Immunotronics—Novel Finite-State-Machine Architectures With Built-In Self-Test Using Self–Nonself Differentiation

D. W. Bradley, *Member, IEEE,* and A. M. Tyrrell, *Senior Member, IEEE*

***Abstract*—A novel approach to hardware fault tolerance is demonstrated that takes inspiration from the human immune system as a method of fault detection. The human immune system is a remarkable system of interacting cells and organs that protect the body from invasion and maintains reliable operation even in the presence of invading bacteria or viruses. This paper seeks to address the field of electronic hardware fault tolerance from an immunological perspective with the aim of showing how novel methods based upon the operation of the immune system can both complement and create new approaches to the development of fault detection mechanisms for reliable hardware systems. In particular, it is shown that by use of partial matching, as prevalent in biological systems, high fault coverage can be achieved with the added advantage of reducing memory requirements. The development of a generic finite-state-machine immunization procedure is discussed that allows any system that can be represented in such a manner to be "immunized" against the occurrence of faulty operation. This is demonstrated by the creation of an immunized decade counter that can detect the presence of faults in real time.**

***Index Terms*—Artificial immune system, error detection, fault tolerance, finite-state machine, immunotronics.**

I. Introduction

FAULT tolerance is becoming ever more important in electronic systems as we rely more and more on their continuous and reliable operation. Electronic system controllers are found in equipment ranging from vending machines through to automotive and spacecraft systems. While the presence of a fault on one system is often just an annoyance, safety-critical systems, such as those on an aircraft, must ensure reliable operation at all times, even in the event of a component failure. Research is continuously looking for improved ways of meeting such requirements.

In recent years, research has taken an interest in how biological organisms manage to survive both individually through self-repair and from one generation to the next through evolution of the species. For example, these challenges have been addressed in hardware fault tolerance through embryonics [43], [46] and evolvable hardware (EHW) [56], [58], respectively.

Our understanding of the natural or, more specifically, human immune system has increased dramatically over the last few decades and has provided a miraculous insight into how the body defends itself from invasion and maintains reliable operation. Through this increased understanding, new techniques inspired by the operation of the human immune system have given rise to improved approaches to computer security [19], virus protection [20], [30], anomaly detection [14], process monitoring [28], robot control [27], and software fault tolerance [62]. The human immune systems provides a distributed fault-tolerant architecture within the body and so suggests a radically different approach to current reliable system design. This paper addresses the challenge of fault-tolerant hardware system design through immunologically motivated techniques.

The requirements for fault-tolerant hardware design are discussed in Section II. Section III extends this into the domain of biological inspiration and introduces evolutionary and developmental approaches to hardware fault tolerance. The development of the immune-inspired hardware fault-tolerance technique or *immunotronics* (immunological electronics) begins in Section IV with a discussion of the similarities and differences between immunology and fault tolerance. Section V demonstrates the immunization cycle and the steps needed to immunize a system for providing fault detection. Results are presented in Section VI and are followed in Section VII with an analysis of the results. Future directions for the work are also discussed. The paper concludes in Section VIII.

II. Fault Tolerance

Over 30 years ago, developments based on the challenge of designing reliable systems from unreliable components resulted in the notion of fault tolerance. Even after years of research, the provision of high-confidence applications and systems is still a very costly process limited only to the most critical of situations [2]. Systems must be protected from a variety of potential faults including transient faults causing a unexpected change in the state of a latch through to permanent faults in the form of a stuck-at-one or stuck-at-zero fault [35], [31]. Real-time fault tolerance can be implemented by the replication of critical systems using *n*-modular redundancy (NMR) [37], [55], error-detecting and correcting codes [51], [18], and self-checking logic circuits [34], [52].

Existing methods have limitations that make alternative fault tolerance architectures attractive in a number of situations, especially when it is impossible to access or replace the defective components in remote systems such as space or hazardous

Manuscript received December 14, 2000; revised June 6, 2001. This work was supported by the Engineering and Physical Sciences Research Council, U.K., and Xilinx Ltd.

The authors are with the Department of Electronics, University of York, Heslington, York YO10 5DD, U.K. (e-mail: amt@ohm.york.ac.uk).

Publisher Item Identifier S 1089-778X(02)06070-8.

environments. The replication of functionally equivalent components is also costly and potentially inefficient if the point of failure is a single transistor.

Reliable operation is achieved using a three-stage process:

1) *detection* of an error or output deviating from the norm;
2) *minimization* or *eradication* of the resulting effects of the fault;
3) *activation* of a suitable recovery procedure. Recovery can take the form of one of two methods: a) backward error recovery can return the system to a previously stored valid state and b) forward error recovery can make selective corrections to the current state until an acceptable state is reached.

Biologically inspired fault tolerance must address these processes also.

A. Error Detection

Detection is perhaps the most critical process within a fault-tolerant architecture. The most sophisticated recovery methods are only as good as the error detection scheme that initiates their operation [1]. It is on this premise that the paper concentrates on the development of a novel error-detection mechanism rather than a complete fault-tolerant architecture.

Hardware error-detection systems can be classified according to their time of application and intervention within the system being protected [1].

1) *Initial testing* can take place to identify faults before the system is put into operation. Error detection mechanisms for initial testing often use automatic test equipment and computer-based techniques with reference systems for comparison [7]. Automatic test pattern generation (ATPG) systems generate specific test patterns that can be applied to systems to diagnose and determine the position of any faults within a system permitting fast repair [7]. Signature analysis is one such ATPG technique that relies on the presence of unique signatures at test points throughout the system [3], [22].
2) *Concurrent* or online detection takes place simultaneously with the normal operation of the protected system. Examples include error-detecting codes [18], [51], NMR [37], and self-checking logic circuits [34], [52]. An important advantage of concurrent fault detection in comparison to other techniques is that recovery procedures can be executed before extensive damage occurs to the system data [1].
3) *Scheduled* or offline detection takes place when normal operation is interrupted either by a user or at regular automated intervals. The techniques applied normally match those used for the initial testing, although as systems become more integrated the ability to gain access to test points becomes more problematic. Techniques and onchip facilities, such as boundary scan [11], have provided industry-standard approaches to system verification and fault detection.
4) *Redundancy testing* is used to verify that any protection mechanisms are themselves fault free, such as ensuring the fault signals are not stuck at a "no fault" state. Concurrent methods of self-checking checkers [34] and scheduled testing of fault signals are suitable here.

The goal of our work lies with the protection of sequential digital systems represented as a finite state machine (FSM). FSMs have been used extensively to model many kinds of systems including sequential circuits, programs, and communications protocols [36]. Many fault-detection systems for FSMs require the generation of unique input–output distinguishing sequences in order to test a circuit and locate the position of a fault [8], [10]. This approach has seen many developments and enhancements to improve the efficiency and error detection capabilities of the test harness [9], [24], [50]. Biologically inspired solutions, using evolutionary algorithms, have also been applied for their abilities to "search" and optimize the test sequence patterns [24]. Another approach has been to model the FSM as an iterative combinational circuit [7], [34]. All these methods of fault detection represent offline approaches. Real-time detection of FSM faults requires the use of concurrent checking hardware. Many examples of this have been investigated [38], [47], [63] using error correcting codes, duplication of functional systems, and extraction of signatures.

The goal of any error detection mechanism is to achieve 100% coverage of all potential faults.

III. Bio-Inspired Systems

Although bio-inspired systems have been present within the electronic and computer science communities for many years [60], it has only been possible in more recent years to realize many of the ideas. The emerging bio-inspired systems can be classified into three distinct domains [54].

