Fault-Tolerant Kogge-Stone Configuration Based Parallel-Prefix Adder Design for High Speed GPUs

[1]A.Avinash
M.Tech Student, Dept. of ECE
Vaagdevi college of engineering,bollikunta, India
avinasharmur@gmail.com

[2]Dr. G. Koteswara Rao
Assistant Professor, Dept. of ECE
Vaagdevi college of engineering,bollikunta, India
koteswarrao_g@vaagdevi.edu.in

Abstract

In the modern era of digital communication, reliable and efficient data transmission is critical across networking systems. However, data loss during transmission remains a significant challenge, particularly in high-speed router architectures. This paper presents the design and implementation of a FIFO-based router architecture aimed at avoiding data losses during transmission by integrating First-In-First-Out (FIFO) buffering with Finite State Machine (FSM) control logic. The FIFO buffer temporarily stores incoming data packets and forwards them in the exact order of arrival, thus minimizing overflow, underflow, and packet corruption. The FSM controller efficiently manages the read and write operations to ensure seamless data flow and prevent collisions or data loss. The proposed design is developed and tested using Verilog HDL on the Xilinx Vivado platform, with synthesis and FPGA implementation validating its hardware feasibility. Simulation and timing analysis demonstrate that the FIFO router effectively handles burst traffic conditions, provides stable data transmission, and achieves optimized resource utilization. This approach offers significant advantages for network-on-chip (NoC) architectures, IoT devices, and real-time embedded systems that require error-free, high-speed communication. Future extensions may include multi-port routing, priority scheduling, and integration with advanced protocols to further enhance robustness and performance.

Keywords — FIFO router, data loss prevention, finite state machine (FSM), hardware description language (HDL), Verilog, FPGA implementation, network-on-chip (NoC), high-speed data transmission, buffer overflow control, packet loss reduction.

I. Introduction

An adder is a fundamental digital logic circuit that performs the addition of numbers, forming an essential component in the arithmetic logic units (ALUs) of microprocessors and other digital computing systems. The most basic type is the ripple carry adder, which connects full adders in a series. While simple, its speed is limited because each adder must wait for the carry bit from the previous one. To overcome this, carry-lookahead adders (CLA) were developed, which compute carry signals in advance based on the inputs. This design serves as the foundation for a more advanced class of adders known as parallel prefix adders. Adders can be implemented as either serial or parallel circuits. A serial adder adds bits one at a time, making it simple but slow. In contrast, parallel adders, such as the ripple-carry adder and carry-lookahead adder, add all bits simultaneously for faster operation. However, the speed of a simple parallel adder like the ripple-carry adder is limited because each stage must wait for the carry from the previous stage. To address this delay, more advanced architectures called parallel prefix adders were developed. These adders are notable for their high speed, which is achieved by calculating the carry bits in parallel. This structure significantly reduces the time it takes to propagate carries across the entire

