論　文

無線センサノードのためのハードリアルタイム保証が可能な仮想マシン

鈴木　誠[†]　猿渡　俊介[††]　南　正輝[††]　森川　博之[††]

A Hard Real-time Virtual Machine for Wireless Sensor Nodes

Makoto SUZUKI[†], Shunsuke SARUWATARI[††], Masateru MINAMI[††], and Hiroyuki MORIKAWA[††]

あらまし　無線センサネットワークにおいては広範囲に分散配置したセンサノードのソフトウェア保守が重要となる．これまで，仮想マシンを用いてソフトウェアモジュールを配信するソフトウェア保守手法が提案されてきたが，実際的なアプリケーションで必要となるハードリアルタイム性はサポートされてこなかった．このような観点から本論文では，ソフトウェアモジュールの小型化と十分な保護機能を実現しつつ，ハードリアルタイム性を保証する無線センサノード向けの仮想マシン VAWS を設計する．VAWS はセンサネットワークで一般的に用いられる小型マイコンに搭載されている割り込み機能に着目し，割り込みベクタ毎に独立した仮想マシンを実行することで，保護機能とハードリアルタイム性の両立を実現する．実装評価を通じ，VAWS は 100 Hz のサンプリングにおいてジッタを 1 us 以内に抑えられる性能を実現できることが示される．

キーワード　無線センサネットワーク，ソフトウェアモジュール，リアルタイム処理，保護，仮想マシン

1. まえがき

無線センサネットワークは無線によって相互接続されたセンサを空間に埋め込むことで実空間の情報を細かい粒度で低コストに取得することを可能とする．一度散布されたセンサノードは数が多く回収が困難であるため，これまでは主として低電力消費化によるネットワークの長寿命化に関する研究が行われてきた．しかしながら，徹底した低消費電力化によるネットワークの長寿命化が実現されるにつれて，センサノードに搭載されるソフトウェア保守が新たな問題となりつつある．

これに向けては，無線センサノードのソフトウェアをモノリシックなバイナリイメージをすべて交換することなく，部分的に修正可能とするソフトウェアモジュール機構が必要である．ソフトウェアモジュール機構は無線センサノード上のソフトウェアをモジュール単位で管理しており，モジュールを無線ネットワーク経由でノードに配信することによって新しい機能を追加したり，ソフトウェアに含まれる不具合を修正することを可能とする仕組である．

無線センサネットワークに適したソフトウェアモジュール機構を実現する場合，低消費電力化を目的としたモジュールの小型化，CPU リソースの制約下でのシステムの保護の実現と実行性能の確保が課題となる．なお，システムの保護とはモジュールの不具合を検知してその実行を安全に停止させる機能を指す．

ソフトウェアモジュールのサイズを小さくしつつシステムの保護を実現するためには仮想マシンを用いた手法が有効であることが一般に知られている [1]〜[6]．しかしながら，仮想マシンを用いた手法では命令解釈と保護機能のオーバーヘッドによりタスクの実行性能が低下してしまう．そこで，Maté [1], [2] や VM* [3] では無線センサネットワークに特化した仮想マシンを構築することで実行性能を改善する試みがなされてきた．これら先行研究ではイベント駆動型のソフトウェアシステムとして設計と実装が行われており，ベストエフォートによるタスクの実行を基本としている．

一方，無線センサネットワークの実際的なアプリ

[†] 東京大学大学院新領域創成科学研究科
Graduate School of Frontier Sciences, The University of Tokyo
[††] 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

ケーションにおいては，高精度な同期サンプリングなどのリアルタイム処理が多く必要とされる．例えば，地震モニタリング [7], [8] のような科学用途の計測や MAC プロトコルのような厳しい時間制約を有するアプリケーションではタスクの時間制約を満たすことが必要である．すなわち，タスクの開始タイミングおよび終了タイミングの保証が可能なハードリアルタイム性が必要である．しかしながら，Maté 等，既存のソフトウェアモジュール機構はベストエフォートの考え方に基づいており，このようなハードリアルタイム性をサポートしていないため，適用可能なアプリケーションが限定される．

例えば，センサ値のサンプリングを一定間隔で行いながら，無線通信・計算処理を行うことを考える．サンプリングを行うタスクが無線通信に関わるタスクによって遅れる場合には，得られるサンプルの精度が悪化してしまう．サンプルの精度が悪化してしまうとアプリケーションの範囲が狭まってしまうという問題が生じる．前述の地震モニタリングの例においては，サンプルの精度が低くなると，耐震診断など，地震工学に基づいた解析を行うことができなくなってしまう．また，無線通信に関わるタスクがサンプリングのタスクによる影響で遅れる場合には，その遅延がデッドラインを超えてしまうとパケットロスとなってしまい，再送が必要になるなど消費電力が増加してしまう．このように複数のタスクが時間制約を有する場合には，タスクの時間制約を完全に予測可能なハードリアルタイム処理を実現する仕組みが必須となる．

このような観点から，本研究では保護機能を実現しつつリアルタイム処理が可能な無線センサノード向けリアルタイム仮想マシン「VAWS」の設計と実装について述べる．VAWS では，無線センサノード上で時間制約のあるタスクが CPU の割り込みによって発生する点に着目し，割り込みベクタ毎に独立した仮想マシンをオペレーティングシステムを介すことなく直接実行することによって簡単な仕組みながらも低オーバヘッドで時間制約を満たすことを保証する．

