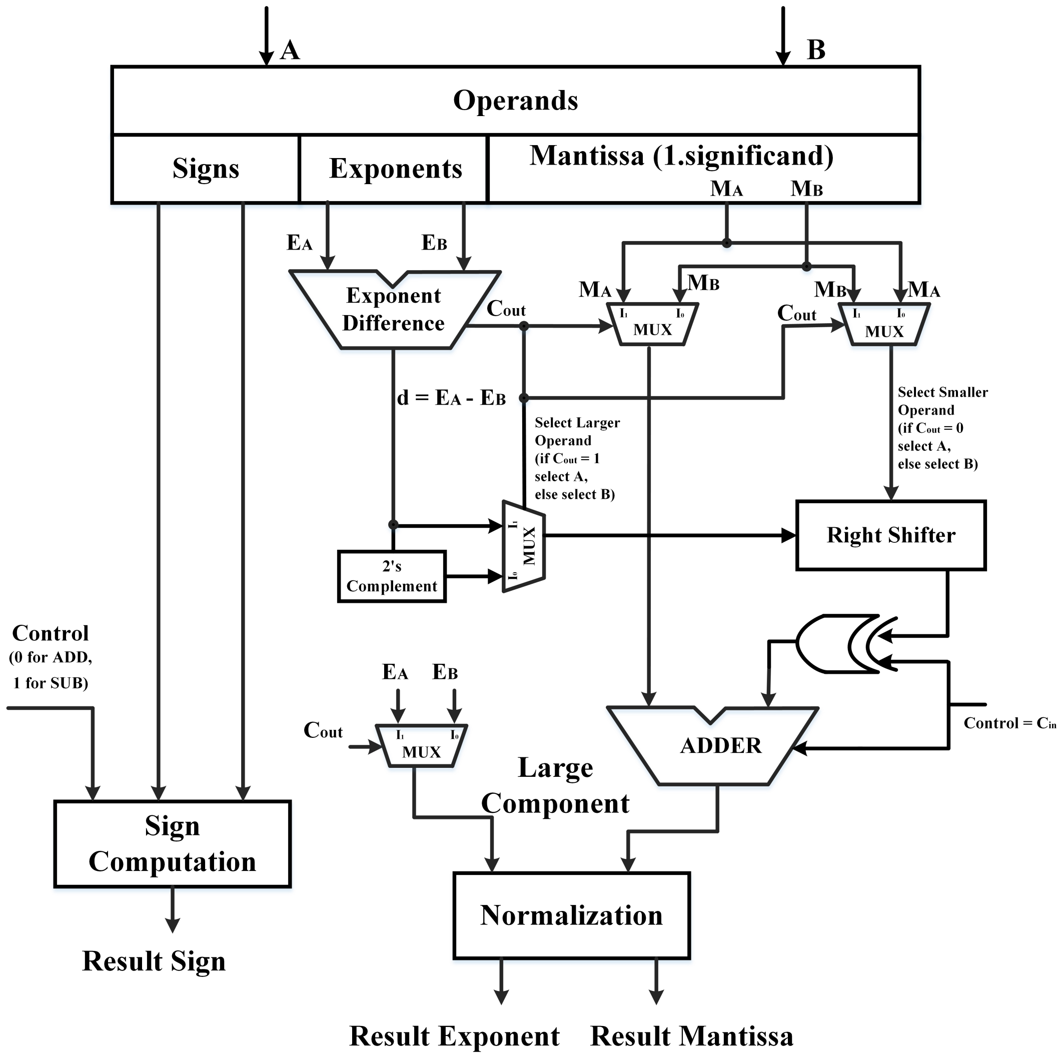


Figure 4. Basic asynchronous floating-point adder (AFPA) architecture.

The basic algorithm is modified by several researchers to improve the performance of the FPA. For example, introducing the FAR/CLOSE algorithm to utilize two paths for two different cases of subtraction [34,39,42], or by replacing LOD with LZA [34–38] to mention a few modifications. It is evident from the basic steps of floating-point addition algorithm that FPA requires various operations with variable computational time in order to provide the final output. The synchronous FPA, however, cannot take the advantage of variable computational time as the time-period of the clock signal is fixed, which is calculated according to the critical path delay or the worst-case delay. It cannot start the next task unless the next clock pulse is available, even if the ongoing computation has finished earlier than the critical path delay. On the other hand, some asynchronous implementations of FPAs can indicate process completion as soon as the computation is done, and therefore do not have to wait for the worst-case delay if the current computation has finished earlier than that. However, asynchronous design approach requires communication protocol to indicate that the process is done before sending the acknowledge signal, as there is no global clock to synchronize the timings. The next section discusses the communication protocols used for asynchronous circuits and compares the effect of different communication protocols on asynchronous circuit design.

3. Communication Protocols for Asynchronous Designs

The absence of a global clock in asynchronous circuits has the potential to overcome the clock distribution and clock skew related challenges, but it creates the need for a communication protocol to detect the process completion and data validity [48,118,119]. As asynchronous circuits are event-driven, a circuit becomes active only after receiving a request signal and goes back to an inactive state once the process is complete and sends the acknowledge signal. This process is controlled by the communication protocol in asynchronous circuits with the help of data encoding, and the most widely accepted encoding protocols are

1. Bundled Data Scheme: In the bundled data scheme, a single bit is encoded with one wire [81] similar to the synchronous circuit [120], hence, it is also known as Single-Rail Protocol.

   The circuit starts its process after receiving a request signal *REQ* and sends acknowledgment *ACK* after the worst-case delay required by the critical path of the circuit to ensure the process completion and data validity, as shown in Figure 5.

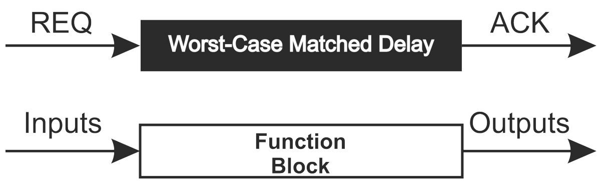
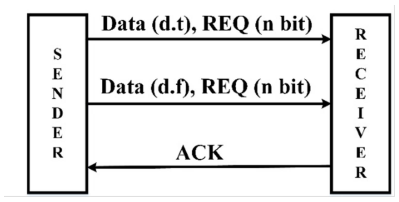


Figure 5. Worst-Case Bundled-data completion detection technique.

2. Dual-Rail Protocol: In the dual-rail protocol, a single bit is encoded with two wires. To encode n-bit data 'd', $2n$ wires are required ($d.t$ and $d.f$ for the true and false value of data, respectively), and the request signal is encoded with the data [121], as shown in Figure 6.

   Once the data is available at the receiver, a completion detection circuit is used to determine the value of data 'd' by observing the $d.t$ and $d.f$ signals. This implies that the dual-rail coding requires more implementation area compared to bundled data or synchronous design, but it can indicate the data validity as soon as the computation is done. However, detection of valid data from $d.t$ and $d.f$ signals requires a completion detection technique, which needs an additional process delay, and it might not deliver the anticipated outcome.



	d.t	d.f
Spacer	0	0
Valid "0"	0	1
Valid "1"	1	0
Not used	1	1

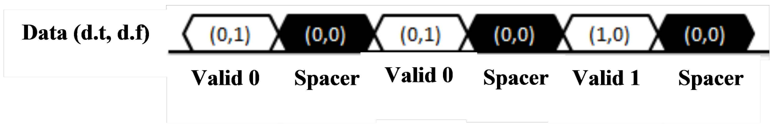


Figure 6. Dual-rail encoding protocol.

The other protocols used to encode data are special cases of *M-of-N* encoding, in which $log_2 N$ bits can be represented using N wires, and one extra wire is used to send the acknowledgment [122]. Circuits with such encoding are known as quasi-delay insensitive (QDI) circuits as it assumes a finite gate and wire delays [123–125]. Dual-rail coding is also a special case of *M-of-N* encoding with $M = 1$ and $N = 2$ [126].

Most of the asynchronous circuit designs utilize either bundled data or dual-rail coding to indicate the process completion. However, all of the existing designs of AFPA utilize dual-rail coding, as dual-rail design encodes the information of data validity with the data and allows the circuit to acknowledge as soon as the computation is done. The asynchronous design of AFPA is rarely discussed, and next section discusses all the existing designs of AFPA in detail.

