Error Correction Algorithm for Folding/Interpolation ADC

Haruo Kobayashi, Hiroshi Sakayori, Tsutomu Tobari and Hiroyuki Matsuura
Teratec Corp. 2-9-32 Naka-cho Musashino Tokyo 180 Japan
tel: 81-422-52-2102 fax: 81-422-52-2125
e-mail: haruo@teratec.yokogawa.co.jp

Abstract— **This paper discusses digital error correction algorithms for folding/interpolation AD converters which yield to very simple circuitry. The relationships between error correction and input signal frequency have been clarified theoretically and also confirmed by simulation.**

I. Introduction

A folding/interpolation analog-to-digital converter (ADC) is one of the most suitable architectures for realizing very high-speed ADCs with relatively small hardware and low power. Flash ADCs are often considered to be the fastest type of ADC because it performs an AD conversion in one clock cycle. However, they require complex hardware and considerable power, and also input capacitance can be large. On the other hand, a folding/interpolation ADC uses significantly less hardware, power and input capacitance, while it can also perform an AD conversion in one clock cycle [1,2,3,4,5]. Hence its clock skew can be smaller and its circuit components (such as comparators) can be allowed to consume more power which make it suitable for high-speed operation.

A folding/interpolation ADC is a kind of two-step ADC; a folding circuit works as a coarse ADC and generates a residue while an interpolation circuit works as a fine ADC. There is usually a slight signal delay from folding to interpolation circuits, and digital error correction circuit has to be employed as in other two-step ADCs. This paper describes digital error correction algorithm which requires only simple digital circuit, and clarifies the relationship between error correction algorithm and input signal frequency.

II. Folding/Interpolation ADC

For simplicity, we hereafter consider a 6-bit folding/interpolation ADC. Fig.1 shows its block diagram where the input is differential, the first, second and third MSBs of the Gray code are generated by folding circuits, and remaining three bits are generated by an interpolation circuit. However, the discussion here can be easily extended to other configurations of folding/interpolation ADCs.

Folding circuits perform analog encoding with differential input pairs and resistor strings (Fig.2), and generate gray code digital output (g_{f5}, g_{f4}, g_{f3}). Fig.3(a) generates MSB (g_{f5}) and Fig.3(b) generates g_{f4} while Fig.3(c) produces g_{f3} and Q and Fig.3(d) generates i_f and I, where Q and I are used in the interpolation circuit (Fig.4). We see that folding circuits can simplify digital encoder circuit and also significantly decrease the number of comparators and latches compared to flash ADCs.

The interpolation circuit reduces the number of differential input pairs and thus also decreases input capacitance. I and Q are sinusoidal signals with 90 degree phase difference utilizing Bipolar transistor characteristics [5]:

$$Q \approx G\cos(2\pi V_{in}/(4RI_b)), \quad I \approx G\sin(2\pi V_{in}/(4RI_b)),$$

where G is a constant. The interpolation circuit generates sinusoidal signals with 22.5 degree phase difference with proper choices of R_1, R_2, R_3 and R_4 values in Fig.4 [5]. The associated comparators and logic circuit find the zero-crossing point and generate Gray code. Waveforms of input versus outputs of folding and interpolation circuits are given in Fig.5.

In the actual ADC implementation, there is a signal delay δt from folding to interpolation circuits; in other words a folding circuit performs AD conversion to $V_{in}(nT)$ while an interpolation circuit performs AD conversion to $V_{in}(nT + \delta t)$, where T is the sampling period and $n = \ldots -2, -1, 0, 1, 2, \ldots$. Thus a digital error correction circuit has to be employed.

Also when the input signal is out of range and overflow or underflow occurs, the output digital signal has to be set to maximum or minimum value respectively.

We will discuss error correction algorithms which handle the above problems in the following sections.

