(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4709268号
(P4709268)

(45)発行日　平成23年6月22日(2011.6.22)　(24)登録日　平成23年3月25日(2011.3.25)

(51)Int.Cl.		F I		
B60R 16/02	(2006.01)	B60R	16/02	650J
G06F 11/18	(2006.01)	G06F	11/18	310C
F02D 45/00	(2006.01)	F02D	45/00	372G
		F02D	45/00	374C
		B60R	16/02	660G

請求項の数 21　(全 24 頁)

(21)出願番号　特願2008-303369(P2008-303369)
(22)出願日　平成20年11月28日(2008.11.28)
(65)公開番号　特開2010-126012(P2010-126012A)
(43)公開日　平成22年6月10日(2010.6.10)
審査請求日　平成22年9月8日(2010.9.8)

(73)特許権者　509186579
日立オートモティブシステムズ株式会社
茨城県ひたちなか市高場２５２０番地
(74)代理人　100100310
弁理士　井上　学
(72)発明者　大川　圭一朗
茨城県ひたちなか市大字高場２５２０番地
株式会社　日立製作所　オートモティブシステムグループ内

審査官　加藤　信秀

最終頁に続く

(54)【発明の名称】車両制御用マルチコアシステムまたは内燃機関の制御装置

(57)【特許請求の範囲】
【請求項１】
複数のコアを用いて車両に搭載された機器を制御するための車両制御用マルチコアシステムであって、
連続的に変化する入力値を取得し、前記機器の制御に必要な出力値を演算する演算処理手段を備え、
前記演算処理手段が前記複数のコア中の特定のコアにおいて特定のタイミングで演算処理を行った出力値と、
前記特定のコアとは異なる、前記複数のコア中の他のコアにおいて、
前記演算処理手段が前記特定のタイミングとは異なるタイミングで演算処理を行った出力値と、
の比較を行い、前記特定のコアに故障が生じていると判断するコア故障判断手段を備えることを特徴とする車両制御用マルチコアシステム。
【請求項２】
前記コア故障判断手段は、
前記演算処理手段の演算処理が実行されるコアが切り替わることによって、
前記演算処理手段の出力値の連続性が失われたときに、
前記特定のコアに故障が生じていると判断することを特徴とする請求項１に記載の車両制御用マルチコアシステム。
【請求項３】

前記コア故障判断手段は、連続した演算周期での演算処理手段の出力値を比較することを特徴とする請求項１または２いずれかに記載の車両制御用マルチコアシステム。
【請求項４】
前記車両制御用マルチコアシステムは、実行するコアとタイミングとがそれぞれ異なる出力値の差分を求め、前記差分が所定の範囲内にあるかどうかを判定する差分判定手段を備え、前記コア故障判断手段は前記差分判定手段の判定結果によって、
前記特定のコアに故障が生じていると判断することを特徴とする請求項１から３いずれかに記載の車両制御用マルチコアシステム。
【請求項５】
前記車両制御用マルチコアシステムは、少なくとも２回以上の演算処理を行った出力値の移動平均値を計算し、前記移動平均値と、前記特定のタイミングにおいて前記演算処理手段が演算処理を行った出力値との差分を求め、前記差分が所定の範囲内にあるかどうかを判定する差分判定手段を備え、前記差分判定手段の判定結果によって前記特定のコアに故障が生じていると判断することを特徴とする請求項１から３いずれかに記載の車両制御用マルチコアシステム。
【請求項６】
前記差分判定手段の判定に用いる所定の範囲は、少なくとも２回以上の演算処理を行った出力値の移動平均値に基づいて求めることを特徴とする請求項４から５いずれかに記載の車両制御用マルチコアシステム。
【請求項７】
前記コア故障判断手段は、前記差分判定手段による前記特定のコアの判定結果が所定の範囲内でないときに故障カウンタを増加し、前記故障カウンタが所定回数を超えたときに前記特定のコアに故障が生じていると判断することを特徴とする請求項４から６いずれかに記載の車両制御用マルチコアシステム。
【請求項８】
前記車両制御用マルチコアシステムは、前記演算処理手段を分割し、実行されるタイミングによって演算処理が実行されるコアが切り替わるように前記複数のコアに割り当てる演算処理割り当て手段を備え、
前記演算処理割り当て手段は、前記コア故障判断手段によって前記特定のコアに故障が生じていると判断された場合、前記特定のコアを割り当ての対象から除外して再度割り当てることを特徴とする請求項１から７のいずれかに記載の車両制御用マルチコアシステム。
【請求項９】
前記コア故障判断手段は、前回の演算周期において前記コア故障判断手段によって正常であると判断されたコアで行うことを特徴とする請求項１から８いずれかに記載の車両制御用マルチコアシステム。
【請求項１０】
前記差分判定手段は、前回の演算周期において前記コア故障判断手段によって正常であると判断されたコアで行うことを特徴とする請求項４から９のいずれかに記載の車両制御用マルチコアシステム。
【請求項１１】
前記差分判定手段および前記コア故障判断手段は、少なくとも３つ以上の奇数のコアで重複して実施され、前記実施の結果の多数決によってコア故障判断を行うことを特徴とする請求項４から１０のいずれかに記載の車両制御用マルチコアシステム。
【請求項１２】
前記差分判定手段および前記コア故障判断手段は、少なくとも２つ以上のコアで重複して実施し、前記少なくとも２つ以上のコア中の各コアそれぞれの判定結果を重み付けして集計し、前記集計の結果を用いてコア故障判断を行うことを特徴とする請求項４から１０のいずれかに記載の車両制御用マルチコアシステム。
【請求項１３】

前記差分判定手段および前記コア故障判断手段は、少なくとも３つ以上のコアで重複して実施し、前記少なくとも３つ以上のコア中の１つのコアだけが前記特定のコアは故障であると判定し、前記少なくとも３つ以上のコア中の他のコアが前記特定のコアは正常であると判定した場合は、故障であると判定したコアに故障が生じていると判断し、前記少なくとも３つ以上のコア中の１つのコアだけが前記特定のコアは正常であると判定し、前記少なくとも３つ以上のコア中の他のコアが前記特定のコアは故障であると判定した場合は、正常であると判定したコアと前記特定のコアとの両方に故障が生じていると判断することを特徴とする請求項４から１２記載の車両制御用マルチコアシステム。

【請求項１４】

前記車両の運転モードを少なくとも２つ以上規定する運転モード切替手段を持ち、前記運転モード切替手段によって前記運転モードが変更されたときは、前記コア故障判断を所定時間行わないことを特徴とする請求項１から１３いずれかに記載の車両制御用マルチコアシステム。

【請求項１５】

前記車両制御用マルチコアシステムは、前記演算処理手段に使用する少なくとも１つ以上の入力値を検出し、前記コア故障判断手段は、前記入力値のいずれかまたは複数に関して、特定の入力タイミングの入力値と、前記特定の入力タイミングより前の入力タイミングの入力値との差分が所定の範囲内にあるかどうかを判定する入力値差分判定手段を備え、前記入力値差分判定手段によって差分が所定の範囲外であると判定されたときには、前記コア故障判断を所定時間行わないことを特徴とする請求項１から１４のいずれかに記載の車両制御用マルチコアシステム。

【請求項１６】

前記入力値差分判定手段は、少なくとも２回以上のタイミングにおいて得られた入力値の移動平均値を計算し、前記入力値の移動平均値と、前記特定の入力タイミングにおける入力値との差分が所定の範囲内にあるかどうかを判定することを特徴とする請求項１５に記載の車両制御用マルチコアシステム。

【請求項１７】

前記入力値差分判定手段の判定に用いる所定の範囲は、前記入力値の移動平均値に基づいて求めることを特徴とする請求項１６に記載の車両制御用マルチコアシステム。

【請求項１８】

前記演算処理手段が前記複数のコアに割り当てられて、

演算処理を行うタイミングは、

前記機器の動作に同期して決定することを特徴とする請求項１から１７いずれかに記載の車両制御用マルチコアシステム。

【請求項１９】

内燃機関を制御するための複数のコアを備えた内燃機関の制御装置であって、連続的に変化する入力値を取得し、前記内燃機関の制御に必要な出力値を演算する演算処理手段を備え、前記演算処理手段が前記複数のコア中の特定のコアにおいて特定のタイミングで演算処理を行った出力値と、

前記特定のコアとは異なる、前記複数のコア中の他のコアにおいて、

前記演算処理手段が前記特定のタイミングとは異なるタイミングで演算処理を行った出力値と、の比較を行い、前記特定のコアに故障が生じていると判断するコア故障判断手段を備えることを特徴とする内燃機関の制御装置。

【請求項２０】

前記コア故障判断手段は、

前記演算処理手段の演算処理が実行されるコアが切り替わることによって、

前記演算処理手段の出力値の連続性が失われたときに、

前記特定のコアに故障が生じていると判断する

ことを特徴とする請求項１９に記載の内燃機関の制御装置。

【請求項２１】

前記演算処理手段を分割し、実行されるタイミングによって
演算処理が実行されるコアが切り替わるように前記複数のコアに割り当てる、
演算処理割り当て手段を備え、
前記演算処理割り当て手段は、
前記演算処理手段を前記複数のコアに割り当てるタイミングを、
前記内燃機関の動作に同期して決定することを特徴とする請求項１９または２０いずれかに記載の内燃機関の制御装置。