VAWS を実装評価した結果，ベストエフォート型の TinyOS では 100 Hz のセンササンプリングを実現する場合に 62 us のジッタが発生したのに対し，VAWS で実現した場合には保護機能を実現しながらも 1 us のジッタに抑えることができた．本研究はセンサノードで一般的に用いられているマイクロプロセッサに搭載されている割り込み機能を使っているため，実行性能の改善を目的とした先行研究と共存が可能であり，Maté や VM*で用いられている実行性能改善手法やモジュールサイズ削減手法を VAWS 上にも適用することができる．

本論文では，2. において一般的な仮想マシン技術と無線センサネットワークにおけるソフトウェアモジュール機構と関する先行研究を調査し，本研究の位置づけを明確にする．次いで 3. でリアルタイム保証と保護機能をサポートする仮想マシン VAWS の設計について述べる．さらに，4. で VAWS の評価を示し，最後に 5. でまとめとする．

2. 関 連 研 究

センサノードでソフトウェアモジュールを実現するためには，ハードウェアの機能的な制約を考慮しなくてはならない．センサノードはコストと消費電力の削減のために，PIC，AVR，MSP430 のような低消費電力・低機能なマイクロコントローラを CPU として備えている．これらのマイクロコントローラはネットワークを介したモジュール単位でのソフトウェアアップデートを想定しておらず，動的にロードする際に必要となるハードウェアサポートが存在しない．具体的には，特権や MMU などのハードウェアサポートを備えないために，保護やタスクへのメモリの確保を実現するためにはソフトウェアによってこれらの機能をエミュレートする必要がある．すなわち，センサノードのためのソフトウェアモジュール機構では，このようなハードウェアサポートを具備しないという制約のなかで，いかにして保護と実行性能の確保を行うかが重要な課題となる．

ハードウェアプラットフォームが備えない機能を実現するためには，仮想マシンを用いた手法が一般に有効である．ここでは，まず既存の仮想マシンについてリアルタイム性とセンサノードへの適用性の観点から議論し，無線センサネットワークに仮想マシンを応用する際の問題点を明らかにする．さらに，無線センサネットワークのための既存のソフトウェアモジュール機構について議論し，本研究の位置づけを明確にする．

2.1 既存の仮想マシン

まず，保護をサポートする既存の仮想マシンについて，センサノードへの適用可能性とリアルタイム性の観点から議論する．なお，センサノードの CPU は前述のように特権のハードウェアサポートを備えないため，VMWare や Xen のように特権サポートが必須と

なる仮想マシンについては議論を行わない．また，携帯電話を主なアプリケーションとして，汎用オペレーティングシステムとリアルタイムオペレーティングシステムを同時に動作させることが可能な商用のリアルタイム仮想マシンモニタも存在するものの，これらの仕組みも CPU の特権サポートを必要とするため議論の対象外とする．

CPU の特権サポートを利用しない場合には，インタプリタを用いてインタプリタ内部で特権をソフトウェアで判断することによって保護が実現される．インタプリタを用いた実装においては，JavaVM を中心に，金融・軍事などのエンタープライズ用途でのリアルタイム処理や，組み込み機器への応用を目的とした仮想マシンが存在する．Java のリアルタイム仕様（RTSJ）[9] はエンタープライズ用途で用いられており，下位のリアルタイムオペレーティングシステムとのインタフェースを定めている．割り込みハンドリングなどのリアルタイム処理に必須となる機能は下位のオペレーティングシステムに委ねるとされているが，RTSJ 仕様に則ったセンサノード向けの JavaVM 実装およびリアルタイムオペレーティングシステムはこれまでのところ存在しない．

simpleRTJ [10] はセンサノードのような低資源・低性能の CPU にも適用可能な組み込み機器向け JavaVM 実装であり，割り込みを仮想マシンでハンドリングすることによって，下位のリアルタイムオペレーティングシステムを必要とせずに単独で動作可能な仮想マシンである．しかしながら，汎用オペレーティングシステムと同様に時分割型のスケジューリングとなっており，タスクに優先度を設定することができないため，ハードリアルタイム処理を実現できない．また，simpleRTJ のような組み込み機器向けの仮想マシン実装は，センサネットワークとは異なりネットワークからの動的なソフトウェアアップデートを考慮していないため，ソフトウェアアップデートにはモノリシックなイメージを交換することが必要となっている．

以上のように，これまでのところ，保護，ハードリアルタイム処理，ソフトウェアモジュールの 3 つを実現できるセンサノードに適用可能な仮想マシンは存在しない．

2.2 無線センサノードのためのソフトウェアモジュール機構

無線センサネットワーク研究においては，保護機能やモジュール性の欠如による低いソフトウェア開発生産性およびソフトウェア保守性という問題に対して，仮想マシンを用いたアプローチをはじめとしていくつかの研究が進められた．

