Abstract: Discrete Fourier transform (DFT) is one of the most important tools used in almost all fields of science and engineering. DFT can be implemented with efficient algorithms generally classified as fast Fourier transforms (FFT). The most widely used approaches are so-called the algorithms for, such as radix-2, radix 4 and split radix FFT (SRFFT). Considerable researches have carried out and resulted in the rapid development on this class of algorithms. Simultaneously, the researches on the algorithms for computing length- DFT have resulted in the presentation of the methods.

Keywords: Discrete Fourier Transform (DFT), Fast Fourier Transform (FFT), General Split Radix, Radix 6.

I. INTRODUCTION
Discrete Fourier transform (DFT) is one of the most important tools used in almost all fields of science and engineering [2]–[4]. DFT can be implemented with efficient algorithms generally classified as fast Fourier transforms (FFT). The most widely used approaches are so-called the algorithms for 2^m, such as radix-2, radix 4 and split radix FFT (SRFFT). Considerable researches have carried out and resulted in the rapid development on this class of algorithms [5]–[8]. Simultaneously, the researches on the algorithms for computing length-N =k^m DFT have resulted in the presentation of the methods for k =3 and k =6 [9]–[12]. Due to the poor efficiency, the algorithms for k^m are of trivial practical meanings when k ≠ 2. However, there exists many applications in which the sequence lengths are3^m or 6^m. The idea of this letter is to develop a useful algorithm for length-N = 6^m DFT. The available published algorithms are reported in [11]. It seems that the general split radix algorithm is more adequate for the length-N=6^m DFT.

In this letter, we propose an algorithm based radix-6 approach. The algorithm is implemented with more efficient than the reported ones. Its computational complexity is approximately equal to 4.071N log_2 N -5.61N +33.55 log_2 N -130.292, which is close to that of standard SRFFT (The complexity of SRFFT is 4N log_2 N – 6N +8). The proposed algorithm is a radix 3/6 algorithm and uses base (1, j). The algorithm decomposes a DFT of size-N =6^m into one length-N/3and four length- N/6 sub DFTs. The flexibility of the decomposition enables the algorithm is competent at the implementation of a non-power-of-six DFT, while its length can exactly divided by 6. Appropriate permutations are used for sub DFTs input sequences to reduce the computational intensity. The rest of this paper is organized as following: Section II Literature Survey. Section III compares and analyzes the performance of the proposed algorithm. Section IV is conclusions and the future works.

II. LITERATURE SURVEY

A. Radix 2/8 FFT Algorithm for Length qx2m
A new radix-2/8 Fast Fourier Transform (FFT) algorithm have been proposed for computing the Discrete Fourier transform of an arbitrary length N= qx2^m, where m is an odd integer. It reduces substantially the operations such as data transfer, address generation, and twiddle factor evaluation or access to the lookup table, which contribute significantly to the execution time of FFT algorithms. It is shown that the arithmetic complexity (multiplications, additions) of the proposed algorithm is, in most cases, the same as that of the existing split-radix FFT algorithm. The basic idea behind the proposed algorithm is the use of a mixture of radix-2 and radix-8 index maps. The algorithm is expressed in a simple matrix form, thereby facilitating an easy implementation of the algorithm, and allowing for an extension to the multidimensional case. For structural complexity, the important properties of the Cooley–Tukey approach such as the use of the butterfly scheme and in-place computation are preserved by the proposed algorithm. It is suitable only for DFT of sequence length N=qx2^m.

B. Radix 2/16 FFT Algorithm for Length Qx2m
A radix-2/16 decimation-in-frequency (DIF) fast Fourier transforms (FFT) algorithm and its higher radix version, namely radix-4/16 DIF FFT algorithm, have been proposed by suitably mixing the radix-2, radix-4 and radix-16 index maps, and combing some of the twiddle factors. It is shown that the proposed algorithms and the existing radix-2/4 and radix-2/8 FFT algorithms require exactly the same number of arithmetic operations (multiplications and additions). By using this technique, it can be shown that all the possible split-radix FFT algorithms of the type radix- 2r/2rs for
computing a \( 2^m \) DFT require exactly the same number of arithmetic operations. This algorithm is suitable only for sequence of length \( N=2^m \), \( m \) is integer.