【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、車両に搭載された機器を制御する組み込み制御装置に係わり、特に複数のＣＰＵから構成されるマルチコアＣＰＵシステムの故障診断に関する。
【背景技術】
【０００２】
制御機器の信頼性向上や処理速度向上のために、制御コンピュータのＣＰＵを複数のプロセッサで構成したり、複数のＣＰＵコアで構成したマルチコアＣＰＵを用いるケースが増えている。このようなコンピュータにおいては、各プロセッサまたは各コアが故障した場合にはそれを速やかに検出し、当該プロセッサまたはコアを停止させ、制御への影響を防ぐなどの処理が必要となる。例えば特許文献１で開示されたような冗長系システムがよく知られている。特許文献１では、複数の冗長化されたサブシステムの出力したデータを比較し、データが一致している最も信頼度の高いサブシステムからの出力を選択するシステムが示されている。また特許文献２では、複数の出力の多数決比較をして最終的な出力を決定するシステムが示されている。また特許文献３では、特別な診断用のプログラムやデータを用いずに、通常の演算処理を行いながら複数のＣＰＵコアの診断を行う方法が示されている。また特許文献４では、各プロセッサにハードウェア障害を検出するためのテストプログラムを起動させるシステムが示されている。
【０００３】
【特許文献１】特開平３－１５９４６号公報
【特許文献２】特開平９－１６５３５号公報
【特許文献３】特開平３－１９６２３２号公報
【特許文献４】特開平６－３３２８７４号公報
【発明の開示】
【発明が解決しようとする課題】
【０００４】
上記のように、データの一致性比較を行うシステムは信頼性向上を目的としており、複数プロセッサまたは複数コアで同一の処理を行い、その処理結果を冗長系システムを構築することが前提となっている。また、特許文献３で提案されている故障診断方法では、通常の演算処理を複数のＣＰＵコアで同様に行い、それぞれの処理結果を比較するため、ＣＰＵのリソース消費が増大する課題がある。これらのシステムでさらに処理速度向上を行うためには、プロセッサまたはコアをさらに増やす必要があり、コストおよび搭載容積の増加を招いてしまう。車両用制御装置においては、コストパフォーマンスと搭載性の観点から、これらのシステムを適用することは難しい。なお、この課題に関し、特許文献４では分散処理を行いつつ故障診断用のテストプログラムを各プロセッサで起動するシステムが提案されている。しかしこのテストプログラムは、通常時の演算処理とは異なる演算処理であり、各プロセッサまたは各コアにとって演算負荷となるという課題がある。
【０００５】
本発明の目的とするところは信頼性向上と処理速度向上との両立であり、特別なテストプログラムなどによる冗長な演算処理，ＣＰＵリソースの消費を抑制しながらマルチコアＣＰＵシステムを用いる車両用制御装置の複数コアの故障判定を実現することにある。
【課題を解決するための手段】

【０００６】
上記課題を解決するため、本発明は、複数のコアを用いて車両に搭載された機器を制御するための車両制御用マルチコアシステムであって、連続的に変化する入力値を取得し、前記機器の制御に必要な出力値を演算する演算処理手段を備え、前記演算処理手段が前記複数のコア中の特定のコアにおいて特定のタイミングで演算処理を行った出力値と、前記特定のコアとは異なる、前記複数のコア中の他のコアにおいて、前記演算処理手段が前記特定のタイミングとは異なるタイミングで演算処理を行った出力値と、の比較を行い、前記特定のコアに故障が生じていると判断するコア故障判断手段を備えることを特徴とする。

【発明の効果】
【００２４】
本発明によれば、複数のコアで同一処理を実行することを抑制できるため、コアのリソースを有効に利用でき、また特別なテストプログラムが不要なため、演算負荷の少ないマルチコアシステムの故障診断が実現できる。

【発明を実施するための最良の形態】
【００２５】
以下、発明の実施の形態を説明する。
【実施例１】
【００２６】
本発明の第１の実施例を図面を用いて説明する。図２はこの発明の実施の形態を示した全体構成図であり、図において６はマイクロコンピュータを備えた制御コントロールユニット（Ｃ／Ｕ）、７はマイクロコンピュータ（ＣＰＵ）、８は読み取り専用記憶装置（ＲＯＭ）、９は読み書き可能記憶装置（ＲＡＭ）、１０は車両、１１は車両１０に備えられたセンサ、１２は車両１０に備えられたアクチュエータ、１３はセンサ１１，アクチュエータ１２の信号を送受する入出力装置（Ｉ／Ｏ）、である。
【００２７】
Ｃ／Ｕ６にはセンサ１１からの信号が入力される。Ｃ／Ｕ６は前記入力信号およびＲＯＭ８に内蔵された制御データを用いて演算を行い、アクチュエータ１２へ制御信号を出力する。
【００２８】
例えばＣ／Ｕ６を車載エンジン制御コントロールユニット、車両１０をガソリン噴射式内燃機関、センサ１１を吸入空気量センサおよびエンジン回転数センサ，アクチュエータ１２を燃料噴射弁とすると、ＣＰＵ７は前記吸入空気量センサおよびエンジン回転数センサからＩ／Ｏ１３を通して得た吸入空気量情報およびエンジン回転数情報をＲＯＭ８に内蔵された制御データと照合して燃料噴射量を算出し、該燃料噴射量に基づいて前記燃料噴射弁の開弁時間を制御する信号をＩ／Ｏ１３を通して前記燃料噴射弁に出力し、燃料噴射制御を行うシステムなどが挙げられる。
【００２９】
ここでＣＰＵ７の内部構成を図３に示す。ＣＰＵ７は２つ以上の中央演算処理装置（コア）で構成されるマルチコアＣＰＵであり、この図ではコアＡ～Ｄ（１４～１７）の４つのコアで構成される例を示している。ここではコアの数が４つの場合の例を示しているが、２つ以上であればいくつであってもよい。コアＡ～Ｄ（１４～１７）はそれぞれがデータバス１８と接続されている。データバス１８はＩ／Ｏポート１９，データ読み込みポート２０，データ書き込みポート２１と接続されており、外部のＩ／Ｏ１３，ＲＯＭ８，ＲＡＭ９とのデータ入出力を行う。なお、本実施例においては、一つのＣＰＵパッケージ内に複数コアを有するマルチコアＣＰＵを用いたシステムを挙げているが、このマルチコアＣＰＵに代えて、複数のＣＰＵパッケージを有するマルチプロセッサシステムを採用して

もよく、本発明においてコアとプロセッサの表記の差異は特別に区別しない。

【００３０】

次に動作について説明する。図４はＣＰＵ７のオペレーティングシステム（ＯＳ）の動作を示したフローチャートである。ＣＰＵ７の起動後、ステップ１では、関数Ａの処理を各コアに割り当てる処理を行う。ここで関数Ａは車両の制御に関わる演算処理手段の一例であり、本発明のマルチコアシステムの故障診断に用いる演算処理手段は、出力結果に一定の連続性が期待できるものであれば何でも良く、例えば前記の燃料噴射量を算出する関数などが挙げられる。なお、ここでいう連続性については後に具体例を用いて説明を行う。次に、ステップ２では、前記関数Ａの処理が含まれるタスクの実行を開始する。ステップ３では、タスク処理で判定された故障判定の結果を用いて処理の分岐を行う。正常判定（コア故障なし）の場合はタスク処理を続行し、再びステップ３に戻る。故障判定（コア故障あり）の場合は処理を終了する。

【００３１】

ここでステップ１の割り当て処理について図５を用いて詳細に説明する。図５はコアＡ～Ｄ（１４～１７）のそれぞれコアの処理を時間軸で表している。ここでタスクは例として１０ms周期とし、ある時刻Ｘを起点としてＸms後、Ｘ＋１０ms後、Ｘ＋２０ms後、Ｘ＋３０ms後と１０msずつ進む様子を示している。関数Ａ２２はステップ１の割り当て処理により、１０ms周期ごとに別のコアに割り当てる。図に示すようにＸms後のタスクではコアＡ１４に、Ｘ＋１０ms後のタスクではコアＢ１５に、Ｘ＋２０ms後のタスクではコアＣ１６に、Ｘ＋３０ms後のタスクではコアＤ１７に、それぞれ割り当てる。ここでは４つのコアの例を示しているが、コアは２つ以上あればいくつでもよい。またここでは１０ms周期毎に処理のタイミングを設定している例を示しているが、周期間隔はいくつであってもよく、例えばエンジン回転数同期のように不定期な周期に演算処理のタイミングを同期させてもよい。またここではコアＡ～Ｄ（１４～１７）を順番に割り当てる例を示しているが、この順番に限られるものではない。

【００３２】

ここでステップ３の故障判定について図を用いて詳細に説明する。図６は、各コアのタスクの１周期における処理において、故障判定に関する動作のフローチャートを示している。ステップ４では、関数Ａを実行する。ステップ５では、関数Ａを実行して得られた数値をメモリに記憶する。ここで記憶する領域は例えば図２に示したＲＡＭ９を用いるが、ＣＰＵ７に搭載されたキャッシュメモリなどがあればそちらを使用してもよい。ステップ６では、前回値と今回値の差分を算出する。前回値とは前回の周期で別のコアのステップ５の処理により記憶された値であり、今回値とはこのタスクにおいて直前のステップ５の処理により記憶された値である。ステップ７では、ステップ６により算出された差分の値を判定する。値が所定の範囲内であればステップ８へ進み、範囲外であればステップ９へと進む。ステップ８では、このコアは正常状態と判定する。ステップ９では、このコアは故障状態と判定する。

【００３３】

ここでステップ４からステップ９の動作についての具体例を説明する。関数Ａをガソリン噴射式内燃機関における燃料噴射パルス幅算出関数とすると、燃料噴射パルス幅Ｔｉは下式（数１）で算出できる。

