# Indirect Testing of Digital-Correction Circuits in Analog-to-Digital Converters With Redundancy †

*Stephen H. Lewis* [1]
*R. Ramuchandran* [2]
*W. Martin Snelgrove* [3]

[1]Solid-State Circuits Research Laboratory
Department of Electrical and Computer Engineering
University of California
Davis, CA 95616

[2]AT&T Bell Laboratories
Allentown, PA 18103.

[3]Department of Electrical Engineering
Carleton University
Ottawa, Ontario
Canada K1S 5B6

*Abstract*

This paper presents a study of indirect fault testing of digital-correction circuits that operate as a part of analog-to-digital converters with redundancy. Design and test techniques that improve the fault coverage are described. The limitations of these techniques and methods to overcome these limitations are presented.

August 31, 1994

† This research was suppotted by AT&T and by NSF Grant No. MIP-9210071.

# Indirect Testing of Digital-Correction Circuits in Analog-to-Digital Converters With Redundancy †

*Stephen H. Lewis* [1]
*R. Ramachandran* [2]
*W. Martin Snelgrove* [3]

[1]Solid-State Circuits Research Laboratory
Department of Electrical and Computer Engineering
University of California
Davis, CA 95616

[2]AT&T Bell Laboratories
Allentown, PA 18103.

[3]Department of Electrical Engineering
Carleton University
Ottawa, Ontario
Canada K1S 5B6

## I. INTRODUCTION

Many signal-processing and communication systems operate in the digital domain on signals that start in analog form and therefore require analog-to-digital converters (ADCs). For high-speed applications such as video, most ADCs have been fabricated until recently with bipolar technologies. To reduce the cost and increase the portability of such systems, however, both increased levels of integration and reduced power dissipations are required, stimulating the study of CMOS and BiCMOS technologies to build the conversion circuits. As a result of this effort, high-speed CMOS and BiCMOS ADCs are beginning to appear as products. The use of redundancy and digital correction has emerged as an effective means of coping not only with the high offsets of amplifiers and comparators in CMOS technologies, but also with the low offsets of BiCMOS and bipolar technologies.

The advantages of using redundancy and digital correction in algorithmic or multistage

† This research was supported by AT&T and by NSF Grant No. MIP-9210071.

ADCs have been known for many years [1-3]. The key advantages are that these techniques reduce the sensitivity of the linearity of such ADCs to offsets in their comparators and offsets in their interstage analog processing. One disadvantage of this approach that is not well known is that the digital-correction logic is difficult to test indirectly (that is, without direct access to both its inputs and outputs) during normal ADC operation [4, 5]. This is because the correction logic eliminates redundancy by converting the set of its uncorrected inputs into a smaller set of corrected outputs. As a result, corrected outputs do not always reveal which operations were carried out in the correction logic to produce these outputs.

Testing the correction logic thoroughly is important to minimize the expense of field returns of an ADC product. Thorough testing can be done by injecting test vectors directly into the correction logic. This approach, however, requires direct access to both the inputs and outputs of the correction logic, which may be difficult to gain in highly integrated systems with a low cost objective. While the testing can be simplified by reducing the amount of redundancy, this also reduces the correction range, which is the range of comparator offsets that can be corrected [4].

This paper describes indirect techniques to overcome this problem without reducing the correction range and is divided into five main parts. In Section II, the subject of fault testing of digital-correction circuits is reviewed. Section III introduces the fault-analysis method for a case where the resolution of each stage except the last in a pipelined ADC is ($log_2 3$) bits. Section IV analyzes the limitations discovered by this fault analysis, and Section V presents other test techniques that can overcome these limitations. Section VI extends the fault-analysis method to a stage resolution of ($log_2 7$) bits.

## II. REVIEW

Both multistage and algorithmic ADCs can use redundancy and digital correction. The topic is reviewed here from the standpoint of a pipelined, multistage ADC but applies as well to multistage ADCs without pipelining and to algorithmic ADCs.

Fig. 1 shows a block diagram of a general, pipelined, multistage ADC with N-bit resolution. It consists of a digital-correction logic circuit and $k$ conversion stages. Each stage contains a sample-and-hold amplifier (SHA), a low-resolution analog-to-digital subconverter (ADSC), a low-resolution digital-to-analog converter (DAC), and a subtracter. The resolution of stage I by itself is $n_i$ bits. To build multistage ADCs with a large tolerance to component nonidealities, redundancy is introduced by making the sum of the individual stage resolutions greater than the resolution of the entire ADC ($\sum_{i=1}^{k} n_i > N$). When the redundancy is eliminated by the digital-correction circuit, it can be used to eliminate the effects of ADSC nonlinearity and interstage offset on the overall linearity [1-3, 6]. In many previous implementations, digital-correction circuits have used both addition and subtraction to correct errors. This approach is difficult to test because the correction logic has three options at each stage (to add, subtract, or pass the inputs unchanged), none of which is forced to occur for any output codes [4]. Consequently, the option used by each stage cannot be uniquely determined by examining the corrected ADC output. Thus, satisfactory performance during a functional test of the ADC does not guarantee that the correction logic is fault free. For example, during functional testing, the ADC could contain a set of comparator offsets that force the correction logic to do only addition. This would leave the subtraction function untested; therefore, any faults in the correction logic that inhibit subtraction may not be detected by such a test. If these comparator offsets were to change after functional testing so that subtraction were required, the undetected faults could cause the ADC output to be incorrect. Therefore, to test such correction logic thoroughly for faults, test vectors would have to be injected directly into the correction logic, bypassing the ADC. This approach increases the test time and requires access to both the inputs and outputs of the correction logic. In a stand-alone ADC, the outputs are already available, but access to the digit&correction-logic inputs requires at least one package pin. Furthermore, in an ADC that is part of a monolithic signal-processing system, even the outputs may not be available. Therefore, indirect test techniques are of potential interest.

Consider, for example, an ADC in which the resolution of each stage is 2 bits, Fig. 2(a) shows a plot of the ideal residue on the y axis versus the held input, $V_{H_1}$ , on the x axis for stage I without any offsets. The plot has a sawtooth shape, and the vertical-jumps occur at the comparator thresholds: $-\frac{1}{2}V_r$, 0 and $\frac{1}{2}V_r$, where $\pm V_r$ is the full-scale range of the ADC. The comparator outputs form a thermometer code, and a simple digital circuit converts this into a binary code, which is the ADSC output and becomes uncorrected input to the digital-correction logic. The ADSC output is shown on the top of Fig. 2(a) for the four regions of the held input (00 in the first region, 01 in the second, and so on). Within each region, the residue crosses zero once. These zero crossings occur at $-\frac{3}{4}V_r$ , $-\frac{1}{4}V_r$, $\frac{1}{4}V_r$, and $\frac{3}{4}V_r$ and represent the DAC outputs for each corresponding ADSC code.

Both addition and subtraction may be required here to correct errors caused by random comparator offsets. For example, suppose the random offset in the bottom comparator is negative so that its threshold moves to the left of $-\frac{1}{2}V_r$. This reduces the range of inputs for which the ADSC output is 00 and increases the range of inputs for which the ADSC output is 01. In the region where 01 results with offset but 00 results without offset, a subtraction is required to correct the ADSC output and ultimately produce the desired output code (01-1=00).

Similarly, if the next comparator contains a positive offset, its threshold shifts to the right of zero, and the range of held inputs for which the ADSC output is 01 increases while that for 10 decreases. In the range where the ADSC output is 01 with offset but 10 without offset, an addition (Ol+l=lO) is required to correct the error. In general, subtraction is required here to correct errors caused by negative comparator offsets and addition is required to correct errors caused by positive offsets.

To make these corrections, the subsequent stages must be able to detect the errors that occur. This requires an increase in the conversion range of the next stage, which is easily implemented by reducing the interstage gain [6]. If the interstage gain is halved (from 4 to 2 in this example) the conversion range of each stage after the first is doubled (to $\pm\frac{1}{2}V_r$ for this example as shown in Fig. 2(a)). If the DAC and SHA are ideal, the amplified residue from Fig. 2(a) remains within the conversion range of the next stage when ADSC nonlinearity shifts the comparator thresholds by no more than $\pm 1/2$ least significant bit (LSB) at a 2-bit level or $\pm V_r/4$. Under these conditions, errors caused by ADSC nonlinearity can be corrected; therefore, the correction range here, which is defined as the amount of comparator threshold movement that can be tolerated without error, is also $\pm V_r/4$. This aspect of digital correction has been extensively covered elsewhere [1-3, 6].

