On improving the algorithmic robustness of a low-power FIR filter

Sourabh Khire and Saibal Mukhopadhyay
Georgia Institute of Technology, Atlanta, GA 30332
Email: sourabh_khire@gatech.edu, saibal@ece.gatech.edu

***Abstract*— Voltage scaling is a promising approach to reduce the power consumption in signal processing circuits. However aggressive voltage scaling can introduce errors in the output signal, thus degrading the algorithmic performance of the circuit. We consider the specific case of the finite impulse response (FIR) filter, and identify two different sources of errors occurring due to voltage scaling: (a) errors introduced because of increased delay along the logic path and (b) errors caused by failures in the memory due to process variations. We design a FIR filter by using a simple feedback based approach to reduce the memory errors and a linear predictor structure for correcting the logic errors. The proposed filter is more robust to both logic and memory errors caused by voltage scaling. The results show a considerable improvement in the output Signal to Noise ratio (at least around 10 dB) for a probability of error (P_{err}) even as high as 0.5. We also utilize the proposed technique for an image filtering application and observe a considerable improvement in the visual quality of the output image along with an improvement of over 10 dB in the Peak Signal to Noise ratio for P_{err} as high as 0.5.**

I. Introduction

Voltage scaling ([1], [2]) helps to reduce the power consumption in digital CMOS circuits. For signal processing applications, maintaining the circuit throughput is of greater value than performing computations faster than the sampling rate. This understanding allows designers to adopt a suitable supply-voltage scaling strategy to reduce dynamic, short circuit and leakage power [3]. However, aggressive supply voltage scaling introduces errors in the circuit output and degrades the functional reliability of the circuit. Hence an effective voltage scaling strategy demands an effective control these errors.

The use of multiple supply voltages and adaptive voltage scaling for low power FIR filtering has been proposed in [4]. Several different approaches including voltage scaling for realizing low power implementations of FIR filters have been presented in [5]. In [6], [7], the authors have proposed approaches to compensate for degradation in the algorithmic performance of signal processing circuits caused due to voltage scaling. Similarly, we suggest using voltage scaling for low power FIR filtering and propose a methodology to improve the functional reliability of the FIR filter that will allow aggressive voltage scaling with minimum impact on the signal fidelity. However in contrast to other approaches we identify and compensate for two different types of errors occurring in a voltage-scaled FIR filter: the logic errors, and the memory errors. To correct memory errors we propose a feedback based approach which adapts the supply voltage for the coefficient memory when memory errors are detected. Finally by including both the logic and the memory error correction circuitry, we show that the voltage-scaled noisy filter can operate at much higher failure probabilities.

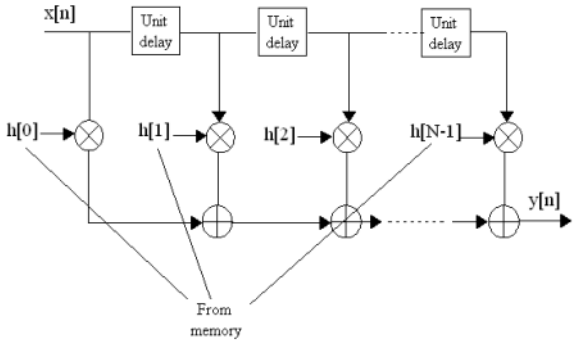

Fig. 1. Direct Form implementation of a FIR filter

II. Identifying the different sources of error in a voltage-scaled FIR filter

The output of a N-tap FIR filter is calculated as follows,

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \tag{1}$$

here, x[n], y[n] and h[n] are the input, output and the impulse response of the filter respectively. The direct form implementation of this FIR filter is shown in Fig. 1. The filter elements can be classified into two broad categories: the logic elements consisting of the multipliers, adders and the shift registers, and the memory element comprising of the coefficient memory.

In an implementation of a FIR filter on a generic DSP architecture the filter coefficients are stored in the one of the memory spaces (program or data memory). Since many programmable DSP's are implemented using full static CMOS technology, the dynamic power (P_d) can be significantly reduced using voltage scaling [5]. Since the logic path is active while computing every output sample, the supply voltage scaling for logic elements can be very effective in reducing the power consumption of the circuit. However it is also important to observe that for each output sample, N coefficients need to be accessed from the memory. Thus to effectively minimize the overall power consumption, voltage scaling needs to be implemented for the logic and the memory elements. However, scaling the supply voltage increases the error rate of both logic and memory, and introduces two different types of errors in the output as explained in the following subsections.