ここでは，ソフトウェアモジュール機構の先行研究に関し，保護機能，実行性能，およびリアルタイム性の観点から議論を行う．表 1 に先行研究をまとめる．表 1 では，センサネットワーク用の主要なソフトウェアモジュール機構について，保護機能の実現手段，対応可能な保護対象および実行性能比の観点から分類を行っている．保護対象に関しては，無線センサノードの保護で一般的に重要となる機能レジスタ，メモリアクセス，CPU 占有に対する保護で分類を行った．機能レジスタは無線センサノードで特徴的な保護対象であり，割り込み制御や汎用入出力制御を行うためのレジスタである．

Maté [1], [2] は仮想マシンを利用する代表的なソフトウェアモジュール機構である．Maté はすべての保護対象をサポートしている．Maté ではアプリケーションに特化した仮想マシンを生成することによって，モジュールサイズの削減，および命令解釈に伴うオーバーヘッドの削減を実現している．例えば，アプリケーションが乱数生成を必要とする場合，乱数生成を 1 つの命令として持つ仮想マシンを生成することで，仮想マシンの命令セットを複数組み合わせた低速な乱数生成処理を行わずに済むようになる．VM* [3] は保護機能については明示していないが，Maté の実行性能改善手法に加えて，仮想マシン自体を動的に拡張可能とする仕組を用意することにより実行性能の更なる改善を試みている．

これら先行研究はイベント駆動型のソフトウェアシステムとしてデザインされている．イベント駆動型のシステムは本来タスクのプライオリティスケジューリングを具備していない．また，run-to-completion で動作するために，高プライオリティのタスクであっても，低プライオリティのタスクの影響を受けてしまう．

t-kernel [6] は仮想マシンを用いるアプローチではないが，ソフトウェアのロード時にモジュールに含まれるすべての命令を解析し修正することによって，ネイティブコードで記述されるにも関わらず部分的な保護が可能なソフトウェアモジュールを実現している．t-kernel ではソフトウェアモジュールのロード時にアクセスするメモリのアドレス変換を行い，すべての分岐命令の前にカーネルへの移行コードを挿入する．これによって，ネイティブなモジュールにおいてもメモ

表 1　既存のソフトウェアモジュール機構
Table 1　Previous works for software modules

Name	Protection Method	Protection Target			Performance Ratio
		Function Register	Memory	CPU Occupation	
Maté	Virtual Machine	OK	OK	OK	1.03〜33.5
VM*	Virtual Machine	OK	OK	OK	No Evaluation
t-kernel	Load-time Code Modification	NG	OK	OK	1.53〜3.03

リアクセスと CPU の占有に対する保護を実現している．t-kernel では，機能レジスタに対する保護は実装しておらず，保護対象を絞ることによって，1.53 倍〜3.03 倍の実行性能比を確保している．

t-kernel ではモジュールに含まれるすべての命令を精査しなければならないため非常に複雑な仕組みとなっており，実際に [6] によれば t-kernel を動作させるためには 28,864 Byte のプログラムメモリが必要となると示されている．さらに，ソフトウェアモジュールに対して移行命令を挿入するために，修正後のモジュールサイズが 6 倍から 9 倍程度に拡大してしまう問題も生じる．現在のセンサノードでは，プログラムメモリは数 10 KB と限られているため，このような複雑な方式は他のタスクの実装を圧迫させることとなる．また, [6] によればハードリアルタイム性をサポートできると述べられているが，具体的な手法については言及されていない．

このようにこれまでのソフトウェアモジュール機構では保護機能とハードリアルタイム処理を同時にサポートできていない．しかしながら，1. でも述べたように，地震モニタリング等，実際的なアプリケーションを考えた場合には，実行性能の改善に加えてハードリアルタイム性が必要となる．

3. VAWS

3.1 デザイン

前節までの議論を踏まえ，筆者らは無線センサノードのためのハードリアルタイム仮想マシン「VAWS」を設計した．ネイティブコードによるソフトウェアモジュール機構では，2. で言及した保護機構を実現しようとすると複雑な構成になってしまうことを考慮し，VAWS では仮想マシンによるアプローチを採用した．

これまでのソフトウェアモジュール機構に関する研究は，保護機能と実行性能のトレードオフをどの様に取り扱うかを課題として進められてきた．これに対して，VAWS は実行性能の改善を直接の目的とするのではなく，時間制約を満たすことがアプリケーション的な観点から重要であると考え，ハードリアルタイム性の実現に重点を置いて設計を行った．

仮想マシンを設計する際には，実マシンのどのリソースを仮想化するのかを考えなければならない．無線センサノードのリソースとしては，割り込み，メモリ，入出力の 3 つがあげられる．

VAWS ではリアルタイムタスクの応答性能を向上させるために，割り込みに関しては仮想化を行わない．割り込みの仮想化を行うことによって，システムレベルでの仮想化が可能となるものの，割り込みの仮想化を実現し，割り込みを複数仮想マシンで共有するためには，どの仮想マシンへの割り込みなのかを判断する必要が生じてしまい，遅延の不確定性が大きくなってしまう．

