IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS

Reliable State Retention-Based Embedded Processors Through Monitoring and Recovery

Sheng Yang, Saqib Khursheed, Bashir M. Al-Hashimi, Fellow, IEEE, David Flynn, and Sachin Idgunji

Abstract—State retention power gating and voltage-scaled state retention are two effective design techniques, commonly employed in embedded processors, for reducing idle circuit leakage power. This paper presents a methodology for improving the reliability of embedded processors in the presence of power supply noise and soft errors. A key feature of the method is low cost, which is achieved through reuse of the scan chain for state monitoring, and it is effective because it can correct single and multiple bit errors through hardware and software respectively. To validate the methodology, ARM® Cortex™-M0 embedded microprocessor (provided by our industrial project partner) is implemented in FPGA and further synthesized using 65-nm technology to quantify the cost in terms of area, latency and energy. It is shown that the proposed methodology has a small area overhead (8.6%) with less than 4% worst-case increase in critical path and is capable of detecting and correcting both single bit and multi bit errors for a wide range of fault rates.

Index Terms—Reliability, power gating, voltage-scaled state retention, voltage scaling, state retention, error correction

I. INTRODUCTION AND RELATED WORK

TRANSISTOR and voltage scaling have been the driving force behind the growth of electronic industry. An undesirable side-effect however is the significant increase in leakage power. High-k metal gate [1], dual-threshold standard cells [2] and supply voltage scaling are some of the most popular methods to reduce leakage power of a design [3].

Power gating is effective in reducing leakage power in sleep mode. It utilizes power-switches (also called sleep transistors) to power-down the logic blocks during idle mode to reduce leakage power consumption [4]. State retention power gating turns off the supply voltage of an idle circuit while keeping its (flip-flop) states in state retention latches. A recent study has reported a reduction of 95% leakage power using power gating technique on ARM926EJ processor [3]. However in the sleep mode, the data stored in state retention latches is vulnerable due to rush current and power supply fluctuations. To increase the reliability of power gating designs due to rush currents and supply voltage fluctuations, it was proposed to turn on power transistors slowly by either controlling their gate-to-source voltage or turn on a portion of the power transistors at a given time [5]. In another study, the use of pump capacitors was proposed to slowly turn on the power transistor and use a voltage monitor circuit to detect the end of the activation process [6]. Voltage-scaled state retention reduces idle circuit’s voltage to a point that is sufficient to hold the state, here referred to as drowsy logic. It has been proposed in [7] to reduce cache memory power consumption by putting idle cache line into drowsy mode (lowered supply voltage). The main problem with leakage power reduction techniques using supply voltage scaling is higher error susceptibility of states when the supply voltage is reduced. The relationship between supply voltage and leakage power is well studied [8]. It is shown that the leakage power $P_{leakage}$ decreases exponentially with the supply voltage $V_{DS}$, however reduction in supply voltage makes the flip-flops more susceptible to soft errors.

This is because the critical charge $Q_{crit}$, which is the amount of charge required to flip its value, decreases linearly with the supply voltage [9], and soft error rate increases exponentially with the decrease of $Q_{crit}$ [10], [11]. These two techniques (state retention power gating and drowsy logic) effectively reduce leakage power of a device but have a negative impact on state reliability during sleep mode.

Two recent works have proposed using hybrid hardware plus software approaches to protect against soft errors and hardware defects. The first method uses software-based online detection of hardware defects [12]. This method maps the states of circuit’s flip-flops to the processor’s register file so the software can access the internal states of the processor and a software routine can be used for error detection. This method is effective in detecting the faulty states of a system but it has long detection and correction time due to using software error correction only. The second method uses compiler and micro-architecture hybrid approach to reduce the soft error rate of processor’s register file [13]. It employs hardware to protect a portion of register file and compiles software to maximize the usage of protected register-files. This method is capable of protecting the register-files while the remaining states are left unprotected.

This paper introduces a low cost and effective methodology to protect an idle circuit from power supply noise and rush current in the case of state retention power gating and from increased soft error rate in case of drowsy logic. It is low cost because it reuses scan chains, which are typically implemented in production designs for manufacturing test.
It is effective because it provides two levels of protection: hardware error detection and correction for a speedy error recovery and a software state recovery method for further protection against multi-bit data corruption. The proposed method targets embedded processors, as they usually employ low-power design techniques and at the same time it is desirable to achieve higher reliability at low-cost. The proposed method is functionally validated on ARM Cortex-M0 embedded microprocessor implemented in FPGA. The system is also constructed using 65-nm technology for detailed trade-off analysis in terms of latency, area and energy overhead, which is discussed in Section V.

The motivation behind developing a low-cost and effective state monitoring and recovery method is presented in Section II. The architectural details of scan based state monitoring and recovery method are presented in Section III. Section IV describes in detail how the proposed method can be incorporated in an embedded processor. Experimental results are presented in Section V and conclusions on the applicability of the techniques are drawn in Section VI.

II. MOTIVATION

State retention power gating and voltage scaling are two effective leakage power reduction techniques commonly employed in modern designs. In this section we explain why it is important to protect both of these through a low-cost state monitoring and recovery method.

In modern designs the memories are well-protected against soft errors, however there is an urgent need for protecting logical part of a design as it was shown that due to process scaling, the effect on logical part of a design is exponentially increasing [14]. It is important to protect state retention logic against soft errors because during sleep mode it stores all the states and therefore state corruption during sleep mode can potentially affect rest of the design after it is powered-up. In addition to state corruption due to soft errors, state retention power gating is also affected by rush currents. This is because power supply rails of an integrated circuit are not perfect, wires have resistance and between the wires there is capacitance and inductance. In sleep mode, when the power transistors of the power gated design are off, the internal capacitance of the power gated circuit is discharged to ground and its leakage current is determined by the power transistors. When the power gated circuit is reactivated, the power transistors are turned on, there is a rush current to charge up its internal capacitance. This sudden change of current induces a voltage across the wires which can be modeled as step response of an RLC circuit [5]. The voltage fluctuation at the power supply rails may corrupt the state retention latches connected to it, which lead to potential reliability problem.