A. Logic error

If T_s denotes the sampling period of the circuit and T_{cp} denotes the critical path delay of the circuit, then the condition to ensure an error-free output is, $T_{cp} \leq T_s$ ([7]).

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

If $V_{dd-crit}$ denotes the critical supply voltage, (defined as the voltage at which $T_{cp} = T_s$), then voltage overscaling (reducing V_{dd} below $V_{dd-crit}$) will increase the critical path delay, which introduces random errors at the output.

These random errors modify (1) as follows,

$$\hat{y}[n] = \sum_{k=0}^{N-1} (h[k]x[n-k] + \varepsilon[k]) \tag{2}$$

where $\hat{y}[n]$ is the corrupted output signal and $\varepsilon[k]$ is the error. Thus the logic errors directly corrupt the output signal. For simulation purposes we designed a 27 tap low-pass FIR filter with a cutoff frequency 1500 Hz and a transition band of 500 Hz using the MATLAB 'fdatool'. To simulate the logic errors we assume a bit-error probability ($P_{log-err}$), and randomly introduce bit-errors in the output signal.

B. Memory error

As mentioned previously, a FIR filter can be implemented on a generic programmable DSP by storing the coefficients in the on-chip memory. For example, the TMS320C2x [8] provides a total of 544 16-bit words of on-chip SRAM; partitioned into program or data memory. Either of these spaces can be used as coefficient memory and be subjected to voltage scaling for reducing the overall power consumption. However supply voltage scaling in the memory leads to memory-errors. This happens because voltage scaling increases the sensitivity of the memory elements to the manufacturing variations [9]. As a result, larger number of memory failures occurs at reduced supply voltages and corrupt the stored values of filter coefficients and result in memory errors.

The frequency response of a FIR filter is given by

$$H(\omega) = \sum_{n=0}^{N-1} h[n]e^{-jwn} \tag{3}$$

However due to corruption of the filter coefficients, the actual transfer function gets modified to,

$$\hat{H}(\omega) = \sum_{n=0}^{N-1} (h[n] + e[n])e^{-j\omega n} = H(\omega) + E(\omega) \tag{4}$$

where,

$$E(\omega) = \sum_{n=0}^{N-1} e[n]e^{-j\omega n} \quad \text{and} \quad e[n] = \hat{h}[n] - h[n]$$

As seen in (4) the corrupted frequency response can actually be separated into two components; the original frequency response ($H(\omega)$) and the error response ($E(\omega)$). Clearly, the memory errors modify the frequency response of the filter.

The logic and memory errors described above can be somewhat likened to the rounding and coefficient-quantization errors in fixed point implementation of digital FIR filters. The characteristics and effects of these rounding and quantization errors is very well understood [10] and hence robust structures can be designed to compensate for them [11]. However, the distribution of logic and memory errors is not well defined, which introduces additional difficulties in modeling these errors and developing approaches to reduce them. We suggest one possible implementation in the next section.

III. Improving circuit robustness to logic and memory errors

A. Forward path correction for logic errors

Logic errors introduce noise directly in the output signal. Since the output signal is readily available for further processing, it is possible to include error correction mechanisms in the forward path itself. Several existing approaches can be used for this. The approach mentioned in [6] uses a checksum based probabilistic error correction technique. Algorithmic noise tolerance (ANT) based detection and correction of errors due to voltage overscaling is proposed in [7], [12]. We correct the logic errors by using an implementation similar to the one proposed in [7]. Here, a N_p tap linear predictor is cascaded at the output of the noisy filter and used for error detection and correction as described below,

1) *Threshold calculation:* The linear predictor output $y_p[n]$ is calculated as, $y_p[n] = \sum_{k=1}^{N_p} h_p(k)y(n-k)$, where $h_p[k]$ are the predictor coefficients obtained by minimizing the mean-square prediction error. Then the threshold, $T = 4 * \sigma_{e_p}$ is calculated, where σ_{e_p} denotes the variance of the prediction error (e_p) with noiseless digital filter.
2) *Error detection:* If $|\hat{e}_p[n]| > T$, then an error is declared. Here $\hat{e}_p[n]$ is the prediction error with noisy filter.
3) *Error correction:* If error is declared, then $y_o[n] = \hat{y}_p[n]$, else $y_o[n] = \hat{y}[n]$. Thus if an error is detected, the predictor output ($\hat{y}_p[n]$) is declared as the system output ($y_o[n]$). If no error is detected, the system output is same as the output of the noisy filter ($\hat{y}[n]$).