1) *Phylogeny* is concerned with the evolution of a species. Bio-inspired fault tolerance research exists in this domain in the form of EHW. Research has investigated the evolution of devices with inherent fault tolerance [56] and also through the evolution of populations of circuits within voting architectures [58].
2) *Ontogeny* is concerned with the development of an individual or organism from a single mother or *zygote* cell through to a multicellular system. The field of *embryonics* has developed fault-tolerant electronic systems based upon a cellular structure, whereby each cell can take on the role of any other cell within the system [40], [41]. Recent work has also shown mathematically that embryonic systems can provide better reliability measures than existing voting systems [45].
3) *Epigenesis* relates to learning within a species. The field of artificial neural networks (ANNs) is the largest research field within this area. Artificial immune systems (AISs) are a rapidly emerging addition to this field, with many similarities and advantages over ANNs discovered [13].

The three domains have developed largely along their own individual paths, although there have been proposals to combine the concepts from more than one domain [4]. The body's own defense mechanisms do not rely on a single solution. The human immune system is a multilayered protection mechanism divided into *innate* and *acquired* immunity. Innate immunity dictates the

TABLE I
COMPARISON OF THE LAYERS OF PROTECTION WITHIN THE HUMAN BODY AND ELECTRONIC HARDWARE

Defence mechanism	Human immune system	Hardware protection
Atomic barrier (physical)	Skin (mechanical) Mucous membranes (trap foreign organisms)	Hardware enclosure (physical/EM protection)
Physiological	Temperature (inhibit growth of pathogens) Acidity (destroy ingested microorganisms)	Environmental settings (temperature control)
Innate immunity	Phagocytes (macrophages) (Kill and digest foreign cells)	N-modular redundancy [1] [37] Embryonics [44] [46] [41]
Acquired immunity	Humoral immunity (bacterial infections) Cell-mediated immunity (viral infections)	Immunotronics [6] [49] [57]

The body incorporates a multilayered defense mechanism. An "electronic" immune system for hardware can be represented in a similar way. Immunotronics adds an additional layer to the hardware immune system.

defense mechanisms with which the body is born and acquired immunity dictates those that are learned as the body develops [33]. Table I compares the defense mechanisms used in nature against those in electronic hardware fault tolerance.

This paper discusses an addition to the epigenetic domain in the form of an AIS for electronic hardware. The defense layer forms an acquired layer of hardware protection created after the system has been developed. We have termed this *immunotronics* (immunological electronics).

IV. IMMUNOLOGICAL TO HARDWARE TRANSITION

Avizienis [2] first noted the similarity in requirements between the immune system and hardware fault tolerance. The immune system suggests alternative ways of implementing the processes of Section II. Five key analogies with the immune system can be made by developing the work of Avizienis.

1) The immune system functions continuously and autonomously. In a mapping to hardware, the analogy is that of error detection and removal without the need for software support.
2) Immune cells are distributed throughout the body to serve all the organs. The hardware equivalent suggests distributed error detection.
3) Immune cells exist in large quantities and with great diversity. Limited diversity is already a common solution to fault-tolerant system design using NMR.
4) The immune system possesses memory. The hardware analogy suggests the training of fault-detection mechanisms to differentiate between fault-free and faulty states.
5) Detection is imperfect within the human immune system. A complementary match between an antigen and an antibody requires only a certain level of specificity [49]. The onset of faults in hardware systems is often due to the impossibility to exhaustively test a system.

Imperfect detection has already been significantly investigated and has seen the development of the *negative selection algorithm* for self–nonself differentiation in computer security and virus protection [20], with remarkable results.

The human immune system provides real-time detection of invaders throughout the body. The creation of a hardware immune system to protect a digital system can therefore be categorized as a concurrent detection mechanism (see Section II-A). Similarities can be seen in the human immune system, where it is possible to view a biological form of redundancy testing with the procedure required to initiate a humoral immune response. B cells only begin proliferation of antibodies when an activation signal is received from helper T cells.

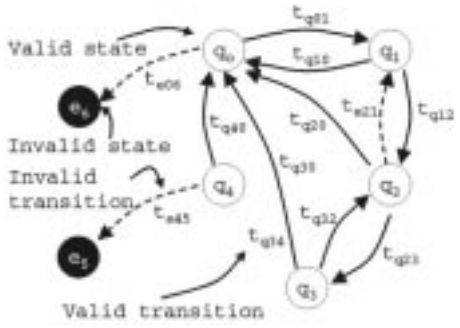


Fig. 1. Definition of valid and invalid states and state transitions for an arbitrary FSM. Normal operation is defined by a valid transition ($t_{q_{pc}}$) between valid states (q_x). Faulty operation is defined by an invalid state (e_x) or invalid transition ($t_{e_{pc}}$). Note how an invalid transition can also occur between valid states (e.g., $t_{e_{21}}$) if the transition is not within the system specification.

A. Hardware Representation

In principle, any hardware system can be represented as an individual or interconnected set of FSMs. FSMs define the acceptable states and transitions between states, as shown in Fig. 1. Under normal and reliable operation or what can be deemed as *self*, only transitions $t_{q_{pc}}$ can occur, where t_q defines a valid transition and $t_{q_{pc}}$ a valid transition from previous state p to current state c. The presence of an invalid state e_c or invalid transition $t_{e_{pc}}$, where e signifies a condition in error, flags a potential problem or the presence of *nonself*. As with many other approaches to state machine fault detection ([8], [10], [39], [50]), concentrating on the transitions between states improves the definition and detection of nonself rather than just treating states as self or nonself. If only states are monitored, then an error can be detected after the transition from a valid state to an invalid one, such as t_{e45} in Fig. 1. If the transition, rather than just the states are analyzed, then transitions between two valid states that are not defined explicitly, such as t_{e21}, are also included in the definition of nonself. Table II shows the practical benefits for a zero to four counter.

B. Feature Mapping

In a similar manner to that of [62], the features and operations of the immune system can be translated into the hardware

TABLE II
INCREMENTAL ZERO TO FOUR COUNTER

Current State	Next State	Valid Next State	Transition	Valid Transition
1	2	Yes	t_{q12}	Yes
4	5	No	t_{e45}	No
2	1	Yes	t_{e21}	No

The benefits of monitoring state transitions. A fault within the state machine forcing the count to change from four to five (row 2), rather than four to zero is detected if just the state is monitored. A fault within the state machine forcing the count to change from two to one (row 3), rather than two to three is detected only if the transition is monitored.

TABLE III
ENTITY FEATURE MAPPING

Immune System	Hardware Fault Tolerance
Self	Valid state transition
Non-self (antigen)	Invalid state transition
Antibody	Error tolerance conditions
Gene used to create antibody	Variables forming tolerance conditions
Antigen presenting cell	Data collection and tolerance condition creation component
Paratope	Invalid state transition tolerance conditions
Epitope	Valid state transition tolerance conditions
Helper T cell	Recovery procedure activator
Memory T cells	Set of tolerance conditions

TABLE IV
PROCESS FEATURE MAPPING

Immune System	Hardware Fault Tolerance
Recognition of self	Recognition of valid transitions
Recognition of nonself	Recognition of invalid transitions
Learning during gestation	Learning of correct transitions
Antibody mediated immunity	Error detection and recovery
Clonal deletion	Isolation of self-recognising tolerance conditions
Inactivation of antigen	Return to normal operation
Life of organism	Operation lifetime of the hardware

domain. Tables III and IV summarize the analogies. The mappings are divided into entities that correspond to physical elements in both the immune system and electronic hardware and processes that correspond to actions or operations that can take place. Tables III and IV show that an immunologically inspired approach based upon the use of FSMs is feasible. The hardware equivalents to the immune system form logical analogies that exist within the domain of state machine operation.

The analogies are developed in Section V to show how these features are put to use.

V. IMMUNIZATION CYCLE

The immunization cycle is developed using a decade counter with the specifications of Table V.

TABLE V
STRUCTURE AND FUNCTION OF THE DECADE COUNTER

Function	0 to 9 counter
States	10
Size (bits)	4
Inputs	Count Enable (CEN) Reset (RST)
Operation	Incremental count (CEN=1, RST=0) Hold (CEN=0, RST=0) Reset (CEN=X, RST=1)

```
Input 1 ...n / previous state / current state
```

Fig. 2. Generic bit-string representation of the data

```
Count enable,reset/previous state/current state

10/0000/0001
10/0001/0010
      ⋮
10/1000/1001
10/1001/0000
```

Fig. 3. Bit-string representation of the decade counter strings, with example sequences.

