Infrastructure for Testing Nodes of a Wireless Sensor Network

Bojan Mihajlović[1], Željko Žilić[1], and Katarzyna Radecka
McGill University, Montreal, Canada[1]

ABSTRACT

Maintaining reliable networks of low cost, low energy wireless sensor network (WSN) nodes is a major concern. One way to maintain a reliable network is to perform in-field testing on nodes throughout their lifetimes, identifying failing nodes so that they can be repaired or replaced. This chapter explores the requirements for a wireless test access mechanism, and introduces a method for remote execution of software-based self-test (SBST) programs. In an effort to minimize overall test energy consumption, an SBST method is derived that takes the least amount of microcontroller cycles, and is compatible with system-level optimizations such as concurrent test execution. To further reduce test energy, compression algorithms compatible with WSN nodes are explored for use on test programs. The efficacy of all proposed methods is evaluated experimentally, using current measurement circuitry applied to a WSN node.

INTRODUCTION

Wireless sensor networks (WSN) have become available for use in various industrial control, environmental monitoring, and military applications. Typical WSN applications require that the network be reliable and maintainable in order to be useful. Such constraints necessitate a unique approach to the design and testing of WSN nodes. It may seem counter-intuitive that reliable networks can be built with what are often inexpensive nodes that are individually unreliable. In fact, one way to maintain a reliable network is to test nodes throughout their lifetimes in order to identify failing nodes so that they can be repaired or replaced. This is especially vital when nodes are deployed in inhospitable environments that accelerate their failure rate. This work is aimed at addressing the aggregation of quality issues in the operation of WSNs: correct operation, reliability, availability, and operation under strict energy constraints.

While much work has been dedicated to the manufacturer testing of embedded systems such as WSN nodes, this chapter addresses the in-field testing of nodes in a deployed WSN. There are several obstacles in realizing this type of testing scenario, starting with the lack of a testing infrastructure. Such as infrastructure needs to define how testing is carried out at a network-level and how test programs and test responses are stored and communicated. There is also a need for test programs that achieve an adequate test quality, or fault coverage, of individual node components and of the node as a whole. All the while, nodes must operate while using as little energy as possible. This poses significant challenges in maintaining wireless signal quality, and consequently, reliable network operation. It means that the overhead of performing testing must be kept to a minimum, so as not to severely impact the operational life of the node. These obstacles are addressed in this chapter, and the approach used is briefly introduced in the following subsections.

Testing Infrastructure

To achieve a reliable WSN, one requires an infrastructure for performing remote node tests. Until recently, the most efficient and often only in-field self-testing involved using built-in self-test (BIST) hardware within wireless nodes. Since this is often not possible due to the performance, area, and energy overheads (Krstic, Lai, Cheng, Chen, & Dey, 2002), a promising type of software-based testing is introduced as an effective alternative. These software-based self-test (SBST) programs work by using an existing microcontroller unit (MCU) instruction set to perform self-testing of all digital and mixed-signal components on a WSN node. The SBST programs allow an equivalent test quality to be achieved while minimizing energy consumption.

An infrastructure is presented that allows a basestation to distribute SBST programs to nodes in the network, remotely execute the SBST programs, and return test results back to the basestation. A method is also presented that harnesses regular network nodes as *helper nodes* in characterizing the wireless links that they can establish with their neighbours. The resulting testing infrastructure allows detection of failed node and even predication of failing nodes to be achieved.

To ensure that nodes are properly tested, any test that is employed must meet an acceptable coverage of modeled faults (fault coverage). The SBST programs that are used must then be built to test the entire node, and must cover every testable component on-board the node. Testable components include the MCU, memory, RF module, and sensors, as these are the primary components of any WSN node. A known-good test result for each test must also be calculated and stored separately. The result can then be compared with actual results remitted by nodes.

Energy Efficiency

Since nodes are power-sensitive devices whose power sources are often on-board batteries, network quality can suffer if some or all nodes exhaust their energy reserves prematurely. Any overhead energy consumption must be minimized, such as the running of self-test programs. To do this, several energy-saving techniques are introduced which can reduce test energy consumption and test time:

- Test optimization – Test time is decreased by selecting the most efficient set of instructions to achieve the same test quality.
- Test combination – There is an inherent overlap in testing separate systems on the same node. The coverage of each test is analyzed and redundancy eliminated.
- Test concurrency – By reordering and rescheduling tests, test energy and test time can be reduced.
- Test program compression – Compressing test programs reduces communication and, in turn, the energy required to perform testing.

By taking this approach, we address WSN quality issues that are currently impediments to correctly operating, reliable, available, and energy-efficient networks.

BACKGROUND

The introduction of ultra-low power network protocols such as IEEE 802.15.4 and its overlay protocol, Zigbee, has allowed wider use of a new class of devices. The new protocols enable devices to save energy otherwise expended in frequent communication, and are well-suited to wireless sensor networks (WSNs). A typical WSN is composed of sensor nodes and a basestation. The nodes' main function is to collect data through environmental sensors, as well as to control actuators to physical processes. At minimum, each sensor node contains an embedded MCU, wireless transceiver, antenna, environmental sensors/actuators, and a battery, as seen in Figure 1. To minimize cost, components can be COTS parts assembled on a printed circuit board (PCB) containing a printed dipole antenna. The MCU is used to collect and process sensor data, which is stored on-board its embedded memory. The MCU also performs wireless network communication through the attached RF module, in infrequent and low-power packet transmissions.

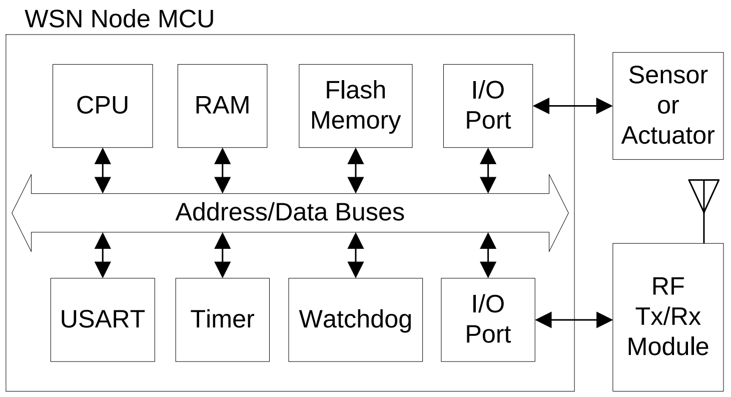

Figure 1. Generic Node Architecture

Data harvested from the environment is relayed by nodes to the basestation through the network. The basestation serves as a focal point of the network, where sensor readings are aggregated and actuator control is focused. A sensor node itself may be required to refine data before announcing its results, or communicate with neighbouring sensor nodes. The following are general network-level requirements that a robust WSN should meet:

- Availability – The network should be online and available to the applications running on-board both the basestation and nodes.
- Scalability – Networks with thousands of nodes may become commonplace in environmental monitoring or military applications. The system must scale to accommodate such uses.
- Self-Organization – Harsh operational environments leads to a high rate of node failures. A robust network must be able to circumvent failed nodes and organize to a new functional state.
- Timing – Real-time operation must be available to support applications where it is necessary.

- Data Aggregation – Sending data from individual nodes in large networks to a single basestation may overwhelm available network bandwidth. The ability of some nodes to poll data from a cluster of their peers can alleviate this pressure.

Node-specific requirements include the following, but are not limited in scope to a node, as failure to meet them can have a network-level effect:

- Low Power Consumption – Nodes are often battery operated and must remain functional for months or years. Since manual replacement of batteries is often not possible, minimizing power consumption is critical in achieving a robust network.
- Low Cost – Since wireless sensor networks are potential replacements for present wired sensor networks, they must remain low-cost in order to make them a viable alternative.

WSN Availability

To ensure that a WSN can operate for a long time, a reliability assessment must be performed on the failure rates of its individual nodes. Even if redundancy techniques or highly reliable hardware are utilized, nodes can be deployed in harsh or even hostile environments that accelerate their failure. Of particular concern are failures that can sever the network and render a portion inaccessible. These failures affect network availability, which can be defined as the probability that the entire network is available for use. The primary challenge is to maintain a working application in the face of node failures, while detecting and even predicting the failure of sensor nodes and their components. The probability that a network will be restored to full operation in a given amount of time after a failure has occurred is defined as network serviceability (Chiang, Zilic, Radecka, & Chenard, 2004) or maintainability (Lala, 2001). In reality the calculation of such a metric is a complex task (Callaway, 2004), although some models exist, such as the Exponential Failure Law (EFL) defined in (Lala, 2001).

WSNs are subject to time-varying component connectivities due to factors impacting their wireless links (Chiang et al., 2004). Phenomena such as multipath fading, the “hidden terminal” problem (Callaway, 2004), and electromagnetic interference (EMI) contribute to the complexity of calculating an important metric such as availability. For the purpose of this paper, the concern is on overall system-level availability of a WSN application. So *perceived availability* can be defined as the probability that a WSN application is functioning correctly over a period of time (Chiang et al., 2004). Since WSNs are by nature distributed systems, the definition of reliability can be divided into component-level (local) and process-level (global) (Hariri & Huitlu, 1995; Raghavendra, Kumar, & Hariri, 1988). While component-level reliability deals with the reliability of individual elements which comprise the distributed system, such as nodes, process-level reliability includes all processes, hardware components, and communication channels which constitute the WSN.

Component-level failures can affect process-level reliability, which in-turn affects perceived availability. For example, a component-level failure of a network gateway node can cause the node to fail, and the network to split, which will affect perceived network availability. If node failures could be detected in such a scenario, then network down-time can be minimized. Periodic node testing is one method to ensure that nodes are continually operating correctly, while a survey of other methods can be seen in (Koushanfar, Potkonjak, & Sangiovanni-Vincentelli, 2004). Since nodes can be repaired or replaced after they are found to be defective, testing can play an essential role in maintaining a high degree of network availability. The following section will explore various system-level testing options available for use on WSN nodes.

