An Efficient Implementation of the Rainflow Counting Algorithm for Life Consumption Estimation

Mahera Musallam and C. Mark Johnson, *Member, IEEE*

***Abstract*—In many reliability design and model-based health management applications where load profiles are variable and unpredictable, it is desirable to have efficient cycle counting methods to identify equivalent full and half cycles within the irregular load profile. Conventional cycle-based lifetime models can then be applied directly to provide information about the life consumption of the products. The use of an off-line rainflow algorithm is a common solution for arbitrary loads, but it cannot be applied in real time in its original form. This paper presents an in-line coding algorithm which uses a stack-based implementation, and a recursive algorithm to pick out the equivalent full and half cycles of the irregular load profile. The method can be integrated easily within time-domain or serial data applications to generate equivalent full and half cycles as they occur. Thus it is of particular significance for life estimation in real-time applications where use of the traditional implementations of the counting algorithm is impractical. In comparison with the off-line traditional rainflow method, the on-line method doesn't require any knowledge of the time history of the load profile because it processes each minimum or maximum when it occurs. Therefore, it provides a more efficient cycle counting method using less memory storage, and making more efficient use of computational resources within the real-time environment.**

***Index Terms*—Cycle counting, life estimation, rainflow algorithm, real time.**

Acronyms

IGBT	Insulated Gate Bipolar Transistor
LC	Life consumption
Tmax	Temperature maxima value
Tmin	Temperature minima value
PWM	Pulse width modulation.

Notation

ΔT	Temperature variations
Tm	Mean temperature
$\Delta\varepsilon$p	Damage indicator based on the accumulated plastic strain over one cycle
$\mathrm{N_f}$	Number of cycles to fail
$\mathrm{n_i}$	Number of cycles in each histogram bin
k	The number of histogram bins
Ptrmin	A pointer that indicates the number of values in the minima stack
Ptrmax	A pointer that indicates the number of values in the maxima stack

Manuscript received October 17, 2011; revised May 25, 2012; accepted July 21, 2012. Date of publication October 10, 2012; date of current version November 27, 2012. This work is funded by EPSRC and the Innovative Electronics Manufacturing Research Center (IeMRC) (UK) through research grants EP/D068525 and EP/H03014X. Associate Editor: P. Lall.

M. Musallam is with the Electrical and Electronic Engineering Department, Manchester Metropolitan University, Manchester M1 5GD (e-mail: m.musallam@mmu.ac.uk).

C. M. Johnson is with the School of Electrical and Electronic Engineering, Nottingham University, NG7 2RD, U.K. (e-mail: Mark.Johnson@nottingham.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2012.2221040

I. Introduction

Cycle-counting algorithms have been used in different random load applications, specifically for the study of fatigue damage. Such counting methods include *level crossing* counting, *peak* counting, *simple range* counting, and the *rainflow* counting method. These methods are useful in the analysis of data to reduce a spectrum of varying load into a set of simple uniform data histograms. Its importance is that it allows the application of Miner's rule [1] to assess the fatigue life of a structure subject to complex loading.

The original algorithm was initially proposed by *Tatso Endo* and *M. Matsuishi* in 1968 [2] to count the cycles or the half cycles of strain-time signals. The principle of counting is carried out on the basis of the stress-strain behavior of the material (Fig. 1). It combines load reversals in a manner that defines a cycle as a closed hysteresis loop. Each closed hysteresis loop has a strain range and mean stress associated with it that can be compared with the constant amplitude. For example, as the material deforms from point a to b, it follows a path described by the cyclic stress-strain curve. At point b, the load is reversed, and the material elastically unloads to point c. When the load is reapplied from c to d, the material elastically deforms to point b, where the material remembers its prior history, i.e. from a to b, and deformation continues along path a to d as if event b-c never occurred.

Many algorithms have been used to define the varying loads using power spectra; Downing and Socie created one of the most widely referenced and utilized rainflow cycle-counting algorithms in 1982 [3], which was included as one of many cycle-counting algorithms in ASTM E 1049-85 [4]. The counting of a cycle using this algorithm employs the 3-point counting rule with equivalent data information. This algorithm is used in Sandia National Laboratories *LIFE2 code* [5], and widely implemented for fatigue analysis studies in wind turbine components. Rychlik gave a mathematical definition for

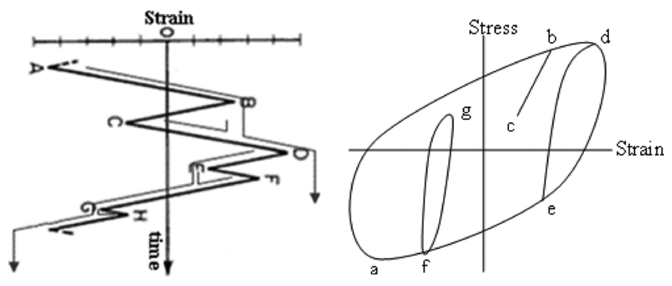

Fig. 1. Stress-strain cycles.

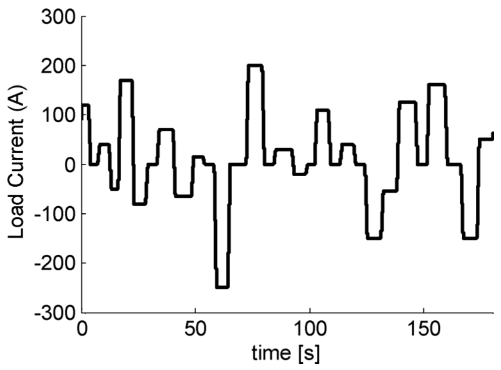

Fig. 2. Sample of load profile.

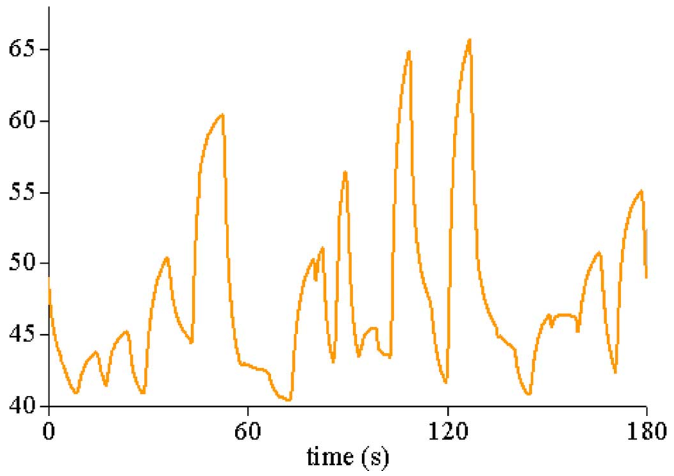

Fig. 3. Sample of the real-time IGBT substrate-solder temperature estimates.

the rainflow counting method [6] thus enabling closed-form computations from the statistical properties of the load signal. Monte Carlo simulations or the narrow-band approximation and corrective factors were also commonly used [7]. Recently, the rainflow algorithm has been written according to the ASTM standard [4], and optimized to be useful in implementation using Matlab tools [8].