4. Asynchronous Floating-Point Adders

The asynchronous implementation of floating-point adders has not been explored much, and most researchers have focused mainly on implementing asynchronous floating-point multiplication and division operations [127–131]. This could perhaps be due to the complexity involved in FPA implementation. The floating-point addition operation consists of various operations of variable latencies: mantissa shifting for exponent matching, addition of aligned mantissa, rounding, and normalization of the computed output. The synchronous FPA utilizes the worst-case delay to determine the clock frequency, hence, there is no need to worry about the process completion, as every process can complete within or before the worst-case delay. However, 60–90% of the addition process can take the benefit of early completion [132]. Implementing an asynchronous floating-point adder by replacing the clock pulse with *REQ* and *ACK* signals can take advantage of early completion detection and reduce the processing time from worst-case delay to average-case delay. The next section discusses the existing AFPA architecture proposed by Noche and Jose [133], Sheikh and Manohar [134–136], and Jun and Wang [115], and these are the only designs available in literature that discusses the implementation of AFPA architecture. The MTNCL approach discussed in [137] does not provide any details on implementing the AFPA; however, this paper is included for the review since it provides the performance comparison of floating-point addition/subtraction operations for both synchronous and asynchronous floating-point co-processor.

4.1. Single-Precision AFPA

This section discusses a single-precision asynchronous floating-point unit (AFPU) implemented using a variable latency algorithm proposed by Noche and Jose [133]. This design introduces the first asynchronous implementation of a single-precision floating-point adder [136] along with other arithmetic operations, while all the previous implementations of floating-point units have focused on multiplication or division operations. The design has used dual-rail differential cascode voltage switch (DCVS) logic for datapath and complementary metal-oxide semiconductor (CMOS) logic for the control path. The AFPU is designed at transistor level using 3.3 V supply voltage and 0.35 µm process, and Cadence software is used to design and test the arithmetic unit at the transistor level. This paper discusses the performance of AFPU for addition operation only.

Registers and adders are the two key components of the datapath for addition operation. Bidirectional shift registers are used to implement the shifter circuit required for exponent matching and normalization with a provision of rounding bit [138]. The two adders required to find the exponent difference and mantissa addition are designed by using 9-bit and 25-bit Carry Lookahead Adders (CLA), respectively [139]. DCVS multiplexers are used to select inputs for registers and adders, and it would be replaced by OR gates to optimize the design if the dual-rail inputs will never be active at the same time. The control circuitry of the AFPU includes logic gates, SR latches, and C-elements [140]. The asynchronous floating-point addition operation is event-driven, and it has utilized the dual-rail protocol with four-phase signaling to detect the process completion.

The process completion time reported by Noche and Jose for single-precision AFPA (SPAFPA) includes

- Time required for unpacking the operands t_{un}
- Time required to check for exceptions in input data (Not a number, zero, or infinity) t_{as}, typically 8 to 9 ns.
- Time required to determine larger operand t_{ad}
- Time required to match the exponents d $\times$ t_{aa}, where d = $|E_A - E_B|$
- Time required to provide the final outcome t_{al}, it will be small if the result is zero, and large for a negative result. However, value of t_{al} is less compared to the shift operation, therefore, the average value of t_{al} is considered during computation.

The completion time t can be calculated as

$$t = t_{un} + t_{as} + t_{ad} + d \times t_{aa} + t_{al} \approx a_a \times d + b_a \approx (22.8 \times d + 59)ns \quad (1)$$

For a single-precision operand, the value of d ranges from 0 to 254, providing the addition completion time ranges from 59 ns to 5850.2 ns, as reported by Noche and Jose. Simulation is performed using 248 test vectors at 25 °C with 5 fF (femtofarad) capacitance connected at the output. The average addition completion time reported is 127.4 ns for ten applications from the SPECfp92 benchmark suite.

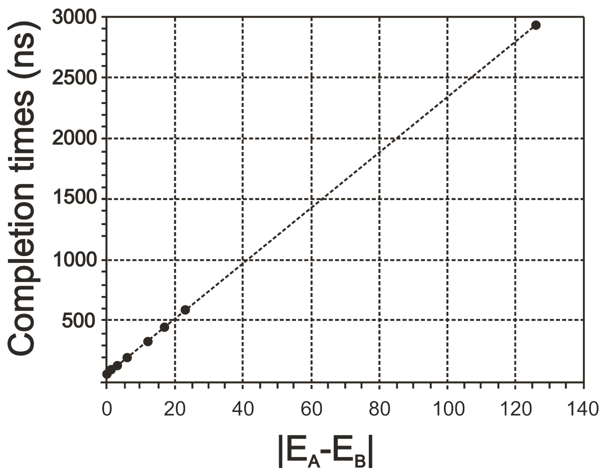


Figure 7. Addition completion times (ordinary cases) in ns as a function of $|E_A - E_B|$ [133].

A graph has been plotted by Noche & Jose between the addition completion time and exponent difference $|E_A - E_B|$ for ordinary cases (no exceptions), as shown in Figure 7. The computation time for the critical path of SPAFPA might be greater than the worst-case delay of its synchronous equivalent, but the computation time is much shorter when the exponent difference *'d'* is small. Cases with a large shift amount are not common, and 45% of cases have the shift amount either 0 or 1 [129,134] and as such the speed of the AFPA is much improved over its synchronous counterpart. A single test case to add two random numbers 4,195,835 and 3,145,727 has been also considered by Noche & Jose to evaluate the SPAFPA performance. The computation is done in 79.0 ns with 0.32 nJ energy and 4.08 mW power consumption.

The SPAFPA design focuses to reduce the processing time from the worst-case delay to average-case delay. Serial architecture is used to implement an area-efficient design, and the AFPU is designed by using 17,085 transistors only. However, it uses shift registers to shift the data for exponent matching, in which the processing time t_s is directly proportional to the exponent difference *'d'*. This would reduce the speed of the SPAFPA if the value of *'d'* is large. A logarithmic/barrel shifter would be a better choice for shifter as an N-bit barrel shifter requires only $log_2 N$ stages to implement, therefore reducing the processing time of shifting operation [141–143]. Furthermore, CLA can be replaced by a faster average case adder such as a parallel prefix adder, carry select adder, ripple carry adder [144], or any advanced adder design, as discussed in the next section. Shifter and adder are two basic modules of AFPA, and improving the processing time of these modules would optimize the processing time of AFPA.

4.2. Operand-Optimized Double-Precision AFPA

Noche and Joes claim to have reduced the process completion time from worst-case to average-case for a single-precision AFPA [133], but the processing time does not include the time to compute the rounding logic. Moreover, the design is completely non-pipelined, and it does not use any other energy optimization technique. Pipelining is a technique where multiple tasks are executed in parallel for different data values and consequently optimize the output, and several asynchronous pipelining techniques are used to optimize the throughput [105,145–153]. Sheikh & Manohar [136] have designed an operand-optimized double-precision AFPA (DPAFPA) along with all four rounding logic, as discussed in this section. The performance of the DPAFPA was compared with a baseline high performance AFPA, and the operating conditions were as follows: Temperature is 25 °C with 1 V supply voltage, in a 65 nm bulk CMOS process at typical-typical (TT) corner. The baseline AFPA consists of a 56-bit Hybrid Kogge Stone Carry Select Adder (HKSCSA) to add mantissa. The adder provides two speculative sum output for two different values of carry-in, and the final output will be selected at the final stage according to the actual value of carry-in. Dual-rail protocol is used with 1-of-4 encoding and radix-4 arithmetic to optimize the energy and speed requirements.

Normalization is done in parallel with the addition operation by using the Leading One Predictor (LOP) technique. The shift amount is speculated by the LOP, and the final outcome has to be shifted by one bit if the estimated shift amount is wrong [43]. The datapath is divided into two separate pipelines (Left and Right) to normalize the summation output. The left pipeline is used for a massive left shift required due to subtraction operation, and all other cases are managed by the right pipeline. 30 pipeline stages are used in datapaths with minimal increase in latency. It has utilized the pre-charge enable half-buffer (PCEHB) pipeline for all data computation [154], which is faster and more energy-efficient compared to the original pre-charge half-buffer (PCHB) pipeline [155]. Moreover, it uses weak condition half-buffer (WCHB) in spite of PCEHB for simple buffers and tokens, as it is more energy efficient. A detailed power breakdown of the FPA datapath is shown in Figure 8, which indicates that addition is the highest power-consuming operation, followed by the right shift operation. The DPAFPA design proposed by Sheikh improves the power saving compared to the baseline AFPA by making the following changes:

- The HKSCSA is replaced by an interleaved asynchronous adder, which utilizes two ripple-carry adders of radix-4. Both of the ripple-carry adders can perform parallelly for different input operands, one adder is used to add even operand pairs, and the other one adds odd operand pairs. The length of maximum carry-chain is seven for radix-4 arithmetic for approximately 90% cases, and the requirement of energy/operation by an interleaved adder is 2.9 pJ/op for the carry length less than 15 with a throughput of 2.2 GHz. On the contrary, the 56-bit adder (HKSCSA) used by the baseline FPA needs 13.6 pJ/op with a throughput of 2.17 GHz. Therefore, the reduction in power consumption by an interleaved adder is more than four times compared to HKSCSA. The number of transistors required by a 56-bit adder is also reduced by 35% for interleaved adders.

