A Comparative Study of Different FFT Architectures for Software Defined Radio

Shashank Mittal, Md. Zafar Ali Khan, and M.B. Srinivas

Center for VLSI and Embedded System Technologies
International Institute of Information Technology, Gachibowli, Hyderabad, INDIA-500032
shashankmittal@research.iiit.ac.in,{zafar, srinivas}@iiit.ac.in

Abstract. Fast Fourier Transform (FFT) is the most basic and essential operation performed in Software Defined Radio (SDR). Thus designing regular, reconfigurable, modular, low hardware and timing-complexity FFT computation block is very important. A single FFT block should be configurable for varying length FFT computation and also for computation of different transforms like Discrete cosine/sine transform (DCT/DST) etc. In this paper, the authors analyze area, timing complexity and noise to signal Ratio (NSR) of Bruun's FFT w.r.t. classical FFT from a SDR perspective. It is shown that architecture of Bruun's FFT is ideally suited for SDR and may be used in preference over classical FFT for most practical cases. A detailed comparison of Bruun's and classical FFT hardware architectures for same NSR is carried out and results of FPGA implementation are discussed.

1 Introduction

The concept of SDR was first introduced in by Mitola [1] and Tuttlebee [2]. SDR refers to a converged system or hardware which can be reconfigured easily to support all wireless communication standards. This device should provide backward support and capability to adapt to future communication standards without any changes in hardware. Thus there is a need for converged and reconfigurable hardware architectures to perform various signal processing functions.

In order to effectively utilize the services provided by different wireless communication standards, especially those with high bandwidth, devices such as mobile phones should also support various multimedia standards which require computation of different transforms like FFT, DCT etc. Here baseband and multimedia operations are on different parts of the chip and different multimedia standards themselves require different transforms for their computations. Further, these computational blocks should be able to operate at varying computational (word or bit) lengths [3]. These converged architectures should have properties like regularity, reconfigurability, low hardware complexity, low timing complexity and low NSR.

Recent research related to FFT computation for SDR applications has dealt mainly with two issues, viz., i) variable length FFT computation architectures

S. Vassiliadis et al. (Eds.): SAMOS 2007, LNCS 4599, pp. 375–384, 2007.

[4-5] that focus on how one can utilize a 16-point or smaller length FFT for 32 or higher length FFT computation and vice-versa. ii) universal computational FFT architectures [6-7] in which a single FFT structure can compute different transforms like decimation in time/frequency (DIT/DIF) FFT, DCT, DST etc.

Bruun's FFT [8] can be considered to be an ideal candidate for SDR since its architecture has low computational complexity and can also be configured to compute FFT, DCT and DST [8]. However it suffers from high NSR [9] for a given fixed point implementation and because of this, is not generally used in practice. In this paper, the authors compare the hardware and timing complexity of Bruun's and DIT/DIF FFT for same NSR. It is shown that Bruun's FFT has less hardware complexity than that of DIT/DIF FFT for a given NSR, in most cases practical for SDR.

The rest of the paper is organized as follows: Section 2 describes the basic FFT computation algorithm while Bruun's FFT architecture is described in section 3. In section 4 we analyze and compare various FFT architectures for SDR. Results of hardware implementation are provided in section 5 and a detailed comparison is carried out. Finally, conclusions are drawn in section 6.

2 Basic FFT Computation Algorithm

A DFT of length N is defined as

$$X(k) = \sum_{i=0}^{N-1} x(i) \cdot W_N^{ik}, 0 \leq k \leq N-1 \tag{1}$$

where W denotes the Nth root of unity, with its exponent evaluated modulo N. Among the many possible ways, DFT can also be implemented by using transversal filters [8]. This filter structure has N, Z-transform based transfer functions, one at each output node of FFT, of the form

$$\begin{aligned} F_n(z) &= z^{-(N-1)} + W^{-1.n} z^{-(N-2)} + \cdots \\ &+ \cdots W^{-k.n} z^{-(N-k-1)} + \cdots W^{-(N-1).n} \end{aligned} \tag{2}$$