System-level Testing

Since WSN nodes are made to be deployed in inhospitable or even hostile environments, it is often impossible to obtain physical access to the devices in order to perform functional testing on them. Accelerated failures can also be attributed to the inhospitable environments in which WSNs might be used. Environmental conditions can compromise the reliability of node operation and by extension, the availability of the WSN itself. To achieve a high reliability and availability in a sensor network, node failures must be detected and/or averted during network operation. Thus, the in-field testing of individual nodes throughout their lifetimes is essential in predicting and detecting node failures and, in turn, improving network availability. Higher-priced systems, such as those in military applications, achieve enhanced reliability by using redundancy and highly reliable parts. However, such configurations have been shown to be unsuitable to power sensitive devices (Hariri & Huitlu, 1995). The cost and power consumption of such methods can quickly become prohibitively large on WSN nodes. It is clear that an alternative way of achieving the needed reliability and availability is needed that allows devices to retain a small power footprint and cost.

There are several methods of functional testing which can be used for system-level testing of nodes, including:

- Boundary Scan
- Hardware Built-in Self-Test (BIST)
- Software-based Self-Test (SBST)

Boundary scan methods require dedicated pins and larger, modified flip-flops to be physically incorporated onto a chip. Devices known as automatic testing equipment (ATE) are traditionally interfaced physically with the device-under-test (DUT). The ATE executes automatic test-pattern generation (ATPG) algorithms which create test vectors to stimulate potential faults based upon some fault model. The test vectors are shifted into the chip through the dedicated pins, the tests executed, and test responses shifted out. To overcome the need for physical connectivity, some research has been done in creating overlays which allow remote testing capability, as in the boundary scan that wirelessly interacts with the DUT in (Chiang et al., 2004). The process of shifting test vectors and responses can also consume a disproportionate amount of power for the testing of a sensor node. Using BIST resolves this issue by using dedicated hardware on-board the DUT to generate test vectors to output a single resulting test signature. However, hardware BIST is often not possible due to the performance, area, and energy overheads of dedicated test circuitry (Krstic et al., 2002). Since BIST is most often not included on sensor node components, creating a custom chip would also be relatively expensive when there are cheaper common off-the-shelf (COTS) parts available.

Recent work in software-based self-test (SBST) programs (Zhang, Zilic, & Radecka, 2006; Kranitis, Paschalis, Gizopoulos, & Xenoulis, 2005; Paschalis & Gizopoulos, 2005) offers a way to achieve high quality testing with a small performance overhead and no area penalty. The SBST programs utilize an existing MCU's instruction set to perform a self-test of all digital and mixed-signal components on a WSN node. Various test sequences are run through the system bus, peripherals, transceiver, and MCU itself in order to uncover faults based on some fault model. A summary of the combined test results, also known as a *test signature*, can then be generated. The signature can either be directly passed onto a basestation, or compared to a known-good result stored within the SBST program itself. The recent work in SBST can be attributed to its numerous advantages in certain systems over traditional methods, including:

- Testing while chip is running at full functional speed (at-speed testing) (Chen & Dey, 2001)
- Generation of deterministic test vectors from SBST program code
- Unique signature responses contained within SBST program code and compared by MCU (Chen & Dey, 2001)
- No need for expensive automatic test equipment (ATE)
- In-field testability
- Energy efficiency

This self-testing approach is considered an inexpensive way of achieving reliability, availability, and serviceability in a WSN. There are other compelling reasons for the use of SBST, such as certain tests that can determine if the hardware or software of a node has been tampered with by an intruder. The same interface is compatible with providing manufacturer testing, where tests are broadcast to multiple nodes simultaneously. This reduces the need for individual node testing using expensive automated testing equipment (ATE) or alternative on-board testing interfaces. Recent work in developing SBST programs for embedded processors includes the work of (Kranitis et al., 2005), which was used as the basis for an overall WSN node testing methodology proposed in (Zhang, Zilic, & Radecka, 2006; Zhang, 2005).

The SBST approaches used thus far on microprocessors can be classified into two categories. The first includes (Shen & Abraham, 1998; Batcher & Papachristou, 1999) that have a high level of abstraction and are functionally oriented. The second category includes (Kranitis et al., 2005, Chen & Dey, 2001; Kranitis, Paschalis, Gizopoulos, & Zorian, 2002), which are structurally oriented and require structural fault-driven test development. From the first category, (Shen & Abraham, 1998) requires a lengthy test set, whereas the approach of (Batcher & Papachristou, 1999) is not purely a software-based method and requires some extra hardware. In the second category, a gate level netlist is necessary in (Chen & Dey, 2001), which is difficult information to obtain for COTS chips because of intellectual property (IP) constraints. The SBST methodology in (Kranitis et al., 2002) is based upon the application of deterministic test patterns targeting structural faults of individual processor components. The work of (Kranitis et al., 2005) builds upon (Kranitis et al., 2002) by defining test priorities for different processor components. The SBST proposed in (Kranitis et al., 2005) is also used as the basis for the methodology proposed in (Zhang, Zilic, & Radecka, 2006; Zhang, 2005), and shares desirable characteristics with functional testing, like high-level test development using the instruction set. The work takes a lower-level approach in its use of RTL information and uses a divide-and-conquer method to target individual components with the stuck-at fault model. This allows a high fault coverage of more than 95%, and is thus used as a model in the development of the SBST programs seen here.

In the early 1980's, an s-graph model at the register transfer level (RTL) was proposed (Thatte & Abraham, 1980; Brahme & Abraham, 1984) to represent a microprocessor, and used functional-level fault models for instruction-level test generation. Many further graph-based functional testing methods were later proposed, such as (van de Goor & Verhallen, 1992; Joshi & Hosseini, 1998), but suffered from the need for large amounts of manual effort in order to produce a relatively low fault coverage. The application of such test sets is also a lengthy process on microprocessors with large numbers of registers and instructions, since most of these methods relied on external ATE to deliver input test patterns and compare test responses.

Reduction of Test Data Volume

As the speed of modern chips has increased, the speed of the testing interfaces to these devices has not followed suit. This, in addition to increasing design complexities have contributed to longer test times of devices seen as of late, as noted by the Semiconductor Industry Association (2005). Compression of test data aims to reduce the volume of information transferred between a DUT and its tester. This is often done to reduce the overall time required to test a device, since the majority of test time is spent in the sending of test data and the receiving of a response. As a summary, the work thus far performed in test data compression has largely fallen into the following groups (Khoche, Volkerink, Rivoir, & Mitra, 2002):

1. Compression of fully-specified test vectors
2. Compression of incompletely-specified test vectors
3. Hardware Built-In Self-Test (BIST)
4. Compression/compaction of test responses

Methods 1, 2, and 4 require ATE to be connected to a DUT, where a remote interface could serve for in-field testing. For each test executed, a compressed test vector known as a *test cube* is generated by the ATE and transferred to the node where it is decompressed on-the-fly. The test is then executed and a response sent back to the ATE. In method 1, the test cube is a directly compressed test vector generated by the ATPG algorithm, while method 2 can employ many different schemes to further compress test vectors. In this method, the fact that most test vectors effectively contain many don't-care (X) bits is used to allow even greater compression. A type of *lossy compression* (Salomon, 2004) is applied, which is effective when only a few select bits of a test vector need to be exactly reproduced on the DUT. In Method 4, test response vectors can similarly be reduced in size. Compression can be performed on-board the DUT by the same algorithm as Method 1, or test responses can be compacted to signatures. These signatures are generated in much the same way as a simple hash, often by linear-feedback shift-register (LFSR) hardware.

All test communication for the aforementioned methods is done through a boundary scan interface. However, it has already been shown that the secret keys of cryptographic chips can be compromised using boundary scan attacks, and it is expected that boundary scan chains will become increasingly inaccessible on future production chips (Hely, Bancel, Flottes, & Rouzeyre, 2006). The test methods are also limited by the speed of the boundary scan chain, which can be significantly slower than a typical integrated circuit's operational frequency while incurring an area overhead. The exception is BIST, which can run at-speed and is improved through deterministic BIST methods (Liang, Hellebrand, & Wunderlicht, 2001), but is expensive in terms of physical device area and disallows the use of many COTS parts.

TESTING INFRASTRUCTURE

The testing of nodes is accomplished through the use of SBST programs within a testing infrastructure. Such an infrastructure for distributing and executing tests was found to be lacking from the literature. As part of the test protocol, SBST programs are dynamically loaded as needed onto nodes by the basestation, whose job it is to also collect back test responses signaling a pass or fail condition. A node-level view of the protocol functionality can be seen in Figure 2.

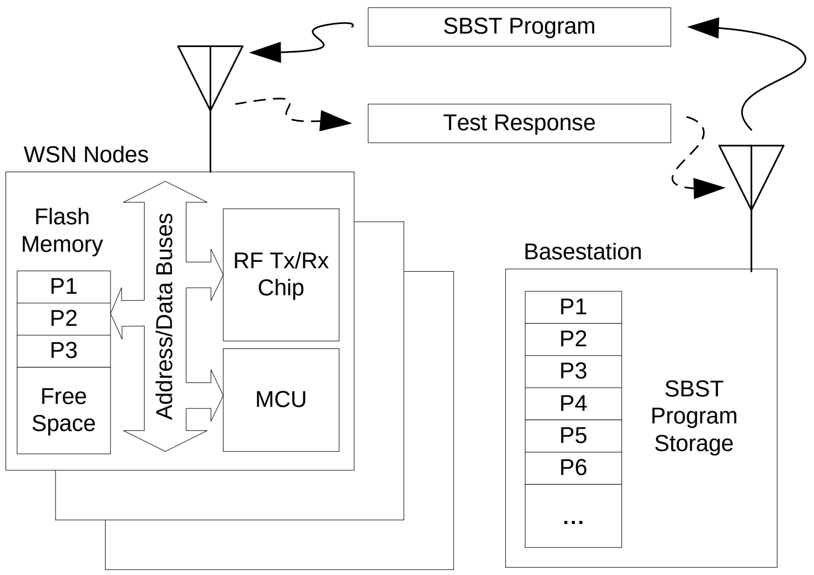

Figure 2. General Description of WSN Node Testing Protocol

The test protocol also defines network-level tests, such as the scenario shown in Figure 3, where shaded arrows represent test channels. It can be seen that the Node Under Test (NUT) performs most of its testing through self-test but is also evaluated by neighbouring nodes. Testing is initiated by the basestation, which transfers a compressed SBST program to the NUT. The on-board MCU decompresses the SBST program and stores it in embedded flash memory, enabling the basestation to remotely activate NUT self-testing. Once activated, the NUT executes the SBST program, which begins by executing MCU core self-testing, and continues to include embedded RAM, embedded flash memory, on-board data/address buses, and peripherals such as sensors. A resulting signature is calculated and sent back to the basestation, where it is compared to a node's known-good signature. If a signature is deemed correct, the RF module and antenna have also effectively been tested as operating correctly on a pass/fail level. For a more comprehensive evaluation of RF module and antenna performance, several nodes neighbouring the NUT can be used to track a decline in communication efficiency. An additional test program can be distributed to instruct the NUT and its neighbours to communicate at predetermined power levels. The results are sent back to the basestation, which tracks and analyzes changes over time to anticipate failing RF modules, and hence, failing nodes.

