DESIGN OF LOW POWER HIGH PERFORMANCE 16-POINT 2-PARALLEL PIPELINED FFT ARCHITECTURE

SENTHIL SIVAKUMAR M & BANUPRIYA M & AROCKIA JAYADHAS S

Faculty in St.Joseph University, Tanzania

ABSTRACT

In this paper, we proposed a low power high performance 16-point 2-parallel pipelined FFT architecture with the use of various low power functional block implementation techniques. This paper presents the techniques to minimize the power consumption of the FFT architecture by reducing number of functional blocks used to implement the FFT processor. The FFT is employed by the modified radix-4 algorithm to significantly reduce the number of complex multipliers. The parallel pipelined technique is introduced to increase the throughput of the circuit at low frequency. The IDR commutator technique is used to reduce the power dissipation by reducing the total elements used for the memory element compared to conventional commutator. Multiplier is a foremost device which increases the size of real-time FFT processors. In this paper, we replaced such complex multipliers by minimum number of adders and shifters with two’s complement and canonical signed-digit (CSD) representations. Also the new low power butterfly architecture is introduced to implementing the radix-4 butterfly operation that reduces the area and power consumption of the FFT processor. The employments of these low power techniques in 16-point FFT architecture is yield 45% of power minimization compare to the conventional FFT architecture. The output values are achieved by the simulation and synthesis of Verilog HDL codes in cadence RTL compiler using tsmc18_w110.

KEYWORDS: Butterfly Architecture, CSD, FFT, Low Power, Parallel and Pipelined FFT architecture, R4SDC FFT.

INTRODUCTION

The Fast Fourier Transform (FFT) is one of the fundamental operations in the field of digital signal processing. The FFT processors are widely used in various areas such as telecommunications, consumer electronics, mobile systems, speech and image processing, etc. Recently, the FFT is used as one of the key components in OFDM based wideband communication systems. An FFT processor can be classified into three main types: Pipelined FFT, Column FFT, and Fully parallel FFT. Pipelined FFT is preferred especially for a high throughput demand or low power solution. In real time applications, input data is a sequential stream. Therefore it does no match the FFT algorithm since the FFT requires temporal reordering of data. For this reason, the commutator is used to reorder the input data. Among several pipelined FFT architectures, radix-4 single path delay commutator is widely used, owing to its
high utilization of multipliers and butterfly elements. In the pipelined architectures, the commutator and complex multiplier at each stage contribute a dominating part of the whole power consumption.

The complex multiplier within the butterfly processing unit is one of the most powers consuming block of the pipelined FFT processor. The switching activity between successive coefficients fed to the complex multiplier can be reduced by coefficient ordering and hence its power consumption be minimized. The power of a pipelined FFT processor is dominated by the size of storage blocks. Moreover, the commutators will take up more proportion of the overall power consumption and acts as a leading actor with the increase of FFT size.

In [5] and [2], the authors proposed a low-power FFT architecture based on multi-rate signal processing and asynchronous circuit technology. The communication is localized, and the sharing of the global memory is eliminated. To reduce the number of operations in FFTs and thus reduce power consumption, some researchers use shifters and adders to replace the complex multiplications by some special constant coefficients. The authors of [4] employ seven shift-and-add units to carry out seven multiplications in parallel, each by a constant coefficient. In [3] and [4], authors are proposed a multiplier less architecture based on common sub expression sharing which replaces the complex multipliers in FFTs, and a low-power commutator architecture which reduces the number of memory accesses. In addition to the pipelined FFT, the parallel-pipelined FFT is a good solution for applications requiring high throughput and high power efficiency [1]. This paper implements a number of FFT IP cores based on both pipelined and parallel-pipelined architectures, through different combinations of hybrid low-power architectures. The low power pipelined architectures are presented by the authors of paper [8], [9] and [11] with high throughput.

In [5] and [6], the authors proposed a low-power FFT architecture based on multi-rate signal processing and asynchronous circuit technology. In [1] and [4], we proposed a multiplier less architecture based on common sub expression sharing, which replaces the complex multipliers in FFTs, and a low-power commutator architecture, which reduces the number of memory accesses. In addition to the pipelined FFT, the parallel-pipelined FFT is a good solution for applications requiring high throughput and high power efficiency. The communication is localized, and the sharing of the global memory is eliminated. To reduce the number of operations in FFTs and thus reduce power consumption, some researchers use shifters and adders to replace the complex multiplications by some special constant coefficients. The high performance low power FFT cores are proposed in [1], for digital signal processing applications. The increased standards of integrated circuits requires the low power systems for the portable systems usage and to maximize their efficiency with respect to the power consumption. Notably, the wide usages of Fast Fourier Transform in the DSP applications direct us to invent the new low power architecture for FFT.