Equation (2) can be factored using repeated application of

$$\begin{aligned} F_n(z) = (z^{-(N/2)} + W^{-(N/2).n})(z^{-((N/2)-1)} \\ + W^{-n.1} z^{-((N/2)-2)} + \cdots + \cdots + W^{-((N/2)-1).n} \end{aligned}$$

so that (2),

$$F_n(z) = \prod_{t=1}^{log_2 N} (z^{-(N/v)} + W^{-(N/v).n}) \tag{3}$$

Where, $v = 2^t$ and $Z^{-1} * x(0) = x(1)$. This transfer function can also be represented as a tree structure after decomposition [8], which results in a classical FFT structure.

3 Bruun's FFT Algorithm

Bruun's algorithm [8] is a very promising method for computation of different discrete transforms, particularly FFT. It utilizes the fact that (2) can also be factored using the following equation (4)

$$1 + aZ^{2q} + Z^{4q} = (1 + \sqrt{2-a}Z^q + Z^{2q})(1 - \sqrt{2-a}Z^q + Z^{2q}) \qquad (4)$$

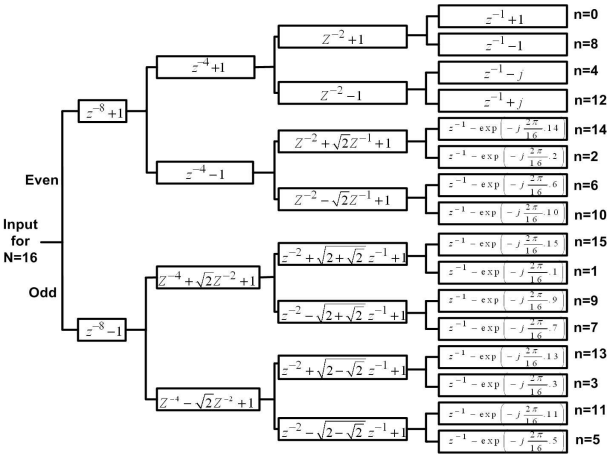

Fig. 1. Bruun's FFT Computation Architecture for length N=16 [8]

This factorization has a nice property that all the coefficients are real except in the last stage. Equation (4) replaces complex multiplications with real ones by requiring only real coefficients to be multiplied with complex inputs which needs only two real multipliers instead of four in a complex multiplication. This recomposition reduce the area required by corresponding hardware implementation to a significant extent. An example architecture of this is given in Fig. 1 [8].

Bruun's algorithm not only reduces the number of computations, which is same as that for a split-radix algorithm, but also maintains the regularity in the structure which is a primary requirement for hardware realization of a reconfigurable system like SDR. It also allows computation of cos/sin transform of an input signal with small changes at the last stage [8], which is essential for universal transform computation.

4 Comparison of Different FFT Architectures for Software Defined Radio

SDR requires flexible DSP algorithms which can be implemented effectively as/on a reconfigurable hardware like FPGAs. Main disadvantages of a reconfigurable system, however include extra hardware required for providing reconfigurability and features like dynamic and partial reconfiguration, etc. One more

disadvantage is that reconfigurable systems tend to be slower than dedicated systems due to extra switches/multiplexers and long interconnect lengths in the critical path of computation. This requires them to operate at higher frequencies which in turn increase the power consumption of the system.

Further, regular architectures can be partitioned easily in to sub-blocks and both classical and Bruun's FFT exhibit regularity in their architecture. In contrast, split-radix algorithm has low computational complexity, same as that of Bruun's FFT, but does not exhibit regularity. Therefore, split-radix FFT architectures are not considered for SDR [3].

In FFT computation, reconfigurability means a provision for change in operational length along with the option of computing other discrete transforms using the same architecture. Both classical and Bruun's FFT can be made to operate at variable lengths but Bruun's FFT has an added advantage that it can also compute different transforms like DCT, DST [8] etc. easily and with minimum changes in internal memory. This feature makes it an ideal candidate for converged systems. DIT and DIF FFT architectures have nearly the same performance, therefore further analysis is carried out for Bruun's and DIT FFT only.