- The right shifter is designed by using three pipeline stages: Stage 1 shifts the mantissa between 0 to 3 bits, Stage 2 shifts the mantissa by 0, 4, 8, or 12 bits, and Stage 3 shifts the mantissa by 0, 16, 32 or 48 bits. The computation time of shifter to shift the mantissa between 0 to 55 bits is fixed in baseline AFPA. Sheikh's AFPA design has split the shifter into two paths: long path and short path, which allows the shifter to select a path according to the shift amount and bypass the other path. The shifter design is data-driven, and it can optimize power consumption.
- The LOP scheme is modified in the design, and only one pipeline (either left or right) is used for normalization. The selection of the left or right pipeline is made before activating the LOP stage. The left and right pipelines provide up to 13% and 18% of power-saving, respectively, compared to baseline AFPA.
- The post-add right pipeline manages the left/right 1-bit shifter, 53-bit mantissa incrementor, rounding operation, and calculation of the final value of the exponent. The DPAFPA design uses the interleaved incrementor, similar to the interleaved adder, compared to the carry-select incrementor used by the baseline AFPA. It will make DPAFPA more energy efficient.
- The design can detect the zero input operands. If one or both operands are zero, the final outcome can be given without using the power-consuming blocks of AFPA.

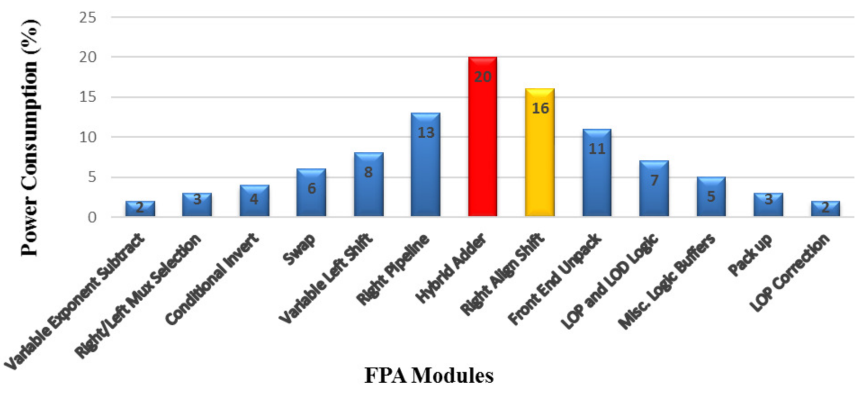


Figure 8. Data encoding protocols [136].

The DPAFPA design requires 30.2 pJ/op, compared to the baseline AFPA, which consumes 69.3 pJ/op, resulting in 56.7% of reduction in energy consumption [136]. The performance of DPFPA is also compared with a synchronous FPA proposed by Quinnell [156], as it is one of the rare designs of fully implemented FPA and it provided a good baseline for analyzing the performance of DPFPA. The synchronous is FPA designed using a standard-cell library with 65 nm SOI (Silicon-On-Insulator) process. Performance comparison of DPAFPA, baseline FPA and a synchronous design of FPA considered by Sheikh & Manohar is given in Table 1 [136].

Table 1. Comparison of DPAFPA with baseline FPA and synchronous FPA [136].

S. No.	Name	Process	VDD (Volts)	Frequency (GHz)	Latency (ps)	Power (mW)	Energy/ Operation (pJ)	GFLOPS/W
1	DPAFPA	65 nm	1	2.15	1060	64.9	30.2	33.1
2	Baseline AFPA	65 nm	1	2.15	1098	149	69.3	14.5
3	Synchronous FPA [156]	65 nm SOI	1.3	666	946	118	177.17	5.64

GFLOPS (gigaflops) is used to measure the performance of a floating-point unit (FLOPS—floating-point operations per second). A high GFLOPS/W for the proposed DPFPA [136] makes the asynchronous design scheme suitable for optimizing the circuit performance. The input set for both baseline AFPA and DPFPA is taken for the right shift amount ranging from 0 to 3, and it considers only non-zero operands.

The use of interleaved adders and shifters helped to reduce the power consumption of the circuit. The shifter architecture is also split and implemented with pipelines, therefore reducing the processing time and power consumption of the circuit. A gate-level simulation tool PRISM is used to design and test AFPA, by using ten billion random input operands, and one billion stored inputs from actual application benchmark. The design is also tested for exceptions (NaN, Zero, Infinity, Denormal numbers). This design implementation of DPAFPA focuses mainly on reducing energy/operation and power consumption using the pipelining technique, with minimal increment in the processing time.

4.3. Double-Precision AFPA with Operand-Dependent Delay Elements

The desynchronization technique provides better performance compared to the synchronous design [140,157], and it can also be used to implement the AFPA. However, it cannot take advantage of the event-driven nature of asynchronous circuits, as the clock signal is replaced by worst-case delay models during desynchronization. Xu and Wang [115] have examined the speed of various sub-operation required to perform floating-point addition and proposed an AFPA design with operand-dependent delay elements, as discussed in this section. A synchronous FPA [158] is used as the baseline (by Xu and Wang), which has the FAR/CLOSE path architecture, a balanced 56-bit shifter with LOP, and rounding by injection technique. This synchronous FPA is redesigned by Xu and Wang by using asynchronous logic with variable-length delay elements to utilize its event-driven property. Various sub-operations of AFPA with different computation time should be identified in order to select the delay models. At least six operations processing at different speeds have been identified [115] as follows:

1. The computation time can be reduced if the computation involves zero because most of the steps of floating-point addition can be skipped.
2. CLOSE path computation time: depends upon the computation time of LOP and compound adder.
3. FAR path computation time: It can be further calculated depending upon the value of exponent difference.
    (a) BIG_FAR computation time: The exponent difference is more than 56 for double-precision FPA.
    (b) NOM_FAR computation time: The exponent difference is less than 56 for double-precision FPA.
4. The FPA design supports EVZOUI (E-unimplemented, V-invalid, Z-divide by zero, O-overflow, U-underflow, I-inexact) exceptions, and E and I exceptions can be detected at the initial stages depending upon the input operands.
    (a) Computation time for evaluating E exceptions.
    (b) Computation time for evaluating I exceptions.

It is evident from the above discussion that at least six delay elements with variable latencies are required. However, Xu and Wang have used only three variable-length delay elements with six multiplexers to design AFPA to reduce the area overhead. A two-phase MOUSETRAP pipelining [159] is used instead of master-slave latches, and it utilized dual-rail protocol for completion detection to generate control information. A total of 10,000 random inputs are taken for simulation from six benchmarks. The design claims to improve the speed of proposed AFPA by 33%, reduce the energy consumption 12% but increase the area by 5% compared to its synchronous counterpart.

4.4. Multi-Threshold NULL Convention Logic (MTNCL)

Liang et al. [137] have proposed a Multi-Threshold NULL Convention Logic (MTNCL) or Sleep Convention Logic (SCL), which is a combination of Multi-Threshold CMOS (MTCMOS) with NULL Convention Logic (NCL). MTCMOS is designed using transistors with different threshold voltages (V_t) viz. Low V_t (high leakage current, fast speed) and High V_t (lesser leakage current, slower speed). Low V_t and high V_t are combined to design MTCMOS to preserve the performance with less leakage. MTCMOS has a sleep mode which maintains minimum power dissipation when the circuit is not active. However, maintaining the sleep signal requires complex logic since it is critical to the timing constraints, and transistor sizing and logic block partitioning is difficult for synchronous circuits. On the other hand, NCL utilizes asynchronous dual-rail design, which requires two wires to implement a single bit, along with a spacer or NULL signal as shown in Figure 6. Combining MTCMOS with NCL in MTNCL allows the circuit to utilize the sleep mode during NULL logic without dealing with clock-related challenges. A modification in the MTNCL architecture is done by placing the power gating high V_t transistor to the pull-down network. This design is known as the Static MTNCL threshold gate structure (SMTNCL), and it eliminates two bypass transistors and removes the output wake-up glitch.

Liang et al. have provided a comparison between a synchronous MTCMOS design and several variations of NCL designs for single-precision floating-point co-processors. The performance of co-processors is provided for addition/subtraction and multiplication operations, however, this paper discusses the performance for addition/subtraction operations only. An average time T_{DD} is considered for MTNCL circuits to process both data and NULL, which is comparable to the synchronous clock period. The multi-threshold designs do not provide any specific architecture for AFPA, but these designs are considered in this survey paper due to less available literature on AFPA.

The comparison is given only for basic NCL designs (Low and High V_t), the best MTNCL design, and the synchronous MTNCL design, as provided in Table 2 [137].

Table 2. Multi-Threshold NULL Convention Logic (MTNCL) Comparison [137].