To eliminate the need for for the correction logic to do subtraction, a systematic offset can be added to the ADSC (which shifts the thresholds of all the comparators) and to the DAC (which shifts the residue plot vertically) [2]. If the systematic ADSC offset is positive and bigger in magnitude than the random offsets in the comparators, the ADSC output is always less than or equal to its ideal value and correction requires either no change or addition.

Fig. 2(b) shows the residue plot with systematic offsets of magnitude $V_r/4$. If the random comparator offsets are no less than $-V_r/4$, not only do the systematic offsets eliminate the need for the correction logic to do subtraction, but also they leave only one way to arrive at code 00 after correction; that is, the ADSC output must be 00 and the correction logic must add zero. This is because the associated correction logic cannot subtract. As a result, the add-zero correction option is forced to occur when $-V_r < V_{H_i} < -\frac{1}{2}V_r$. Forcing this option to occur for some values of the held input helps to test the correction logic indirectly through the ADC.

Also, after building in the systematic offsets, the top comparator (whose threshold is a$\frac{3}{4}V_r$ in Fig. 2(b)) in each stage except the last can be removed [4]. Fig. 3(a)-(c) show a block diagram of the resulting ADSC, a truth table, and a plot of the ideal residue versus the held input, respectively. Since there are only two comparators in Fig. 3(a), only three ADSC outputs can occur (00, 01, and 10). The resulting stage resolution, $n_i$, is (fog $_2 3) \approx 1.5$ bits for $1 \leq i < k$. Although this stage resolution is believed to be optimal in some sense [7], this approach works for any stage resolution. Section VI analyzes a case with a stage resolution of($log_2 7$) bits.

Removal of the top comparator facilitates the indirect testing of the correction logic for faults by forcing a correctable error to occur in the ADSC output when $\frac{1}{2}V_r < V_{H_i} < V_r$. That is, to obtain code 11 out of a stage after correction, the correction logic must add one to 10 (the maximum ADSC output here). Since code 00 can only be obtained after correction by adding zero to 00, the ability of the correction logic to both add zero and one can be verified under some circumstances by testing the complete ADC for the presence of all its output codes, This simplifies the testing of the digital-correction logic but does not entirely eliminate the problem because there are still two ways to obtain codes 01 and 10 out of each stage after correction, as described in the next section.

## III. **FAULT ANALYSIS**

Fig. 4(a) shows a block diagram of the correction logic associated with the residue plot in Fig. 3(c). It consists of $k-1$ stages because there is one correction stage for each conversion stage except the last. (The digital outputs of the last stage are not corrected, and the resolution of the last stage, $n_k$, is assumed to be 2 bits.) The inputs are: $I_1,\ldots,I_{2k}$, and these come from the ADSC outputs of the corresponding conversion stages. The corrected outputs are: $0_1,\ldots,$ $O_{k+1}$, where $0_1$ is the most significant bit (MSB) and $0_{k+1}$ is the least significant bit (LSB). Fig. 4(b) shows a gate-level diagram of the correction logic for each stage. (They are all identical.) In Fig. 4(b), the primary inputs and outputs are $(c_i, i_1, i_2)$ and $(o_1, o_2)$, respectively. The outputs $o_3$ of gate $G_3$ and $o_4$ of gate $G_4$ are internal to the correction logic. Gates $G_2$-$G_4$ implement an exclusive-or function on inputs $i_2$ and $c_i$. These gates are shown to display the internal nodes in the exclusive-or function, which are important in the fault analysis. Fig. 4(c) shows the associated truth table. Because the corresponding conversion stage has only two comparators, it cannot produce $i_1 = 1$ and $i_2 = 1$ simultaneously, and these cases are omitted from the truth table. When $c_i = 0$, the correction stage adds zero to the uncorrected input; that is, $o_1 = i_1$ and $o_2 = i_2$. When $c_i = 1$, the stage adds one to the uncorrected input. Therefore, $c_i$ is the correction applied to stage $i$.

The last column in Fig. 4(c) shows that two of the six cases are tested automatically during an all-codes test. An all-codes test establishes the ability of an N-bit ADC to generate all $2^N$ possible output codes. Because an all-codes test is commonly done on many ADCs, the extent to which it tests the correction logic indirectly for faults is important. There are two key steps in the fault analysis with an all-codes test. The first is to find the set of corrected output codes for which each output uniquely stems from one corresponding uncorrected input. This will be called the set of one-to-one correspondences. The elements in this set must be inputs and their corresponding outputs of the correction logic during an all-codes test on the ADC. As a result, each element in this set tests the correction logic to some extent during an all-codes test. So the second key step in the fault analysis is to determine the total extent to which this set tests the correction logic.

When the correction logic can add, subtract, or pass the inputs unchanged, the set ofone-to-one correspondences is null; that is, every corrected output code could result from more than one corresponding uncorrected input. Therefore, only the outputs of the correction logic are known through an all-codes test. Since the corresponding inputs are unknown, the resulting fault coverage cannot be computed. The following analysis shows that when the correction logic can only add zero or one to the uncorrected inputs, and when both options are forced to occur under certain circumstances, the fault coverage provided by an all-codes test can be determined. Furthermore, potential faults untested by the all-codes test can be resolved by special techniques.

Table 1 shows the set of one-to-one correspondences for a pipelined ADC with10-bit resolution in which each conversion stage except the last uses two comparators (Fig. 3) and the last conversion stage uses three comparators (Fig. 2(b)) [4]. The correction logic here can only add zero or one to the ADSC output of each stage. The ten corrected outputs of the digital-correction logic are shown on the right of Table 1, and the uncorrected inputs are shown in pairs on the left. For 10-bit resolution, there are nine conversion stages and therefore nine pairs of uncorrected inputs. In each pair, the left-most column is $i_1$, and the right-most column is $i_2$. To obtain the four codes shown in Table 1 during an all-codes test, all the correction stages except the last must process the inputs $I_{1,2}$ = 00 and 10 but not necessarily 01. For codes 1 and [illegible] Table 1, $c_i = 0$ in every correction stage. Therefore, these codes test the ability of every con[illegible] n stage to add zero to the uncorrected inputs $I_{1,2}$ = 00, corresponding to the first row in Fig. [illegible] Similarly, for codes 3 and 4 in Table 1, $c_i = 1$ in every correction stage. Therefore, these codes test the ability of every correction stage to add one to the uncorrected inputs $i_{1,2} = 10$, corresponding to the last row in Fig. 4(c).

A fault analysis has been done to see whether any nodes could be stuck at zero or one without being detected by the all-codes test [8, 9]. The analysis includes the use of a fault simulator [10]. The result of each simulation is summarized by the fault coverage, which is defined as the ratio of the number of detected stuck-at-zero and stuck-at-one faults to the total number of possible stuck-at-zero and stuck-at-one faults. The number of possible faults is determined by a fault model that allows one node at a time to be stuck at zero or one. For 10-bit resolution, the resulting fault coverage is 63.64% on the digital-correction logic alone and 71.51% on the correction logic and thermometer-to-binary converters together. The fault coverage numbers here are weak functions of the ADC resolution. For example, with 11-bit resolution, the resulting fault coverage is 63.51% on the digital-correction logic alone and 71.35% on the correction logic and thermometer-to-binary converters together. These numbers represent lower bounds on the fault coverage because many more vectors than just those shown in Table 1 are processed by the ADC in completing an all-codes test. The two key limitations here are that the vectors shown in Table 1 do not detect whether $i_2$ and/or $o_3$ are stuck at zero in any correction stage. This is because rows 2 and 5 in Fig. 4(c) are not exercised by this test. Although the fault coverage depends on whether the correction logic is considered alone or in conjunction with the thermometer-to-binary converters, the limitations are independent of this distinction. This is because the only potential unchecked fault in the thermometer-to-binary converters is also a potential unchecked fault in the digital correction logic; that is, whether $i_2$ is stuck at zero.