The data are protected by forming a set of *tolerance conditions* to protect the data set S that forms self. Individual strings $s \in S$ are formed from the combination of user input and previous and current states from the state machine. Fig. 2 shows one such form of the self strings s.

For the decade counter defined in Table V, the structure of s is that given in Fig. 3. With the organization of s as defined in Fig. 3, the decade counter possesses 40 strings that define self and hence valid operation.

The immunization cycle is divided into four phases that fit into a typical hardware development cycle [55]. Hence, the complete cycle becomes requirements, specification, design, implementation, testing, *data gathering*, *tolerance condition generation* in service operation, *error detection*, and *fault removal*. The new phases are italicized and are described in some detail in the following sections.

A. Data Gathering

The goal of the data gathering stage is to create a data set S that represents a complete or substantial percentage of all possible valid state transitions within normal operation of the FSM. The test bench setup consists of a Xilinx Virtex XCV300 field-programmable gate array (FPGA) [26] situated on a Virtual

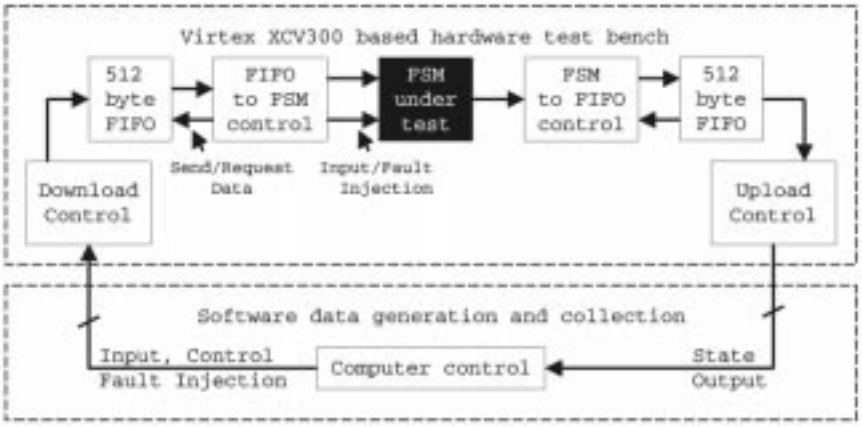


Fig. 4. Complete hardware and software testbench for self-data generation, collection, and processing. Inputs are injected into the state machine under test and the resulting system state (or output) collected and returned to the computer. Data is processed to create the set S of self strings ready to undergo the immunization process.

Workbench development board from Virtual Computer Corporation [12]. The hardware configuration permits any FSM design to be inserted into a generic test bench on the FPGA and undergo an immunization cycle.

Self data can be collected from the hardware in one of two ways: 1) from a set of predefined inputs and operations used for the previous testing phase 2) or through the injection of random inputs. The first is analogous to conventional testing techniques. Many of the approaches are discussed and cited in Section II. The second approach using random information is somewhat more analogous to the immune system in the sense that self (cells) are collected randomly, ready to be presented to immature T cells during the centralized negative selection process. In both situations, it is assumed that at the time of data generation the FSM under analysis is fault-free. The random approach is adopted initially for its simplicity and likening to the immune system. If a complete description of the FSM is already provided, then the data are already available and this stage, in some instances, is not required. The decade counter contains ten valid states and two inputs, giving a total of $(2^2 \times 10) = 40$ self strings to be collected. Experimentation showed this to be possible within 10 s after an average 200 000 random input combinations. This confirms that, although the adoption of an immune-inspired method of data collection can realize the desired operation, a functional approach or a function combined with random data generation can improve the rate at which an equivalent coverage is reached [29].

The combined hardware/software testbench is shown in Fig. 4. The software component provides random, cyclic, or user-defined (specific-data defined or generated elsewhere) inputs to the state machine under test. The FSM is inserted into the test harness in the development hardware. Data are gathered from the state machine and returned to the computer whereby the inputs and previous and current state (or output) data are concatenated to create individual strings s. The strings are then filtered to create the set of unique self strings S.

B. Tolerance Condition Generation

The negative selection algorithm was developed by Forrest *et al.* [20] for the detection of viruses within computer systems and network intrusion. It applies techniques inspired by the operation of the human immune system for pattern detection. The negative selection algorithm was developed from a theoretical analysis of matching and binding probabilities within the immune system from work carried out by Percus *et al.* [48]. Existing fault tolerance architectures, such as NMR [1] and embryonics [42], work by checking constantly for the presence of valid operation. In contrast, the negative selection algorithm works by checking constantly for the presence of invalid operation. The negative selection algorithm has already proven very successful, with further advances forming a complete immune system for computers under the name of ARTIS [23]. The algorithm is based upon the method of selecting a set of strings R of length l from a randomly generated original set of data R_o. Each string $r \in R$ fails to match any of the self strings $s \in S$, also of length l, in at least c contiguous positions. Any strings that match in at least c contiguous positions are deleted. The mature set of tolerance conditions R are generated from an initial randomly generated set R_o corresponding to immature tolerance conditions, which undergo a negative selection process. The set of self strings S correspond to the set of self strings defined in Fig. 3. Fig. 5, adopted from [16], summarizes the operation.

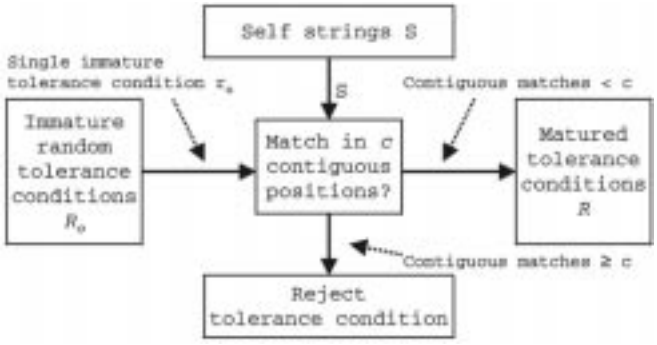


Fig. 5. Negative selection algorithm is used to create a set of tolerance conditions R that fail to match any self strings s in at least c contiguous positions.

Using the negative selection algorithm from [20], the probability P_m that two random strings match (here between a self string $s \in S$ and an individual randomly generated immature

tolerance condition r_o ($r_o \in R_o$) in at least c contiguous positions is given by

$$P_m \approx m^{-c}[(l-c)(m-1)/m+1] \tag{1}$$

where $m^{-c} \ll 1$ and m is the number of alphabet symbols (two for a binary FSM). When this does not hold, the exact formula described in [59] can be applied.

If the number of tolerance conditions N_r is limited, the theoretical probability that the system fails to detect nonself is given by

$$P_{f_t} = (1-P_m)^{N_r}. \tag{2}$$

It is, therefore, possible to trade off the fault-detecting capabilities of the hardware immune system against the storage requirements. The hardware immune system will be a valuable protection mechanism in remote environments such as space-borne applications where the operation of a system is changed or reconfigured remotely. It is envisaged that reprogramming of a remote system or controller is carried out and then the hardware immune system can immunize the updated device automatically. The use of programmable hardware for such systems means that only a limited hardware space may be available, depending on the configuration of the new hardware. The ability to control N_r is of great use. From this, it is possible to predict the initial number of immature tolerance conditions N_{r_o} needed to generate N_r matured tolerance conditions

$$N_{r_o} = \frac{N_r}{(1-P_m)^{N_r}}. \tag{3}$$

Equations (1)–(3) are applied in Section VI to assess the detection capabilities of the hardware immune system for the decade counter.