Supply voltage scaling is another effective way of reducing leakage current in idle circuits [7], but it comes at the cost of higher error susceptibility [9]. In 65nm technology node the soft error rate increases by 3 times when supply voltage is reduced from 1V to 0.5V [15]. Next, we demonstrate the effect of supply voltage scaling and process variation on error susceptibility of a deep sub-micron design. The critical charge of a storage node determines its sensitivity to noise. Critical charge is the amount of charge required to flip its value resulting in what is called single event upset (SEU) [16]. Critical charge is calculated through charge injection [17], which uses current pulse with different magnitude to inject charge into the storage node. The critical charge is reached when the current pulse causes the stored value to flip. In a recent study, it was shown that critical charge of a flip-flop increases with the increase in supply voltage. When comparing the critical charge at their respective nominal operating voltages, a flip-flop designed using 45-nm technology showed lower critical charge than the one designed with 65-nm technology [17]. Deep sub-micron designs are also affected by process variation, which is mainly due to sub-wavelength lithography, random dopant distribution, line edge roughness and stress engineering [18]. It affects transistor gate length, threshold voltage, oxide thickness and effective mobility. We investigated the effect of supply voltage scaling on the critical charge of a flip-flop in the presence of process variation, using 45-nm CMOS technology with 0.9-V nominal supply voltage setting. The temperature is kept constant at 25°C. The relation between critical charge and supply voltage variation is shown in Fig. 1. As can be seen, the critical charge of the flip-flop reduces with the reduction in supply voltage (10% of nominal supply voltage). The graph also shows the worst-case critical charge of the flip-flop at $1\sigma$, $2\sigma$ and $3\sigma$ variation at different supply voltages. These results demonstrate that the critical charge of a flip-flop is negatively affected not only because of reduction in the supply voltage but also by process variation as shown in Fig. 1. This means that supply voltage scaling and process variation can significantly increase the soft error rate. Therefore this work focuses on state protection of drowsy logic (affected by radiations and voltage scaling) and that of state retention logic (affected by radiations and rush currents) through a low-cost and effective state monitoring and recovery method.

III. PROPOSED METHODOLOGY FOR STATE MONITORING AND RECOVERY

In this work, scan chains of a design are exploited for state monitoring and recovery. The methodology targets designs with two different operation modes: active mode and sleep mode. Active mode (normal mode of operation) is the same for
State recovery is done using either hardware error correction code (CRC or Hamming) [19] and state monitoring and recovery. State monitoring is done using A. State Monitoring and Recovery Block (SMRB)

working of these parts. Finally, the proposed methodology is incorporated in a design and active mode to initiate state monitoring and recovery. The PSMC block monitors the sleep mode and recover its states if the design can still hold their states. However the sleep-active mode transition control is similar for both power gating and drowsy logic. The state monitoring and recovery control can be integrated into conventional sleep-active mode transition control flow.

Fig. 2 shows the overall flow of the proposed methodology to add reliability to an embedded processor employing state retention flip-flops. The flow input is low-power embedded processor and the output is added reliability through the following four parts. The first is State Monitoring and Recovery Block (SMRB), second is Scan Chain Configuration block, third is Power and State Monitoring Controller (PSMC) and the last part is Synthesis Flow. The SMRB monitors the states of the design during sleep mode and recover its states if errors are detected. For a particular type of error detection and correction code, Scan Chain Configuration block is used for configuring scan chains for state monitoring without affecting manufacturing test. The PSMC block monitors the sleep mode and active mode to initiate state monitoring and recovery. Finally, the proposed methodology is incorporated in a design through a synthesis flow by inserting state monitoring and recovery block along with controller for power and state monitoring, and scan chains. In this section, we explain the working of these parts.

A. State Monitoring and Recovery Block (SMRB)

In the proposed methodology, scan chains are reused for state monitoring and recovery. State monitoring is done using hardware error detection code (CRC or Hamming) [19] and state recovery is done using either hardware error correction code (Hamming) or software state recovery. The merits of different coding schemes (Hamming and CRC) are justified in Sec. IV-B and Sec. IV-C. The architecture of state monitoring and recovery block (SMRB) is shown in Fig. 3 which consists of three blocks: the circuit under protection (CUP), the state monitoring block and the error correction block. The scan chain inputs to the CUP are controlled by a multiplexer with a ‘sel’ signal, which allows the scan chains to be recirculated, conditionally corrected (inverted), or as conventional manufacturing test scan inputs. The scan enable, ‘se’, to the CUP is asserted to configure all the registers into scan-shift mode rather than functional mode. Before switching to sleep mode, the state monitoring block encodes the CUP states, where signal ‘sel’ is set to ‘0’ and ‘se’ is set to ‘1’. The CUP is in scan mode, its scan-out ports are connected to its scan-in ports and the state monitoring block. Assume that each scan chain contains ‘l’ flip-flops, by circulating the scan chains for ‘l’ clock cycles, the state monitoring block generates and stores the parity bits of the CUP states. After switching to active mode, the state monitor decodes the states, where signal ‘sel’ is ‘1’ and ‘se’ is ‘1’. The CUP is in scan mode, its scan-in ports are connected to the error correction block, its scan-out ports are connected to the state monitoring block and the error correction block. The state monitoring block checks the states of the CUP against the stored parity bits. When errors are detected state monitoring block sends the error locations to the error correction block which corrects the corrupted states and feeds back to the circuit. In manufacturing test mode, the control signal ‘sel’ is ‘2’ and scan enable signal ‘se’ is controlled by the tester, the CUP’s scan-in ports are connected to the test scan-in ports and its scan-out ports are connected to the test scan-out ports. This means the proposed architecture has no impact on the scan chains during manufacturing test. The methodology has little impact on design’s timing (critical path) in normal mode. This is because all state monitoring is done in scan mode (Sec. V-C for more details). However it has a cost in terms of area overhead, wake-up latency and energy consumption (details are shown in Section V). For the state monitoring block to generate parity bits, the circuit’s states need to be circulated through the scan chains. If the number of registers in each scan chain is ‘l’ and the clock period is ‘T’, the encoding and decoding time will be ‘l × T’. If ‘l’ is large, each encoding and decoding cycle can take a long time and therefore consume a significant amount of energy.