## IV. ANALYSIS OF LIMITATIONS

This section analyzes the limitations of the indirect testing revealed by the fault analysis described in Section III. Fig. 5 shows the residue plot in Fig. 3(c) in greater detail. The input range is divided into six subregions, which are labeled at the bottom of Fig. 5. For each subregion, the uncorrected input, the correction, and the corrected output of the corresponding correction stage are shown. The set of one-to-one occurrences in Table 1 fails to detect whether $i_2$ and/or $o_3$ are stuck at zero in any correction stage. Consider the $o_3$-**stuck-at-zero** fault first. Such a fault is not detected because there are two ways to arrive at a corrected output of 10: 0 1+1 in region 4 and 10+0 in region 5. Of these, only the 01+1 option requires that $o_3$ not be stuck at zero. The option chosen depends on the offset of comparator $C_1$ (whose nominal threshold is $\frac{1}{4}V_r$) and on the held input, $V_{H_i}$. Therefore, it is not possible to determine whether the 0 1+ 1 option was selected merely by examining the corrected output code.

To study this problem in detail, represent the offset of comparator $C_1$ as $V_{os_1}$. *In* this analysis, the most important values of $V_{os_1}$ **turn** out to be near $-\frac{1}{4}V_r$. Therefore, define $\delta$ as the maximum shift in $V_{os_1}$ from $-\frac{1}{4}V_r$ in a given stage that would cause an overall ADC nonlinearity of no more than 1/2 LSB at an N-bit level in the presence of an $o_3$-stuck-at zero fault. Then consider the following four cases:

1. $V_{os_1}$ c $-\frac{1}{4}V_r$.
2. $-\frac{1}{4}V_r \leq V_{os_1} \leq -\frac{1}{4}V_r + \delta$.
3. $-\frac{1}{4}V_r + \delta < V_{os_1} \leq \frac{1}{4}V_r$.
4. $V_{os_1} > \frac{1}{4}V_r$.

In case 1, the 0 1+1 option is not selected for any $V_{H_i}$; therefore, an $o_3$-stuck-at-zero fault cannot be detected. However, with such large offset, the residue exceeds the conversion range of the next stage, causing uncorrectable errors and failure of the all-codes test. Therefore, this case is not a concern from the standpoint of indirect testing of the correction logic.

In case 2, the 01+1 option is selected for $0 < V_{H_i} < \delta$. From the definition of $\delta$, an $o_3$-stuck-at-zero fault would cause no more than 1/2 LSB of ADC nonlinearity here. Since such small nonlinearity may be also caused by normal ADC operation, the $o_3$-stuck-at-zero fault may be undetected here. This is a potential problem because the magnitude of the errors caused by the fault depend on $V_{os_1}$, which may change after testing.

In case 3, an $o_3$-**stuck-at-zero** fault is detected because it causes an error that is big enough to cause ADC nonlinearity of more than 1/2 LSB. Therefore, this case is not a concern here.

In case 4, as in case 1, the residue exceeds the conversion range of the next stage, causing uncorrectable errors and failure of the all-codes test. Therefore, this case is not a concern here.

In summary, only case 2 is a concern from an indirect-testing standpoint. The probability that this case occurs depends on $\delta$ and the probability density function of the comparator offsets. To determine the value of $\delta$ in any stage, first note that the value of $\delta$ increases from stage to stage by a factor equal to the interstage gain. Next, for simplicity, consider the case when the ADSC in the last conversion stage uses the comparator thresholds shown in Fig. 2(a) (that is, without the systematic offset). Then $\delta = V_r / 1024 = 1/2$ LSB at a 10-bit level in the first conversion stage. Finally, consider the case when the last ADSC uses the comparator thresholds shown in Fig. 2(b). Here, the value of $\delta$ increases due to the systematic offset because the +1 correction in the first stage need not occur until the held input in the last stage is more than the offset. As a result, $\delta$ in the first stage increases by the amount of the offset in the last stage($\frac{1}{4}V_r$) divided by the combined interstage gain between the first and last stage ($2^8$). This gives $\delta = V_r / 512$ in the first conversion stage, increasing to $\delta = V_r / 4$ in the eighth conversion stage.

Now assume that the probability density function of comparator offsets is Gaussian with zero mean and a standard deviation that is much less than $\frac{1}{4}V_r$. Under these conditions, the probability that an $o_3$-stuck-at-zero fault is undetected is very small in every correction stage except the last. These conditions are often true in practice. For example, in fully differential comparators, the structure can be perfectly balanced, resulting in zero systematic or mean offset. Also, a typical value for the standard deviation of the offset is between 10 and 20 mV, which is much less than a typical value of $\frac{1}{4}V_r$ in a 5-V implementation (0.5 V) [4] and also in a 3-V implementation (0.25 V)[11].

Now consider the $i_2$ stuck-at-zero fault. Such a fault may not be not detected because there are two ways to arrive at a corrected output of 01 (OO+l in region 2 and 01+0 in region 3) and two ways to arrive at a corrected output of 10 (Ol+l in region 4 and 10+0 in region 5) in Fig. 5. To fail to detect an $i_2$-stuck-at-zero fault, the range of held inputs for which both the 01+0 and the 01+1 options fail to be carried out must be small enough to cause errors that can be mistaken for normal ADC nonlinearity. This requires not only that the offset of the $C_1$ comparator has to be almost $-\frac{1}{4}V_r$, but also that the offset of the $C_2$ comparator has to be almost $\frac{1}{4}V_r$. If the offsets are independent random variables, this event is less likely than failing to detect an $o_3$-stuck-at-zero fault. Therefore, the main point of this section is that the actual fault coverage is usually much higher than the theoretical lower bounds given at the end of Section III in practice.

## V. OTHER TEST TECHNIQUES

To improve the fault coverage without knowledge of the probability density function of comparator offsets, the output of the top comparator in the last conversion stage can be ignored under the control of a test input to the ADC. Ignoring this comparator output, the potential ADSC outputs of the last stage are $i_{1,2}$ = 00, 0 1, and 10 as in all the other stages; that is, $i_{1,2}$ = 11 can no longer occur in the last stage. As a result, the maximum output of the ADC is 1 1 l... 110 (LSB = 0) instead of 11 l... 111 (LSB = 1). Therefore, this comparator output should be ignored only during fault testing and not during normal ADC operation if generating an all-ones output is important. Table 2 shows the set of one-to-one correspondences under this condition. Although the all-ones code can no longer occur and is not included in Table 2, there are 19 codes in Table 2 instead of only four in Table 1. The new codes in Table 2 appear because ignoring the output of the top comparator in the last stage reduces the number of options in the correction logic to produce certain output codes. Specifically, to make the LSB $(O_{k+1})$ = 1, $i_{1,2}$ = 01 in the last stage. As a result, the carry input to the next-to-last stage is 0. Therefore, to make the two LSBs $(O_k \text{and} O_{k+1})$ = 1, $i_{1,2}$ =01 in the last two stages. This process continues backward toward the MSB and accounts for all the new codes in Table 2. In general, for j $< k+1$, to make the j LSBs $(O_{k+1-(j-1)}, \ldots, O_{k+1})$ = 1, $i_{1,2}$ = 01 in the last j stages.

To obtain the corrected output codes listed in Table 2 during fault testing, all the correction stages must process all the potential inputs ($i_{1,2}$ = 00, 01, and 10). In particular, the $i_{1,2}$ = 01 input covers the $i_2$ stuck-at-zero fault in any correction stage. Therefore, in addition to testing rows 1 and 6 in Fig. 4(c), the vectors listed in Table 2 also test rows 2 and 3. For 10-bit resolution, the fault coverage provided by the vectors in Table 2 is 8 1.82% on the digital-correction logic alone and 84.30% when including the thermometer-to-binary converters. The fault coverage numbers here are weak functions of the ADC resolution. For example, with 1 l-bit resolution, the resulting fault coverage is 81.76% on the digital-correctionlogic alone and 84.38% on the correction logic and thermometer-to-binary converters together. The coverage is less than 100% here because these tests do not include the case when $i_{1,2} = 01$ and $c_i$=1 in each correction stage (row 5 in Fig. 4(c)). Thus, the $o_3$-stuck-at-zero fault is still not covered.