【００３４】

〔数１〕

$$Ti = K \times Qa / Ne \times Coef + Ts$$

Ｔｉ：燃料噴射パルス幅（ms）

Ｋ：比例定数

Ｑａ：吸入空気量（kg／ｓ）

Ｎｅ：エンジン回転数（ｒ／min）

Ｃｏｅｆ：空燃比補正係数

Ｔｓ：無効噴射パルス幅（ms）

【００３５】
ここでＱａ（吸入空気量）は吸入空気量センサの入力信号から、Ｎｅ（エンジン回転数）はエンジン回転数センサの入力信号からそれぞれ求められ、Ｋ（比例定数），Ｃｏｅｆ（空燃比補正係数）およびＴｓ（無効噴射パルス幅）はＲＯＭ８にあらかじめ格納するものとする。図７にＱａ，Ｎｅの変化に対するＴｉの値をグラフで表したものを示す。Ｑａ，Ｎｅは車両の運動に応じて連続的に変化するため、Ｔｉも連続的に変化する。
【００３６】
ここで特定のコアに故障が発生し、当該コアの計算結果が０になってしまった場合のＴｉのグラフを図８に示す。例としてコアは全部で４つとし、故障コアはそのうちの１つとする。時刻ｔＸにおいて故障が発生したとすると、図のように時刻ｔｘ以降、断続的にグラフが途切れ、連続性を失った状態となる。これは図５で示したように周期毎に別のコアで計算を行うため、故障の発生したコアのみの計算結果が０となるためである。
【００３７】
ここでステップ６の処理により今回値と前回値との差分ΔＴｉを計算した結果を図９に示す。ΔＴｉはコア故障前の時刻ｔｘ以前では０点付近を示しているが、コア故障後の時刻ｔｘ以降では断続的に上下に大きく振幅する、連続性を欠いたグラフとなっている。コア正常時のΔＴｉの取りうる値の下限値ΔＴｉｍｉｎ，上限値ΔＴｉｍａｘを定めれば、ステップ７の故障判定処理において所定範囲をΔＴｉｍｉｎ～ΔＴｉｍａｘとすることで、範囲内であれば正常状態（ステップ８）、範囲外であれば、今回値が演算処理されたコアが、今回値が演算処理されたタイミングにおいて故障状態（ステップ９）である旨の故障判定を行うことができる。
【００３８】
また、コアに故障が発生した際に、当該コアの計算結果がランダムな値となる場合のＴｉのグラフを図１０に示す。このような場合でも時刻ｔｘ以降では所定範囲ΔＴｉｍｉｎ～ΔＴｉｍａｘを越えるΔＴｉが発生するため、故障判定を行うことができる。
【００３９】
なお、本実施例においては前回値と今回値の差分を算出することで、短い周期間隔の計算結果の変化から、計算結果の時間変化の連続性を監視している。しかし、本発明の要旨は上記態様に限られず、計算結果の時間変化に連続性があるかどうかを判断できる範囲で、コア故障判断のための計算結果の比較を行う周期を定めて良い。一例としては、今回値と前々回値の差分により故障判定を行うことも可能である。また、ラジエターの温度制御のように応答性が遅く、求められる演算周期間隔が長い制御は、計算結果の比較を行う周期間隔を長く定めても連続性の判断は可能である。しかし、上述の燃料噴射制御のように応答性の速い制御に関しては、短い周期間隔で計算結果の比較を行うことが望ましい。
【００４０】
また、本実施例において、計算結果の比較手段として二つの計算結果の差分を挙げているが、計算結果の時間変化に連続性があるかどうかを判断できる手段であれば任意の手段でよい。例えば、前回値と今回値とで除法を行い、その結果が所定範囲内かどうかを判定してもよく、また第一の計算結果，第二の計算結果，第三の計算結果の順で連続して演算処理された３つの値のうち、第二の計算結果が、第一と第三の計算結果との間の範囲に入る値であるかどうかを判定してもよい。
【００４１】
また、演算処理手段の割り当ては、計算結果の比較を行うコアが同一でなければ順番は不同であり、どのような順番であってもよい。また、演算負荷低減の観点から、同一の演算処理を同時に異なるコアでは実行しないことが望ましいが、設計上の都合により、演算処理の一部が複数のコアで同時に実行されてもよい。すなわち、あるコアで演算処理が終了する前に、次に同一の演算処理を開始しても、演算処理が重ならない時刻の演算負荷は低減できる。
【００４２】
また、本発明において利用できる演算処理手段は、計算結果の時間変化が厳密に連続性

を有するものでなくてもよく、実質上一定の連続性が期待できれば良い。例えば、断続的にＨ出力とＬ出力を繰り返すＰＷＭ（Pulse Width Modulation）出力であっても、出力値の積分値は連続性を持つため、出力には実質上一定の連続性が期待できる。

【００４３】

以上が第１の実施例である。このように本方式では、監視専用のコアを設ける必要がなく、すべてのコアを通常の処理に使用できるという利点がある。また処理に冗長性を持たせて複数のコアで同時に同一の処理を実行して故障判定を行う必要もなく、コアのリソースを有効に利用できるという利点がある。また、車両の制御信号は演算周期毎に急変する可能性が低いことを利用し、車両の制御に必要な演算処理をそのまま用いて故障判定を行うため、特別なテストプログラムが不要であり、故障判定のための演算負荷の増加を抑制できるという利点がある。また、多数決比較を行う場合には少なくとも３つ以上のプロセッサまたはコアが必要となるが、本実施例の構成によれば、コアの数は２つ以上であればいくつであってもよく、コア数の制限が少ないという利点がある。

【実施例２】

【００４４】

本発明の第２の実施例を説明する。第１の実施例と異なる点はステップ３の故障判定であり、この動作について図１１のフローチャートを用いて説明する。

【００４５】

ステップ４，ステップ５の動作は図６と同一である。ステップ５の次はステップ１０に進む。ステップ１０では、記憶しておいた移動平均値と今回値の差分を算出する。移動平均値については後述するステップ１１で説明する。ステップ７からステップ９の動作は図６と同一である。ステップ８にて正常判定とされた場合はステップ１１に進む。ステップ１１では、記憶しておいた移動平均値に今回値を加味して新しい移動平均値を算出する。ここで移動平均値の計算方法は単純移動平均でも加重移動平均でもよい。また移動平均値の計算に少なくとも２回分の値が必要であるため、ステップ７の判定はタスク周期の３回目以降から行う。

【００４６】

図１２および図１３を用いて本実施例の効果について示す。図１２は第１の実施例で説明した方式を用いてステップ６の処理により前回値との差分ΔＴｉを計算した結果のグラフである。ここでΔＴｉは関数Ａの結果、Ｔｉが振動状態となる事象を表している。例えば、スロットルバルブの劣化などで、Ｑａの制御が吸入空気量センサの入力信号に基づいて算出した所望の制御量の通り制御できない場合などが該当する。ここで所定範囲ΔＴｉｍｉｎ０～ΔＴｉｍａｘ０は振動の振幅の範囲が必要である。図１３は同じＴｉについて、本実施例のステップ１０の処理により前回値との差分ΔＴｉを移動平均値を用いて計算した結果のグラフである。このようにＴｉが振動状態であってもΔＴｉは収束するグラフとなり、所定範囲ΔＴｉｍｉｎ～ΔＴｉｍａｘは図１２の所定範囲ΔＴｉｍｉｎ０～ΔＴｉｍａｘ０と比べて狭い範囲で収まる。

【００４７】

本実施例の構成によれば、移動平均値を用いることで差分の変動幅を抑えることができ、故障判定の所定範囲を狭く設定することができる。したがって、故障判定できる領域が多くなり、故障判定の精度を高めることができるという利点がある。以上が第２の実施例である。

【実施例３】

【００４８】

本発明の請求項３に対応する第３の実施例を説明する。第２の実施例と異なる点はステップ３の故障判定であり、この動作について図１４のフローチャートを用いて説明する。

【００４９】

ステップ４，ステップ５の動作は図６と同一である。ステップ５の次はステップ１２に進む。ステップ１２では、記憶しておいた移動平均値からステップ７で使用する所定範囲の算出を行う。ここで算出方法は例えば下式（数２，数３）で求めることができる。

【００５０】
〔数２〕
　ΔＴｉｍｉｎ＝－α×｜移動平均値｜
〔数３〕
　ΔＴｉｍａｘ＝＋α×｜移動平均値｜
【００５１】
　ここでαは１以上の任意の係数であり、値を大きくすることで故障判定の誤判定を少なくすることができるが、その反面故障の検出精度が低くなるという性質を持つ。ここでは数２および数３の算出方式を例に挙げたが、移動平均値に基づいたものであればどのような算出方法であってもよい。ステップ１２の次はステップ１０に進む。以降のステップ１０からステップ１１までの動作は図１１と同一である。
【００５２】
　以上が第３の実施例である。このように本方式では、所定範囲をあらかじめ設定する必要がないため、計算結果の範囲の限定が難しい関数についても最適な所定範囲を得ることができるという利点がある。また数２および数３の算出方式を採用することで、係数の設定値により故障判定の誤判定の度合いと故障の検出精度とを任意に選択できるという利点がある。
【実施例４】
【００５３】
　本発明の第４の実施例を説明する。第１から第３の実施例と異なる点はステップ３の故障判定であり、この動作について図１５のフローチャートを用いて説明する。
【００５４】
　ステップ４からステップ７の動作は図６と同一である。ステップ７では、差分の値を判定する。値が所定の範囲内であればステップ８へ進み、範囲外であればステップ１２へと進む。
【００５５】
　ステップ１２では、故障カウンタを１つ増分する。ステップ１３では、故障カウンタが所定回数を越えたかどうかを判定する。所定回数を越えない場合はステップ８へと進み、所定回数を越えた場合はステップ９へと進む。ステップ８およびステップ９の動作は図６と同一である。
【００５６】
　ここで図１５は第１の実施例の図６を元にして作成したが、第２の実施例の図１１および第３の実施例の図１４を元にした場合でも同様である。
【００５７】
　以上が第４の実施例である。このように本方式では、ステップ７で差分の値を判定した値が所定の範囲外であっても即座に故障とは判定せず、故障カウンタを設けて判定することで、故障の誤判定を抑制できる利点がある。また故障が一時的なものであり、継続して使用できるような場合にも対応できるという利点がある。
【実施例５】
【００５８】
　本発明の第５の実施例を説明する。第１から第４の実施例と異なる点はＣＰＵ７のＯＳの動作であり、この動作について図１６のフローチャートを用いて説明する。
【００５９】
　ステップ１からステップ３の動作は図４と同一である。ステップ３で正常判定（コア故障なし）の場合はタスク処理を続行し、再びステップ３に戻る。故障判定（コア故障あり）の場合はステップ１４に進む。
【００６０】
　ステップ１４ではタスクの実行を一時中断する。ステップ１５ではステップ３で故障と判定されたコアを除いた残りのコアに関数Ａの処理を割り当てる処理を行う。ステップ１５の次はステップ２に戻り、タスクの実行を再開する。