III. Error Correction Algorithm

We assume that the folding circuit digitizes $V_{in}(nT)$ and generates digital signals g_{f5}, g_{f4}, $g_{f3}(= q_f)$ and i_f, while the interpolation circuit digitizes $V_{in}(nT+\delta t)$ and generates digital signals g_{i2}, g_{i1}, g_{i0}, q_i, and i_i. Here g_{f5} and g_{f4} are the first and second MSBs in the Gray code of $V_{in}(nT)$ while g_{i2}, g_{i1} and g_{i0} are the third, second and first LSBs in the Gray code of $V_{in}(nT+\delta t)$. Also q_f, i_f, q_i and i_i are obtained as follows:

$q_f = 1$ when $Q(nT) \geq 0$, otherwise $q_f = 0$,
$i_f = 1$ when $I(nT) \geq 0$, otherwise $i_f = 0$,
$q_i = 1$ when $Q(nT+\delta t) \geq 0$, otherwise $q_i = 0$,
$i_i = 1$ when $I(nT+\delta t) \geq 0$, otherwise $i_i = 0$.

Error correction here means to estimate g_{i5}, g_{i4} and g_{i3} (which correspond to the first, second and third MSBs in the Gray code of $V_{in}(nT+\delta t)$ and are not actually generated) from g_{f5}, g_{f4}, q_f, i_f, q_i and i_i. Note that g_{i2}, g_{i1} and g_{i0} are *not* modified in this algorithm. Our algorithms use i_f and i_i (which are *not* Gray code) as well as Gray code

g_{f5}, g_{f4}, $q_f(=g_{f3})$ and q_i in order to obtain g_{i5}, g_{i4} and g_{i3}.

Since there is a signal delay $\delta t > 0$, $q_f \neq q_i$ or $i_f \neq i_i$ can happen, and we will assign the following 'case number' according to the values of (q_f, i_f, q_i, i_i) :

case 0 : $(0,0,0,0), (0,1,0,1), (1,0,1,0)$ or $(1,1,1,1)$,

case 1 : $(0,0,1,0)$,	case 2 : $(1,0,1,1)$,
case 3 : $(1,1,0,1)$,	case 4 : $(0,1,0,0)$,
case 5 : $(1,0,0,0)$,	case 6 : $(1,1,1,0)$,
case 7 : $(0,1,1,1)$,	case 8 : $(0,0,0,1)$,
case 9 : $(0,0,1,1)$,	case 10 : $(1,0,0,1)$,
case 11 : $(1,1,0,0)$,	case 12 : $(0,1,1,0)$.

Note that in case 0, $(q_f = q_i, i_f = i_i)$ and in case 1 to 8, $(q_f = q_i, i_f \neq i_i)$ or $(q_f \neq q_i, i_f = i_i)$ while in case 9 to 12, $(q_f \neq q_i, i_f \neq i_i)$. In case 0, no correction is required and in case 1 to 8, correction works properly while in case 9 to 12, correction cannot work as shown later.

Table 1 shows digital codes of $V_{in}(nT)$ and $V_{in}(nT+\delta t)$ of several values when $V_{in}(nT)$ is *larger* than $V_{in}(nT+\delta t)$ by 3 LSBs. We see that only cases 0, 1, 2, 3 and 4 appearing there and in cases 1,2,3 or 4, Gray code (g_{f5}, g_{f4}, g_{f3}) of $V_{in}(nT)$ is *larger* than Gray code (g_{i5}, g_{i4}, g_{i3}) of $V_{in}(nT+\delta t)$ by 1. On the other hand, Table 2 shows digital codes of $V_{in}(nT)$ and $V_{in}(nT+\delta t)$ of several values when $V_{in}(nT)$ is *smaller* than $V_{in}(nT+\delta t)$ by 2 LSBs. We see that only cases 0, 5, 6, 7 and 8 appearing there and in cases 5,6,7 or 8, Gray code (g_{f5}, g_{f4}, g_{f3}) of $V_{in}(nT)$ is *smaller* than Gray code (g_{i5}, g_{i4}, g_{i3}) of $V_{in}(nT+\delta t)$ by 1. Tables 1 and 2 also show that in case 0 no correction is required. Thus we obtain the following error correction algorithm:

Algorithm 1 :
In case 0 :
Gray code (g_{i5}, g_{i4}, g_{i3}) = Gray code (g_{f5}, g_{f4}, g_{f3}).
In cases 1, 2, 3 or 4 :
Gray code (g_{i5}, g_{i4}, g_{i3}) = Gray code (g_{f5}, g_{f4}, g_{f3}) +1.
In cases 5, 6, 7 or 8 :
Gray code (g_{i5}, g_{i4}, g_{i3}) = Gray code (g_{f5}, g_{f4}, g_{f3}) −1.