B. Scan Chain Configuration

To reduce latency and energy consumption, shorter scan chains are needed, but at the same time, it will increase the area overhead due to additional state monitoring blocks. To make each scan chain shorter, the number of scan chains can be increased. The scan chains can be configured to reduce encoding and decoding time and can also be reconfigured for standard manufacturing test. This is demonstrated next through an example. Assume the test scan width (I/O width for manufacturing test) is 4 bits and the state monitoring block employs Hamming (7,4) code to monitor circuit’s states, with input width of 4 bits per state monitoring block. A circuit with 128 flip-flops and 4 scan chains (1 test input bit per scan chain) will take \((128 ÷ 4) = 32\) clock cycles for encoding and decoding the data. This is because state monitoring block’s width is 4 bits. Next, consider that 128 flip-flops are re-ordered into 16 scan chains allowing 4 state monitoring blocks to work in parallel. The number of encoding and decoding clock cycles will then be \((128 ÷ 16) = 8\), resulting in 4x speed-up. The configuration with 16 scan chains is shown in Fig. 4 (a), as can be seen, 4 state monitoring blocks are operating in parallel taking data from 128 flip-flops configured in 16 scan chains. Fig. 4 (b) shows how this configuration can be reused for 4 bit scan chain operation during manufacturing test, the output of So\([15:12]\) is fed back to Si\([7:4]\) and so on, until test data is scanned out through So\([15:12]\). This is why the proposed method does not affect pin count of the design. The detailed area, energy and latency trade-offs related to scan chain configuration is discussed in Section V.

C. Power and State Monitoring Controller (PSMC)

Fig. 5 (a) shows the conventional sleep control flow. Assume a circuit starts from active mode. When signal ‘sleep’ is ‘1’ sleep controller starts the sleep sequence, for power gating design it includes saving the circuit’s states and turning off the sleep transistors while for drowsy logic the supply voltage is scaled down, then the circuit enters the sleep mode. When signal ‘sleep’ is ‘0’ it starts the wake-up sequence, for power gating design it includes turning on the power transistors and restoring the circuit’s states when the power supply become stable while for drowsy logic the power supply is restored and the circuit enters active mode. Fig. 5 (b) shows how the state monitoring control can be integrated into the conventional sleep control flow, where parity bits are generated and stored before the sleep sequence, and the circuit’s states are checked against the stored parity bits after the wake-up sequence.

D. Synthesis Flow

Fig. 6 shows the design synthesis flow for incorporating the proposed state monitoring and recovery methodology using conventional design flow. It has three inputs, which includes the original design, the scan chain configuration file, and templates for SMRB (Fig. 3) and PSMC (Fig. 5). The synthesis flow consists of four main steps: it first inserts scan chains into the original design, the scan chain configuration file, and storage, the added circuitry will require additional test vectors to ensure its correct functionality. In terms of design validation, additional operation modes such as encoding and decoding of the proposed methodology need to be validated to ensure correct design functionality.
The proposed hardware based error detection and correction methodology (Section III) has been validated using a 32X32-FIFO design implemented in FPGA [20]. It was shown that all single bit errors are correctly detected and corrected, however it does not guarantee 100% error correction, for example in case of clustered errors (multi-bit errors per codeword). In this work, the methodology is extended by incorporating software based state recovery method to handle such multiple bit errors. The practicality of the proposed methodology is further demonstrated on an embedded system processor core.

The State Monitoring and Recovery Block (SMRB) together with Power and State Monitoring Controller (PSMC), as discussed in the previous section are implemented in the embedded system (Fig. 7). As can be seen from Fig. 7 the components of the embedded system are divided into three main power domains (PD). PD-1 is associated with functional blocks that are always-ON, which includes SRAM, error register, data bus, I/O peripherals, power and state monitoring controller (PSMC) and interrupt controller. PSMC and interrupt controller are required to be active during sleep mode to respond to external interrupt requests. SRAM, error register, data bus and I/O peripherals are not protected by the SMRB and to ensure their correct operation, they are placed in always-on power domain (PD-1). Embedded processor core is placed in PD-2, which is powered-up only during active mode. Finally, PD-3 is associated with the state monitoring and recovery block (SMRB), which is where the parity bits are stored during sleep mode and is active only during encoding and decoding to ensure processor’s states integrity. PSMC controls the power up and power down sequence of the whole embedded system (Fig. 7). It also controls the SMRB, which monitors the processor states during sleep mode and recovers the faulty states when an error is detected. All errors detected by the state monitoring block but not correctable using hardware are recorded in the memory mapped error register, so the processor has access to it through the data bus.

Fig. 8 shows how power and state monitoring control flow (Fig. 5) is implemented in the embedded system. The wakeup sequence is initiated by the “Peripherals” which sends an interrupt request through the “Interrupt Controller” to the PSMC. The PSMC first turns on the power supply of PD-2 and PD-3, and then initiates the state “Decoding” through the SMRB. SMRB re-generates the parity of the processor states and compares it with the stored parity bits (generated before going to sleep mode). In case of a mismatch (error detection), the following sequence of events occur: firstly, an attempt is made to recover the states through Hamming code, and if successful, the corrected states are fed back through the processor scan-in port. Secondly, the state integrity is checked by comparing the parity generated by the CRC code of the corrected states (through Hamming) with the one generated before going to sleep mode. In case of a mismatch, which indicates that the hardware error correction is not successful, this event is registered in error register. Now the processor core is in an unreliable state so must be reset. The reset handler is a software routine, which will check the reset entry reason by examining the error register, and if it is due to state corruption, it will branch to the software recovery routine before servicing the wake-up event. After the processor’s architectural states are recovered, the processor is redirected to the appropriate instruction for the active mode of operation. Software recovery is executed by a set of instructions that are loaded in the ROM of embedded processor. In case of Cortex-M0 embedded processor, the instructions for software state recovery require 184 clock cycles.

IV. RELIABLE STATE RETENTION EMBEDDED PROCESSOR

A. Error Injection and Detection

The error rate is application, environment and implementation dependent [14], this is why we have considered a wide range of bit error rate in this work from $10^{-12}$ to $10^{-1}$ errors per bit-hour. When considering soft errors, a study has reported that the error rate is in the range of $10^{-12}$ to $10^{-7}$ errors per bit-hour [21]. It is well-known that there is an exponential relationship between soft error rate (SER) and critical charge [15]. A recent publication [14] has indicated that logic SER per chip is expected to rise per technology generation if additional mitigation techniques are not implemented. In addition, power supply fluctuations and rush currents may also induce clustered errors, which may lead to higher error rates. This is why we
errors are generated using a model presented in [24]:

\[ P_i = \gamma \cdot \sum_{j=1}^{N_{ini}} \left( \frac{1}{d_{ij}} \right) ^\alpha \]  

(1)

Where, \( P_i \) is the error probability of each node, \( d_{ij} \) is the distance between two flip-flops \( i \) and \( j \), \( \alpha \) is the clustering factor, higher \( \alpha \) implies higher clustering. We varied \( \alpha \) from 0 to 2 in our experiments, where 0 represents uniform distribution of errors (Fig. 9a) and \( \alpha > 0 \) represents clustered errors (Fig. 9b). \( N_{ini} \) is the number of initial errors and \( \gamma \) is the normalization factor, which is calculated using the following equation:

\[ \gamma = \frac{\epsilon_{tg}}{N_{rst}} \cdot \sum_{i=1}^{N_{rst}} \sum_{j=1}^{N_{ini}} \left( \frac{1}{d_{ij}} \right) ^\alpha \]  

(2)

Where, \( \epsilon_{tg} \) is the target error rate, \( N_{rst} \) is the number of uncorrupted bits after initial error injection. The number of clusters is decided by the initial injected errors \( N_{ini} \), given a specific error rate for a design, \( N_{ini} \) is inversely proportional to the number of errors per cluster. In the case-study, \( N_{ini} \) is chosen to be 10% of total errors.