Counting algorithms are commonly used in structural engineering or mechanical vibration applications [9], [10] with arbitrary operational conditions. More recently, counting algorithms have been applied for fatigue analysis [11], [12], and to failure analysis in power electronic modules employed in real-time applications [13]. In all of these applications, it is useful to identify an appropriate cycle counting method to generate regular spectrums of load profiles with varying frequency occurrences and amplitudes. Rainflow counting algorithms are widely used as a rule for pairing local minima and maxima to generate equivalent load cycles. In typical implementations, this counting method considers the entire time history of the load as an input, and the equivalent cycles are then determined at the end of that time history. This approach works because most algorithms start the counting sequence at the highest maximum or lowest minimum value, something which can only be determined once a full data set is captured. Thus, to apply the algorithm to continuous time series data, one has to choose an appropriate length of time history, and then process the data in blocks. This approach is inconvenient because the algorithm must process all of the stored data at the same time, taking significant computational resources; and derived quantities such as life consumption estimates are only updated at the end of each block.

In this paper, a rainflow counting algorithm is presented with a real-time implementation in which the extremal points are processed as they occur to generate the equivalent cycles. We illustrate a comparative example for estimating the life consumption of IGBT power modules under the effect of an arbitrary load profile using the traditional algorithm and the proposed on-line method. The new counting approach is shown to overcome the data storage and processor resource limitations of the traditional, off-line implementations.

II. Traditional Method (The Off-Line Method)

In life estimation studies of a product or a structure, counting algorithms are unnecessary for simple periodic loadings. Cycles can be counted easily, and therefore the life expectancy of the studied product can be simply estimated. For random load processes, and complex loading profiles such as the sample in Fig. 2, load cycles cannot be assessed by simply counting successive maxima and minima. Instead, a cycle counting algorithm such as the rainflow algorithm must be applied to successfully extract all of the equivalent cycles.

A. Application of the Traditional Method

Typically, the rainflow algorithm is used to extract cycles from a load history, which can be obtained from measurement or from simulation. To illustrate the use of the off-line rainflow counting algorithm, the load profile in Fig. 2 is considered as an example to study the substrate-solder (solder layer connecting the isolation substrate to the baseplate) failure mechanism.

As a first step, the load profile in Fig. 2 was applied to the electro-thermal model presented in [14]. This model is designed for a typical IGBT full bridge module, and it is used to estimate the temperatures of the substrate-solder layer. The temperature profile is presented in Fig. 3. The traditional rainflow counting algorithm [6] was applied off-line using Matlab [8] to generate the full, and half cycles, as illustrated in Fig. 4.

Following the principle of the traditional rainflow algorithm, the Matlab code functions work on the temperature-time data to find the extrema and minima values from the temperature-time signal. The full, and half cycles are defined accordingly. Fig. 4 shows the cycles extracted from the signal (temperature-time data) using the off-line Matlab code of the rainflow algorithm.

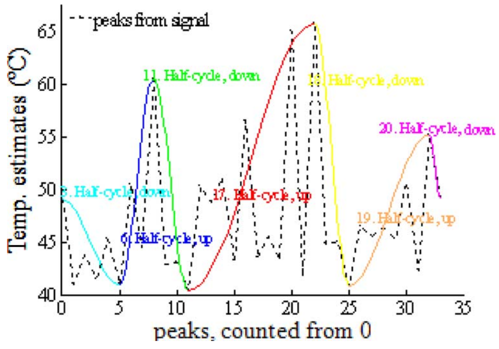

Fig. 4. Rainflow cycles sample extracted from temperature data for random load profile.

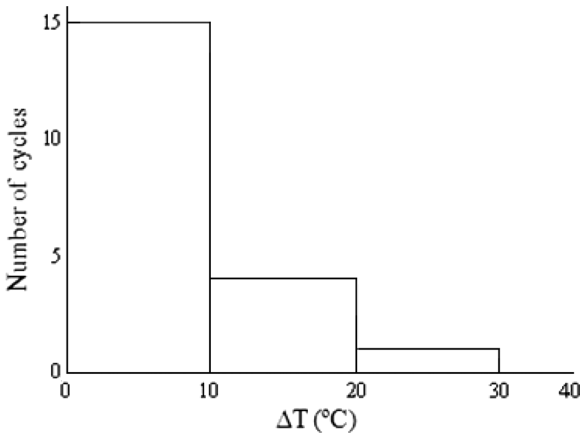

Fig. 5. Rainflow histogram of IGBT substrate-solder temperature variations (ΔT).

In Fig. 4, the dotted line represents the actual signal (temperature-time) data. The figure shows that there are 20 half cycles up, and 19 half cycles down. In other words, these data can be approximated to 20 full cycles of temperature data. Once the cycles are defined, another subroutine is used to create data histograms of the temperature values with fixed intervals, as represented in Fig. 5.

Using this method, the actual values of ΔT are not used, but instead ΔT values are extrapolated to the center of the bin value, or the value of the higher edge of the bin, to account for higher temperature variations. This method reduces the total amount of data that needs to be analysed in any subsequent life consumption calculation, but with a consequential loss of resolution and accuracy. If the traditional method uses the selected cycles directly (i.e. the ones in Fig. 4) to calculate the life consumption, the results will be more accurate.