Let us consider to simplify Algorithm 1. Noting that increment or decrement by 1 for Gray code of g_{f5}, g_{f4} and g_{f3} can be implemented with one bit value change among g_{f5}, g_{f4} and g_{f3}, we obtain the following:

Algorithm 2 :
Error corrected Gray code g_{i5}, g_{i4}, g_{i3} is given as follows:
In cases 0, 1, 3, 5 or 7 : $g_{i5} = g_{f5}$, $g_{i4} = g_{f4}$, $g_{i3} = q_i$.
In cases 2 or 6 : $g_{i5} = g_{f5}$, $g_{i4} = \overline{g_{f4}}$, $g_{i3} = q_i$.
In cases 4 or 8 : $g_{i5} = \overline{g_{f5}}$, $g_{i4} = g_{f4}$, $g_{i3} = q_i$.

Proof In case 0, no correction is required and thus $g_{i5} = g_{f5}$, $g_{i4} = g_{f4}$, $g_{i3} = q_f(=q_i)$. It follows from Tables 1 and 2 that when $q_f \neq q_i$(i.e., in cases 1, 3, 5 or 7), $g_{i3} = q_i$, in cases 2 or 6, $g_{i4} = \overline{g_{f4}}$, and in cases 4 or 8, $g_{i5} = \overline{g_{f5}}$. □

We note that it is straightforward to extend these algorithms for folding/interpolation ADCs with other resolution.

Let us consider the limitations of the error correction. Table 3 shows digital codes of $V_{in}(nT)$ and $V_{in}(nT+\delta t)$ of several values when $V_{in}(nT)$ is *larger* than $V_{in}(nT+\delta t)$ by 9 LSBs. We see that cases 9, 10, 11 and 12 as well as cases 0, 1, 2, 3 and 4 appear there. On the other hand, Table 4 shows digital codes of of $V_{in}(nT)$ and $V_{in}(nT+\delta t)$ of several values when $V_{in}(nT)$ is *smaller* than $V_{in}(nT+\delta t)$ by 10 LSBs. We see that cases 9, 10, 11 and 12 as well as cases 0, 5, 6, 7 and 8 appear there. Thus in cases 9, 10, 11 and 12, we can not decide whether (g_{f5}, g_{f4}, g_{f3}) should be decreased or increased by 1. From these considerations, we see that the error correction works properly if the following is satisfied:

$$-2^m\ LSBs < V_{in}(nT+\delta t) - V_{in}(nT) < 2^m\ LSBs \quad (1)$$

where m is the number of bits generated by the interpolation circuit and in this example $m = 3$.

IV. Overflow/Underflow Detection

If overflow or underflow of input occurs, the digital output of the ADC should be set to (1,0,0,0,0,0) or (0,0,0,0,0,0) in the Gray code respectively. We see that folding circuits in Figs.3(a) and (b) generate the first two MSBs (1,0) or (0,0) automatically when overflow or underflow occurs respectively. However because of cyclic nature of interpolation circuit signals, the interpolation circuit cannot set the remaining bits to (0,0,0,0), and thus overflow/underflow detection circuit is required. We can implement out-of-range detection circuit with a folding circuit as shown in Fig.6, and if its comparator output "out-rng" is 0, the first, second, third and fourth LSBs are set to 0 by digital gates.