【００６１】
ここでステップ１５の再割り当て処理について図１７を用いて詳細に説明する。図１７はコアＡ～Ｄ（１４～１７）のそれぞれコアの処理を時間軸で表している。図５との相違点について以下説明する。この例ではＸ＋２０ms後の時点でコアＣ１６に故障が発生した場合を示している。このときまずステップ１４によりタスクの実行を一時中断する。次にステップ１５の再割り当て処理により、関数Ａの再割り当てを行う。ここでは故障の発生したコアＣ１６を除いた残りのコアＡ１４，コアＢ１５，コアＤ１７に関数Ａの処理を割り当てる処理を行う。ここで再割り当て処理は関数Ａだけでなく、コアＣ１６が行っていたすべての処理を残りのコアに割り当ててもよい。再割り当て処理のあとタスクの実行を再開する。

【００６２】
以上が第５の実施例である。このように本方式では、任意のコアに故障が発生した場合でも当該コアを除いた残りのコアで処理を継続するため、故障判定および制御処理を正常に続行できるという利点がある。

【実施例６】

【００６３】
本発明の第６の実施例を説明する。第１から第５の実施例と異なる点はステップ３の故障判定であり、この動作について図１８および図１９のフローチャートを用いて説明する。

【００６４】
図１８のステップ４，ステップ５の動作は図６と同一である。ステップ５の次はステップ１６に進む。ステップ１６では、前回の演算周期で正常と判定されたコアに割り込みイベントを発行し、この判定処理に関してこのコアでの処理は終了する。

【００６５】
図１９はステップ１６の割り込みイベントを受けたコアでの動作を示している。ステップ６では割り込みイベントを受けた当該コアで算出された前回値と、割り込みイベントを発行したコアで算出された今回値の差分を算出する。ステップ７からステップ９までの処理は、割り込みイベントを受けた当該コアで行うことを除いては図６と同一である。

【００６６】
以上の動作についてコアＡ～Ｄ（１４～１７）のそれぞれコアの処理を時間軸で表したものが図２０である。ここで故障判定処理２３は図１９の処理を表している。Xms後を例にとると、コアＡ１４にて図１８のステップ４により関数Ａの処理を行い、ステップ５を経てステップ１６にて前回の演算周期で正常と判定されたコアＤ１７に割り込みイベントを発行する。割り込みイベントを受けたコアＤ１７では図１９のステップ６からステップ９で表される故障判定処理２３の処理を行う。

【００６７】
以上が第６の実施例である。このように故障判定処理２３を前回の演算周期で正常と判定されたコアで行うことにより、異常が発生したコアに他のコアの故障判定処理２３を割り振ってしまう事態を抑制できる。例えば、コアに故障が発生してステップ６の差分計算に異常が発生し、本来所定範囲外となるはずの差分結果が所定範囲内に入ってしまう現象や、ステップ７の判定処理において本来故障と判定されるべきところが正常と判定されてしまう現象を抑制できるという利点がある。

【００６８】
ここでは割り込みイベントを用いて故障判定処理２３の処理を行う例を示したが、前回の演算周期で正常と判定されたコアで故障判定処理２３を行う方法であれば実現手段は何でもよく、特に割り込みイベントが必須というわけではない。また、ステップ６からステップ９の処理のいずれかのみを前回の演算周期で正常と判定されたコアで行ってもよい。

【実施例７】

【００６９】
本発明の第７の実施例を説明する。第１から第５の実施例と異なる点はＣＰＵ７のＯＳ

の動作およびステップ３の故障判定であり、この動作について図２１および図２２のフローチャートを用いて説明する。

【００７０】

図２１のステップ１からステップ２の動作は図４と同一である。ステップ２の次はステップ１７を実行する。ステップ１７では、少なくとも３つ以上の奇数のコアで重複して実施されたコア故障判定の結果を判定する。ここで正常判定（コア故障なし）とされたコアの数が故障判定（コア故障あり）とされたコアの数より少ない場合は正常状態とみなしてタスク処理を続行し、再びステップ１７に戻る。それ以外の場合は故障状態とみなして処理を終了する。あるいは第５の実施例の図１６のように再割り当て処理を行ってもよい。

【００７１】

ステップ１７の故障判定について図２２のフローチャートを用いて説明する。ステップ４，ステップ５の動作は図６と同一である。ステップ５の次はステップ１８に進む。ステップ１８では、少なくとも３つ以上の奇数のコアに割り込みイベントを発行し、この判定処理に関してこのコアでの処理は終了する。割り込みイベントを受けた各コアは、図１９で示される故障判定処理を行う。

【００７２】

以上の動作についてコアＡ～Ｄ（１４～１７）のそれぞれコアの処理を時間軸で表したものが図２３である。ここで故障判定処理２３は図１９の処理を表している。Ｘms後を例にとると、コアＡ１４にて図１８のステップ４により関数Ａの処理を行い、ステップ５を経てステップ１８にて少なくとも３つ以上の奇数のコアに割り込みイベントを発行する。割り込みイベントを受けた各コアでは図１９のステップ６からステップ９で表される故障判定処理２３の処理を行う。

【００７３】

以上が第７の実施例である。このように演算負荷の少ない故障判定処理２３の処理を、少なくとも３つ以上の奇数のコアで行い、それらの結果を多数決にて判定することにより、演算負荷の増大を抑えながら、故障判定の信頼性を向上することができるという利点がある。

【００７４】

ここでは割り込みイベントを用いて故障判定処理２３の処理を行う例を示したが、少なくとも３つ以上の奇数のコアで故障判定処理２３を行う方法であれば実現手段は何でもよく、特に割り込みイベントが必須というわけではない。

【００７５】

また、ステップ６からステップ９の処理のいずれかのみを少なくとも３つ以上の奇数のコアで行うよう構成することで、故障判定の一部分の信頼性を向上することもできる。

【実施例８】

【００７６】

本発明の第８の実施例を説明する。第１から第５の実施例と異なる点はＣＰＵ７のＯＳの動作およびステップ３の故障判定であり、この動作について図２４および図２５のフローチャートを用いて説明する。

【００７７】

図２４のステップ１からステップ２の動作は図４と同一である。ステップ２の次はステップ１９を実行する。ステップ１９では、少なくとも２つ以上のコアで重複して実施されたコア故障判定の結果を重み付けして集計する。重み付け計算方法は例えば下式（数４）が挙げられる。

【００７８】

〔数４〕

$$R = W1 \times C1 + W2 \times C2 + \cdots + Wn \times Cn$$

Ｒ：集計値

Ｗｎ：ｎ番目のコアに対する重み付け（０～１）

Ｃｎ：ｎ番目のコアの判定結果値。正常判定時＝０，故障判定時＝１

【００７９】
ここで重み付けは、各コアの処理内容の重要度によって定めてもよい。あるいは各コアが正常と判断されてからの経過時間によって定めてもよい。また重み付け計算方法は数４に限定されるものではなく、任意の計算方法を用いてもよい。
【００８０】
ステップ２０では、ステップ１９で計算された集計結果を用いて故障判定を行う。例えばステップ１９で前述の数４によって集計された場合、集計値Ｒが所定値以上であるときに故障と判定する方法が挙げられる。正常状態と判定された場合はタスク処理を続行し、再びステップ１９に戻る。それ以外の場合は故障状態とみなして処理を終了する。あるいは第５の実施例の図１６のように再割り当て処理を行ってもよい。
【００８１】
ステップ１９の故障判定について図２５のフローチャートを用いて説明する。ステップ４，ステップ５の動作は図６と同一である。ステップ５の次はステップ２１に進む。ステップ２１では、少なくとも２つ以上のコアに割り込みイベントを発行し、この判定処理に関してこのコアでの処理は終了する。割り込みイベントを受けた各コアは、図１９で示される故障判定処理を行う。
【００８２】
以上の動作についてコアＡ～Ｄ（１４～１７）のそれぞれコアの処理を時間軸で表したものが図２３である。ここで故障判定処理２３は図１９の処理を表している。Ｘms後を例にとると、コアＡ１４にて図１８のステップ４により関数Ａの処理を行い、ステップ５を経てステップ１８にて少なくとも２つ以上のコアに割り込みイベントを発行する。割り込みイベントを受けた各コアでは図１９のステップ６からステップ９で表される故障判定処理２３の処理を行う。
【００８３】
以上が第８の実施例である。このように故障判定処理２３を少なくとも２つ以上のコアで行い、それらの結果を重み付けして集計して判定することにより、故障判定の信頼性を向上することができるという利点がある。またコア毎に性質や重要度が異なる場合にもその性質や重要度を加味した最適な故障判断を行うことができるという利点がある。またコアの数は２つ以上であればいくつでもよく、コア数の制限を受けないという利点がある。
【００８４】
ここでは割り込みイベントを用いて故障判定処理２３の処理を行う例を示したが、少なくとも２つ以上のコアで故障判定処理２３を行う方法であれば実現手段は何でもよく、特に割り込みイベントが必須というわけではない。
【実施例９】
【００８５】
本発明の第９の実施例を説明する。第７または第８の実施例と異なる点はＣＰＵ７のＯＳの動作であり、この動作について図２６のフローチャートを用いて説明する。
【００８６】
図２６のステップ１からステップ２の動作は図４と同一である。ステップ２の次はステップ２２を実行する。ステップ２２では、少なくとも３つ以上のコアで重複して実施されたコア故障判定の結果、１つのコアだけが故障と判定し他のコアが正常と判定した場合はステップ２３へと進む。それ以外の場合はステップ２４へと進む。ステップ２３では、ステップ２２にて故障判定したコア（１つのコア）が故障状態であると判断する。
【００８７】
ステップ２４では、ステップ２２と同様に少なくとも３つ以上のコアで重複して実施されたコア故障判定の結果、１つのコアだけが正常と判定し他のコアが故障と判定した場合は、ステップ２５へと進む。それ以外の場合はステップ２６へと進む。ステップ２５では、ステップ２４にて正常判定したコア（１つのコア）と今回の演算周期において演算したコアの両方が故障状態であると判断する。
【００８８】