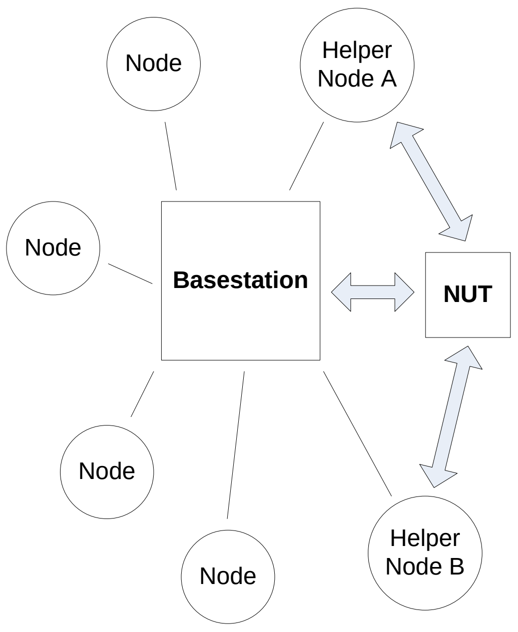

Figure 3. Node Testing Architecture

RF MODULE TEST SCHEME

Traditional test schemes that can be applied to WSN node RF transceivers work by looping transmitter output back into the receiver, and capture a test response in the baseband of the receiver (Dabrowski, 2003; Halder, Bhattacharya, Srinivasan, & Chatterjee, 2005; Ozev, Orailoglu, & Olgaard, 2002). A digital signal processor (DSP) is commonly used to analyze the test response at the receiver end. The advantage of this method is that receiver and transmitter subsystems are decoupled, but the testing method requires the addition of hardware to provide the loopback path.

An SBST method is introduced here to test RF modules, consisting of an RF transceiver and on-board printed antenna, with the aid of other nodes in the network. Several RF operating metrics are characterized and compared with some specifications of the IEEE 802.15.4 (2006) communication standard, parts of which are seen in Table 1. Instead of specifying tests for each component of an RF module, the module is considered in its entirety and functionally verified based upon published specifications.

Transmission Power and Receiver Sensitivity

To ascertain RF module transmission power and receiver sensitivity, a simple SBST test is introduced here. Test packets are sent in long bursts from the basestation to the NUT, where their reception is tallied. After completion, a test response is sent from the node to the basestation containing the packet error rate (PER) of the transmission. The receiver sensitivity of the node can be calculated with the Friis transmission equation (Stutzman & Thiele, 1997). Using the same method, the roles of basestation and node can be swapped to find NUT transmission power. The transmission power that achieves the required PER can be determined by varying the power in steps, as this is a software-programmable value on many RF transceivers.

Specification	Requirement	
Transmission Power	Minimum output power	-3 dBm
Receiver Sensitivity	Minimum input signal power yielding a packet error rate (PER) of < 1%	-85 dBm
Adjacent Channel Rejection	Min. ratio of the adjacent channel signal level to desired signal level with PER of < 1% (interference with adjacent channel)	0 dB
Alternate Channel Rejection	Min. ratio of the alternate channel signal level to desired signal level with PER of < 1% (interference with alternate channel)	30 dB

Table 1. Select RF Specifications of the IEEE 802.15.4 Protocol

Adjacent Channel and Alternate Channel Rejection

Many COTS RF transceivers use radio bands that many other devices are licensed to operate on. Since there is a high likelihood of interference between devices, IEEE 802.15.4 specifications require RF modules to reject the interference generated by other devices on neighbouring channels. A test is thus proposed for the ability of the RF module to perform this task. Here, *adjacent channel* refers to the channels closest in frequency to the channel being used, while *alternate channels* are those in-turn closest to the adjacent channels. For example, if channel 13 is being used, channels 12 and 14 are adjacent, while channels 11 and 15 are alternate.

An adjacent/alternate channel rejection test can be constructed as seen in Figure 4. In this scenario, the NUT is sent a long burst of packets from the basestation on a particular channel, while the helper node generates interference on either the adjacent or alternate channel. Using data from the receiver sensitivity test, transmission power on channel n_2 is set so that received signal strength on that channel is slightly higher than the minimum specification. This signal on channel n_2 is made purposely weak so that it will be sensitive to adjacent/alternate channel interference. Setting distances $d_1 = d_2$, the transmission power on channel n_1 is found which causes a PER of 1% in channel n_2. Again using the Friis equation, the adjacent/alternate channel rejection ratio can be found.

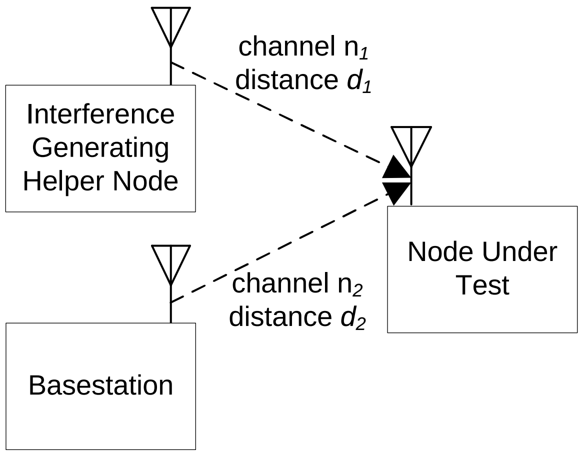

Figure 4. Test Framework of Adjacent Channel and Alternate Channel Rejection

ENERGY EFFICIENCY OF MCU TESTING

In this section, the dynamic programming idea of (Aho & Johnson, 1976) is used to construct the SBST programs proposed in (Kranitis et al., 2005) for testing an MCU. Since SBST programs are created using the MCU instruction set, a typical instruction uses one of many addressing modes and contains parameters such as a source register, destination register, and operand value. Energy optimizations can thus be performed based upon *program length* (number of cycles), *instruction type,* and *instruction parameters*.

Software energy consumption can be shown to be proportional to program length, but some published works including (Nikolaidis & Laopoulos, 2001; Chang, Kim, & Lee, 2000) indicate that there is a dependency between instruction energy consumption and the values of instruction parameters. These instruction parameters are collectively referred to as *energy-sensitive factors*. The list of energy optimization criteria are addressed in our discussion by the use of *instruction combination*, *instruction selection*, and *operand selection*, respectively.

We experimentally verify that an instruction's energy consumption is proportional to the Hamming distance between the previous and current values of its energy sensitive factors. The average current drawn by the MCU can be measured as it repeatedly executes certain instructions or a short instruction sequence with different configurations of energy sensitive factors. From this, it can be determined that executing the same instructions with different addressing modes has a differing energy cost. Such measurements offer the opportunity to employ instruction-level energy reduction methods like instruction selection/combination which uses the least number of MCU cycles, as well as operand selection with least Hamming distance and weight.

Instruction Selection and Combination

The goal of instruction selection is to reduce program length by replace existing instructions with ones that require the least amount of cycles, while maintaining equivalent fault coverage. Instruction combination involves finding overlapping fault coverage between separate component tests. By combining component tests into a test superset, redundant testing can be eliminated.

Consider the testing of register files using the March X algorithm (van de Goor, 1993), where the operation set performed on each register is O_{REG} = *{Write 0, Read 0, Write 1, Read 1}*. Here, the most intuitive implementation of the *WriteX* sub-operation could be the set of instructions *I(Reg, WriteX) = mov X,* R_N. In Table 2, we compare the set of instructions used to implement the O_{REG} = *{Write 0}* operation before and after instruction selection. As is typical in low-power MCUs, there is a differing execution time for instructions depending on the addressing mode used. In I_1, immediate-mode addressing of the *mov* instruction would, for example, take two clock cycles. The register-mode addressing used in all but the first instruction of I_2 would then take one clock cycle. Using instruction selection would save *n-1* clock cycles in the execution of the I_2 test over that of I_1, while properly propagating faults and conserving fault coverage.

It is also common that the testing of one MCU component results in collateral coverage of other, non-targeted components. For example, arithmetic component testing can cause collateral coverage of the data bus. Instruction combination is useful in eliminating overlapping fault coverage in order to reduce total SBST program length, while also conserving fault coverage. Two or more component tests should be combined into an SBST program superset, where redundant instructions can be identified and eliminated.

before	*after*
I_1:	I_2:
mov 0, R_1	mov 0, R_1
mov 0, R_2	mov R_1 , R_2
…	…
mov 0, R_N	mov R_{N-1} , R_N

Table 2. The Operation {Write 0} Before and After Instruction Selection

Operand Selection

Operand selection aims to reduce test energy consumption by minimizing the Hamming weight of all instruction operands and the Hamming distance of successive operands. Setting *don't–care* bits in the operand to logic 0 serves to minimize Hamming weight, while a minimum bitwise change between the operands of successive instructions minimizes Hamming distance. Minimizing Hamming weight works on the principle that less energy is required for the transfer of logic 0 than of logic 1 in the binary coding used on MCU data buses. Minimizing Hamming distance reduces transistor switching, and thus CMOS switching power, which makes up the majority of power consumption in integrated circuits like MCUs.