V. Circuit Implementation

If q_f, i_f, q_i, i_i, "out-rng" as well as $g_{f5}, g_{f4}, g_{i2}, g_{i1}, g_{i0}$ are captured in acquisition memory, a digital processor can perform the above error correction by software. However this requires 4-bit extra memory per word, and some applications may demand real-time error correction. Now we consider the logic design which implements the algorithms. Algorithm 2 and the above mentioned overflow/undeflow handling yield to the following Boolean expressions :

$$g_{i5} = (\overline{q_f} \cdot (i_f \oplus i_i) \cdot \text{out-rng}) \oplus g_{f5},$$

$$g_{i4} = (q_f \cdot (i_f \oplus i_i) \cdot \text{out-rng}) \oplus g_{f4},$$

$$g_{i3} = q_i \cdot \text{out-rng}, \quad g_2 = g_{i2} \cdot \text{out-rng},$$

$$g_1 = g_{i1} \cdot \text{out-rng}, \quad g_0 = g_{i0} \cdot \text{out-rng}.$$

Fig.7 shows the circuit implementation for g_{i5}, g_{i4}, g_{i3} whereas Fig.8 shows error correction circuitry for g_{i2}, g_{i1}, g_{i0} with an encoder for interpolation outputs. One sees that these are very simple.

VI. Error Correction and Input Signal Frequency

The signal delay δt from folding to interpolation circuits restricts the maximum frequency of the input signal, and this section clarifies their relationships. Let the input be a sinusoidal signal $V_{in}(t) = A\sin(2\pi f_{in} t)$, where $\pm A$ is ADC input range and f_{in} is an input signal frequency. Also let k be the number of bits generated by folding circuit (in this example $k = 3$). Then it follows from equation (1) that if the following equation is satisfied, the error correction works properly.

$$2^{-k+1}A \geq \max_t |A\sin(2\pi f_{in}(t+\delta t)) - A\sin(2\pi f_{in}t)|$$
$$\approx \max_t A|\cos(2\pi f_{in}t)|2\pi f_{in}\delta t, \quad \text{when } \delta t \text{ is small,}$$
$$= 2A\pi f_{in}\delta t.$$

Thus we obtain

$$f_{in}\delta t \leq 1/(2^k\pi). \qquad (2)$$

The above equation gives us the maximum signal frequency, for a δt, for which the error correction works properly.

VII. Simulation Results

Fig.9 shows the simulation results of Algorithm 2 with a sinusoidal input and several values of signal delay δt, and S/N was obtained by FFT. A solid line indicates S/N versus δt *with* error correction and a dashed line indicates S/N versus δt *without* error correction. $f_{in} = 17/1024$ and δt is changed from 0 to 4.0. Note that $1/(f_{in}2^3\pi) \approx 2.4$ which is the maximum allowable δt for error correction given by (2). Simulation results in Fig.9 show that when $\delta t \leq 2.4$, S/N *with* error correction maintains the value of $\approx$ 6bit $\times$ 6.02 + 1.8 dB = 37.92 dB while S/N *without* error correction degrades, and when $\delta t > 2.4$, S/N *with* error correction also degrades. These results confirm that our error correction algorithm works properly within the range of equation (2).

VIII. Conclusion

We have described digital error correction algorithms for folding/interpolation ADCs which can be implemented with simple digital circuitry. The relationships between error correction and input signal frequency were analyzed. Simulation results confirmed that the algorithm and analysis were correct.

IX. Acknowledgements

We would like to thank H. Nakamura, M. Morimura, H. Hosomatsu and K. Kobayashi of Teratec Corp. for their continuous encouragement and guidance. Thanks are also due to K. Wilkinson of Yokogawa Electric Corp. for valuable discussions.

References

[1] J. J. Corcoran et.al., "A 400MHz 6b ADC," *ISSCC*, Feb. 1984.

[2] R. J. Grift, et.al., "An 8b Video ADC Incorporating Folding and Interpolation Techniques," *J. of Solid-State Circuits*, Dec. 1987.

[3] R. J. Plassche et.al., "An 8b 100MHz Full-Nyquist ADC," *J. of Solid-State Circuits*, Dec. 1988.

[4] J. Valburg and R. J. Plassche, "An 8b 650MHz Folding ADC," *J. of Solid-State Circuits*, Dec. 1992.

[5] W. Colleran and A. A. Abidi, "A 10b 75MHz Two-Stage Pipelined Bipolar ADC," *J. of Solid-State Circuits*, Dec. 1993.