The above scheme works under the assumptions that, (a) the magnitude of error introduced by the noisy filter is very large and, (b) the interval between two successive errors is greater than $2N_p$.

According to us, using the above scheme requires some modifications for reasons described below.

The coefficients ($h_p[k]$) of the linear predictor are obtained by minimizing mean-squared prediction error. This requires solving the *normal equations* [13] of the form,

$$\sum_{k=1}^{N_p} h_p[k]R_n[|i-k|] = R_n[i] \quad 1 < i < p; \tag{5}$$

where R_n is the short-time autocorrelation of the input signal. Thus the optimal predictor coefficients obtained from (5), depend on the input signal. If the logic errors introduce too much disturbance in $y[n]$ then the coefficients are no longer optimal for the noisy signal $\hat{y}_p[n]$. This means that the noisy prediction error ($\hat{e}_p[n]$) is much larger than noiseless prediction error ($e_p[n]$) and also $\hat{y}_p[n]$ is not necessarily a good estimate of the noiseless signal as seen from Fig. 2 and Fig. 3 respectively.

The difference between $e_p[n]$ and $\hat{e}_p[n]$ is acceptable, as long as there is a big change in $\hat{e}_p[n]$ when a logic error occurs. Hence error detection ability of the circuit is not affected much. However from the Fig. 3 we see that $\hat{y}_p[n] \neq y_p[n]$, and the difference is considerably large. This means that $\hat{y}_p[n]$ is not a good estimate of the noiseless signal and hence should not be directly used as output. (Remember that $y_p[n]$ is not available to us). To minimize

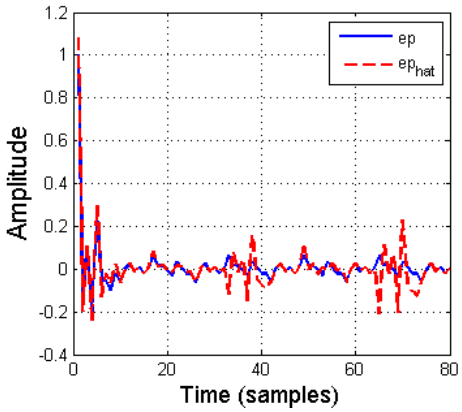

Fig. 2. Comparison of the prediction error with noiseless (ep) and noisy signal (ep_{hat}) and the predictor input

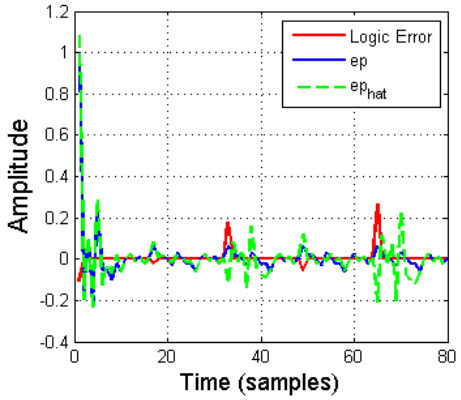

Fig. 3. Comparison of logic error ($y-\hat{y}[n]$), prediction error ($ep = y-y_p[n]$) and noisy-estimate error ($ep_{hat} = y-\hat{y}_p[n]$)

this problem, we modify the error correction circuitry. If no error is detected, then $y_o[n] = \hat{y}[n]$. If error is declared, then $y_o[n] = \frac{1}{M}\sum_{m=1}^{m=M} y_o[n-m]$. Thus, if an error is detected, the system output ($y_o[n]$) is a time-average of the past M correct samples. To reduce complexity the order M, of this moving average filter is kept small (M = 4 or 5). This output smoothening can reduce the SNR degradation provided that the output samples do not change very rapidly.