B. Life Consumption Example Based on the Traditional Rainflow Counting Algorithm

Counting algorithms are necessary to estimate the lifetime in products for unpredictable load applications. One such example is lifetime consumption estimation for substrate-solder in power electronic modules which are employed in harsh environment conditions. Previous thermal fatigue tests were performed for the solder-substrate joints, and then investigated by simulating the crack growth process under a set of prescribed field temperature profiles that cover the period of the operational life [15]. Predicted is the overall crack length in the solder joint for all different thermal profiles and number of cycles for each profile [16]. The model relates the plastic strain $(\Delta\varepsilon\mathrm{p})$ with the two design variables (ΔT, Tm) as in

$$Ln(\Delta\varepsilon_p) = -17.73 + 0.2984\Delta T + 0.07957Tm \\ -0.001722\Delta T^2 - 0.0002478Tm^2 - 0.0005389\Delta TTm \quad (1)$$

To estimate the lifetime of the substrate-solder in terms of the number of cycles to failure, the model in (1) was integrated with the Coffin-Manson model [17], [18]. Thus, the number of cycles to fail $\mathrm{N_f}$ (i.e. number of cycles needed for the crack length to reach L) can be calculated as in

$$N_f = \frac{L}{0.0056(\Delta\varepsilon_p)^{1.023}} \quad (2)$$

The constants 0.0056 and 1.023 are derived based on the characteristics of the solder material [16]. Through application of a fatigue damage model and the Palmgren-Miner linear damage accumulation rule, the effects of different loads can be combined, and the life consumption (LC) calculated as follows [13].

$$LC = \sum_{i}^{k} \frac{n_i}{N_{fi}}$$

$\mathrm{n_i}$: number of cycles in each bin from 1 to k.

$$LC = \left[\frac{n_1}{Nf_1}\right]_1 + \left[\frac{n_2}{Nf_2}\right]_2 + \left[\frac{n_3}{Nf_3}\right]_3 \\ + \left[\frac{n_4}{Nf_4}\right]_4 + \cdots + \left[\frac{n_k}{Nf_k}\right]_k \quad (3)$$

For this example, using a histogram bin interval of 10 °C, the total life consumption using the off-line rainflow algorithm can be estimated to be 6.34×10^{-5}. The method of calculation is shown in Table I.

TABLE I
EXAMPLE CALCULATION OF LIFE CONSUMPTION FOR SUBSTRATE-SOLDER UNDER THE INFLUENCE OF RANDOM LOAD USING THE TRADITIONAL OFF-LINE RAINFLOW COUNTING ALGORITHM

Number of cycles	0≤ΔT≤10 (°C)	10<ΔT≤20 (°C)	20<ΔT≤30 (°C)
n (cycles)	15	4	1
ΔT	10	20	30
$\mathrm{N_f}$	$3.8 \times 10^{+5}$	$2.3 \times 10^{+5}$	$1.4 \times 10^{+5}$
LC = n/ $\mathrm{N_f}$	3.93×10^{-5}	1.71×10^{-5}	6.98×10^{-6}
Total LC	6.34×10^{-5}		

III. Real-Time Rainflow Coding Algorithm (The On-Line Method)

A. Challenges in Implementing the Traditional Algorithms

For real-time applications, applying the traditional rainflow counting algorithm is very challenging. Significant amounts of data must be stored and processed periodically to obtain a description of the frequency and magnitude of the data in equivalent regular cycles. This effort is often described through a frequency histogram covering finite ranges of data swings, and possibly also the mean values of these data sets. In addition to the requirement that the algorithm must be applied to a stored set of data, it must also generate a buffer that can accommodate the successive extremal (minima & maxima) values. The size of the buffer, and therefore the interval between successive applications of the rainflow algorithm, must be determined before the algorithm is executed. In general, the buffer must be large enough to capture the full operational loading range of the system, which may mean anything from a few hours to many days of data, depending on the mission profile and the environmental conditions. Clearly, this work could involve huge quantities of data which will require substantial computational resource to process each time the algorithm is executed. If this work is done within a real-time control platform, careful coding will be required to ensure that real-time processes are not interrupted.

B. Development of the Real-Time Rainflow Coding Algorithm

To address the challenges in implementing the rainflow counting algorithm in real time, a new approach is proposed in which each maxima or minima value is processed as it occurs with the help of two flexible buffers or stacks. Full and half cycles are identified recursively, based on the stack contents, and the life consumption models are updated accordingly. It should be stressed that the algorithm for picking out the full and half cycles is identical to that used in the conventional approach, and the results can therefore be expected to be the same for identical data sequences. The key difference is that it is clearly impossible to provide any pre-ordering of data to identify the global extrema that are used as the starting point for many implementations. The paper presents the new algorithm in detail, together with results of its implementation and integration with real-time interfacing tools to ensure that real-time control processes are not interrupted.

Once the load profile is launched into operation, the real-time rainflow counting algorithm defines two flexible-sized buffers to process the data. The algorithm starts by identifying each minimum or maximum data value as it occurs by employing a 3-point counting rule. It then applies a recursive algorithm, as illustrated in Figs. 6 and 7, to the new extremal point, and those stored in two stacks containing previous minimum and maximum values. The recursive method works on both the minimum and maximum values simultaneously in their respective stacks to pick out the full and half cycles. Initial conditions for the stacks' contents can be defined in two ways. In the first method, the stacks are initialized with the first two extremal values, one maximum, one minimum. The second method is to initialize the stacks with artificial values that are larger, in absolute terms, than the highest expected maximum or lowest expected minimum values. With the first initialization method, a number of half cycles will be generated as successively larger extreme values (either maxima or minima) are encountered in the real-time data. These values will replace those in the left-most stack position. With the second initialization method, only full cycles are produced, and the assumed initial values are never replaced. In this case, the final cycle resulting from the two artificial values is not counted.

Note that for practical implementations the impact of the choice of initialization method will decrease with dataset size, and hence run time.

In Fig. 6, T refers to the data values which could be temperatures, and $\Delta\mathrm{T}$ is the variation in these values, i.e., temperature variations. Tmean is the mean value, and it equals $\mathrm{Tmin} + \Delta\mathrm{T}/2$. If a maximum value already exists in the maxima stack, it will be compared with the new value; otherwise, the new value will become the first value on the stack. If the new value is greater than the first value in the stack, then the minima stack will be checked for saved values, and this is indicated by its pointer Ptrmin. If only one minimum value exists, then a half cycle will be announced, and the temperature variation $\Delta\mathrm{T}$ will be the absolute value of the difference between this minimum value and the old maximum one. Tmean is calculated consequently. The old maximum value will then be removed from the maxima stack by replacing it with the new value.

Alternatively, if the minimum pointer indicates more than one value in the minima stack, this shows that a full cycle is decided, and $\Delta\mathrm{T}$ will be the absolute difference between the new minimum value and the old maximum one. In this case, the old maximum value and the new minimum value will be removed.

In the case of having a new maxima value that is smaller than the old one, the new value will be saved in the maxima stack alongside the old values. If the maxima stack contains more than one value, and that is indicated by its pointer Ptrmax, the whole operation will be repeated recursively.

