Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 2, February 2014, pg.308 – 315

RESEARCH ARTICLE

VLSI Architecture for Implementing Kaiser Bessel Window Function Using Expanded Hyperbolic CORDIC Algorithm

M.Mohana Arasi (Assistant Professor), J.Anand (P.G.Scholar),
P.Bharat (P.G. Scholar), J.Anbu Selvan (P.G.Scholar)
Department of Electronics and Communication Engineering,
Bannari Amman Institute of Technology, Sathyamangalam-638 401, India
Email ID: mass.anand14@gmail.com, bharat.p.9019@gmail.com, anbuselvanjs@gmail.com

Abstract - Windowing techniques have been widely used for preprocessing of samples before fast Fourier transform (FFT) in real time spectral analysis to minimize spectral leakage and picket fence effect. Among all popular window functions, Kaiser-Bessel window is an obvious choice for its better spectral characteristics. In this paper, CORDIC (CO-ordinate Rotation Digital Computer) based VLSI architecture for implementing Kaiser-Bessel window has been proposed for real time applications. The parallel pipelined technique has been adopted for the present design to ensure high throughput. Various architectural design and implementation issues have been discussed. The physical synthesis for ASIC implementation of proposed architecture using Synopsys design compiler(Design Vision) and commercially available 0.18 µm CMOS yields the core area of 52 mm2 and worst case dynamic power of 890 mW at an operating frequency and voltage of 400 MHz and 1.8 V respectively.
Keywords: VLSI architecture, Kaiser-Bessel windowing function, CORDIC

1 Introduction

Windowing plays a key role in signal processing applications such as spectral analysis [1–4], where it refers to the process of modifying the input samples before utilizing them for computing discrete Fourier transform (DFT), such that undesired effects in spectral domain due to data truncation i.e., spectral leakage [5–7] and picket fence effect [8] can be minimized. Thus, the selection of appropriate window function for desired applications is based on their spectral characteristics [8–10]. Since windowing is used before the FFT [4, 8] and STFT [10, 11] for digital spectral analysis system in real time application, and plenty of literatures presenting flexible [12, 13] and high speed architectures [14, 15] for implementing fast Fourier transform (FFT) are available. But none of them, to the knowledge of the authors, explicitly described efficient implementation of the window functions to meet the specification of FFT in terms of

speed and flexibility in new trend of real time applications, except only recently, attempt has been made [16] to implement a set of popular window functions in hardware using CORDIC [17] arithmetic units. Except popular window functions such as Hanning, Hamming and Blackman, Kaiser-Bessel window [8] is also an obvious choice for better spectral characteristics and this is widely used in signal processing [5, 18], communication [19] and biomedical signal processing [20, 21]. Thus it becomes essential to have high throughput hardware realization for Kaiser-Bessel window functions to match the speed of the FFT architecture in order to develop an efficacious real time spectral analysis system. Depending on the intended application, the system designer can choose from a handful of window length and time-bandwidth product to satisfy the tradeoff between various parameters [8] such as accuracy of amplitude and spectral purity.

This is needless to mention all real time applications which require variable length FFT processor e.g. the STFT (Short Term Fourier Transform) is generally computed using variable length FFT processor which demand the variable length (flexible or adaptive) windowing architecture for preprocessing of signal before FFT. In this case, flexible windowing architecture is an obvious choice over ROM based or on chip RAM based implementation to compute windowing coefficients on the fly. To the knowledge of authors, there is no flexible VLSI/ASIC architecture for computing Kaiser-Bessel windowing coefficients available in the literature. In this context, authors in this paper have proposed CORDIC based flexible VLSI architecture for Kaiser-Bessel windowing to compute the coefficients on the fly and to meet the real time specifications in terms of throughput with little compromising for power and area compared to ROM based implementation. However, the optimization of the area and power using different circuit level techniques can be adopted which is not in the scope of this paper. The proposed architecture is based on zeroth order Bessel function [22] and expanded hyperbolic CORDIC [23] which are presented in subsequent section. The ASIC implementation of the proposed architecture using commercially available 0.18 μm CMOS technology shows that the design can work at the maximum clock speed of 400 *MHz*.

Rest of this paper is organized as follows: Section 2 discuss the existing system. Detailed design of the proposed architecture has been described in Section 3. Section 4 presents the simulation results of the proposed architecture using Xilinx tools, and Section 5 provides the concluding remarks highlighting the features of this work.