Consider ALU testing, where a set of instructions performs a single O_{ALU} = *{add with carry}* operation. In Table 3 the set of instructions I_4 implements the operation *add with carry* using two 0x8000 operands, which have a combined Hamming weight of 2. This is the lowest Hamming weight of any two operands that satisfy the test, and considerably lower than the combined Hamming weight of 32 in the operands of I_3. Additionally, instruction selection is performed on the second instruction of I_4 to reduce program length. Considering all of the instruction-level energy reduction methods covered in this section, and the SBST method proposed in (Kranitis et al., 2005), the pseudo-code for our energy-saving SBST method can be seen in Figure 5.

before	*after*
I_3:	I_4:
mov 0xFFFF, R_N	mov 0x8000, R_N
mov 0xFFFF, R_M	mov R_N , R_M
add R_N , R_M	add R_N , R_M

Table 3. The Operation {add with carry} Before and After Operand Selection

```
Extract the set of all microcontroller components $C = C_1, C_2, \ldots, C_n$;
Extract the set of all operations each component performs $O = O_1, O_2, \ldots, O_m$;
Extract the set of instruction sequences
    $I(C,O) = I_1, I_2, \ldots, I_p$, during execution, which cause component C to perform operation O;
Reorder C with the test priority from high to low;
for ( $i = 1$, I < $n$+1, i++ )
    Choose $C_i$ from set $C$;
    for ( $j = 1$, j < $m$+1, j++ )
        Choose $O_j$ from set $O$;
        Choose test sequence $I_k$ from $I(C_i,O_j)$ that has the smallest program length;
        Set operand values of $I_k$ to those with the least Hamming distance and weight;
        Apply $I_k$ with chosen operands to the input of $C_i$ and propagate the result to primary
        outputs;
    endfor
endfor
```

Figure 5. Energy-Reduced SBST Algorithm

ENERGY EFFICIENCY OF EMBEDDED MEMORY TESTING

There is a need for SBST programs that test the flash memory often found on WSN nodes. Flash memory is prone to failure over time and is a prime candidate for the implementation of the SBST concept. Unlike RAM, once flash memory has once been written to, it cannot be overwritten without first performing a block-erase. This makes conventional memory testing methods such as (van de Goor, 1991) unsuitable. Instead, a March-type algorithm called *March FT* is proposed in (Yeh, Wu, Cheng, Chou, Huang, & Wu, 2002), which can be seen in Equation 1. It has the highest fault coverage of all published approaches for most conventional faults and for all disturb faults. Here we aim to reduce the software energy required to perform the March FT algorithm. The extending of the single-bit-convert (SBC) memory addressing to flash memory is explored, as well as the interleaving of flash memory testing with other component tests.

$$(f); \downarrow (r1, w0, r0); \updownarrow (r0); (f); \uparrow (r1, w0, r0); \updownarrow (r0) \qquad (1)$$

where:

$\uparrow$ and $\downarrow$ are increasing and decreasing address orders, respectively.

(ex.: $\uparrow$ means from address 0 to address $n-1$).

$\updownarrow$ means address order is irrelevant.

n is the total size of memory to be tested.

$w0/1$ is writing 0 or 1 into a cell, respectively.

$r0/1$ is reading a cell, expecting a value of 0 or 1, respectively.

f is the erasing of a block of flash memory.

Single-bit-convert (SBC) Addressing for Memory Testing

The SBC memory addressing scheme was introduced in (Cheung & Gupta, 1996) as an energy-optimized way of addressing RAM for testing. It is an alternative to the way incremental addressing counts up or down through all testable memory addresses. SBC works by minimizing the Hamming distance between consecutive addresses, which reduces the switching activity on the address bus. In (Cheung & Gupta, 1996), address bus transitions are reduced by 50%, which accounts for a total energy consumption reduction of 18% to 77% depending on the size of RAM. As a candidate for use in March-type testing algorithms, SBC addressing can be used for any $\updownarrow$ portions where incremental addressing is not explicitly enforced.

Interleaving Flash Memory Testing with Other Tests

When a block-erase or chip-erase is performed on flash memory, the segment is generally filled with a sequence of logic 1 bits. Once a flash-write operation resets any bit to logic 0, the segment containing the bit must be block-erased to reset it back to logic 1. This erase operation is much slower than a read or write operation on flash memory. A block of embedded flash memory on-board an MCU, for example, can be erased by executing a series of instructions located in either flash memory or RAM. When flash memory is subjected to an erase operation, all timing control is transferred to an embedded flash controller and MCU execution is stalled until the erase operation completes. The MCU resumes execution after completion, provided a different segment than the one containing program code was erased. However, if the erase operation is initiated from RAM, the MCU will execute code from RAM while the flash erase is taking place. This allows the otherwise wasted MCU idle time to be used to execute other tests from RAM, and is the premise of the test interleaving shown in Figures 6a and 6b.

The efficiency of interleaving flash memory testing with other component tests depends upon several factors. The size of the flash block being erased, and hence its timing, must approximately match the time required to execute an unrelated component test. Figure 6a shows test routines requiring differing amounts of time and average power. In Figure 6b, the *other component test* is rescheduled into the same time slot as the flash-erase operation. This potentially causes a reduction in software energy consumption based upon the principle that executing simultaneous tests requires less energy than the sum of the energies of executing individual tests. This is because an MCU has an overhead power consumption, known as a baseline, simply being powered-on and ready to execute instructions. Equations for test time and test energy reductions can be seen in Equations 2 and 3, respectively.

$$\text{Time Reduction (\%)} = \frac{\mathrm{Min}(T_{FE}, T_O)}{T_{TOTAL}} \times 100 \qquad (2)$$

$$\text{Energy Reduction (\%)} = \frac{E_B - E_A}{E_B} \times 100 \qquad (3)$$

where:

$T_{TOTAL} = T_{CPU} + T_{FE} + T_{FP} + T_{FR} + T_O$ (T_{CPU}, T_{FE}, T_{FP} , T_{FR}, T_O are defined in Figure 6a).

E_B and E_A are the total software energy consumed before and after time interleaving, respectively.

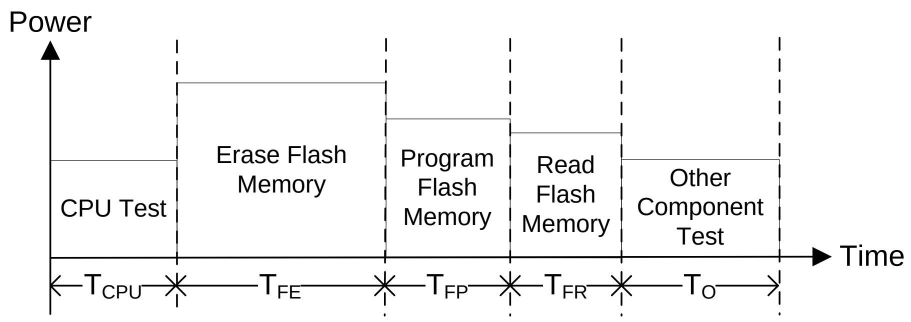

Figure 6a. Concept of Time Interleaving Individual Node Tests - Before Time Interleaving

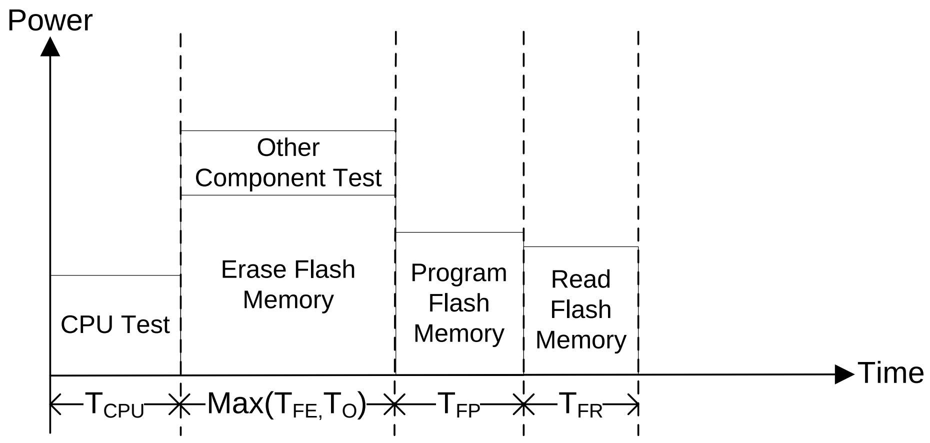

Figure 6b. Concept of Time Interleaving Individual Node Tests - After Time Interleaving

ENERGY EFFICIENCY THROUGH REDUCTION OF TEST DATA VOLUME

It can be seen that none of the methods mentioned in the previous sections can be applied to compressing the volume of test data contained in SBST programs. Since the programs either self-generate test vectors or contain them within the program code, a way to reduce test data volume is to compress the SBST programs themselves. In the following section we investigate the viability of compressing SBST programs by characterizing their structure.

Characterization of SBST Program Code

Data compression is the process of removing redundancy from a data set in order to represent the original with a smaller set. The less random a data set, the more redundancy that can be extracted to yield better compression. If the compression of SBST programs is to be considered, the programs themselves should be characterized as being of low entropy. An analysis is therefore performed on three real SBST programs from (Zhang, Zilic, & Radecka, 2006) used in testing WSN nodes, denoted P_n. As a useful measure of entropy, a histogram of symbol frequencies is produced by inputting SBST program machine code into MATLAB 7.0, with a symbol size of 1 byte. Code compilation is targeted at the Texas Instruments MSP430 MCU that is part of our WSN research platform.

Figure 7 shows that the frequency of some symbol occurrences is reasonably high in the measured SBST program. The many peaks and valleys visually depict that the program composition is somewhat redundant and thus a good candidate for compression. A similar analysis of two additional SBST programs found similarities in all of their frequently occurring bytes, which indicates that the programs share similar characteristics. These similarities can be attributed to:

1. Instruction set structure – The target MCU instruction set is composed of instructions for single-operand arithmetic, two-operand arithmetic, and conditional branching. Single-operand instructions share a 6-bit prefix such that the first byte of each instruction is *0x10–0x13* (bytes 16–19 in decimal). Likewise, the first byte of conditional branch instructions falls into the range *0x20–0x23* (bytes 32–35 in decimal).

2. Commonly used instructions and operands – Certain instructions are used more often in testing than others. For example, an efficient way to perform certain flash memory March testing is to write to an array of data elements (*mov* instruction), then read the array using an exclusive-or (*xor* instruction) between consecutive data elements. Certain operands are also more prevalent than others, such as passing *0x00* in the immediate addressing mode to test stuck-at-one faults in buses.

3. Shared code between programs – There is inherent overlap between SBST programs if they are to be executed independently of each other. Segments of code holding constants and performing hardware initialization must often be replicated.

In the following section, several general-purpose compression algorithms are explored for use on SBST programs due to the lack of appropriate SBST-specific techniques in the literature. Algorithm attributes are compared with each other and against node-specific requirements.