4.1 NSR

In [9], Bruun's FFT was rejected only due to its higher NSR. To understand this, a comparison between NSR performance of Brunn's and DIT FFT is provided in this subsection. Signal power followed by noise power calculations for these FFT algorithms are explained below.

If an overall scaling factor $1/k$ is used, then the output signal power (SP) is given by [9]

$$SP = N/(3K^2)$$

Classical complex FFT has scaling factor $k = N$, whereas Bruun's FFT has scaling factor $k = N^2/8$ for $N{>}8$ [9]. For a complex input sequence, the output signal power for both FFT algorithms is as follows:
i) Signal power for classical FFT (SP_{cfft})

$$SP_{cfft} = 1/(3N) \tag{5}$$

ii) Signal power for Bruun's FFT (SP_{bfft})

$$SP_{bfft} = 64/(3N^3) \tag{6}$$

The reduction in signal power with $(1/N^3)$ in Bruun's FFT as compared to $(1/N)$ in classical FFT is a significant disadvantage for the Bruun's algorithm. Noise power introduced due to 'b' bit quantization at the output of a real multiplier is $\sigma_e^2 = 2^{-2b}/12$. Total output noise at each output node is equal to the sum of the noise propagated to that node with scaling which is done on per stage basis in FFT. This attenuates noise introduced at early stages by scaling introduced in later stages. Scaling is performed by shifters and because of truncation they also contribute noise in the computation.

i)Noise power in classical FFT (NP_{cfft})

$$NP_{cfft} = (4/3)\cdot 2^{-2b} \quad [10] \tag{7}$$

From (5) and (7) NSR value for classical FFT (NSR_{cfft}) can be written as:

$$NSR_{cfft} = 4N\cdot 2^{-2b} \tag{8}$$

ii) Bruun's FFT
In Bruun's FFT, first stage requires scaling by (1/2) while all the middle stages need scaling by (1/4) and no scaling is required for the last stage of computation.Due to this unequal scaling requirement, noise power is calculated as follows:

a) O/P noise power due to noise sources of first stage (NP_1):
A FFT computation consists of $v = \log_2 N$ stages. Using the procedure in [10],

$$NP_1 = (1/3)\cdot 2^{-2b}\cdot (1/2)^{(3v-7)} \tag{9}$$

Note that 2^v noise sources at first stage are connected to one particular output node of FFT.

b) O/P noise power due to noise sources in middle stages (NP_2):
Number of noise contributing butterflies and therefore noise sources contributing to overall noise at a particular output node decreases by 2 per stage as the computation progresses from one stage to the next stage. (NP_2) is given by,

$$NP_2 = (32/21)\cdot 2^{-2b} \tag{10}$$

c) O/P noise power due to noise sources at last stage (NP_3):

$$NP_3 = (1/3)\cdot 2^{-2b} \tag{11}$$

Total noise power for Bruun's FFT (NP_{bfft}) is

$$NP_{bfft} = NP_1 + NP_2 + NP_3$$

In this calculation we can neglect NP_1 for $v \geq 3$, so equation (10) and (11) gives,

$$NP_{bfft} = (39/21)\cdot 2^{-2b} \tag{12}$$

From (6) and (12), NSR value for Bruun's FFT (NSR_{bfft}) can be written as:

$$NSR_{bfft} = \frac{39}{448}\cdot 2^{-2b}\cdot N^3 \tag{13}$$

In this paper, a simplified noise analysis has been done for complex input sequence-based Bruun's FFT using a general procedure given in [10] as compared to that in [9]. From (8) and (12), we see that for the same bit width 'b' the NSR of Bruun's FFT increases in proportion to N^3 while that of DIT FFT is proportional to N. However this comparison is not fair as the hardware requirement of the two architectures is different.