The random generation of tolerance conditions goes some way to achieving the desired goal of covering and protecting against the occurrence of nonself strings. One shortfall of this approach, in terms of storage space, is the overlap in detectors that can occur creating a less efficient protection against nonself strings (although this could be treated potentially as creating redundancy within the tolerance conditions). D'haeseleer developed a method of improving the coverage of the string space through the development of the *greedy detector generating algorithm* [15], [17]. The greedy detector generator achieves a better coverage of nonself strings by not generating the detectors randomly, but instead extracting those tolerance conditions that match the most nonself strings first and then extracting others and placing them as far apart as possible. As discussed in [17], this has the benefits of either reducing the probability of failing to detect a nonself string P_f for a given number of tolerance conditions or, alternatively, reducing the number of tolerance conditions N_r for a fixed failure probability P_f. In [15], D'haeseleer discusses the operation of the greedy detector generating algorithm in detail. Both the random and greedy detector generator algorithm are implemented as software components of the complete hardware immunization procedure to permit comparison of P_f and N_r.

Analyzing P_f for variations in c and N_r enables the match length c to be chosen to give the best coverage of nonself strings for the available storage space. Such data for the decade counter are provided in Section VI.

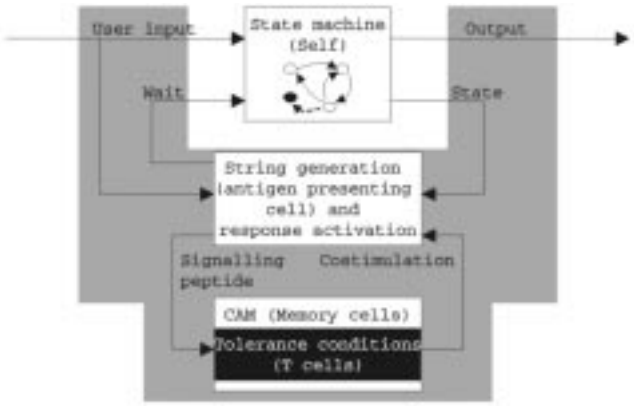


Fig. 6. Architecture of the hardware immune system. Hardware immune system acts as a wrapper to the state machine under protection. User inputs, state, and/or output data are gathered from the state machine and used to create a search string for passing to the CAM [32]. Partial-matching CAM searches all memory locations for a match in c contiguous positions and returns a result in $l-c+1$ clock cycles.

C. In-Service Operation—Configuration, Architecture, and Fault Detection

It is now demonstrated how the hardware immune system is incorporated into the complete system architecture whereby it acts as a "wrapper" to the state machine being protected or *immunized*. The detection process is significantly less complex than the previous stages and permits a complete hardware implementation with no final requirement for software control.

1) Architecture of the Hardware Immune System: The operational state machine and combined immune system comprise three main components and form the complete system, as shown in Fig. 6.

The state machine under protection is the system being protected. Under normal operation, only self strings are present. The presence of a fault creates a nonself state, analogous to the presence of an antigen.

The string generation component gathers the user inputs and system state (or output) from the state machine, combining it with the previous system state (or output) to create a search string for presentation to the immune system memory.

The partial-matching content-addressable memory (CAM) stores the tolerance conditions and returns a positive result if c contiguous bits out of l match the search string. A CAM [32] permits parallel searching of all memory locations in a single clock cycle. It achieves this by accessing the device using the data, rather than by an address. The data are presented to the CAM and a found or not-found signal returned in addition to the address where the data were found. Fig. 7 shows the structure of a typical CAM device.

Partial matching in c contiguous bit locations is provided for by modifying the generic CAM architecture of Fig. 7 to provide further bit masking inputs. Individual bits can then be set to a *match* or *don't care* state.

For development purposes, the CAM is configured as a 64-bit-wide (32-bit data and 32-bit mask) 128-word deep device. Although the CAM is a fixed width, shorter tolerance

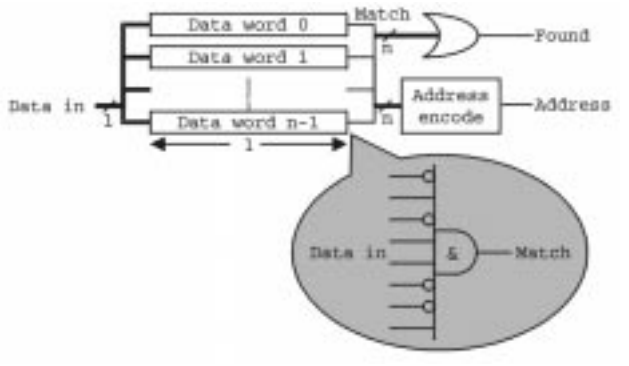


Fig. 7. Basic structure of a CAM. Parallel searching of the memory locations by data (rather than address) creates a search time that is independent of the depth n of the CAM.

conditions can be stored by permanently fixing the masking bits to a *don't care* state. The mask size and masking pattern are also programmable by the user. Variation in the masking bits allows a c-sized window to be moved across the tolerance conditions. A search for an individual string requires a single clock cycle. With a variable mask for detection of length c, the required number of clock cycles is increased to $l - c + 1$ (for the decade counter with $l = 10$ and a match length $c = 7$ 4 clock cycles are neeed). The hardware immune system ensures that a match is still found within a single state machine clock cycle through the use of a memory read clock multiplied by a similar ratio.

The use of a partial matching CAM for storage of the tolerance conditions is very apparent in the operation of the natural immune system. In a reversal of roles, models of the immune system have been used on several occasions to develop novel forms of CAMs [25], [21].

2) Error Detection: The immunized state machine is monitored at every change of state and the gathered data sent to the tolerance condition memory and searched. A match is deemed to have occurred if any tolerance condition matches the generated string in c contiguous positions. The faster memory read clock driving the CAM permits the result of search to be returned to the "string generation and response activation" component of Fig. 6 before the current internal state of the state machine propagates to the output on the next clock cycle. If the internal states of the hardware are always monitored, rather than just the outputs, then it is possible to detect a fault before the effects have propagated to the output.

D. Fault Removal

The goal of this paper to date has been to investigate immunologically inspired approaches to fault detection. Future work aims to develop these methods to incorporate fault removal techniques. Potential methods of achieving this are now discussed.

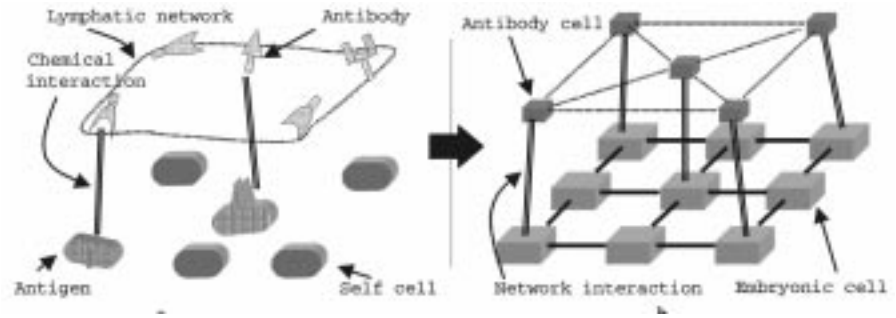
1) Classical Architectures: In the human immune system, invaders are destroyed to prevent a detrimental effect to the body. A simple approach would be to replicate the protected state machine and switch to a spare if the first is detected as faulty, in essence, creating a form of NMR as discussed in Section II. In this form, the hardware immune system would essentially be a novel form of voting architecture. Unless a disastrous failure occurs, it is not ideal to disable a large hardware component. If a self string is detected accidentally as nonself through incomplete coverage of self strings or the presence of a fault within the tolerance condition storage, then a fault-free state machine may be deactivated completely. Again, a similar problem is created for transient errors and the deactivation of a system that operates normally to specification. A similar problem is encountered for the presence of transient errors.

