一般社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

スーパーサンプリング DAC　$\alpha = -2 + \sqrt{3}$
無限スプライン関数による補間法

小林 芳直† 　肥後 信嗣‡

† 東京大学 〒113-0033 東京都文京区本郷 7-3-1 理学部 7 号館
‡ SLDJ 〒164-0011 東京都中野区中央 5-42-4-302
E-mail: † masa19@estate.ocn.ne.jp, ‡ higon@nifty.com

あらまし I2S 入力の 16 ビットの DAC の新方式を開発した。CD や TV のデジタル音声データのサンプル点とサンプル点の間を無限 Spline 関数で補間することにより、値が連続、一次微分が連続、２次微分が連続した滑らかなアナログ波形に変換することができる。無限スプライン関数の係数計算と実行には無理数を含む多数の積和算が必要になるが、これをデータの流れに同期して計算し、実行するためのハードウェアについて説明する。

キーワード 無限スプライン関数、スーパーサンプリング、オンザフライ、

Super-Sampling DAC　$\alpha = -2 + \sqrt{3}$
An infinite-spline function used for interpolation

Yoshinao Kobayashi† and Nobutsugu Higo‡

† Faculty of Science, Tokyo University 7-3-1 Hongo, Bunkyo-ward, Tokyo, 113-0033 Japan
‡ SLDJ Corporation 5-42-4-302, Chuou, Nakano-ward, Tokyo 164-0011 Japan
E-mail: † masa19@estate.ocn.ne.jp, ‡ higon@nifty.com

Abstract An infinite-spline function is proposed for audio DAC. A series of audio digital data of TV or CD can be interpolated by an infinite-spline function that generates a continuous, first and second order continuously differentiable wave that realizes a soft and clear sound. We call it a super sampling DAC. The super sampling DAC requires numerous multiply-adder calculation with irrational number for the generation of the co-efficient of infinite-spline function that consume amount of logic cells and the maximum frequency becomes low. This paper describes how to build this system effectively and execute it staying with the input data stream.

Keywords : Infinite spline function, super sampling, data interpolation, irrational number

1. 無限 Spline 関数

Spline 関数は次の 3 次関数で定義される。

$$S_j(x) = a_j x^3 + b_j x^2 + c_j x + d_j \quad \cdots①$$

ただし $\mathbf{x}$ の変域は $\mathbf{0} \le \mathbf{x} \le \mathbf{1}$ である。

Spline 関数は一つのサンプル点から次のサンプル点までの一つの区間で定義される。区間が移動すれば係数は再計算され新しい Spline 関数が定義される。このとき隣り合った区間では関数の値は連続し、1 次微分と2次微分も連続する。この良く整理された理論から係数の間には次の3式の関係がある。

$$a_j + b_j + c_j + d_j = d_{j+1} \quad \cdots②$$

∵値が連続

$$3a_j + 2b_j + c_j = c_{j+1} \quad \cdots③$$

∵1次微分が連続

$$6a_j + 2b_j = 2b_{j+1} \quad \cdots④$$

∵2次微分が連続

ここで d_j はサンプル点の値であり既知である。 a_j b_j

c_j は Spline 関数の係数であり未知数である。Spline 関数を決定するとは既知の値d_j を与えて連立 3 元一次方程式② ③ ④を解いて未知数 a_j b_j c_j を求めることである。
ここで④式から

$$a_j = (b_{j+1} - b_j)/3 \qquad \cdots⑤$$

⑤式を②式に代入して

$$c_j = d_{j+1} - d_j - {}^{b_{j+1}}/_{3} - {}^{2b_j}/_{3} \qquad \cdots⑥$$

②式と⑥式を③式に代入して次の連立一次方程式を得る。

$$b_{j-1} + 4b_j + b_{j+1} = 3(d_{j-1} - 2d_j + d_{j+1}) \qquad \cdots⑦$$

⑦式は Spline 関数の良く知られた係数の公式である。Spline 関数はb_j を求めれば a_j c_j が求まるのでこの式だけ解けばよい。
この式を連続したn式について解こうとすると、与式が**n**式に対して未知数が$(n+2)$個なので解けない。これを便宜的に両端の b_j を0として解くのが自然 Spline 関数の解法である。nが有限の場合は自然 Spline 関数で解ける。ここではnが無限の場合の解を与える。