B. Feedback correction for memory errors

The memory errors directly modify the frequency response of the filter. This may lead to errors in each and every output sample (since every output sample is now calculated from corrupted memory coefficients) and hence these errors are certainly not bursty or isolated. Moreover, there is no correlation between the different values of the coefficients. The linear predictor works on the assumption that neighboring samples of the input signal are correlated. This means that the linear predictor cannot be used to correct these memory errors. To satisfactorily address this problem the corrupted filter coefficients need to be corrected before the input signal is fed to the filter. We propose a feedback approach to identify and correct erroneous filter coefficients as described below.

Reducing the supply voltage to the SRAM array increases the probability of failure (due to read disturb, access or write failure) of the memory elements [9]. Thus we can

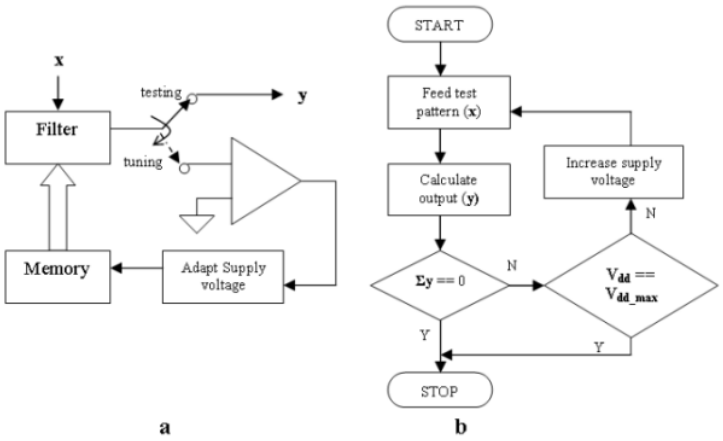

Fig. 4. Memory Error Correction a)Block diagram b)Flow chart

reduce the memory errors if we can detect failures in the coefficient memory, and then increase the supply voltage till the failure rate is zero, or at an acceptable value. Ideally, we would like to detect memory errors by comparing the corrupt coefficients directly with the designed coefficients. However the true coefficient values are not available to us. Instead what is available to us is the corrupted output signal. Hence we can indirectly detect memory errors by observing deviation in the observed output from the expected output. For instance, let us assume that the expected output is $y[n] = 0,\ \forall\ n$. Then for a fixed value of $y[n]$, a special input test pattern ($x[n]$) can be generated as shown below.

$$\begin{aligned} y[n] &= \sum_{k=0}^{N-1} h[k]x[n-k] \\ \Rightarrow\ y[n] &= \sum_{k=1}^{N-1} h[k]x[n-k] + h[0]x[n] \\ \Rightarrow\ x[n] &= \frac{1}{h[0]}\left(y[n] - \sum_{k=1}^{N-1} h[k]x[n-k]\right) \end{aligned}$$

Starting with $x[0] = 1$, it is possible to find the remaining values of $x[n]$, such that $y[n] = 0$ for $n = 0, 1 \ldots N-1$. Since the filter coefficients are known, the test pattern can be generated offline (by software), and then fed to the filter during the tuning phase. If all the coefficients are correct, then the expected output is $y[n] = 0$ for $n = 0, 1 \cdots N-1$.

However if the filter coefficients are corrupted, then $y[n] \neq 0$ for at least one $n = 0 \cdots N-1$. Thus if $y[n] \neq 0$, an error is detected and the and hence the supply voltage for the memory elements needs to be increased by a pre-defined step-size, i.e. $V_{dd} = V_{dd} + \Delta$, where Δ is the step-size. The block diagram and the flowgraph are shown in Fig. 4. From the flowgraph it can be seen that V_{dd} is increased only after every N samples. Thus the entire test pattern ($x[n]$) is allowed to pass through the circuit and if the sum of all the N output samples is found to be non-zero, then V_{dd} is increased by Δ. This is done because changing V_{dd} instantaneously after each non-zero sample is not practically possible. This process can continue till all the output samples settle to the expected value (i.e. 0) or until a pre-defined maximum value of supply voltage (V_{dd-max}) is reached.

