1 Business Proposal

The University of Seville is willing to sell or license the patents [1], [2] and [3] described in Sections 4 and 5 of this document.

2 Introduction

The computation of sine and cosine functions is fundamental in a wide range of applications, including that of signal processing [4,5]. Obvious examples are the computation of discrete cosine transforms (DCT), discrete sine transforms (DST), and their inverses (IDCT and IDST) [6]. A fused sine-and-cosine implementation is of major interest because various methods compute both and numerous applications require both [4]. In this contribution, the focus is on the implementation of devices that provide the sine and cosine of multiples of a constant angle ϕ, that is, $\sin(n\phi)$ and $\cos(n\phi)$, where n is an integer given as an input. Applications of such devices include the following:

- Implementing the sine and/or cosine functions in arithmetic units. For example, suppose an arithmetic unit must compute the sine and/or cosine of a number x using the IEEE 754-1985 double-precision format [7]. x is coded in a 64-bit word with 3 fields called *sign* (1 bit), *exponent* (11 bits), and *significand* (52 bits). The significand is a number in the range $[1, 2-2^{-52}]$ coded in fixed-point, while the exponent is an integer laying in the range $[-1023, 1024]$ coded in excess 1023. If the exponent is lower than -27, then the unit can simply return 1 as the cosine and x as the sine, assuming rounding to the nearest representable value [8]. Otherwise, it can return the sine and/or cosine of $n\phi$, where ϕ is the constant $2^{-27-52} = 2^{-79}$ and n is the integer $x2^{79}$.

- Generating the *twiddle factors* of a Discrete Fourier Transform [9]. The discrete Fourier transform (DFT) of a complex sequence x of length L is another complex sequence X of the same length defined by $X(k) = \sum_{t=0}^{L-1} x(t)W_L^{tk}$, where $W_L = e^{\phi i}$ and $\phi = -2\pi/L$. The twiddle factors are the integer powers of W_L, and there are L different twiddle factors. Thus, the twiddle factor of index n is $W_L^n = (e^{\phi i})^n = e^{n\phi i} = \sin(n\phi)i + \cos(n\phi)$.

Since the sine and cosine functions are computationally expensive, in applications where a low latency is required, the generator is implemented using lookup tables (LUT). This implementation approach is problematic if the input space is large. For example, consider the arithmetic unit mentioned in Section 2: as stated previously, exponents lower than -27 can be dismissed. However, even if the input angle is restricted to $[0, 2\pi)$, it is still necessary to consider 30 different exponent values and 2^{52} different significant values. A direct implementation would therefore require an LUT of $30 * 2^{52}$ entries. Another example is given by DFT engines for long sequences as required in PLC[10], DVB-T2[11], photon counting[12], and radio astronomy[13]. In such applications, the coefficient tables are large in comparison with other elements of the engine [14]. In this contribution, we propose an innovative technique to reduce the resources required to implement a sine/cosine generator.

The rest of the document is organized as follows. In the next section, the notation used is introduced optimization techniques are presented to reduce the number of entries of the required LUT to a number proportional to the input space. In Section 4, optimization techniques are given enable LUTs to be employed with a total number of entries that grows sublinearly with the input space. The new proposed technique is introduced in Section 5, and experimental performance results are shown in Section 6. The last section provides a summary of the conclusions.

3 Argument Reduction

As mentioned in the introduction, our objective is to efficiently implement a device that provides $\sin(n\phi)$ and $\cos(n\phi)$, where n is an integer provided as an input to the device, and ϕ is a constant angle that depends on the application. Hereinafter, the input of the device will be denoted as I and the number of bits of I will be denoted as w. Furthermore, the following definitions are used:

Definition 1. *A real number ϕ is trigonometric-rational if and only if $\frac{\phi}{\pi}$ is rational.*

For example, the angle $\phi = -2\pi/L$, used in the definition of the twiddle factors in Section 2, is trigonometric-rational, while the angle $\phi = 2^{-79}$ mentioned in the arithmetic unit application is not.

Definition 2. *The trigonometric Carmichael function of a trigonometric-rational number ϕ is the minimum natural number $\ddot{\lambda}(\phi)$ such that $\frac{\ddot{\lambda}(\phi)\phi}{2\pi}$ is an integer.*

This definition is useful in the calculation of the size of the output space of the generator. This size is the minimum of $\ddot{\lambda}(\phi)$ and the size of the input space. Note that $\ddot{\lambda}(0) = 1$. In the DFT example, $\ddot{\lambda}(\phi)$ is equal to the length of the transform. The function $\ddot{\lambda}$ can also be employed to make the following simplification. Suppose that ϕ has been defined as a trigonometric-rational number whose absolute value is very large: $|\phi| \gg 2\pi$. In this case, an angle α with $|\alpha| < 2\pi$ can be found such that the functionality of the generator, that is, computing $\sin(n\phi)$ and $\cos(n\phi)$, is equivalent to computing $\sin(n\alpha)$ and $\cos(n\alpha)$. In order to obtain such α:

1. take the integer $k = \frac{\ddot{\lambda}(\phi)\phi}{2\pi}$
2. take the remainder r of the division $\frac{k}{\ddot{\lambda}(\phi)}$. Note that k and r have the same sign and $|r| < \ddot{\lambda}(\phi)$.
3. $\alpha = \frac{r2\pi}{\ddot{\lambda}(\phi)}$.

Definition 3. *The trigonometric Shannon entropy of a trigonometric-rational number ϕ is $\ddot{H}(\phi) = log_2(\ddot{\lambda}(\phi))$.*

If ϕ is trigonometric-rational and the size of the input space is as large as possible, that is, $\ddot{\lambda}(\phi)$, then the minimum number of bits required to code the input is $\lceil \ddot{H}(\phi) \rceil$.