Circuit Type	Transistors	T_{DD} (ns)	Energy/ Operation (pJ)	Idle Power (nW)
MTCMOS Synchronous	104,571	10.0	124.3	156,000
NCL Low V_t	158,059	14.1	27.4	12,300
NCL High V_t	158,059	32.7	28.5	208.0
SMTNCL with SECRII w/o nsleep	90,041	10.0	12.1	112.1

It would require some basic understanding of available SMTNCL designs to understand their reportedly best MTNCL (SMTNCL with SECRII w/o nsleep) architecture [137]. The basic design of MTNCL uses the Early Completion Input-Incomplete (ECII) feature, which puts a stage to sleep only when all the inputs are NULL. A variation of the design known as SECII puts the combinational logic of NCL circuit to sleep during the NULL cycle to reduce power dissipation. Another variation in the design known as SECRII makes the completion and registration logic to sleep along with the combinational logic when the circuit is not active. Signals *sleep* and *nsleep* ($\overline{sleep}$) are used to put the circuit in sleep mode. However, when the SMTNCL circuit combines with bitwise MTNCL, it would remove the need for *nsleep* signal, and provides the SMTNCL with SECRII w/o *nsleep* architecture. This design is reported as the best design by Liang et al. [137] when simulated for 25 sets of randomly selected floating-point numbers as it requires 86% less energy, three orders of magnitude less idle power and 14% less area; however, the speed is slower (not less than 2×) compared to the synchronous MTCMOS design.

5. Discussion

The asynchronous circuit design has the potential to improve the performance of digital circuits, especially in terms of speed and power consumption. It can also eliminate the limitations of synchronous circuits imposed by clock signals due to technology scaling. The performance of existing AFPA designs is analyzed in Section 4, and their comparison is given in Table 3.

Table 3 indicates that most of the asynchronous implementation of floating-point adders provides better performance compared to their synchronous counterparts. However, all the existing AFPA designs have been implemented using the dual-rail protocol, which requires two wires to implement a single bit data. Moreover, the process of determining data validity in dual-rail circuits requires a large number of gates, as each bit in the datapath needs to be examined in the process. The logic to determine data validity might take a considerable amount of time and higher power consumption for some applications, therefore, it may not deliver the anticipated outcome [160].

A different approach has been used to implement a few asynchronous circuits using the bundled data scheme with completion detection techniques [112,161–165], which can indicate the data validity as soon as the process is complete. A speculative completion detection scheme is designed for asynchronous fixed-point adders [161,162], and for barrel shifters [163], where the datapath channel is implemented with multiple delay models, including the worst-case delay. The bundled data adder and shifter implementation using a speculative completion detection technique provided a better performance compared to synchronous design without increasing the silicon area significantly like dual-rail circuits. However, there is no design available for AFPA using bundled data protocol. Shift and add are two fundamental operations of AFPA, and since a bundled data implementation for asynchronous shifter and fixed-point adder are already available with a speculative completion detection scheme, there exists a possibility of implementing a bundled data AFPA with speculative completion detection technique. Moreover, the speculative design of a bundled data fixed-point adder can be replaced by a deterministic completion detection technique adder proposed by Lai [165,166] to further improve the AFPA performance.

Table 3. Comparison of existing asynchronous floating-point adder (AFPA) designs.

Features/Citations	**SPAFPA** [133]	**DPAFPA** [136]	**DPAFPA with Operand-Dependent Delay Elements** [115]	**SMTNCL with SECRII w/o nsleep** [137]
Floating-point format	Single-Precision	Double-Precision	Double-Precision	Single-Precision
Adder	CLA	HKSCSA	Not mentioned	Not mentioned
Shifter	Shift Registers	Logarithmic Shifter	Not mentioned	Not mentioned
Pipeline	Non-pipelined	PCEHB and WCHB	MOUSETRAP	4-stage pipeline
Communication Protocol	Dual-Rail with 4-phase handshaking	Dual-Rail with 4-phase handshaking	Dual-Rail	Dual-Rail with 4-phase handshaking
Processing Time	59 ns to 5850.2 ns for $0 \leq d \leq 254$ and applications from SPECfp92 benchmark suite provide an average time of 127.4 ns	737 ps for zero operand cases and 1060 ps for non-zero operands with shift align (0–3); Latency reduction by 32.8% and 3.5% respectively	Six benchmarks with 10,000 operand pairs are designed, and it provides 33% performance advantage over its synchronous version	Requires more processing time than synchronous MTCMOS for 25 sets of randomly selected floating-point nos. (not more than $2\times$)
Energy and Power Consumption	Energy consumption is 0.32 nJ and power dissipation is 4.08 mW, reported for addition of two random operands 4,195,835 and 3,145,727	Energy consumption is 30.2 pJ and power dissipation is 464.9 mW, for non-zero operands with shift align (0–3); 56.7% reduction in energy consumption	Energy consumption is 12% less than its synchronous version and 15% less energy than desynchronized design, for six benchmarks with 10,000 operand pairs	86% less energy, and three orders of magnitude less idle power than the synchronous MTCMOS design for 25 sets of randomly selected floating-point nos.
Area	Serial architecture is used and the AFPU is implemented by using only 17,085 transistors	12% less transistors are used compared to baseline AFPA	5% more area required compared to the desynchronized design	14% less area than the synchronous MTCMOS design
Other Features	0.35 µm process, 3.3 V, 25 °C	65 nm bulk process, 1 V, 25 °C	65 nm process	130 nm process, 1.2 V

6. Conclusions

Floating-point addition and subtraction are the most frequent arithmetic operation in the typical scientific applications, yet very few research articles are available for the design of asynchronous floating-point adder. The existing designs of AFPA are designed using dual-rail coding which requires a large implementation area, but their speed and power consumption has been improved compared to their baseline FPA. This survey paper discussed all the four existing designs of AFPA, comparing different performance features with their respective baseline FPA. An absolute comparison of all the designs is not possible as all the existing designs have different performance features; however, these are the only implementation of AFPA available in the literature and it shows that the asynchronous design technique has the potential to improve the performance of AFPA. It also discusses the probable outcome of AFPA designed using bundled data protocol using some completion detection technique.

Author Contributions: Conceptualization, P.S.; Formal analysis, P.S. and E.C.; Investigation, P.S.; Methodology, P.S. and E.C.; Writing—original draft preparation, P.S.; Writing—review and editing, P.S., E.C. and S.O.; Supervision, E.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the European Regional Development Fund in the Research Centre of Advanced Mechatronic Systems project, grant number CZ.02.1.01/0.0/0.0/16_019/0000867 within the Operational Programme Research, Development, and Education.

Acknowledgments: This work is supported by Taylor's University under its TAYLOR'S RESEARCH SCHOLARSHIP programme through grant TUFR/2017/001/01.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACK	Acknowledge
AFPA	Asynchronous Floating-Point Adder
AFPU	Asynchronous Floating-Point Unit
CHP	Communicating Hardware Processes
CLA	Carry Lookahead Adders
CMOS	Complementary Metal-Oxide Semiconductor
DCVS	Differential Cascode Voltage Switch
DPAFPA	Double-Precision Asynchronous Floating-Point Adder
ECII	Early Completion Input-Incomplete
EDA	Electronics Design Automation
fF	femtofarad
FLOPS	Floating-Point operations per second
FPA	Floating-Point Adder
FPGA	Field Programmable Gate Array
GFLOPS	Gigaflops
HKSCSA	Hybrid Kogge Stone Carry Select Adder
IAS	Institute for Advanced Study
LOD	Leading One Detector
LOP	Leading One Predictor
LZA	Leading Zero Anticipator
MSB	Most Significant Bit
MTCMOS	Multi-Threshold CMOS
MTNCL	Combination of MTCMOS with NCL
NCL	Null Convention Logic
ORDVAC	Ordnance Discrete Variable Automatic Computer
PCEHB	Pre-Charge Enable Half-Buffer

PCHB	Pre-Charge Half-Buffer
QDI	Quasi-Delay Insensitive
REQ	Request
SCL	Sleep Convention Logic
SECII	MTNCL with combinational logic slept
SECRII	MTNCL with combinational, completion and registration logic slept
SMTNCL	Static MTNCL
SOI	ilicon-On-Insulator
SPAFPA	Single-Precision Asynchronous Floating-Point Adder
TT	typical-typical
UNCLE	Unified NULL Convention Logic Environment
WCHB	Weak condition half-buffer