To overcome this limitation, the $o_3$-stuck-at-zero fault can be tested directly. Fig. 6 shows a block diagram of the correction logic and corresponding thermometer-to-binary converters configured for direct testing. The thermometer-to-binary-converter inputs here are disconnected from the comparator outputs in each stage except the last. Instead these inputs are forced to be 0. In the last stage, the correction-logic inputs are disconnected from the corresponding thermometer-to-binary converter outputs and forced to be 0 and X, where X represents don't care. The corrected output is observed and should be 0 1 l...11X because the forced inputs should cause $i_{1,2} = 01$, and $c_i = 0$ in each correction stage. Then $c_i$ in the last correction stage is changed from 0 to 1 under external digital control. As a result, the last stage adds one to 01 and produces $o_{1,2} = 10$. Since $o_1$ from the last stage is $c_i$ in the next-to-last stage, the next-to-last stage also adds one to 01 and produces $o_{1,2} = 10$. This process continues backward, and eventually should cause $c_i = 1$ in every stage. The resulting output is again observed and should be lOO...OOX. This test overcomes the $o_3$-stuck-at-zero fault by forcing each correction stage to exercise the 0 1+1 option. Also, this test overcomes the $i_2$-stuck-at-zero fault by presenting inputs that should cause $i_2 = 1$ in each stage. Therefore, this test combined with the all-codes test using all comparators (Table 1) provides 100% fault coverage. Although direct testing is still required to guarantee complete fault coverage, direct access is required here only to one correction-logic input ($c_i$ in the last correction stage) instead of to all correction-logic inputs, as is required without indirect testing.

Complete fault coverage depends only on forcing each correction stage to exercise the 01+1 option during direct testing (row 5 in Fig. 4(c)) and both the 00+0 and 10+1 during indirect testing (rows 1 and 6 in Fig. 4(c)). Although nonidealities in the analog circuits could be large enough to interfere with the indirect testing, the magnitudes of such nonidealities also would be large enough to cause gross errors during functional testing of-the ADC. Therefore, such nonidealities are not of concern from an indirect-testing standpoint.

## VI. EXTENSION TO HIGHER STAGE RESOLUTION

The example described above has concentrated on a stage resolution, $n_i$, of $(log_2 3) \approx 1.5$ bits. Although this stage resolution is believed to be optimal in some sense [7], the ideas of adding systematic offsets and removing the top comparator can be applied to improve the testability of the correction logic with any stage resolution. This section extends the fault analysis to another practical case [12, 13]: a multistage ADC with a stage resolution of $(log_2 7) \approx 2.8$ bits. For simplicity, only the digital-correction logic (and not the thermometer-to-binary converters) are considered in this case.

Fig. 7 shows a plot of the ideal residue versus the held input in any stage except the last. As in Fig. 3, Fig. 7 includes offsets that eliminate the need for the correction logic to do subtraction. Here, the magnitude of the offsets is $V_r/8$. Also, as in Fig. 3, the comparator that would determine the top decision level in Fig. 7 (where $V_{H_i} = 7/8 V_r$) has been removed. As a result, Fig. 7 shows six decision levels and seven possible ADSC codes. Therefore, the stage resolution here is $(log_2 7) \approx 2.8$ bits.

Fig. 8(a) shows a block diagram of the correction logic associated with the residue plot in Fig. 7. The uncorrected inputs and corrected outputs are labeled on the bottom and top of Fig. 8(a), respectively. The correction logic still consists of one stage for each conversion stage except the last. (There is still no correction in the last stage.) All of the correction stages are still identical. Fig. 8(b) shows the logic equations that define the operation of each correction stage. The primary inputs and outputs are identified as $(c_i, i_1, i_2, i_3)$ and ($o_1, o_2$, and $o_3$), respectively. The outputs $o_{5-9}$ are internal to the correction logic. The logic equations in Fig. 8(b) stem from the requirement that each stage should add zero or one to its uncorrected input when $c_i = 0$ or 1, respectively. Fig. 8(c) shows the associated truth table and shows that the top and bottom rows are tested automatically by an all-codes test, as in Fig.4(c).

Table 3 shows the list of one-to-one correspondences for an example with N = 1 l-bit resolution when the last conversion stage uses a comparator at a threshold of $\frac{7}{8}V_r$. *The* eleven corrected outputs of the digital-correction logic are shown on the right of Table 3, and the uncorrected inputs are shown on the left. For 1 l-bit resolution, there are five conversion stages and therefore five triplets of uncorrected inputs. In each triplet, the left-most column is $i_1$, the middle column is $i_2$, and the right-most column is $i_3$. The pattern in Table 3 is similar to that in Table 1. For codes 1-4 in Table 3, $c_i = 0$ in every correction stage. Therefore these codes test the ability of every correction stage to add zero to the uncorrected inputs $i_{1-3} = 000$, corresponding to the first row in Fig. 8(c). Similarly, for codes 5-8 in Table 3, $c_i = 1$ in every correction stage. Therefore, these codes test the ability of every correction stage to add one to the uncorrected inputs $i_{1-3} = 110$, corresponding to the last row in Fig. 8(c). The fault coverage provided by the vectors in Table 3 obtained during an all-codes test with $k = 5$ stages and N = 11 bits is 50.68%. The key limitations here are that the test does not detect whether $i_3$ and $o_{4-6,8}$ are stuck at zero in any correction stage.

Table 4 shows the list of one-to-one correspondences when the output of the top comparator in the last stage is ignored. The pattern here is similar to that in Table 2. To make $O_{2k} = O_{2k+1} = 1$, $i_{1-3} = 011$ in the last stage. As a result, the carry input to the next-to-last stage is 0, eliminating the option of incrementing in the next-to-last stage. Therefore, when $O_{2k} = O_{2k+1} = 1$, there is only one way to produce the four combinations of $O_{2(k-1)}$ and $O_{2(k-1)+1}$. This process also continues backwards as in Table 2. To obtain the corrected output codes listed here during an all-codes test, all the correction stages must process all the potential inputs ($i_{1-3}$ = *000,001,* 010,011, 100, 101, and 110) with $c_i = 0$. Therefore, in addition to testing rows 1 and 14 in Fig. 8(c), the vectors listed in Table 4 also test rows 2-7. In particular, the $i_{1-3}$ =011 input covers the $i_3$-, $o_5$-, and o s-stuck-at-zero faults any correction stage. The fault coverage provided by the vectors in Table 4 obtained during an all-codes test with $k = 5$ *stages* and $N = 11$ bits is 67.12%. The test is still incomplete because it does not include these cases:

1. Row 9 or row 13 in Fig. 8(c), which leaves the $o_6$-stuck-at-zero fault untested.
2. Row 11 in Fig. 8(c), which leaves the $o_4$-stuck-at-zero fault untested.

These limitations can be overcome by using two direct tests similar to the one direct test described in Section V. This exercise shows that, when the stage resolution is increased from $(\log_2 3)$ bits to $(\log_2 7)$ bits, the fault coverage is reduced and the number of direct tests required to reach 100% fault coverage is increased. These results favor the use of the lowest stage resolution.

## VII. CONCLUSION

This paper presents a study of indirect fault testing of digital-correction circuits that operate as a part of analog-to-digital converters (ADCs) with redundancy. It shows that a non-zero lower-bound on the fault coverage provided by indirect testing can be determined when the correction logic can only add zero or one to its uncorrected inputs at each stage and when both options are forced to occur under certain circumstances. The paper gives the theoretical values of the lower bound for several different configurations of ADCs and shows that the actual fault coverage is usually much higher than these theoretical lower bounds in practice. Furthermore, the paper shows that these lower bounds are insensitive to nonidealities in the analog circuits. To raise the lower bounds on the fault coverage, the paper describes other testing techniques. Although direct testing is still required to guarantee complete fault coverage, such supplemental direct testing requires much less direct access to the correction-logic inputs than in traditional ADCs and may not be required in practice for the ADCs analyzed in this paper. Finally, the paper shows that ADCs constructed with a stage resolution of $(\log_2 3)$ bits are easier to test than ADCs constructed with a stage resolution of $(\log_2 7)$ bits.