Table 1: Values returned by the circuit specified in [4] when the input I has exactly three bits ($\phi = \pi/2^{3-1} = \pi/4$)

I	S	n	$x = S/4$	$\pi x = S\phi$	$n\phi$	$sen(\pi x)$ $= sen(S\phi)$ $= sen(n\phi)$	$\cos(\pi x)$ $= \cos(S\phi)$ $= \cos(n\phi)$
000	0	0	0	0	0	0	1
001	1	1	$1/4$	$\pi/4$	$\pi/4$	$1/\sqrt{2}$	$1/\sqrt{2}$
010	2	2	$2/4$	$2\pi/4$	$2\pi/4$	1	0
011	3	3	$3/4$	$3\pi/4$	$3\pi/4$	$1/\sqrt{2}$	$-1/\sqrt{2}$
100	-4	4	-1	$-4\pi/4$	$4\pi/4$	0	-1
101	-3	5	$-3/4$	$-3\pi/4$	$5\pi/4$	$-1/\sqrt{2}$	$-1/\sqrt{2}$
110	-2	6	$-2/4$	$-2\pi/4$	$6\pi/4$	-1	0
111	-1	7	$-1/4$	$-\pi/4$	$7\pi/4$	$-1/\sqrt{2}$	$1/\sqrt{2}$

Definition 4. *ϕ is trigonometric-binary if and only if it is trigonometric-rational and $\ddot{H}(\phi)$ is an integer.*

The latter definition is relevant since, in many applications, the constant angle ϕ is trigonometric-binary. For example, consider the algorithms designed to compute efficiently the DFT called *Fast Fourier Transform* (FFT) [15] algorithms: many of these algorithms require the length of the transform L to be a power of 2 [16], that is, $log_2(L)$ must be an integer, and hence ϕ must be trigonometric-binary since $\ddot{H}(\phi) = log_2(\ddot{\lambda}(-2\pi/L)) = log_2(L)$. Moreover, in applications where ϕ is trigonometric-binary, it is irrelevant whether the representation of n is either unsigned or two's complement as long as $w \geq \ddot{H}(\phi)$. As an example, consider the circuit specified in [4]. This circuit computes the sine and cosine of πx where x is a number in the interval $[-1, 1)$ coded in fixed-point two's complement. Let S be the value represented by the input I in integer two's complement, $x = S/2^{w-1}$, and hence $\pi x = S\pi/2^{w-1}$. Thus, the circuit computes the sine and cosine of $S\phi$, where $\phi = \pi/2^{w-1}$. Let n be the value represented by I in unsigned integer representation. It is easy to prove that $\sin(S\phi) = \sin(n\phi)$ and $\cos(S\phi) = \cos(n\phi)$. Therefore, the functionality of the circuit is equivalent to the computation of the sine and cosine of $n\phi$. This is exemplified in Table 1 for $w = 3$.

Hereinafter, an unsigned notation for n is assumed. In the following subsections, we will see optimization techniques that require a trigonometric-rational value of ϕ. In these subsections, the trigonometric Carmichael function of ϕ is abbreviated to L.

3.1 Periodicity

If the size of the input space of the device is greater than L, then the periodicity of the sine and cosine can be used to compute $\sin(n\phi)$ and $\cos(n\phi)$ in the following way:

1. Compute $n \bmod L$. As noted by [4], if ϕ is trigonometric-binary, then this computation has no cost since the result is simply the $\ddot{H}(\phi)$ least significant bits of the input I.
2. Use a subgenerator to compute the sine and cosine of $(n \bmod L)\phi$. The input space of the subgenerator is $\mathbb{Z}_L$, smaller than the original input space, and hence it can be implemented using a smaller LUT.

In the optimization shown in the following subsections, it is assumed that the input space of the generator is $\mathbb{Z}_L$.

3.2 Sign Reduction

If the input space of the device is $\mathbb{Z}_L$, then it is possible to implement it with another subgenerator with the same value of ϕ but whose input space is $\mathbb{Z}_{\lfloor L/2 \rfloor + 1}$, that is, its size is roughly half of the size of the original input space. This optimization takes into account the following trigonometric identities:

$$\begin{aligned} \sin(\alpha) &= -\sin(2\pi - \alpha) \\ \cos(\alpha) &= \cos(2\pi - \alpha) \end{aligned} \tag{1}$$

If $n \leq L/2$, then the input of the subgenerator is n and its output is the output of the device. Otherwise, the input of the subgenerator is $L - n$, the cosine output of the device is the cosine output of the subgenerator, and the sine output of the device is the opposite of the sine output of the subgenerator. Note that, if ϕ is trigonometric-binary, then $L - n$ can be obtained by simply taking the two's complement. In the next subsection, another optimization is presented for the implementation of the subgenerator when L is even.

3.3 Quadrant Reduction

If L is even and the size of the input space of the device is $\lfloor L/2 \rfloor + 1 = L/2 + 1$, then it can be implemented using a subgenerator with the same value of ϕ but whose input space is reduced to $\mathbb{Z}_{\lfloor L/4 \rfloor + 1}$. This optimization uses the following trigonometric identities:

$$\begin{aligned} \sin(\alpha) &= \sin(\pi - \alpha) \\ \cos(\alpha) &= -\cos(\pi - \alpha) \end{aligned} \tag{2}$$

If $n \leq L/4$, then the input of the subgenerator is n and its output is the output of the device. Otherwise, the input of the subgenerator is $L/2 - n$, the sine output of the device is the sine output of the subgenerator, and the cosine output of the device is the opposite of the cosine output of the subgenerator. Again, the computation of $L/2 - n$ is simply a two's complement if ϕ is trigonometric-binary. In turn, the optimization described in the next subsection can be used to implement the subgenerator if L is multiple of 4.

3.4 Octant Reduction

Finally, if L is a multiple of 4 and the input space of the device is $\mathbb{Z}_{\lfloor L/4 \rfloor +1} = \mathbb{Z}_{L/4+1}$, then it can be implemented with a subgenerator with the same value of ϕ but whose input space is reduced to $\mathbb{Z}_{\lfloor L/8 \rfloor +1}$ by applying the following trigonometric identities:

$$\begin{aligned} \sin(\alpha) &= \cos(\pi/2 - \alpha) \\ \cos(\alpha) &= \sin(\pi/2 - \alpha) \end{aligned} \tag{3}$$

If $n \leq L/8$, then the input of the subgenerator is n and its output is the output of the device. Otherwise, the input of the subgenerator is $L/4 - n$, the sine output of the device is the cosine output of the subgenerator and the sine output of the device is the cosine output of the subgenerator. Once more, a simple two's complement provides $L/4 - n$ if ϕ is trigonometric-binary.

With the previous optimizations, a sine/cosine generator can be implemented with an LUT of $\lfloor L/8 \rfloor + 1$ entries. In order to exemplify, the circuit described in [4] is implemented using direct access memory as an LUT. As previously discussed, the functionality of the circuit is equivalent to computing the sine and cosine of $n\phi$, where n is the number provided by input I in unsigned integer notation, the angle ϕ is $2\pi/2^w$, and w is the number of bits of I. In this case, ϕ is trigonometric-binary. The output is provided in some type of sign-magnitude notation, such as one of the IEEE 754-1985 floating-point formats. We will also assume that $w > 3$ so $\ddot{\lambda}(\phi)$ is multiple of 4 and an LUT of only $2^{w-3} + 1$ entries is required. The implementation is shown in Figure 1. The LUT should return the sine and cosine of angles in the range $[0, \pi/4]$ and, since they are all positive, there is no need to store the sign bits. Instead, the sign is computed using a simple XOR gate *(4c)*. In order to prevent the problem of dealing with a direct access memory with a number of positions that is not a power of two, the access of the entry of the LUT corresponding to $n = 2^{w-3}$ is detected by a simple logic gate *(4a)* and is treated separately. In this case the LUT returns the sine and cosine of $\pi/4$, that is, $1/\sqrt{2}$, using a pair of multiplexers *(2b)*. The address lines of the memory are fed with the $w - 3$ least significant bits of I or with its two's complement depending on I_{w-3}, using the adder *(3)* and the multiplexer *(2a)*. The gate *(4b)* is employed to ascertain whether the magnitude of the sine and the cosine should be interchanged with the multiplexers *(2c)*.