### C. New Radix-6 FFT Algorithm

A new radix-6 FFT algorithm suitable for multiply-add instruction have been proposed. The new radix-6 FFT algorithm requires fewer floating-point instructions than the conventional radix-6 FFT algorithms on processors that have a multiply-add instruction. Techniques to obtain an algorithm for computing radix-6 FFT with fewer floating-point instructions than conventional radix-6 FFT algorithms have been proposed. The number of floating-point instructions for the new radix-6 FFT algorithm is compared with those of conventional radix-6 FFT algorithms on processors with multiply-add instruction.

### III. PERFORMANCE ANALYSIS

In this section, we consider the performance of the proposed algorithm by analyzing its computational complexity and comparing it with the existing algorithms reported in [11].

Let \( M_N \) and \( A_N \) be, respectively, the number of real multiplications and real additions. We assume that a 3-points DFT requires 4 real multiplications and 12 real additions (some algorithms assume that a 3-points DFT is calculated with 2 real multiplication and 12 real additions, since one need not multiply 1/2 and the multiplication by 1/2 can be evaluated with bit shift) as shown in Fig.1.

The general butterfly of the proposed algorithm requires 16 real multiplications and 40 real additions. In general butterfly, we evaluate with 8 real multiplications and 16 real additions because \( w_{2^m} w_{3^m} w_{6^m} w_{w_{2^m} w_{3^m}} \) and \( w_{2^m} w_{w_{2^m} w_{3^m}} \). We calculate with 8 real multiplications and 8 real additions because we share real additions with which have been undertaken in evaluating. We evaluate with only 4 real additions, because \( 1+u+u^*=0 \). Furthermore, we perform at cost of 12 real additions, because all multiplications and some additions have been calculated. There are six special cases. The first special case, when \( k=0 \), requires 8 real multiplications and 32 real additions. In this case is evaluated with 8 real additions (one need not multiply 1) is implemented with 4 real multiplications and 6 real additions, because we use real additions which have been undertaken in evaluating above calculation, can be calculated with only 2 real additions, because we need not add the duplicate portion between \( u \) and \( u^* \). In the same way can be performed by only 4 real multiplications and 16 real additions. This special butterfly is illustrated in Fig. 2.

### TABLE I: Comparison Of Arithmetic Complexity

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3-point</td>
<td>( M_N )</td>
<td>( A_N )</td>
<td>( M_N )</td>
<td>( A_N )</td>
<td>( M_N )</td>
</tr>
<tr>
<td>3-point</td>
<td>36</td>
<td>20</td>
<td>156</td>
<td>21</td>
<td>482</td>
</tr>
<tr>
<td>6-point</td>
<td>68</td>
<td>70</td>
<td>2018</td>
<td>22</td>
<td>788</td>
</tr>
<tr>
<td>12-point</td>
<td>216</td>
<td>1164</td>
<td>4638</td>
<td>1164</td>
<td>1164</td>
</tr>
<tr>
<td>24-point</td>
<td>422</td>
<td>1004</td>
<td>10704</td>
<td>2596</td>
<td>10430</td>
</tr>
<tr>
<td>48-point</td>
<td>648</td>
<td>3808</td>
<td>1796</td>
<td>2584</td>
<td>1578</td>
</tr>
<tr>
<td>96-point</td>
<td>396</td>
<td>14472</td>
<td>5754</td>
<td>3025</td>
<td>360</td>
</tr>
<tr>
<td>192-point</td>
<td>2972</td>
<td>5042</td>
<td>8282</td>
<td>11354</td>
<td>3546</td>
</tr>
<tr>
<td>384-point</td>
<td>1080</td>
<td>25916</td>
<td>13594</td>
<td>30510</td>
<td>4468</td>
</tr>
<tr>
<td>768-point</td>
<td>776</td>
<td>10890</td>
<td>20230</td>
<td>30954</td>
<td>9390</td>
</tr>
</tbody>
</table>

The second special case, when \( k=2^{r-2} \times 3^{m-1} \), requires the number of operations equals that of the first case. In this case, all rotator factors of sub DFTs it can be omitted, so it can be evaluated with 8 real additions, can be implemented with 4 real multiplications and 6 real additions, can be calculated with only 2 real additions similarly can be performed by only 4 real multiplications and 6 real additions. The third special case is when \( k=2^{r-3} \times 3^m \). This butterfly requires 12 real multiplications and 36 real additions. In this case, requires extra 4 real multiplication and 4 real additions over the first case. The computations of the rest ones are similar with that of the first case. The fourth special case is when \( k=2^{r-3} \times 3^{m-1} \). This butterfly requires also 12 real multiplications and 36
real additions. In this case requires extra 4 real multiplication and 4 real additions over that of in the second case as shown in Fig.3. The computations of the rest equations are similar with that of the second case.

