Sigma Delta Modulator with Improved Performance through Evolutionary Algorithm

T. K. Bandopadyay [1], Manish Saxena[2], Raghav Shrivastava[3]

[1] Bansal Institute of Science and Technology,
Kokta, Anand Nagar, Bhopal (M.P.)-462021, India

[2]Bansal Institute of Science and Technology,
Kokta, Anand Nagar, Bhopal (M.P.)-462021, India
manish.saxena2008@gmail.com

[3] Bansal Institute of Science and Technology,
Kokta, Anand Nagar, Bhopal (M.P.)-462021, India
raghav_agar@rediffmail.com

Abstract: ***Sigma–delta modulation is the most popular form of analog-to-digital conversion used in audio applications. The sigma delta conversion technique has been in existence for many years, but recent technological advances now make the devices practical and now area of usage becoming very large. The main thing of these converters is that they are the only low cost conversion method which provides both high dynamic range and flexibility in converting low bandwidth input signals. In this paper we explain how performance is improved using genetic algorithm (GA). In particular, the proposed converter makes use of low-distortion swing suppression SDM architecture which is highly suitable for low oversampling ratios to attain high linearity over a wide bandwidth. GA- based search engine is a stochastic search method which can find the optimum solution within the given constraints.***

Keywords: over sampling, analog to digital convertor, genetic algorithm, crossover, mutation.

1. Introduction

Sigma delta modulators are the most suitable Analog-to-Digital converter (ADC) topologies for digitizing with high-resolution analog signals. With these architectures, a resolution up to 19–21 bits can be reached using standard Integrated Circuit (IC) technologies. Designers use sampling rates much higher than the Nyquist rate, typically higher by a factor between

8 and 512, and utilizing all preceding input values, they generate each output. The most popular approach is based on a sampled-data solution with

switch capacitor implementation. For this reason these features make the solutions very attractive

for a number of applications. For instance, they have gained increasing popularity in audio applications, in receivers for communication systems, in sensor interface circuits, and in measurement systems. Because of the diversity of architectures implementing converter, it cannot exist a generic model for all ADCs. Each architecture implementation of converter requires its own model. Genetic algorithms (GAs) have been successfully applied to a wide range of optimization problems including design, scheduling, routing, and signal processing. In sigma-delta (ΣΔ) modulator design [3], GA can be effectively used to optimize the scaling coefficients in order to achieve the desired signal-to-noise ratio. ΣΔ modulators were traditionally used for audio applications where the over-sampling ratio is high and a high resolution can be achieved with a realizable clock frequency. Recently ΣΔ modulators are exploited for wideband applications like WLAN, thus preventing the excess increase in the OSR and resorting to higher order modulators. Higher order modulator with low OSR requires the optimization of system parameters in order to achieve the required dynamic range. The requirements that the

ADC has to fulfill are set by both the standard characteristics and the receiver architecture.

2. Sigma delta modulator architecture

The sigma delta conversion technique [1][2][6] has been in existence for many years, but recent technological advances now make the devices practical and their use is becoming widespread. The converters have found homes in such applications as communications systems, consumer and professional audio, industrial weight scales, and precision measurement devices. The key feature of these converters is that they are the only low cost conversion method which provides both high dynamic range and flexibility in converting low bandwidth input signals. This application note is intended to give an engineer with little or no sigma delta background an overview of how a sigma delta converter works. The comparator, just like in the analogue version, decides whether its input value is higher or lower than a certain threshold and puts out a single bit signal, the bit stream. BTW, due to the preceding integrator this threshold is arbitrary. In order to obtain the bit stream in the digital modulator it is sufficient to strip off the comparator's input MSBit. A 1-Bit DAC [5] can output two different values only.