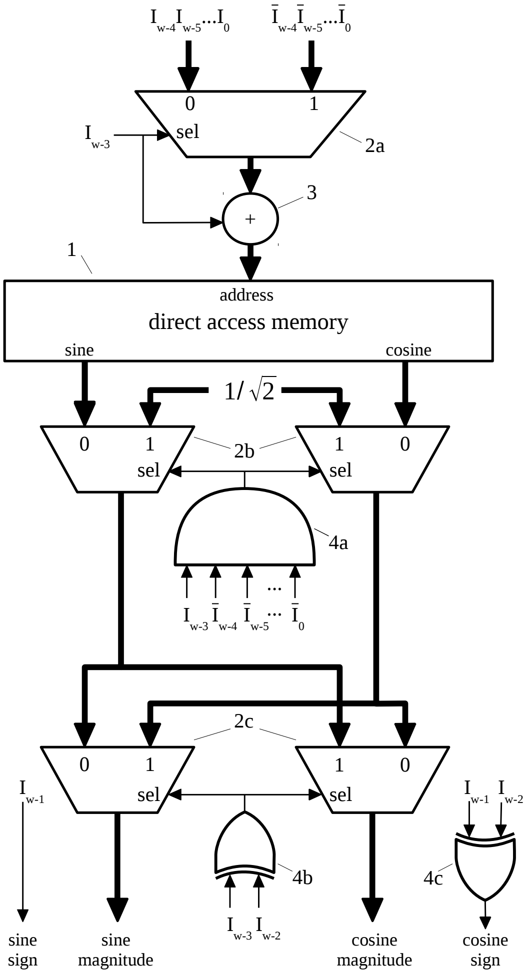


Figure 1: Argument reduction employed to optimize a sine/cosine generator.

4 Sublinear Optimizations

The optimizations described in the previous sections have the following drawbacks:

- They require a subgenerator with an input space greater than $\ddot{\lambda}(\phi)/8$, that is, its input space grows linearly with $\ddot{\lambda}(\phi)$. In many applications, the subgenerator cannot be directly implemented using an LUT with a number of entries proportional to the input space since it is excessively large. For example, even if the octant optimization could be directly applied to the arithmetic unit mentioned in Section 2, it would only reduce the size of the input space of the required subgenerator to roughly $30 * 2^{49}$.

- They can only be directly applied if ϕ is trigonometric-rational. Furthermore, the quadrant and octant optimizations require $\ddot{\lambda}(\phi)$ to be even and a multiple of 4, respectively. Hence, in order to apply them in the arithmetic unit of the previous example, a workaround similar to that shown in [4,17] is necessary. For example, assuming the angle is positive, the arithmetic unit could execute the following steps to compute the sine and cosine:

1. Divide the angle by 2π.
2. Take the fixed-point representation I of the fractional part of the previous division.
3. Return the sine and cosine of $n\phi$, where n is the number represented by I in unsigned integer notation, $\phi = \frac{2\pi}{2^w}$, and w is the width of I.

This last step can be carried out by a generator that can be implemented using the optimizations described in the previous section. However, this approach has its own drawbacks: first, the cost of a division is introduced; second, if the angle is not zero, then the result of the division is not rational and hence its representation cannot be exact and an error is introduced [17].

In the following subsections, optimizations without these drawbacks are described.

4.1 Branching

This optimization, used in [4], accepts an arbitrary value of ϕ, although it was originally employed for a trigonometric value. When branching is applied, the generator is implemented using two subgenerators, M_0 and M_1, which we call *branches*. The inputs of the branches are denoted by $A(0)$ and $A(1)$, while the widths of these inputs are denoted by $L(0)$ and $L(1)$, respectively. These are chosen so that the width of the input of the generator, I, is $w = L(0) + L(1)$. M_0 provides the sine and cosine of integer multiples of ϕ, that is, the sine and cosine of $n_0\phi_0$, where n_0 is the value represented by $A(0)$ and $\phi_0 = \phi$. On the other hand, M_1 provides the sine and cosine of $n_1\phi_1$, where n_1 is the value represented by $A(1)$ and $\phi_1 = 2^{L(0)}\phi$. The least significant bits of I are connected to $A(0)$, while the rest are connected to $A(1)$. Since I is the concatenation of $A(0)$ and $A(1)$, the value represented by I is

$$n = n_0 + n_1 2^{L(0)} \tag{4}$$

and hence,

$$n\phi = n_0\phi + n_1 2^{L(0)}\phi = n_0\phi_0 + n_1\phi_1 \tag{5}$$

Since the sines and cosines of $n_0\phi_0$ and $n_1\phi_1$ are provided by M_0 and M_1, the sine and cosine of their sum can be computed by applying the following trigonometric identities:

$$\begin{aligned} \sin(A+B) &= \sin(A)\cos(B) + \cos(A)\sin(B) \\ \cos(A+B) &= \cos(A)\cos(B) - \sin(A)\sin(B) \end{aligned} \tag{6}$$

Alternatively, we can say that each subgenerator M_k provides the complex $sen(n_k\phi_k)i + cos(n_k\phi_k) = e^{n_k\phi_k i}$, and the generator can provide the value $e^{n\phi i}$ by computing the complex product $e^{n_0 \phi_0 i} e^{n_1 \phi_1 i}$. Indeed, computing the product of two complexes, each of a unitary module, is equivalent to computing the sine and cosine of the sum of two angles from the sine and cosine of those angles and implies four real products, a real sum, and a real subtraction. A generalization of this branching technique was proposed in [18] to compute twiddle factors. Note that the sum of the sizes of the input spaces of M_0 and M_1 is minimum when $L(0)$ and $L(1)$ differ by no more than 1. In this case, such a sum grows with the square root of the size of the original input space, that is, sublinearly [18].