ARCHITECTURE OF FAST FOURIER TRANSFORM

The N–point DFT performs the transformation of N-point time domain data into N-point frequency domain data [1]. Discrete means that the data is sampled at a given time instead of being continuous. The DFT operates on an N-point sequence of numbers x(n). This sequence is usually
obtained by uniform sampling of a finite period of some continuous function. The number of complex multiplication and addition operations for the simple form of an N-input Discrete Fourier Transform (DFT) requires the order of $N^2$.

The DFT of $x(n)$ is an N-point sequence $X(k)$, that is defined by equation (1).

$$X(K) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad 0 \leq k \leq N-1 \quad (1)$$

Where,

$$W_N^k = e^{-j2\pi k/N} \rightarrow \text{Twiddle factor.}$$

Twiddle factor is a periodic function in the period N. The complex algorithm of DFT leads the use of FFT in practical DSP applications. FFT reduces the computation time required to compute a discrete Fourier transform and improves the performance by factor 100 or more over direct evaluation of DFT.

Algorithm:

DFT: $O(N^2)$ operation and time complexity

FFT: $O(N \log N)$ operation and time complexity

Let $N$ be a composite number of $v$ integers so that $N = r_1 r_2 \ldots r_v$, and define

$$N_t = N/(r_1 r_2 \ldots \ldots r_t) ; 1 \leq t \leq v-1,$$

Where, $t$ is the stage number of the decomposed DFT and $r_t$ its radix. The pipelined FFT processor is obtained by decomposing an N-point DFT into $v$ stages. The final stage is defined as follows,

$$X(r_t, \ldots, r_v, m_s + \ldots + r_t m_1 + m_t) = \sum_{q_{t-1} = 0}^{2^{t-1}} W^{m_1 q_t}_{r_t} x_{r_t}(q_{t-1}, m_1, \ldots, m_t) \quad (2)$$

Whereas the computations for intermediate stages $t$ (totally $v-2$ intermediate stages) are given as follows:

$$x_{r_t}(q_t, m_1) = W^{m_1 q_t}_{r_t} \sum_{p=0}^{2^{t-1}} W^{m_q}_{r_t} x_{r_t}(N_t p + q_t, m_2, \ldots, m_v) \quad (3)$$

Where,

$$0 \leq q_t \leq N_t - 1; \quad 2 \leq t \leq v \quad \text{and} \quad N_t = N/(r_1 r_2 \ldots \ldots r_t) ; \quad 2 \leq t \leq v - 1; \quad 0 \leq m_t \leq q_t - 1$$

The term ‘FFT’ is actually slightly ambiguous, because there are several commonly used ‘FFT’ algorithms. Here the radix-$4$ FFT algorithm is described for low power 16-point 2-parallel pipelined FFT design.

**RADIX-4 ALGORITHMS**

The radix-4 FFT equation essentially combines two stages of a radix-2 FFT into one, so that half as many stages are required [1]. Since the radix-4 FFT requires fewer stages and butterflies than the radix-2 FFT, the computations of FFT can be further improved. For example, to calculate a 16-point FFT, the radix-2 takes $\log_2 16=4$ stages but the radix-4 takes only $\log_4 16=2$ stages. Next, we discuss the numerical issue that arises from a finite length problem. Most people use a fixed-point DSP to perform the calculation in their embedded system because the fixed-point DSP is highly programmable and is cost efficient. The drawback is that the fixed-point DSP has limited dynamic range, which is worsened
by the summation overflow problem that occurs all the time in FFT. A scheme is needed to overcome this issue.

![Flow graph of a 16-point radix-4 FFT algorithm](image)

*Fig 1: Flow graph of a 16-point radix-4 FFT algorithm.*

When the number of data points $N$ in the DFT is a power of $4 (N = 4^v)$, then it is more efficient computationally to employ a radix-4 algorithm instead of a radix-2 algorithm. A radix-4 FFT algorithm is obtained by splitting the $N$-point input sequence $x(n)$ into four sub sequences $x(4n)$, $x(4n + 1)$, $x(4n + 2)$ and $x(4n + 3)$. The radix-4 FFT algorithm is obtained by selecting $L=4$ and $M=N/4$ in the unified approach. This leads to $n = 4m + 1$ and $k = (N=4) p + q$. The radix-4 FFT algorithm is obtained by following the decomposition procedure outlined in the previous section v time’s recursively. The signal flow graph of a 16-point radix-4 FFT algorithm is shown in Fig 1.