Similarly, when a new minimum value arrives, it will be compared with the first value in the minima stack. If the new value is less than the old one, the maxima stack will be checked for saved values, and this result is indicated by its pointer Ptrmax. If only one maximum value exists, a half cycle will be announced, and the temperature variation $\Delta\mathrm{T}$ will be the absolute value of the difference between this maximum value and the old minimum one. The old minimum value will then be removed from the minima stack by replacing it with the new one. If the maximum pointer indicated more than one value in the maxima stack, then a full cycle is decided, and $\Delta\mathrm{T}$ will be the absolute difference between the new maximum value and the old minimum one. In this case, the old minimum value and the new maximum value will be removed. In the case of having a new minima value that is greater than the old one, the new value will be saved in the minima stack alongside the old values. If the minima stack contains more than one value, and that is indicated by its pointer Ptrmin, the whole operation will be repeated recursively. Fig. 7 shows a flowchart of the recursive algorithm that works on minimum values.

For both minima and maxima values, the stack size changes dynamically depending on the complexity of the loading data. Sequences of extremal values that reduce in absolute value (such as a damped oscillatory response) will lead to an increase in stack size. Conversely, the stack size reduces when extremal values of magnitude larger than the smallest value stored in the stack are encountered. Thus the stack sizes will vary with time depending on the load profile. Note that it is also possible to

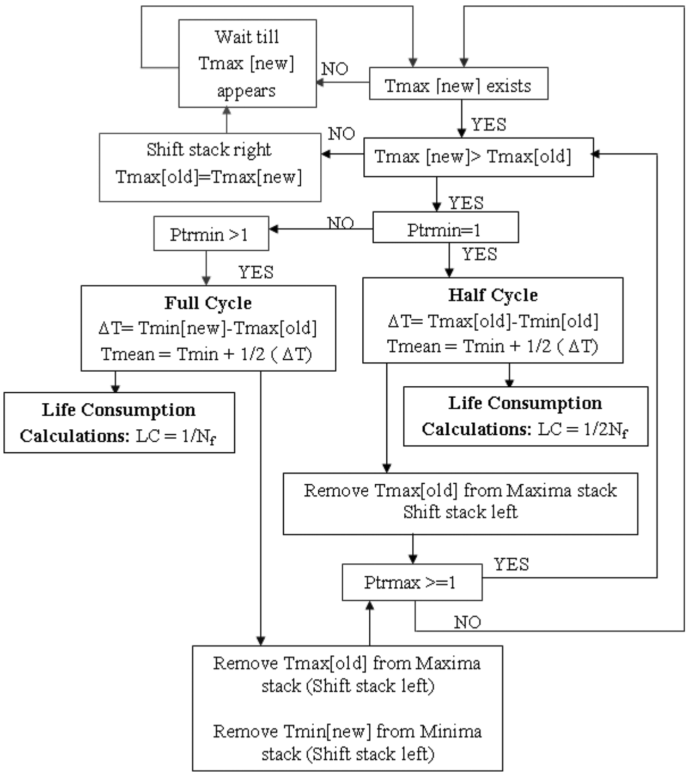
Wait till Tmax [new] appears
NO
Tmax [new] exists
YES
NO
Shift stack right Tmax[old]=Tmax[new]
Tmax [new]> Tmax[old]
YES
Ptrmin >1
NO
Ptrmin=1
YES
YES
Full Cycle ΔT= Tmin[new]-Tmax[old] Tmean = Tmin + 1/2 (ΔT)
Half Cycle ΔT= Tmax[old]-Tmin[old] Tmean = Tmin + 1/2 (ΔT)
Life Consumption Calculations: LC = 1/N_f
Life Consumption Calculations: LC = 1/2N_f
Remove Tmax[old] from Maxima stack Shift stack left
YES
Ptrmax >=1
NO
Remove Tmax[old] from Maxima stack (Shift stack left)
Remove Tmin[new] from Minima stack (Shift stack left)

Fig. 6. Schematic example of the recursive algorithm to pick out half and full cycles which works on maximal values. $\mathrm{Ptrmin} = 1$ means one minimum value in the minima stack, $\mathrm{Ptrmin} > 1$ means more than one minimum value are in the minima stack, and $\mathrm{Ptrmax} >= 1$ means more than one maximum value are in the maxima stack.

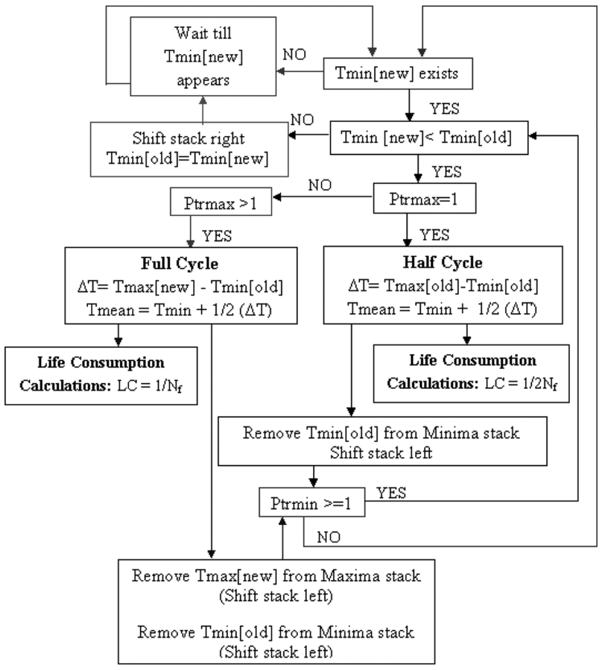
Wait till Tmin[new] appears
NO
Tmin[new] exists
YES
NO
Shift stack right Tmin[old]=Tmin[new]
Tmin [new]< Tmin[old]
YES
Ptrmax >1
NO
Ptrmax=1
YES
YES
Full Cycle ΔT= Tmax[new] - Tmin[old] Tmean = Tmin + 1/2 (ΔT)
Half Cycle ΔT= Tmax[old]-Tmin[old] Tmean = Tmin + 1/2 (ΔT)
Life Consumption Calculations: LC = 1/N_f
Life Consumption Calculations: LC = 1/2N_f
Remove Tmin[old] from Minima stack Shift stack left
Ptrmin >=1
YES
NO
Remove Tmax[new] from Maxima stack (Shift stack left)
Remove Tmin[old] from Minima stack (Shift stack left)

Fig. 7. Schematic example of the recursive algorithm to pick out half and full cycles which works on minimal values. $\mathrm{Ptrmax} = 1$ means one maximum value is in the maxima stack, $\mathrm{Ptrmax} > 1$ means more than one maximum value are in the maxima stack, and $\mathrm{Ptrmin} >= 1$ means more than one minimum value are in the minima stack.

combine the maxima and minima stacks into one containing alternate maximum and minimum values to which a common algorithm can be applied.