In SRAM the read-disturb failures present the primary bottleneck for scaling the supply voltage in the coefficient memory. Hence we show our experiments by considering only the read-disturb failures. Fig. 5 shows the relation

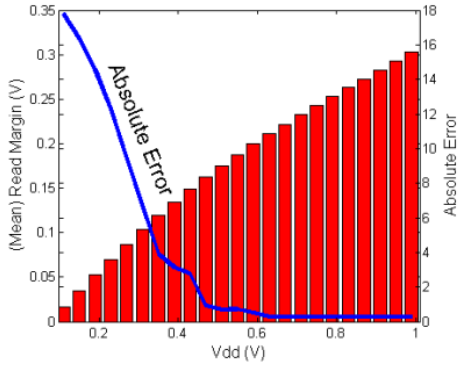

Fig. 5. Illustration of the effect of V_{dd} on the read margin and the absolute error in coefficient values)

between supply voltage (V_{dd}), the Read margin (data obtained from [9]). From Fig. 5 we can see that that as V_{dd} increases, the Read Margin increases and hence the absolute error between the true coefficients and the corrupted coefficients reduces. Here we have assumed that a bit-error in the coefficient occurs if the Read margin is below a threshold value ($R_{thresh} = 50mV$).

C. Combining the two

The final implementation assumes both logic and memory errors are occurring together and introduces both the feedback and feedforward correction circuitry. The block diagram of the final implementation is shown Fig. 6. The performance of the proposed configuration is discussed in the next section.

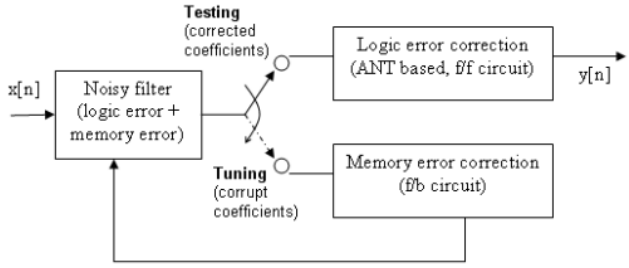

Fig. 6. Final implementation including both the memory and logic error correction blocks

IV. Results and Discussion

The final implementation combines the memory-error and the logic-error correction circuitry to improve the overall functional performance of the FIR filter. To obtain a quantitative measure of performance improvement, the output SNR is calculated as follows, $SNR = 10\log(\frac{\sigma_s}{\sigma_n})$. σ_s is the variance of the error-free output $y\_ideal[n]$ and σ_n is the variance of the noise. Here, the noise is equal to $y\_ideal[n] - y\_noisy[n]$ in case of no error correction mechanism and, equal to $y\_ideal[n] - y\_corrected[n]$ when error correction is introduced. Thus we can obtain two SNR's: **SNR_noisy** and **SNR_corrected** corresponding to $y\_noisy[n]$ and $y\_corrected[n]$ respectively. The input signal in these simulations is a set of 3000 samples extracted from a random speech signal.

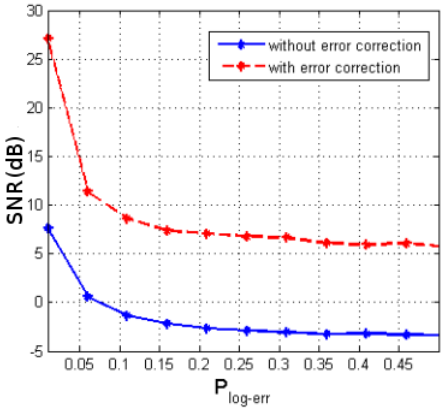

Fig. 7. Variation of the output SNR with $P_{log-err}$

Fig. 7 shows the improvement in SNR_corrected over SNR_noised due to the addition of the linear predictor based logic correction block as the bit-error probability ($P_{log-err}$) is increased. This figure is generated by assuming that the time interval between two successive isolated errors is 12 ($< 2N_p$) and there are no memory errors. The performance is expected to deteriorate as the burst interval is decreased. Fig. 7 shows an improvement of around 15 dB in the SNR at lower values of $P_{log-err}$. However, the performance of the correction circuitry degrades with increasing $P_{log-err}$.

Fig. 8 shows the variation of the probability of memory error ($P_{mem-err}$) with supply voltage. From Fig. 8 we can observe that in the proposed memory-error correction scheme, increasing the supply voltage helps to bring down the probability of memory failures (due to increasing Read Margin).