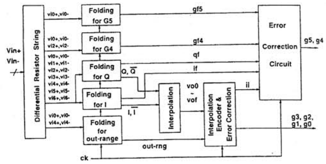

Fig.1 Block diagram of a 6b folding / interpolation ADC

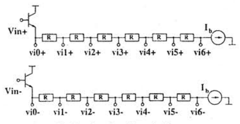

Fig.2 Input resistor ladder

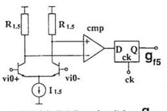

Fig.3 (a) Folding circuit for g_{f5}

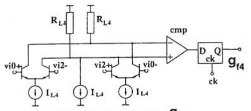

Fig.3 (b) Folding circuit for g_{f4}

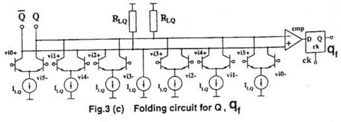

Fig.3 (c) Folding circuit for Q, q_f

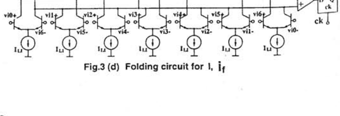

Fig.3 (d) Folding circuit for I, i_f

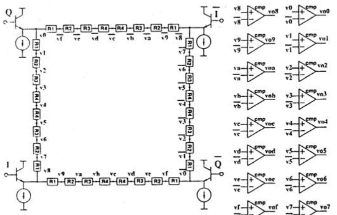
Fig.4 I, Q-type interpolation circuit

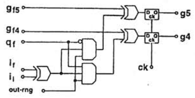

Fig.7 Error correction for MSBs

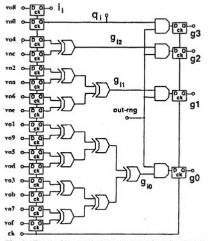

Fig.8 Encoding and error correction for LSBs

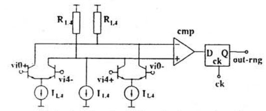

Fig.5 Folding circuit for out-of-range detection

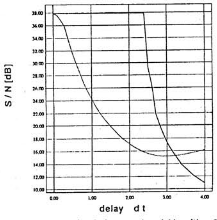

Fig. 9 Simulation results of Algorithm 2
——— with error correction
……… without error correction