The error injection methodology is shown in Algorithm 1, which consists of two stages. First initial errors \( N_{ini} \) are evenly injected (step-1 to step-7), which is kept at 10% of target error rate \( \epsilon_{tg} \). Next, in the second phase, the error probability \( P_i \) of all flip-flops in the system is recalculated and then clustered errors are injected (step-8 to step-14).

**Algorithm 1 Methodology for Error Injection**

1: for \( i = 1 \) to \( N \) do
2: \( P_{init} = \epsilon_{tg} \cdot \beta \)
3: \( / / P_{init} \) is the initial injection ratio
4: \( / / \epsilon_{tg} \) is target error rate, \( \beta \) is the initial injection ratio
5: \( / / N \) represents the total number of flip-flops
6: \( R_i \leftarrow \) Generate a random number
7: if \( R_i < P_{init} \) then
8: \( \text{Inject error at node } i \)
9: end if
10: end for
11: for \( i = 1 \) to \( N_{rst} \) do
12: Calculate \( P_i \)
13: \( / / P_i \) is calculated using Eq. (1) and Eq. (2)
14: \( R_i \leftarrow \) Generate a random number
15: if \( R_i < P_i \) then
16: \( \text{Inject error at node } i \)
17: end if
18: end for

Those error distribution patterns are generated on a workstation in the form of memory maps indicating error locations. To inject errors at specific location in the design, the memory map is transferred into the FPGA test bench through UART as shown in Fig. 10, where fault injection registers hold the error location map, which is XORed with the states of the protected circuit through scan chains.

**B. Hardware State Recovery**

Hamming code can achieve single error tolerance in each codeword, it can correct multiple errors in a circuit if these errors are not in the same codeword. It has small encoder and decoder area overhead and fast encoding and decoding. If the bit error rate is \( \lambda \) and there are \( R \) flip-flops in the system, \( n \) is the size of the code word and \( k \) is the size of the message,
Random error

Fig. 10: Error injection setup

Fig. 11: Bit error rate vs. failure rate for Cortex-M0 processor
with and without protection

for example Hamming (63,57) code have \( n = 63 \) and \( k = 57 \). The unprotected system failure rate is:

\[
\text{System Failure Rate} \approx \lambda \cdot R
\]  

(3)

The number of times a state is checked affects state integrity of a storage element, state integrity of a flop that is checked once a day is lower than the one checked every minute. This concept is called ECC (Error Correction Code) scrubbing [25] and the checking period is termed as scrubbing interval. In our study this scrubbing interval is equal to sleep duration. Assuming every particle strike only cause single bit upset and at sleep frequency of 0.1 Hz (average sleep duration or scrubbing interval up to 10 seconds), the system failure rate of a protected system is given by [25]:

\[
\text{Protected System Failure Rate} \approx 0.5 \cdot \frac{R}{k} \cdot \lambda^2 n^2
\]  

(4)

Using Eqs. 3 and 4, the system failure rate (SFR) can be estimated, Fig. 11 shows how Hamming (63,57) code reduces the SFR under different soft error rates (SER) with the range between \(10^{-12}\) to \(10^{-7}\) (err/bit-hr), this range is based on a study presented in [21]. Supply voltage scaling increases the soft error rate due to reduction in critical charge [9]. Using the proposed technique, the reliability of the idle circuit can be improved to compensate the reduction in reliability due to supply voltage scaling. This can be demonstrated through the following example. In the case study of Cortex-M0, the system failure rate (SFR) is plotted against bit error rate (BER) in Fig. 11. It is assumed that a system operating at 1V supply and its bit error rate \( \lambda_{1V} \) is \(10^{-7}\). From Fig. 11 it can be seen that for an unprotected system BER \( \lambda_{1V} \) of \(10^{-7}\) corresponds to SFR \( f \) of \(10^{-4}\). It was shown in [15] that BER increases by 3 times when supply voltage is reduced from 1V to 0.5V for a 65-nm technology node. This means that the BER at 0.5V \( \lambda_{0.5V} \) is \(3 \times 10^{-7}\). Using the proposed hardware correction through Hamming (63,57) code, it can be seen that even at BER \( \lambda_{0.5V} \) of \(3 \times 10^{-7}\) the SFR \( f_p \) is significantly lower \((10^{-8})\) than the SFR \( f \) at 1V \((10^{-4})\) of an unprotected system. Instead of using normal flop, we also considered using radiation-hardened flip-flop, (Single Event Upset Tolerant (SEUT) architecture in [14]) recently reported by Intel and compared the two techniques: SEUT flops in comparison to normal flops protected by Hamming code, assuming same scan chain configuration (for example, 57 parallel scan chains). In terms of reliability, the proposed technique (normal flops protected by Hamming Code) performs better than the one using SEUT flops. It is reported in [14] that using SEUT flops, the improvement in reliability is 30-times in comparison to a normal flip-flop. Fig. 11 can be used for reliability comparison of the two techniques. For example, at \(10^{-7}\) errors per bit-hour, the SFR of a normal flip-flop is \(10^{-4}\), which reduces to \(\frac{10^{-7}}{33} = 33 \times 10^{-6}\) using SEUT flops, however SFR reduces further to \(10^{-10}\) using the proposed technique. The impact of the proposed protection method on circuit idle power is also analyzed in Section V-C. It is shown in Fig. 11 that Hamming code is effective in protecting the system when SER is between \(10^{-12}\) to \(10^{-7}\), however it starts to lose effectiveness for bit error rates above \(10^{-5}\). It should be noted that the reliability of the stored parity bits is taken into account by the error detection (CRC) and correction (Hamming and software state recovery) and therefore they are protected.