ステップ２６およびステップ２７は、第７の実施例の図２１のステップ１７に相当する多数決の故障判定処理、または第８の実施例の図２４のステップ１９およびステップ２０に相当する各コアの重み付けの集計による故障判定処理である。ステップ２７で正常と判定された場合はタスク処理を続行し、再びステップ２２に戻る。それ以外の場合は今回の演算周期において演算したコアが故障状態とみなして処理を終了する。あるいは第５の実施例の図１６のように再割り当て処理を行ってもよい。

【００８９】

以上が第９の実施例である。このように故障判定処理２３を少なくとも３つ以上のコアで行い、その結果１つのコアだけが故障と判定し他のコアが正常と判定した場合は、今回の演算周期において演算したコアには異常がなく、別のコア（１つのだけが故障と判定したコア）に故障が発生していると判断することで、故障の誤判定を低減することができ、かつ今回の演算周期において演算したコア以外のコアの故障を検出できるという利点がある。また１つのコアだけが正常と判定し他のコアが故障と判定した場合は、正常と判定したコアと今回の演算周期において演算したコアの両方に故障が生じていると判断することで、同時に２つのコアの故障を検出できるという利点がある。

【実施例１０】

【００９０】

本発明の第１０の実施例を説明する。第１から第９の実施例と異なる点はＣＰＵ７のＯＳの動作であり、この動作について図２７のフローチャートを用いて説明する。

【００９１】

図２７のステップ１からステップ２の動作は図４と同一である。ステップ２の次はステップ２９を実行する。ステップ２９では、車両の運転モードの切り替えが発生したかどうかを判定する。ここで運転モードとは車両の状態を規定するものであり、例えばガソリン噴射式内燃機関における機関停止状態，始動状態，完爆状態，セルフシャットオフ状態が挙げられる。この運転モードは故障の判定に使用する関数Ａの演算に影響する可能性があるものであれば何でもよく、故障判定のために新たに規定したものであってもよい。ステップ２９で運転モードの切り替えが発生した場合はステップ３０へ進み、発生していない場合はステップ３１へと進む。ステップ３０では、運転モードの切り替え発生から所定時間が経過したかどうかを判定する。所定時間が経過している場合はステップ３１へ進み、経過していない場合は再びステップ３０に戻る。ステップ３１の動作は図４のステップ３と同様であり、正常判定（コア故障なし）の場合はタスク処理を続行し、ステップ２９に戻る。故障判定（コア故障あり）の場合は処理を終了する。あるいは第５の実施例の図１６のように再割り当て処理を行ってもよい。

【００９２】

以上が第１０の実施例である。運転モードが変更されたときは車両の制御が大きく変動する特異点であることが多く、故障の判定に使用する関数Ａの計算結果が前回値と比べて大きく変わり、連続性を失う場合が多くなる。このような場合に差分を用いた故障判定を行うと前回値との差分ΔＴｉが想定した所定範囲ΔＴｉｍｉｎ～ΔＴｉｍａｘを越えることがあり、故障状態と誤判定される可能性がある。本実施例はこのような運転状態が急変する事態において所定時間の故障判定休止期間を設けることで、故障の誤判定を抑制できる利点がある。

【実施例１１】

【００９３】

本発明の第１１の実施例を説明する。第１から第１０の実施例と異なる点はステップ３の故障判定であり、この動作について図２８のフローチャートを用いて説明する。

【００９４】

ステップ４，ステップ５の動作は図６と同一である。ステップ５の次はステップ３２に進む。ステップ３２では、関数Ａに使用する少なくとも１つ以上の入力値のいずれかまたは複数の前回値と今回値の差分を算出する。ステップ３３では、ステップ３２により算出された差分の値を判定する。値が所定の範囲内であればステップ６へ進み、範囲外であれ

ばステップ３４へと進む。ここで複数の入力値について判定を行う場合は、例えばそれらのすべてが範囲内であればステップ６へと進み、それ以外であればステップ３４へと進む処理が挙げられるが、他の方法として入力値の重要度によって重み付けをして判定処理を行ってもよい。

【００９５】

ステップ６からステップ９の動作は図６と同一であり、ステップ８では、このコアは正常状態と判定する。ステップ９では、このコアは故障状態と判定する。

【００９６】

ステップ３４では、このコアは正常状態と判定する。ここで正常状態の代わりに「故障判定中」という状態を規定してもよい。

【００９７】

また、ステップ３２からステップ３４の入力値の判定処理はこの例ではステップ４の関数Ａの処理と同一のコアのタスクで行っているが、第６の実施例で示したように別のコアで行ってもよい。

【００９８】

以上が第１１の実施例である。入力値が大きく変化した場合は関数Ａの実行結果も大きく変動することがあるが、この変動が第１の実施例で示した故障判定で述べた、所定範囲ΔＴｉｍｉｎ～ΔＴｉｍａｘを越えるものであった場合、故障と誤判定することになる。これを防ぐにはΔＴｉｍｉｎ～ΔＴｉｍａｘを広く取ることが必要であるが、それによって故障の検出精度が低下するという欠点がある。そこで本実施例のように、入力値が大きく変化した場合は故障判定を行わないようにすることで、故障の検出精度を保ったままで誤判定を抑制できるという利点がある。また特に誤判定を引き起こしやすい入力値を選択することや、入力値に重み付けを行うことで、より効果的に誤判定を抑制できるという利点がある。

【実施例１２】

【００９９】

本発明の第１２の実施例を説明する。第１１の実施例と異なる点はステップ３の故障判定であり、この動作について図２９のフローチャートを用いて説明する。

【０１００】

ステップ４，ステップ５の動作は図６と同一である。ステップ５の次はステップ３２に進む。ステップ３５では、関数Ａに使用する少なくとも１つ以上の入力値のいずれかまたは複数の前回までの移動平均値と今回値の差分を算出する。ステップ３６では、前回までの移動平均値に今回値を加味して新しい移動平均値を算出する。ここで移動平均値の計算方法は単純移動平均でも加重移動平均でもよい。ステップ３３以降の動作は図２８と同一である。また移動平均値の計算に少なくとも２回分の値が必要であるため、ステップ３３の判定はタスク周期の３回目以降から行う。

【０１０１】

以上が第１２の実施例である。第１１の実施例では入力値が振動状態のように変動する場合、その前回値と今回値との差分は第２の実施例の図１２で示したΔＴｉのグラフと同様に０点を挟んで上下に振動するグラフとなる。このような場合、故障判定の機会を増やすためには入力値の判定に用いる所定範囲を広く取る必要があるが、その場合入力値が所定範囲内で大きく変化すると関数Ａの実行結果も大きく変動し、正常状態であるにもかかわらず故障と誤判定する可能性がある。そこで本実施例のように移動平均値を用いることにより、第１１の実施例の図１３と同様に差分の振幅が小さくなるため、入力値の判定に用いる所定範囲を狭くすることが可能になる。これにより故障判定の機会を減らさずに、かつ入力値が急変動しても故障と誤判定するのを抑制することができるという利点がある。

【実施例１３】

【０１０２】

本発明の第１３の実施例を説明する。第１２の実施例と異なる点はステップ３の故障判

定であり、この動作について図３０のフローチャートを用いて説明する。

【０１０３】

ステップ４，ステップ５の動作は図６と同一である。ステップ５の次はステップ３７に進む。ステップ３７では、記憶しておいた入力値の移動平均値からステップ７で使用する所定範囲の算出を行う。ここで算出方法は例えば下式（数５，数６）で求めることができる。

【０１０４】

〔数５〕

ΔＩＡＭＩＮｎ＝－αｉａｎ×｜ＩＡｎ｜

〔数６〕

ΔＩＡＭＡＸｎ＝＋αｉａｎ×｜ＩＡｎ｜

ΔＩＡＭＩＮｎ：ｎ番目の入力値の判定用範囲の下限値

ΔＩＡＭＡＸｎ：ｎ番目の入力値の判定用範囲の上限値

αｉａｎ：ｎ番目の入力値の判定用範囲算出係数

ＩＡｎ：ｎ番目の入力値の移動平均値

【０１０５】

ここでαｉａｎは１以上の任意の係数であり、値を大きくすることで故障判定の機会を増やすことができるが、その反面故障と誤判定する可能性も増加する。ここでは数５および数６の算出方式を例に挙げたが、移動平均値に基づいたものであればどのような算出方法であってもよい。ステップ３７の次はステップ３５に進む。ステップ３５以降の動作は図２９と同一である。

【０１０６】

以上が第１３の実施例である。このように本方式では、入力値判定用の所定範囲をあらかじめ設定する必要がないため、入力値の取りうる範囲の限定が難しい場合においても最適な所定範囲を得ることができるという利点がある。また数５および数６の算出方式を採用することで、係数の設定値により故障判定の誤判定の度合いと故障判定の機会の頻度とを任意に選択できるという利点がある。