adder, overcoming the primary bottleneck of simpler designs. The concept of parallel prefix computation has led to various adder designs, including the Sklansky, Brent-Kung, and Kogge-Stone adders. Among these, the Kogge-Stone adder is widely recognized as one of the fastest and is a popular choice for high-performance arithmetic circuits in industries. It achieves its speed by computing generate and propagate signals in parallel stages, but this comes at the cost of significant area and wiring complexity. To balance the trade-off between speed, area, and power, the Sparse Kogge Stone (SKS) adder was introduced. The "sparsity" refers to how many carry bits are generated by the carry-tree structure. The SKS adder is a hybrid design that combines a parallel prefix carry tree for the most significant bits with slower but more area efficient ripple-carry adders for the less significant bits. This architecture reduces 1 computational complexity, power consumption, and area without a major impact on speed, making it an efficient solution for modern digital circuits. History Parallel prefix adders, such as the Kogge–Stone, Brent–Kung, Han–Carlson, and Sklansky adders, represent a class of high-performance adders optimized for minimal delay. They work by pre-computing generate (G) and propagate (P) signals for each bit and then combining them in a tree-like prefix structure. The strength of prefix adders lies in their ability to achieve logarithmic delay with respect to operand size, making them highly attractive for wide-word arithmetic in digital signal processors, GPUs, and modern CPUs. Among them, the Kogge–Stone Adder (KSA), introduced in 1973 by Peter M. Kogge and Harold S. Stone, is widely regarded as the fastest prefix adder due to its minimal logic depth and regular interconnection pattern. However, this performance advantage comes at the cost of higher wiring complexity and area overhead. The Sparse Kogge–Stone Adder (SKSA) was later introduced as an advancement over the conventional KSA to balance speed and hardware cost. Instead of computing carries for every bit through the prefix network, the SKSA computes carries only at fixed intervals (segments), while the intermediate bits are resolved using short ripple carry adders. This hybrid approach significantly reduces the wiring overhead and area while maintaining near-logarithmic delay. Consequently, sparse prefix structures offer a practical trade-off for VLSI and FPGA implementations where power, area, and routing congestion are critical design constraints. The evolution of adders has followed the progression from simple ripple-based structures to sophisticated hybrid prefix-based designs, each step driven by the need for improved performance and fault tolerance. In modern applications such as error resilient computing, cryptography, and fault-tolerant architectures, the ability of adders to withstand soft errors and transient faults is as important as their speed. The Sparse Kogge–Stone Adder, with its structured tree and partial ripple correction, provides an efficient foundation for implementing reliable arithmetic units capable of both high performance and fault resilience. Recent Advancement in This Area 2 Recent advancements in Sparse Kogge–Stone Adders (SKSA) have increasingly emphasized fault tolerance to ensure reliable operation in safety-critical and high-performance systems. Early efforts primarily addressed performance–area trade-offs, but modern designs now integrate mechanisms for error detection,

fault localization, and correction. Techniques such as triple modular redundancy (TMR), duplication with comparison, and voter-based correction have been adapted into SKSA structures to safeguard against transient faults and soft errors. In addition, mismatch based detection methods have been proposed, where discrepancies between tree derived and ripple-derived carries are used to identify faulty sections. More advanced approaches incorporate half-cycle or pipelined correction strategies, which reduce recovery latency and allow error-free outputs to be produced within a single clock cycle. Recent research has also explored hybrid SKSA architectures that combine fault tolerance with low-power and approximate computing techniques, enabling resilience while maintaining energy efficiency. Together, these advancements have transformed SKSA from a high-speed adder into a robust arithmetic unit capable of sustaining reliable operation under fault-prone conditions, making it highly suitable for FPGA, ASIC, and mission-critical computing platforms.

II. Literature Review

The design of efficient router architectures capable of minimizing data loss during high-speed communication has been an active research area. Routers operating in contemporary digital systems require effective buffering mechanisms to handle burst traffic and prevent packet loss caused by congestion, buffer overflow, or underrun.

FIFO (First-In-First-Out) buffers have been widely adopted in network-on-chip (NoC) routers and embedded systems for temporary data storage owing to their simplicity and ability to maintain packet order integrity. Previous research has investigated various FIFO architectures ranging from synchronous and asynchronous FIFOs to parameterizable and modular designs that offer scalability and reconfigurability.

Finite State Machines (FSMs) have been leveraged to control the read/write operations in FIFOs systematically, thus preventing overflow and underflow conditions, which are critical to maintaining reliable data transmission. Advances in FSM-based control circuits have enhanced data handling efficiency and hardware resource optimization.

Several works have explored enhancements to FIFO designs to address power consumption, fault tolerance, and operational reliability. Techniques such as power gating, dynamic voltage and frequency scaling, and online fault detection mechanisms have been proposed to improve FIFO energy efficiency and robustness under fault conditions.

Despite the progress, existing router buffering methods including basic register-based buffers, circular queues, and priority-based scheduling often face challenges related to complexity, resource overhead, and incomplete elimination of packet loss under heavy traffic. The integration of FIFO buffering with FSM control logic in a hardware-friendly design is therefore pivotal to achieving optimal router performance in NoC architectures and real-time embedded environments.

This study builds on the theoretical foundations of FIFO buffering and FSM control, and further validates the practical application via Verilog HDL implementation and FPGA deployment,

focusing on overcoming limitations in conventional router designs.