C. Application Example

Using this method, the life consumption and remaining life of a product can be estimated by integrating automatically the relevant lifetime models with the derived values of $\Delta\mathrm{T}$ and Tmean each time they occur. To illustrate how this method works on some sample data, consider the sequence of values presented in Fig. 8.

Once the code starts running, it defines two flexible-sized buffers: the maxima, and minima stacks. The values are compared based on the 3-point counting rule to define a minima or a maxima value. In this example, the first two extremal points are saved to the stacks: 43 is saved in the minima stack, and 81 is saved in the maxima stack. Each stack pointer refers to the number of values in their relevant stacks. The next value 27 arrives, and based on the 3-point counting rule it is found to be a minima. It is then compared with the previous minima value. In this case, $27 < 43$, and there is only one value in the maxima stack ($ptrmax = 1$). Thus a half cycle is decided, which is the difference between the maxima value and the previous minima (i.e. $\Delta T = 81 - 43 = 38$), and the old minima value is removed from its stack. This process continues, and the next value 65 is a maxima, but as it is less than the previous saved maxima 81, it is put in the maxima stack, and the maxima pointer is incremented to indicate two maxima values ($ptrmax = 2$). The process continues, and the next value 10 is a minima and less than the previous minima, thus a full cycle is defined as there is more than one value in the maxima stack. The full cycle is determined with $\Delta T = 65 - 27 = 38$, and in this case both values 65 and 27 are removed from their stacks. The next value 97 arrives,

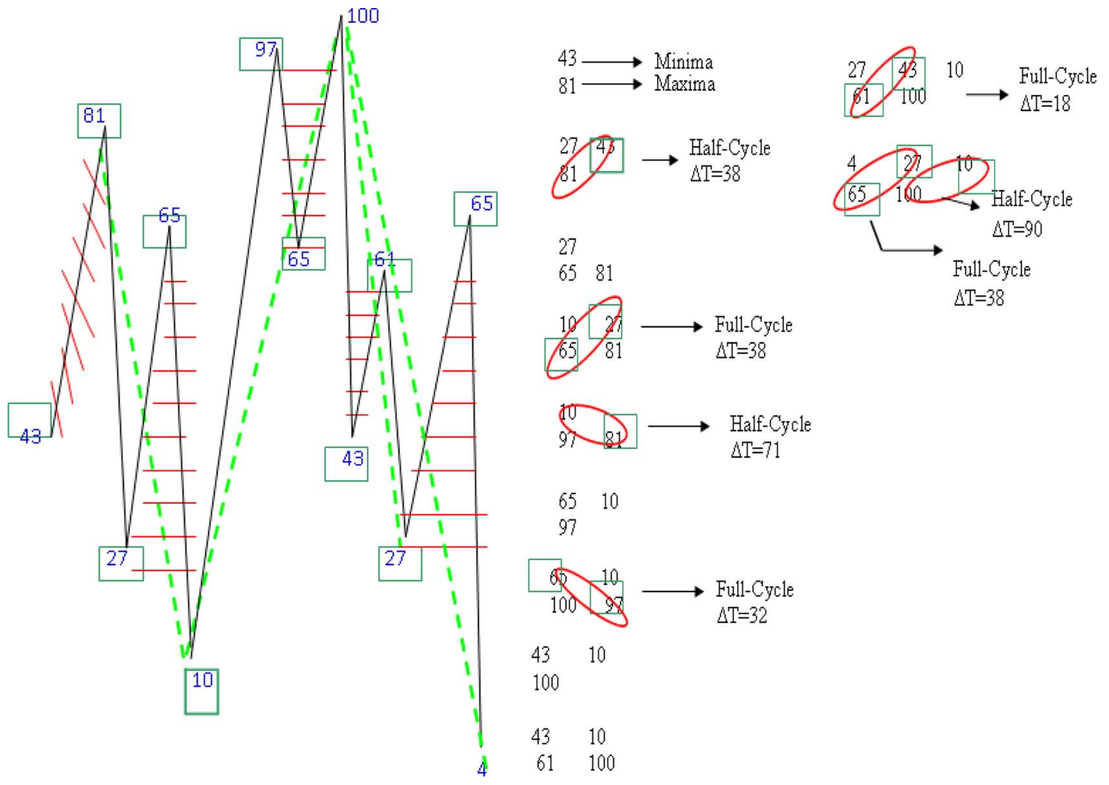

Fig. 8. Example of implementing the on-line rainflow algorithm.

and it is a maxima and bigger than the previous value 81, a half cycle is decided because the minima pointer indicates just one value $(ptrmin = 1)$ in the minima stack. $\Delta\mathrm{T} = 81 - 10 = 71$, and the value 81 is then removed from its stack. Note that the final minimum value 10 remains on the stack. The next value 65 is a minima, but bigger than the previous stored minima, thus it is pushed onto the stack. When the value 100 arrives, it is a maxima, and bigger than the previous stored maxima. The minima stack has more than one value $(ptrmin = 2)$, and thus a full cycle is defined with $\Delta T = 97 - 65 = 32$. Both values 97 and 65 are removed from their relevant stacks.

The algorithm code continues working on all the required data. Each time a new value arrives, the code checks all contents of the stacks recursively until the conditions for full or half cycles or both do not apply. Where the conditions to pick out full and half cycles from the available maxima and minima values don't apply, the values will remain stored in their relevant stacks until a new value arrives where the comparison conditions do apply. Such a condition is clearer for the last few values in this example. When 65 arrives as a maxima, there are already two minima values 27 and 10 stored in their stack; and because 65 < 100, 65 stays in the maxima stack. When the last value 4 arrives, it is a minima, and it is less than the previous value in the minima stack, i.e. 4 < 27. $Ptrmax$ indicates that there are two values saved in the maxima stack, so a full cycle is decided, and $\Delta T = 65 - 27 = 38$. The code works recursively, removing 65 and 27 from their stacks, and then checking for applicable conditions again. The conditions still apply for the minimum values 4 and 10; and as $Ptrmax$ indicates, that one value is left on the maxima stack, and another half cycle is decided with $\Delta T = 100 - 10 = 90$. The value 10 is then removed from its stack.