## ACKNOWLEDGEMENTS

The authors are grateful to T. R. Viswanathan for his support and to V. Gopinathan and the reviewers for their helpful comments.

## REFERENCES

[1] G. G. Gorbatenko, "High-Performance Parallel-Serial Analog to Digital Converter with Error Correction," ***IEEE National Convention Record,*** pp. 39-43, New York, March,1966.

[2] 0. A. Horna, "A 150 Mbps A/D and D/A Conversion System," ***Comstat Technical Review,*** vol. 2, pp. 52-57, 1972.

[3] S. Taylor, ***High Speed Analog-to-Digital Conversion,*** p. 30-42, University of California at Berkeley, 1978. Ph. D. Thesis.

[4] S. H. Lewis, H. S. Fetterman, G. F. Gross, Jr., R. Ramachandran, and T. R. Viswanathan, "A 10-b 20-Msample/s Analog-to-Digital Converter," ***IEEE Journal of Solid-State Circuits,*** vol. ***27,*** pp. 351-358, March, 1992.

[5] C. Mangelsdorf, S.-H. Lee, M. Martin, H. Malik, T. Fukuda, and H. Matsumoto, "Design for Testability in Digitally Corrected ADCs,'' ***Digest of Technical Papers, IEEE International Solid-State Circuits Conference,*** pp. 70-71, February, 1993.

[6] S. H. Lewis and P. R. Gray, "A Pipelined 5-Msample/s 9-bit Analog-to-Digital Converter,'' ***IEEE Journal of Solid-State Circuits,*** vol. ***SC-22,*** pp. 954-96 1, December, 1987.

[7] S. H. Lewis, "Optimizing the Stage Resolution in Pipelined, Multistage, Analog-to-Digital Converters for Video-Rate Applications," ***IEEE Transactions on Circuits and Systems II,*** vol. ***39,*** pp. 5 16-523, August, 1992.

[8] D. P. Siewiorek and R. S. Swarz, ***Reliable Computer Systems,*** Digital Press, 1992. Second Edition.

[9] P. K. Lala, ***Fault Tolerant & Fault Testable Hardware Design,*** Prentice/Hall International, Inc., London, 1985.

[10] A. K. Bose, P. Kozak, H. N. Nham, E. Pacas-Skewes, and K. Wu, "A Fault Simulator For MOS LSI Circuits," ***IEEE Nineteenth Design Automation Conference Proceedings,*** pp. 400-409, Las Vegas, Nevada, June, 1982.

[11] T. B. Cho and P. R. Gray, "A lo-bit, 20-MS/s, 35-mW Pipeline A/D Converter," ***Proceedings of the IEEE 1994 Custom Integrated Circuits Conference,*** pp. 499-502, May, 1994.

[12] M. K. Mayes and S. Chin, "A Multistep A/D Converter Family with Efficient Architecture," ***IEEE Journal of Solid-State Circuits,*** vol. ***24,*** pp. 1492-1497, December, 1989.

[13] C. S. G. Conroy, D. W. Cline, and P. R. Gray, "An 8-b 85-MS/s Parallel Pipeline A/D Converter in l-urn CMOS," ***IEEE Journal of Solid-State Circuits, vol. 28,*** pp. 447-454, April, 1993.

## FIGURE AND TABLE CAPTIONS

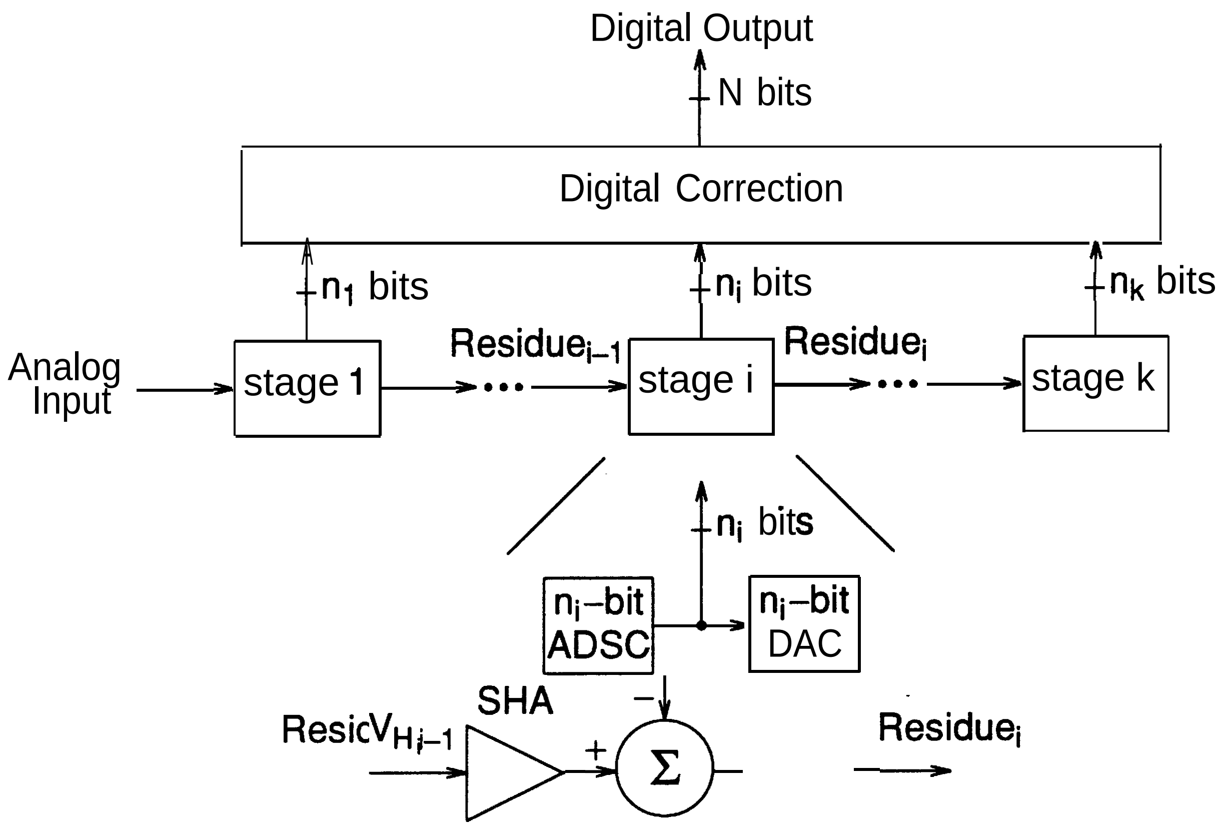
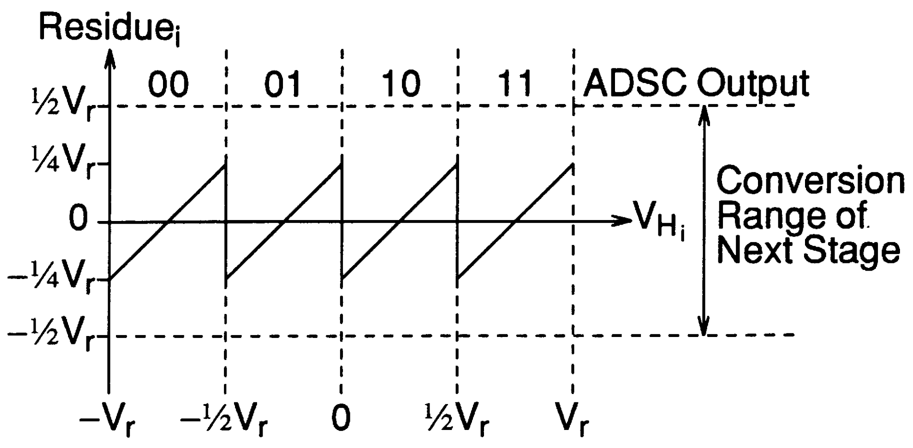
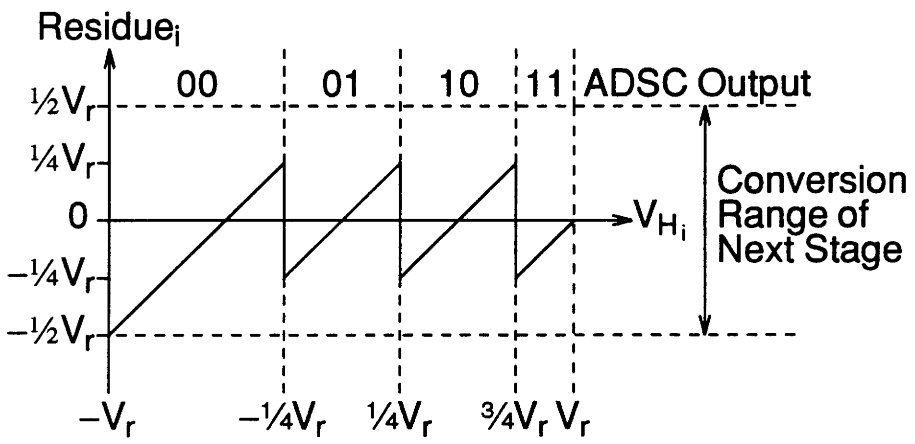