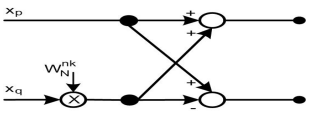

Fig. 2. Butterfly structure for Classical FFT computation

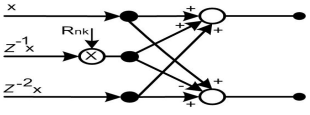

Fig. 3. Butterfly structure for Bruun's FFT computation

4.2 Hardware Complexity

Butterfly architectures used in classical and Bruun's FFT computation are shown in Figs 2 and 3 respectively. Here W_N^{nk} and R_{nk} represents complex and real valued coefficients for classical and Bruun's FFT butterflies respectively. A classical FFT butterfly require complex multiplication which is equivalent to four real multiplications and two real additions. Overall it requires 4 real multiplications and 6 real additions. Similarly a Bruun's FFT butterfly require to multiply a complex input with a real coefficient and therefore needs only two real multiplications. It overall needs two real multiplications and six real additions to get it's output. Now it is clear from Figs. 3 and 4 that Bruun's FFT will reduce the number of multiplications by half without adding any extra arithmetic block. Since an N point FFT consists of N/2 butterflies per stage, hardware complexity for different FFT algorithms is as follows:

i) Classical FFT

$$C^m_{cfft} = 2Nlog_2N, \quad C^a_{cfft} = 3Nlog_2N \tag{14}$$

ii) Bruun's FFT
In Bruun's computation, last stage consists of classical butterflies, therefore complexity will be given by,

$$C^m_{bfft} = Nlog_2N + N, \quad C^a_{bfft} = 3Nlog_2N \tag{15}$$

Here C^m and C^a represent the cost function for real multipliers, adders in FFT computation and subscripts $_{cfft}$, $_{bfft}$ represent classical and Bruun's FFT respectively.

In literature, complexity of a single multiplier is assumed to be 6-8 times the complexity of an adder [11]. In this analysis, authors assumed this number to be 5 based on their experiments. Bruun's FFT also uses classical butterfly at the last stage which has the complexity of $13N$ and is different from it's regular butterfly. Therefore, total hardware complexity for classical and Bruun's FFT in terms of real adders (C_{cfft}) and (C_{bfft}) using (14) and (15) is

$$C_{cfft} = 5 \cdot 2N \cdot \log_2 N + 3N \cdot \log_2 N \Rightarrow 13N \log_2 N \tag{16}$$

$$C_{bfft} = 5N(\log_2 N - 1) + 3N(\log_2 N - 1) + 13N \Rightarrow 8N \log_2 N + 5N \tag{17}$$

4.3 Comparison Between Hardware Complexity of Different FFT Architectures for Same NSR

In this subsection we first find the bit widths required by Bruun's FFT to get same NSR as DIT FFT. Let b1 denote the bit width required by Bruun's FFT to obtain the same NSR as that of DIT FFT for bit width b. Equating the NSR's of both FFTs and using (8) and (13) we have,

$$NSR_{cfft} = NSR_{bfft}$$

which simplifies as,

$$b1 = b + \log_2 N - 3 \tag{18}$$

Equation (18) indicates that for 8 ($= 2^3$) point Bruun's FFT, no extra bit is required for equal NSR. But Bruun's FFT requires 3 extra bits for 64-point FFT to achieve same NSR as classical FFT. In general hardware complexity of an adder increases linearly with the number of bits [12], for example, as in high speed parallel-prefix adders. In order that classical FFT will have more hardware complexity compared to Bruun's FFT the following condition must be satisfied: using equation (16), (17) and (18)

$$13N \log_2 N \cdot b \geq (8N \log_2 N + 5N) \cdot b1$$

$$13 \log_2 N \cdot b \geq (8 \log_2 N + 5) \cdot (b + \log_2 N - 3)$$

by simplifying above equation,

$$8(\log_2 N)^2 - (5b + 19) \log_2 N + (5b - 15) \leq 0$$

$$\log_2 N = \frac{(5b + 19) \pm \sqrt{(5b + 19)^2 - 32(5b - 15)}}{16} \tag{19}$$

for $b = 8$,

$$N \leq 2^{6.92} \Rightarrow N \leq 121$$

Usually in wireless communication applications, bit sizes of 10-16 or more have been used. As an example, using b=12 in (19) gives

$$N \leq 2^{9.26} \quad \Rightarrow \quad N \leq 613$$

This shows that up to 613≃512 point FFT with bit width b=12, Bruun's FFT will provide less hardware complexity than DIT FFT for same NSR. Therefore, Bruun's FFT can be used with higher bit length operations for compensating the high NSR value while maintaining it's low hardware complexity.