References

1. Overton, M.L. *Numerical Computing with IEEE Floating Point Arithmetic*; Siam: Philadelphia, PA, USA, 2001.
2. Goldberg, D. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv. (CSUR)* **1991**, *23*, 5–48. [CrossRef]
3. Behrooz, P. *Computer Arithmetic: Algorithms and Hardware Designs*; Oxford University Press: New York, NY, USA, 2000; Volume 19, pp. 512583–512585.
4. Joldeş, M.; Muller, J.-M. Algorithms for Manipulating Quaternions in Floating-Point Arithmetic. In Proceedings of the IEEE 27th Symposium on Computer Arithmetic (ARITH), Portland, OR, USA, 7–10 June 2020.
5. Oberman, S.F.; Flynn, M.J. Design issues in division and other floating-point operations. *IEEE Trans. Comput.* **1997**, *46*, 154–161. [CrossRef]
6. Pappalardo, F.; Visalli, G.; Scarana, M. An application-oriented analysis of power/precision trade-off in fixed and floating-point arithmetic units for VLSI processors. In *Circuits, Signals, and Systems*; Citeseer: Clearwater Beach, FL, USA, 2004; pp. 416–421.
7. Haener, T.; Roetteler, M.; Svore, K. Quantum Circuit Libraries for Floating-Point Arithmetic. U.S. Patent No. 10,699,209, 30 June 2020.
8. Yamada, H.; Murabayashi, F.; Yamauchi, T.; Hotta, T.; Sawamoto, H.; Nishiyama, T.; Kiyoshige, Y.; Ido, N.; Hitachi Ltd. Floating-Point Addition/substraction Processing Apparatus and Method Thereof. U.S. Patent 5,684,729, 4 November 1997.
9. Gorshtein, V.Y.; Grushin, A.I.; Shevtsov, S.R. Floating Point Addition Methods and Apparatus. U.S. Patent 5,808,926, 15 September 1998.
10. Kawaguchi, T. Floating Point Addition and Subtraction Arithmetic Circuit Performing Preprocessing of Addition or Subtraction Operation Rapidly. U.S. Patent 5,931,896, 3 August 1999.
11. Iourcha, K.I.; Nguyen, A.; Hung, D. Fast Adder/Subtractor for Signed Floating Point Numbers. U.S. Patent 6,175,851, 16 January 2001.
12. Resnick, D.R.; Moore, W.T. Floating-Point Adder Performing Floating-Point and Integer Operations. U.S. Patent 6,529,928, 4 March 2003.
13. Seidel, P.-M.; Even, G. Fast IEEE Floating-Point Adder. U.S. Patent 10/138,659, 20 March 2003.
14. Pangal, A.; Somasekhar, D.; Vangal, S.R.; Hoskote, Y.V. Floating Point Adder. U.S. Patent No. 6,889,241, 3 May 2005.
15. Lutz, D.R.; Hinds, C.N. Data Processing Apparatus and Method for Performing Floating Point Addition. U.S. Patent No. 7,433,911, 7 October 2008.
16. Nystad, J. Floating-Point Adder. U.S. Patent No. 9,009,208, 14 April 2015.
17. Langhammer, M. Variable Precision Floating-Point Adder and Subtractor. U.S. Patent No. 10,055,195, 21 August 2018.
18. Quinnell, E.C. High Performance Floating-Point Adder With Full in-Line Denormal/Subnormal Support. U.S. Patent No. 10,108,398, 23 October 2018.
19. Langhammer, M.; Pasca, B. Floating-Point Adder Circuitry with Subnormal Support. U.S. Patent Application No. 15/704,313, 14 March 2019.

20. Stiles, D. Method and Apparatus for Performing Floating Point Addition. U.S. Patent No. 5,764,556, 9 June 1998.
21. Eisen, L.E.; Elliott, T.A.; Golla, R.T.; Olson, C.H. Method and System for Performing a High Speed Floating Point Add Operation. U.S. Patent No. 5,790,445, 4 August 1998.
22. Oberman, S.F. Floating Point Arithmetic Unit Including an Efficient Close Data Path. U.S. Patent No. 6,094,668, 25 July 2000.
23. Nakayama, T. Hardware Arrangement for Floating-Point Addition and Subtraction. U.S. Patent No. 5,197,023, 23 March 1993.
24. Govindu, G.; Zhuo, L.; Choi, S.; Prasanna, V. Analysis of high-performance floating-point arithmetic on FPGAs. In Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, NM, USA, 26–30 April 2004; p. 149.
25. Malik, A.; Ko, S.-B. Effective implementation of floating-point adder using pipelined LOP in FPGAs. In Proceedings of the Canadian Conference on Electrical and Computer Engineering, Saskatoon, SK, Canada, 1–4 May 2005; pp. 706–709.
26. Karlstrom, P.; Ehliar, A.; Liu, D. High performance, low latency fpga based floating point adder and multiplier units in a virtex 4. In Proceedings of the 2006 NORCHIP, Linkoping, Sweden, 20–21 November 2006; pp. 31–34.
27. Akkaş, A. Dual-mode floating-point adder architectures. *J. Syst. Arch.* **2008**, *54*, 1129–1142. [CrossRef]
28. Tao, Y.; Deyuan, G.; Xiaoya, F.; Xianglong, R. Three-operand floating-point adder. In Proceedings of the 2012 IEEE 12th International Conference on Computer and Information Technology, Chengdu, China, 27–29 October 2012; pp. 192–196.
29. Ehliar, A. Area efficient floating-point adder and multiplier with IEEE-754 compatible semantics. In Proceedings of the 2014 International Conference on Field-Programmable Technology (FPT), Shanghai, China, 10–12 December 2014; pp. 131–138.
30. Mathis, B.; Stine, J. A Well-Equipped Implementation: Normal/Denormalized Half/Single/Double Precision IEEE 754 Floating-Point Adder/Subtracter. In Proceedings of the 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 15–17 July 2019; Volume 2160, pp. 227–234.
31. Nannarelli, A. Tunable Floating-Point Adder. *IEEE Trans. Comput.* **2019**, *68*, 1553–1560. [CrossRef]
32. Hassan, H.S.; Ismail, S.M. CLA based Floating-point adder suitable for chaotic generators on FPGA. In Proceedings of the 2018 30th International Conference on Microelectronics (ICM), Sousse, Tunisia, 16–19 December 2018; pp. 299–302.
33. Villalba, J.; Hormigo, J.; González-Navarro, S. Fast HUB Floating-point Adder for FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *66*, 1028–1032. [CrossRef]
34. Farmwald, P.M. *On the Design of High Performance Digital Arithmetic Units*; Stanford University: Stanford, CA, USA, 1982.
35. Hokenek, E.; Montoye, R.K. Leading-zero anticipator (LZA) in the IBM RISC System/6000 floating-point execution unit. *IBM J. Res. Dev.* **1990**, *34*, 71–77. [CrossRef]
36. Quach, N.; Flynn, M.J. *Leading One Prediction-Implementation, Generalization, and Application*; Computer Systems Laboratory, Stanford University: Stanford, CA, USA, 1991.
37. Oklobdzija, V.G. An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **1994**, *2*, 124–128. [CrossRef]
38. Suzuki, H.; Morinaka, H.; Makino, H.; Nakase, Y.; Mashiko, K.; Sumi, T. Leading-zero anticipatory logic for high-speed floating point addition. *IEEE J. Solid-State Circuits* **1996**, *31*, 1157–1164. [CrossRef]
39. Oberman, S.F.; Al-Twaijry, H.; Flynn, M.J. The SNAP project: Design of floating point arithmetic units. In Proceedings of the 13th IEEE Sympsoium on Computer Arithmetic, Asilomar, CA, USA, 6–9 July 1997; pp. 156–165.
40. Pillai, R.; Al-Khalili, D.; Al-Khalili, A.J. A low power approach to floating point adder design. In Proceedings of the International Conference on Computer Design VLSI in Computers and Processors, Austin, TX, USA, USA, 12–15 October 1997; pp. 178–185.
41. Oberman, S.F.; Flynn, M.J. Reducing the mean latency of floating-point addition. *Theor. Comput. Sci.* **1998**, *196*, 201–214. [CrossRef]