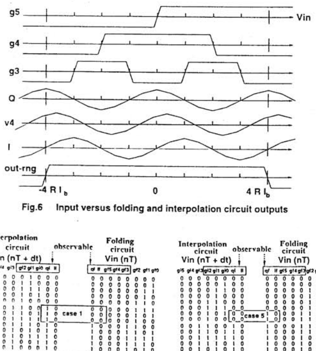

Fig.6 Input versus folding and interpolation circuit outputs

Interpolation circuit Vin (nT + dt)								observable	Folding circuit Vin (nT)							
gi5	gi4	gi3	gi2	gi1	gi0	qi	ii		qf	if	gf5	gf4	gf3	gf2	gf1	gf0
0	0	0	0	1	0	0	0		0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0		0	0	0	0	0	0	0	1
0	0	0	1	1	1	0	0		0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0		0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0		0	0	0	0	0	1	1	0
0	0	1	1	0	0	1	0		0	0	0	0	0	1	1	1
0	0	1	1	0	1	1	0	case 1	0	0	0	0	0	1	0	1
0	0	1	1	1	1	1	0		0	0	0	0	0	1	0	0
0	0	1	1	1	0	1	0		1	0	0	0	1	1	0	0
0	0	1	0	1	0	1	0		1	0	0	0	1	1	0	1
0	0	1	0	1	1	1	0		1	0	0	0	1	1	1	1
0	0	1	0	0	1	1	0		1	0	0	0	1	1	1	0
0	0	1	0	0	0	1	0		1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1		1	0	0	0	1	0	1	1
0	1	1	0	0	1	1	1	case 2	1	0	0	0	1	0	0	1
0	1	1	0	1	1	1	1		1	0	0	0	1	0	0	0
0	1	1	0	1	0	1	1		1	1	0	1	1	0	0	0
0	1	1	1	1	0	1	1		1	1	0	1	1	0	0	1
0	1	1	1	1	1	1	1		1	1	0	1	1	0	1	1
0	1	1	1	0	1	1	1		1	1	0	1	1	0	1	0
0	1	1	1	0	0	1	1		1	1	0	1	1	1	1	0
0	1	0	1	0	0	0	1		1	1	0	1	1	1	1	1
0	1	0	1	0	1	0	1	case 3	1	1	0	1	1	1	0	1
0	1	0	1	1	1	0	1		1	1	0	1	1	1	0	0
0	1	0	1	1	0	0	1		0	1	0	1	0	1	0	0
0	1	0	0	1	0	0	1		0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	1		0	1	0	1	0	1	1	1
0	1	0	0	0	1	0	1		0	1	0	1	0	1	1	0
0	1	0	0	0	0	0	1		0	1	0	1	0	0	1	0
1	1	0	0	0	0	0	0		0	1	0	1	0	0	1	1
1	1	0	0	0	1	0	0	case 4	0	1	0	1	0	0	0	1
1	1	0	0	1	1	0	0		0	1	0	1	0	0	0	0
1	1	0	0	1	0	0	0		0	0	1	1	0	0	0	0
1	1	0	1	1	0	0	0		0	0	1	1	0	0	0	1
1	1	0	1	1	1	0	0		0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	0		0	0	1	1	0	0	1	0
1	1	0	1	0	0	0	0		0	0	1	1	0	1	1	0
1	1	1	1	0	0	1	0		0	0	1	1	0	1	1	1
1	1	1	1	0	1	1	0	case 1	0	0	1	1	0	1	0	1
1	1	1	1	1	1	1	0		0	0	1	1	0	1	0	0
1	1	1	1	1	0	1	0		1	0	1	1	1	1	0	0

Table 1 Vin (nT + dt) - Vin (nT) = - 3 LSBs
(within error correction range.)

Interpolation circuit Vin (nT + dt)								observable	Folding circuit Vin (nT)							
gi5	gi4	gi3	gi2	gi1	gi0	qi	ii		qf	if	gf5	gf4	gf3	gf2	gf1	gf0
0	0	0	0	0	0	0	0		0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0		0	0	0	0	0	0	1	0
0	0	0	0	1	1	0	0		0	0	0	0	0	1	1	0
0	0	0	0	1	0	0	0		0	0	0	0	0	1	1	1
0	0	0	1	1	0	0	0		0	0	0	0	0	1	0	1
0	0	0	1	1	1	0	0		0	0	0	0	0	1	0	0
0	0	0	1	0	1	0	0	case 5	1	0	0	0	1	1	0	0
0	0	0	1	0	0	0	0		1	0	0	0	1	1	0	1
0	0	1	1	0	0	1	0		1	0	0	0	1	1	1	1
0	0	1	1	0	1	1	0		1	0	0	0	1	1	1	0
0	0	1	1	1	1	1	0		1	0	0	0	1	0	1	0
0	0	1	1	1	0	1	0		1	0	0	0	1	0	1	1
0	0	1	0	1	0	1	0		1	0	0	0	1	0	0	1
0	0	1	0	1	1	1	0		1	0	0	0	1	0	0	0
0	0	1	0	0	1	1	0	case 6	1	1	0	1	1	0	0	0
0	0	1	0	0	0	1	0		1	1	0	1	1	0	0	1
0	1	1	0	0	0	1	1		1	1	0	1	1	0	1	1
0	1	1	0	0	1	1	1		1	1	0	1	1	0	1	0
0	1	1	0	1	1	1	1		1	1	0	1	1	1	1	0
0	1	1	0	1	0	1	1		1	1	0	1	1	1	1	1
0	1	1	1	1	0	1	1		1	1	0	1	1	1	0	1
0	1	1	1	1	1	1	1		1	1	0	1	1	1	0	0
0	1	1	1	0	1	1	1	case 7	0	1	0	1	0	1	0	0
0	1	1	1	0	0	1	1		0	1	0	1	0	1	0	1
0	1	0	1	0	0	0	1		0	1	0	1	0	1	1	1
0	1	0	1	0	1	0	1		0	1	0	1	0	1	1	0
0	1	0	1	1	1	0	1		0	1	0	1	0	0	1	0
0	1	0	1	1	0	0	1		0	1	0	1	0	0	1	1
0	1	0	0	1	0	0	1		0	1	0	1	0	0	0	1
0	1	0	0	1	1	0	1		0	1	0	1	0	0	0	0
0	1	0	0	0	1	0	1	case 8	0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1		0	0	1	1	0	0	0	1
1	1	0	0	0	0	0	0		0	0	1	1	0	0	1	1
1	1	0	0	0	1	0	0		0	0	1	1	0	0	1	0
1	1	0	0	1	1	0	0		0	0	1	1	0	1	1	0
1	1	0	0	1	0	0	0		0	0	1	1	0	1	1	1
1	1	0	1	1	0	0	0		0	0	1	1	0	1	0	1
1	1	0	1	1	1	0	0		0	0	1	1	0	1	0	0
1	1	0	1	0	1	0	0	case 5	1	0	1	1	1	1	0	0
1	1	0	1	0	0	0	0		1	0	1	1	1	1	0	1
1	1	1	1	0	0	1	0		1	0	1	1	1	1	1	1