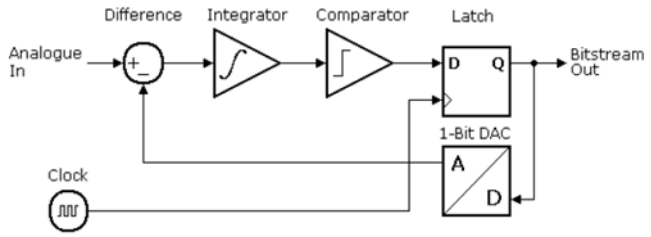

Figure 1: General block diagram of SDM

They are termed VRef- and VRef+ and those of the 1-Bit DDC (digital-to-digital converter) DRef- and DRef+ correspondingly. In both types of modulators they determine its input range. Note that in this example the clock rate, which here is also the sample rate, is 64 times higher than the frequency of the input signal. Delta sigma converters require much more in order to produce a sufficient number of bit stream pulses. Conventional converters require a sample rate of more than twice the highest input frequency.

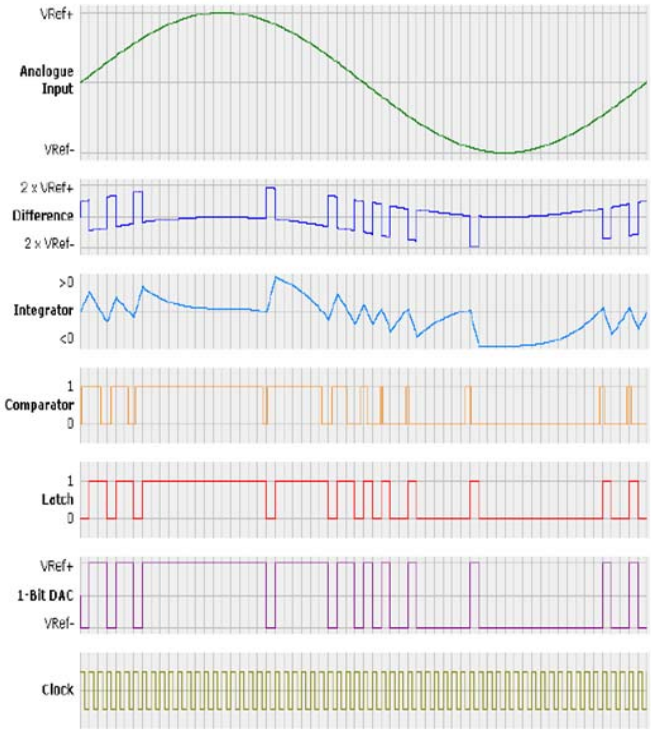

Figure 2: Signals within a First Order Analogue Modulator

It is obvious: The more bit stream pulses are produced the better is the approximation of the input signal by the average bit stream.

Once again: The average (low pass filtered) bit stream never (!) exactly represents the input signal. It is always (!) superimposed by some kind of noise.

One way to reduce this noise is to further increase the clock rate. Due to the sampling theorem the sampling rate must be higher than twice the maximum input frequency. Any further increase is called "oversampling rate". Example: Assume an audio signal with a bandwidth of up to 20 kHz (and probably slightly more). A typical sampling rate (for DAT etc.) is 48 kHz. In a typical delta sigma converter the clock frequency (which usually is also the sample rate) will be 64 x 48 kHz = 3072 kHz. This is equal to an oversampling rate of 64. In the example above (Figure 4) the clock frequency is 64 times higher than the frequency of the input signal. This means that the oversampling rate must be less than 32 for the given input frequency. (I don't know why only oversampling rates in the form of 2n are actually implemented. In my opinion any other form of this factor should be possible, too.) Another - and better - way to reduce the noise is to use a higher order delta sigma modulator. Bit streams produced by higher order modulators produce less noise at the low pass filter outputs. Normally this noise is random. First order modulators show some strong frequencies in the power spectrum (non-random noise or residual tones), which is disadvantageous. If the input signal is close to the limits of the input range this effect is worst with first order modulators.

3. Genetic Algorithm

In a genetic algorithm [8], a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, is evolved toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm[9][10]. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. Genetic algorithms find application in bioinformatics, phylo-genetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields [7].