42. Beaumont-Smith, A.; Burgess, N.; Lefrere, S.; Lim, C.-C. Reduced latency IEEE floating-point standard adder architectures. In Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Cat. No. 99CB36336), Adelaide, Australia, 14–16 April 1999; pp. 35–42.
43. Bruguera, J.D.; Lang, T. Leading-one prediction with concurrent position correction. *IEEE Trans. Comput.* **1999**, *48*, 1083–1097. [CrossRef]
44. Nielsen, A.M.; Matula, D.W.; Lyu, C.N.; Even, G. An IEEE compliant floating-point adder that conforms with the pipeline packet-forwarding paradigm. *IEEE Trans. Comput.* **2000**, *49*, 33–47. [CrossRef]
45. Seidel, P.-M.; Even, G. On the design of fast IEEE floating-point adders. In Proceedings of the 15th IEEE Symposium on Computer Arithmetic, ARITH-15, Vail, CO, USA, 11–13 June 2001; pp. 184–194.
46. Renaudin, M. Asynchronous circuits and systems: A promising design alternative. *Microelectron. Eng.* **2000**, *54*, 133–149. [CrossRef]
47. Maitham Shams; Jo C. Ebergen; Mohamed I. Elmasry Asynchronous Circuits. Available online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.1778&rep=rep1&type=pdf (accessed on 12 October 2020).
48. Spars, J.; Furber, S. *Principles Asynchronous Circuit Design*; Springer: Boston, MA, USA, 2002.
49. Tabassam, Z.; Naqvi, S.R.; Akram, T.; Alhussein, M.; Aurangzeb, K.; Haider, S.A. Towards Designing Asynchronous Microprocessors: From Specification to Tape-Out. *IEEE Access* **2019**, *7*, 33978–34003. [CrossRef]
50. Krstic, M.; Grass, E.; Fan, X. Asynchronous and GALS design-overview and perspectives. In Proceedings of the 2017 New Generation of CAS (NGCAS), Genova, Genoa, 6–9 September 2017; pp. 85–88.
51. Shin, Z.; Oh, M.-H.; Lee, J.-G.; Kim, H.Y.; Kim, Y.W. Design of a clockless MSP430 core using mixed asynchronous design flow. *IEICE Electron. Express* **2017**, *14*, 20170162. [CrossRef]
52. Davis, A.; Nowick, S.M. An introduction to asynchronous circuit design. *Encycl. Comput. Sci. Technol.* **1997**, *38*, 1–58.
53. Donno, M.; Ivaldi, A.; Benini, L.; Macii, E. Clock-tree power optimization based on RTL clock-gating. In Proceedings of the 40th annual Design Automation Conference, Anaheim, CA, USA, 2–6 June 2003; pp. 622–627.
54. Srivastava, N.; Manohar, R. Operation-Dependent Frequency Scaling Using Desynchronization. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, *27*, 799–809. [CrossRef]
55. Naqvi, S.R.; Veeravalli, V.S.; Steininger, A. Protecting an asynchronous NoC against transient channel faults. In Proceedings of the 2012 15th Euromicro Conference on Digital System Design, Izmir, Turkey, 5–8 September 2012; pp. 264–271.
56. Wilcox, S.P. *Synthesis of Asynchronous Circuits*; University of Cambridge, Computer Laboratory: Cambridge, UK, 1999.
57. Naqvi, S.R. An asynchronous router architecture using four-phase bundled handshake protocol. In Proceedings of the International Multi-Conference on Computing in the Global Information Technology, Venice, Italy, 24–29 June 2012; pp. 200–205.
58. Naqvi, S.R. *A Non-Blocking Fault-Tolerant Asynchronous Networks-on-Chip Router*; Technische Universität Wien: Vienna, Austria, 2013.
59. Sadeghi, R.; Jahanirad, H. Performance-based clustering for asynchronous digital circuits. In Proceedings of the 2017 Iranian Conference on Electrical Engineering (ICEE), Tehran, Iran, 2–4 May 2017; pp. 238–243.
60. Wheeldon, A.; Morris, J.; Sokolov, D.; Yakovlev, A. Self-timed, minimum latency circuits for the internet of things. *Integration* **2019**, *69*, 138–146. [CrossRef]
61. Rodrigo, F.L.P.B. FD-SOI Technology Opportunities for More Energy Efficient Asynchronous Circuits. Ph.D. Thesis, Grenoble Alpes, Grenoble, France, 2019.
62. Oliveira, D.L.; Verducci, O.; Torres, V.L.; Saotome, O.; Moreno, R.L.; Brandolin, J.B. A Novel Architecture for Implementation of Quasi Delay Insensitive Finite State Machines. In Proceedings of the 2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Lima, Peru, 8–10 August 2018; pp. 1–4.
63. Davies, M.; Lines, A.; Dama, J.; Gravel, A.; Southworth, R.; Dimou, G.; Beerel, P. A 72-port 10G ethernet switch/router using quasi-delay-insensitive asynchronous design. In Proceedings of the 2014 20th IEEE International Symposium on Asynchronous Circuits and Systems, Potsdam, Germany, 12–14 May 2014; pp. 103–104.

64. Teifel, J.; Manohar, R. Highly pipelined asynchronous FPGAs. In Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 28 February–2 March 2004; pp. 133–142.
65. van Gageldonk, H.; van Berkel, K.; Peeters, A.; Baumann, D.; Gloor, D.; Stegmann, G. An asynchronous low-power 80C51 microcontroller. In Proceedings of the Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, San Deigo, CA, USA, 30 March–2 April 1998; pp. 96–107.
66. Fant, K.M. *Logically Determined Design: Clockless System Design with NULL Convention Logic*; Wiley: NewYork, NY, USA, 2005.
67. Merolla, P.A.; Arthur, J.V.; Alvarez-Icaza, R.; Cassidy, A.S.; Sawada, J.; Akopyan, F.; Jackson, B.L.; Imam, N.; Guo, C.; Nakamura, Y.; et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* **2014**, *345*, 668–673. [CrossRef]
68. Stevens, K.S.; Rotem, S.; Ginosar, R.; Beerel, P.; Myers, C.J.; Yun, K.Y.; Koi, R.; Dike, C.; Roncken, M. An asynchronous instruction length decoder. *IEEE J. Solid-State Circuits* **2001**, *36*, 217–228. [CrossRef]
69. Singh, M.; Tierno, J.A.; Rylyakov, A.; Rylov, S.; Nowick, S.M. An adaptively pipelined mixed synchronous-asynchronous digital FIR filter chip operating at 1.3 gigahertz. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2009**, *18*, 1043–1056. [CrossRef]
70. Singh, M.; Tierno, J.A.; Rylyakov, A.; Rylov, S.; Nowick, S.M. An adaptively-pipelined mixed synchronous-asynchronous digital FIR filter chip operating at 1.3 gigahertz. In Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems, Manchester, UK, 8–11 April 2002; pp. 84–95.
71. Aeschlimann, F.; Allier, E.; Fesquet, L.; Renaudin, M. Asynchronous FIR filters: Towards a new digital processing chain. In Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems, Crete, Greece, 19–23 April 2004; pp. 198–206.
72. Vezyrtzis, C.; Jiang, W.; Nowick, S.M.; Tsividis, Y. A flexible, event-driven digital filter with frequency response independent of input sample rate. *IEEE J. Solid-State Circuits* **2014**, *49*, 2292–2304. [CrossRef]
73. Liu, T.-T.; Alarcón, L.P.; Pierson, M.D.; Rabaey, J.M. Asynchronous computing in sense amplifier-based pass transistor logic. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2009**, *17*, 883–892.
74. Nielsen, L.S.; Sparso, J. Designing asynchronous circuits for low power: An IFIR filter bank for a digital hearing aid. *Proc. IEEE* **1999**, *87*, 268–281. [CrossRef]
75. Chang, K.-L.; Chang, J.S.; Gwee, B.-H.; Chong, K.-S. Synchronous-logic and asynchronous-logic 8051 microcontroller cores for realizing the internet of things: A comparative study on dynamic voltage scaling and variation effects. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2013**, *3*, 23–34. [CrossRef]
76. Christmann, J.F.; Beigne, E.; Condemine, C.; Leblond, N.; Vivet, P.; Waltisperger, G.; Willemin, J. Bringing robustness and power efficiency to autonomous energy harvesting microsystems. In Proceedings of the 2010 IEEE Symposium on Asynchronous Circuits and Systems, Grenoble, France, 3–6 May 2010; pp. 62–71.
77. Shepherd, P.; Smith, S.C.; Holmes, J.; Francis, A.M.; Chiolino, N.; Mantooth, H.A. A robust, wide-temperature data transmission system for space environments. In Proceedings of the 2013 IEEE Aerospace Conference, Big Sky, MT, USA, 2–9 March 2013; pp. 1–13.
78. Peper, F.; Lee, J.; Adachi, S.; Mashiko, S. Laying out circuits on asynchronous cellular arrays: A step towards feasible nanocomputers?. *Nanotechnology* **2003**, *14*, 469. [CrossRef]
79. Vacca, M.; Graziano, M.; Zamboni, M. Asynchronous solutions for nanomagnetic logic circuits. *ACM J. Emerg. Technol. Comput. Syst. (JETC)* **2011**, *7*, 15. [CrossRef]
80. Karaki, N.; Nanmoto, T.; Ebihara, H.; Utsunomiya, S.; Inoue, S.; Shimoda, T. A flexible 8b asynchronous microprocessor based on low-temperature poly-silicon TFT technology. In Proceedings of the ISSCC, 2005 IEEE International Digest of Technical Papers, Solid-State Circuits Conference, San Francisco, CA, USA, 10 February 2005; pp. 272–598.
81. Kondratyev, A.; Lwin, K. Design of asynchronous circuits using synchronous CAD tools. *IEEE Des. Test Comput.* **2002**, *19*, 107–117. [CrossRef]
82. Reese, R.B.; Smith, S.C.; Thornton, M.A. Uncle-an rtl approach to asynchronous design. In Proceedings of the 2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems, Lyngby, Denmark, 7–9 May 2012.
83. van Berkel, K. *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*; Cambridge University Press: Cambridge, UK, 1993.