さらに，VAWS では 1 つの割り込みベクタを 1 つの仮想マシンに排他的に割り当てる．仮想マシン上でハードリアルタイム性を実現するためには，割り込み要求が発生した場合に仮想マシンに要求を通知し，仮想マシンがその要求に応じてタスクを切り替えるといったアプローチが考えられる．しかしながら，仮想マシン自身が要求を受け取り解釈し，実行しているタスクを切り替えるという処理はオーバーヘッドが大きいだけでなく仮想マシン自体の機構を複雑化させる原因となる．たとえば，任意の割り込みによってタスクを切り替える必要性が生じた場合，実行中のタスクのコンテキストのみならず仮想マシンのコンテキストをも保存し，これらのコンテキストを切り替えなければならない．そこで，VAWS では無線センサノードで利用されている CPU に一般的に搭載されている割り込み処理機能に着目し，割り込みベクタごとに仮想マシンを用意することで，単純な機構ながらも低オーバーヘッドで保護機能を実現しつつリアルタイム処理を実現する．

メモリに関しては，ソフトウェアモジュールの実現のために仮想化は必須である．メモリの仮想化が行われない場合には，ソフトウェアモジュール生成時にアクセスするメモリの物理アドレスを明示的に指定しな

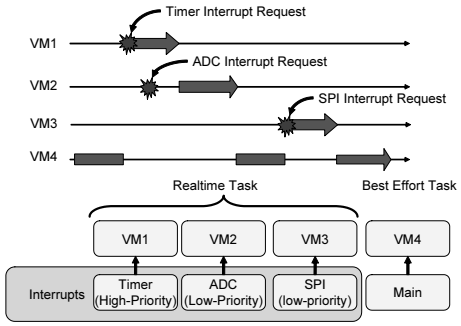


図 1　VAWS のスケジューリング
Fig. 1　VAWS Scheduling

type (1 Byte)	text length (1 Byte)	data length (1 Byte)	bss length (1 Byte)	text segment (~256 Byte)	data segment (~256 Byte)

図 2　モジュール構成
Fig. 2　Module Constitution

ければならない．また，仮想化を行わない場合には不正なアドレスへのメモリアクセスを検出することが不可能となる．メモリの仮想化はそれぞれの仮想マシンがメモリのアドレス変換およびアドレスの正当性のチェックすることによって仮想化を行う．入出力に関してはマップされた機能レジスタを隠蔽し機能レジスタ制御を行う仮想命令セットを提供し，仮想命令セットの実行前に検証を行うことによって仮想化を行う．

3.2　ハードリアルタイムサポート

図 1 に，Timer，ADC，SPI に起因するリアルタイムタスクと 1 つのベストエフォートタスクを VAWS で実現した時のタスクスケジューリングの様子を示す．図 1 に示されるように，VAWS では 1 つの割り込み要因につき 1 つの仮想マシンが存在し，それぞれの仮想マシン上で 1 つのリアルタイムタスクが実行される（VM1〜VM3）．これは，割り込み要求発生時に直ちに仮想マシンのインタプリタのタスクが開始されるように，仮想マシンのインタプリタを割り込みベクタに対応させることで実現している．また，仮想マシン上で実行されるタスクの優先度と割り込み優先度の設定を仮想マシン間で横断的に対応させることで，優先度に基づいたスケジューリングを実現している．なお，モジュール化と保護のために，リセット時のマイクロコントローラのエントリポイント（Main）から仮想マシンのインタプリタを実行することで，ベストエフォートタスクを実現することも可能である（VM4）．

無線センサノードにおける時間制約を有するタスクは，割り込みに起因して開始要求が発生する．たとえば，TDMA 型の MAC プロトコルにおける，タイマ割り込みに起因した無線通信モジュールの状態切り替え要求，フラッシュメモリとのシリアル通信における，I^2C（Inter-Integrated Circuit）や SPI（Serial Peripheral Interface）のバス割り込みに起因するバスアクセス要求，センサ値の計算処理における，AD 変換終了割り込みに起因した計算要求などがあげられる．

PIC18，AVR，MSP430 といったようなマイクロコントローラでは，優先度に応じた多重割り込みが可能である．すなわち，タスクの開始要求の要因となる割り込みハンドラにタスクを割り当てたうえで，タスクの優先度と割り込みの優先度とを対応付けることができれば，deadline-monotonic scheduling [11], [12] でのリアルタイム処理の保証が可能となる．deadline-monotonic scheduling は 1 つのプロセッサの場合に最適な優先度割り当てが可能であることが知られている．

以上のような仕組みによって，優先度に応じたタスクスケジューリングが実現される．例えば，ベストエフォートタスクを実行している時に割り込み要求が発生した場合，即座に対応する割り込みハンドラに遷移し対応するタスクが実行される．また，高位の優先度を有するタスクを実行中に低位の割り込み要求が発生した場合は高位の優先度を有するタスクがそのまま実行される．

なお，VAWS では 1 つの割り込みを 1 つの仮想マシンに排他的に割り当てるため，1 つのソフトウェアモジュールで複数の割り込みに対応することができない．複数のデバイスの割り込みを利用する際には，タスクを分割することで対応する．