2. Impulse 関数

サンプル点の前後の区間について次のような無限級数の計算をする。

$$e_j = \sum_{k=0}^{+\infty} \alpha^k d_{j-k} \qquad \cdots⑧$$

$$f_j = \sum_{k=0}^{+\infty} \alpha^k d_{j+k} \qquad \cdots⑨$$

ただし

$$\alpha = -2 + \sqrt{3} \cong -0.267949192 \qquad \cdots⑩$$

という定数である。
e_j は該当する区間より過去側にあるデータの影響を集大成したものでありこれを Back Impulse という。
e_j は次の漸化式を満たす。

$$e_j = d_j + \alpha e_{j-1} \qquad \cdots⑪$$

f_j は該当する区間より未来側にあるデータの影響を集大成したものでありこれを Front Impulse という。
f_j は次の漸化式を満たす。

$$f_j = d_j + \alpha f_{j+1} \qquad \cdots⑫$$

3. Front Impulse, Back Impulse の計算

Front Impulse の計算を Figure1 で説明する。スーパーサンプリング DAC,以下 SSDAC、の内部は 15 状態を持つ FSM：Finite state machine, ステートマシンで制御されていてそれぞれの状態を Time1～Time15 で表す。この FSM は I2S からの新しいデータの到来で起動され、スプライン関数の係数計算を終了し、スプライン関数を実行するパイプライン装置を起動して停止する。

Figure1 Calcuration of FI and BI Independent calculator

name	interval	Impulse Function	
		Front Impulse	Back Impulse
Time1	1	Write new data	
Time2	1		
Time3	1	Init FI with DO	
Time4	13	Run FI	
Time5	1	Run FI	
Time6	1		Init BI with CBI
Time7	1		Run BI
Time8	1		
Time9	1		
Time10	1		
Time11	1		save BI
Time12	1		
Time13	1		
Time14	1		
Time15	1		

SSDAC の係数計算には出力する区間の前方に 13 個のサンプル点のデータが必要である。このフロント側のガードエリアのデータが揃うと有意な精度をもって Spline 関数の係数を決定できる。つまりスプライン関数はサンプル点のデータが 13 個以上、具体的には全部で 16 個揃ってから、16 サンプル点分の時間遅れで FSM が起動される。
FSM はデータの到着で起動される。入力されたデータはまず FPGA の内部の SSRAM に格納される。この時 RA：レジスタアドレス、のプリデクリメントが行われる。つまり最新のデータは一つ前のデータの一つ下のアドレスに格納される。SSRAM は１６アドレスであり、この書き込みと同時に 17 サンプル点前のデータが破棄される。ここで RA=0 に格納されるのが最新のデータ

であり、RA が一つ登るたびに未来から現代へのデータが呼び出される。
この SSRAM は FPGA でよく使われる同期式の SSRAM であり RA:SSRAM のアドレス、を与えてから２クロック後に RD：SSRAM のデータが確定する。データの書き込みにはアドレスとデータを同時に与える。
入力データは一度 SSRAAM に格納されてから使用される。入力データは SSRAM への書き込み専用、SSRAM の出力は SSDAC への入力専用と機能分担している。このように機能分担により SSRAM の性質が替わった場合の対応が容易になる。
さて最初のデータは今から出力したい区間の 15 サンプル区間だけ未来のデータなのでこれをd_{15}とする。この d_{15} を Time1 で SSRAM に書き込み、同じデータが Time3 で SSRAM の DO に現れるので、このデータを Front Impulse の初期値として与える。

$$\mathrm{FI} = d_{15} \qquad \cdots⑬$$

FI は

$$f_j = d_j + \alpha f_{j+1} \qquad \cdots⑭$$

という漸化式を実行する関数である。ただし

$$\alpha = -2 + \sqrt{3} \cong -0.267949192 \qquad \cdots⑩$$

という定数である。次に RD1：d_{14}
を入力して FI を一回実行させると FI の値は

$$\mathrm{FI} = d_{14} + \alpha d_{15} \qquad \cdots⑮$$

になる。
Time4 のステートは 13 クロック分あり、この間この計算を繰り返すと RD13：d_2 で FI は次のようになる。

$$\mathrm{FI} = d_2 + \alpha d_3 + \alpha^2 d_4 + \cdots + \alpha^{13} d_{15} \qquad \cdots⑯$$