4.4 Timing Complexity

Timing complexity for Bruun's and classical FFT can be realized from the butterfly structures shown in Figs. 2 and 3. In Fig. 2, critical path delay is the summation of delays offered by one real multiplier (T_m) and two real adders (T_a) for a particular stage of classical FFT. Therefore, $\log_2 N$ stages in FFT structure will make total delay for DIT FFT (T_{cfft}) to be

$$T_{cfft} = (T_m + 2T_a)log_2 N$$

In Bruun's FFT, critical path delay per stage is sum of delays offered by one real multiplier and one real adder, so total delay T_{bfft} is given by

$$T_{bfft} = (T_m + T_a)log_2 N,$$

which is less than the critical path delay of classical FFT.

In Bruun's FFT, operations will be done with higher bit length for same NSR but that will add just one more level of basic gate computation in delay of adders/multipliers. But a new adder attached sequentially in classical FFT's critical path will increase the delay to at least 5-7 levels of basic gates computation [12]. Therefore Bruun's FFT has less computational delay as compared to classical FFT which reduces linearly with the number of stages. Thus it may be claimed that Bruun's FFT architecture is an optimal FFT architecture for reconfigurable and converged hardware systems like SDR.

5 Hardware Implementation Results for Different FFT Architectures and Comparisons

Both classical and Bruun's FFT architectures have been coded in Verilog HDL, simulated and synthesized on Virtex series FPGA, v50bg256 (speed grade -6) from Xilinx using Mentor's Leonardo Spectrum. Butterflies in Fig. 2 and 3 are basic components of FFT and they determine the hardware requirement and also the critical path delay of the design. Therefore we have implemented and compared hardware architecture of these butterflies for different bit lengths. Coefficient length is fixed for a constant NSR and only precision of arithmetic operations needs to be changed. Thus one operand in multiplier will have constant bit length.

Table 1 provides a comparison of area and delay performance of classical and Bruun's butterfly architectures. It can be seen from the table that Bruun's butterfly consumes less number of LUTs (and therefore less area) for 10 bit operation as compared to 6 bit operation in classical butterfly, for a comparable NSR. This is due to the fact that two 2-bit functions will take 2 LUTS while a single function with 4 bits takes only one LUT to implement. Table 1 also shows that delay in Bruun's butterfly is very less compared to classical butterfly for reasonable bit lengths which is due to reasons explained in subsection 4.4. This reduction in delay allows designer to operate FFT with lower frequency which reduces the dynamic power consumption.

Table 1. A Comparison of Hardware and Timing performance of Butterfly Architectures for different bit lengths

Synthesis	Parameters	Classical Butterfly		Bruun's Butterfly			
Device	of comparison	6 bits	8 bits	6 bits	8 bits	10 bits	11 bits
FPGA	Area(LUTs)	134	258	76	88	116	138
(Virtex)	Time(ns)	12.1	12.5	10.5	10.7	10.9	11.1

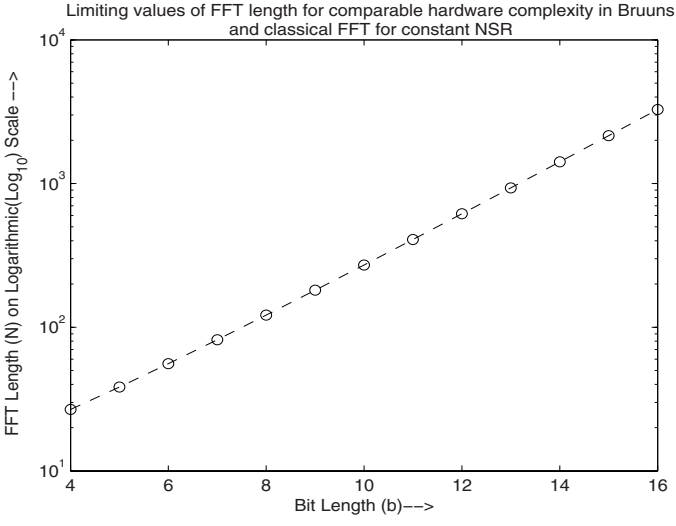

Fig. 4. Limiting values of FFT length N ($\log_{10}$ scale) for comparable hardware complexity and constant NSR in Bruun's and classical FFT, according to equation (19)