タスクと割り込みの対応付けを行うために，図 2 に示されるように，タスクの優先度および割り込み要因を示す `type` フィールドをソフトウェアモジュールに含ませる．`type` フィールドは表 2 のようなビット構成になっており，ロードされるアプリケーションがハードリアルタイムタスクとして実行されるかベストエフォートタスクとして実行されるかを示す．また，`priority` にタスクの優先度を示し，`int_source` にどの割り込みに対応するかを示す．

3.3　保　　護

VAWS は，Maté [1], [2] や VM* [3] と同様の方法で，機能レジスタおよびデータメモリへの不正なアクセス

表 2　type フィールド
Table 2　Type Field

Bit Field	Meaning
$type_8$	real-time task or normal task
$type_7$	priority (if real-time task)
$type_1 - type_6$	int_source (if real-time task)

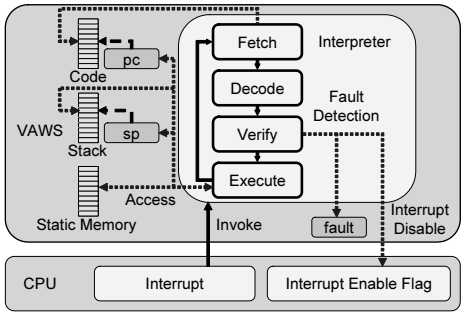


図 3　VAWS アーキテクチャ
Fig. 3　VAWS Architecture

表 3　VAWS のコンテキスト
Table 3　VAWS Context

Name	Size	Meaning
`pc`	1 Byte	Program Counter
`sp`	1 Byte	Stack Pointer
`fault`	1 Byte	Fault Detection

と CPU 占有に対する保護機能を提供する．これに向けては，これらの不具合の検出機能と，検出によってタスクの実行を終了させるタスク管理を行わなければならない．

VAWS は図 3 に示すように，インタプリタ（Interpreter）を実行エンジンとして持つスタックアーキテクチャによる仮想マシンである．インタプリタはコンテキストとして表 3 に示すようにプログラムカウンタ (`pc`)，スタックポインタ (`sp`)，フォルト (`fault`) を保持している．図 4 に擬似コードとして示すように，`pc` が指すコード中に含まれる仮想命令を（1）フェッチ，(2）デコードし，それぞれの命令に応じて検証と実行を行う．例えば，(3）`VAWS_OP1` とデコードされた場合には，(4）`<vaws_op1 verify code>`においてアプリケーションから渡されたパラメータを実行前に検証したうえで，(5）`<vaws_op1 implementation>`で対応する機能を実現する．

CPU の機能レジスタへのアクセスに関しては，機能レジスタへのアクセスをメモリアクセスとして実現するのではなく，機能レジスタへアクセスするための専用の仮想命令を用意し，この命令を通してのみアクセス可能とする．また，メモリアクセスに関しては，アクセス先が仮想マシンに割り当てられたメモリの範囲内であるか検証することによって保護を実現する．これによって，機能レジスタやオペレーティングシステムが保持するデータをモジュールの不具合による不正なアクセスから保護できる．具体的には，表 4 に示すように，スタック操作，条件分岐，四則演算によって構成される計算処理および制御構造を実現するのに必要十分な基本となる命令セットに加えて，`SET_FR_PORTA` のように機能レジスタへアクセスするための仮想命令を追加した．なお，仮想マシンによる解釈のオーバーヘッドの削減のために，Maté や VM* と同様に，`SEND`，`RECV`，`SENSE` といったように，アプリケーションに特化した仮想命令も拡張可能とした．スタックメモリの操作に関しては，`POP` と `STORE` の命令において `sp` が `0` 以上であること，`PUSH` と `LOAD` の命令において `sp` が `STACK_LENGTH` より小さいことを検証する．また，静的領域のメモリ操作に関しては，`LOAD` と `STORE` の検証コードにおいて，アクセス先が `0` から `DATA_LENGTH` 未満であることを検証する．

CPU の占有に対する保護は仮想的な Watch Dog Timer によって実現する．具体的には，実行開始時に `wdt` を `0` とし，1 つの仮想命令を解釈・実行するたび

```
while(1){
    inst=code[pc]; … (1)
    switch(inst){ … (2)
    case VAWS_OP1: … (3)
        <vaws_op1 verify code> … (4)
        <vaws_op1 implementation> … (5)
        break;
    case …
        …
    default:
        fault=VAWS_FAULT_ILLEGAL_OPCODE;
    }
    pc++;
    wdt++;
    if(wdt==VAWS_WDT_MAX){
        fault=VAWS_FAULT_WDT;
    }
    if(fault!=VAWS_FAULT_NONE){
        …
        break;
    }
    if(pc>=vaws_code_size) break;
}
```

図 4　インタプリタの構造
Fig. 4　Structure of the Interpreter

表 4　VAWS 命令セット
Table 4　VAWS Instruction Set

Category	Example
Stack Operation	PUSH,POP,LOAD,STORE
Branch	BRLS, BRGT, BRNE, BRGE
Calculation	ADD, SUB, MUL, DIV
Function Register	SET_FR_PORTA, GET_FR_PORTA
Extention	SEND, RECV, SENSE