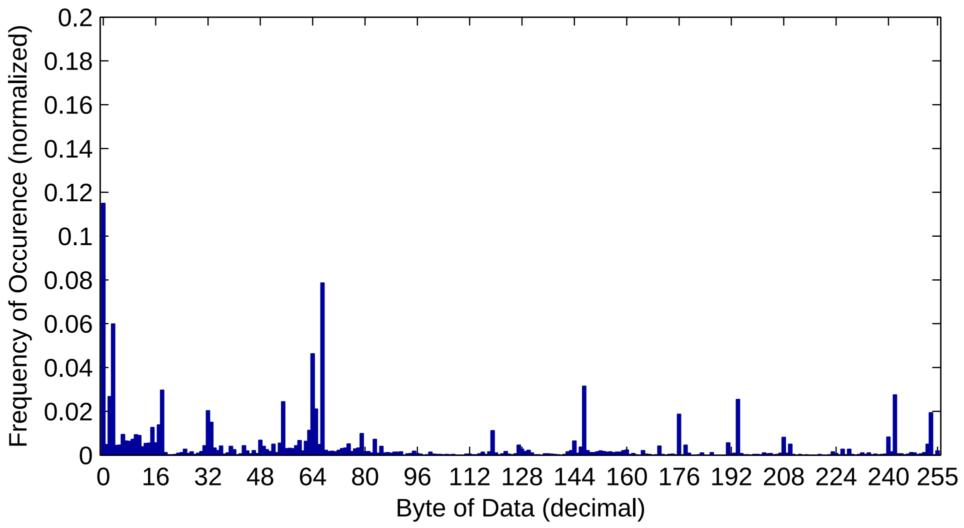

Figure 7. Data Byte Frequency of SBST Program P_1

General-Purpose Compression Algorithms

There has been extensive work performed on entropy coding algorithms over the years, a summary of which can be seen in (Lelewer & Hirschberg, 1987). To uncover an effective method of compressing SBST programs, in this section three groups of prominent algorithms are evaluated: Lempel-Ziv-Welch (LZW), Dynamic Huffman, and Bentley-Sleator-Tarjan-Wei (BSTW). The algorithms are compared against the following node-specific requirements:

1. Low energy consumption – By transferring compressed SBST programs instead of uncompressed ones, the power-hungry RF transceiver can have a smaller duty cycle and energy can be saved on-board a node.

2. Low cost – Of the components in a typical node, a disproportionate amount of cost is attributed to the MCU. A selection should therefore be carefully made in choosing an MCU meeting the minimum requirements of the WSN application. Since a typical application such requires modest resources, an MCU with small amounts of processing power, RAM, and flash memory is often enough. It is typical for an MCU to contain 1kB of RAM, which must be taken into account when selecting a compression algorithm. This is important since the price difference between a common MCU with 5kB of RAM compared to 1kB can be almost double according to Texas Instruments (n.d.).

LZW Compression

The Lempel-Ziv-Welch (LZW) algorithm (Welch, 1984) was innovative in its approach to codebook construction, since new symbols are created out of combinations of symbols already in the codebook (Lelewer & Hirschberg, 1987). This means that the algorithm takes $O(n^2)$ time (where *n* is the set of source symbols), since both source symbol and codebook matching can vary in length. To take advantage of the full benefits of the LZW algorithm, the codebook size should be allowed to grow to several times that of the symbol set size. This allows more complex symbol combinations to be constructed for greater compression efficacy. Since the MCU and transceiver chip on-board a WSN node typically have register boundaries set at one or two-byte increments, making efficient use of registers requires a minimum codebook size of 256 symbols. Using even this small symbol size, the LZW algorithm would quickly occupy the limited memory resources of the node. Even if the codebook is limited to 4*S* entries (where *S* denotes the number of unique symbols of the codebook), in a worst-case scenario the remaining 3*S* entries would require about 300kB of memory. Nevertheless, this algorithm is considered because of its ubiquity and as a measure of the compression performance that can be achieved with larger amounts of memory.

Dynamic Huffman Coding

Dynamic Huffman coding is an adaptive compression scheme that, unlike static Huffman coding (Huffman, 1952), dynamically generates its codebook as compression is taking place. One of the schemes known as Dynamic Huffman coding was made by Faller (1973), Gallager (1978), and Knuth (1985), after which it came to be known as Algorithm FGK. Different improvements were later made by Vitter (1987). As source symbols are mapped to codes, they are placed into a codebook with a hierarchical tree data structure. The observed frequency of source symbols is used to swap branches of the tree in order to continually give frequently occurring symbols smaller codes. As with all adaptive methods, compression performance improves as experience is built. The advantage these methods hold over original Huffman compression is that they only require $O(n)$ time, and that the codebook need not be transmitted since the receiver may reconstruct it by adapting his codebook lockstep with the sender.

While dynamic Huffman techniques offer a complexity less than that of LZW compression, the implications of maintaining a large hierarchical tree codebook are that it too requires significant memory resources. Each codebook node consists of the symbol frequency and the symbol itself, while the Huffman code of each symbol can be inferred from its position in the tree. Since nodes and branches of the tree are often swapped, in a practical implementation they must also be addressed with pointers which themselves use memory. The result of tracking symbol frequency is that memory usage grows logarithmically as n increases. To limit memory usage, an upper limit to symbol frequency can be declared, in which case the algorithm will become static beyond that limit. Alternatively, the frequency tables can be deleted and statistics gathering restarted, but this negates the experience that the codebook has built.

BSTW Algorithm

This algorithm by Bentley, Sleator, Tarjan and Wei (BSTW) (1986) is an adaptive compression method simpler than the dynamic Huffman coding that can outperform static Huffman in some cases (Bentley et al., 1986). As source symbols are input, the most frequent ones are given the shortest codes. In coding a source symbol, the algorithm outputs a code based upon its location in the codebook, then uses a Move-to-Front (MTF) scheme to place that symbol at the front of the codebook. This scheme ensures that symbols occurring frequently are efficiently encoded with small codes, especially if they appear in bursts. More importantly, symbol frequency is not explicitly tracked, as this is implicit in a symbol's position in the codebook and does not require additional memory.

The algorithm itself requires only $O(n)$ time, and transmission of the codebook is also unnecessary for the same reasons as with dynamic Huffman techniques. In BSTW the outputted codes are actually indices to a changing codebook. The source data set is remapped from its original distribution into a data set resembling a *geometric distribution*, which can be imagined as source symbols being sorted into an order of descending frequency. An example of a reordered version of SBST program P_1 can be seen in Figure 8, whose original representation is in Figure 7. Also seen in Figure 8 is a superimposed geometric distribution which approximates the BSTW-reordered P_1.

Several implementation options for BSTW are possible, where the complexities of accessing the codebook data structure differ. The pseudo-code for one version of the algorithm is depicted in Figure 9, which in addition to a codebook uses an equally-sized structure to cross-reference codebook data. A linear search of $O(n)$ is then saved since $2S$ memory is used instead of only S in the smallest implementation. After applying the BSTW algorithm, the resulting dataset must be re-encoded with a scheme such as universal codes or Golomb-Rice codes in order to achieve a smaller representation (compression). Such coding schemes give optimal or near-optimal results to data sets following geometric distributions. In the following section the three groups of compression algorithms that have been presented will be compared to each other to find a good candidate for use on SBST programs.

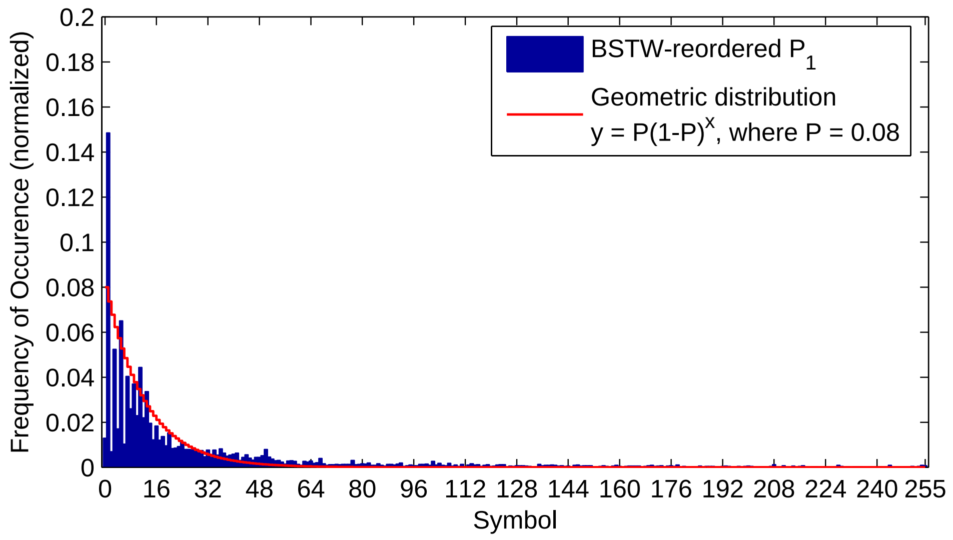

Figure 8. Data Byte Frequency of BSTW-reordered SBST Program P_1 vs. Geometric Distribution $y = P(1-P)^x$ (where P = 0.08)

```
// codebook stores symbols, indices stores indices of codebook, inverse-indexed by symbol.
for ( i = 0, I < size_of_source, i++ )
    index = indices [current_symbol]
    destination = index                                   // write index to destination
    if ( index ≠0 )
        Remove current symbol from codebook and shift codebook to fill hole
        Fill indices with new index to codebook
        Place current_symbol at front of codebook and fill indices with its index
    endif
    Increment source and destination by a symbol
endfor
```

Figure 9. BSTW Algorithm Pseudo-code

Compression Algorithm Comparison

In evaluating a compression algorithm for its potential suitability, the metrics of algorithmic complexity and memory footprint can be used as direct measures of meeting relevant node-specific requirements. These metrics are therefore compared in Table 4 between the three aforementioned algorithms, where *n* represents the number of symbols processed, and *S* denotes the number of unique symbols of the codebook. The following section explores coding schemes which need to be applied to some compression methods, including BSTW, for unambiguous decompression to be possible. The coding schemes are evaluated based upon the same requirements as the algorithm comparison: complexity and memory requirements.