【図面の簡単な説明】

【０１０７】

【図１】本発明の基本構成図。

【図２】発明の実施の形態を示した全体構成図。

【図３】ＣＰＵの内部構成を示した図。

【図４】オペレーティングシステム（ＯＳ）の動作を示したフローチャート。

【図５】関数Ａの割り当てと各コアの処理を時間軸で示した図。

【図６】各コアのタスクの１周期における故障判定に関する動作のフローチャート。

【図７】時刻ｔに対するＱａ，Ｎｅ，Ｔｉのそれぞれの変化を表したグラフ。

【図８】コアに故障が発生した場合のＴｉのグラフ。

【図９】前回値との差分ΔＴｉのグラフ。

【図１０】コアに故障が発生した際に、当該コアの計算結果がランダムな値となる場合のＴｉのグラフ。

【図１１】実施例２における各コアのタスクの１周期における故障判定に関する動作のフローチャート。

【図１２】Ｔｉが振動状態となる場合の前回値との差分ΔＴｉのグラフ。

【図１３】Ｔｉが振動状態となる場合の前回値に移動平均値を用いた差分ΔＴｉのグラフ。

【図１４】実施例３における各コアのタスクの１周期における故障判定に関する動作のフローチャート。

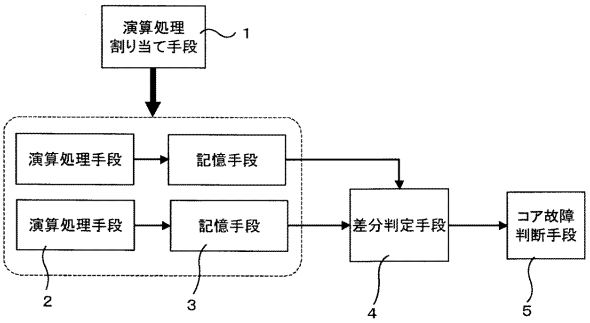
【図１５】実施例４における各コアのタスクの１周期における故障判定に関する動作のフローチャート。

【図１６】実施例５におけるオペレーティングシステム（ＯＳ）の動作を示したフローチ

ャート。
【図１７】実施例５における関数Ａの割り当てと各コアの処理を時間軸で示した図。
【図１８】実施例６における各コアのタスクの１周期における故障判定に関する動作のフローチャート。
【図１９】実施例６における割り込みイベントを受けたコアの故障判定に関する動作のフローチャート。
【図２０】実施例６における各コアの処理を時間軸で示した図。
【図２１】実施例７におけるオペレーティングシステム（ＯＳ）の動作を示したフローチャート。
【図２２】実施例７における各コアのタスクの１周期における故障判定に関する動作のフローチャート。
【図２３】実施例７における各コアの処理を時間軸で示した図。
【図２４】実施例８におけるオペレーティングシステム（ＯＳ）の動作を示したフローチャート。
【図２５】実施例８における各コアのタスクの１周期における故障判定に関する動作のフローチャート。
【図２６】実施例９におけるオペレーティングシステム（ＯＳ）の動作を示したフローチャート
【図２７】実施例１０におけるオペレーティングシステム（ＯＳ）の動作を示したフローチャート。
【図２８】実施例１１における各コアのタスクの１周期における故障判定に関する動作のフローチャート。
【図２９】実施例１２における各コアのタスクの１周期における故障判定に関する動作のフローチャート。
【図３０】実施例１３における各コアのタスクの１周期における故障判定に関する動作のフローチャート。
【符号の説明】
【０１０８】
１　演算処理割り当て手段
２　演算処理手段
３　記憶手段
４　差分判定手段
５　コア故障判断手段
６　制御コントロールユニット（Ｃ／Ｕ）
７　マイクロコンピュータ（ＣＰＵ）
８　読み取り専用記憶装置（ＲＯＭ）
９　読み書き可能記憶装置（ＲＡＭ）
１０　車両
１１　車両に備えられたセンサ
１２　車両に備えられたアクチュエータ
１３　入出力装置（Ｉ／Ｏ）
１４～１７　コアＡ～Ｄ
１８　データバス
１９　Ｉ／Ｏポート
２０　データ読み込みポート
２１　データ書き込みポート
２２　車両の制御に関わる処理関数Ａ
２３　故障判定処理