C. Software state recovery

Software state recovery is invoked when error correction through Hamming fails due to more than one bit error in the same codeword. We have used CRC-16 in this work for error detection because it is well-known for its high detection capability, low area overhead and simple design [26]. The polynomial used in this work is the same as the one used to encode data in USB 2.0 [27], that is \((x^{16} + x^{15} + x^2 + 1)\). To enable faster execution parallel-CRC implementation [28]
is used. We conducted an experiment to analyze the effect of higher error rates (5 \times 10^{-3} to 55 \times 10^{-3}) on error detection capability of CRC-16 and error detection and correction capability of different Hamming codes. Two different error distribution coefficients were used (Eq. 1). Fig. 12 shows the effect of evenly distributed errors (\alpha = 0) and the effect of clustered errors that are generated using \alpha = 2. It can be seen from the figure that the error detection capability of CRC-16 remains unaffected across higher error rates and clustered errors, however error detection and correction capability of Hamming (7,4) reduces significantly as error rate increases, which is further deteriorated when clustered errors are injected. It should be noted that conventional scan chain configuration is used in this work, where neighboring flops are connected together. The impact of clustered errors can be reduced by configuring scan chains to achieve maximum physical separation across parallel scan chains. In this work clustered errors are dealt with CRC detection and software state recovery. For software state recovery, the corrupted states are recovered through software checkpoints [29]. It is implemented by saving the architectural states before going to sleep mode. The states are placed in the SRAM for software state recovery, which is in “Always ON” power domain as shown in Fig. 7. In case of multiple errors in the same codeword, the errors are detected by CRC-16 and software recovery is invoked that restores the saved states of the processor, subsequently all detected errors are corrected by software state recovery. This clearly demonstrates the effectiveness of the proposed software state recovery method.

This work exploits the advantages of both techniques (hardware error correction and software state recovery) to achieve an effective state monitoring and recovery. For example, hardware error correction (Hamming code) is faster than software state recovery technique. On the other hand, software recovery technique has negligible area overhead and it can be used to recover burst errors, which cannot be corrected using Hamming code. It should be noted that in case of software recovery, the correction energy and latency cost is incurred only when errors are detected. The detailed trade-off analysis of the two techniques is discussed in the next section.

V. TRADE OFF ANALYSIS

The proposed technique utilizes error detection through CRC-16 and single-bit error correction through Hamming code, where CRC-16 and Hamming code are implemented in hardware; multi-bit error correction is achieved through software state recovery. The use of hardware and software has an overhead in terms of area, wake-up latency and energy consumption, which is analyzed in this section. Furthermore, its impact on leakage power saving is also evaluated. We analyzed the trade-offs on an embedded processor shown in Fig. 7. As a case-study, we have used ARM Cortex-M0 processor core provided by our industrial project partner. It is chosen because it is a typical low-power embedded processor with state-retention sleep mode, separate power-domains and utilizes scan chains for manufacturing test. The deep sleep mode has not been considered, because during this mode the whole circuit is completely powered down and there is no state integrity problem. The embedded system with the proposed state monitoring and recovery technique (Fig. 7) is synthesized using a 65-nm STMicroelectronics gate library and Synopsys design compiler.

This section is organized as follows: a generic set of equations to evaluate the implementation cost (area, latency and energy consumption) of the proposed technique is presented in Sec. V-A. The overhead due to CRC-16 and different Hamming codes on a synthesized design are discussed in Sec. V-B and Sec. V-C respectively. The cost of using software state recovery is discussed in Sec. V-D and finally the impact on leakage power saving is evaluated in Sec. V-E.

A. Generic set of equations to evaluate overhead on area, latency and energy consumption

The proposed scan based state protection methodology affects the following three parameters: firstly, it increases wake up latency that is the time required for circuit to switch from sleep mode to active mode; secondly, it incurs some area overhead due to the additional coding circuitry and parity storage; lastly, it consumes energy in every sleep and wake up cycle. The generalized relation between coding scheme selection, scan chain configuration and the overhead is shown in this section.

For both Hamming and CRC code, additional coding circuitry and parity storage lead to area overhead, which can be estimated using the following equation

\[ A_{ov} = N_c \times A_c + N_s \times A_s + A_{PSMC} \]  

(5)

Where \( A_c \) is the area of a single coder, \( N_c \) is number of coders used, \( A_s \) is the area of a single storage unit and \( N_s \) is the number of parity storage units, and \( A_{PSMC} \) is the area of the PSMC (Power and State Monitoring Control) block. Number of coders is proportional to the number of parallel scan chains. The number of parity storage units is proportional to the number of flip-flops in a design and the coding scheme used. Eq. (5) can be rewritten as

\[ A_{ov} = \frac{W_{sc}}{k} \times A_c + \frac{N_{ff} \cdot d}{k} \times A_s + A_{PSMC} \]  

(6)

Where \( W_{sc} \) is the number of parallel scan chains created, \( N_{ff} \) is the number of flip-flops in a design. For any error correction code, a codeword consists of data bits for storing information and parity bits for protecting the information, \( k \) is the number of data bits and \( d \) is the number of parity bits in a codeword. It shows that the area overhead increases with the number of parallel scan chains \( W_{sc} \), due to additional encoder and decoder. The area overhead also increases with the ratio of parity bit to data bit of the selected coding scheme due to redundancy storage (more details in Sec. V-B and Sec. V-C).

The circuit states needs to be circulated to generate parity bits, the number of clock cycles required are equal to the number of flip-flops in each scan chains \( L_{sc} \). If the clock period is \( T_{clk} \), the increase in wake up latency can be calculated with

\[ T_{ov} = T_{clk} \times L_{sc} \]  

(7)
As expected, the hardware error detection and correction latency increases with the number of flip-flops in each scan chain. The power overhead for hardware error detection and correction has four components: scan shifting of circuit states $P_{sc}$ (including switching power of connected gates), the operation of encoder and decoder $P_c$, parity storage power per bit $P_s$ and the PSMC block $P_{PSMC}$

$$P_{ov} = P_{sc} + N_s \times P_s + N_c \times P_c + P_{PSMC}$$  

$$= P_{sc} + \frac{W_{sc}}{k} \times P_c + \frac{N_{ff} \cdot d}{k} \times P_s + P_{PSMC}$$

The power overhead increases with number of parallel scan chains because more coders are working in parallel. From Eq. (7) and (9), energy consumed in each sleep and wake up cycle can be calculated by

$$E_{ov} = P_{ov} \times T_{ov}$$

The above equation shows that the total energy is proportional to the number of flip-flops $L_{sc}$ in each scan chain (more details with example in Sec. V-B and Sec. V-C). The above set of equations can be solved for any embedded processor to determine the overhead in terms of area, power, latency and energy. Table I shows the values of these parameters for Cortex M0 embedded processor (synthesized using 65-nm STMicroelectronics gate library) for all different Hamming codes used in this work. For example, Hamming (63,57) has 57 data bits (k), 6 parity bits (p). The area and power of the coder ($A_c$ and $P_c$) reduces with each Hamming code, that is from Hamming (63,57) to Hamming (7,4) due to reduction in parity generation tree. The rest of the variables are constant across all Hamming codes and their approximate values are shown in the last row of Table I.