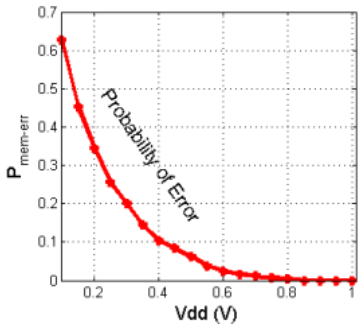

Fig. 8. Variation of $P_{mem-err}$ with Vdd

To demonstrate the algorithmic robustness of the proposed filter we need to verify the functional performance of the combined circuit at lower operating voltages, or in other words at higher probabilities of bit-failures. Thus to verify the circuit robustness we need to approximate the behavior shown in Fig. 5 and Fig. 8 and simulate these memory errors. To do this, a bit-failure probability ($P_{mem-err}$) is assumed and then a randomly generated number for each bit of each coefficient is compared with $P_{mem-err}$ to determine if it is corrupted. For example, if we fix the supply voltage to be $V_{dd} = 0.2V$, then from Fig. 8 we can expect the errors in the coefficients to occur with an error probability of $P_{mem-err} = 0.35$, and then compute the corrupted filter coefficients.

Thus, if the correction circuit starts with a operating

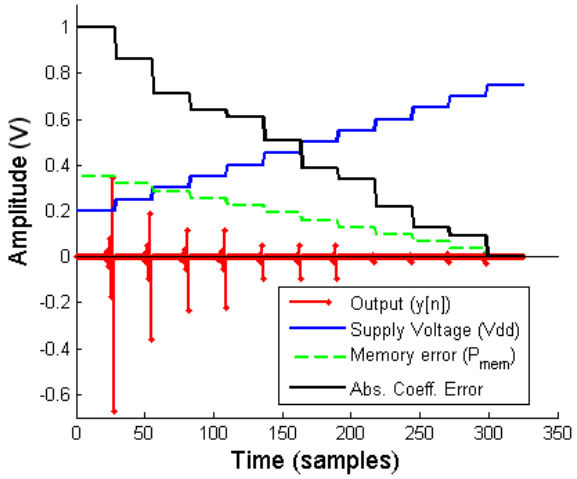

Fig. 9. Illustration of the proposed memory error correction scheme.

value of supply voltage (V_{dd}), in the simulation we start by assuming an initial value of $P_{mem-err}$. Now as the correction circuit increases V_{dd} in steps (to reduce memory-errors), in the simulation we reduce $P_{mem-err}$ in steps. Thus we basically move along the curve in Fig. 8. Also as mentioned previously, the supply voltage adaptation terminates if $V_{dd} == V_{dd-max}$. For simulation purposes this upper-bound can be specified in terms of no. of steps ($max-steps$), i.e. the no. of times $P_{mem-err}$ is reduced (or V_{dd} is increased).

Fig. 9 simulates the run-time behavior of the memory correction scheme. We start with $V_{dd} = 0.2V$ or $P_{mem-err} = 0.35$. Now each output sample is compared to 0. After every N output samples we check if $\sum_{n=0}^{N-1} |y[n]|$ is non-zero. If it is non-zero then we increase V_{dd}, i.e. effectively reduce $P_{mem-err}$. This process continues until all the output samples settle to 0 (assuming that no bounds such as V_{dd-max} or $max-steps$ are set). In our case $N = 27$. Thus from Fig. 9, we see that after every $N = 27$ samples V_{dd} is increased (and $P_{mem-err}$ reduced). This continues for around 12 iterations (each iteration = N samples). At the end of the memory-correction phase all the output samples settle to 0, the normalized absolute error approaches 0 and V_{dd} and $P_{mem-err}$ settle down to $0.75V$ and 0.03 respectively. (The absolute error is normalized to [0, 1] for better illustration).