Table 2 Vin (nT + dt) - Vin (nT) = 2 LSBs
(within error correction range.)

Interpolation circuit Vin (nT + dt)								observable	Folding circuit Vin (nT)							
gi5	gi4	gi3	gi2	gi1	gi0	qi	ii		qf	if	gf5	gf4	gf3	gf2	gf1	gf0
0	0	1	1	0	1	1	0		0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0		0	0	0	0	0	0	0	1
0	0	1	1	1	0	1	0		0	0	0	0	0	0	1	1
0	0	1	0	1	0	1	0	case 1	0	0	0	0	0	0	1	0
0	0	1	0	1	1	1	0		0	0	0	0	0	1	1	0
0	0	1	0	0	1	1	0		0	0	0	0	0	1	1	1
0	0	1	0	0	0	1	0		0	0	0	0	0	1	0	1
0	1	1	0	0	0	1	1	case 9	0	0	0	0	0	1	0	0
0	1	1	0	0	1	1	1		1	0	0	0	1	1	0	0
0	1	1	0	1	1	1	1		1	0	0	0	1	1	0	1
0	1	1	0	1	0	1	1		1	0	0	0	1	1	1	1
0	1	1	1	1	0	1	1	case 2	1	0	0	0	1	1	1	0
0	1	1	1	1	1	1	1		1	0	0	0	1	0	1	0
0	1	1	1	0	1	1	1		1	0	0	0	1	0	1	1
0	1	1	1	0	0	1	1		1	0	0	0	1	0	0	1
0	1	0	1	0	0	0	1	case 10	1	0	0	0	1	0	0	0
0	1	0	1	0	1	0	1		1	1	0	1	1	0	0	0
0	1	0	1	1	1	0	1		1	1	0	1	1	0	0	1
0	1	0	1	1	0	0	1		1	1	0	1	1	0	1	1
0	1	0	0	1	0	0	1	case 3	1	1	0	1	1	0	1	0
0	1	0	0	1	1	0	1		1	1	0	1	1	1	1	0
0	1	0	0	0	1	0	1		1	1	0	1	1	1	1	1
0	1	0	0	0	0	0	1		1	1	0	1	1	1	0	1
1	1	0	0	0	0	0	0	case 11	1	1	0	1	1	1	0	0
1	1	0	0	0	1	0	0		0	1	0	1	0	1	0	0
1	1	0	0	1	1	0	0		0	1	0	1	0	1	0	1
1	1	0	0	1	0	0	0		0	1	0	1	0	1	1	1
1	1	0	1	1	0	0	0	case 4	0	1	0	1	0	1	1	0
1	1	0	1	1	1	0	0		0	1	0	1	0	0	1	0
1	1	0	1	0	1	0	0		0	1	0	1	0	0	1	1
1	1	0	1	0	0	0	0		0	1	0	1	0	0	0	1
1	1	1	1	0	0	1	0	case 12	0	1	0	1	0	0	0	0
1	1	1	1	0	1	1	0		0	0	1	1	0	0	0	0
1	1	1	1	1	1	1	0		0	0	1	1	0	0	0	1
1	1	1	1	1	0	1	0		0	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	case 1	0	0	1	1	0	0	1	0
1	1	1	0	1	1	1	0		0	0	1	1	0	1	1	0
1	1	1	0	0	1	1	0		0	0	1	1	0	1	1	1
1	1	1	0	0	0	1	0		0	0	1	1	0	1	0	1
1	0	1	0	0	0	1	1	case 9	0	0	1	1	0	1	0	0
1	0	1	0	0	1	1	1		1	0	1	1	1	1	0	0

Table 3 Vin (nT + dt) - Vin (nT) = -9 LSBs
(out of error correction range.)