2. Existing System

The basic digital spectral analysis system based on fast Fourier transform (FFT) is shown in Fig. 1, Here Kaiser-Bessel window is being used for preprocessing of samples before the FFT. In Fig. 1, the windowing processor can be represented for real time application relating input(si) and output(so) as in Eq. 1.

$$so(r) = wk(n)si(N + r - n) \qquad (1)$$

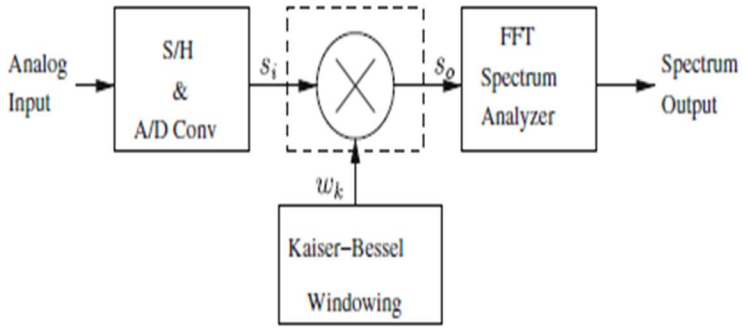


Figure 1 Blocks for spectral analysis system

Here *wk* represents Kaiser-Bessel window coefficients, *r* indicates present time variable and *n* is discrete index ranging from 0 to *N*, where *N* is the windowing or observation length.

2.1 Kaiser-Bessel Window

Here the mathematical details with some useful observation are presented for Kaiser-Bessel window. The mathematical expression for Kaiser-Bessel window is given by the following equation,

$$w_k(n)=\begin{cases}\dfrac{I_0\left(\beta*\left\{1-\left[\frac{2n}{N}\right]^2\right\}^{\frac{1}{2}}\right)}{I_0(\beta)} & \text{for} -\left(\dfrac{N}{2}-1\right)\le n\le\left(\dfrac{N}{2}-1\right)\\ 0 & otherwise\end{cases} \quad (2)$$

where I0(•) represents *zero*th order Bessel function [22] that can be approximated by the following expression:

$$I_0(x)=\frac{e^x\left(1+\frac{1}{8x}\right)}{\sqrt{2\pi x}} \quad (3)$$

As a study, this Kaiser-Bessel window using Bessel function as in Eq. 3 has been simulated in Xilinx for different values of β (time-bandwidth product) including Blackman window. It is observed that Blackman window can be realized with Kaiser-Bessel window by setting the value of β equal to 8.5. i.e. the same architecture can be used for both Kaiser and Blackman window for different desired applications.

2.2 Expanded Hyperbolic CORDIC Algorithm and Architecture

As presented in previous subsection, the hyperbolic CORDIC is an obvious choice to compute and implement transcendental functions in Kaiser-Bessel window. Authors have not presented the details of conventional hyperbolic CORDIC in this paper due to space constraints, however readers may refer to [17, 24–26] for more details on conventional hyperbolic CORDIC arithmetic unit. The main limitation of conventional hyperbolic CORDIC is the range of convergence i.e. for rotation mode of conventional hyperbolic CORDIC, the maximum input hyperbolic angle $|z0|max \simeq 1.182$ for m sufficiently large. Similarly for vectoring mode, the maximum accumulated hyperbolic angle $|zm|max \simeq 1$. This range of convergence mentioned here restricts the use of conventional hyperbolic CORDIC to compute transcendental function to implement Kaiser-Bessel windowing. Hence the expanded hyperbolic [23] is used to meet the range of convergence for the proposed architecture. The expanded hyperbolic CORDIC has been optimized for the hardware, timing and appropriate iterations has been chosen for various

transcendental functions to meet the range of convergence. The more details of expanded hyperbolic CORDIC with its range of convergence have been presented here.

3. Proposed Architecture for Kaiser-Bessel Window

This section, at the beginning, presents the basic flow for computation of Kaiser-Bessel window, then describes the proposed architectural design with different constraints for fixed point hardware implementation and subsequently major blocks are described in detail. The Kaiser-Bessel window as in Eq. 2 requires the computation of Bessel function kernel in both numerator and denominator. The argument to the Bessel functions in the numerator of Kaiser-Bessel window. Now