Compression Algorithm	Complexity over Source Data	Codebook Memory Usage (worst case)
Lempel-Ziv-Welch (LZW)	$O(n^2)$	$(S-1)+\frac{n(n+1)}{2}$
Dynamic Huffman (Algorithm FGK)	$O(n)$	$(S-1)\log_2(n)+S+Pointers$
Algorithm BSTW	$O(n)$	$2S$

Table 4. Compression Algorithm Comparison – Complexity and Memory Usage

Static Coding

Applying the BSTW algorithm described above on a set of source data coded in binary would generate a reordered dataset still in binary coding. In order to achieve compression, the dataset must be transformed into a smaller representation. Re-encoding the dataset using a static code can achieve this, since static codes are effectively a simplified form of compression that makes little or no use of probability estimation. Instead, they are generated in such a way as to be optimally suited to one particular probability density function (PDF). For example, the output data of the BSTW algorithm has been shown in Figure 8 to approximately conform to geometric distribution, so a static code suited to a geometric distribution could be used to express it. A code containing intrinsic delimiter bits can also be generated so that boundaries between codewords can be detected, which allows the resulting dataset to be unambiguously decoded. Codes such as these are said to contain the *prefix property*, such that no code is a prefix of any other code in the dataset. In the following discussion several types of coding are evaluated for complexity and memory requirements, as per the requirements already discussed.

Universal Codes

Universal codes are often used in conjunction with adaptive schemes because of the benefits of their prefix property. These codes map positive binary-coded integers into variable-length binary codewords. If the source symbol set is remapped in descending order of frequency (*monotonically decreasing*), such as BSTW-remapped Figure 8 is close to being, the advantage of using them includes the property that the resulting codes will be within a constant factor of the optimal code (Lelewer & Hirschberg, 1987). Asymptotically optimal universal codes include Elias-δ and Fibonacci codes, while Elias-γ codes are not (Lelewer & Hirschberg, 1987). All three codes include intrinsic inter-symbol delimiters, and a comparison between the mapping of their symbols can be seen in Table 5. Each code is catered to a different probability distribution, but notable differences includes the fact that errors in an Elias-γ code are often not recoverable, while in Fibonacci codes a 1-bit error can at most cause the loss of 3 symbols before the decoder is resynchronized. Each of the codes is also generated according to algorithms of differing complexity, but Fibonacci codes are especially difficult to develop as the most popular method is recursive. This property would likely preclude Fibonacci codes from being generated as needed for *on-the-fly* decoding, and an implementation would need to store the codes in system memory for lookup purposes.

Rice Coding

Rice coding (Rice, 1979), a special form of Golomb coding (Golomb, 1966), is a form of static coding that allows for a degree of probability estimation to accommodate different source symbol PDFs. Codes are generated by dividing a source symbol by a *divisor*, where the quotient of the division operation is represented in unary coding and the remainder in binary coding. Unary and Rice codes for some symbols can be seen in Table 5, both of which can be seen to also include inter-symbol delimiters. Rice codes differ from Golomb codes in that their divisor setting is restricted to powers of 2. This restriction ensures that the division operation can be performed on an MCU by simple bit-shift instructions, while the remainder is a logical bit-mask with the original symbol. Such a simple implementation requires a minimum of memory and allows codes to be generated deterministically on-the-fly as source symbols are processed. Compared to the generation algorithms of universal codes already reviewed, Rice coding is potentially of equal or less complexity. Other promising advantages of using Rice coding include the ability to vary the divisor setting between datasets without penalty, making the coding scheme more flexible than universal coding.

Symbol	Static Code				
	Elias-δ	Elias- γ	Fibonacci	Unary	Rice*
0	0	0	11	1	1 00
1	1000	100	0 11	01	1 01
2	1001	101	00 11	001	1 10
3	10100	11000	10 11	0001	1 11
4	10101	11001	000 11	00001	01 00
5	10110	11010	100 11	000001	01 01
6	10111	11011	010 11	0000001	01 10
7	11000000	1110000	0000 11	00000001	01 11

* with divisor setting of 4.

Table 5. Comparison of Symbol Coding for Various Static Codes

EXPERIMENTAL RESULTS

WSN Research Platform

Energy measurement and RF module characterization experiments are performed with a real WSN node, known as the *WSN research platform*, consisting of a Texas Instruments MSP430F149 MCU and an RF module, assembled on a custom printed circuit board (PCB). The RF module is made up of a Chipcon CC2420 wireless transceiver, employing the IEEE 802.15.4/ZigBee protocol at 2.4GHz, and a printed dipole antenna. The transceiver has 8 programmable transmission power levels which range from 0dBm to -25dBm. It also has an integrated received signal strength indicator (RSSI) which gives a discrete value for received signal power that can be read from an internal register. The transmit and receive gains of the on-board antenna, respectively G_T and G_R, have both been experimentally found to be -8 dB from prior work in (Chenard, Zilic, Chu, & Popovic, 2005).

Energy consumption measurements are made by connecting the WSN research platform to a custom current-measurement circuit depicted in Figure 10. Both the research platform and current-measurement circuit are developed in-house (Zhang, Zilic, & Radecka, 2006; Zhang, 2005). The output of the current-measurement circuit includes a voltage that is measured across a known resistance, giving an instantaneous current reading. Small currents are thus amplified to easily-measured voltages. This allows instantaneous power to be found, which when plotted over time, can be integrated to find total node energy consumption. All current-measurement circuit outputs are analyzed in this way using an Agilent Infiniium 54830D 600MHz mixed-signal oscilloscope. A portion of the node data bus (D_0–D_3) is forwarded to the oscilloscope to differentiate between test routines. The energy consumed by those test routines can then be calculated with Equation 4.

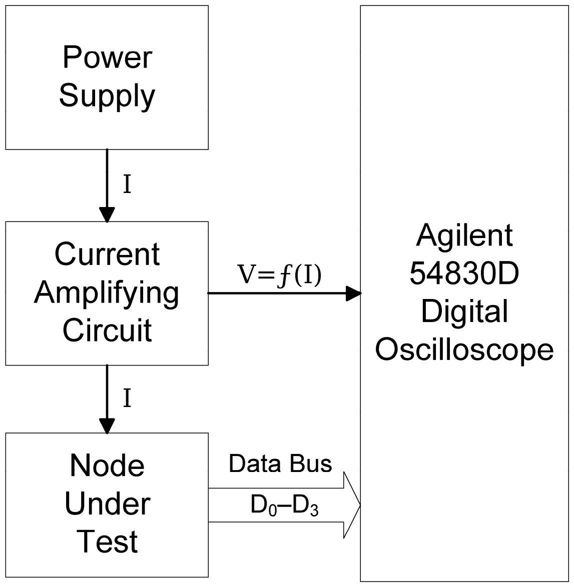
Power Supply
I
Current Amplifying Circuit
V=ƒ(I)
I
Node Under Test
Data Bus
D0–D3
Agilent 54830D Digital Oscilloscope

Figure 10. Current Measurement Method

$$E = \frac{V_{dd}}{R}\int_{t_N}^{t_M} V\,dt \tag{4}$$

where:

E is energy, in joules.

V_{dd} is the supply voltage to the node.

t_N and t_M are the measurement start and finish times, respectively.

V = *f(*I*)* is the voltage output of the current measurement circuit.

I is the instantaneous current drawn by the node.

R is the resistance across which V is measured.

RF Module Characterization

The previously described RF module test scheme is tested here against select IEEE 802.15.4 specifications shown in Table 1. Experimental results for this RF module characterization can be seen in Table 6, which can be seen to all exceed the required specification.

The low-power IEEE 802.15.4 communication protocol has been developed to favor the use of battery-powered nodes. Such nodes are able to save energy by duty-cycling their MCU and RF transceiver, wherein they would spend most of their operational lives in a sleep mode. The protocol allows nodes to listen for periodic beacon transmissions in order to determine if a message is pending for them. Since beacon frequency can be varied, this mechanism allows the application designer to decide on a balance between node energy consumption and message latency.

Table 7 shows energy consumption measurements of various node operating modes. It can be seen that mode 4 costs much less energy than mode 5. A periodic beacon network therefore offers a sizable node power savings, and its frequency can be catered to the timing requirements of the application in order to maximize energy savings. This can be done to periodically transfer and activate SBST programs on the node. To save additional energy, the node transmission power level should be set as low as possible while maintaining the required PER on the link.

Specification	Requirement	Measured Result
Transmit Power	Minimum: -3 dBm	-1 dBm
Receiver sensitivity	Maximum: -85 dBm	-88 dBm
Adjacent channel rejection	Minimum: 0 dB	7 dB
Alternate channel rejection	Minimum: 30 dB	43 dB

Table 6. RF Module Test Results

Mode	Mode Description	Current (mA)
1	Send 5 packets with maximum power level	20.0
2	Send 5 packets with medium power level	15.0
3	Send 5 packets with minimum power level	12.0
4	Sleep between periodic beacon packets	0.39
5	Continuous listen on RF channel	20.7

Table 7. RF Module Operating Modes

MCU Testing Efficiency

The energy efficiency ideas for MCU testing we have previously discussed are experimentally tested here on our WSN research platform. Table 8 shows the energy consumption of an MCU SBST program before and after optimizations. The measurements show that operand and instruction selection achieved a 21.2% energy reduction in the running of the test, while also achieving a 20.1% test time reduction.

Metric	Unmodified Test	After Operand Selection	After Operand and Instruction Selection
Energy (μJ)	2.558	2.430	2.014
CPU cycles	940	940	751

Table 8. Node Energy Consumption during MCU Testing

Embedded Memory Testing Efficiency

Two techniques were previously proposed to improve the energy efficiency of embedded memory SBST programs: SBC addressing, and the interleaving of flash memory testing with other tests. Here we compare the energy consumption of SBC addressing against binary addressing, and determine the energy effect of interleaving flash memory testing with either RAM testing or RF module testing. The testing modes, themselves portions of component tests, can be seen in Table 9. Modes 5–7 are the first three elements of March FT algorithm for flash memory testing while modes 8 and 9 are the main elements of the March X algorithm used for RAM testing. Modes 8 and 9 are made to iterate on a single RAM memory block 10 times in order to approximate the time required for mode 5. Since our WSN research platform contains only 2kB of RAM, this is used as a strategy to show the maximum energy savings effect of fully interleaving a flash memory block erase.