A summary of how the on-line rainflow method works on the previous set of data is illustrated in Table II. Each column cell represents the occurrence of a minimum or a maximum in their relevant stack. The numbers in brackets show the sequence of removing that value from its relevant stack.

IV. Implementation of the Real-Time Version of the Rainflow Algorithm

As an example of applying the on-line version of the rainflow cycle counting algorithm in a real-time application, the rainflow code was integrated with the electro-thermal models presented in [14]. The models in [14] are designed for a typical IGBT full bridge module, and then used to estimate the temperatures of significant points within the power modules such as the die junction temperature. These temperatures are estimated relative to a (measured) reference ambient temperature (in this case 40 °C). The models are implemented on a DSpace platform [19] along with a fixed frequency PWM current controller in which the current control loop is updated synchronously each PWM cycle. With the knowledge of the operational profile, the power dissipated by each active device within the module is determined, giving the on-state and switching losses as a function of the measured device, load current, and the estimated temperature. The real-time rainflow code was applied to the resulting temperature-time data to identify and pick out full and half temperature cycles.

As an example of a random load profile, the test was conducted by incorporating the load profile presented earlier in

TABLE II
SUMMARY EXPLAINING THE ON-LINE RAINFLOW METHOD USING AN EXAMPLE OF RANDOM DATA

Minina Stack	~~43~~ (1)	~~27~~ (2)	~~10~~ (7)		~~65~~ (4)		~~43~~ (5)		~~27~~ (6)		4	
Maxima Stack	~~81~~ (3)	~~65~~ (2)		~~97~~ (4)		100		~~61~~ (5)		~~65~~ (6)		
Ptrmax	1	1	1	1	1	1	1	2	1	2	1	1
Ptrmin	1	2	1	1	2	1	2	2	2	2	2	1
Half Cycle (ΔT)		(81-43)=38		(81-10)=71								(100-10)=90
Full Cycle (ΔT)			(65-27)=38			(97-65)=32			(61-43)=18		(65-27)=38	

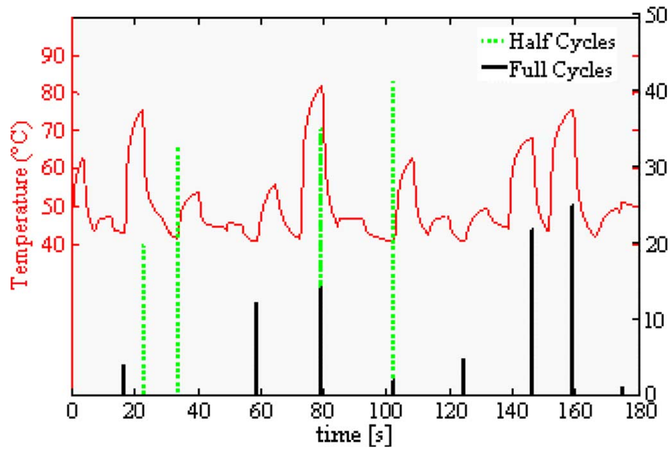

Fig. 9. Defined full and half cycles for the junction temperature data using the real-time rainflow algorithm.

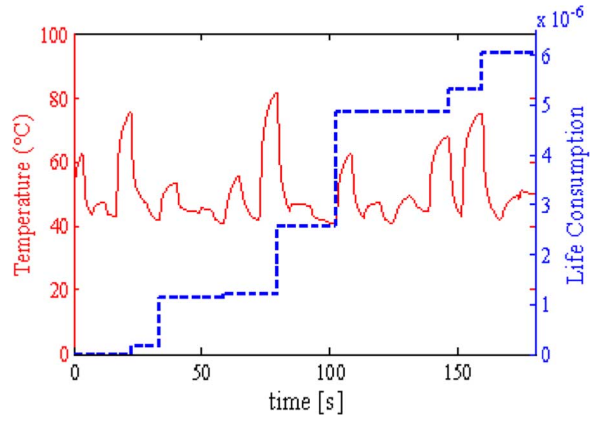

Fig. 10. Real-time life consumption for the bond wire alongside the temperature variations.

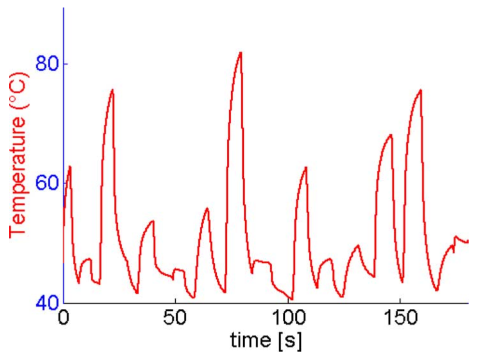

Fig. 11. Sample of the temperature estimates.

Fig. 2 with the real-time thermal model [14], which includes the on-line rainflow algorithm. For the purpose of this work, the test was running for 180 seconds.

The real-time rainflow algorithm works automatically on the temperature changes to pick out full and half cycles of the generated temperature cycles for the IGBT device junction. The corresponding junction temperatures, alongside the identified full and half cycles, are shown in Fig. 9.

Variations in junction temperature cause fatigue of the bonded interface between the Silicon IGBT dies and the Al wires that are bonded onto the upper surface of the die to provide an electrical connection [15]. To estimate the life consumption (LC) of the power modules studied in this work, lifetime models for the bond wire [15], [20]–[22] were incorporated and updated automatically with the identified half and full cycles. LC is updated regularly each time a new full or half cycle is defined. Fig. 10 shows the resulting on-line life consumption of the wire bond, and clearly illustrates that the life consumption responds to the temperature changes within the power module. Note that the nonlinear nature of the model (power law with exponent of -3.597) means that some of the smaller cycles do not produce a noticeable change in the life consumption. For this test, the on-line life consumption for the bond wire under the influence of the variable load is 6.06×10^{-6}. Thus, during the 180s period under which the load profile was applied, 0.000606% of the wire bond life was consumed.

V. VALIDITY OF THE ON-LINE COUNTING ALGORITHM

The temperature-time data for the IGBT device junction obtained from the real-time estimation test performed earlier can be used to obtain off-line life estimations using the traditional algorithm. Fig. 11 represents the instantaneous junction temperature of the IGBT power module under study.