the basic flow of Kaiser-Bessel window computation is as follows: At the beginning, denominator of Kaiser-Bessel window is computed for given β using Bessel function kernel as in Eq. 3 and then the same kernel is used to compute Bessel function values for the sequence of arguments called Bessel function arguments in the numerator of the Kaiser- Bessel window as in Eq. 2. Each computed sequence is divided by computed Bessel function value of denominator to yield Kaiser-Bessel window coefficient for given β and N, and further multiplies with input samples for windowing time domain output sequences which further processed using FFT for digital spectrum analysis. The block diagram of the proposed architecture is shown in Fig. 2 and all the blocks are described subsequently. The operation of this proposed architecture is that the denominator of the Kaiser-Bessel window is computed first through lower path of multiplexer and demultiplexer, and latch the resultant value of denominator at *REG* 2 for further processing with numerator values of this window. Here finite state machine(FSM) is used for switching from lower to upper path of the mux and demux as in Fig. 2, and *load* is a control signal used to initialize for specify value of N and β. The Bessel arguments generator(BAG) block computes the sequence of arguments for Bessel function in the numerator of Kaiser Bessel window. Now this is clear that the Kaiser-Bessel window involves the computation of transcendental functions which requires floating point or high precision fixed point arithmetic units for hardware implementation. But, in this paper we have emphasized to design the proposed architecture for VLSI implementation with finite machine precision. Thus the choice of machine precision is crucial in this case, for example, as previously discussed, the maximum value of arguments to Bessel function for both numerator and denominator path is the value of β. So exponential computation of this value is so large that the fixed point hardware implementation restricts the range of the values to be computed. Here we have presented a generic machine precision(word length) format for real values. i.e., a format *tb*, *db* in twos complement form has been chosen to represent the machine precision for each blocks used for implementation, here *tb* and *db* stand for total number of bits in a word and number of bits after binary point respectively. The selection of exact word length precision of each block is discussed in proceeding subsections.

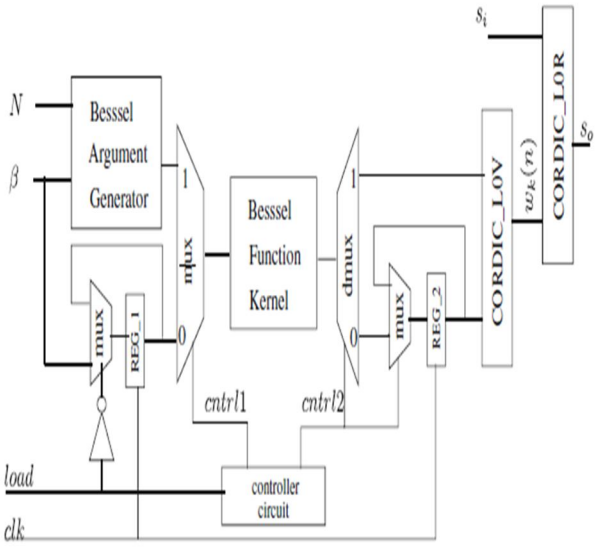


Figure 2 Proposed Kaiser-Bessel windowing Architecture.

3.1 Bessel Argument Generator(BAG) Architecture

The sequence of Bessel argument $A(n) = \sqrt{\beta^2 - \left(\frac{2n\beta}{N}\right)^2}$ is computed with given β, N and n, where n ranges from $-\left(\frac{N}{2}-1\right)$ to $\left(\frac{N}{2}-1\right)$ Now changing index n to another time index variable p, which varies from 0 to $N-2$. Now $A(p) = \sqrt{\beta^2 - m^2\left(p - \frac{N}{2} + 1\right)}$ here the sequence $m(p)$ is generated using the sequence generator equation as follows;

$$m(p) = m(p-1) + \frac{\beta}{N_1} \quad (4)$$

where $m(-1) = -\beta$ and $N1 = N/2$. The architecture to generate $A(p)$ is shown in Fig. 3.Here the encoder is designed as $2v = N1$ where v is an integer, and shifter is used to divide the value of β by $N1$. Loadable down counter is used to initialize after each sequence of $m(p)$. The *N min* 2 is decrement by 2, which is loaded to loadable down counter. In this paper, LHS and RHS are referred to left hard shift and right hard shift respectively. In Fig. 3, *CORDIC EHV* is an expanded hyperbolic CORDIC used in vectoring mode. We have constrained the proposed design for maximum value of β to 15 and hence the maximum value of $A(p)$ to be 15. So word length for *CORDIC EHV* is chosen as 32, 27 because of following reason; The highest value in this format represented is $(15.99999999)_{10}$. Thus the exponential output to be computed in next consecutive block can be accommodated within 32-bits machine precision. Other than this format is not suitable for our implementation. For example; If we choose 32, 26 format, then the maximum value in this format can be $(31.9999999)_{10}$ and hence exponential output of this value cannot be accommodated within 32-bit machine precision, which may require higher machine precision to compute exponential function. In other end, if we choose 32, 28 format then the range of β has to be restricted to $(7.99999999)_{10}$. In BAG

architecture, *CORDIC EHV* requires ten negative indexed iterations and two repeated iterations, taking a total of 69 stages to complete iterations with scale factor compensation. Thus BAG architecture with an additional adder stage as shown in Fig.3 has a total latency of 70 clock cycles.