84. van Berkel, K.; Burgess, R.; Kessels, J.L.; Peeters, A.; Roncken, M.; Schalij, F. A fully asynchronous low-power error corrector for the DCC player. *IEEE J. Solid-State Circuits* **1994**, *29*, 1429–1439. [CrossRef]
85. Peeters, A.; van Berkel, K. Single-rail handshake circuits. In Proceedings of the Second Working Conference on Asynchronous Design Methodologies, London, UK, 30–31 May 1995; pp. 53–62.
86. Van Berkel, K.; Burgess, R.; Kessels, J.; Peeters, A.; Roncken, M.; Schalij, F.; van de Wiel, R. A single-rail re-implementation of a DCC error detector using a generic standard-cell library. In Proceedings of the Second Working Conference on Asynchronous Design Methodologies, London, UK, 30–31 May 1995; pp. 72–79.
87. Kessels, J.; Kramer, T.; den Besten, G.; Peeters, A.; Timm, V. Applying asynchronous circuits in contactless smart cards. In Proceedings of the Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)(Cat. No. PR00586), Eilat, Israel, 2–6 April 2000; pp. 36–44.
88. Kessels, J.; Kramer, T.; Peeters, A.; Timm, V. DESCALE: A design experiment for a smart card application consuming low energy. In *European Low Power Initiative for Electronic System Design*; Springer US: New York, NY, USA, 2001; pp. 247–262.
89. Martin, A.J.; Burns, S.M.; Lee, T.-K.; Borkovic, D.; Hazewindus, P.J. The Design of an Asynchronous Microprocessor. Available online: https://apps.dtic.mil/dtic/tr/fulltext/u2/a447727.pdf (accessed on 12 October 2020).
90. Martin, A.J.; Lines, A.; Manohar, R.; Nystrom, M.; Penzes, P.; Southworth, R.; Cummings, U.; Lee, T.K. The design of an asynchronous MIPS R3000 microprocessor. In Proceedings of the Seventeenth Conference on Advanced Research in VLSI, Ann Arbor, MI, USA, 15–16 September 1997; pp. 164–181.
91. Renaudin, M.; Vivet, P.; Robin, F. ASPRO: An Asynchronous 16-bit RISC Microprocessor with DSP Capabilities. In Proceedings of the 25th European Solid-State Circuits Conference, Duisburg, Germany, 21–23 September 1999; pp. 428–431.
92. Manohar, R.; Kelly, C. Network on a chip: Modeling wireless networks with asynchronous VLSI. *IEEE Commun. Mag.* **2001**, *39*, 149–155. [CrossRef]
93. Boahen, K.A. A burst-mode word-serial address-event link-I: Transmitter design. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2004**, *51*, 1269–1280. [CrossRef]
94. Patel, G.N.; Reid, M.S.; Schimmel, D.E.; DeWeerth, S.P. An asynchronous architecture for modeling intersegmental neural communication. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2006**, *14*, 97–110. [CrossRef]
95. Martin, A.J.; Moore, C.D. *CHP and CHPsim: A Language and Simulator for Fine-Grain Distributed Computation*; Tech. Rep. CS-TR-1–2011; Department of Computer Science, California Institute of Technology: Pasadena, CA, USA, 2011.
96. Edwards, D.; Bardsley, A. Balsa: An asynchronous hardware synthesis language. *Comput. J.* **2002**, *45*, 12–18. [CrossRef]
97. Spear, C. *System Verilog for Verification: A Guide to Learning the Testbench Language Features*; Springer US. Science & Business Media: Berlin, Germany, 2008.
98. Yakovlev, A.; Vivet, P.; Renaudin, M. Advances in asynchronous logic: From principles to GALS & NoC, recent industry applications, and commercial CAD tools. In Proceedings of the Conference on Design, Automation and Test in Europe, Grenoble, France, 18–22 March 2013; pp. 1715–1724.
99. Tiempo Secure. ACC: Asynchronous Circuit Compiler. Available online: Http://www.tiempoic.com/products/sw-tools/acc.html (accessed on 23 December 2018).
100. Tiempo Secure. TAM16: 16-Bit Microcontroller IP Core. Available online: Http://www.tiempoic.com/products/ip-cores/TAM16.html (accessed on 23 December 2018).
101. Cortadella, J.; Kishinevsky, M.; Kondratyev, A.; Lavagno, L.; Yakovlev, A. Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. Inf. Syst.* **1997**, *80*, 315–325.
102. Akram, T.; Naqvi, S.R.; Haider, S.A.; Kamran, M. Towards real-time crops surveillance for disease classification: Exploiting parallelism in computer vision. *Comput. Electr. Eng.* **2017**, *59*, 15–26. [CrossRef]
103. Catalunya, U.P.d. Petrify: A Tool for Synthesis of Petri Nets and Asynchronous Circuits. Available online: https://www.cs.upc.edu/~jordicf/petrify/distrib/home.html (accessed on 24 December 2018).
104. Theodoropoulos, G.K.; Tsakogiannis, G.; Woods, J. Occam: An asynchronous hardware description language? In Proceedings of the EUROMICRO 97, 23rd EUROMICRO Conference: New Frontiers of Information Technology (Cat. No. 97TB100167), Budapest, Hungary, 1–4 September 1997; pp. 249–256.

105. Endecott, P.; Furber, S.B. *Modelling and Simulation of Asynchronous Systems Using the LARD Hardware Description Language*; SCS Europe BVBA: Machester, UK, 1998; pp. 39–43.
106. Kangsah, B.; Wollowski, R.; Vogler, W.; Beister, J. DESI: A tool for decomposing signal transition graphs. In Proceedings of the 3rd ACiD-WG Workshop, Heraklion, Greece, 27–28 January 2003.
107. Bardsley, A.; Tarazona, L.; Edwards, D. Teak: A token-flow implementation for the balsa language. In Proceedings of the 2009 Ninth International Conference on Application of Concurrency to System Design, Augsburg, Germany, 1–3 July 2009; pp. 23–31.
108. Blunno, I.; Lavagno, L. Automated synthesis of micro-pipelines from behavioral Verilog HDL. In Proceedings of the Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000) (Cat. No. PR00586), Eilat, Israel, 2–6 April 2000; pp. 84–92.
109. WorkCraft. Available online: Https://workcraft.org/(accessed on 24 December 2018).
110. Electrical Engineering and NUS Engineering. Asynchronous High Level Synthesis Tool (VERISYN). Available online: http://async.org.uk/besst/verisyn/ (accessed on 24 December 2018).
111. Sune Frankild and Hans P. Palbøl. Visual STG Lab. Available online: Http://vstgl.sourceforge.net/ (accessed on 24 December 2018).
112. DEMO Session. Available online: Http://conferences.computer.org/async2007 (accessed on 2 January 2019).
113. Handshake-Solution. TideTimeless Design Environment. Available online: Http://www.handshakesolutions.com (accessed on 2 January 2019).
114. Rahbaran, B.; Steininger, A. Is asynchronous logic more robust than synchronous logic?. *IEEE Trans. Depend. Secur. Comput.* **2008**, *6*, 282–294. [CrossRef]
115. Xu, J.; Wang, H. Desynchronize a legacy floating-point adder with operand-dependant delay elements. In Proceedings of the 2011 IEEE International Symposium of Circuits and Systems (ISCAS), Rio de Janeiro, Brazil, 15–18 May 2011; pp. 1427–1430.
116. Coonen, J.T. Special Feature an Implementation Guide to a Proposed Standard for Floating-Point Arithmetic. *Computer* **1980**, *13*, 68–79. [CrossRef]
117. IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std 754-1985* **2008**, *l*, 1–20. [CrossRef]
118. Nowick, S.M.; Singh, M. Asynchronous design—Part 1: Overview and recent advances. *IEEE Des. Test* **2015**, *32*, 5–18. [CrossRef]
119. Delvai, M.; Steininger, A. Solving the fundamental problem of digital design-a systematic review of design methods. In Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD'06), Dubrovnik, Croatia, 30 August–1 September 2006; pp. 131–138.
120. Bainbridge, J. *Asynchronous System-on-Chip Interconnect*; Springer London. Science & Business Media: London, UK, 2013.
121. Gilchrist, B.; Pomerene, J.H.; Wong, S. Fast carry logic for digital computers. *IRE Trans. Electron. Comput.* **1955**, *EC-4*, 133–136. [CrossRef]
122. Toosizadeh, N. Enhanced Synchronous Design Using Asynchronous Techniques. Ph.D. Thesis, University of Toronto, Toronto, ON, Canada, 2010.
123. Clark, W.A. Macromodular computer systems. In Proceedings of the Spring Joint Computer Conference, New York, NY, USA, 18–20 April 1967; pp. 335–336.
124. Stucki, M.J.; Ornstein, S.M.; Clark, W.A. Logical design of macromodules. In Proceedings of the Spring Joint Computer Conference, New York, NY, USA, 18–20 April 1967; pp. 357–364.
125. Naqvi, S.R.; Najvirt, R.; Steininger, A. A multi-credit flow control scheme for asynchronous NoCs. In Proceedings of the 2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Karlovy Vary, Czech Republic, 8–10 April 2013; pp. 153–158.
126. Verhoeff, T. A Theory of Delay-Insensitive Systems. Ph.D. Thesis, Eindhoven University of Technology, The Netherlands, 1994.
127. Williams, T.E.; Horowitz, M.A. A zero-overhead self-timed 160-ns 54-b CMOS divider. *IEEE J. Solid-State Circuits* **1991**, *26*, 1651–1661. [CrossRef]
128. Matsubara, G.; Ide, N. A low power zero-overhead self-timed division and square root unit combining a single-rail static circuit with a dual-rail dynamic circuit. In Proceedings of the Third International Symposium on Advanced Research in Asynchronous Circuits and Systems, Eindhoven, The Netherlands, 7–10 April 1997; pp. 198–209.