Fig. 1 - Block Diagram of a general, pipelined analog-to-digital converter with redundancy and digital correction

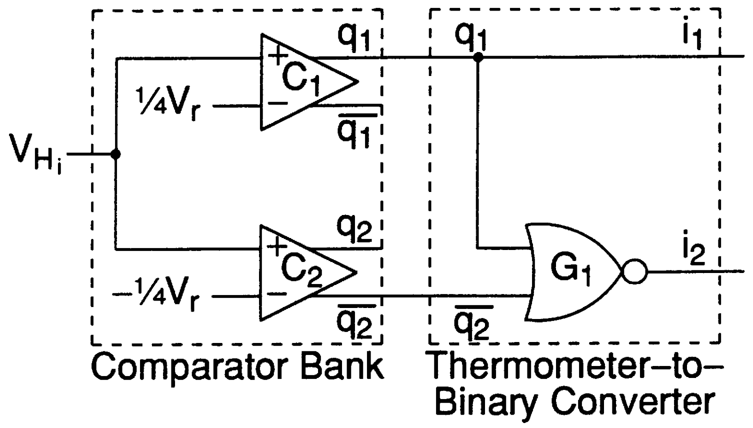


(a)



(b)

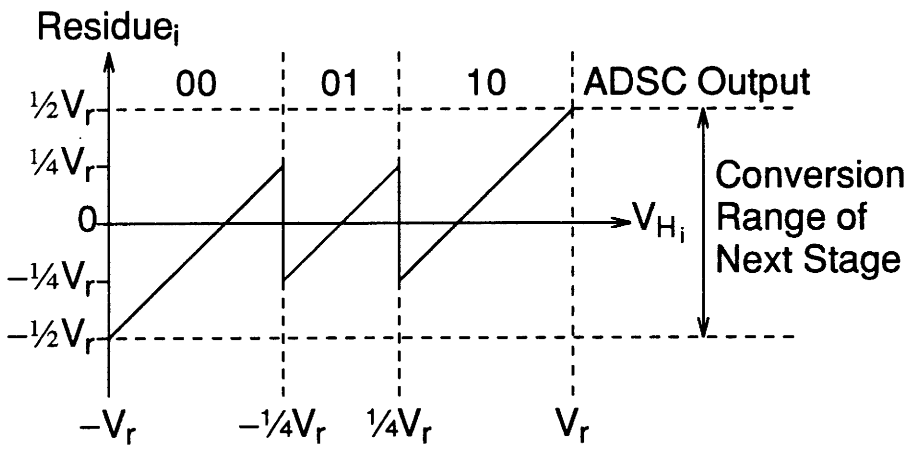
Fig. 2 - Ideal residue versus held input using three comparators
(a) Without systematic offsets
(b) With systematic offsets



(a)

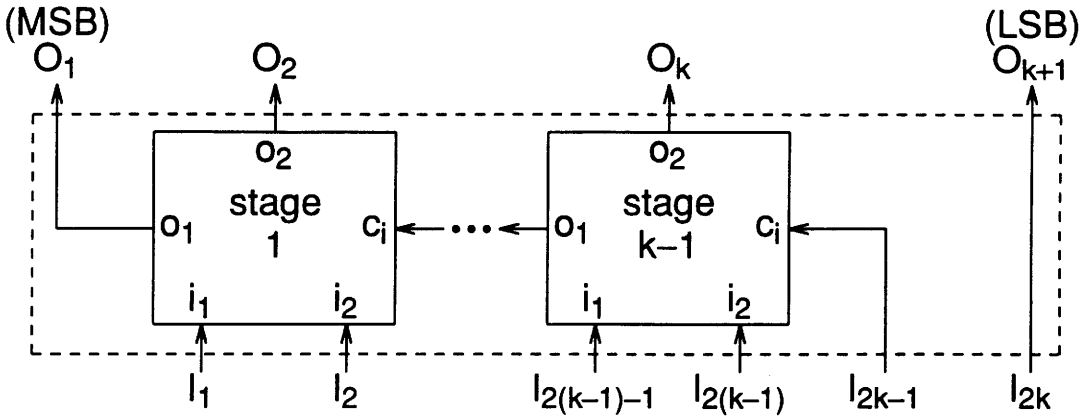
| Row Label | Held Input Range | $q_1$ | $q_2$ | $i_1$ | $i_2$ |
|---|---|---|---|---|---|
| 1 | $-V_r < V_{H_i} < -\frac{1}{4}V_r$ | 0 | 0 | 0 | 0 |
| 2 | $-\frac{1}{4}V_r < V_{H_i} < \frac{1}{4}V_r$ | 0 | 1 | 0 | 1 |
| 3 | $\frac{1}{4}V_r < V_{H_i} < V_r$ | 1 | 1 | 1 | 0 |

(b)

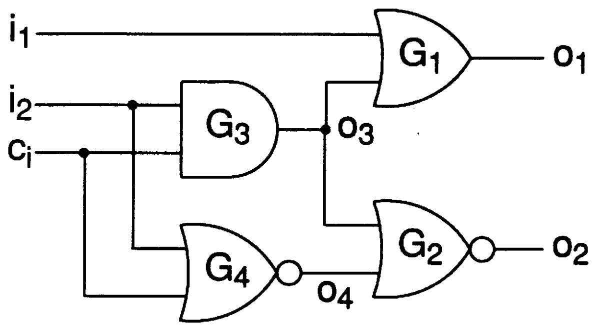


(c)

Fig. 3 - (a) Block diagram of ADSC with two comparators
(b) Truth table
(c) Ideal residue versus held input



(a)



(b)

| Row Label | $c_i$ | $i_1$ | $i_2$ | $o_1$ | $o_2$ | $o_3$ | $o_4$ | Tested? |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | yes |
| 2 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | [illegible] |
| 3 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | [illegible] |
| 4 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | no |
| 5 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | no |
| 6 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | yes |

(c)

Fig. 4 - (a) Block diagram of the digital-correction circuit when each conversion stage except the last has two comparators and the last stage has two or three comparators