In Fig. 4 straight line shows the limiting values of FFT length (N) on $\log_{10}$ scale for which classical FFT will have more hardware complexity as compared to Bruun's FFT for a particular bit length (b) and same NSR according to (19). Bruun's FFT will have more hardware complexity for values of N greater than or equal to the limiting values for maintaining same NSR. This clearly shows that for practical operational bit lengths in FFT computation, N should be high to invert the low hardware complexity advantage offered by Bruun's algorithm.

A programmable shifter at the output will adjust the final output bit length according to requirement of next computing block's input in the system. Bit length adjustment adds to reconfigurability in SDR as different modules may require different bit lengths for their operation. This block can also perform re-ordering of FFT outputs and also helps in increasing the SNR value at the output by doing appropriate shifting. Moreover, it can be well adjusted with Bruun's FFT due to it's low hardware and timing complexity.

Our analysis covers nearly all basic and important properties required for SDR. VLSI implementation results show that Bruun's FFT requires less area and also has less delay which in turn reduce both static and dynamic power

consumption. Thus it may be suggested that Bruun's FFT provides a better modular and universal architecture.

6 Conclusion

This paper provides an insight in to an important problem of finding and analyzing optimal FFT architectures for SDR. The authors have shown that Bruun's FFT can operate at larger bit lengths that are practical for a SDR, give better hardware performance in terms of area and delay compared to the classical FFT, while maintaining comparable NSR. It is suggested that Bruun's FFT may be preferred over classical and split-radix FFT in applications such as SDR.

References

1. Mitola, J.: The Software Radio Architecture. IEEE Communications Magazine, 26–38 (May 1995)
2. Tuttlebee, W.: Evolution of radio systems into the 21st century. In: Proc. IEE Int. conf. on Radio receivers an associated systems (1995)
3. Reed, J.H.: Software Radio: A Modern Approach to Radio Engineering. Prentice Hall, Englewood Cliffs (2000)
4. Lin, Y.T., Tsai, P.Y., Chiueh, T.D.: Low-power variable-length fast Fourier transform processor. In: IEE Proceedings-Computers and Digital Techniques (July 2005)
5. Hung, C.-P., Chen, S.-G., Chen, K.-L.: Design of an efficient variable-length FFT processor. In: International Symposium on Circuits and Systems(ISCAS)'04 (2004)
6. Britanak, V., Rao, K.R.: Two-Dimensional DCT/DST Universal Computational Structure for 2m X 2n Block Sizes. IEEE Transaction on Signal Processing 48(11), 3250–3255 (2000)
7. Tell, E., Seger, O., Liu, D.: A converged hardware solution for FFT, DCT and Walsh transform. In: Seventh International symposium on Signal Processing and Its Applications, pp. 609–612 (2003)
8. Bruun, G.: Z-Transform DFT filters and FFT's. IEEE Trans.Acoust. Speech, Signal Processing ASSP-26, 56–63 (1978)
9. Storn, R.: Some Results in Fixed Point Error Analysis of the Bruun-FFT Algorithm. IEEE Transaction on Signal Processing. 41(7) (1993)
10. Oppenheim, A.V., Schafer, R.: Digital Signal Processing.Pearson Education (2004)
11. Parhami, B.: Computer Arithmetic: Algorithms and Hardwrae Designs. Oxford University Press, New York (2000)
12. Zimmermann, R.: Binary Adder Architectures for Cell-Based VLSI and their Synthesis, PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, Hartung-Gorre Verlag (1998)