The radix-4 [12]butterfly, shown in Fig 2, is constructed by merging 4-point DFT with associated coefficients between DFT stages. The four outputs of the radix-4 butterfly namely $X(4n)$, $X(4n+1)$, $X(4n+2)$ and $X(4n+3)$ are expressed in terms of its inputs $x(n)$, $x(n)+N/4$, $x(n)+N/2$ and $x(n)+3N/4$ as follows:

![Radix-4 Butterfly](image)

*Fig 2: Radix-4 Butterfly*

The inputs are,

\[
\begin{align*}
    x(n) &= x_1 + jx_2 \\
    x(n) + N/4 &= x_1 + jx_2 \\
    x(n) + N/2 &= x_1 + jx_2 \\
    x(n) + 3N/4 &= x_2 + jx_4
\end{align*}
\]
Here, $x_a$, $x_b$, $x_c$, $x_d$, $y_a$, $y_b$, $y_c$, and $y_d$ are real and imaginary parts of inputs. The twiddle factors are,
\[ W^a = C_a + jS_a \]
\[ W^b = C_b + jS_b \]
\[ W^c = C_c + jS_c \]
\[ W^d = C_d + jS_d \]
\[ \text{(5)} \]

Here $C_a$, $C_b$, $C_c$, $S_a$, $S_b$, $S_c$, and $S_d$ are real and imaginary parts of twiddle factors.
\[ X(4n) = x_a + jy_a \]
\[ X(4n + 1) = x_b + jy_b \]
\[ X(4n + 2) = x_c + jy_c \]
\[ X(4n + 3) = x_d + jy_d \]
\[ \text{(6)} \]

The outputs are calculated based on the following equations.
\[ x'_a = [x_a + x_b + x_c + x_d] \]
\[ x'_b = [x_a + x_b - x_c - x_d] \]
\[ x'_c = [x_b - x_a + x_c + x_d] \]
\[ x'_d = [x_c - x_a - x_b + x_d] \]
\[ y'_a = [x_a - x_b - x_c + x_d] \]
\[ y'_b = [x_b + x_c + x_d - x_a] \]
\[ y'_c = [x_c - x_b + x_a + x_d] \]
\[ y'_d = [x_b - x_c + x_a - x_d] \]
\[ \text{(7)} \]

The radix-4 butterfly requires three complex multiplications. The multiplication with ‘i’ is accomplished by negation and swapping of the real and imaginary parts. Radix-4 has a computational advantage over radix-2 because radix-4 butterfly does the work of four radix-2 butterflies using three multipliers instead of four multipliers in four radix-2 butterflies. On the negative side, a radix-4 butterfly is more complicated to implement than a radix-2 butterfly. While radix-2 and radix-4 FFTs are certainly the most widely known algorithms, it is also possible to design FFTs with even higher radix butterflies. They are not often used because the control and dataflow of their butterflies are more complicated and the additional efficiency gained diminishes rapidly for radices greater than four. Implementation of this architecture provides 75 percent utilization of complex multiplier and 100% employment of adders/subtractor \[11\] and \[12\]. The architecture design of radix-4 16-point 2-parallel pipelined FFT is illustrated by the employment of low power technique is described below.

**LOW POWER RADIX-4 16-POINT 2-PARALLEL PIPELINED FFT**

In this paper, we proposed new low power butterfly structure for 16-point 2-parallel pipelined FFT architecture implementation. The proposed low power structure reduces the total power dissipation of FFT section along with the high performance IDR commutator, low power multiplier less architecture, and low power butterfly structure. Four different techniques are used to reduce the power dissipation of FFT architecture such as: Parallel-pipelined architecture with low frequency, IDR commutator, Multiplier less architecture, and low power butterfly structure. The first technique involves the use of a parallel-pipelined architecture at a lower frequency to meet the given throughput. The second technique proposes low-power architecture of the commutator (IDR) with a minimal number of memory write
operations. The third technique replaces the complex multiplier with a minimum number of adders and shifters by using both two’s complement and canonical signed-digit (CSD) representations. The last low power butterfly architecture aims to reduce the area and power by removing unwanted sub blocks from low power architecture proposed in [1]. These techniques are illustrated in the following sections.