Mode	Mode Description
5	Flash memory block erase (512 bytes)
6	Flash memory block test (*r*1, *w*0, *r*0) (512 bytes)
7	Flash memory block test (*r*0) (512 bytes)
8	RAM block test (*w*0) (10 blocks = 5 kB)
9	RAM block test (*r*0) (10 blocks = 5 kB)
A	RF Module Initialization
B	RF packet transmission (4 packets)

Table 9. Test Mode Descriptions

SBC Addressing of Flash Memory

The current consumption of SBC addressing compared to binary addressing can be seen in Figure 11. As expected, the average current of the SBC addressing can be seen to be slightly lower than that of binary addressing. However, SBC addressing requires greater test time, which means its energy consumption *(power × time)* is greater than that of binary addressing. Interestingly, this result is contrary to the one found in (Cheung & Gupta, 1996). The increased test time can be attributed to the need for extra instructions such as *shift*, *xor*, and *mov*, which convert addresses from binary to SBC. The additional current consumption of these extra instructions then exceeds the energy savings of using SBC addressing.

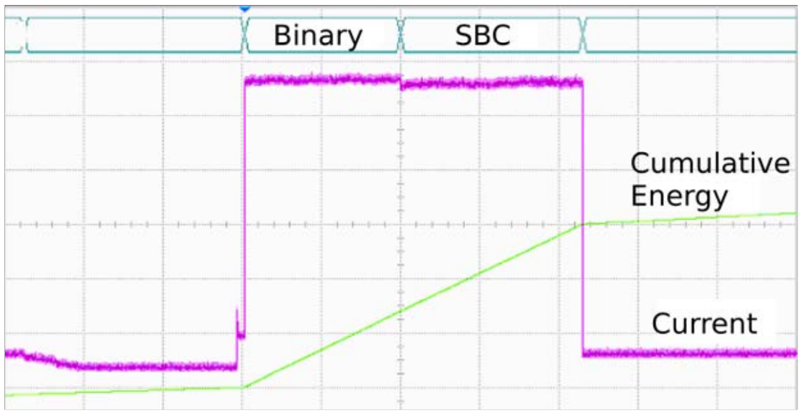

Figure 11. Energy Comparison of Binary and SBC addressing

Interleaving of Flash Memory Testing with Other Tests

The concept we introduced of interleaving flash memory testing with other tests is experimentally tested here by running RAM testing and RF module testing during a flash memory block erase operation. Energy measurements for the interleaving of flash memory erase and RAM testing can be seen in Table 10, which show a 13.4% energy reduction and a 20.7% time reduction with the use of interleaving. Similarly, results for the interleaving of flash memory erase with RF module testing seen in Table 11 yield a 10.7% energy reduction and 14.2% time reduction with the use of interleaving. In the case of interleaving RAM testing, it is apparent that modes 8 and 9 are slightly synthetic in their use of a 10-iteration block operation. However, a meaningful energy savings would result in the case of an MCU with at least 5 kB of RAM to be tested. It should also be noted that the increased energy consumption seen in Table 11 over results of Table 10 for the same test modes is a result of the contribution of the RF module to total node energy consumption.

Mode	Test Item	Energy Before Interleaving (μJ)	Energy After Interleaving (μJ)
5	Flash memory erase	143.10	189.94
8	RAM (*w*0)	73.06	
9	RAM (*r*0)	78.54	
6	Flash (*r*1, *w*0, *r*0)	413.16	421.77
7	Flash (*r*0)	7.92	7.86
	Total Energy (μJ)	715.78	619.57
	Total Time (ms)	91.24	72.3

Table 10. Interleaving Flash Memory Testing with RAM Testing

Mode	Test Item	Energy Before Interleaving (μJ)	Energy After Interleaving (μJ)
5	Flash memory erase	231	1276
A	RF Initialization	280.83	
B	RF packet transmission	983	
6	Flash ($r1$, $w0$, $r0$)	723.03	707.46
7	Flash ($r0$)	13.2	12.87

Total Energy (μJ)	2236	1996
Total Time (ms)	105.2	90.24

Table 11. Interleaving Flash Memory Testing with RF Module Testing

SBST Program Compression

The first experiment compares the compression ratios of the three general-purpose algorithms previously presented on three real SBST programs. Results include an estimate of the minimum memory requirements for using each algorithm, based upon knowledge of their functioning and required data structures. The second experiment measures the energy required to receive compressed and uncompressed SBST programs on a real WSN node.

Experimental Setup

In the experiment comparing compression ratios, three real SBST programs from (Zhang, Zilic, & Radecka, 2006; Zhang, 2005) (denoted P_n) are used with algorithm families LZW, Dynamic Huffman, and BSTW. The SBST programs are compiled into machine-code for the MCU used in our *WSN research platform*. Compression ratio results for algorithm LZW are collected using the UNIX utility *compress* v.4.2.4, while for the dynamic Huffman family of algorithms, an implementation of algorithm FGK created by Toub (n.d.) is used. To evaluate the performance of several static coding methods, two variations of algorithm BSTW are actually implemented in MATLAB 7.0. Functions for determining the length of various static codes are also implemented in MATLAB 7.0, which are applied to the BSTW-remapped data. As an example, the equation for determining the length of a Rice-encoded symbol is given in Equation 5.

The energy measurement experiment is performed with the WSN research platform. The energy contribution of only the transceiver is found by measuring the node energy consumption while the MCU is in a low-power sleep state and the transceiver is operating in *listen mode*. Since the current draw of the *listen mode* approximates the published current draw of *receive mode* of the CC2420 chip according to Chipcon AS (2004), it is used to isolate the transceiver component of total WSN node power consumption. Such an approximation is necessary since the transceiver would otherwise never be in *receive mode* with the MCU in a low-power sleep state.

$$\left\lfloor \frac{symbol}{divisor} \right\rfloor + 1 + \log_2(divisor) \tag{5}$$

Compression Ratio Comparison

While algorithm BSTW has been introduced as solely using the MTF heuristic, in this experiment an alternate heuristic denoted *swap* is also explored. In the MTF heuristic, when a new symbol is encountered it is moved to the front of the list, while using the swap heuristic it is exchanged with the element at the front of the list. A comparison of compression ratios and memory usage for the aforementioned algorithms can be seen in Table 12.

Algorithm LZW gives both the best compression ratios across all three SBST programs, as well as the greatest memory usage. Since results are collected using a compiled utility, the minimum memory requirements are estimated to be 8kB, although the value is likely much higher. Even this very conservative estimate is enough to disqualify algorithm LZW from use on WSN nodes for requiring excessive memory resources, although its results are a useful benchmark for gauging the relative compression performance of the other algorithms. Programs P_1 and P_2 are compressed by LZW in the 30% range while P_3 sees a 55% reduction.

The memory requirements for algorithm FGK are estimated from the analysis in Table 4 for the same reasons as algorithm LZW, although it is found to require approximately 1.7kB of memory. Achieved compression ratios exceed those of BSTW (MTF) with Rice codes for divisor settings of 8 and 64, for all three SBST programs. Compared to BSTW (MTF) with universal codes, performance is better for P_1 and P_3 by a small margin.

Algorithm BSTW with static codes has the lowest memory requirements of those compared (see Table 4), at 0.5kB. When heuristic MTF is combined with Rice coding with a divisor of 32, better performance over algorithm FGK is seen for P_1 and P_2 by a slim margin. Using a divisor of 16, the performance of algorithm FGK is exceeded by a slightly larger margin for the same programs. Algorithm FGK gives better results for P_3 in all cases at a cost of 3x the memory, although by a slim 3.6% compared to Rice coding with a divisor of 16.

Algorithm		Compression Ratio (%)			RAM (Bytes)
		P_1	P_2	P_3	
Uncompressed		0	0	0	0
Lempel-Ziv-Welch (LZW)		38.8	32.2	55.4	7936†
Dynamic Huffman (Algorithm FGK)		15.8	19.4	27.5	1728‡
BSTW (MTF)	Rice Codes (divisor = 8)	7.7	16.7	18.4	512
	Rice Codes (divisor = 16)	18.8	22.5	23.9	
	Rice Codes (divisor = 32)	17.3	19.7	19.5	
	Rice Codes (divisor = 64)	9.9	10.3	10.7	
	Elias-γ Codes	8.6	24.2	19.2	
	Fibonacci Codes	15.6	23.9	22.2	
BSTW (swap)	Rice Codes (divisor = 8)	-31.9	-24.7	-29.6	512
	Rice Codes (divisor = 16)	-1.4	3.1	-0.6	
	Rice Codes (divisor = 32)	7.4	10.3	7.6	
	Rice Codes (divisor = 64)	5.3	7.8	5.3	
	Elias-γ Codes	-2.8	-14.7	5.1	
	Fibonacci Codes	8.3	0	13.4	

† codebook of 4096 strings, where the first 256 symbols are 1 byte, and the rest at least 2 bytes (very conservative estimate). ‡ assuming 256 symbols, 16 kB blocks, and 2 byte pointers.

Table 12. Achieved Compression Ratio vs. Memory Usage for three SBST Programs (denoted P_n)

Energy Expenditure Comparison

Reducing the volume of data received by the node directly leads to a reduction in node energy consumption, since the transceiver presents the highest power consumption of the system. In this experiment the goal is to quantify the energy savings of transferring compressed SBST programs to a WSN node from a basestation. Algorithm BSTW and Rice coding (with a divisor of 16) is used for its compatibility with the memory limitations of nodes. Both compressed and uncompressed versions of the same three SBST programs from Table 12 are transmitted, and the total energy consumption of the node is measured in receiving the programs.

When the transceiver is receiving data, the ratio of current draw of the MCU to the transceiver is found to be approximately 1:4.5, seen in Table 13. In isolating the transceiver current-draw, the energy usage for receiving both compressed and uncompressed SBST programs can be seen in Table 14. The result is that the reception of compressed SBST programs over uncompressed ones yields an energy savings of 18.8%, 22.5%, and 23.9% for programs P_1, P_2, and P_3, respectively. These energy savings are directly proportional to the achieved compression ratios seen in Table 12.

Component	Average Current (mA)
MCU operation	4.327
Transceiver operation	20.970

Table 13. Contribution of current-draw for components when transceiver is receiving data

Wireless Reception of SBST Program Type	Energy Usage (mJ)		
	P_1	*P_2*	*P_3*
Uncompressed	14.727	0.779	9.420
Compressed	11.955	0.603	7.167

Table 14. Energy Usage for Wireless Reception of three SBST Programs (denoted P_n)

CONCLUSION

In this chapter we present an infrastructure for the distribution and remote execution of SBST tests, as well as for the remitting of test responses. As part of the testing scenario, both self-testing and testing through *helper nodes* is used to verify the correct operation of node components. Once uncovered, failed and failing nodes can be repaired or replaced before they affect network availability.