The traditional rainflow algorithm was applied off-line to the resulting temperature-time data to identify and pick out full and half temperature cycles, as shown in Fig. 12. It is used then to

TABLE III
CALCULATION OF LIFE CONSUMPTION FOR THE WIRE BOND UNDER THE INFLUENCE OF RANDOM LOAD USING THE TRADITIONAL OFF-LINE RAINFLOW COUNTING ALGORITHM

Number of cycles	5≤ΔT≤15 (°C)	15<ΔT≤25 (°C)	25<ΔT≤35 (°C)	35<ΔT≤45 (°C)
n (cycles)	10	4	4	2
ΔT	5	15	25	35
N_f	428482343	8236216.93	1311407.987	390944.66
LC = n/ N_f	$2.33\text{x}10^{-8}$	$4.86\text{x}10^{-7}$	$3.05\text{x}10^{-6}$	$5.11\text{x}10^{-6}$
Total LC	$8.67\text{x}10^{-6}$			

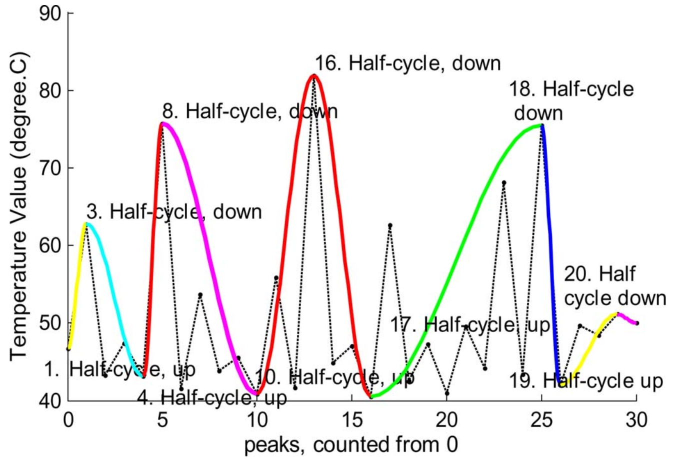

Fig. 12. Rainflow full and half temperature cycles.

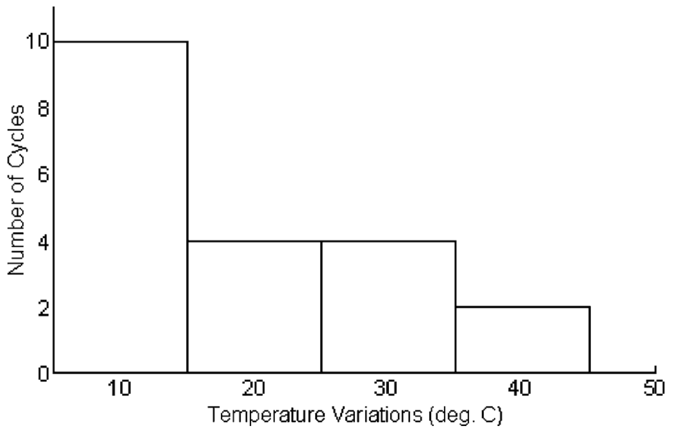

Fig. 13. Rainflow histograms of temperature variations ($\Delta\mathrm{T}$).

create a temperature range histogram with fixed intervals, as represented in Fig. 13.

Through application of a fatigue damage model for bond wire, [13], [15], and the Palmgren-Miner linear damage accumulation rule, the effects of different loads can be combined, and the life consumption (LC) calculated as in

$$LC = \sum_{i}^{k} \frac{n_i}{N_{fi}}$$

$\mathrm{n_i}$: number of cycles in each bin from 1 to k.

$$LC = \left[\frac{n_1}{Nf_1}\right]_1 + \left[\frac{n_2}{Nf_2}\right]_2 + \left[\frac{n_3}{Nf_3}\right]_3 + \left[\frac{n_4}{Nf_4}\right]_4 + \cdots + \left[\frac{n_k}{Nf_k}\right]_k \quad (4)$$

$$N_f = (1.4 * 10^{11})\Delta T^{-3.597} \quad (5)$$

For this example, using a histogram bin interval of 10 $^\circ\mathrm{C}$, the total life consumption using the off-line rainflow algorithm can be estimated to be 8.67×10^{-6}. The method of calculation is shown in Table III.

From these results, the life consumption for the wire bond was 8.67×10^{-6} while it was 6.06×10^{-6} using the on-line method. In comparison, the results show a small but significant difference (i.e. less than 30%). This difference can be attributed to the use of the frequency histogram approach in the off-line method, which groups cycle information into bins, and thus loses resolution. On the other hand, using the on-line method for short execution-time applications, say on the order of 180 seconds, some extremal values (minima or maxima or both values) may remain stored in their relevant stacks at the end of the sequence. These values can be removed by using artificial termination values to trigger stack emptying in a similar way to the second initialization method described in Section III. However, this issue is of little concern for real-time applications which operate continuously over long periods of time where the impact of the initialization and termination conditions will be insignificant.

The coding approach of the rainflow algorithm presented in this work can be more easily integrated within real-time applications when compared to the conventional approach. Firstly, it needs less data storage as it doesn't need to consider the whole time history of the applied load; and secondly, the processing of data is distributed over the whole operating period rather than being concentrated. The real-time rainflow code can be coupled and synchronized with other control functions with little risk of process interruptions during operation. Moreover, as this approach delivers updates to life consumption estimates at each extremal point, there is no need to determine an arbitrary load cycle duration to support the block processing of the traditional method. The real-time rainflow algorithm thus makes it easier to derive life consumption estimates for rapidly varying and complex load profiles in real-time applications such as wind turbines, rail traction, and automotive drive trains. Finally, unlike many implementations of the off-line method, where the cycles are reduced to a uniform data histogram with finite bin width, the on-line method preserves the complete information (mean, and $\Delta\mathrm{T}$) for each cycle. It can thus be expected to offer better resolution, and higher accuracy.

VI. Conclusion

Many real life applications operate under conditions of continuous, arbitrary power cycling, or thermal cycling which makes it difficult to track the real-time degradation within the system using conventional methods for cycle counting. Commonly used implementations of the rainflow algorithm consider the entire time history of the load as an input, and the equivalent cycles are then determined at the end of that time history, making the algorithm poorly suited to real-time implementation.