(b) Gate-level diagram of each correction stage

(c) Truth table

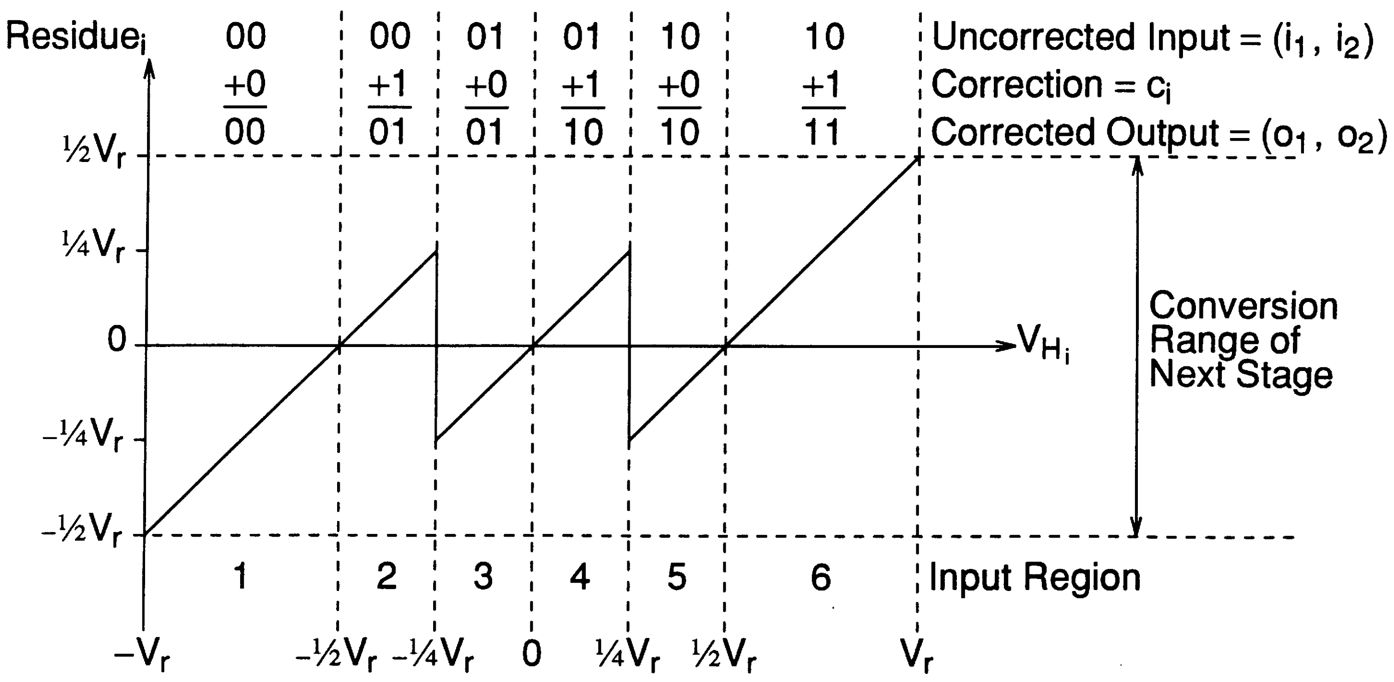


Fig. 5 - Ideal residue versus held input with two comparators, showing the uncorrected inputs, the corrections, and the corrected outputs of the corresponding correction stage for six regions of the held input

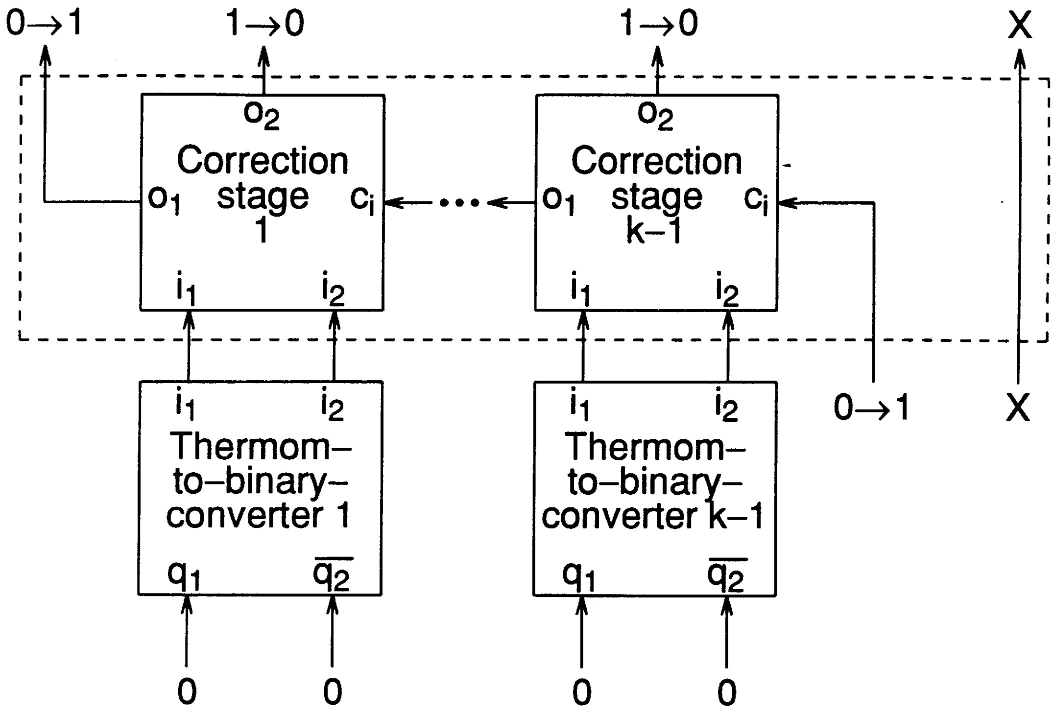


Fig. 6 - Direct-testing configuration

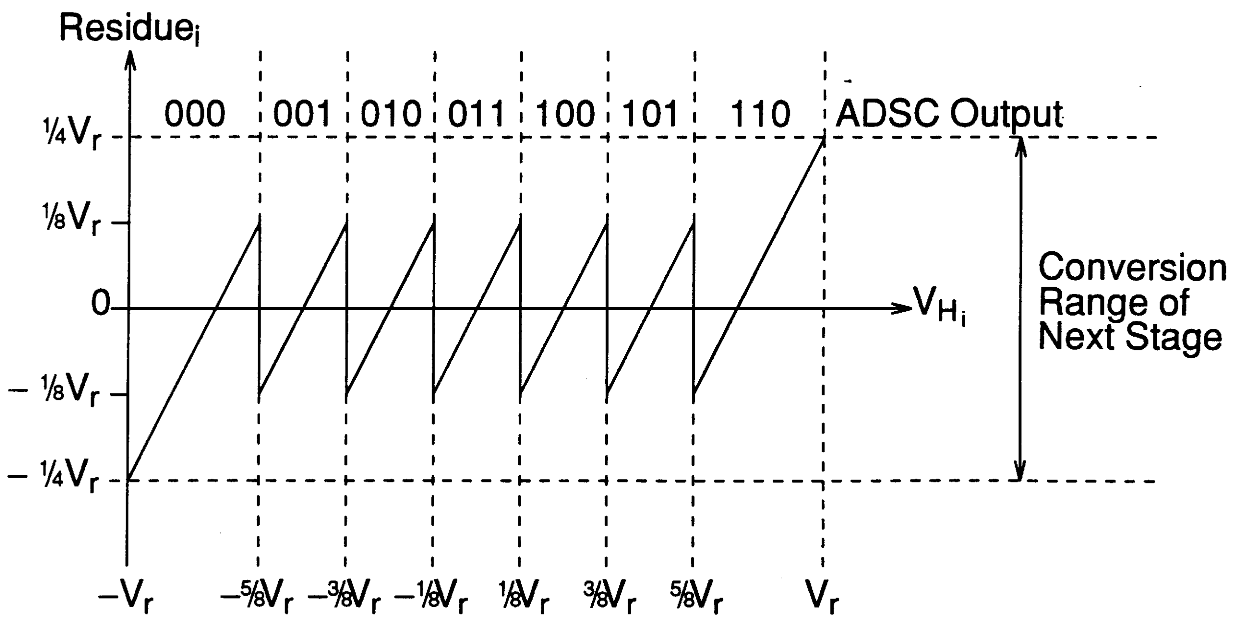
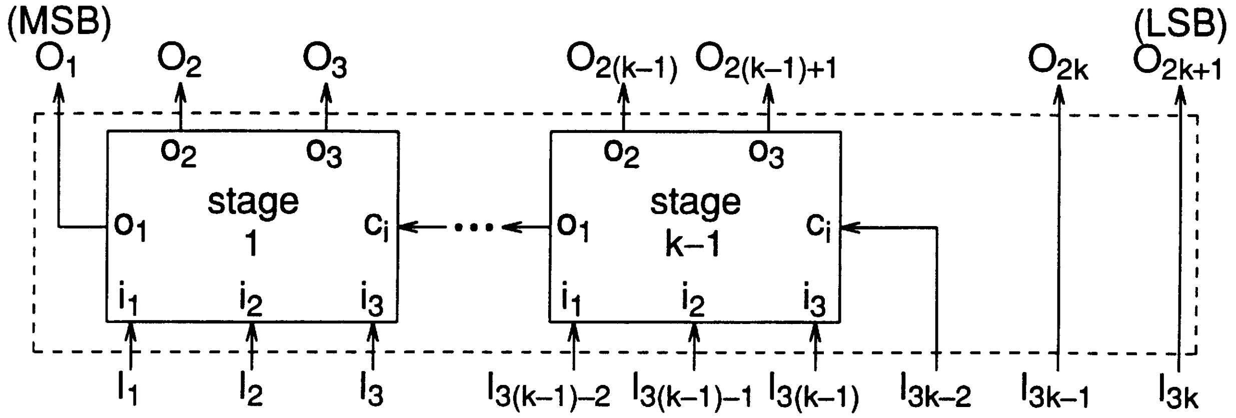