129. Won, J.-H.; Choi, K. Low power self-timed floating-point divider in 0.25 um technology. In Proceedings of the 26th European Solid-State Circuits Conference, Stockholm, Sweden, 19–21 September 2000; pp. 113–116.
130. Chen, R.-D.; Chou, Y.-C.; Liu, W.-C. Comparative design of floating-point arithmetic units using the Balsa synthesis system. In Proceedings of the 2011 International Symposium on Integrated Circuits, Singapore, 12–14 December 2011; pp. 172–175.
131. Sheikh, B.R.; Manohar, R. An asynchronous floating-point multiplier. In Proceedings of the 2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems, Lyngby, Denmark, 7–9 May 2012; pp. 89–96.
132. Wimer, S.; Albeck, A.; Koren, I. A low energy dual-mode adder. *Comput. Electr. Eng.* **2014**, *40*, 1524–1537. [CrossRef]
133. Noche, J.R.; Araneta, J.C. An asynchronous IEEE floating-point arithmetic unit. *Sci. Diliman* **2007**, *19*.
134. Sheikh, B. Operand-Optimized Asynchronous Floating-Point Arithmetic Circuits. 2012. Available online: https://ecommons.cornell.edu/bitstream/handle/1813/29239/brs39thesisPDF.pdf?sequence=1 (accessed on 12 October 2020).
135. Manohar, R.; Sheikh, B.R. Operand-Optimized Asynchronous Floating-Point Units and Method of Use Thereof. U.S. Patent No. 9,524,270, 20 December 2016.
136. Sheikh, B.R.; Manohar, R. An operand-optimized asynchronous IEEE 754 double-precision floating-point adder. In Proceedings of the 2010 IEEE Symposium on Asynchronous Circuits and Systems, Grenoble, France, 3–6 May 2010; pp. 151–162.
137. Zhou, L.; Parameswaran, R.; Parsan, F.A.; Smith, S.C.; Di, J. Multi-Threshold NULL Convention Logic (MTNCL): An ultra-low power asynchronous circuit design methodology. *J. Low Power Electron. Appl.* **2015**, *5*, 81–100. [CrossRef]
138. Kishinevsky, M.; Kondratyev, A.; Taubin, A.; Varshavsky, V. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*; John Wiley & Sons, Inc.: New York, NY, USA, 1994.
139. Ruiz, G. Addition to "Evaluation of three 32-bit CMOS" adders in DCVS logic for self-timed circuits. *IEEE J. Solid-State Circuits* **2000**, *35*, 1517. [CrossRef]
140. Blunno, I.; Cortadella, J.; Kondratyev, A.; Lavagno, L.; Lwin, K.; Sotiriou, C. Handshake protocols for de-synchronization. In Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems, Crete, Greece, 19–23 April 2004; pp. 149–158.
141. Oberman, S.F. Design Issues in High Performance Floating Point Arithmatic Units. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 1996.
142. Thornton, J. *Design of a Computer-The Control Data 6600*; Scott, Foresman and Co.: Glenview, IL, USA, 1970.
143. Palmer, J.F.; Ravenel, B.W.; Nave, R. Numeric Data Processor. U.S. Patent No. 4,338,675, 6 July 1982.
144. Smith, S.C.; DeMara, R.F.; Yuan, J.S.; Hagedorn, M.; Ferguson, D. NULL Conv. Mult. Accumulate Unit Cond. Rounding, Scaling, Saturation. *J. Syst. Arch.* **2002**, *47*, 977–998. [CrossRef]
145. Burleson, W.P.; Ciesielski, M.; Klass, F.; Liu, W. Wave-pipelining: A tutorial and research survey. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **1998**, *6*, 464–474. [CrossRef]
146. Hauck, O.; Huss, S. Asynchronous wave pipelines for high throughput datapaths. In Proceedings of the 1998 IEEE International Conference on Electronics, Circuits and Systems Surfing the Waves of Science and Technology (Cat. No. 98EX196), Lisboa, Portugal, 7–10 September 1998; Volume 1, pp. 283–286.
147. Lines, A.M. Pipelined Asynchronous Circuits. Ph.D. Thesis, California Institute of Technology, Pasadena, CA, USA, 1998.
148. Molnar, C.E.; Jones, I.W.; Coates, W.S.; Lexau, J.K.; Fairbanks, S.M.; Sutherland, I.E. Two FIFO ring performance experiments. *Proc. IEEE* **1999**, *87*, 297–307. [CrossRef]
149. Singh, M.; Nowick, S.M. MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines. In Proceedings of the 2001 IEEE International Conference on Computer Design: VLSI in Computers and Processors, ICCD, Austin, TX, USA, 23–26 September 2001; pp. 9–17.
150. Sutherland, I.; Fairbanks, S. GasP: A minimal FIFO control. In Proceedings of the Seventh International Symposium on Asynchronous Circuits and Systems, ASYNC 2001, Salt Lake City, UT, USA, 11–14 March 2001; pp. 46–53.
151. Ozdag, R.O.; Beerel, P.A. High-speed QDI asynchronous pipelines. In Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems, Manchester, UK, 8–11 April 2002; pp. 13–22.

152. Winters, B.D.; Greenstreet, M.R. A negative-overhead, self-timed pipeline. In Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems, Manchester, UK, 8–11 April 2002; pp. 37–46.
153. Naqvi, S.R.; Lechner, J.; Steininger, A. Protection of Muller-Pipelines from transient faults. In Proceedings of the Fifteenth International Symposium on Quality Electronic Design, Santa Clara, CA, USA, 3–5 March 2014; pp. 123–131.
154. Fang, D.; Manohar, A. Non-uniform access asynchronous register files. In Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems, Crete, Greece, 19–23 April 2004; pp. 78–85.
155. Lines, A. Pipelined Asynchronous Circuits. Master's thesis, California Institute of Technology, Pasadena, CA, USA, 1995.
156. Quinnell, E.; Swartzlander, E.E.; Lemonds, C. Floating-point fused multiply-add architectures. In Proceedings of the 2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 4–7 November 2007.
157. Andrikos, N.; Lavagno, L.; Pandini, D.; Sotiriou, C.P. A fully-automated desynchronization flow for synchronous circuits. In Proceedings of the 44th annual Design Automation Conference, San Diego, CA, USA, 4–8 June 2007; pp. 982–985.
158. Hu, W.; Wang, J.; Gao, X.; Chen, Y.; Liu, Q.; Li, G. Godson-3: A scalable multicore RISC processor with x86 emulation. *IEEE Micro* **2009**, *29*, 17–29. [CrossRef]
159. Singh, M.; Nowick, S.M. MOUSETRAP: High-speed transition-signaling asynchronous pipelines. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2007**, *15*, 684–698. [CrossRef]
160. Poole, N. Self-timed logic circuits. *Electron. Commun. Eng. J.* **1994**, *6*, 261–270. [CrossRef]
161. Nowick, S.M. Design of a low-latency asynchronous adder using speculative completion. *IEE Proc.-Comput. Digit. Tech.* **1996**, *143*, 301–307. [CrossRef]
162. Nowick, S.M.; Yun, K.Y.; Beerel, P.A.; Dooply, A.E. Speculative completion for the design of high-performance asynchronous dynamic adders. In Proceedings of the Third International Symposium on Advanced Research in Asynchronous Circuits and Systems, Eindhoven, The Netherlands, 7–10 April 1997; pp. 210–223.
163. Beerel, P.A.; Kim, S.; Yeh, P.-C.; Kim, K. Statistically optimized asynchronous barrel shifters for variable length codecs. In Proceedings of the 1999 International Symposium on Low Power Electronics and Design (Cat. No. 99TH8477), San Diego, CA, USA, 17 August 1999; pp. 261–263.
164. Peter, A.B.; Kim, K.-S. Statistically Optimized Asynchronous Barrel Shifters for Variable Length Codecs. *J. Korean Inst. Commun. Inf. Sci.* **2003**, *28*, 891–901.
165. Lai, K.K.; Chung, E.C.; Lu, S.-L.L.; Quigley, S.F. Design of a Low Latency Asynchronous Adder using Early Completion Detection. *J. Eng. Sci. Technol.* **2014**, *9*, 755–772.
166. Lai, K.K. *Novel Asynchronous Completion Detection For Arithmetic Datapaths*; Tech. Rep.; Taylor's University: Subang Jaya, Malaysia, 2016.