2) Immunologically Inspired Architectures: As first steps toward enhancing the architecture and immunization cycle, discussed in Sections V-AC, the detection of a nonself string may signal the user to request the next action. If the condition is deemed valid, then the self string can be added to a list of acceptable, but immune activating strings. The possibility of providing two sets of tolerance conditions has been discussed in [5] so that a set of potential recovery states corresponding to self string can also be stored. It may then be possible to automatically correct the faulty output through an output multiplexer selecting either the normal output or the immune-activated response. The continuous usage of the state machine, even after a fault has first been detected, means that transient errors do not result in the deactivation of the complete system.

Analyzing the operation of the immune system further, during the humoral response, antibodies bind to antigens to prevent their binding to further cells in the body. The phenomenal cellular redundancy in the body enables normal operation to persist when cells are neutralized and later destroyed due to infection. In an FSM architecture, a similar technique could be used through the use of spare states or latch bits within the hardware. The detection of a fault would then cause the state machine to be reconfigured to ensure the faulty state was circumvented and a spare state used. To implement this would require some knowledge of what the next valid operation should have been, either through inclusion of a complementary set of tolerance conditions failing to match nonself or through prior programming of the spare states to correspond to valid ones—essentially having two overlapping state machines operating in tandem.

3) Total Biologically Inspired Architectures: Two other biologically inspired approaches to fault detection and tolerance were introduced in Section I in the form of embryonics and EHW. The integration of immunotronics with embryonics has been investigated in [4] to create a learning cellular architecture of functional and antibody cells. Embryonics is based upon the development of multicellular organisms. When biological multicellular organisms develop, cells differentiate according to "instructions" stored in their DNA. Different parts of the DNA are interpreted depending on the position of the cell within the embryo. Before differentiation, cells are (theoretically) able to take over any function within the body because each one possesses a copy of the DNA. Correspondingly, every electronic cell in an embryonic array stores not only its own configuration register, but also those of its neighbors. To differentiate, every cell selects a configuration register according to its position within the array. Position is determined by a set of coordinates that is calculated from the coordinates of the nearest neighbors.

Every embryonic cell currently performs self-checking continuously. If a failure is detected, the faulty cell issues a status signal that eliminates the cell. The surviving cells recalculate



(a) (b)

Fig. 8. (a) Antigen-antibody interactions within the human immune system. (b) Mapping of lymphatic interactions to an integrated immunotronic-embryonic multilayered fault-tolerant architecture.

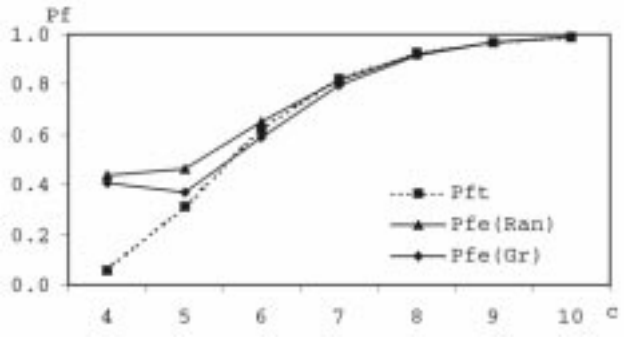


Fig. 9. Theoretical (P_{f_t}) and experimental (P_{f_e}) variations in the probability of failing to detect the presence of a fault for $4 \leq c \leq 10$, $N_r = 10$. Experimental plots show the random detector generator $P_{f_e}(\mathrm{Ran})$ and the greedy detector generator $P_{f_e}(\mathrm{Gr})$ algorithms applied to the generation of tolerance conditions.

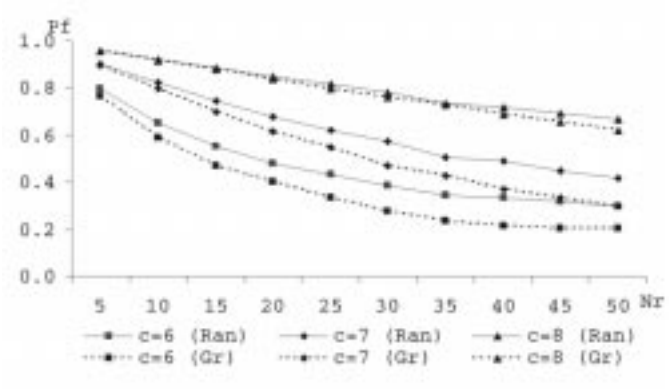


Fig. 10. Comparison of random and greedy detector generator algorithms for variation in failure probability P_f against the number of tolerance conditions N_r for match length $6 \leq c \leq 8$.

their coordinates and select a new configuration register. By doing so, every cell performs a new function. A detailed description of the embryonics architecture can be found in [41] and [46]. The integration of a cellular hardware immune system within the architecture, as shown in Fig. 8, removes the need for self checking from each embryonic cell. This simplifies the cell architecture and removes the need for duplication of functional units within each cell. Removing the checking circuit also removes another place for potential faults to manifest. The hardware immune system layer contains cells intertwined within the embryonic cells, with each immune or *antibody* cell continuously monitoring its neighboring embryonic cells for faults. Interaction between neighboring antibody cells also allows for error detection within each antibody cell due to the repeated checking of every embryonic cell by more than one antibody cell. If the results from antibody cells that have checked the operation of the same embryonic cell differ, then the fault antibody cell is deactivated and the array reconfigured.

VI. Results

Section VI presents the results from the immunization of the decade counter example discussed through the development of the hardware immune system in Section V. Fig. 9 shows the mean results over 100 repetitions for match lengths $4 \leq c \leq 10$, with N_r fixed to ten tolerance conditions using both the exhaustive random and greedy detector generator algorithms. Random generation is implemented through the use of a L'Ecuyer with Bayes–Durham shuffle algorithm [53] to ensure that sufficiently diverse random values are generated over the repetitions. The number of self strings N_s is fixed at 40 for the counter. The theoretical prediction is provided for comparison.

Fig. 9 shows that for a fixed number of final tolerance conditions, the failure probability P_f increases as c increases as individual tolerance conditions, in general, are matching progressively fewer and fewer nonself conditions. For $c \geq 6$, the failure probability P_{f_e} for both generators agrees with the theoretical prediction P_{f_t}. Below this value, the theoretical predictions do not correspond well with the results. The theoretical approximation of (1) still holds with the low values of c, as can be confirmed by calculation using the exact formula for c contiguous locations in [59]. The increased failure probability at the lower values of c is due to the limited number of unique tolerance conditions that can actually be generated that match only nonself (see Table VI). In addition to this, an analysis of the matured tolerance conditions confirms the presence of similarities in the tolerance conditions generated by the exhaustive random generator. For lower match lengths, the greedy detector generator improves the results by extracting the best tolerance conditions first creating a lower failure probability. Analyzing the results from both the random and greedy detector generators further, Fig. 10 shows the effects of failure probability P_f for $c = 6, 7, 8$ over a range of five to 50 tolerance conditions. As with Fig. 9, it is again possible to see the improvement through a reduction in failure probability P_f. As the match length c increases, the

TABLE VI
OPTIMAL NUMBER OF TOLERANCE CONDITIONS REQUIRED TO COVER ALL DETECTABLE NONSELF STRINGS FOR THE DECADE COUNTER

Match length	Optimal detector set size	Nonself strings detectable
1	-	-
2	-	-
3	-	-
4	6	584 (59%)
5	14	664 (67%)
6	42	784 (80%)
7	103	913 (93%)
8	222	932 (95%)
9	472	954 (97%)
10	984	984 (100%)

No unique detectors are possible for match lengths $c \leq 3$.