Fig. 7 - Ideal residue versus held input with six comparators



(a)

| | | | |
|---|---|---|---|
| $o_1 = i_1 + o_4$ | where | $o_4 = c_i \bullet i_2 \bullet i_3$ | $o_7 = i_2 \bullet \overline{i_3}$ |
| $o_2 = o_5 + o_6 + o_7$ | | $o_5 = \overline{c_i} \bullet i_2$ | $o_8 = \overline{c_i} \bullet i_3$ |
| $o_3 = o_8 + o_9$ | | $o_6 = c_i \bullet \overline{i_2} \bullet i_3$ | $o_9 = c_i \bullet \overline{i_3}$ |

(b)

| Row Label | $c_i$ | $i_1$ | $i_2$ | $i_3$ | $o_1$ | $o_2$ | $o_3$ | $o_4$ | $o_5$ | $o_6$ | $o_7$ | $o_8$ | $o_9$ | Tested? |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | yes |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | no |
| 3 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | no |
| 4 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | no |
| 5 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | no |
| 6 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | no |
| 7 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | no |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | no |
| 9 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | no |
| 10 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | no |
| 11 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | no |
| 12 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | no |
| 13 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | no |
| 14 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | yes |

(c)

Fig. 8 - (a) Block diagram of the digital-correction circuit when each conversion stage except the last has six comparators and the last stage has six or seven comparators

(b) Logic equations of each correction stage

(c) Truth table

Table 1 - The set of digital-correction input-output codes for which each output corresponds to one input with k=9 conversion stages and when each conversion stage except the last uses two comparators. (The last conversion stage uses three comparators.)

| Code Label | $I_{1,2}$-$I_{17,18}$ | $O_1$-$O_{10}$ |
|---|---|---|
| 1 | 00 00 00 00 00 00 00 00 00 | 0000000000 |
| 2 | 00 00 00 00 00 00 00 00 01 | 0000000001 |
| 3 | 10 10 10 10 10 10 10 10 10 | 1111111110 |
| 4 | 10 10 10 10 10 10 10 10 11 | 1111111111 |

Table 2 - The set of digital-correction input-output codes for which each output corresponds to one input with k=9 conversion stages and when each conversion stage uses two comparators.

| Code Label | $I_{1,2}$-$I_{17,18}$ | $O_1$-$O_{10}$ |
|---|---|---|
| 1 | 00 00 00 00 00 00 00 00 00 | 0000000000 |
| 2 | 00 00 00 00 00 00 00 00 01 | 0000000001 |
| 3 | 00 00 00 00 00 00 00 01 01 | 0000000011 |
| 4 | 00 00 00 00 00 00 01 01 01 | 0000000111 |
| 5 | 00 00 00 00 00 01 01 01 01 | 0000001111 |
| 6 | 00 00 00 00 01 01 01 01 01 | 0000011111 |
| 7 | 00 00 00 01 01 01 01 01 01 | 0000111111 |
| 8 | 00 00 01 01 01 01 01 01 01 | 0001111111 |
| 9 | 00 01 01 01 01 01 01 01 01 | 0011111111 |
| 10 | 01 01 01 01 01 01 01 01 01 | 0111111111 |
| 11 | 10 01 01 01 01 01 01 01 01 | 1011111111 |
| 12 | 10 10 01 01 01 01 01 01 01 | 1101111111 |
| 13 | 10 10 10 01 01 01 01 01 01 | 1110111111 |
| 14 | 10 10 10 10 01 01 01 01 01 | 1111011111 |
| 15 | 10 10 10 10 10 01 01 01 01 | 1111101111 |
| 16 | 10 10 10 10 10 10 01 01 01 | 1111110111 |
| 17 | 10 10 10 10 10 10 10 01 01 | 1111111011 |
| 18 | 10 10 10 10 10 10 10 10 01 | 1111111101 |
| 19 | 10 10 10 10 10 10 10 10 10 | 1111111110 |

Table 3 - The set of digital-correction input-output codes for which each output corresponds to one input with k=5 conversion stages and when each conversion stage except the last uses six comparators. (The last conversion stage uses seven comparators.)

| Code Label | Uncorrected Input $I_{1-3}$-$I_{13-15}$ | Corrected Output $O_1$-$O_{11}$ |
|---|---|---|
| 1 | 000 000 000 000 000 | 00000000000 |
| 2 | 000 000 000 000 001 | 00000000001 |
| 3 | 000 000 000 000 010 | 00000000010 |
| 4 | 000 000 000 000 011 | 00000000011 |
| 5 | 110 110 110 110 100 | 11111111100 |
| 6 | 110 110 110 110 101 | 11111111101 |
| 7 | 110 110 110 110 110 | 11111111110 |
| 8 | 110 110 110 110 111 | 11111111111 |

Table 4 - The set of digital-correction input-output codes for which each output corresponds to one input with k=5 conversion stages and when each conversion stage uses six comparators.

| Code Label | Uncorrected Input $I_{1-3}$-$I_{13-15}$ | Corrected Output $O_1$-$O_{11}$ |
|---|---|---|
| 1 | 000 000 000 000 000 | 00000000000 |
| 2 | 000 000 000 000 001 | 00000000001 |
| 3 | 000 000 000 000 010 | 00000000010 |
| 4 | 000 000 000 000 011 | 00000000011 |
| 5 | 000 000 000 001 011 | 00000000111 |
| 6 | 000 000 000 010 011 | 00000001011 |
| 7 | 000 000 000 011 011 | 00000001111 |
| 8 | 000 000 001 011 011 | 00000011111 |
| 9 | 000 000 010 011 011 | 00000101111 |
| 10 | 000 000 011 011 011 | 00000111111 |
| 11 | 000 001 011 011 011 | 00001111111 |
| 12 | 000 010 011 011 011 | 00010111111 |
| 13 | 000 011 011 011 011 | 00011111111 |
| 14 | 001 011 011 011 011 | 00111111111 |
| 15 | 010 011 011 011 011 | 01011111111 |
| 16 | 011 011 011 011 011 | 01111111111 |
| 17 | 100 011 011 011 011 | 10011111111 |
| 18 | 101 011 011 011 011 | 10111111111 |
| 19 | 110 011 011 011 011 | 11011111111 |
| 20 | 110 100 011 011 011 | 11100111111 |
| 21 | 110 101 011 011 011 | 11101111111 |
| 22 | 110 110 011 011 011 | 11110111111 |
| 23 | 110 110 100 011 011 | 11111001111 |
| 24 | 110 110 101 011 011 | 11111011111 |
| 25 | 110 110 110 011 011 | 11111101111 |
| 26 | 110 110 110 100 011 | 11111110011 |
| 27 | 110 110 110 101 011 | 11111110111 |
| 28 | 110 110 110 110 011 | 11111111011 |
| 29 | 110 110 110 110 100 | 11111111100 |
| 30 | 110 110 110 110 101 | 11111111101 |
| 31 | 110 110 110 110 110 | 11111111110 |