13 未来先までの Front Impulse の計算が完了する。
データ精度が 24 ビットの場合 Front Impulse のガードエリアは 13 が必要十分条件である。13 以上遠い未来は FI の計算結果に LSB の半分以下の影響しか与えない、13 より近い未来は近いほどより多くの影響を FI に与える。ガードエリアより狭い範囲の計算しかできないと FI の計算精度が悪くなり無限スプライン関数の形が乱れる。
データが 16 ビットの場合はガードエリアは９、データが 32 ビットの場合はガードエリアは 17 必要である。ガードエリアを必要以上に広くとることの弊害はなく 24 ビットデータ用のガードエリアで 16 ビットデータを扱っても問題は発生しない。SSDAC は 16 ビット出力だが 24 ビットデータを受け入れるのでガードエリアも 24 ビット対応の 13 を使う。ガードエリアの変更は Time4 に滞留するカウント数の変更だけで対応できる。

同様に Back Impulse の計算ができる。しかし Back Impulse 計算は次の 1 回の計算だけでよい。

$$e_j = d_j + \alpha e_{j-1} \qquad ⑰$$

なぜなら一回前の係数計算でe_{j-1}はすでに求まっているので Impulse 関数に初期値としてe_{j-1}を与えてからこの RD15：d_0 を与えればよい。
Back Impulse の計算に必要な一つ前の BI の値は何時セットしてもよいが、Back Impulse を実行するのは Time7 なのでそれより前にする。Time7 で SSRAM から Back Impulse の計算に必要な RD15：d_0 のデータが出力される。

$$\mathrm{BI} = d_0 + \alpha BI \qquad \cdots⑱$$

Figure1 では前回の BI の値を Back Impulse 関数に入力するタイミングを Time7 の一つ前の Time6 にしている。

さて Front Impulse と Back Impulse は時間軸で対象であり全く同じ演算を時間で逆方向に行っているだけなので HW も全く同じである。さらに動作タイミングにオーバーラップが無く、1 クロックの間隙がある。これは二つの関数の計算を同一 HW を使いタイムシェアして実行できることを示唆している。HW をシェアした場合の工程表を Figure2 に示す。

Figure2 Calcuration of FI and BI HW sharing

name	interval	Impulse Function	
		Front Impulse	Back Impulse
Time1	1	Write New data	
Time2	1		
Time3	1	Init IM24 with DO	
Time4	13	Run IM24	
Time5	1	Run IM24	
Time6	1	Save FI	Init IM24 with CBI
Time7	1		Run IM24
Time8	1		
Time9	1		
Time10	1		
Time11	1		save BI
Time12	1		
Time13	1		
Time14	1		
Time15	1		

こうしてf_2 と　e_0 の値が求まった。

SSDAC は複数の積和算演算が必要なので HW サイズが大きくなりやすい。大きな回路サイズを必要とする機能の一つが Impulse Function である。Impulse Function は無理数の定数積和算が必要でありビット数も多い。Front Impulse と Back Impulse　の HW を共用することができれば 1000 エレメント単位の削減ができる。この削減によるパフォーマンスの変化はない。