The fifth special case is when \( k \mod 3^m = 0 \) and \( k \mod 2^{-r} \neq 0 \). This butterfly requires 16 real multiplications and 36 real additions. In this case requires extra 8 real multiplication and 4 real additions over the first case. The sixth special case is when \( k \mod 3^{-m} = 0, k \mod 3^m \neq 0 \) and \( k \mod 2^{-r} \neq 0 \). This butterfly requires 16 real multiplications and 36 real additions. In this case, requires extra 8 real multiplication and 4 real additions over the second case. The decomposition in the proposed algorithm is conducted recursively until the lengths of all sub DFTs cannot be exactly divided by 6. In general, there are only 1 the first special butterfly (if \( r \geq 1 \) and \( m \geq 1 \)), 1 the second special case butterfly (if \( r \geq 2 \) and \( m \geq 1 \)), 1 the third special case butterfly and 1 the fourth special case butterfly (if \( r \geq 3 \) and \( m \geq 1 \)). The total number of the fifth and sixth type of butterflies is \( 2^{-r} \cdot 4 \). In additions, there are \( 2^{-1}(3^{m-1} -1) \) general butterfly. Thus, the arithmetic complexity of the proposed algorithm can be given as follows

\[
M_N = \begin{cases} 
\frac{M_N}{6} + \frac{4M_N}{6} + \frac{8N}{3} & r = 1, m \geq 1 \\
\frac{M_N}{6} + \frac{4M_N}{6} + \frac{8N}{3} - 16 & r = 2, m \geq 1 \\
\frac{M_N}{6} + \frac{4M_N}{6} + \frac{8N}{3} - 24 & r \geq 3, m \geq 1 
\end{cases} 
\]

\[
A_N = \begin{cases} 
\frac{A_N}{6} + \frac{4A_N}{6} + \frac{20N}{3} & r = 1, m \geq 1 \\
\frac{A_N}{6} + \frac{4A_N}{6} + \frac{20N}{3} - 16 & r = 2, m \geq 1 \\
\frac{A_N}{6} + \frac{4A_N}{6} + \frac{20N}{3} - 2^{r+1} - 8 & r \geq 3, m \geq 1 
\end{cases} 
\]

A comparison of arithmetic complexity with the published algorithms have made and the results are shown in Table I presents radix-k FFT (C-FFT) to calculate a length-\( N = k^m \) DFT using convolutions. The number of real multiplications for the implementation of C-FFT is less than that of T-FFT introduced, but the number of real additions required is more. If real multiplication requires sufficiently more time than real addition required on digital equipment, C-FFT will better be than T-FFT. But CPU time required by real multiplication is the same as real addition on recent processors, and T-FFT shows its advantages over C-FFT. In the implementation of General SRFFT, the length-\( N = 3^m \times 2^m \) DFT are evaluated accordingly to the DFT of length-\( q \times 2^m \) where \( q = 3 \), and the length-\( q = 3^m \) sub-DFTs are computed with radix-3 algorithm in [10]. In addition, we assume that the number of real multiplications of the length-\( N = 3^m \) scaled DFTs \( M_{3N} = M_N + 2 \). From this table, we can see that the proposed algorithm achieves a reduction of the number of operations.

**IV. CONCLUSIONS AND FUTURE WORK**

A radix 3/6 FFT algorithm is presented for length-\( N = 6^m \) DFT. The proposed algorithm is a mixture of radix-3 and radix-6 algorithm. It can evaluate a non-power-of-six DFT, as long as its length- can be divided by 6. In order to reduce the number of operations, all sub DFTs are reordered favorably. The proposed algorithm shows that its implementation requires less real operations as compared with the published algorithms. Its arithmetic complexity is about \( 4.071N \log_2 3 - 5.61N + 33.555 \log N - 130.292 \), which is close to that of standard SRFFT. Due to being an irregular integer for the sequence lengths, it is difficult to gain a completely accurate formula of computational complexity. In the future, since the proposed algorithm shows advantages for computing length-\( N = 2^i \times 3^j \) DFT, we will do some works in this way. The algorithms presented in [8] can be improved by scaled DFT, and the comparison of its computational complexity with the proposed algorithm needs to make.

**V. REFERENCES**