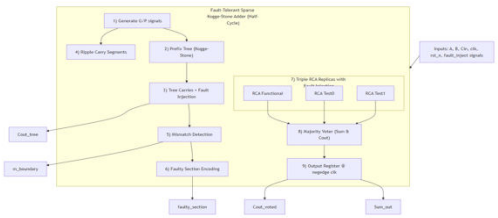
III. Methodology

The proposed Fault-Tolerant Sparse Kogge-Stone Adder (FT-SKSA) architecture is engineered to deliver high-speed arithmetic operations while ensuring robust resilience against transient and permanent hardware faults. The methodology integrates a high-performance parallel-prefix core with redundant computational units and a systematic error-handling mechanism. The design and implementation were executed through the following key stages.

A. Architectural Overview

The core of the FT-SKSA is a hybrid design that merges the speed of a Sparse Kogge-Stone (SKS) parallel-prefix tree with the reliability of Triple Modular Redundancy (TMR). The n-bit operand (implemented as 16-bit in this work) is divided into k segments. The SKS tree computes carry signals only at these segment boundaries, while the internal bits of each segment are processed by simpler Ripple-Carry Adders (RCAs). This sparsity reduces area and wiring complexity compared to a full Kogge-Stone structure [1], [12].

The fault tolerance is achieved through a multi-layered approach, as illustrated in Fig. 1:



Fault Injection and Detection: A mismatch mechanism compares the fast SKS tree carries with a reliable RCA reference.

Fault Masking and Correction: TMR with majority voting ensures correct output even if one computational unit fails.

Output Stabilization: A half-cycle latching strategy ensures glitch-free, stable outputs.

B. Sparse Kogge-Stone Prefix Tree

The Kogge-Stone tree is employed for its minimal logic depth and high-speed parallel carry computation [11]. For each bit position i, the generate (Gi) and propagate (Pi) signals are computed as:

$$Gi=Ai \cdot Bi$$

$$Pi=Ai \oplus Bi$$

These signals are then processed through $\log_2(n)$ stages of the prefix network. At each stages, the group generate and propagate signals are computed over an interval of 2^s bits using the prefix operator:

$$(G,P)_s[i] = (G,P)_{s-1}[i] \bullet (G,P)_{s-1}[i-2^{s-1}]$$

$$\text{where } (G,P)_a \bullet (G,P)_b = (G_a + P_a \cdot G_b,\ P_a \cdot P_b)$$

The "sparse" aspect is implemented by terminating the prefix tree at segment boundaries (e.g., every 4 bits for a 16-bit adder). The final tree carry at a boundary j is:

$$C_{tree\_raw}[j] = G_{final}[j] + (P_{final}[j] \cdot C_{in})$$

C. Fault Injection and Mismatch Detection

To validate the fault tolerance, a controlled fault injection mechanism is integrated. The raw tree carry at each segment boundary (except the MSB) is XORed with a fault mask bit to simulate stuck-at or bit-flip errors:

$$C_{tree}[j] = C_{tree\_raw}[j] \oplus tree\_fault\_mask[j]$$

A dedicated, fault-free Ripple-Carry Adder (RCA) operates in parallel as a golden reference. The carry outputs from this reference RCA (*Crip*) are compared with the potentially fault-injected tree carries (*Ctree*) at each segment boundary. A mismatch signal is generated as:

$$mismatch[j] = C_{tree}[j+1] \oplus C_{rip}[j+1]$$

A high value on any *mismatch*[*j*] indicates a fault within the corresponding segment of the prefix tree, enabling precise fault localization.

D. Triple Modular Redundancy (TMR) and Majority Voting

The core fault correction is achieved through TMR. Three independent instances of the main adder—comprising the SKS tree and segmented RCAs—are executed in parallel:

- RCA_FUNC: The primary functional unit, susceptible to injected faults.
- RCA_T0 and RCA_T1: Two redundant replica units, which remain fault-free in our model.

The Sum (*S*) and Carry-out (*Cout*) from these three units are fed into a majority voter. The voter logic for each output bit *i* is given by the Boolean equation:

$$S_{voted}[i] = (S_{func}[i] \cdot S_{t0}[i]) + (S_{func}[i] \cdot S_{t1}[i]) + (S_{t0}[i] \cdot S_{t1}[i])$$

This ensures that the final output matches the value produced by at least two of the three units, effectively masking a single point of failure.