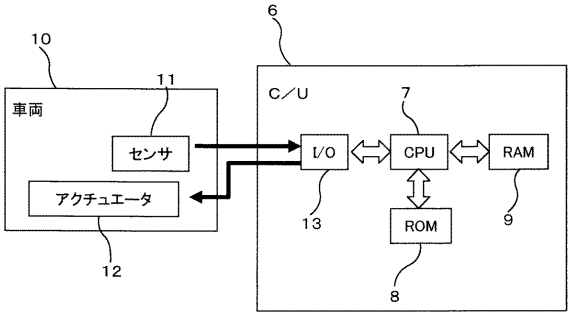
【図１】

図 1



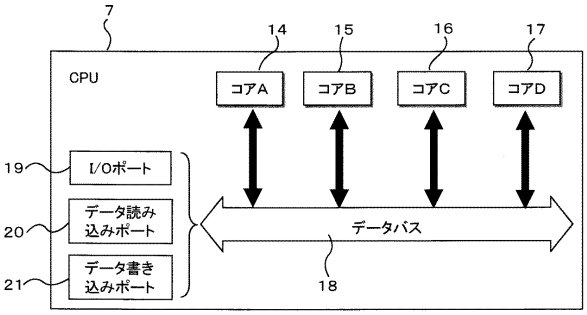
【図２】

図 2



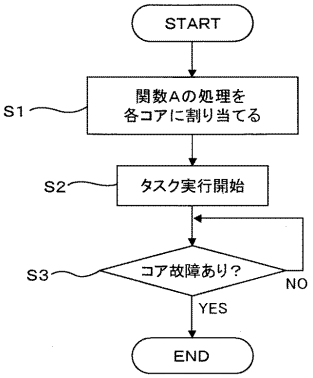
【図３】

図 3



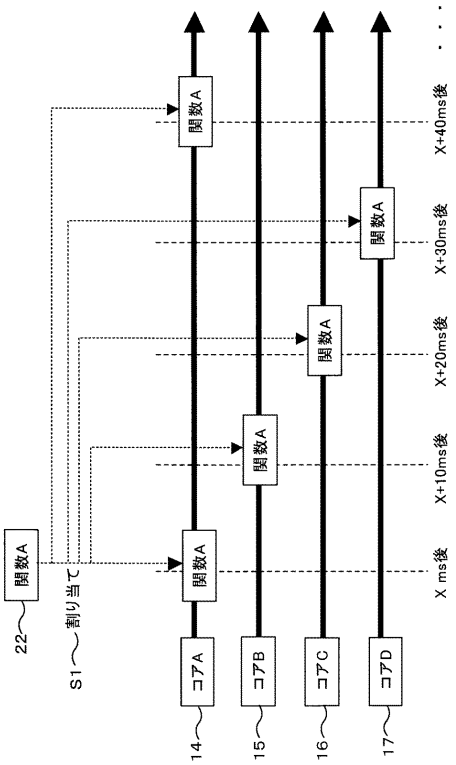
【図４】

図 4



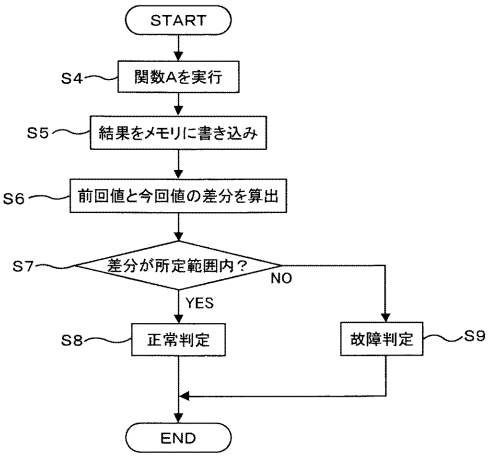
【図５】

図 5



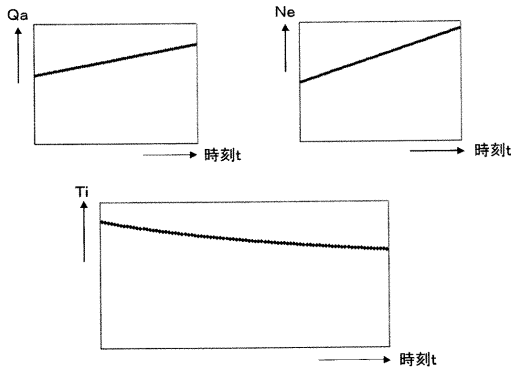
【図６】

図 6



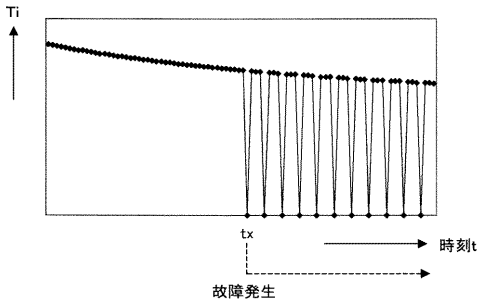
【図７】

図 7



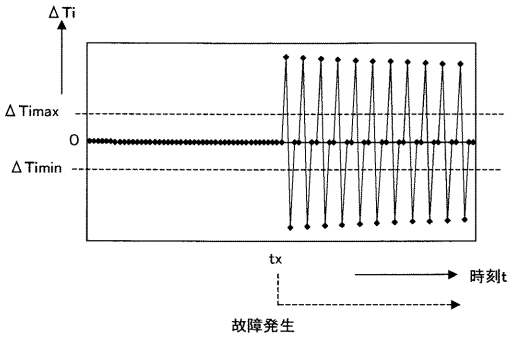
【図８】

図 8



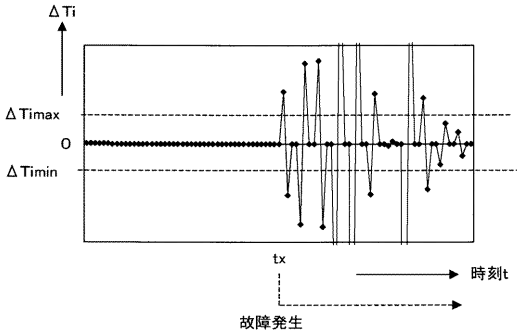
【図９】

図 9



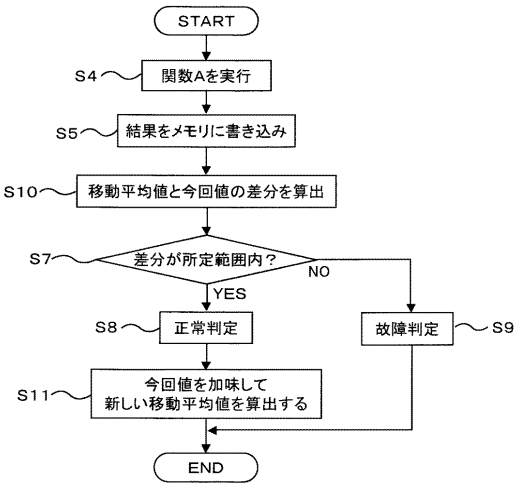
【図１０】

図 10



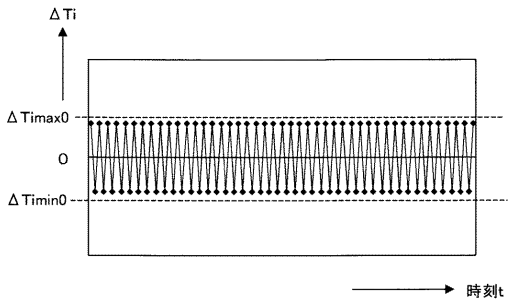
【図１１】

図 11



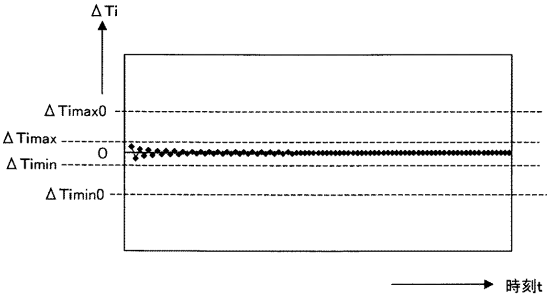
【図１２】

図 12



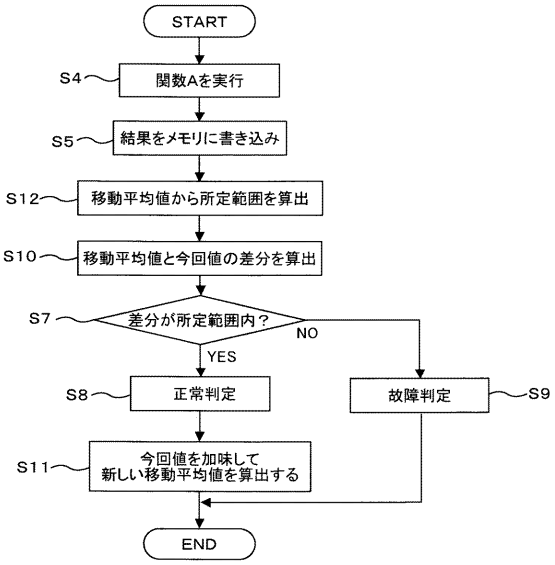
【図１３】

図 13



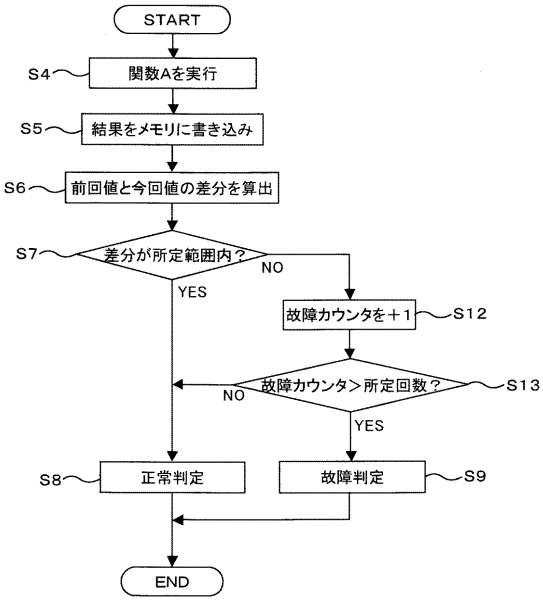
【図１４】

図 14



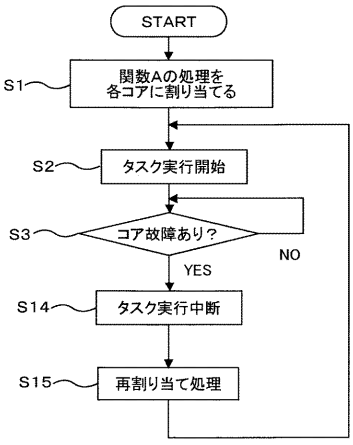
【図１５】

図 15



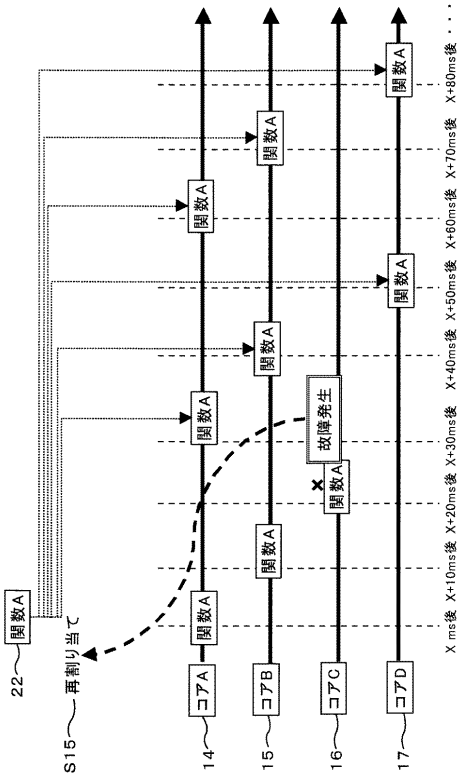
【図１６】

図 16



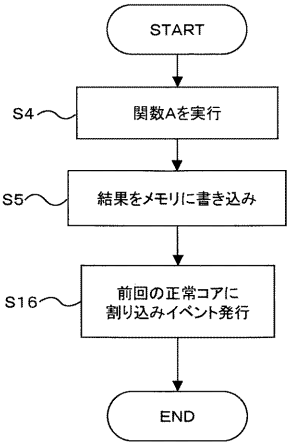
【図１７】

図 17



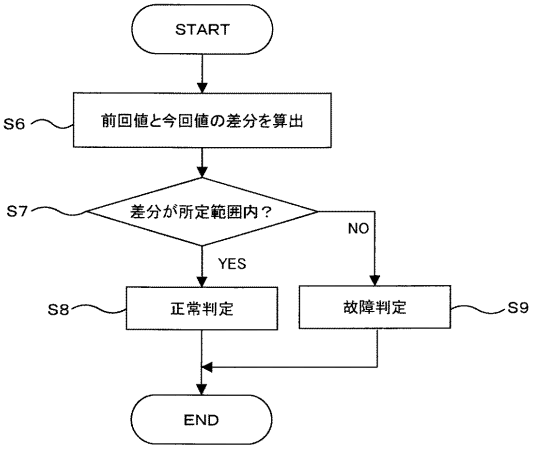
【図１８】

図 18



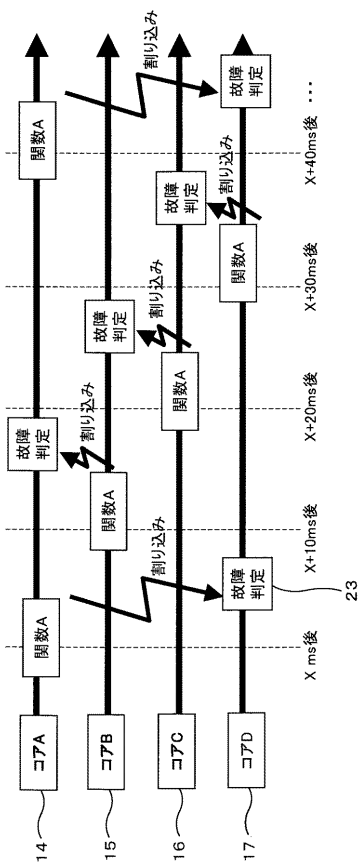