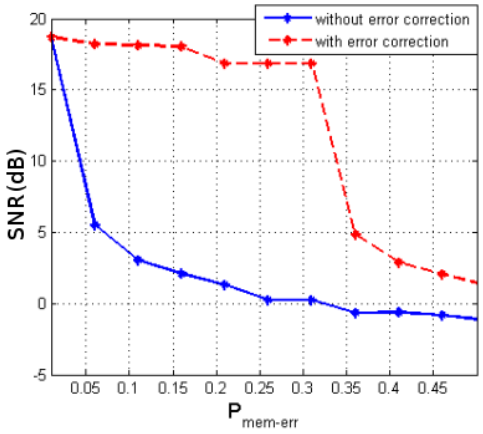

Fig. 10. Variation of the output SNR with $P_{mem-err}$

Using these assumptions, we first obtain Fig. 10, which is generated by assuming that only memory errors are present. Fig. 10 shows the improvement in SNR_corrected over SNR_noised due to the addition of memory correction block as the probability of memory failure ($P_{mem-err}$) is increased. The figure shows that at lower values of $P_{mem-err}$ a significant improvement of around 12 dB in the SNR is achieved by using the proposed memory correction scheme. The performance of the circuit deteriorates as the failure probabilities increase. It is also necessary to remember that whenever memory errors are involved, the degradation in output SNR may not be monotonic. This is because every coefficient of a FIR filter does not contribute equally to the output. Hence the amount of output degradation (or improvement on correction) actually depends on which coefficient is corrupted (and corrected).

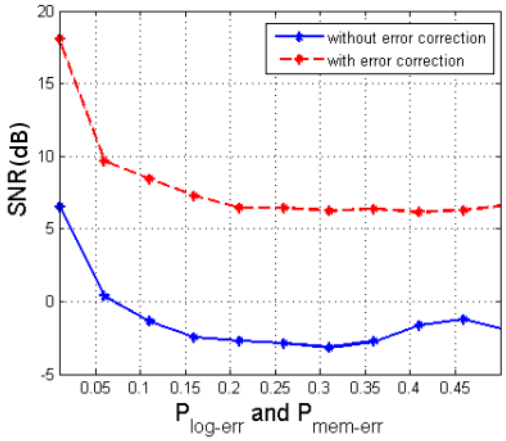

Fig. 11. Variation of the output SNR with $P_{log-err}$ and $P_{mem-err}$

Fig. 11 shows the improvement in SNR_corrected over SNR_noised due to the addition of logic and memory correction block as the bit-error probability ($P_{log-err}$) and the probability of memory failure ($P_{mem-err}$) is varied. The figure shows an constant improvement of around 10 dB in the output SNR for the entire range of P_{err}.

In all the above simulations the error correction circuits themselves are assumed to be error free. This is made possible by ensuring $T_{cp} \leq T_s$ for the correction circuits. The P_d for the logic error correction circuit can be minimized by using a smaller tap-length or reduced precision linear predictor. As demonstrated in [7], the P_d overhead due to the correction circuits is compensated for by the increased power savings afforded due to voltage overscaling. Also since the calibration of the optimal supply voltage for the coefficient memory happens offline (training phase), the memory error correction circuit does not contribute to the P_d during run-time.

Applications to Image Processing

Image filtering is very commonly used for image enhancement. For example, low pass filtering is used for image blurring, noise removal etc. and high pass filtering is used for edge detection, image sharpening etc. Since an image is a 2D signal, the impulse response of the FIR filter

is also 2D. However it is possible to decompose the 2D kernel into a set of orthogonal 1D sub-filters [14]. Thus the 2D filtering operation can be separated into successive 1D filtering operations along the rows and columns of the image. So the proposed correction technique can directly be applied for reducing the output degradation due to logic and memory errors in a voltage-scaled FIR filter for images.

Fig. 12. a) Original Image (Lena) b) Filtered image without errors c) Filtered Image with memory and logic errors ($P_{mem-err} = P_{log-err} = 0.1$) d) Filtered image after correction.

Here we have chosen an example application of image blurring. As seen in Fig 12 the image (d) obtained by using by using the proposed logic and memory error correction schemes looks significantly better than the one without any correction (c). The output Peak Signal to Noise ratio (PSNR) is defined as, $PSNR = 10\log_{10}(\frac{Max^2}{MSE})$.
Here, Max is the maximum possible pixel value of the image and MSE is the mean square error between the error-free and the noisy image. Thus we can obtain two PSNRs: $PSNR\_noisy$ and $PSNR\_corrected$ corresponding to the noisy image, and the corrected image respectively. For the images shown in Fig. 12, the $PSNR\_noisy$ is 17.70 dB and $PSNR\_corrected$ is 28.68 dB. This shows that the corrected image is not only visually closer to the error-free output image but also significantly better in terms of the PSNR.