SBST programs are constructed with instructions requiring the least amount of cycles, while their operands are selected to contain the least Hamming distance and weight. This reduces test energy consumption, verified through experimental results collected from a current measurement circuit connected a WSN research platform. Further, a March-family algorithm is used to test embedded flash memory, where it is shown that the energy optimizations traditionally yielded by SBC addressing are not efficient in this case. The time interleaving of embedded flash memory tests with other components tests is used to reduce both test time and energy, while the use of *helper nodes* also enables the verification of RF module specifications.

Finally, SBST program compression is explored through general-purpose algorithms in an effort of further reduce the energy consumption associated with the distribution of test programs. While most algorithms require more memory than is available on a typical node, the BSTW algorithm is found to suit such devices. A compression scheme involving the BSTW algorithm combined with the memory-less Rice coding is found to give the best overall compression ratio for devices with small amounts of memory.

REFERENCES

Aho, A., & Johnson, S. (1976). Optimal Code Generation for Expression Trees. *Journal of the ACM*, 23(3), 488-501.

Batcher, K., & Papachristou, C. (1999). Instruction randomization self test for processor cores. *Proceedings of the VLSI Test Symposium* (pp. 34-40).

Bentley, J. L., Sleator, D. D., Tarjan, R. E., & Wei, V. K. (1986). A locally adaptive data compression scheme. *Communications of the ACM*, 24(4), 320-330.

Brahme, D., & Abraham, J. A. (1984). Functional testing of microprocessors. *IEEE Transactions on Computers*, (C-33), 475-485.

Callaway, E. H. (2004) *Wireless sensor networks: architectures and protocols*. Boca Raton, FL: Auerbach Publications.

Chang, N., Kim, K., & Lee, H. G. (2000). Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. *Proceedings of the International Symposium on Low Power Electronics and Design* (pp. 185-190).

Chen, L., & Dey, S. (2001). Software-based self-testing methodology for processor cores. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (20), 369-280.

Chenard, J.-S., Zilic, Z., Chu, C. Y., & Popovic, M. (2005). Design methodology for wireless nodes with printed antennas. *Proceedings of the Design Automation Conference* (pp. 291-296).

Cheung, H. & Gupta, S. K. (1996). A BIST methodology for comprehensive testing of RAM with reduced heat dissipation. *Proceedings of the International Test Conference* (pp. 386-395).

Chiang, M. W., Zilic, Z., Radecka, K., Chenard, J.-S. (2004). Architectures of increased availability wireless sensor network nodes. *Proceedings of the International Test Conference* (pp. 1232-1241).

Chipcon AS. (2004). *CC2420 data sheet* (revision 1.4). Oslo, Norway: Chipcon AS.

Dabrowski, J. (2003). BiST model for IC RF-transceiver front-end. *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems* (pp. 295-302).

Faller, N. (1973). An adaptive system for data compression. *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers* (pp. 593-597).

Gallager, R. G. (1978). Variations on a theme by huffman. *IEEE Transactions on Information Theory*, 24(6), 668-674.

Golomb, S. W. (1966). Run-length encodings. *IEEE Transactions on Information Theory*, 12(3), 399-401.

Halder, A., Bhattacharya, S., Srinivasan, G., & Chatterjee, A. (2005). A system-level alternate test approach for specification test of RF transceivers in loopback mode. *Proceedings of the International Conference on VLSI Design* (pp. 289-294).

Hariri, S., & Huitlu, H. (1995). Hierarchical modeling of availability in distributed systems. *IEEE Transactions on Software Engineering*, (21), 50-56.

Hely, D., Bancel, F., Flottes, M.-L., & Rouzeyre, B. (2006). Secure scan techniques: A comparison. *Proceedings of the International On-Line Testing Symposium* (pp. 119-124).

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9), 1098-1101.

Institute of Electrical and Electronics Engineers. (2006). *Wireless medium access control (MAC) and physical layer (PHY) specifications for low rate wireless personal area networks (LRWPANs)* (IEEE standard 802.15.4).

Joshi, B. S., & Hosseini, S. H. (1998). Efficient algorithms for microprocessor testing. *Proceedings of the Annual Reliability and Maintainability Symposium* (pp. 100-104).

Khoche, A. Volkerink, E., Rivoir, J., & Mitra, S. (2002). Test vector Compression Using EDA-ATE synergies. *Proceedings of the VLSI Test Symposium* (pp. 97-102).

Knuth, D. E. (1985). Dynamic huffman coding. *Journal of Algorithms*, 6(2), 163-180.

Koushanfar, F., Potkonjak, M., & Sangiovanni-Vincentelli, A. (2004). Fault tolerance in wireless sensor networks. In I. Mahgoub and M. Ilyas (Eds.), *Handbook of Sensor Networks* (section VIII), Boca Raton, FL: CRC Press.

Kranitis, N., Paschalis, A., Gizopoulos, D., & Zorian, Y. (2002). Effective software self-test methodology for processor cores. *Proceedings of Design, Automation and Test in Europe* (pp. 592-597).

Kranitis, N, Paschalis, A., Gizopoulos, D., & Xenoulis, G. (2005). Software-based self-testing of embedded processors. *IEEE Transactions on Computers,* (54), 461-475.

Krstic, A., Lai, W.-C., Cheng, K.-T., Chen, L., & Dey, S. (2002). Embedded software-based self-test for programmable core-based designs. *IEEE Design & Test of Computers,* 19(4), 18-27.

Lala, P. K. (2001). *Self-checking and fault-tolerant digital design* (1st ed.). San Francisco, CA: Morgan Kaufmann.

Lelewer, D. A., & Hirschberg, D. S. (1987). Data compression. *ACM Computing Surveys*, 19(3), 261-296.

Liang, H.-G., Hellebrand, S., & Wunderlicht, H.-J. (2001). Two-dimensional test data compression for scan-based deterministic BIST. *Proceedings of the International Test Conference* (pp. 894-902).

Nikolaidis, S., & Laopoulos, T. (2001). Instruction-level power consumption estimation embedded processors low-power applications. *International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications* (pp. 139-142).

Ozev, S., Orailoglu, A., & Olgaard, C. V. (2002). Multi-level testability analysis and solutions for integrated bluetooth transceivers. *IEEE Design & Test of Computers*, 19(5), 82-91.

Paschalis, A., & Gizopoulos, D. (2005). Effective software-based self-test strategies for on-line periodic testing of embedded processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* (24), 88-99.

Raghavendra, C. S., Kumar, V. K. P., & Hariri, S. (1988). Reliability analysis in distributed systems. *IEEE Transactions on Computers*, (37), 352-358.

Rice, R. F. (1979). *Some practical universal noiseless coding techniques* (JPL Technical Report No. 79-22). Pasedena, CA: Jet Propulsion Laboratory.

Salomon, D. (2004). *Data compression: The complete reference* (3rd ed.). New York, NY: Springer.

Semiconductor Industry Association. (2005). *International technology roadmap for semiconductors*: *Test and test equipment.*

Shen, J., & Abraham, J.A. (1998). Native mode functional test generation for processors with applications to self test and design validation. *Proceedings of the International Test Conference* (pp. 990-999).

Stutzman, W. L., & Thiele, G. A. (1997). *Antenna Theory and Design* (2nd ed.). New York, NY: John Wiley & Sons.

Thatte, S. M., & Abraham, J. A. (1980). Test generation for microprocessors. *IEEE Transactions on Computers*, (C-29), 429-441.

Texas Instruments Corporation. (n.d.). *MSP430 ultra-low power microcontrollers, MSP430x1xx – flash ROM no LCD – price list per 1000 units.* Retrieved February, 2007, from http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?familyId=911&sectionId=95&tabId=1527&family=mcu

Toub, S. (n.d.). *Adaptive Huffman compression*. Retrieved July, 2002, from http://www.gotdotnet.com

van de Goor, A. (1991). *Testing semiconductor memories: theory and practice*. New York, NY: John Wiley & Sons.

van de Goor, A., & Verhallen, T. (1992). Functional testing of current microprocessors (applied to the Intel i860). *Proceedings of International Test Conference* (pp. 684-695).

van de Goor, A. (1993). Using march tests to test SRAMs. *IEEE Design & Test of Computers*, 10(1), 8-14.

Vitter, J. S. (1987). Design and analysis of dynamic huffman codes. *Journal of the ACM*, 34(4), 825-845.

Welch, T. A. (1984). A technique for high-performance data compression. *IEEE Computer*, 17(6), 8-19.

Yeh, J.-C., Wu, C.-F., Cheng, K.-L., Chou, Y.-F., Huang, C.-T., & Wu, C.-W. (2002). Flash memory built-in self-test using march-like algorithms. *IEEE International Workshop on Electronic Design, Test and Applications* (pp. 137-141).

Zhang, R. (2005). *Energy reduced software-based self-testing for wireless sensor network nodes*. Master of Engineering thesis, McGill University, Montreal, Quebec, Canada.

Zhang, R., Zilic, Z., & Radecka, K. (2006). Energy efficient software-based self-test for wireless sensor network nodes. *Proceedings of the VLSI Test Symposium* (pp. 186-191).

KEY TERMS & DEFINITIONS

Automatic test pattern generation (ATPG): An algorithm by which test vectors are automatically generated from a design specification.

BSTW: An efficient, adaptive compression algorithm that requires little memory.

Built-in self-test (BIST): A hardware module that generates test vectors, applies them to testable components, and constructs a test signature from the aggregated results.

Energy-sensitive factors: the parameters of instructions that a microcontroller executes which affect its total energy consumption.

Fault coverage: Based upon a fault model, the proportion of all possible faults that are tested against by a test program.

Packet error rate (PER): The proportion of packets that have been received over a wireless link containing one of more errors.

Perceived availability (*of a network*): The probability that a network is functioning correctly over a period of time, as seen by a user of that network.

Rice coding: A form of static coding that allows for a degree of probability estimation to accommodate different source symbol probability densities.

Software-based self-test (SBST) program: Software created to test the components of the microcontroller is it executing on, and/or attached peripherals.