**PARALLEL PIPELINED ARCHITECTURE**

The pipelined FFT is viewed as the leading architecture for real time applications. However, the use of only one processor element (PE) in each stage limits the throughput of pipelined FFTs. Therefore, an increased throughput requires further parallelization. To meet the growing demand of high processing rate, parallel pipelined FFT processors are required because they can increase the FFT processing rate greatly, high throughput and high power efficiency applications.

In parallel pipelined architectures, only the number of PEs is increased, and the sizes of FIFOs remain the same. Hence, the area overhead of the parallel-pipelined architectures is not significant. For a given throughput, parallel-pipelined FFTs can operate at lower frequencies than the pipelined FFTs, resulting in lower power consumption. In a 16-point 2-parallel-pipelined FFT, 2 PEs are allocated to each stage of the FFT, doubling the throughput. This architecture is shown in Fig. 4.1, where the input data is separated into two streams, namely $X(2n)$ and $X(2n+1)$. Two commutators in stage 1, each having half of the storage units of the original commutator, provide the data to the butterfly units. The coefficients are divided into even (coefficient 1) and odd (coefficient 2) sections according to even and odd input data streams. These coefficients are then fed into the corresponding complex multipliers. Due to the separate processing for odd and even data, a shuffle unit is needed in stage 2 to implement the inter stage data shuffle. The shuffle unit is composed of two triple port SRAMs (TM1 and TM2) and an addressing control unit. The four outputs from the two triple port SRAMs are then fed into the simplified butterfly units in stage 2. The input data is separated into four streams, namely, $X(4n)$, $X(4n+1)$, $X(4n+2)$, and $X(4n+3)$. There are four commutators in both stage 1 and stage 2. Each of the commutators has $1/4$ size of the commutators in corresponding stages of the R4SDC pipelined FFT. The coefficients in each stage are divided into four sections. Coefficients 1 to 4 correspond to the four input streams. Only 3 complex multipliers are used in stage 2, since the coefficient set for the first butterfly consists of only (7fff, 0000), which can easily be implemented without a multiplier. In this architecture, since the number of PEs per stage is equal to four, no shuffle unit is needed. The outputs of the multiplier units in stage 2 are fed into each of the four simplified butterfly elements in stage 3.
MULTIPLIER LESS ARCHITECTURE

In FFTs, the conventional complex multiplier consists of four real multipliers, one adder and one subtractor. However, since the complex coefficients for all stages can be pre-computed, we can use shift and add operations with common sub expression sharing for those stages that have few coefficients. In [4], we proposed a multiplier less architecture to substitute the complex multiplier. For example, the number of coefficients used in the second stage of a 64-point FFT or the first stage of a 16-point FFT is 16. The coefficients for 16-point R4SDC FFT, is given in table 1.

Table 1

<table>
<thead>
<tr>
<th>Coefficient sequence m1=0,1</th>
<th>Original quantized coefficient</th>
<th>Coefficient sequence m1=2,3</th>
<th>Original quantized coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>7fff, 0000</td>
<td>W0</td>
<td>7fff, 0000</td>
</tr>
<tr>
<td>W0</td>
<td>7fff, 0000</td>
<td>W2</td>
<td>5a82, a57d</td>
</tr>
<tr>
<td>W0</td>
<td>7fff, 0000</td>
<td>W4</td>
<td>0000, 8000</td>
</tr>
<tr>
<td>W0</td>
<td>7fff, 0000</td>
<td>W6</td>
<td>a57d, a57d</td>
</tr>
<tr>
<td>W0</td>
<td>7fff, 0000</td>
<td>W0</td>
<td>7fff, 0000</td>
</tr>
<tr>
<td>W1</td>
<td>7641, cf04</td>
<td>W3</td>
<td>30fb, 89be</td>
</tr>
<tr>
<td>W2</td>
<td>5a82, a57d</td>
<td>W6</td>
<td>a57d, a57d</td>
</tr>
<tr>
<td>W3</td>
<td>30fb, 89be</td>
<td>W9</td>
<td>89be, 30fb</td>
</tr>
</tbody>
</table>

Canonic Signed Digit (CSD)

It is common to use the redundancy of signed digit code to replace the conventional multiplier digits, such that addition operations in a multiplication can be reduced with the increase of average shift length across the zeros in the multiplier. Canonical Signed-Digit (CSD) is one widely used signed digit approach. The advantage of CSD form is that no value has more than (N+1)/2 nonzero bits, often fewer,
and so the multiplication by a constant requires no more than that number of additions for its implementation. An algorithm for transform from two’s complement to CSD form can be found in,[1].