4.2 Tree Generator

The implementation of the branches was not detailed in the previous subsection. In the generator described in [4], the branch M_0 computes its output using the Taylor series, while M_1 is implemented with an LUT of affordable size. Further optimization could be achieved if one or both branches were, in turn, implemented with sub-branches. This recursive application of the branch optimization is used by the tree generator described in [1]. In general, the tree generator requires: a set of subgenerators that we will call *leaves*; complex multipliers; and, if the implementation is sequential or pipelined, registers. The following notation is employed for its description:

- w: width of the input of the tree generator
- $I = I_{w-1} I_{w-2} \dots I_1 I_0$: input of the tree generator
- $n = \sum_{t=0}^{w-1} I_t 2^t$: number represented by the input of the tree generator
- m: number of leaf subgenerators employed
- M_0,M_1,…,M_{m-1}: the m leaves
- $L(k)$: width of the input of the leaf M_k
- $A(k) = A(k)_{L(k)-1} \dots A(k)_0$: input of the leaf M_k
- $n_k = \sum_{t=0}^{L(k)-1} A(k)_t 2^t$: number represented by the input A(k)
- $SL(k) = \sum_{t=0}^{k-1} L(t) = \begin{cases} 0 \text{ if } k = 0 \\ L(k-1) + SL(k-1) \text{ if } k > 0 \end{cases}$: total number of input lines of the leaves with index lower than k
- ϕ_k: angle defined by $\phi_k = (2^{SL(k)})\phi$

Each leaf subgenerator M_k provides the sine and cosine of $n_k \phi_k$. The leaves are chosen such that the sum of the widths of their inputs is equal to the width of the input of the tree generator:

$$w = SL(m) = \sum_{k=0}^{m-1} L(k) \tag{7}$$

The input lines of each leaf M_k are connected to the input lines of the tree generator from $I_{SL(k)}$ to $I_{SL(k+1)-1}$, that is, each input line $A(k)_t$ is connected to $I_{t+SL(k)}$:

$$A(0) = I_{L(0)-1} \dots I_1 I_0$$

$$A(1) = I_{L(0)+L(1)-1} \dots I_{L(0)+1} I_{L(0)}$$

$$\vdots$$

$$A(m-1) = I_{w-1} \dots I_{SL(m-1)+1} I_{SL(m-1)}$$

Hence, the input value n represented by I becomes:

$$\begin{aligned} n = \sum_{t=0}^{w-1} I_t 2^t = \sum_{k=0}^{m-1} \sum_{t=SL(k)}^{SL(k)+L(k)-1} I_t 2^t = \sum_{k=0}^{m-1} \sum_{t=0}^{L(k)-1} I_{t+SL(k)} 2^{t+SL(k)} = \\ \sum_{k=0}^{m-1} \sum_{t=0}^{L(k)-1} A(k)_t 2^{t+SL(k)} = \sum_{k=0}^{m-1} (\sum_{t=0}^{L(k)-1} A(k)_t 2^t) 2^{SL(k)} = \sum_{k=0}^{m-1} n_k 2^{SL(k)} \end{aligned} \tag{8}$$

and therefore the angle whose sine and cosine must be computed by the tree generator can be written as:

$$n\phi = \sum_{k=0}^{m-1} n_k 2^{SL(k)} \phi = \sum_{k=0}^{m-1} n_k \phi_k \tag{9}$$

Hence, the angle $n\phi$ is the sum of the subangles $n_k\phi_k$ or, alternatively, $e^{n\phi i}$ is the product $e^{n_0\phi_0 i} e^{n_1\phi_1 i} \dots e^{n_{m-1}\phi_{m-1} i}$. Again, since the sine and cosine of the subangles are provided by the leaves, the sine and cosine of $n\phi$ can be computed with complex multiplications. Taking this into account, the structure of the generator described in [1] becomes a directed rooted binary tree with m leaves. Each vertex corresponds to a component whose output is a complex of unitary module. Each internal vertex has exactly two children, and corresponds to a complex multiplier that computes the product of the outputs of the components associated to these children. The components corresponding to the leaves are the m subgenerators and provide the complex values $e^{n_k\phi_k i}$. The output of the tree generator is the output of the component corresponding to the root vertex. Hereinafter, the height of the tree will be denoted as h. The following recommendations may improve the efficiency of the design:

- It is desirable to minimize the height of the tree h in order to reduce latency and rounding errors. This is achieved if the structure of the generator is a complete binary tree.

- If each leaf is implemented with an LUT, the total number of entries is minimum when the width of the inputs of those LUTs differ by no more than 1. To this end, let q be the quotient obtained by dividing w by m, and let r be the remainder. A total of r LUTs must have inputs of width $q+1$. The other LUTs must have inputs of width q.

- If the above recommendation is followed, then the total number of entries decreases when m increases. For a fixed height h, the maximum possible value of m is 2^h, and therefore the total number of entries can be minimized by using 2^h leaves.

In order to ascertain the power of this approach, suppose we use a complete binary tree with height $h = \lfloor log_2(w) \rfloor$. In this case, the number of subgenerators m would be no greater than w, and each subgenerator would have no more than 2 input lines. If each subgenerator is implemented with an LUT, then an upper bound on the total number of entries is $4w$, that is, the total number of entries grows logarithmically with the size of the input space of the tree generator. Note that a tree generator can be combined with the argument reduction mentioned in Section 3. For example, argument reduction is first applied in [4] and hence only a subgenerator with an input space of roughly 1/8 of the original size is required. The subgenerator is then implemented with a tree generator of height $h = 1$.

In the following sub-subsections we will see several optimizations that can be applied to the tree generator. In the rest of the document $\phi > 0$ is assumed for the sake of simplicity, although in practice this is not a restriction since $\cos(n\phi) = \cos(n|\phi|)$ and $\sin(n\phi) = \text{sgn}(\phi) * \sin(n|\phi|)$.

4.2.1 Quadrant Restriction

This optimization can be applied to *quadrant-restricted* sine/cosine generators, which are defined as follows:

Definition 5. *Given a device with an integer input $n \geq 0$ that computes one or more trigonometric functions of $n\phi$, where $\phi > 0$ is a constant, the device is quadrant-restricted if and only if $\frac{\pi}{2\phi}$ is an upper bound on its input space.*

If a sine/cosine generator is quadrant-restricted then it must compute the sine and cosine of an angle $n\phi$ in the interval $[0, \pi/2]$. Since both functions are positive in that interval, the following optimizations are possible:

- As in the example of Section 3, if the generator is implemented with an LUT there is no need to store the sign bits.

- If it is implemented with a tree generator, no signed adders, subtracters, nor multipliers are required.