### B. Overhead due to CRC error detection

As discussed in Sec. IV-C, CRC-16 is chosen for error detection because of high detection capability, low redundancy and simple encoder and decoder implementation. Table II shows the overhead on area, latency and energy consumption using four different scan chain configurations in comparison to a design without CRC unit. It can be seen that the area overhead increases linearly with the number of parallel scan chains (“Width”), from 3.5% to 6.1% as in case of 16 to 80 scan chains respectively. This is because additional logic is required for encoding and decoding each additional scan chain through state monitoring and recovery block as shown in Fig. 3. From system level view point, the proposed scheme has little affect on the critical path of the design, this is further discussed in Sec. V-C. However it requires some additional clock cycles (14 as in case of 57 parallel scan chains, see Table III, column 3, row 4) after switching from sleep to active mode to ensure data integrity, but that does not affect normal operation of the design. The next column (“Latency”) shows the sleep and wakeup latency in number of clock cycles, which depends on scan chain length, longer scan chains require longer time to encode and decode the circuit’s states. As expected the coding latency reduces from 49 to 10 at the cost of area overhead, which increases from 3.5% to 6.1%. The next column shows the power consumption, which is estimated by Synopsys PrimeTime PX (using VCD file). It increases with the number of scan chains due to additional logic in state monitoring blocks to encode more scan chains in parallel. The last column shows the overall energy consumed by the CRC protected system. The majority of energy consumption comes from shifting of scan chains including the gate switching power connected to each scan chain. This scan shifting power dissipation can be reduced through combinational logic isolation [30]. Power dissipation due to interconnect parasitic (RC) is not taken into account, which can be added through layout extraction of the synthesized design. It can be seen that the energy consumption reduces from 1.27-nJ to 0.27-nJ with reduction in scan chain length from 49 to 10. This is because energy consumption reduces with latency (number of clock cycles), which is dependent on scan chain length.

### C. Overhead due to Hamming error detection and correction

Next, the implementation trade-off is analyzed using four different Hamming codes where each is implemented with different scan chain configurations. The results are shown in Table III. The maximum increase in critical path under timing constraints is less than 4% over all Hamming codes including all scan chain configurations (Table III). It is little affected because the logic depth of the critical path does not increase and the system core is separated from the monitoring and recovery unit (Fig. 4(a)) through a clamp gate which is inactive during functional mode and adds very small load capacitance on flip-flops connected to the state monitoring block. Area overhead is shown in the next column, it can be seen that Hamming (63,57) has the smallest area overhead (8.6%) in comparison to a design without error correction capability. Hamming code requires redundancy for code storage, which increases from Hamming (63,57) to Hamming (7,4) resulting in higher area overhead (upto 30.9%). The area overhead can be estimated using eq. (6) and the parameters listed in Table I. For example in case of Hamming (63,57) with 57 parallel scan chains ($W_{sc}$), by substituting the parameters in the same order

<table>
<thead>
<tr>
<th>Code</th>
<th>k</th>
<th>d</th>
<th>$A_c$ ($\mu$m²)</th>
<th>$P_c$ ($\mu$W/MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming (63,57)</td>
<td>57</td>
<td>6</td>
<td>665</td>
<td>0.47</td>
</tr>
<tr>
<td>Hamming (31,26)</td>
<td>26</td>
<td>5</td>
<td>282</td>
<td>0.25</td>
</tr>
<tr>
<td>Hamming (15,11)</td>
<td>11</td>
<td>4</td>
<td>119</td>
<td>0.15</td>
</tr>
<tr>
<td>Hamming (7,4)</td>
<td>4</td>
<td>3</td>
<td>36</td>
<td>0.06</td>
</tr>
</tbody>
</table>

$k \rightarrow$ Data bits, $d \rightarrow$ Parity bits, $A_c \rightarrow$ Coder Area $P_c \rightarrow$ Coder Power, $P_s \rightarrow$ Storage Power $A_{PSMC} \rightarrow$ PSMC Area, $P_{PSMC} \rightarrow$ PSMC Power
as listed in Eq. (6), the area overhead is \( \frac{57}{8} \times 665 + \frac{768.6}{8} \times 19.5 + 900 = 3141.4 \mu m^2 \). This leads to estimated area overhead of \( \frac{3141.4}{37814} = 8.3\% \), where 37,814 \( \mu m^2 \) is the area of the original design. It can be seen from Table III that it matches closely to the exact area overhead (8.6\%), as calculated by Synopsys design compiler. The next column shows the sleep and wakeup latency in number of clock cycles, as expected the coding latency varies with the scan chain length from 7 to 196 as in case of Hamming (63,57) and Hamming (7,4). The power consumption is shown next, which is lowest (27.69 \( \mu W \) per MHz) in case of Hamming (63,57) code, but increases with different Hamming codes due to additional load capacitance of redundant storage elements, that is when the redundancy increases from 8.6\% to 30.9\% as in case of Hamming (63,57) to Hamming (7,4). The power consumption of state monitoring and recovery can be estimated by using eq. (9) and the parameters listed in Table I. For example in case of Hamming (63,57) with 57 parallel scan chains (\( W_{sc} \)), by substituting the parameters in the same order as listed in Eq. (9), the overhead is \( 25.8 + \frac{57}{8} \times 0.47 + \frac{768.6}{8} \times 9.8 \times 10^{-3} + 0.3 = 27.36 \mu W \) per MHz. It can be seen from Table III that the estimation matches closely to the one (27.69 \( \mu W \) per MHz) calculated by Synopsys design compiler. In terms of power consumption, the main contributor is shifting of scan chains due to gate switching power connected to each scan cell. It is pattern (CUP states) dependent and in the worst case, it can lead to toggling of each flip-flop and gates connected to each one of them, in every clock cycle. The scan power reduction technique for toggle suppression [30] can also be used to reduce the state monitoring power dissipation to avoid exceeding the power distribution for the CUP. The last column (Table III) shows the energy consumption, as expected it varies with the scan chain length resulting in lower latency and energy consumed for encoding and decoding. It can be seen that the lowest and the highest energy consumption are 0.2-nJ and 6.09-nJ, for scan chain length of 7 and 196 respectively.