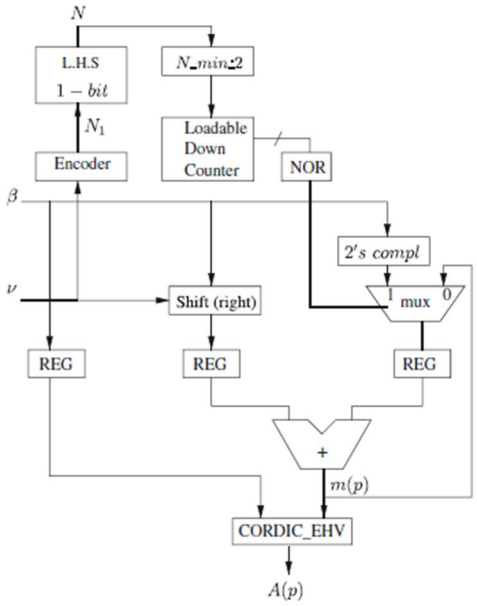
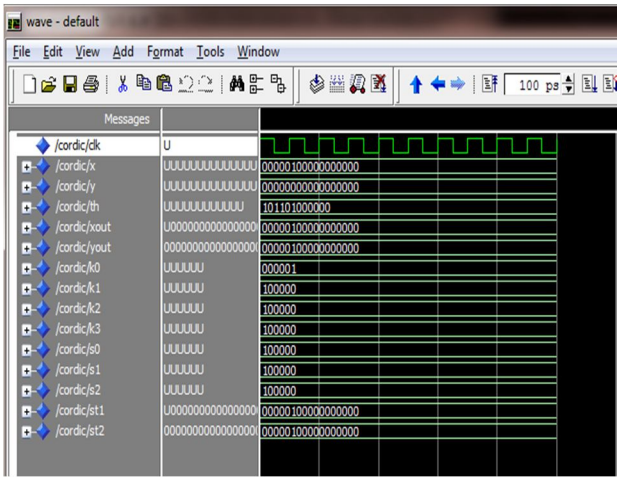


Figure 3 BAG architecture.

4. Simulation Results



5. Conclusion

The feature of this proposed architecture is flexibility in terms of varying window length(N) in the range of 8 to 215 and computing for time bandwidth product(β) up to a maximum value of 15, which fulfills the required specification for most of the applications. Although this design restricts the range of β to 15, but in same design spelling, this proposed architecture can be extended for the computation for higher value of β. Synthesis result shows that this implementation can be used in today's real time applications to meet up to maximum

throughput of 400 Msamples/s at the cost of latency of 170 clock cycles. This latency may be reduced using modified expanded hyperbolic CORDIC [26] at the extra silicon cost. This proposed architecture is suitable to use before FFT for real time digital spectral analysis system and also can be used for filtering applications [19] to compute real time Kaiser-Bessel window filter coefficients. The RTL logic of this proposed architecture is technology J Sign Process Syst independent and can be implemented on suitable FPGA for real time application as well. The detail error analysis for this proposed implementation of kaiser-bessel windowing is to be reported in future communication.