For example, the tree generator used in [4] is quadrant-restricted, and hence the complex multiplier requires no signed arithmetic components and the LUT employed to implement the branch M_1 does not need to store the sign bits. Note that, even if a tree generator is not quadrant-restricted, it may contain quadrant-restricted branches that can benefit from these optimizations.

4.2.2 Leading Zeros of the Sine

This optimization is useful when the sine values of a quadrant-restricted generator are coded in fixed-point. In this case, an upper bound on the sine output is $\sin(nmax\phi)$, where $nmax$ is the maximum of the input space. Consequently, if the k most significant bits of the fixed-point representation of

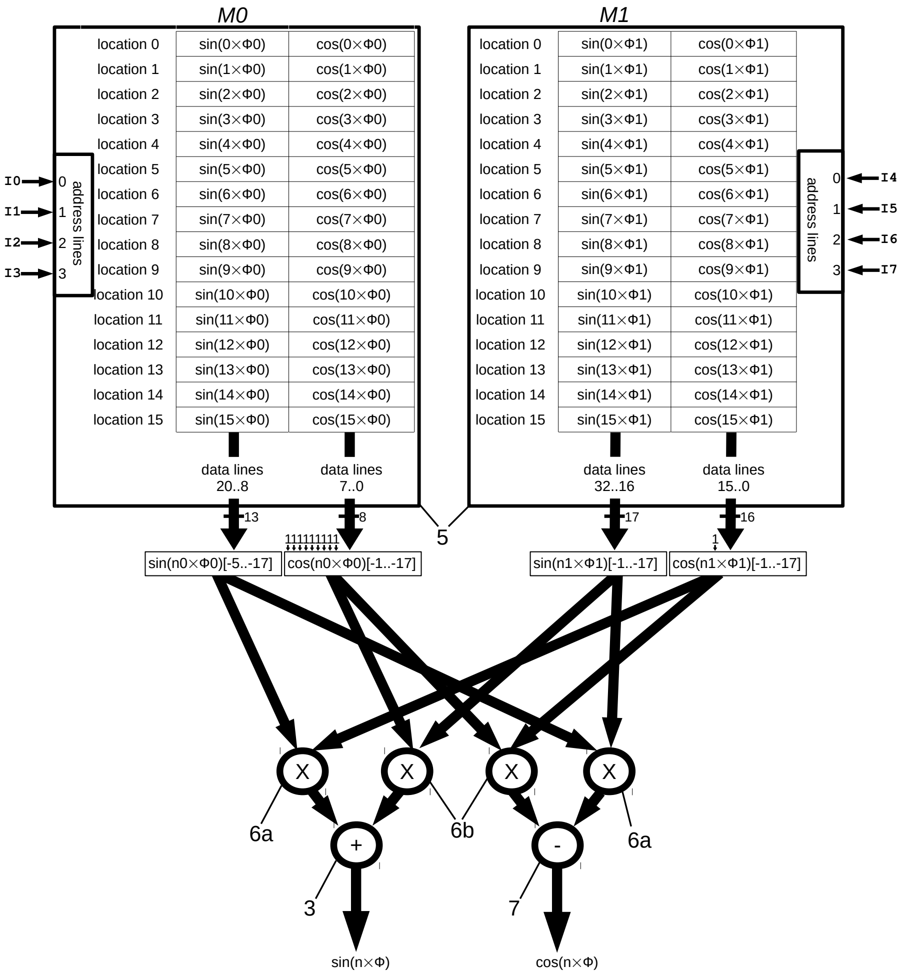
$\sin(nmax\phi)$ are 0, those bits of the sine output of the generator are always 0, and the following optimizations are possible:

- If the generator is implemented with an LUT, then there is no need to store the k most significant bits of the sine.
- If the generator feeds a complex multiplier of a tree generator, the size of its real multipliers can be reduced.

4.2.3 Leading Ones of the Cosine

This optimization is useful when the cosine values of a quadrant-restricted generator are coded in fixed-point using all the bits for the fractional part. In this case, the representable value nearest to $\cos(0) = 1$ corresponds to the word with all the bits equal to 1. A lower bound on the cosine output is $\cos(nmax\phi)$, where $nmax$ is the maximum of the input space, and therefore, if the k most significant bits of the fixed-point representation of $\cos(nmax\phi)$ are 1, those bits of the cosine output of the generator are always 1. Hence, if the generator is implemented with an LUT, there is no need to store the k most significant bits of the cosine.

To exemplify these optimizations, suppose a generator must provide the sine and cosine of $n\phi$ in fixed-point notation with 8 fractional bits and no integer bits rounding to the nearest representable value. In this example $\phi = 2\pi/2^{11} = \pi/2^{10}$, that is, it is trigonometric-binary and $\ddot{\lambda}(\phi)$ is a multiple of 4, and hence we first apply argument reduction and treat the case $n = 2^8$ separately as in the example of Section 3. A subgenerator still has to be subsequently implemented with an input space of size 2^8 ($w = \ddot{H}(\phi) - 3 = 8$). We implement this subgenerator with a tree generator of height $h = 1$, and therefore there are 2 leaves ($m = 2^h = 2$) as shown in Figure 2. Since this tree generator is quadrant-restricted it does not require signed arithmetic components. Each leaf is implemented with direct access memory *(5)* of height 2^4. Note that the sine/cosine values must be stored with a precision of 17 bits to compensate for rounding errors. The leaf M_0 provides the sines and cosines of the multiples of $\phi_0 = 2^0\phi = \pi/2^{10}$, while the leaf M_1 provides the sines and cosines of the multiples of $\phi_1 = 2^4\phi = \pi/2^6$. The greatest angle whose sine and cosine is stored in M_0 is $15\pi/2^{10}$. The 4 most significant bits of the representation of the sine of this angle are 0 and, since the generator is quadrant-restricted, the 4 most significant bits of the other representations of the sines stored in M_0 are also 0, and therefore there is no need for them to be stored. On the other hand, the 9 most significant bits of the representation of the cosine of that angle are equal to 1, and therefore there is no need for them to be stored either. Similar optimizations can be applied to M_1, but in this case only one bit can be saved. Note that we only need two integer multipliers of size 13×17 *(6a)* and two of size 17×17 *(6b)* instead of four multipliers of size 17×17 thanks to the leading zeros of the sine. The leading ones of the cosine cannot be employed to reduce the size the arithmetic components in a similar way. In the last stage, an adder *(3)* provides the sine of the tree generator and a subtracter *(7)* provides the cosine.