A typical genetic algorithm requires:

1. A genetic representation of the solution domain,

2. A fitness function to evaluate the solution domain.

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear

chromosomes and trees is explored in gene expression programming.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g., computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions (usually randomly) and then (usually) to improve it through repetitive application of the mutation, crossover, inversion and selection operators. Figure 3 shows GA:

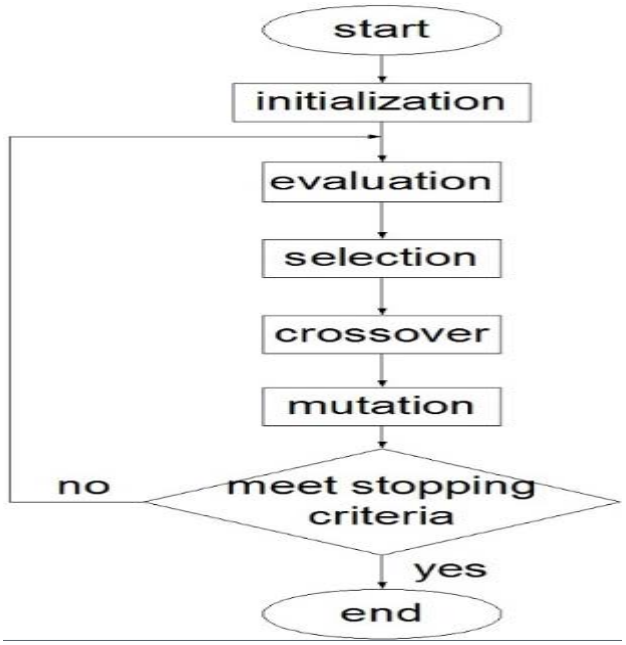

Figure 3: Flow chart of genetic algorithm

4. Improving SNDR of 2nd order SDM

A High SNDR means better Performance, what if we got about 42dB SNDR of 2nd order SDM then we don't require higher order SDM likewise we can't use 3rd order SDM in audio application, for audio application we have to look 5th order SDM and we also concern with low price, low complexity. So this project is to improve SNDR of 2nd order SDM. Generally SNDR of 2nd order SD modulator with standard parameters is measured about 36-38 dB. Our work is to improve it near about 4-5 dB. A simulation is to be performed of SD modulator, So that we can get better output with high SNDR. MATLAB/SIMULINK is to be used for the test. Using the sigma delta tool box we built the model of sigma delta 2nd order modulator shows [4] in figure 4. GA algorithm toolbox is to be used for the optimization Jitter input sine wave is provided as input signal in over system. Here switch makes input nonlinear. Thermal noise is added to the signal. The amplifier gain coefficient a1 & a3 are given to differential comparator. The output from comparator is added with white noise. The noise signal is given to the integrator. Amplifier gain coefficient a2, a4 given to second differential comparator. The output of comparator is provided to integrator. The digital output from comparator is taken out as Y out and analog output of convertor is observed for enhance the performance and feedback to the system. Generally gain coefficient of amplifier in modulation application we make constant and independent, in studies we find that if an optimized value of gain coefficient used modulation system then results would be more refined and improved. To find an optimized value of gain coefficient requirement of an optimizing iteration process rises.