References

1. Zivanovic, M., & Carlosena, A. (2006). On asymmetric analysis windows for detection of closely spaced signal components. Mechanical Systems and Signal Processing, 20(3), 702– 717.
2. Muthuswamy, J., & Thakor, N.V. (1998). Spectral analysis methods for neurological signals. Journal of Neuroscience Methods, 83(1), 1–14.
3. Banerjee, A., Dhar, A.S., Banerjee, S. (2001). FPGA realization of a CORDIC based FFT processor for biomedical signal processing. Microprocessors and Microsystems, 25(3), 131–142.
4. Sansaloni, T., Perez-Pascual, A., Torres, V., Almenar, V., Toldo, J.F., Valls, J. (2008). FFT spectrum analyzer project for teaching digital signal processing with FPGA devices. IEEE Transactions on Education, 50(3), 229–235.
5. Barros, J., & Diego, R.I. (2006). Effects of windowing on the measurement of harmonics and inter harmonics in the IEC standard framework. In Instrumentation and measurement technology conference (IMTC2006) (pp. 2294–2299).
6. Xu, F. (2006). Algorithm to remove spectral leakage, close-in noise, and its application to converter test. In Instrumentation and measurement technology conference (IMTC2006) (pp. 1038–
1042).
7. Prabhu, K.M.M., & Bhoopathy Bagan, B.K. (1989). Variable parameter window families for digital spectral analysis. IEEE Transactions on Acoustics, Speech and Signal Processing, 37(6), 946–949.
8. Higgins, R.J. (1990). Digital signal processing in VLSI. Englewood\ Cliffs: Prentice Hall.
9. Prabhu, K.M.M., & Agrawal J.P. (1978). Selection of data windows for digital signal processing. In Proceedings of the IEEE international conference on acoustics, speech, and signal processing (IEEE-ICASSP-1978) (pp. 79–82).
10. Yuegang,W., Shao, J., Hongtao, X. (2007). Non-stationary signals processing based on STFT. In The eighth international conference on electronic measurement & instruments (pp. 301–304).
11. Zhang, S., Yu, D., Sheng, S. (2006). A discrete STFT processor for real-time spectrum analysis. In Proceedings on circuits and systems, IEEE Asia-Pacific conference, (IEEE-APCCAS6) (pp. 1943–1946).
12. Cheng-Ying, Y., Sau-Gee, C., Jen-Chuan, C. (2006). Efficient CORDIC design for multimode OFDM FFT. In Proceedings on acoustics, speech and signal processing, (ICASSP06) (pp. 1036– 1039).
13. Shuenn-shyang, W., & Chien-Sung, L. (2008). An area-efficient design of variable length fast Fourier transform processor. Journal of Signal Processing Systems, 51, 245–256.
14. Zhong, K., Zhu, G., He, H. (2003). A single-chip, ultra high speed FFT architecture. In Proceedings on ASIC 5th international conference.
15. Samad, A., Ragoub, A., Othman, M., Shariff, Z.A.M. (1998). Implementation of a high speed fast Fourier transform VLSI chip. Microelectronics Journal, 29(11), 881–887.
16. Ray, K.C., & Dhar, A.S. (2006). CORDIC based unified VLSI architecture for implementing window functions for real time spectral analysis. IEE Proceedings - Circuits Devices and Systems, 153(6), 539–544.
17. Haviland, G.L., & Tuszynsky, A.A. (1980). A CORDIC arithmetic processor chip. IEEE Transactions on Computers, 29(2), 68–79.
18. Arulalan, M.R., Jamadagni, H.S., Rao, A. (2008). Novel Window functions for digital filters. In Fifth international conference on information technology: new generations (pp. 1184–1185).
19. Hou, W., & Kwon, H.M. (1999). Complementary filter design for testing of IS-95 Code Division Multiple Access wireless communication systems. IEEE Transactions on Instrumentation and Measurement, 48(1), 34–38.
20. Gnecchi, J.A.G., Garcia, J.C.H., Alvarado, J.D. (2007). Auxiliary neuro feedback system for diagnostic of attention deficit hyperactivity disorder. In Electronics, robotics and automotive mechanics conference (pp. 135–138).

21. Bergen, S.W.A., & Antoniou, A. (2005). Application of Parametric window functions to the STDFT Method for Gene Prediction. In Proceedings on communication, computers and signal processing, (IEEE-PACRIM05) (pp. 324–327).
22. Rade, L., & Westergren, B.B. (1992). Beta mathematics hand books (3rd Ed.). Cleveland: CRC Press.
23. Hu, X., Harber, R.G., Bass, S.C. (1991). Expanding the range of convergence of the CORDIC algorithm. IEEE Transactions on Computers, 40(1), 13–21.
24. Padala, S.K., & Prabhu, K.M.M. (1999). Pipelined CORDIC processors for generating Gaussian random numbers. Signal Processing, 72(3), 177–181.
25. Hu, Y.H. (1992). CORDIC-based VLSI architectures for digital signal processing. IEEE Signal Processing Magazine, 9 (3), 16–35.
26. Boudabous, A., Fahmi Ghozzi, M., Kharrat, W., Masmoudi, N. (2004). Implementation of hyperbolic functions using CORDIC algorithm. In Proceedings on international conference on microelectronics, (ICM04) (pp. 738–741).
27. Harris, D. (2003). A taxonomy of parallel prefix networks. In IEEE conference on signals, systems, and computers (pp. 2213– 2217).
28. Hu, Y.H. (1992). The quantization effects of the CORDIC algorithm. IEEE Transactions on Signal Processing, 40(4), 834–844.
29. Sung, T.U., & Hsin, H.C. (2007). Fixed point error analysis of CORDIC arithmetic for special-purpose signal processing. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, A(9), 2006–2013.
30. Park, S.Y., & Cho, N.I. (2004). Fixed-point error analysis of CORDIC processor based on the variance propagation formula. IEEE Transactions on Circuits and Systems-1: Regular Papers, 51(3), 573–584.