E. Half-Cycle Output Latching

To mitigate the impact of transient glitches and ensure temporal stability, the voted outputs are registered on the falling edge of the system clock. This half-cycle correction strategy provides a clean timing window for the voting logic to resolve, delivering stable, fault-corrected results without introducing a full clock cycle of latency. The final outputs are thus:

$$Sum_{out} \leftarrow S_{voted} \quad \text{on negative clock edge}$$

$$Cout_{out} \leftarrow Cout_{voted} \quad \text{on negative clock edge}$$

This approach shortens the error recovery path compared to full-cycle pipelining, enhancing system responsiveness.

F. Design Implementation and Verification Flow

The complete 16-bit FT-SKSA was designed and captured using the Verilog HDL. The functional description was synthesized and implemented targeting a Xilinx FPGA using the Xilinx Vivado toolchain. The verification process involved:

Functional Simulation: To verify correct arithmetic operation in a fault-free scenario.

Fault Injection Simulation: To validate the detection and correction capabilities by activating the tree_fault_mask signals.

Post-Synthesis Analysis: To evaluate timing performance (setup/hold times), power consumption, and resource utilization (LUTs, I/O ports).

IV. Implementation Setup

The proposed FT-SKSA architecture was realized through a structured digital design flow, from HDL modeling to post-synthesis verification. The following steps detail the implementation process.

A. Hardware Description using Verilog HDL

The design was captured at the Register Transfer Level (RTL) using Verilog. The system was partitioned into hierarchical modules to ensure modularity, readability, and ease of debugging.

Top-Level Module (ft_ks_adder_16bit): This module instantiates all sub-modules and defines the primary I/O ports, including:

Inputs: A[15:0], B[15:0], Cin, clk, rst_n, tree_fault_mask[3:0].

Outputs: Sum_out[15:0], Cout_out.

Generate & Propagate (GP) Module: This combinational module computes the initial generate (G_i = A_i & B_i) and propagate (P_i = A_i ^ B_i) signals for all 16 bit positions.

Sparse Kogge-Stone Prefix Tree Module: This module implements the multi-stage prefix tree to compute carry signals at segment boundaries (e.g., bits 4, 8, and 12). It was designed using for generate loops to create the recursive prefix structure, ensuring scalability.

Segmented Ripple-Carry Adder (RCA) Module: A single 16-bit RCA module was designed and instantiated three times to create the TMR structure (RCA_FUNC, RCA_T0, RCA_T1). The RCA_FUNC module incorporates the fault injection logic, where the tree-derived carry into a segment is XORed with the corresponding bit from the tree_fault_mask register.

Majority Voter Module: A purely combinational module that takes the 16-bit sum and 1-bit carry-out from each of the three RCAs. It implements the majority function V = (A&B) | (A&C) | (B&C) for each bit of the sum and the final carry-out.

Output Register Module: A negative-edge-triggered register module that latches the outputs of the majority voter, providing the final, stable Sum_out and Cout_out.

B. Functional Verification via Simulation

The RTL design was rigorously verified using a testbench to simulate various operational scenarios.

Fault-Free Operation: The adder was tested with a comprehensive set of input vectors, including corner cases (e.g., A=0xFFFF, B=0x0001) to ensure correct arithmetic functionality in the absence of faults.

Fault Injection and Correction Test: The fault tolerance was validated by systematically asserting bits in the tree_fault_mask register. For example:

Test Case: A = 0x000A, B = 0x0005, tree_fault_mask = 4'b0010 (injecting a fault at the boundary of segment 1).

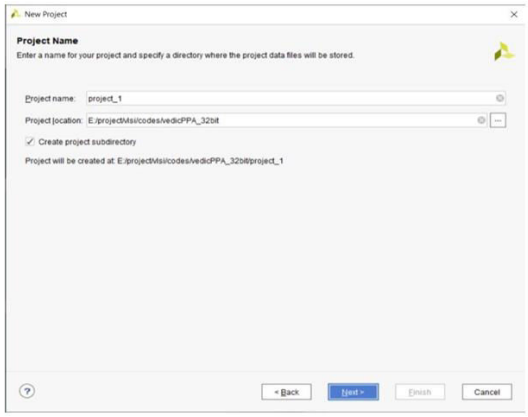
Expected Behavior: RCA_FUNC produces an erroneous result due to the injected fault, while RCA_T0 and RCA_T1 produce the correct sum (0x000F). The majority voter corrects the error, and the final output remains 0x000F.