M0

location 0	sin(0×Φ0)	cos(0×Φ0)
location 1	sin(1×Φ0)	cos(1×Φ0)
location 2	sin(2×Φ0)	cos(2×Φ0)
location 3	sin(3×Φ0)	cos(3×Φ0)
location 4	sin(4×Φ0)	cos(4×Φ0)
location 5	sin(5×Φ0)	cos(5×Φ0)
location 6	sin(6×Φ0)	cos(6×Φ0)
location 7	sin(7×Φ0)	cos(7×Φ0)
location 8	sin(8×Φ0)	cos(8×Φ0)
location 9	sin(9×Φ0)	cos(9×Φ0)
location 10	sin(10×Φ0)	cos(10×Φ0)
location 11	sin(11×Φ0)	cos(11×Φ0)
location 12	sin(12×Φ0)	cos(12×Φ0)
location 13	sin(13×Φ0)	cos(13×Φ0)
location 14	sin(14×Φ0)	cos(14×Φ0)
location 15	sin(15×Φ0)	cos(15×Φ0)

M1

location 0	sin(0×Φ1)	cos(0×Φ1)
location 1	sin(1×Φ1)	cos(1×Φ1)
location 2	sin(2×Φ1)	cos(2×Φ1)
location 3	sin(3×Φ1)	cos(3×Φ1)
location 4	sin(4×Φ1)	cos(4×Φ1)
location 5	sin(5×Φ1)	cos(5×Φ1)
location 6	sin(6×Φ1)	cos(6×Φ1)
location 7	sin(7×Φ1)	cos(7×Φ1)
location 8	sin(8×Φ1)	cos(8×Φ1)
location 9	sin(9×Φ1)	cos(9×Φ1)
location 10	sin(10×Φ1)	cos(10×Φ1)
location 11	sin(11×Φ1)	cos(11×Φ1)
location 12	sin(12×Φ1)	cos(12×Φ1)
location 13	sin(13×Φ1)	cos(13×Φ1)
location 14	sin(14×Φ1)	cos(14×Φ1)
location 15	sin(15×Φ1)	cos(15×Φ1)

Figure 2: Optimization of a quadrant-restricted tree generator of height 1.

5 Complement Generator

In the optimizations described in Section 4, the angle whose sine/cosine must be computed is decomposed into two subangles, A and B. Two subgenerators, called branches, are employed to compute the sine/cosine of A and B, and then the sine/cosine of $A + B$ is computed by applying the identities 6. This method presents the following drawbacks:

- If the maximum value of one of the angles A or B is small, then its sine is close to 0, while its cosine is close to 1. In this case, the product $\cos(A)\cos(B)$ can be orders of magnitude greater than $\sin(A)\sin(B)$, and hence smearing may occur when computing $\cos(A + B) = \cos(A)\cos(B) - \sin(A)\sin(B)$.

- Unlike the optimization described in sub-subsection 4.2.2, the one described in 4.2.3 fails to help in the reduction of the required arithmetic components.

These problems can be solved by using a *sine/complement generator*. In the same way as a sine/cosine generator, a sine/complement generator receives an integer n and computes trigonometric functions of $n\phi$, where ϕ is a constant. The only difference lies in it computing the *complement of the cosine* of $n\phi$ instead of its cosine. This is defined as follows:

Definition 6. *The complement of the cosine of x is* $\text{com}(x) = 1 - \cos(x)$

It is possible to compute the sine and the complement of the cosine of the sum of two angles, A and B, from the sines and complements of the cosines of those angles by using a device called a *trigonometric adder* [2]. Similar to the complex multiplier, the trigonometric adder can be implemented with adders *(3)*, subtracters *(7)*, and multipliers *(6)* as depicted in Figure 3. This trigonometric adder implementation uses the following trigonometric identities derived from those of 6:

$$\begin{aligned} \sin(A+B) &= \sin(A) + \sin(B) - [\sin(A)\,\text{com}(B) + \text{com}(A)\sin(B)] \\ \text{com}(A+B) &= \text{com}(A) + \text{com}(B) + [\sin(A)\sin(B) - \text{com}(A)\,\text{com}(B)] \end{aligned} \tag{10}$$

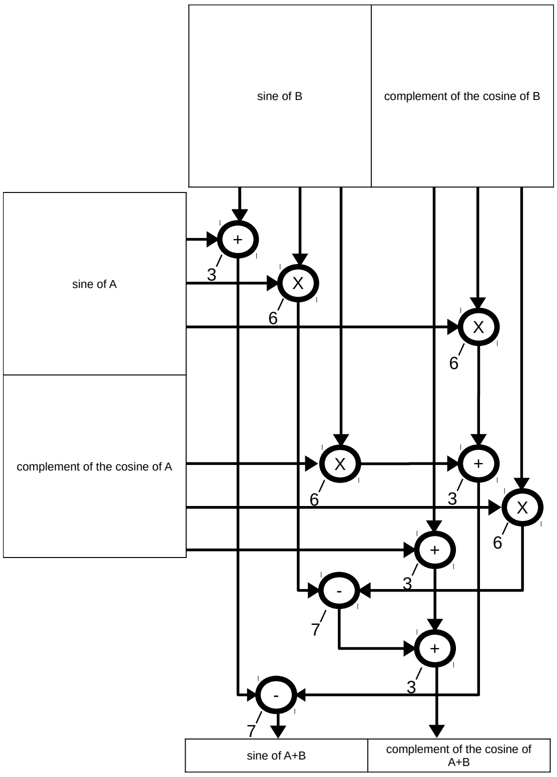

sine of B
complement of the cosine of B
sine of A
complement of the cosine of A
+
X
X
X
+
X
+
-
+
-
3
6
6
6
3
6
3
7
3
7
sine of A+B
complement of the cosine of A+B

Figure 3: Implementation of a trigonometric adder.

Trigonometric adders enable the implementation of a sine/complement generator using a tree structure similar to that described in subsection 4.2. Such an implementation, described in [3], requires a set of sine/complement subgenerators (the leaves of the tree) as well as trigonometric adders (the internal vertex). Furthermore, if the generator is quadrant-restricted, then optimizations similar to those described in subsection 4.2 can be applied:

- Since the complement of the cosine is also positive in $[0, \pi/2]$, the trigonometric adders can be implemented without signed arithmetic components.

- If fixed-point representation is used, the leading zeros of the sines make it possible to reduce the size of the integer multipliers and that of the leaves.