【図１９】

図 19



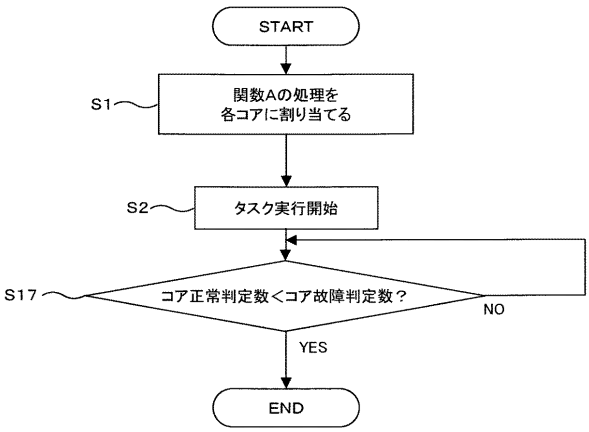
【図２０】

図 20



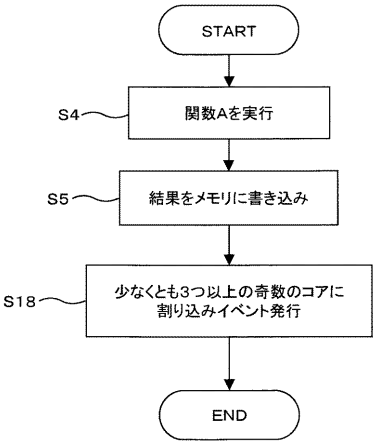
【図２１】

図 21



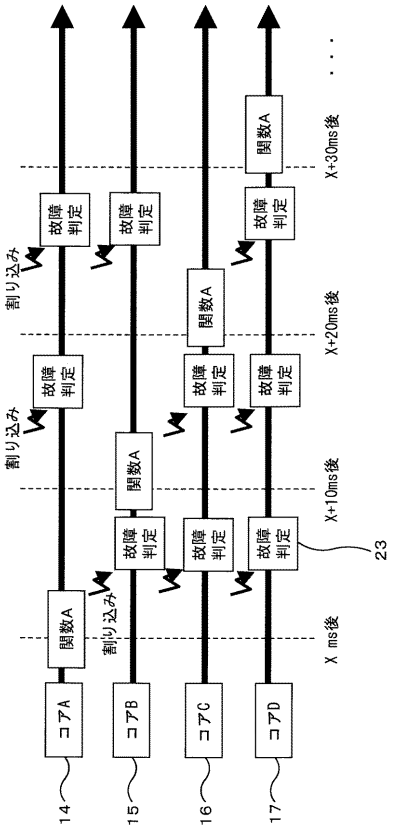
【図２２】

図 22



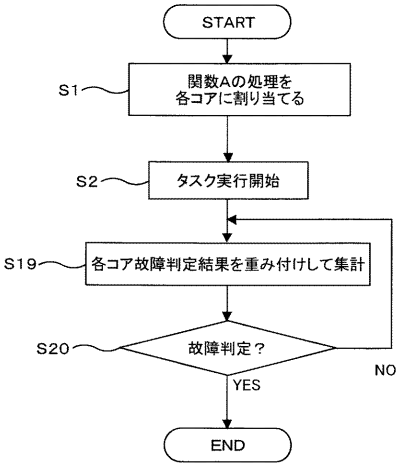
【図２３】

図 23



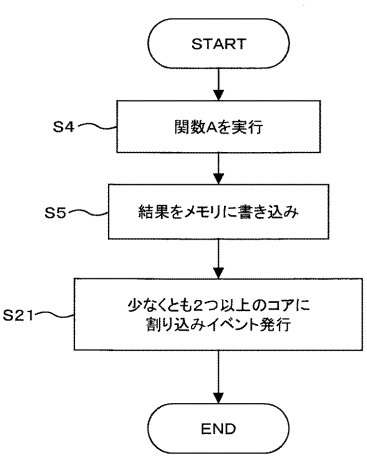
【図２４】

図 24



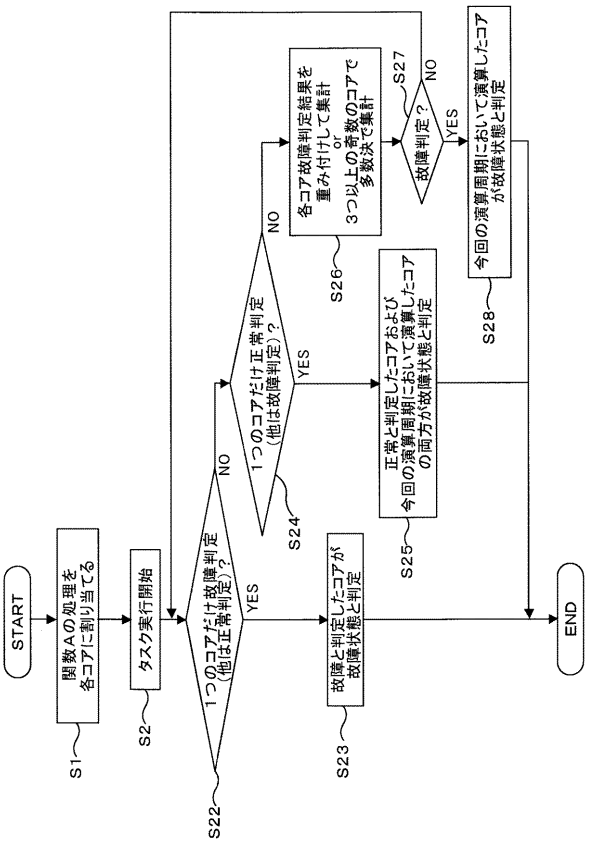
【図２５】

図 25



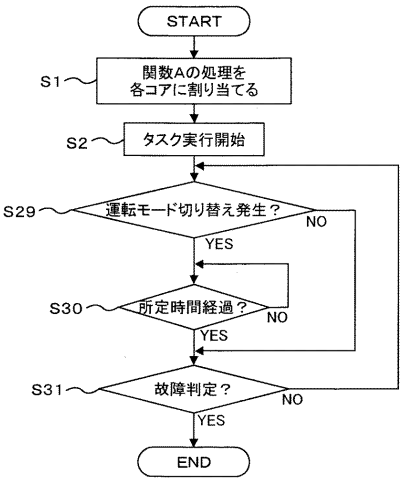
【図２６】

図 26



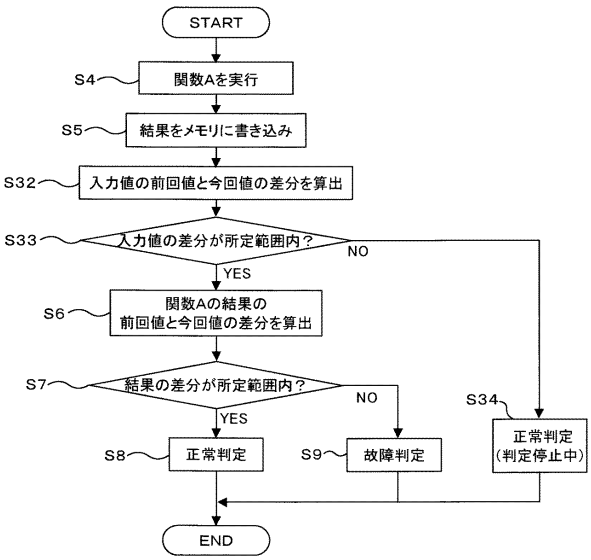
【図２７】

図 27



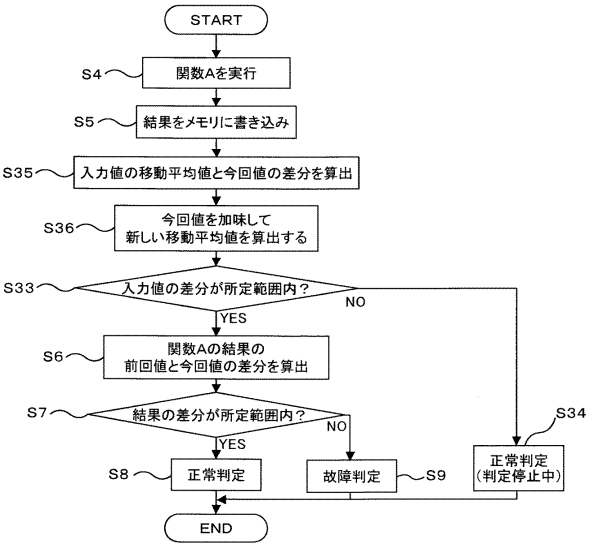
【図２８】

図 28



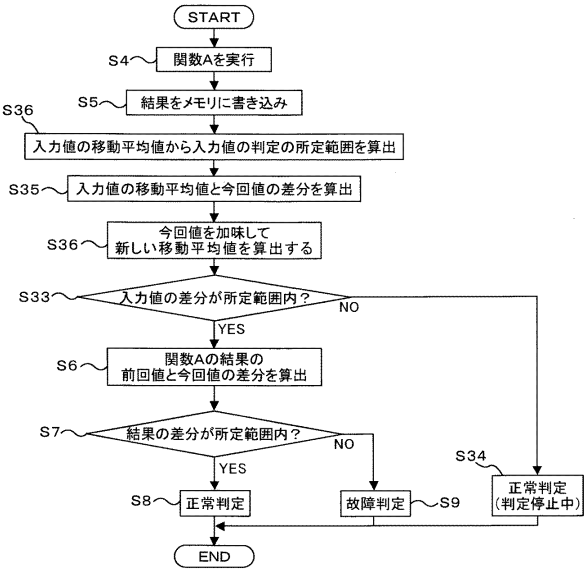
【図２９】

図 29



【図３０】

図 30


START
S4
関数Aを実行
S5
結果をメモリに書き込み
S36
入力値の移動平均値から入力値の判定の所定範囲を算出
S35
入力値の移動平均値と今回値の差分を算出
S36
今回値を加味して
新しい移動平均値を算出する
S33
入力値の差分が所定範囲内？
NO
YES
S6
関数Aの結果の
前回値と今回値の差分を算出
S7
結果の差分が所定範囲内？
NO
YES
S34
正常判定
（判定停止中）
S8
正常判定
故障判定
S9
END

フロントページの続き

(56)参考文献 特表２００８－５１８２９６（ＪＰ，Ａ）
特開２００６－２６０５６８（ＪＰ，Ａ）
特開２００８－０１０９５９（ＪＰ，Ａ）
特開２００６－１６４２７７（ＪＰ，Ａ）
特開２００８－２５７５８９（ＪＰ，Ａ）
特表２００９－５１６２７７（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｂ６０Ｒ １６／０２
Ｆ０２Ｄ ４５／００
Ｇ０６Ｆ １１／１８