4. b_jの計算

f_2 と　e_0 の値が求まったので、これを使ってb_jを計算できる。

$$b_j = -3(\sqrt{3}-1)d_j + 3(2\sqrt{3}-3)(f_{j+1} + e_{j-1}) \quad \cdots⑲$$

に値を代入して

$$b_1 = -3(\sqrt{3}-1)d_1 + 3(2\sqrt{3}-3)(f_2 + e_0) \quad \cdots⑳$$

が求まる。現在出力しようとしている区間が$d_1 \sim d_0$ に対して一つ先の　b_1　　が求まった。これはa_j　c_j　の計算には区間の両端の　b_{j+1}　b_j　が必要になるので、毎回次のb_{j+1}　の係数計算をしておけば、b_j　は 1 回前の計算結果が使えるのでスプライン関数の係数を決定できる。

b_jを計算するハードウェアは実際には　$b_j/3$ を計算している。b_jは常に　$b_j/3$　の形で使われるので予め$b_j/3$を求めておけば複雑な　1/3　の処理が不要になる。

$$\beta_j \equiv b_j/3 = -(\sqrt{3}-1)d_j + (2\sqrt{3}-3)(f_{j+1} + e_{j-1}) \quad \cdots㉑$$

β_jを使った　a_j　c_j　の計算式は次のようになる。

$$a_j = \beta_{j+1} - \beta_j \quad \cdots㉒$$

$$c_j = d_{j+1} - d_j - \beta_{j+1} - 2\beta_j \quad \cdots㉓$$

β_jの計算には二つの変数に、それぞれ無理数の定数をかけ、それらを加算する演算が必要である。この様子を Figure3 に示す。

Figure3 Calculation of CB　Elephant method

name	interval	CB calculation	
		D section	(FI BI)section
Time1	1		
Time2	1		
Time3	1		
Time4	13		
Time5	1		
Time6	1		
Time7	1		
Time8	1	ND*(-√3+1)	(FI + BI)*(2√3-3)
Time9	1		Add All
Time10	1		
Time11	1		save CB
Time12	1		
Time13	1		
Time14	1		
Time15	1		

β_jの計算は SSDAC の演算処理の中でもっとも複雑なものであり、エレメントを消費するとともに最大動作速度を制限する。この計算方法では最大動作速度は 20MHz 程度になる。
ここで

$$GI(x) = (1-\sqrt{3})x \quad \cdots㉔$$

なる関数を考えるとβ_j　は次のように書き直すことができる。

$$\beta_j = \mathrm{GI}(d_j) - 2GI(f_{j+1} + e_{j-1}) - (f_{j+1} + e_{j-1}) \quad \cdots㉕$$

つまり GI 関数を使えば　β_j　の計算はおよそ半分の回路サイズになり 2 クロックで得ることができる。処理クロック数は増えるが最大動作速度が上がる。β_jの計算手順は次のようになる。

Process1 ： d_jを GI の入力レジスタに入れる
Process2 ： $\mathrm{GI}(d_j)$を計算し退避させる。
Process3 ： $(f_{j+1} + e_{j-1})$ GI の入力レジスタに入れる
Process4 ： $\mathrm{GI}(f_{j+1} + e_{j-1})$を計算し退避させる。
Process5 ： β_j の計算をする

実際には Process2 と Process3 は同時に行うことができるので 4 クロックで計算を完了することができる。

β_jの計算は一回の係数計算で 1 回だけであり処理クロック数が増えるのは構わない。それよりも回路サイズが減り最大動作速度が上がるメリットが大きい。

d_jが用意できるのが Time7，e_{j-1}が用意できるのが Time8 なのでこの手順で計算しても Time7 の 1 クロックは隠れるので、都合 Time8 から 3 クロックでβ_j　が求まったことになる。

GI()の各クロックにおける動作を Figure4 に示す。

Figure4 Calculation of CB Time sharing method

name	interval	CB calculation	
		D section	(FI BI)section
Time1	1		
Time2	1		
Time3	1		
Time4	13		
Time5	1		
Time6	1		
Time7	1	Init GI with ND	
Time8	1	Go GI	Init GI with (FI + BI)
Time9	1		GoGI
Time10	1		Add All
Time11	1		save CB
Time12	1		
Time13	1		
Time14	1		
Time15	1		

このようなマルチクロック処理により回路サイズが減ると同時に最大動作速度を上げることができる。犠牲となったのは係数計算に必要なクロック数が 2 クロック増えるという極めて軽微なものである。