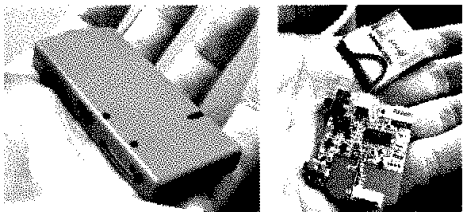


図 5　PAVENET モジュール
Fig. 5　PAVENET Module

に wdt をカウントアップする，VAWS_WDT_MAX で定められた回数となった場合には，無限ループのフォルト状態であると判断し，実行エンジンのメインループから脱出する．

以上のような方法によって不具合を検出した場合，fault に違反の要因を書き込み，割り込み許可フラグを disable とし，対応する仮想マシンのみを終了させる．これによって，仮に不具合を含むソフトウェアモジュールを実行したとしても，CPU や他のタスクに影響を与えることなく安全に終了されるとともに，割り込みが不許可となるために，そのソフトウェアモジュールに含まれるタスクが実行されないようになる．

4. 実装評価

筆者らは，図 5 に示される PAVENET モジュール [13] を利用して VAWS の実装を行った．PAVENET モジュールは CPU として 20 MHz で駆動される Microchip 社の PIC18LF4620 [14]，無線通信モジュールとして 38,400 bps で無線通信を行う ChipCon 社の CC1000 を具備している．Microchip 社の PIC18 は高位優先度と低位優先度の 2 つの割り込みベクタを有しており，すべての割り込み要因に対して割り込み優先度をソフトウェアで選択することが可能である．

筆者らの実装では，1 つの仮想マシンあたり 2180 Byte のプログラムメモリを消費した．データメモリの消費量は code 領域，stack 領域，data 領域にどれだけ割り当てるかに依存するが，それぞれ 256 Byte，8 Byte，8 Byte 割り当てた時のデータメモリの消費量は 280 Byte であった．

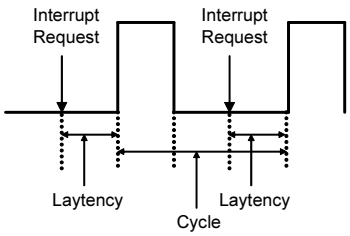


図 6　ハードリアルタイム評価アプリケーション
Fig. 6　Evaluation Application of Hard Real-time Guarantee

また，JavaCC（Java Compiler Compiler）[15] を利用して制御構造，ポインタ，配列などを解釈可能な C ライクな構文を有する VAWS 用コンパイラの実装を行った．JavaCC は構文情報を与えることで，パーサを生成するコンパイラコンパイラである． なお，VAWS 用コンパイラではエントリポイントの記述によってハードリアルタイムタスクとベストエフォートタスクを区別した．ハードリアルタイムタスクの場合は，

```
rtask priority int_source(void){}
```

と記述し，ベストエフォートタスクの場合は，

```
task main(void){}
```

と記述する．

4.1　ハードリアルタイム保証

ハードリアルタイム保証が実現されていることを確認するために，汎用出力ピンに図 6 に示されるような矩形波を出力するアプリケーションを実行し，出力される矩形波の周期（Cycle）の計測を行った．リアルタイム保証が行えないシステムにおいては，タスクの開始要求が発生してから実行が開始されるまでの遅延時間（Latency）が不定となるために，サンプリングジッタが発生してしまう．サンプリングジッタが存在すると取得したサンプル値に誤差が発生してしまうために測定精度が低下してしまう．

地震モニタリング [8] をアプリケーションとして想定した 100 Hz の矩形波を出力するプログラムを実行し，並列して無線のリスニングを行うタスクを動作させる場合とさせない場合とを比較した．VAWS との比較のために，MICA2 上で動作する TinyOS [16] も同様のアプリケーションを実行して評価した．MICA2 と PAVENET モジュールの CPU の比較を表 6 に示

表 5　リアルタイム保証性能
Table 5　Evaluation of Real-time Guarantee

Task	VAWS			TinyOS (default)			TinyOS (optimized)		
	Max Cycle (ms)	Min Cycle (ms)	Jitter (us)	Max Cycle (ms)	Min Cycle (ms)	Jitter (us)	Max Cycle (ms)	Min Cycle (ms)	Jitter (us)
Sampling	10.001	10.000	1	9.764	9.763	1	10.003	10.000	3
Sampling + Listening	10.001	10.000	1	11.735	7.794	3,941	10.062	10.000	62

表 6　評価プラットフォームの CPU 仕様
Table 6　CPU Specification of Evaluation Platforms

Platform/CPU	PAVENET PIC18LF4620	MICA2 ATmega128L
Performance	5 MIPS	7.3728 MIPS
RAM Size	4 KB	4 KB
ROM Size	64 KB	128 KB
Current Consumption	9 mA	8 mA

表 7　仮想命令の実行サイクル数
Table 7　Cycle Per Instruction of Virtual Instruction

Instruction	VAWS	Maté
`and`	30.27:1	33.5:1
`rand`	1.721:1	9.5:1
`sense`	1.007:1	3.4:1
`send`	1.002:1	1.03:1