Interpolation circuit Vin(nT + dt)								observable	Folding circuit Vin(nT)							
gi5	gi4	gi3	gi2	gi1	gi0	qi	ii		qf	if	gf5	gf4	gf3	gf2	gf1	gf0
0	0	0	0	0	0	0	0		1	0	0	0	1	1	1	1
0	0	0	0	0	1	0	0		1	0	0	0	1	1	1	0
0	0	0	0	1	1	0	0		1	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	case 5	1	0	0	0	1	0	1	1
0	0	0	1	1	0	0	0		1	0	0	0	1	0	0	1
0	0	0	1	1	1	0	0		1	0	0	0	1	0	0	0
0	0	0	1	0	1	0	0	case11	1	1	0	1	1	0	0	0
0	0	0	1	0	0	0	0		1	1	0	1	1	0	0	1
0	0	1	1	0	0	1	0		1	1	0	1	1	0	1	1
0	0	1	1	0	1	1	0		1	1	0	1	1	0	1	0
0	0	1	1	1	1	1	0	case 6	1	1	0	1	1	1	1	0
0	0	1	1	1	0	1	0		1	1	0	1	1	1	1	1
0	0	1	0	1	0	1	0		1	1	0	1	1	1	0	1
0	0	1	0	1	1	1	0		1	1	0	1	1	1	0	0
0	0	1	0	0	1	1	0	case12	0	1	0	1	0	1	0	0
0	0	1	0	0	0	1	0		0	1	0	1	0	1	0	1
0	1	1	0	0	0	1	1		0	1	0	1	0	1	1	1
0	1	1	0	0	1	1	1		0	1	0	1	0	1	1	0
0	1	1	0	1	1	1	1	case 7	0	1	0	1	0	0	1	0
0	1	1	0	1	0	1	1		0	1	0	1	0	0	1	1
0	1	1	1	1	0	1	1		0	1	0	1	0	0	0	1
0	1	1	1	1	1	1	1		0	1	0	1	0	0	0	0
0	1	1	1	0	1	1	1	case 9	0	0	1	1	0	0	0	0
0	1	1	1	0	0	1	1		0	0	1	1	0	0	0	1
0	1	0	1	0	0	0	1		0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1		0	0	1	1	0	0	1	0
0	1	0	1	1	1	0	1	case 8	0	0	1	1	0	1	1	0
0	1	0	1	1	0	0	1		0	0	1	1	0	1	1	1
0	1	0	0	1	0	0	1		0	0	1	1	0	1	0	1
0	1	0	0	1	1	0	1		0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1	case10	1	0	1	1	1	1	0	0
0	1	0	0	0	0	0	1		1	0	1	1	1	1	0	1
1	1	0	0	0	0	0	0		1	0	1	1	1	1	1	1
1	1	0	0	0	1	0	0		1	0	1	1	1	1	1	0
1	1	0	0	1	1	0	0	case 5	1	0	1	1	1	0	1	0
1	1	0	0	1	0	0	0		1	0	1	1	1	0	1	1
1	1	0	1	1	0	0	0		1	0	1	1	1	0	0	1
1	1	0	1	1	1	0	0		1	0	1	1	1	0	0	0
1	1	0	1	0	1	0	0	case11	1	1	1	0	1	0	0	0
1	1	0	1	0	0	0	0		1	1	1	0	1	0	0	1
1	1	1	1	0	0	1	0		1	1	1	0	1	0	1	1

Table 4 Vin(nT + dT) - Vin(nT) = 10 LSBs
(out of error correction range.)