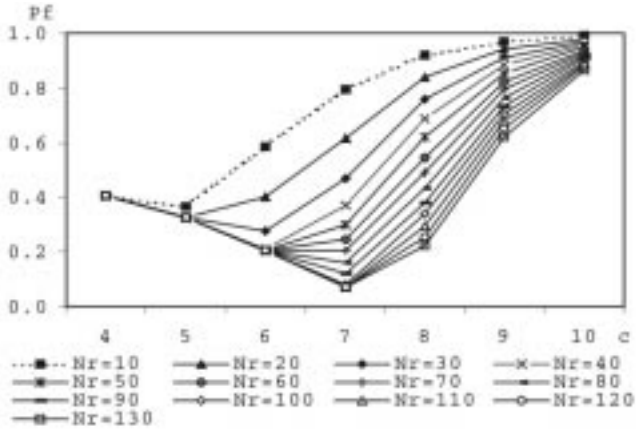


Fig. 11. Failure probability P_f against match length c for $10 \leq N_r \leq 130$, using greedy detector generator.

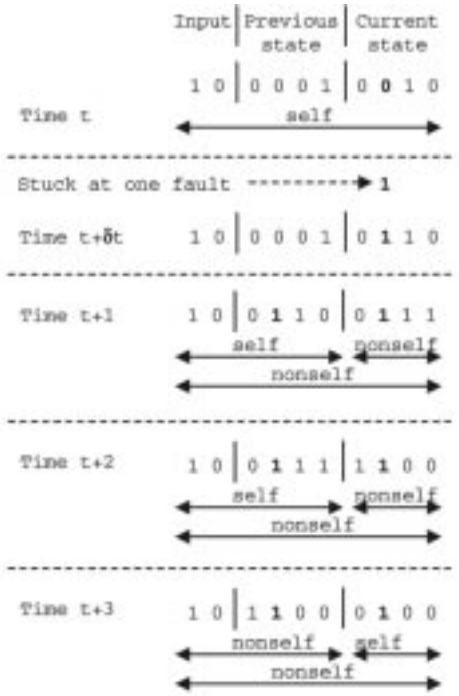


Fig. 12. Propagation of a stuck at one fault through a data string. To prevent a faulty system output requires the fault to be detected at time $T \leq t + 1$.

improved nature of tolerance condition extraction reduces progressively as individual tolerance conditions match fewer and fewer nonself strings.

Having demonstrated the improved coverage ability of the greedy detector generator, the random generator is no longer used for generation of tolerance conditions. The following results apply just the greedy detector generator for production of tolerance conditions.

Table VI shows the optimal number of tolerance conditions required for each match length to cover all *detectable* nonself strings. The term detectable has been added as many nonself strings may be undetectable for a given match length due to strong similarities with self strings. A tolerance condition generated to detect these nonself strings would also detect a self string making the detection process invalid. This is discussed further in Section VII. With a match length of $c = 4$, only six tolerance conditions are needed to detect about 60% of all the possible (984) faults. As would be expected with a match length $c = 10$, every tolerance condition detects a unique nonself string providing 100% coverage of all nonself strings.

The ideal match length can be determined by first setting a limit on the available storage space. To complement the hardware immune system architecture of Fig. 6, the upper limit is set to 128 10-bit words. Fig. 11 shows the failure probability P_f against the match length c for a variation in tolerance conditions from $10 \leq N_r \leq 130$. As the number of tolerance conditions increases, the optimal (in terms of failure probability for a given number of tolerance conditions) match length c can be chosen from the minimum point of each plot. For $10 \leq N_r \leq 20$, $c = 5$ provides the best coverage of nonself; for $30 \leq N_r \leq 70$, $c = 6$ should be chosen. For $N_r \geq 80$, $c = 7$ provides the best coverage up to and including the limit imposed of 128 tolerance conditions. Although not shown in Fig. 11, a match length $c = 8$ supersedes the coverage provided by $c = 7$ with almost a twofold increase in tolerance conditions at $N_r = 220$.

Fig. 11 demonstrates that a match length $c = 7$ (with $N_r = 103$ from Table VI) is the ideal match length for the decade counter. The previous plots have analyzed the *total* detectable faults that may occur throughout the operation of the state machine. It is also useful to consider what fraction of the faults will be detected within a single clock cycle, i.e., before the effect of the fault propagates to the output of the system. A fault will first result in an error in the "current state" substring within each string s, which will then propagate through the state machine as shown in Fig. 12 producing in this particular case a stuck-at-one fault.

The matching assessment is now modified to analyze the failure probability P_f for the detection of a fault within a single cycle. Strings are defined to be a single-cycle detectable string if both the input and previous state bits correspond to a self string and the current state bits correspond to a nonself string. For the decade counter, the total number of single-cycle detectable strings is 600, i.e., 600 out of the total 984 nonself strings result from the propagation from a self to a nonself state (rather than from a nonself to another nonself state). Fig. 13 compares the fraction of single cycle detectable faults to all the detectable faults for the chosen match length $c = 7$.

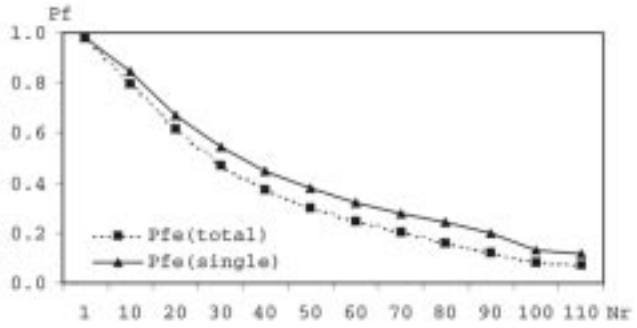


Fig. 13. Failure probability P_{f_e} against the number of tolerance conditions N_r for total fault detection $P_{f_e}(\text{total})$ and single cycle detectable $P_{f_e}(\text{single})$ faults. Match length $c = 7$.

Fig. 13 shows that an almost constant 7% increase in failure probability occurs when single-cycle detections are considered. Further analysis of these data indicate that as the match length c increases, the percentage difference between total and single cycle detectable faults decreases. Given $c = 8$, a 2% difference is observed.

VII. Analysis

The error-detection procedure implemented relies on the ability to extract the internal state bits from the FSM being immunized. Doing so permits a high level of error detection and enables any future recovery procedure activated before the effects of the fault propagate to the system output. If the state machine is itself an embedded device then direct access to the internal state may be impossible. Fault detection is still possible in such instances, although not before the first fault occurs at the output, by monitoring the outputs rather than the internal states.

The presence of undetectable nonself strings or *holes* [15] is the result of self strings matching over $c - 1$ contiguous bits and consequently inducing other strings that are unable to be detected because any tolerance conditions matching the induced strings would also match the self strings. The cause and analysis of this effect has been investigated by D'haeseleer in [15] and [16]. Within the hardware immune system, preliminary investigations have shown that two factors, both creating the same symptoms, can cause variation in the number of undetectable nonself strings (holes) for a particular match length.

1) *State Assignment:* Conventional digital-design techniques concentrate on minimizing the next-state logic (and, therefore, cost) within the state machine architecture. One approach to doing this is to ensure only a single bit changes on each transition [61]. This is one step toward an efficient use of resources. The effect of this on the immunization procedure is to make the presence of holes even more apparent. Changing the state assignment would reduce the number of holes.
2) *Self-String Structure:* Closely related to the state assignment, the self strings are formed from the state data and so themselves change little on each new detection cycle. Reorganization of the self strings can vary the number of holes.

Two methods of reducing the number of holes are the following.

1) *Optimize the Self-String Structure:* Preliminary analysis has shown that varying the structure of the self strings or state assignment varies the number of holes within the nonself strings, as discussed previously. Optimizing the state assignment, while beneficial to the hardware immune system, can have detrimental effects on both the speed and size of the FSM. Variation of the structure of self strings is applied easily by concatenating the user inputs and previous and current state data together in different ways. This has been investigated for the decade counter for the match length $c = 7$: mixing the bits together with inputs $(i_0 \ldots i_{n-1})$, previous $(p_0 \ldots p_{n-1})$, and current $(c_0 \ldots c_{n-1})$ state bits forming self strings in the form $p_0c_0p_1c_1i_0i_1p_2c_2p_3c_3$ increases the percentage of *total* detectable nonself strings from 93% to 98% and the number of *single-cycle* detectable nonself strings from 88% to 96% using 104 tolerance conditions compared to the original 103.
2) *Vary the Match Length:* Varying the match length is an ideal way to remove the presence of holes completely. Any undetectable nonself strings can then be covered by an increased match length. Although initially a simple solution, implementing this in hardware would entail multiple CAMs and increasing the size of the system significantly. A beneficial side effect of doing so would be that replication would provide redundancy within the memories as longer tolerance conditions that are designed to protect nonself string using a lower match length would also overlap the detection other nonself strings.