す．Maté は TinyOS 上で動作していることから，サンプリングジッタに関しては TinyOS と同程度となる．この際，Timer コンポーネント [17] を利用した場合と，MicroTimer コンポーネント [18] を利用した場合とを評価した．Timer コンポーネントを利用した場合には，割り込み要求発生時にタスク実行要求がタスクキューにポストされ，TinyOS のタスクスケジューラによってタスクがスケジュールされるまで実行が遅延される．このため，他のタスクの影響を大きく受ける．MicroTimer コンポーネントを利用した場合には，割り込みハンドラにおいて直接タスクが実行されるため，他のタスクの影響は小さくなるものの，他の割り込みハンドラの影響を受けてしまう．

出力された矩形波の周期を，矩形波の立ち上がりから次の立ち上がりまでの間隔を計測のみを行う PAVENET モジュールによって計測し，2000 回の周期を保存した．PAVENET モジュールに具備される PIC18LF4620 は CPU 上での 1 サイクルはシステムクロックの 4 サイクルとなり，20 MHz で駆動されている．このため，PIC18LF4620 でのタイマは 1 サイクルが 0.2 us となり，測定された 1 周期のカウント数に 0.2 us を乗ずることによって周期を求めた．出力された矩形波の最大周期，最小周期およびそれらの差であるサンプリングジッタを表 5 に示す．水晶の固有差が 3×10^{-5} であることから，周期が 10 ms であることを考慮し，最小の単位を 1 us として示した．Timer コンポーネントを利用した TinyOS ではネイティブコードを実行しているにも関わらず 3,941 us のサンプリングジッタが発生している．MicroTimer を利用した TinyOS では 3 us と低ジッタでを実現できているものの，無線通信など他のタスクを実行できなくなってしまうという問題がある．これに対して，VAWS ではサンプリングジッタが 1 us 以下であり，CPU の 1 サイクルに抑えられていることが確認できた．このジッタは 2 台の PAVENET モジュールの水晶発信子の固有差に起因するものである．すなわち，VAWS ではタスク開始要求発生後に遅延時間が決定的であり，リアルタイム処理が可能であると言える．このように，TinyOS とは異なり VAWS はタスクの優先度を割り込みの優先度に対応付ける仕組みを持っているために，TinyOS において MicroTimer を利用した場合と比較しても，低ジッタでのサンプリングが実現できていることが分かる．

4.2 実行性能

VAWS は割り込みに起因するタスク制御のみを規定する仕組みであり，このタスク制御は仮想マシンの命令セットには依存しない．このため，Maté や VM* における実行性能改善手法と同様の方法で命令解釈のオーバーヘッドを削減することができる．本節においては，Maté などの実行性能改善手法との共存可能であることを示すため，アプリケーションに特化した命令セットを用いて実行性能の評価を行う．

まず，基礎的な評価として，`and`，`rand`，`sense`，`send` の 4 つの命令に対して，ネイティブコードで同様のプログラムを実行した時との実行性能の比較を行った．評価結果を表 7 に示す．なお，参考のために [1] に示された Maté におけるネイティブ命令との実行性能比を併記する．CPU アーキテクチャ，コンパイラ，センサ，無線通信モジュールが異なるものの，VAWS と Maté で仮想命令の実行速度比の関係は同

表 8 実行性能
Table 8 Execution Performance

Application	VAWS (ms)	PAVENET OS (ms)	Peformance Ratio	Module Size
CTL	0.273	0.006	42.7	20 Byte
CTR	4.355	3.732	1.17	28 Byte
STR	5.900	5.700	1.03	16 Byte
RTL	3.104	2.513	1.24	17 Byte

様の傾向にあることが分かる．このように，インタプリタを用いた仮想マシンでは，実行ロジックにおいて `rand()` のように計算量の大きい仮想命令や，`sense()` や `send()` のように CPU 外部の低速デバイスを利用する仮想命令ではオーバーヘッドが相対的に削減される．すなわち，VAWS においてもアプリケーションに特化した粒度の大きい仮想命令を導入することによって，命令解釈や保護によるオーバーヘッドを大幅に削減可能なことが分かる．

次に，アプリケーション実行時の実行速度の評価を行った．評価対象とするアプリケーションとして，カウントアップした値を LED に表示するアプリケーション（CTL），カウントアップした値を無線によって送信するアプリケーション（CTR），加速度を取得し無線送信するアプリケーション（STR），無線によって受信した値を LED に表示するアプリケーション（RTL）の 4 つのアプリケーションを用いた．

測定した実行サイクル数を表 8 に示す．CTL は計算処理が主となるアプリケーションであるため実行性能が 42.7 倍と著しく実行性能が低下している．これに対して，CTR，STR や RTL のように無線通信やセンサなど入出力待ちが大きいプログラムにおいては，それぞれ 17 %，3 %，24 %のパフォーマンス低下となった．アプリケーションの実行に関しても，Maté と同様に粒度の大きい命令を利用することでオーバーヘッドの大幅な削減が可能であることが分かる．また，これら 4 つのモジュールは表 8 に示した通り数 10 Byte で実現されており，通信リソースの小さいセンサノードでも低コストで配信することが可能であると言える．つまり，VAWS は，Maté や VM*と同様にアプリケーションに特化した仮想命令セットを構築することによって実行性能の向上とモジュールサイズの削減が可能であることが分かる．