The results presented in Table II and Table III demonstrate the trade-off between energy consumption, latency and area overhead across different scan chain configurations. Increasing the number of scan chains increases the area overhead and coding power however the wake-up latency and energy consumption is reduced. From the two tables it can be observed that scan chain configuration of 57x14 gives the best results. Area overhead is 8.6\% as in case of Hamming (63,57) (Table III) and 5.3\% as in case of CRC-16 (Table II), since this configuration requires only 14 clock cycles for state monitoring and recovery, the energy overhead is 0.37-nJ and 0.39-nJ in case of Hamming (63,57) and CRC-16 respectively.

The results shown in Table II and Table III are specific to ARM Cortex M0 embedded processors, but the trend observed is also valid for other embedded processors. That is higher number of parallel scan chains minimize wake-up latency and energy consumption at a small cost of additional area, when considering hardware error detection and correction through CRC and Hamming codes.

D. Overhead due to software state recovery

Next we discuss the overhead of software state recovery using the scan chain configuration (57x14) and compare it with Hamming (63,57) code. In terms of area it has 5.3\% overhead (Table II) due to using CRC-16 for error detection and the state recovery power is 13.69-\( \mu W \) per MHz (obtained using Synopsys PrimeTime PX). Once errors are detected, the software state recovery has additional wakeup latency of 184 clock cycles and energy overhead of 2.52-nJ (184 \times 10^{-6} \times 13.69 \times 10^{-6}). In comparison to Hamming (63,57), software state recovery introduces 14-times higher wakeup latency (184+14 \times 10^{-6}) and for CRC-16 (Table II) and for Hamming (63,57) (Table III) when scan chain length is 14 and consumes about 6.5-times more energy (184+14 \times 10^{-6}) in comparison.

Hardware error correction uses Hamming code for both error detection and correction. It is most effective against single event upset for the error rate between \( 10^{-12} \) and \( 10^{-7} \) errors per bit-hour which is the soft error rate range for the 65nm, 90nm and 130nm technology nodes [21]. Hardware error correction requires larger area overhead (8.6\% in case of Hamming (63,57)) but have shorter sleep and wakeup latency compared to software state recovery. Software state recovery relies on CRC for error detection and uses software check point for state recovery. It is most effective against multiple errors and higher error rates (greater than \( 10^{-7} \) errors per bit-hour), due to using CRC-16 for error detection (Fig. 12). However the software recovery has higher wakeup latency and consumes more energy in each sleep cycle.

E. Impact on leakage power saving

Error correction has encoding and decoding energy overhead in each sleep cycle and leakage power overhead due to the additional circuitry for coder and code storage, which adds to the idle power of the design. Fig. 13 shows the trade off between idle power and sleep frequency (number of sleep and wake-up cycles per second) of the processor core with and

<table>
<thead>
<tr>
<th>Code</th>
<th>Scan Chain</th>
<th>Cell Area</th>
<th>Latency</th>
<th>Power per MHz</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Width</td>
<td>Length</td>
<td>inc(%)</td>
<td>clock cycles</td>
<td>(( \mu W ))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(( \mu u m^2 ))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Without CRC</td>
<td>16</td>
<td>49</td>
<td>3.5</td>
<td>49</td>
<td>26.9</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>25</td>
<td>4.0</td>
<td>25</td>
<td>27.04</td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>14</td>
<td>5.3</td>
<td>14</td>
<td>27.29</td>
</tr>
<tr>
<td>CRC-16</td>
<td>80</td>
<td>10</td>
<td>6.1</td>
<td>10</td>
<td>27.52</td>
</tr>
</tbody>
</table>
TABLE III: Area, Latency, Power and Energy overhead due to Hamming codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Width</th>
<th>Length</th>
<th>( \mu m^2 )</th>
<th>inc(%)</th>
<th>Latency</th>
<th>Power per MHz</th>
<th>Energy (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Hamming</td>
<td>( W_{sc} )</td>
<td>( L_{sc} )</td>
<td>37814</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamming (63,57)</td>
<td>57</td>
<td>14</td>
<td>41070</td>
<td>8.6</td>
<td>14</td>
<td>27.69</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td>114</td>
<td>7</td>
<td>41734</td>
<td>10.4</td>
<td>7</td>
<td>28.46</td>
<td>0.2</td>
</tr>
<tr>
<td>Hamming (31,26)</td>
<td>26</td>
<td>31</td>
<td>42861</td>
<td>13.4</td>
<td>31</td>
<td>28.15</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>52</td>
<td>16</td>
<td>43145</td>
<td>14.1</td>
<td>16</td>
<td>28.46</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>8</td>
<td>43997</td>
<td>16.4</td>
<td>8</td>
<td>29.38</td>
<td>0.24</td>
</tr>
<tr>
<td>Hamming (15,11)</td>
<td>11</td>
<td>72</td>
<td>43829</td>
<td>15.9</td>
<td>72</td>
<td>28.92</td>
<td>2.10</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>36</td>
<td>43948</td>
<td>16.2</td>
<td>36</td>
<td>29.53</td>
<td>1.03</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>18</td>
<td>44305</td>
<td>17.2</td>
<td>18</td>
<td>29.84</td>
<td>0.54</td>
</tr>
<tr>
<td>Hamming (7,4)</td>
<td>4</td>
<td>196</td>
<td>49357</td>
<td>30.5</td>
<td>196</td>
<td>31.07</td>
<td>6.09</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>98</td>
<td>49393</td>
<td>30.6</td>
<td>98</td>
<td>31.38</td>
<td>3.04</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>49</td>
<td>49501</td>
<td>30.9</td>
<td>49</td>
<td>32.15</td>
<td>1.57</td>
</tr>
</tbody>
</table>