A close observation of these coefficients reveals that seven of them are (7fff, 0000), and one of them is (0000, 8000). These are the quantized representations of (1, 0) and (0, -1) in a 16-bit fractional two’s complement format, respectively. In each set, the first entry corresponds to the cosine function (the real part, \(W_r\)) and the second entry corresponds to the sine function (the imaginary part, \(W_i\)). For trivial coefficients, such as (7fff, 0000) and (0000, 8000), complex multiplication is unnecessary. For example, the data can directly pass through the multiplier unit without any multiplication, when it is multiplied with the coefficient set (7fff, 0000). Similarly, only an additional unit, which swaps the real and imaginary parts of the input data and inverts the imaginary part, is needed for that data which is multiplied by (0000, 8000). The rest of the coefficients are nontrivial coefficients.

They are composed of only 6 constants (7641, 5a82, 30fb, a57d, 89be, and cf04). However, 89be, a57d, and cf04 are the one’s complements of 7641, 5a82, and 30fb, respectively. Hence, only multiplications with these constants (7641, 5a82, and 30fb) would be required to implement all multiplications with these nontrivial coefficients. For example, a multiplication with the constant a57d could be realized by first multiplying the data with 5a83, and then two’s complementing the result. Note that a multiplication by the constant 5a82 is already available. Therefore, the multiplication with the constant 5a83 can simply be obtained by adding the data to the already existing multiplication with 5a82. The other two constants (89be and cf04) can be realized in a similar manner, using constants 7641 and 30fb, respectively.

The constant 5a82 is represented by two’s complement format, and 7641 and 30fb are represented by CSD format as follows: 5a82 (0101101010000010), 7641 (1000-10-1001000001), and 30fb (010-1000100000-10-1). The mixed use of CSD and two’s complement minimizes the number of addition/shift operations. We can use the shift-add based implementation of multiplications with the three constants to carry out those nontrivial complex multiplications. According to the previous representation, these multiplications with the three constants are given by:

\[
\begin{align*}
5a82X &= X <<1 + X <<7 + X <<9 + X <<11 + X <<12 + X <<14, \\
7641X &= X + X <<6 - X <<9 - X <<11 + X <<15, \\
30fbX &= -X - X <<2 + X <<8 - X <<12 + X <<14,
\end{align*}
\]

Where,\(X\) represents the input data.

**COMMON SUB EXPRESSION SHARING**

Common sub expression sharing shares the sub expression among several multiplication-accumulation operations in order to reduce the total number of operations. This approach is very effective for reducing the hardware cost of multiple constant multiplications, especially for the filter-like operation.
In the previous equations, the number of operations required for the computation of 5a82X, 7641X, and 30fbX are shown in Table 2. In all nontrivial coefficient multiplications, the proportions of the multiplications referring to 5a82, 7641, and 30fb are 50%, 25%, and 25%, respectively. Hence, the average operations for a nontrivial coefficient multiplication are 4.5 additions, 2.5 subtractions, and 7 shifts.

\[ Y(2) = \sum_{i=0}^{N-1} A_i \times X_{i-1} \] (11)

The weights \( A_i \) are the filter coefficients.

### Table 2: Operations required before common sub expression sharing

<table>
<thead>
<tr>
<th>Operation</th>
<th>5a82X</th>
<th>7641X</th>
<th>30fbX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Subtraction</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Shift</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

### Table 3: Operations required after common sub expression sharing

<table>
<thead>
<tr>
<th>Operation</th>
<th>5a82X</th>
<th>7641X</th>
<th>30fbX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Subtraction</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Shift</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

However, if \( 5X = X + X << 2 \) and \( 65X = X + X << 6 \) are pre-computed, (8), (9) and (10) can be rewritten as

\[ 5a82X = 5X << 12 + 5X << 9 + 65X << 1, \] (12)

\[ 7641X = X << 15 + 65X << 5X << 9, \] (13)

\[ 30fbX = 65X << 8 - X << 12 - 5X. \] (14)