5. Spline 関数の実行

Spline 関数は 3 回の積和算で実行される。

$$S_j(x) = a_j x^3 + b_j x^2 + c_j x + d_j \quad \cdots①$$

$$S_j(x) = ((a_j x + b_j)x + c_j)x + d_j \quad \cdots㉖$$

$$B = a_j x + b_j \quad \cdots㉗$$

$$C = Bx + b_j \quad \cdots㉘$$

$$S_j(x) = Cx + d_j \quad \cdots㉙$$

FSM は
Time11 でa_j b_j のロード
Time13 でc_j のロード
Time15 でd_j のロード

を行う。Spline 関数はパイプラインで連続して実行される。左右の 2 本のパイプラインが 32 クロック周期で円運動をしているのでその回転に同期させて新しい係数のセットアップが必要になる。この様子を Figure5 に示す。

Figure5 Spline Function Multiplier-Adder

state	Spline Pipeline						
	signal	Pipe1	Pipe2	Pipe3	Pipe4	Pipe5	Pipe6
		Ax		Bx		Cx	
			B=Ax + bj		C=Bx+cj		SSD=Cx+dj
Time11	Load aj, bj	Ax		Bx		Cx	
Time12			B=bj		C=Bx+cj		SSD=Cx+dj
Time13	Load cj	Ax		Bx		Cx	
Time14			B=Ax + bj		C=Bx+cj		SSD=Cx+dj
Time15	Load dj	Ax		Bx		Cx	
			B=Ax + bj		C=Bx+cj		SSD=Cx+dj
		Ax		Bx		Cx	
			B=Ax + bj		C=Bx+cj		SSD=Cx+dj
		Ax		Bx		Cx	
			B=Ax + bj		C=Bx+cj		SSD=Cx+dj
				Bx		Cx	
					C=Bx+cj		SSD=Cx+dj
						Cx	
							SSD=Cx+dj

Spline 関数の実行のためには 6 段のパイプラインが組まれていてそれぞれは次のような働きをする。

Pipeline 1 : aj に x を積算する。
X はスーパサンプリングの倍数で決まる変数であり、32 倍スーパサンプリングの場合は 5 ビットのバイナリカウンタが使われる。
64 倍スーパサンプリングの場合は 6 ビットのバイナリカウンタが使われる。
Pipeline 2 :Pipeline1 の計算結果にb_jを加算する。
Pipeline 3:Pipeline2 の計算結果に x を積算する。
Pipeline 4:Pipeline3 の計算結果にc_jを加算する。
Pipeline 5:Pipeline2 の計算結果に x を積算する。
Pipeline 6:Pipeline3 の計算結果にd_jを加算する。

パイプラインに初期値を与えるためのa_j b_j は同時で構わない。何故なら Pipeline2 の初期値はb_jであり、この瞬間は Pipeline1 の値が参照されない。この時の x の値は 0 である。このように Pipeline2 に直接初期値を与えることにより FSM と Spline Function の同期を 2 クロック早めることができる。

さてここで Pipeline1 の計算は一定値に一定値を加算するだけなので積和算器の代りにアキュムレータに置き換えることができる。このときの Spline Function の動作を Figure6 に示す。

Figure6 Spline Function w/Accumulator

state	Spline Pipeline					
	signal	Pipe 1	Pipe2	Pipe 3	Pipe 4	Pipe 5
			Bx		Cx	
		B+= δ aj		C=B+cj		SSD=Cx+dj
Time11	Load aj, bj		Bx		Cx	
Time12		B=bj		C=B+cj		SSD=Cx+dj
Time13	Load cj		Bx		Cx	
Time14		B+= δ aj		C=B+cj		SSD=Cx+dj
Time15	Load dj		Bx		Cx	
		B+= δ aj		C=B+cj		SSD=Cx+dj
			Bx		Cx	
		B+= δ aj		C=B+cj		SSD=Cx+dj
			Bx		Cx	
		B+= δ aj		C=B+cj		SSD=Cx+dj
			Bx		Cx	
				C=B+cj		SSD=Cx+dj
					Cx	
						SSD=Cx+dj