Fig. 13 shows the PSNR values for the noisy and the corrected images for different values of $P_{mem-err}$ and $P_{log-err}$. From the figure, we can see that even for error-probabilities as high as 0.5, the scheme shows an improvement of around 10 dB in the PSNR.

V. Conclusion

In this paper we presented a FIR filter which is more robust to errors introduced in the output signal due to voltage scaling. To come up with this robust design the total errors in the output signal were classified into logic errors (occurring due to violation of the logic path delay condition) and memory errors (occurring due to failures in the coefficient memory elements). A linear predictor based error correction circuit was employed for detecting

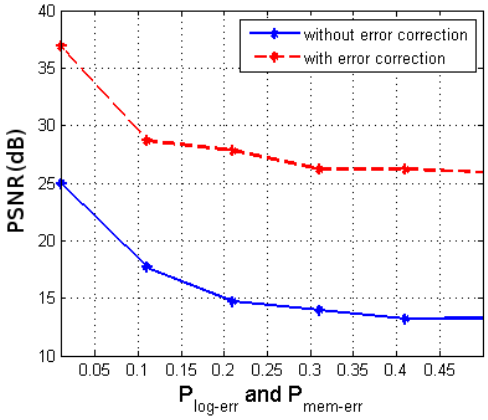

Fig. 13. Variation of PSNR with P_{err} for the Lena Image

and correcting logic errors. A simple feedback based circuit was used to detect and reduce memory failures by adjusting the supply voltage of the coefficient memories. The results indicated a considerable improvement in the output SNR (at least around 10 dB) for a probability of error even as high as 0.5.

References

[1] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power cmos digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.

[2] R. Gonzalez, B. M. Gordon, and M. A. Horowitz, "Supply and threshold voltage scaling for low power cmos," *IEEE J. Solid-State Circuits*, vol. 32, no. 8, pp. 1210–1216, August 1997.

[3] J. M. Rabaey, A. Chandrakasan, and B. Nicolić, *Digital Integrated Circuits (2nd Edition)*. Prentice-Hall.

[4] S. Dhar and D. Maksimović, "Low-power digital filtering using multiple voltage distribution and adaptive voltage scaling (poster session)," in *ISLPED '00: Proceedings of the 2000 international symposium on Low power electronics and design*, Rapallo, Italy, 2000, pp. 207–209.

[5] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, "Low-power realization of fir filters on programmable dsp's," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 6, no. 4, pp. 546–553, 1998.

[6] M. Ashouei, S. Bhattacharya, and A. Chatterjee, "Improving snr for dsm linear systems using probabilistic error correction and state restoration: A comparative study," in *ETS '06: Proceedings of the Eleventh IEEE European Test Symposium*, May 2006, pp. 35–42.

[7] R. Hegde and N. R. Shanbhag, "Soft digital signal processing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 9, no. 6, pp. 813–823, 2001.

[8] *TMS320C2x/C5x Users Guides*, Texas Instruments, 1993.

[9] M. Cho, J. Schlessman, W. Wolf, and S. Mukhopadhyay, "Accuracy-aware sram: a reconfigurable low power sram architecture for mobile multimedia applications," in *ASP-DAC '09: Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, January 2009, pp. 823–828.

[10] D. S. K. Chan and L. R. Rabiner, "Analysis of quantization errors in the direct form for finite impulse response digital filters," *IEEE Trans. Audio Electroacoust.*, vol. 21, no. 4, pp. 354–366, August 1973.

[11] H. A. Spang, III and P. M. Schultheiss, "Reduction of quantizing noise by use of feedback," *IRE Transactions On Communications Systems*, vol. 10, no. 4, pp. 373–380, December 1962.

[12] B. Shim, S. R. Sridhara, and N. R. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 5, pp. 497–510, 2004.

[13] T. F. Quatierei, *Discrete-Time Speech Signal Processing: Principles and Practice*. Prentice-Hall.

[14] W.-S. Lu, H.-P. Wang, and A. Antoniou, "Design of two-dimensional fir digital filters by using the singular-value decomposition," *IEEE Trans. Circuits Syst.*, vol. 31, no. 1, pp. 35 – 46, January 1990.