The common sub expressions for the three constants are 101 and 1000001. Pre-computation requires 2 additions and 2 shifts. In [4], the operations required for the computations of 5a82X, 7641X, and 30fbX are shown in Table 3. The results of the pre-computation can be used for both multiplications with the real part (\( W_r \)) and the imaginary part (\( W_i \)) of nontrivial coefficients. Therefore, after the common sub expression sharing, the average operations for a nontrivial coefficient multiplication are 3.5 additions, 1.5 subtractions, and 6 shifts, including the operations in pre-computation. Therefore, the operation saving of additions/subtractions is 28.8%, and that of shift is 14.3%.
SHIFT-ADD ARCHITECTURE

Fig. 4: Block diagram of the shift-and-add module in multiplier less unit of the 16-point R4SDC FFT.

Fig 4, shows the shift-and-add module for the three constants in a 16-point FFT. The module carries out the multiplications in which the real part \((X_r)\) or the imaginary part \((X_i)\) of input data is multiplied by \(W_r\) and \(W_i\), respectively. The shift-and-add module is equipped with five single-bit control signals, \(s_1\)–\(s_5\). First, the input data is fed into the common sub expression block. The signal \(s_1\) indicates which constant channels will be chosen for processing the input data. Each channel carries out shift, negation, and addition operations for a given constant. The control signal \(s_3\) indicates that the constant 7641 block outputs the product either by 7641 or 7642. Similarly, the signals \(s_2\) and \(s_4\) control the outputs of constant 5a82 and 30fb blocks respectively. The controllable invert units following the constant units either invert the outputs of the constant units or pass them unchanged. The swap unit provides the appropriate swapping for input data, depending on whether the coefficient is (30fb, 7641) or (7641, 30fb). The output switch unit selects the final outputs. Only 11 adders are used in the shift-and-add module.

Fig. 5: Block diagram of the multiplier less unit in the 16-point R4SDC FFT.
Based on the above discussion, the complex multiplication unit in a 16-point radix-4 pipelined FFT can be substituted by a multiplier less unit. The block diagram of the unit is depicted in Fig 5. Only data that has to be multiplied with nontrivial complex coefficients is fed into the shift-and-add units. Two shift-and-add units are needed for both the real part (Xr) and the imaginary part (Xi). There are two single-bit control signals, s6 and s7, in the multiplier less unit. Signal s6 indicates whether the input data corresponds to a nontrivial complex coefficient. When signal s7 is asserted to logic 1 state, the real and imaginary parts of the input data are swapped, and the imaginary part is inverted. Otherwise, the swap unit passes the input data unchanged. Here, in the multiplier less unit, 22 adders are used to substitute the four real multipliers in the complex multiplier unit. Due to the use of the multiplier less unit, the ROM unit storing the coefficients is replaced by an FSM unit for generating the control signals (s1–s7). The multiplier less approach can also be used for 16-point 2-parallel-pipelined FFTs and 64-point 4-parallel-pipelined FFTs. Similar multiplier less architectures such as the one shown in Fig. 5 are employed in place of complex multipliers in those FFTs.

**IDR COMMUTATOR**

![IDR Commutator Architecture](image)

**Fig 6:** IDR commutator architecture based on dual port RAMs for R4SDC.

The new proposed low power commutator architecture is shown in Fig 6. The architecture, which is termed as IDR, uses dual port RAMs as FIFO elements, however, the interconnection topology among the RAM blocks is different from that of the conventional approach. Table 4 illustrates which RAM blocks are enabled for write access during each period. It can be seen that there are at most three RAM blocks selected in a given period. For stage t, when $m_t$ is equal to 1, new $N_t$ input data is processed. The first $N_t$ data will be written into DM0. The previous $N_t$ data stored in DM0 will be read out and written into DM2 for vacating space for the new data. The same applies for DM2 and DM4. The other three RAM blocks (DM1, DM3 and DM5) will be disabled for write access during this period. For $m_t = 0$ and 3, the number of RAM blocks enabled is two, because the previous data stored in DM2 and DM3 are no longer needed for subsequent outputs. Therefore, during the four periods, each RAM is enabled 5/3
times on average. Whereas for DR and TR architectures this is corresponds to 4 and 10/3 times, respectively. Hence our new commutator architecture is significantly good power efficient compared to other commutator architectures.