In this case, the optimization described in sub-subsection 4.2.3 cannot be applied but, if fixed-point representation is used, we can use the following optimization that we call *leading zeros of the complement*: if a branch is quadrant-restricted, an upper bound on its complement output is $\mathrm{com}(nmax\phi)$, where $nmax$ is the maximum of the input space of the branch. Therefore, if the k most significant bits of the fixed-point representation of $\mathrm{com}(nmax\phi)$ are 0, those bits of the complement output of the branch are always 0. If the branch is implemented with an LUT, then there is no need to store those bits. Note that, unlike the optimization described in 4.2.3, this optimization enables a reduction of the size of the required multipliers.
A sine/cosine generator can be implemented with a sine/complement generator by simply adding a trivial arithmetic circuit to subtract the complement of the cosine from 1. In fact, if fixed-point notation is used, then this arithmetic circuit is not necessary since the trigonometric adder corresponding to the root can be easily modified to provide $\cos(n\phi)$ instead of $\mathrm{com}(n\phi)$ at no additional cost. To this end, instead of $\mathrm{com}(n\phi)$, the root vertex computes its opposite using the following equation:

$$-\mathrm{com}(A+B) = [\mathrm{com}(A)\,\mathrm{com}(B) - \sin(A)\sin(B)] - [\mathrm{com}(A) + \mathrm{com}(B)] \tag{11}$$

subsequently 1 can be added to $-\mathrm{com}(n\phi)$ in order to obtain $\cos(n\phi)$. Note that this last operation is merely toggling the integer bit of the representation of $-\mathrm{com}(n\phi)$. If a sine/cosine generator or a branch of it is quadrant-restricted, then it should be implemented employing a complement generator due to the following reasons:

- The smearing problems are lessened by using the formulae 10.

- In contrast to the leading ones of the cosine, the leading zeros of the complement make it possible to reduce the size of the multipliers.

As an example, Figure 3 shows how to implement a sine/cosine generator with an input I of width $w = 11$ using a sine/complement generator whose topology is a tree of height $h = 2$. The sine/complement generator uses $m = 2^h = 4$ subgenerators, M_0, M_1, M_2 and M_3, which have been implemented with direct access memories *(8)*. Following the recommendations of subsection 4.2 to minimize the total number of memory locations, M_3 has 2 address lines ($q = \lfloor w/m \rfloor = 2$) and each of the other 3 remaining memories ($r = w - mq = 3$)

has an additional address line, and therefore $L(3) = 2$ and $L(2) = L(1) = L(0) = 3$. Hence, $SL(0) = 0$, $\phi_0 = 2^0\phi$, $SL(1) = 3$, $\phi_1 = 2^3\phi$, $SL(2) = 6$, $\phi_2 = 2^6\phi$, $SL(3) = 9$, and $\phi_3 = 2^9\phi$. Each memory M_k contains the sines and the complement of the cosines of the multiples of $\phi_k = (2^{SL(k)})\phi$, and therefore its output provides the sine and the complement of the cosine of $n_k\phi_k$, where n_k is the value of its address lines. Each address line t of each memory M_k is connected to $I_{t+SL(k)}$, that is, the inputs of M_0, M_1, M_2 and M_3 are connected to $I_2I_1I_0$, $I_5I_4I_3$, $I_8I_7I_6$, and $I_{10}I_9$, respectively. Hence, $n = n_02^{SL(0)} + n_12^{SL(1)} + n_22^{SL(2)} + n_32^{SL(3)} \implies n\phi = n_02^{SL(0)}\phi + n_12^{SL(1)}\phi + n_22^{SL(2)}\phi + n_32^{SL(3)}\phi = n_0\phi_0 + n_1\phi_1 + n_2\phi_2 + n_3\phi_3$. Three trigonometric adders are used *(9)*. Those connected directly to the memories are employed to compute the sine and the complement of the cosine of the angles $n_0\phi_0 + n_1\phi_1$ and $n_2\phi_2 + n_3\phi_3$. The other adders computes the sine and the complement of the cosine of $n\phi = n_0\phi_0 + n_1\phi_1 + n_2\phi_2 + n_3\phi_3$. A trivial arithmetic circuit *(10)* subtracts the complement of the cosine of $n\phi$ from 1 to obtain the cosine of $n\phi$.

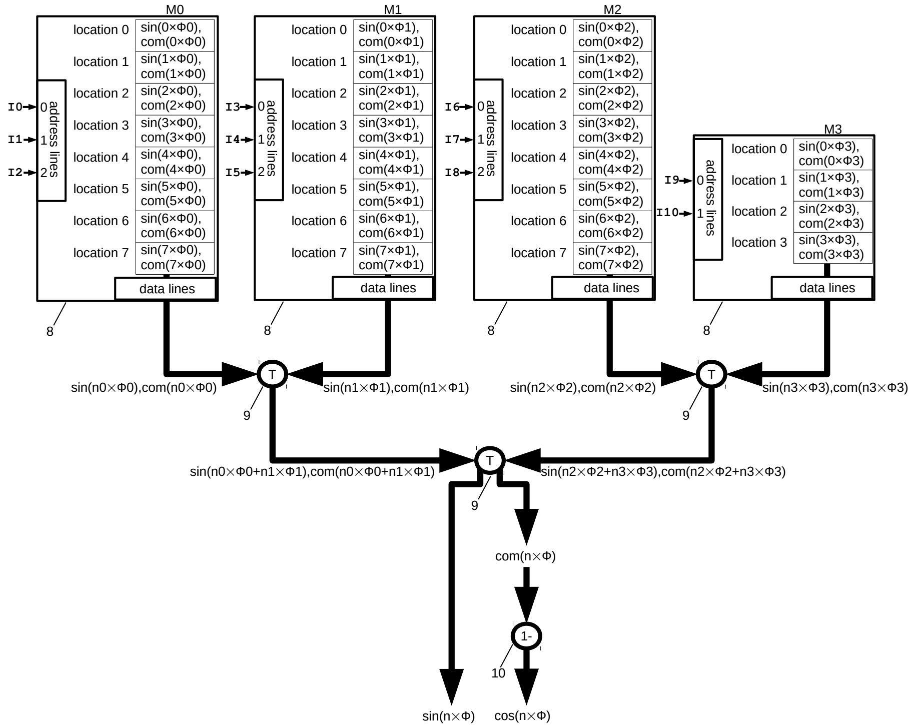


Figure 4: sine/cosine generator implemented with a sine/complement generator.