C. Synthesis and Physical Design Setup

The verified Verilog code was synthesized using Xilinx Vivado, targeting a specific FPGA family (e.g., Virtex-7).

Project Creation and Configuration: A new project was created, and the target device (xc7vx485tffg1157-1) was selected, as shown in Fig. 2.

.



Synthesis Process: The Vivado synthesis engine (XST) was run to translate the RTL design into a gate-level netlist optimized for the target FPGA. Key synthesis reports were analyzed:

Utilization Report: Confirmed the number of Look-Up Tables (LUTs), flip-flops, and I/O ports used.

Timing Report: Provided data on the critical path delay and ensured setup and hold times were met.

D. Post-Synthesis Verification

After successful synthesis, the gate-level netlist was simulated to verify that the design's timing-accurate behavior matched the pre-synthesis RTL simulation. This step ensured that the synthesis process did not introduce any functional errors.

E. Performance Analysis

The post-synthesis design was analyzed to evaluate its key performance metrics:

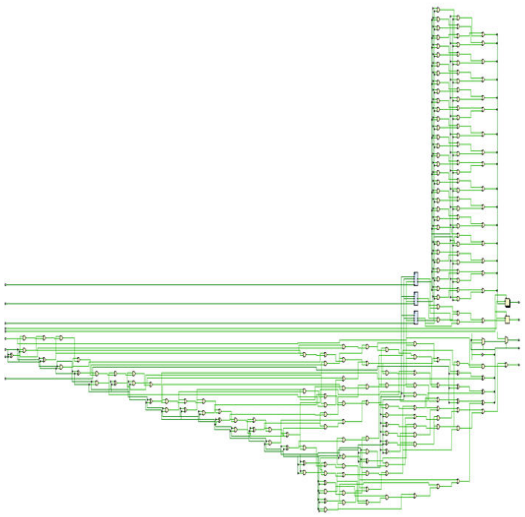
Timing Performance: The critical path delay was extracted from the Static Timing Analysis (STA) report to determine the maximum operating frequency.

Area Overhead: The resource utilization report was used to quantify the hardware cost of the TMR redundancy and fault detection logic.

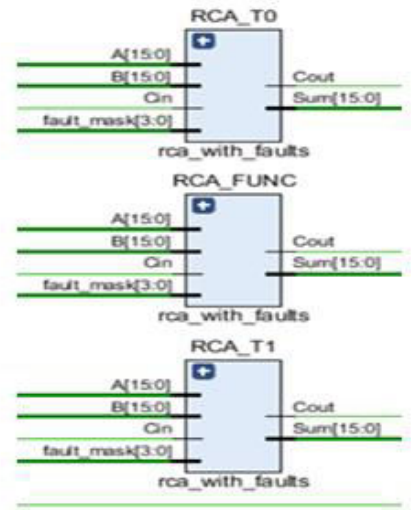
Power Estimation: The Vivado power analysis tool was used to estimate the dynamic and static power consumption of the implemented design.

This structured implementation flow, from RTL design to post-synthesis analysis, ensured a robust and validated fault-tolerant adder suitable for high-reliability applications.

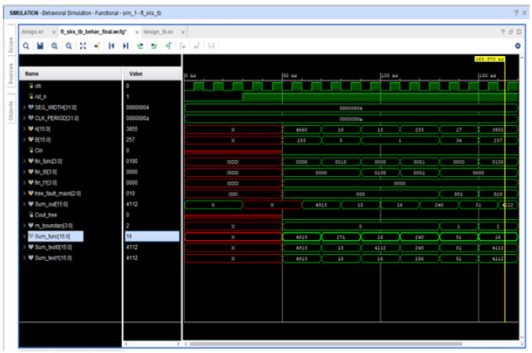
Results



Schematic of the proposed fault tolerant SKA.