VIII. Conclusion

This paper has demonstrated a novel approach to FSM error detection using probabilistic negative selection methods inspired by the human immune system. The acquired immune response in the human immune system is learned through a process of centralized maturation to create a collection of antibodies able to detect the invasion of nonself into the body. This analogy has been applied to the field of electronic hardware error detection to provide FSMs using a generic immunization procedure.

A immunization cycle has been developed that integrates with a typical hardware development cycle to permit any finite-state-based system to be immunized in a methodical way. The system is analyzed, self strings gathered, and tolerance conditions generated. The match length c is choosen to make optimum use of the available tolerance condition storage space. This is carried out currently by hand, although future automation would be straightforward. The architecture also permits a tradeoff between the storage space and failure probability, ensuring that the most effective tolerance conditions are always stored first. The implementation of a CAM ensures that an error can be detected in a single clock cycle, providing the ability to activate any future recovery procedures before the resulting error propagates to create a faulty output.

The hardware immune system currently goes some way to achieving three of the five original analogies between the human immune system and hardware fault tolerance discussed in Section IV.

1) The operational hardware immune system functions continuously and autonomously and is designed to allow full implementation in hardware. This is facilitated by the simple (compared to tolerance condition generation) search and detection process created through the use of a CAM.
2) The immunotronic error detection mechanisms are trained to differentiate between faulty and fault free transitions. The hardware immune system possesses memory to store the set of tolerance conditions that perform this operation.
3) Detection of invalid conditions is imperfect.

The most notable exception in this work from the key analogies is the omittance of any distributed forms of fault detection. A single FSM is not the ideal architecture for a distributed approach. If a system were built upon a set of several interconnecting FSMs, then a distributed approach could certainly be considered to protect the system against faults. Each state machine could then possess a set of individual tolerance conditions. The ideal solution, however, will be provided by the implementation of an integrated immunotronic-embryonic architecture.

ACKNOWLEDGMENT

The authors would also like to thank the anonymous referees for their helpful and incisive comments.

REFERENCES

[1] A. Avizienis, "Fault-tolerance: The survival attribute of digital systems," *Proc. IEEE*, vol. 66, pp. 1109–1125, Oct. 1978.
[2] ——, "Toward systematic design of fault-tolerant systems," *IEEE Computer*, vol. 30, pp. 51–58, Apr. 1997.
[3] D. Bahr, "Understanding signature analysis," *Electron. Test*, vol. 5, no. 11, p. 28, 33, Nov. 1982.
[4] D. W. Bradley, C. Ortega-Sánchez, and A. M. Tyrrell, "Embryonics + Immunotronics: A bio-inspired approach to fault tolerance," in *Proc. 2nd NASA/DoD Workshop on Evolvable Hardware*, July 2000, pp. 215–223.
[5] D. W. Bradley and A. M. Tyrrell, "Hardware fault tolerance: An immunological solution," in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics*, vol. 1, 2000, pp. 107–112.
[6] ——, "Immunotronics: Hardware fault tolerance inspired by the immune system," in *Proceedings of the 3rd International Conference on Evolvable Systems: From Biology to Hardware (ICES2000)* , J. Miller, A. Thompson, P. Thomson, and T. C. Fogarty, Eds: Springer-Verlag, Apr. 2000, vol. 1801, Lecture Notes in Computer Science, pp. 11–20.
[7] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems*. Rockville, MD: Computer Science, 1976.
[8] G. Buonanno, F. Fumi, D. Sciuto, and F. Lombardi *et al.*, "FsmTest: Functional test generator for sequential circuits," *Integr. VLSI J.*, vol. 20, no. 3, pp. 303–325, 1996.
[9] G. Buonanno *et al.*, "How an evolving fault model improves the behavioral test generation," in *Proc. IEEE Great Lakes Symp. VLSI*, Mar. 1997, pp. 124–129.
[10] K. T. Cheng and J. Y. Jou, "Functional test generation for finite state machines," in *Proc. IEEE Int. Test Conf.*, Sept. 1990, pp. 162–168.
[11] C. Collins, "JTAG boundary-scan for low cost system testing," *Xcell*, no. 31, pp. 34–35, 1999.
[12] The Virtual Workbench (1999). [Online]. Available: http://www.vcc.com/VW.html
[13] D. Dasgupta, "Artifical neural networks and artificial immune systems: Similarities and differences," in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics*, Oct. 1997, pp. 363–374.
[14] D. Dasgupta and S. Forrest, "An anomaly detection algorithm inspired by the immune system," in *Artificial Immune Systems and Their Applications*, D. Dasgupta, Ed. Berlin, Germany: Springer-Verlag, 1998, pp. 262–277.
[15] P. D'haeseleer, "Further efficient algorithms for generating antibody strings," Dept. Comput. Sci., Univ. New Mexico, Tech. Rep. CS95-3, 1995.
[16] ——, "An immunological approach to change detection: Theoretical results," in *Proc. 9th IEEE Computer Security Foundations Workshop*, County Kerry, Ireland, June 1996.
[17] P. D'haeseleer, S. Forrest, and P. Helman, "An immunological approach to change detection: Algorithms, analysis and implications," in *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1996, pp. 110–119.
[18] S. Dutt and N. R. Mahapatra, "Node-covering, error-correcting codes and multiprocessors with very high average fault tolerance," *IEEE Trans. Comput.*, vol. 46, pp. 997–1014, Sept. 1997.
[19] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for Unix processes," in *Proce. 1996 IEEE Symp. Computer Security and Privacy*, May 1996, pp. 120–128.
[20] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1994, pp. 202–212.
[21] C. J. Gibert and T. W. Routen, "Associative memory in an immune-based system," in *Proc. 12th Int. Conf. Artificial Intelligence*, July 1994, pp. 852–857.
[22] S. Z. Hassan, "Signature testing of sequential machines," *IEEE Trans. Comput.*, vol. C-33, pp. 762–764, Aug. 1984.
[23] S. A. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," *Evol. Comput.*, vol. 8, no. 4, pp. 443–473, 2000.
[24] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Automatic test generation using genetically-engineered distinguishing sequences," in *Proc. IEEE VLSI Test Symp.*, Apr. 1996, pp. 216–223.
[25] J. E. Hunt and D. E. Cooke, "Learning using an artificial immune system," *J. Network Comput. Applicat.*, vol. 19, no. 2, pp. 189–212, 1996.
[26] Virtex Data Sheet (1999). [Online]. Available: http://www.xilinx.com/partinfo/virtex.pdf
[27] A. Ishiguro, T. Kondo, Y. Watanabe, and Y. Uchikawa *et al.*, "Immunoid: An immunological approach to decentralized behavior arbitration of autonomous mobile robots," in *Parallel Problem Solving from Nature IV*, H. M. Voight *et al.*, Eds: Springer-Verlag, 1996, vol. 1141, Lecture Notes in Computer Science, pp. 666–675.
[28] A. Ishiguro, Y. Watanabe, and Y. Uchikawa, "Fault diagnosis of plant systems using immune networks," in *Proc. 1994 IEEE Int. Conf. Multisensor Fusion and Integration for Intelligent Systems*, Oct. 1994, pp. 34–42.
[29] M. Karam and G. Saucier, "Functional versus random test generation for sequential machines," *J. Electron. Testing Theory Applicat.*, vol. 4, pp. 33–41, 1993.
[30] J. O. Kephart, "A biologically inspired immune system for computers," in *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, R. A. Brooks and P. Maes, Eds. Cambridge, MA: MIT Press, 1994, pp. 130–139.
[31] Test Technology Overview, R. Klenke. (1998). [Online]. Available: http://www.cedcc.psu.edu/ee497f/rassp_43/
[32] T. Kohonen, *Content-Addressable Memories*, 2nd ed. Berlin, Germany: Springer-Verlag, 1987.
[33] J. Kuby, *Immunology*, 3rd ed. San Francisco, CA: Freeman, 1997.
[34] P. K. Lala, *Digital Circuit Testing and Testability*. New York: Academic, 1997.
[35] ——, *Fault Tolerance and Fault Testable Hardware*. Englewood Cliffs, NJ: Prentice-Hall, 1985.
[36] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines—A survey," *Proc. IEEE*, vol. 84, pp. 1090–1126, Aug. 1996.
[37] P. A. Lee and T. Anderson, *Fault Tolerance Principles and Practice, Volume 3 of Dependable Computing and Fault-Tolerance Systems*, 2nd ed. Berlin, Germany: Springer-Verlag, 1990.
[38] R. Leveugle and G. Saucier, "Optimized synthesis of concurrently checked controllers," *IEEE Trans. Comput.*, vol. 39, pp. 419–425, Apr. 1990.