Pipeline1 の長さが 6 から 5 に短縮されて積和算器を一つ削減することができた。
パイプラインが 1 段短縮されたが動作タイミングは変更されていない。これは Pipeline1 の初期値はb_jであり、これを飛び級で与えているためである。

6. Overflow 制御

Spline Function の計算結果は理論的には入力データの 2 倍程度まで振れる可能性がある。しかし入力データが自然界の音を録音したものであればこの Overflow は発生しない。DAC の精度を落とさず Overflow にも対応するために SSDAC では Spline Function の計算結果が Overflow を起こした時に単にクリップさせる機能を入れている。
Overflow の処理を Figure7 に示す。

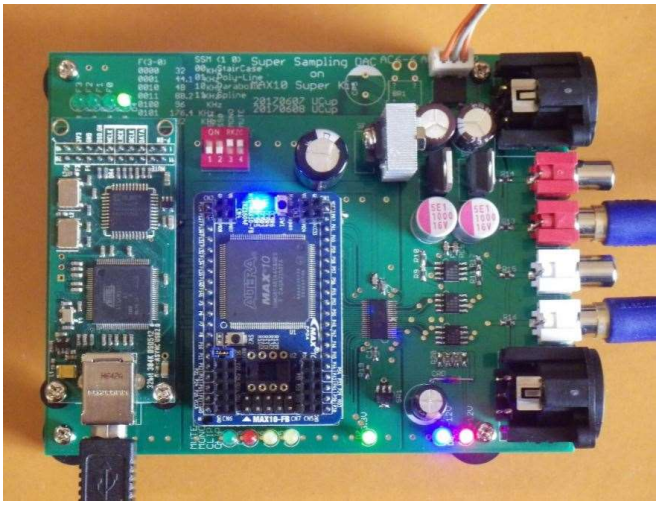
Figure7 Overflow Control

state	SSD(17 downto 0)			
	17	16	15	out data
overflow	0	1	1	0111111111111111
	0	1	0	
	0	0	1	
good	0	0	0	SSD(15 downto 0)
	1	1	1	
overflow	1	1	0	1000000000000000
	1	0	1	
	1	0	0	

クリップの機能が動作するのは元のデータがクリップしているか、人為的に作られたデータの時だけである。

7. SSDAC 基板

USB インタフェースに amanero Combo384, SSDAC の係数計算と実行に ALTERA MAX10 10M08SAE144C8G、DAC に Burr-Brown DAC8822、IV 変換に JRC MUSES8920E を使用した PCB の写真を示す。



8. まとめ

スーパーサンプリング方式により新方式の DAC を実現できた。本方式の欠点は HW サイズが大きくなりやすいことである。この欠点を克服するために、動作タイミングと計算量の解析から、大幅な 30%程度の削減と、動作速度の向上を実現できた。スーパーサンプリング方式は波形の伝送特性が良いので、今後様々な分野に展開することができる。
スーパーサンプリング方式は従来のオーバーサンプリング DAC の前処理に使うことができる。オーディオ用 DAC としては、プリエコー、リンギング、ポストエコーのない新境地を開拓した。長年の音楽ファンの悩みの解消につながることを期待している。

9. 謝辞

本論文の基礎理論の監修を頂いた小出昭夫博士に感謝します。

文 献

[1] 限界性能への挑戦と音質へのこだわり Burr-BrownAudio(2009)
http://www.tij.co.jp/jp/lit/ml/jajt042/jajt042.pdf

[2] Analog Devices シグマ・デルタ ADC/DAC の原理 AN283(2008)http://www.analog.com/media/jp/technical-documentation/application-notes/AN-283_jp.pdf.

[3] Nuno Pereira, Nuno Paulino, Design and Implementation of Sigma Delta Modulators (ΣΔM) for Class D Audio Amplifiers using Differential Pairs (2014.

[4] 「デジタルオーディオの基本と応用」河合一 誠文堂新光社 3.3.7 ロールオフ特性と再生波形、音質に対する考察(2011).