Functions schematic of proposed fault injecting modules



Simulated design for the proposed system

v. Future Scope

The presented Fault-Tolerant Sparse Kogge-Stone Adder with Half-Cycle Correction demonstrates strong resilience against injected faults and ensures reliable performance in critical applications. However, there are several directions for extending this work. One promising avenue is to scale the design for higher bit-width adders (32-bit, 64-bit, or beyond) to evaluate its efficiency and robustness for modern high-performance processors. Additionally, integrating the architecture into pipelined arithmetic units could further improve throughput, enabling deployment in real-time signal processing, cryptographic accelerators, and error-sensitive communication. Another potential enhancement is the exploration of adaptive fault tolerance, where the system dynamically adjusts the level of redundancy (e.g., dual vs. triple modular redundancy) depending on error rate, environmental conditions, or power constraints.

vi. Conclusion

The proposed design successfully integrates the high-speed characteristics of a Sparse Kogge-Stone prefix adder with the robustness of fault-tolerant mechanisms, ensuring reliable operation even in the presence of hardware faults. By combining the parallel carry computation of the KS tree with the redundancy of multiple Ripple-Carry Adders and Triple Modular Redundancy (TMR) voting, the system achieves both performance and correctness. Mismatch detection between tree-generated and ripple generated carries further enhances fault localization, while fault injection experiments validate the design's ability to withstand single-segment errors and deliver correct outputs through majority voting. Simulation results confirm that the architecture consistently recovers the correct sum under various injected fault scenarios, with only multi-replica errors leading to incorrect results, as expected in a TMR framework. The half-cycle latching mechanism ensures that corrected outputs are registered with minimal latency, making the design highly suitable for safety-critical and error-prone environments such as space electronics, cryptographic accelerators, and dependable communication hardware.

Overall, the implementation provides a balanced trade-off between speed, reliability, and hardware efficiency, demonstrating the effectiveness of combining prefix adders with TMR-based fault correction.

References

[1] Hoe, D. H., Martinez, C., & Vundavalli, S. J. (2011). Design and Characterization of Parallel Prefix Adders using FPGAs. IEEE 43rd Southeastern Symposium on System Theory.

[2] Kondalkar, M. B., Chavan, A. P., & Narashimaraja, P. (2013). Improved Fault Tolerant Sparse KOGGE Stone ADDER. International Journal of Computer Applications.

[3] Pritty, A. (2024). Optimized Fault-Tolerant Adder Design Using Error Analysis. Journal of Circuits, Systems and Computers.

[4] Martinez, C. D., Bollepalli, L. P. D., & Hoe, D. H. K. (2012). A Fault Tolerant Parallel-Prefix Adder for VLSI and FPGA Design. 44th IEEE Southeastern Symposium on System Theory.

[5] Ghosh, S., Ndai, P., & Roy, K. (2008). A Novel Low Overhead Fault Tolerant Kogge-Stone Adder using Adaptive Clocking. Design, Automation and Test in Europe.

[6] Kondalkar, M. B., Chavan, A. P., & Narashimaraja, P. (2013). Improved Fault Tolerant Sparse KOGGE Stone ADDER. International Journal of Computer Applications, 75(10), 36-41.

[7] Di Carlo, S., Miele, A., Prinetto, P., & Trapanese, A. (2006). Microprocessor fault-tolerance via on-the-fly partial reconfiguration. MICROELECTRONICS RELIABILITY.

[8] Banerjee, N., Augustine, C., & Roy, K. (2008). Fault-Tolerance with Graceful Degradation in Quality: A Design Methodology and its Application to Digital Signal Processing Systems. IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems.

[9] Martinez, C. D., Bollepalli, L. P. D., & Hoe, D. H. K. (2012). A fault tolerant parallel-prefix adder for VLSI and FPGA design. ResearchGate.

[10] Hoe, D. H. K., Martinez, C., & Vundavalli, S. J. (2011). Design and Characterization of Parallel Prefix Adders using FPGAs. IEEE 43rd Southeastern Symposium on System Theory.

[11] R. N. Nawkhare. (2015). Improved Fault Tolerant Sparse Kogge Stone Adder. International Journal of Advanced Research, 3(12), 876-883.

[12] Vundavalli, J. (2010). Design and Analysis of wide bit adders for FPGA Implementation. MSEE Thesis, University of Texas at Tyler.

[13] Stone, P. M., & Stone, H. S. (1973). A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence equations. IEEE Transactions on Computers.

[14] Lynch, T., & Swartzlander, E. E. (1992). A Spanning Tree Carry Lookahead Adder. IEEE Transactions on Computers.

[15] Talib, M., & El-Maleh, M. (2017). Design of Fault Tolerant Adders: A Review. IEEE International Conference on Smart Systems.