5. ま と め

本論文では，無線センサノードのためのハードリアルタイム保証が可能な仮想マシンである「VAWS」の設計と実装について述べた．VAWS によれば保護を実現しつつリアルタイム性が必要なタスクを実現することができる．また，本研究は既存の研究の仕組みと共存可能であり，Maté や VM*での実行性能改善およびモジュールサイズ削減に関する知見を VAWS にも応用することができる．

本論文では，保護およびハードリアルタイム保証が可能なソフトウェアモジュール機構の実現に主眼を置いたため，実行性能の最適化については触れなかった．命令セットの構成方法と性能の関係を中心に，実行性能の改善についても研究を進める予定である．また，現在の VAWS 用のコンパイラは C ライクな簡略な実装となっており，浮動小数点や構造体をサポートしないなどの制限がある．センサネットワークのアプリケーションを構築するためのプログラミング言語として必要十分な機能についても検討を進めたい．

現存のセンサノードで保護機能を実現することを目的としたため仮想マシンというアプローチを選択したが，これらの機能はハードウェアで実現した方が低消費電力に実現可能である．VAWS の保護機能は簡略な条件分岐で実現されており，このようなハードウェア支援を具備した低消費電力な CPU を実現することも可能であると考えている．今後は，保護機能のハードウェア化まで見据え，無線センサノードのための保護機能と消費電力の関係についても評価検討を行っていく予定である．

文 献

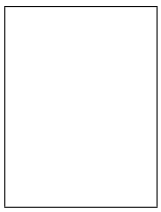
[1] P. Levis, and D. Culler, “Maté: A tiny virtual machine for sensor networks,” Proceeding of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, pp.85–95, San Jose, CA, USA, Oct. 2002.

[2] P. Levis, D. Gay, and D. Culler, “Active sensor networks,” Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, pp.343–356, Boston, MA, USA, Nov. 2005.

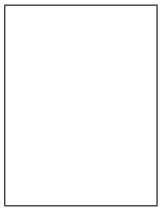
[3] J. Koshy, and R. Pandey, “VM*: Synthesizing scal-

able runtime environments for sensor networks," Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, pp.243–254, San Diego, CA, USA, Nov. 2005.
[4] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, pp.455–462, Tampa, FL, USA, Nov. 2004.
[5] C.C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes," Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, pp.163–176, Seattle, WA, USA, Jun. 2005.
[6] L. Gu, and J.A. Stankovic, "t-kernel: Providing reliable OS support to wireless sensor networks," Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, pp.1–14, Boulder, CO, USA, Nov. 2006.
[7] M. Suzuki, S. Saruwatari, N. Kurata, and H. Morikawa, "Demo abstract: A high-density earthquake monitoring system using wireless sensor networks," Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, pp.373–374, Sydney, Australia, Nov. 2007.
[8] 鈴木誠，倉田成人，猿渡俊介，森川博之，"無線センサネットワークによる地震モニタリングシステムの実装と評価," 信学技報，no.USN2007-66，pp.65–70，Jan. 2008.
[9] rtsj: Real Time Specification for Java, https://rtsj.dev.java.net/.
[10] simpleRTJ: a small footprint Java VM for embedded and consumer devices, http://www.rtjcom.com/.
[11] J.Y. Leung, and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," Performance Evaluation, vol.2, no.4, pp.237–250, 1982.
[12] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software, pp.133–137, Atlanta, Georgia, May 1991.
[13] 猿渡俊介，森川博之，青山友紀，"シングル CPU で実現される無線センサノードの実装," 信学総大，no.B-7-101，p.197，Mar. 2006.
[14] Microchip Technology Inc., PIC18F2525/2620/4525/4620 Data Sheet, 2007, http://ww1.microchip.com/downloads/en/DeviceDoc/39626d.pdf.
[15] javacc: JavaCC Home, http://javacc.dev.java.net/.
[16] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, pp.93–104, Cambridge, MA, USA, Nov. 2000.
[17] http://tinyos.cvs.sourceforge.net/tinyos/tinyos-1.x/tos/system/TimerC.nc.
[18] http://tinyos.cvs.sourceforge.net/tinyos/tinyos-1.x/apps/HighFrequencySampling/MicroTimerM.nc.

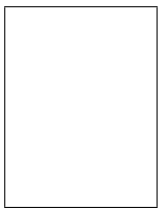
（平成 xx 年 xx 月 xx 日受付）



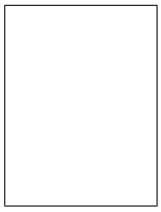
鈴木　誠　（学生員）



猿渡　俊介　（正員）



南　正輝　（正員）



森川　博之　（正員）

Abstract Wireless sensor networks need software modularity, which enables sensor nodes to modify a part of software via on-line, because of its maintenance cost caused by a large number of nodes. The software modules have to be safely performed in real-time for time sensitive applications such as high quality sampling. To realize the software modularity, this paper shows hard real-time virtual machine, called as VAWS. VAWS realizes hard real-time support and system protection with simple structure, which places a virtual machine on each interrupt vector. The paper demonstrates that VAWS performs 100 Hz sensor sampling with 1 us jitter while performing wireless communication.

Key words Wireless Sensor Networks, Software Module, Real-Time Processing, Protection, Virtual Machine