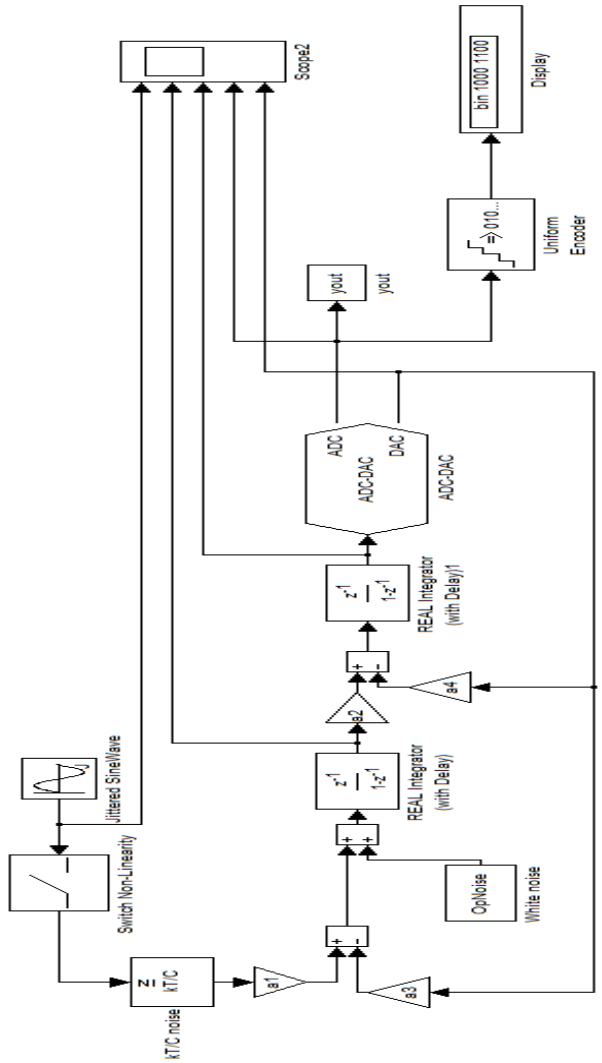

Figure 4: Block diagram of second order SDM with imperfections.

So here we are using genetic algorithm to find a value for better output and improved SNDR. Now we apply genetic algorithm of gain coefficient a1, a2, a3, and a4. After iteration we find an optimized value of gain coefficient, at that value SDM gives best results and SNDR also improved.

5. Results

The PSD of conventional and modified 2nd order Sigma Delta Modulator are shown below in Fig. 5 Measured SNDR for conventional design is 36.5 dB. And after apply genetic algorithm SNDR is 42.7dB shows in Fig.6 PSD of 2nd order SDM.

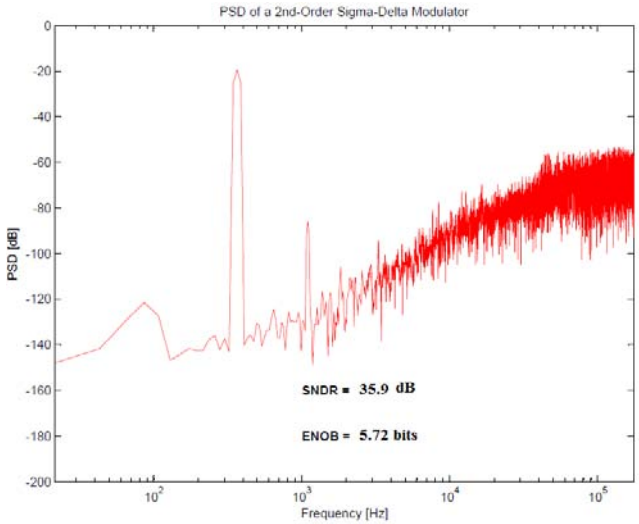

Figure 5: SNDR of conventional 2nd order SDM

Table shows result of iterations. We use 20 generation in our genetic algorithm so 20 iteration executed and stall generation fixed on value 2.

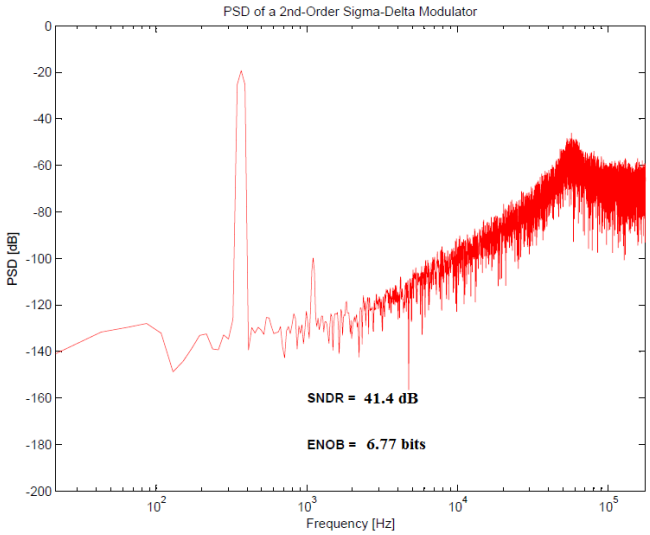

Figure 6: SNDR after apply genetic algorithm

```
****************************************
Modulator Cofficients ==> b = 0.5
SNDR = 36.4537
optimizing b using GA
```