Table 4: RAMs Selected In Different Periods

<table>
<thead>
<tr>
<th>m_i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMs selected</td>
<td>DM1, DM3</td>
<td>DM0, DM2, DM4</td>
<td>DM1, DM3, DM5</td>
<td>DM0, DM2</td>
</tr>
</tbody>
</table>

In Fig 6, the signal bus \( m[6:0] \) generated by the controller unit is used to control the four multiplexers. The control signals \( c4, c5, \) and \( c6 \) are for the butterfly unit. The values on the RAMs’ output ports \( (A, B, D, E \) and \( F) \) are obtained by controlling their read addresses \( (radrra – radrrf) \) which are generated by the controller unit. The RAM blocks are enabled only if their outputs are currently needed in order to prevent unnecessary switching activity. Each input word occupies a word slot of duration \( T \) and is numbered according to its appearance in time.

**Low Power Butterfly Architecture**

Here, we are proposing new low power butterfly architecture shown fig7. The conventional architecture of butterfly consists of 6adders/subtractors. In low power butterfly architecture presented in [1], two five input summation blocks are used to implement the butterfly instead of using 6adders/subtractors. The inversions are implemented using one’s complement methods. Six inverters are used to generate the inputs of summation blocks by the control of \( C_4, C_5 \) and \( C_6 \). In this architecture we removed the unwanted sub blocks with the aims to reduce the area and power from architecture presented...
in [1]. This architecture was constructed based on the equations 5, 6 and 7. Input data’s are two’s complemented based on the control signals c4, c5 and c6. The decoder and multiplexers are removed by replacing select lines and two’s complement blocks instead of using one’s complement.

**RESULT ANALYSIS**

The four different power minimization techniques were implemented and analyzed by comparing with conventional techniques. All the presented all high performance low power architecture outputs are constructed in Verilog Hardware Description Language and synthesized in cadence RTL compiler using tsmc18_wl10 technology library file. The low power techniques were synthesized and observed the power and area of each technique separately. Also by employing the low power techniques in the 16-point 2-parallel pipelined FFT architecture, we obtained the power and area of 16-point FFT processor proposed. The area and power of conventional 16-point FFT and its sub-blocks are also obtained to compare with the proposed low power 16-point FFT. The comparison of power and area of proposed and conventional 16-point FFT and its sub-blocks are illustrated below.

**Power Analysis**

This comparison result gives us very brief and concise information between conventional and proposed high speed low power architectures as shown in Fig 8. As can be seen by the figures above the scheme III outperforms all the other architectures both in power and area. So in respect with the above comparisons results we will compare the area and power for our designed architectures.

Here, the multiplier less units achieves power savings encircling 40%. For commutators, IDR architectures perform better in each stage 32% power saving. Low-power butterfly consumes only about half the power of the conventional add-sub-based butterfly architecture. However, in this scheme, sum-based from all of the results, it is obvious that 16 point parallel-pipelined architectures can achieve the best power savings (30%). The proposed low power butterfly unit gives 33% of power saving compared to conventional butterfly unit.
Area Analysis

The area comparison chart shown in figure 9. Here the multiplier less unit performs around 20 percentage of area saving. The commutator having very minimum level of area rescues. Low power butterfly unit gives better area saving, more than 30%. In this high speed low power 16-point Fast Fourier transform scheme has reduces area nearly 8%. The proposed low power butterfly unit gives more than 20% of area saving compared to conventional butterfly unit.

CONCLUSIONS

In this paper, the high-performance low power 16-point 2-parallel pipelined FFT processor is designed and implemented based on combinations of hybrid low-power techniques. The total power of the device is reduced using the low-power techniques: parallel-pipelined architectures, the multiplier less architecture, IDR commutator architecture, and the low-power butterfly architecture. All the low power techniques are constructed in Verilog HDL and synthesized by Cadence RTL compiler using tsmc18_wl10 technology library file. The power and area impact parameters of each technique has been observed and compared with the conventional FFT blocks to analyze the performance. Based on this analysis the proposed low power techniques of 16-point FFT, up to 45% power saving is achieved. Also
this low power FFT architecture can be employed to implement the higher size FFT processors with low power and high performance.

REFERENCES

[6]. Kai Hwang Computer Arithmetic: principles, architecture, and design John Wiley & Sons, Inc. 1979 pp. 149-151
[12]. Implementing the Radix-4 Decimation in Frequency (DIF) Fast Fourier Transform (FFT) algorithm using a TMS320C80 DSP.