This paper has presented an efficient, real-time approach to implement the rainflow algorithm for applications that require the use of cycle counting to generate life consumption estimates for an arbitrary load profile. The method uses a stack-based implementation in which successive extremal values are compared with previous values through a recursive algorithm to identify the equivalent full and half cycles of the load profile. For in-service health management applications, as exemplified by power electronic module converters, this technique allows lifetime models to be integrated into the real-time control algorithm of the converter to provide life consumption estimates which evolve with the load history. A comparison of the real-time algorithm with a conventional implementation showed that the new algorithm is capable of predicting life consumption with greater accuracy, and with smaller data storage requirements. The new algorithm thus presents an attractive method for implementing fast, efficient cycle counting in real-time environments.

References

[1] M. A. Miner, "Cumulative damage in fatigue," *J. Appl. Mech. Trans. ASME*, vol. 12, pp. A159–A164, 1945.

[2] M. Matsuishi and T. Endo, "Fatigue of metals subjected to varying stress," *Japan Soc. Mech. Engineering*, 1968.

[3] S. D. Downing and D. F. Socie, "Simple rainflow counting algorithms," *International Journal of Fatigue*, vol. 4, no. 1, pp. 31–40, 1982.

[4] *Standard Practices for Cycle Counting in Fatigue Analysis, Annual Book of ASTM Standards*, ASTM E 1049-85 (Reapproved 1997), 1999, vol. 03.01, Philadelphia, USA.

[5] L. Schluter, Programmer's Guide for LIFE2's Rainflow Counting Algorithm Sandia Report, SAND90-2260, 1991.

[6] I. Rychlik, "A new definition of the rainflow cycle counting method," *International Journal of Fatigue*, vol. 9, no. 2, pp. 119–121, 1987.

[7] N. Metropolis, "The beginning of the Monte Carlo method," *Los Alamos Science (1987 Special Issue dedicated to Stanislaw Ulam)* pp. 125–130, 1987 [Online]. Available: http://library.lanl.gov/la-pubs/00326866.pdf

[8] *Rain Flow Counting Method* 2007 [Online]. Available: http://web-scripts.softpedia.com/script/Scientific-Engineering-Ruby/Earth-Sciences/Rain- Flow-Counting- Method-32997.html

[9] S. Ariduru, *Fatigue Life Calculation by Rainflow Cycle Counting Method* 2004 [Online]. Available: http://wind.nrel.gov/designcodes/papers/FatLifeCalcByRFCycleCountingMeth_Ariduru.pdf

[10] M. Meggiolaro and J. Pinho de Castro, "An improved multiaxial rainflow algorithm for non-proportional stress or strain histories," *International Journal of Fatigue*, vol. 42, pp. 194–206, 2011.

[11] K. L. Singh and V. R. Ranganath, "Cycle counting using rainflow algorithm for fatigue analysis," in *15th National Conference on Aerospace Structures*, Coimbatore, Tamil Nadu, India, October 15–16, 2007.

[12] B. Perrett, *An Evaluation of a Method of Reconstituting Fatigue Loading From Rainflow Counting* 1989 [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/a222664.pdf

[13] M. Musallam, M. Johnson, C. Yin, H. Lu, and C. Bailey, "Real-time life expectancy estimation in power modules," in *IEEE Electronics System Integration Conference–ESTC*, Greenwich, London, England, September 1–4, 2008, pp. 231–236.

[14] M. Musallam and C. M. Johnson, "Real-time compact thermal models for health management of power electronics," *IEEE Power Electronics Trans.*, vol. 25, no. 6, pp. 1416–1425, 2010.

[15] D. Chamund and C. Rout, *Reliability of High Power Bipolar Devices* October 2009 [Online]. Available: http://www.dynexsemi.com/assets/Application_Notes/DNX_AN5945.pdf, Application Notes

[16] H. Lu, T. Tilford, C. Bailey, and D. R. Newcombe, "Lifetime prediction for power electronics module substrate mount-down solder interconnect," in *Proceeding of the International Symposium on High Density Packaging and Microsystem Integration*, , China, June 26–28, 2007, pp. 40–45.

[17] L. Coffin, Jr., *Met. Eng. Q.*, vol. 3, pp. 15–24, 1963.

[18] S. Manson, *Thermal Stress and Low-Cycle Fatigue*. New York: McGraw-Hill, 1966.

[19] M. Musallam, M. Johnson, C. Yin, C. Bailey, and M. Mermet-Guyennet, "Real-time life consumption power modules prognosis using on-line rainflow algorithm in metro applications," in *IEEE Energy Conversion Congress &Expo–ECCE*, Atlanta, Georgia, USA, September 12–16, 2010.

[20] M. Pecht and A. Dasgupta, "Physics of Failure: An approach to reliable product development," *Journal of the Institute of Environmental Sciences*, vol. 38, pp. 30–34, 1995.

[21] H. Lu, W.-S. Loh, C. Bailey, and M. Johnson, "Computer modelling analysis of the globtop's effects on aluminium," in *IEEE Electronics System Integration Conference-ESTC*, Greenwich, London, England, September 1–4, 2008, pp. 1369–1375.

[22] C. Yin, H. Lu, M. Musallam, C. Bailey, and M. Johnson, "A prognostics assessment method for power electronics modules," in *IEEE Electronics System Integration Conference–ESTC*, Greenwich, London, England, September 1–4, 2008, pp. 1353–1358.

Mahera Musallam received the BA degree in electrical engineering from Birzeit University, Palestine; and the M.Sc. degree in Automation and Control, and the Ph.D. degree in power electronics and control, in 2001, and 2005, respectively, both from the University of Newcastle Upon Tyne University, UK. In January 2006, she joined Rolls-Royce/Royal Academy of Engineering Research Group at Sheffield University. Since October 2006, she was a member of a research team within a nation-wide collaborative program at Nottingham University on investigating prognostics and diagnostics environments for high reliability power electronics. She joined Manchester Metropolitan University on 1st January 2012 as a Lecturer in the Electrical and Electronic Engineering Department. Mahera's research areas include prognostics and health management of high reliability power electronic systems, i.e. power converters in wind energy, traction applications, and distribution networks.

C. Mark Johnson (S'89–M'91) received the BA degree in engineering, and the Ph.D. degree in electrical engineering from the University of Cambridge, UK, in 1986, and 1991 respectively. From 1992 to 2003, he was a Lecturer, and later a Reader at the University of Newcastle where he led research into Silicon Carbide electronics. In 2003, he was appointed as Rolls-Royce/RAEng Research Professor of Power Electronic Systems at the University of Sheffield, and in 2006 he was appointed to a personal chair at the University of Nottingham, where he leads research into power device packaging and integration technologies, reliability, health management, and power electronic applications. He is a member of the executive committee of the UK Innovative Electronics Manufacturing Research Centre (IeMRC).