[39] H. Ma and S. Devadas, "Test generation for sequential finite state machines," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1987, pp. 288–291.
[40] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, and S. Durand, "Embryonics: A new family of coarse-grained field-programmable gate array with self-repair and self-reproducing properties," in *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, E. Sanchez and M. Tomassini, Eds: Springer-Verlag, 1996, vol. 1062, Lecture Notes in Computer Science, pp. 197–220.
[41] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti, "Toward robust integrated circuits: The embryonics approach," *Proc. IEEE*, vol. 88, pp. 516–541, Apr. 2000.
[42] D. Mange, A. Stauffer, and G. Tempesti, "Embryonics: A microscopic view of the molecular architecture," in *Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware (ICES98)*, M. Sipper, D. Mange, and A. Pérez-Uribe, Eds: Springer-Verlag, Sept. 1998, vol. 1478, Lecture Notes in Computer Science, pp. 185–195.
[43] P. Marchal, P. Nussbaum, C. Piguest, S. Durand, D. Mange, E. Sanchez, A. Stauffer, and G. Tempesti, "Embryonics: The birth of synthetic life," in *Toward Evolvable Hardware*, E. Sanchez and M. Tomassini, Eds: Springer-Verlag, 1996, vol. 1062, Lecture Notes in Computer Science, pp. 166–198.
[44] C. Ortega-Sánchez and A. M. Tyrrell, "MUXTREE Revisited: Embryonics as a reconfiguration strategy in fault-tolerant processor arrays," in *Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware (ICES98)*, M. Sipper, D. Mange, and A. Pérez-Uribe, Eds. Berlin, Germany: Springer-Verlag, Sept. 1998, vol. 1478, Lecture Notes in Computer Science.
[45] ——, "Reliability analysis in self-repairing embryonic systems," presented at the Proceedings of the 1st NASA/DOD Workshop on Evolvable Hardware, July 1999.
[46] C. Ortega-Sánchez, D. Mange, S. Smith, and A. Tyrrell, "Embryonics: A bio-inspired cellular architecture with fault-tolerant properties," *Genetic Program. Evolvable Mach.*, vol. 1, no. 3, pp. 187–215, July 2000.
[47] F. Özgüner, "Design of totally self-checking asynchronous and synchronous sequential machines," in *Proc. 7th Int. Symp. Fault Tolerant Computing*, June 1977, pp. 124–129.
[48] J. K. Percus, O. E. Percus, and A. S. Perelson, "Probability of self-nonself discrimination," in *Theoretical and Experimental Insights Into Immunology*, A. S. Perelson and G. Weisbuch, Eds. Berlin, Germany: Springer-Verlag, 1992, pp. 63–70.
[49] ——, "Predicting the size of the T-cell receptor and antibody combining region from consideration of efficient self-nonself discrimination," in *Proceedings of the National Academy of Science*, vol. 90. London, U.K., 1993, pp. 1691–1695.
[50] I. Pomeranz and S. M. Reddy, "On achieving a complete fault coverage for sequential machines using the transition fault model," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 341–346, June 1991.
[51] D. K. Pradham, Ed., *Fault Tolerant Computing: Theory and Techniques—Volume 1*: Prentice-Hall, 1986.
[52] ——, *Fault Tolerant Computing: Theory and Techniques—Volume 2*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
[53] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1992.
[54] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic and epigenetic view of bio-inspired hardware systems," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 83–97, Apr. 1997.
[55] N. Storey, *Safety-Critical Computer Systems*: Addison-Wesley, 1996.
[56] A. Thomspon, "Evolutionary techniques for fault tolerance," in *Proc. UKACC Int. Conf. Control*, 1996, pp. 693–698.
[57] A. M. Tyrrell, "Computer know Thy self!: A biological way to look af fault tolerance," in *Proc. 2nd EuroMicro/IEEE Workshop Dependable Computing Systems*, Sept. 1999, pp. 129–135.
[58] A. M. Tyrrell, G. S. Hollingworth, and S. L. Smith, "Evolutionary strategies and intrinsic fault tolerance," in *Proc. 3rd NASA/DoD Workshop Evolvable Hardware*, July 2001, pp. 98–106.
[59] J. V. Uspensky, *Introduction to Mathematical Probability*. New York: McGraw-Hill, 1937, ch. 5, pp. 77–79.
[60] J. von Neumann, *Theory of Self-Reproducing Automata*. Urbana, IL: Univ. Illinois Press, 1966.
[61] J. F. Wakerly, *Digital Design, Principles and Practices*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1994.
[62] S. Xanthakis, S. Karapoulios, R. Pajot, and A. Rozz, "Immune system and fault tolerant computing," in *Artificial Evolution*, J. M. Alliot, Ed: Springer-Verlag, 1996, vol. 1063, Lecture Notes in Computer Science, pp. 181–197.
[63] C. Zeng, N. Saxena, and E. J. McCluskey, "FSM synthesis with concurrent error detection," in *Proc. Int. Test Conf.*, Sept. 1999, pp. 672–679.

D. W. Bradley (S'00–M'01) received the M.Eng. degree in electronic systems engineering from the University of York, York, U.K. in 1998. He is working toward the Ph.D. degree in electronics at the same university.

He is currently a Microprocessor Engineer with ARM Ltd., Cambridge, U.K. His current research interests include microprocessor architectures, fault tolerance, biologically inspired hardware design, and artificial immune systems.

A. M. Tyrrell (S'84–M'85–SM'96) received the B.Sc. degree in electrical and electronic engineering from the Bolton Institute of Technology, Bolton, U.K., in 1982 and the Ph.D. degree in electrical and electronic engineering from Aston University, Birmingham, U.K., in 1985.

He joined the Electronics Department at York University in April 1990, where he is currently Chair in Digital Electronics. Previously, he was a Senior Lecturer at Coventry Polytechnic. Between August 1987 and August 1988, he was Visiting Research Fellow at Ecole Polytechnic Lausanne, Switzerland, where he was researching the evaluation and performance of multiprocessor systems. From September 1973 to September 1979, he was with STC at Paignton Devon, working on the design and development of high-frequency devices. He has authored or coauthored over 120 papers. His current research interests include the design of biologically inspired architectures, evolvable hardware, field-programmable gate array system design, parallel systems, fault-tolerant design, and real-time systems. Over the last six years, his research group at York have concentrated on bio-inspired systems. This work has included the creation of embryonic processing array, intrinsic evolvable hardware systems, and the immunotronics hardware architecture.

Dr. Tyrrell is a Fellow of the Institution of Electrical Engineers.