Fig. 13: Impact of the proposed methodology on processor core idle power

without protection taking into account all sources of leakage in the design, including the storage of parity bits during sleep mode. For illustration purposes, we have used 20% duty cycles (activity factors) and the design is operating at 66 MHz. Activity factor is calculated by taking the ratio of total active time to total runtime of the processor. The leakage power of the processor core in sleep mode is estimated by using Synopsys PrimeTime PX. Fig. 13 shows the relation between idle power and sleep frequency for retention voltage \( V_{ret} \) of 1V and 0.5V with and without protection. Four different scenarios are shown: \( V_{ret} = 1V \) without protection, \( V_{ret} = 0.5V \) without protection, \( V_{ret} = 0.5V \) with hardware error correction and \( V_{ret} = 0.5V \) with software state recovery. The idle power was normalized to that of unprotected system with \( V_{ret} = 1V \). It can be observed that the idle power is lowest for an unprotected system with retention voltage \( V_{ret} = 0.5V \) when compared with hardware error correction and software state recovery. This is because hardware error correction and software state recovery require additional circuitry to improve system reliability. When comparing the two recovery methods, software recovery has higher idle power, which is due to its higher latency (number of clock cycles for state recovery) in comparison to hardware recovery method. For example at sleep frequency of 100 Hz (100 sleep and wake-up cycles per sec). Fig. 13 shows that hardware error correction has 0.21 idle power while software recovery has 0.4 idle power. For these two error recovery methods, idle power is also related to sleep frequencies. Idle power is similar with or without protection until sleep frequency is greater than 10 Hz, this is because when number of sleep and wake-up cycles is small the energy used for error checking is negligible. For software state recovery, when the sleep frequency is higher than 400 Hz its idle power is greater than the unprotected system with \( V_{ret} = 1V \) (without voltage scaling), which means it consume more energy than saved.

VI. CONCLUSION

In this paper a low-cost and efficient state monitoring and recovery methodology is proposed for improving the reliability of an embedded processor during sleep mode. It achieves low overhead in terms of latency, area and energy consumption (Table II and Table III). Low area-overhead is because it reuses scan chains that are used in designs for manufacturing test. It has little affect on critical path (worst-case is less than 4%) because of isolating state protection logic through a clamp gate during functional mode. Energy efficiency is achieved by partitioning scan chains to minimize the time needed for state monitoring and recovery. The proposed solution is effective because it uses two methods of error recovery. First method of error recovery is through hardware implementation of Hamming code that is capable of 1-bit error correction per code word; to recover multi-bit (burst) errors, it employs software state recovery method that restores the last known good states of the design. It is shown that for bit error rates up to \( 10^{-7} \) errors per bit-hour (maximum soft error rate observed at normal operating conditions), the hardware based Hamming error correction improves the system reliability by more than 2 order of magnitude compared to a design without error protection (Fig. 11). However, the system failure rate increases significantly at higher bit error rate and in case of clustered errors (multi-bit error per code word), software state recovery method is incorporated to achieve efficient state recovery. An industry-standard embedded microprocessor ARM® Cortex™-M0 is used to demonstrate that the proposed state monitoring and
recovery method can be incorporated into conventional design flow (Fig. 6). It has been validated on an FPGA and further synthesized using 65-nm technology library using Synopsys EDA tool suite.

ACKNOWLEDGMENT

The authors would like to thank Dr. Harry Oldham (ARM Fellow) and the anonymous reviewers for their valuable feedback that helped improve the quality of manuscript.

REFERENCES


Sheng Yang received the B.E degree in Electronic Engineering from University of Southampton, UK, in 2008. He is currently working towards his Ph.D. degree in University of Southampton, UK. His research interests include: error resilient embedded system design, hardware/software codesign and computer architecture.

Saqib Khursheed received the B.E. degree in Computer Engineering from NED University, Karachi, Pakistan, in 2001, the M.Sc. degree in Computer Engineering from King Fahd University (KFUPM), Dhahran, Saudi Arabia, in 2004, and the Ph.D. degree in Electronics and Electrical Engineering from University of Southampton, Southampton, U.K., in 2010. From 2005 to 2007, he served as a Lecturer with KFUPM. Currently he is working as a Research Fellow in the School of Electronics and Computer Science, University of Southampton. He is the program co-chair of DATE 2012 Friday Workshop on 3D ICs. His research interests include fault modeling, diagnosis, reliability and test generation for deep submicron designs.
Bashir M. Al-Hashimi (M’99-SM’01-F’09) received the B.Sc. degree (with 1st-class classification) in Electrical and Electronics Engineering from the University of Bath, UK, in 1984 and the Ph.D. degree from York University, UK, in 1989. Following this he worked in the microelectronics design industry and in 1999, he joined the School of Electronics and Computer Science, Southampton University, UK, where he is currently a Professor of Computer Engineering and Director of the Pervasive Systems Center. He has authored one book on SPICE simulation, (CRC Press, 1995), and coauthored two books, Power Constrained Testing of VLSI circuits (Springer, 2002), and System-Level Design Techniques for Energy-Efficient Embedded Systems (Springer, 2004). He edited the book, System-on-Chip: Next Generation Electronics (IEE Press, 2006). He has published over 240 papers in journals and refereed conference proceedings. Prof. Al-Hashimi is a Fellow of the IEE and a Fellow of the British Computer Society. He is the Editor-in-Chief of the IEE Proceedings: Computers and Digital Techniques, and a member of the editorial board of the Journal of Electronic Testing: Theory and Applications (JETTA), Journal of Embedded Computing, and Journal of Low Power Electronics. He was the General Chair of the 11th IEEE European Test Symposium (UK 2006), Technical-Programme Chair of DATE 2009, and the General Chair of DATE 2011. He is the coauthor of two Best Paper Awards: the James Beausang at the ITC 2000, relating to low power BIST for RTL data paths, and at the CODES-ISSS Symposium 2009, relating to low-energy fault-tolerance techniques. He is a co-author of a paper on test data compression which has been selected for a Springer book featuring the most influential work over the ten years of the DATE conference.

David Flynn a Fellow in R&D at ARM Ltd, has been with the company since 1991, specializing in System-on-Chip IP deployment and methodology. He is the original architect behind ARM’s synthesizable CPU family and the AMBA on-chip interconnect standard. His current research focus is low-power system-level design. He holds a number of patents in on-chip bus, low power and embedded processing sub-system design and holds a BSc in Computer Science from Hatfield Polytechnic, UK and a Doctorate in Electronic Engineering from Loughborough University, UK. He is currently Visiting Professor with the Electronics and Computer Science Faculty at Southampton University, UK.

Sachin Idgunji is a Principal Engineer in the Research and Development Group at ARM. He has been with ARM since 2006 and his current area of research is in power efficient multi-core system architecture and is presently working on projects that investigate the opportunities and challenges of 3D-ICs in multi-core systems on chip (SoC). He has filed and received patents in areas of low power, fault tolerant circuits and systems and has worked extensively on various aspects of low power design and how process variation at advanced nodes impacts SoC design. Sachin received his B.E. in Electronics in 1990 from Shivaji University in India and has over 20 years of experience. Prior to joining ARM he held technical lead roles at Synopsys, IBM, Control Net and PCS-Data General.