Generation		Best f-count	Mean f(x)	Stall f(x)	Generations
1	10	63.05	63.82	0	
2	15	63.47	63.81	1	
3	20	61.25	63.04	0	
4	25	61.27	62.88	1	
5	30	61.14	62.97	0	
6	35	60.85	61.84	0	
7	40	61.15	61.44	1	
8	45	60.68	61.82	0	
9	50	61.21	61.34	1	
10	55	61.24	62.25	2	
11	60	61.04	61.79	0	
12	65	61.17	62.69	1	
13	70	56.89	60.95	0	
14	75	57.06	60.81	1	
15	80	57.12	59.42	2	
16	85	56.7	58.73	0	
17	90	56.89	59.32	1	
18	95	56.94	59.76	2	
19	100	56.39	57.62	0	
20	105	56.89	57.57	1	

```
Optimization terminated: maximum number of generations exceeded.
****************************************
Modulator Cofficients ==> b = 0.91476
SNDR =  41.418 dB
```

6. Conclusion

In this paper we have discussed methods of operation, design, and use of sigma–delta modulators. Although we considered the design of a sigma–delta modulator as used for analog-to-digital conversion, the results derived here could easily be generalized. Because low complexity and low cost are critical requirements so we tried to improve results of 2nd order SDM. SNDR of 3rd order SDM has SNDR about 41-43dB but cost and complexity is high. As a result, the proposed receiver has much lower complexity but is still able to attain the same performance.

7. Acknowledgment

Dr. T.K. Bandopadyay one of the authors is indebted to Dean (R&D) of BIST Bhopal for giving permission for sending the paper to the journal. HOD of E.C. Dept. Manish Saxena is also thankful to the Chairmen, Bansal Institute of Science & Technology Bhopal for giving permission to send the paper for publication. Last but not least, I would also like to thanks our colleagues for supporting us.

References

[1] Schreier R., Temes G. C, “Understanding Delta-Sigma Data Converters,” New York: IEEE Press, 2005.
[2] D. Reefman, J. Reiss, E. Janssen, and M. Sandler, “Description of Limit Cycles in Sigma–Delta Modulators,” IEEE Trans. on Circuits and Systems I: Regular Papers, vol. 52, pp. 1211–1223 (2005).
[3] G. I. Bourdopoulos, A. Pnevmatikakis, V. Anastassopoulos, and T. L. Deliyannis, Delta-Sigma Modulators: Modeling, Design and Applications (Imperial College Press, London, UK, 2003.

[4] Abdelghane Denbdouga, Nour-Eddine Bougechal, Souheal kouda and Samir Barrea "Modeling of second order sigma delta modulator with imperfections"international journal on electrical engineering and informatics vol. 3 Nov 2011
[5] V. Mladenov, H. Hegt, and A. V. Roermund, "On the Stability Analysis of High Order Sigma-Delta Modulators," Analog Integrated Circuits and Signal Processing, vol. 36 (2003).
[6] James C. Candy, Gabor C. Temes. "Oversampling Methods for A/D D/A Conversion, Oversampling Delta-Sigma Converters," New Jersey, IEEE Press, 1992, p. 2-3.
[7] Fogel, David B. (editor) (1998). Evolutionary Computation: The Fossil Record. New York: IEEE Press. ISBN 0-7803-3481-7.
[8] K. A. De Jong, "Are genetic algorithms function optimizers?" in Parallel Problem Solving from Narure, 2 , R. Manner and B. Manderick, Eds. Amsterdam: North Holland, 1992, pp. 3-13.
[9] D.E. Goldberg, Genetics Algorithms: in Search Optimisation and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[10] Zbigniew Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin, 3 editions, 1996.