6 Results

In order to measure the possible enhancements that complement generators may provide, several twiddle factor generators are implemented in a field-programmable gate array (FPGA) chip. They provide an output rounded

Table 2: DSP usage of the twiddle factor generators with a tree structure of height 1

DFT length	sine/cosine DSPs	sine/complement DSPs	saving
2^{15}	12	7	41.7%
2^{16}	12	8	33.3%
2^{17}	12	8	33.3%
2^{18}	16	12	25.0%
2^{19}	16	12	25.0%
2^{20}	16	12	25.0%

Table 3: DSP usage of the twiddle factor generators with a tree structure of height 2

DFT length	sine/cosine DSPs	sine/complement DSPs	saving
2^{21}	59	33	44.1%
2^{22}	71	47	33.8%
2^{23}	77	51	33.8%
2^{24}	76	51	32.9%
2^{25}	77	53	31.2%
2^{26}	79	53	32.9%

to the nearest representable value in fixed-point with 16 fractional bits and no integer bits. All the considered sequence lengths are powers of 2 and it is therefore possible to apply argument reduction and only quadrant-restricted generators are needed. The sequence lengths range from 2^{15} to 2^{26}. The generators corresponding to lengths in the range $[2^{15}, 2^{20}]$ are implemented using trees of height 1, while those corresponding to lengths in the range $[2^{21}, 2^{26}]$ are implemented with trees of height 2. The recommendations of subsection 4.2 are followed to minimize the size of the leaves. The FPGA chip used is a Xilinx Virtex 7 XC7VX485T-2FFG1761. Synthesis is carried out with the Vivado Design Suite tool of Xilinx version 2017.2.1 using the default options. The leaves are implemented directly with LUTs, while the multipliers are implemented with the digital signal processing (DSP) blocks of the FPGA. As stated earlier, the complement implementation enables a reduction of the size of the required multipliers. Hence, the number of DSP blocks required is remarkably reduced in the complement implementations. The reduction ranges from 25% to 41.7% in the trees of height 1 and from 31.2% to 44.1% in the trees of height 2 as shown in Table 2 and Table 3 respectively.

7 Conclusions

In this document, we propose a sine/cosine computation technique. In the proposed technique, the complement of the cosine is computed before the cosine itself in order to reduce the size of the required multiplications. Several twiddle factor generators are implemented in a Xilinx Virtex 7 XC7VX485T-2FFG1761

FPGA chip using this technique and the traditional technique. The proposed technique enables a reduction of the number of required DSP blocks by between 25% and 44%. This remarkable saving must be taken into account when designing sine/cosine generators.

The following abbreviations are used in this document:

ASIC	Application-Specific Integrated Circuit
DCT	Discrete Cosine Transform
DSP	Digital Signal Processing
DFT	Discrete Fourier Transform
DST	Discrete Sine Transform
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
IDCT	Inverse Discrete Cosine Transform
IDST	Inverse Discrete Sine Transform
LUT	Look Up Table

References

[1] Guerrero, D.; Viejo, J.; Ruiz-de Clavijo, P.; Juan, J.; Bellido, M.J.; Millan, A.; Ostua, E.; Villar, J.I.; Quiros, J.; Muñoz, A. Digital electronic circuit for calculating sines and cosines of multiples of an angle. WO2018104566A1, 2018.

[2] Guerrero, D.; Millan, A.; Juan, J.; Viejo, J.; Bellido, M.J.; Ruiz-de Clavijo, P.; Ostua, E. Dispositivo electrónico calculador de funciones trigonométricas. P201831134, 2019.

[3] Guerrero, D.; Millan, A.; Juan, J.; Viejo, J.; Bellido, M.J.; Ruiz-de Clavijo, P.; Ostua, E. Dispositivo electrónico calculador de funciones trigonométricas y usos del mismo. P201831133, 2019.

[4] de Dinechin, F.; Istoan, M.; Sergent, G. Fixed-point Trigonometric Functions on FPGAs. *SIGARCH Comput. Archit. News* **2014**, *41*, 83–88. doi:10.1145/2641361.2641375.

[5] Lin, K.; Hou, C. Implementation of trigonometric custom functions hardware on embedded processor. Proceedings of the IEEE 2nd Global Conference on Consumer Electronics (GCCE 2013);, 2013; pp. 155–157. doi:10.1109/GCCE.2013.6664782.

[6] Huang, H.; Xiao, L.; Liu, J. CORDIC-Based Unified Architectures for Computation of DCT / IDCT / DST / IDST. *Circuits, Systems, and Signal Processing* **2014**, *33*, 799–814. doi:10.1007/s00034-013-9661-9.

[7] Goldberg, D. What Every Computer Scientist Should Know About Floating-point Arithmetic. *ACM Comput. Surv.* **1991**, *23*, 5–48. doi:10.1145/103162.103163.

[8] Lefevre, V.; Muller, J. Worst cases for correct rounding of the elementary functions in double precision. Proceedings of the 15th

IEEE Symposium on Computer Arithmetic. ARITH-15 2001; , 2001; pp. 111–118. doi:10.1109/ARITH.2001.930110.

[9] Kulshreshtha, T.; Dhar, A.S. CORDIC-Based High Throughput Sliding DFT Architecture with Reduced Error-Accumulation. *Circuits, Systems, and Signal Processing* **2018**, *37*, 5101–5126. doi:10.1007/s00034-018-0810-z.

[10] IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications. https://standards.ieee.org/content/ieee-standards/en/standard/1901-2010.html, accessed on 15 April 2019.

[11] Lin, S.Y.; Wey, C.L.; Shieh, M.D. Low-cost FFT processor for DVB-T2 applications. *IEEE Transactions on Consumer Electronics* **2010**, *56*, 2072–2079. doi:10.1109/TCE.2010.5681074.

[12] Stanton, R.H. Photon Counting - One More Time. Proceedings of the 31st Annual SAS Symposium on Telescope Science; , 2012; pp. 177–184.

[13] Nakahara, H.; Nakanishi, H.; Sasao, T. On a Wideband Fast Fourier Transform for a Radio Telescope. *SIGARCH Comput. Archit. News* **2012**, *40*, 46–51. doi:10.1145/2460216.2460225.

[14] Qureshi, F.; Gustafsson, O. Analysis of twiddle factor memory complexity of radix-2i pipelined FFTs. Proceedings of the 2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers; , 2009; pp. 217–220. doi:10.1109/ACSSC.2009.5470121.

[15] Nash, J.G. Distributed-Memory-Based FFT Architecture and FPGA Implementations. *Electronics* **2018**, *7*. doi:10.3390/electronics7070116.

[16] Cooley, J.W.; Lewis, P.A.W.; Welch, P.D. Historical notes on the fast Fourier transform. *Proceedings of the IEEE* **1967**, *55*, 1675–1677. doi:10.1109/PROC.1967.5959.

[17] Smith, R.A. A continued-fraction analysis of trigonometric argument reduction. *IEEE Transactions on Computers* **1995**, *44*, 1348–1351. doi:10.1109/12.475133.

[18] Kang, H.; Yang, B.; Lee, J. Low complexity twiddle factor multiplication with ROM partitioning in FFT processor. *Electronics Letters* **2013**, *49*, 589–591. doi:10.1049/el.2013.0689.