RC-013

システム実時間測定のための Lossless データ圧縮伸張方法
Lossless compression and decompression method for real-time system monitoring

高野 光司† 大庭 信之† 中田 武男†
Kohji Takano Nobuyuki Ohba Takeo Nakada

1. はじめに

組み込みシステムにおいては，様々な入出力を高いリアルタイム性をもって処理する必要がある．このような機器で発生する問題を解決するためには，プロトタイピングや実環境でのシステムテスト・検証が有効である[1][2]．組み込みシステムは，さまざまな割り込み要因によって，複数のタスクを切り替えながら処理を進めていくため，そのリアルタイム検証には，実時間での測定とシステムの振る舞いを示すトレースデータの記録が必要である．トレースデータは大容量となるため，通常外部の大容量記憶装置へ送る必要がある．しかし，情報を送る伝送路の帯域幅の許容量で記録できるデータ量が制限されてしまうことが多い．

伝送路の帯域を有効利用してより多くの情報を取得するためには，情報を圧縮して送ればよい．しかし，リアルタイム性を確保し，かつ実装が容易であるような，小型高速な圧縮ハードウェアが必要となる．ここで圧縮伸張は，取得した情報の使用目的が検証であることから，圧縮伸張によって情報が欠落しない lossless な方式が望ましい．

本論文では，組み込み機器やマルチプロセッサシステムのリアルタイム検証で必要となる，実時間の測定データをリアルタイムで lossless データ圧縮伸張する方式について提案し，実際にハードウエア実装と実験を行い，その結果を報告する．

2. システムの実時間計測で用いる情報

2.1 既存技術

従来よりシステムの挙動を取得する様々な方法が提案されている．そのひとつとして，検証対象のソフトウエアに検証用コードを挿入して関数遷移を取得する方法[3][4]が広く使われている．これは，関数の call/return に何らかの検証用のコードを挿入(instrument)して，その検証用コード内部で情報の蓄積を行う方法である．この方法は検証用コード挿入だけによってシステムの挙動に関する情報が取得可能なため利便性が高いが，instrument のオーバーヘッドによる挙動の変化がある．一方，検証対象ソフトウエアの変更をせずに MPU から出力されるトレース情報[5][6]を用いる方法[7]が提案されている．トレース情報は，MPU の備える専用のハードウエアから MPU の挙動に関する情報として直接外部へ出力される．このトレース情報を専用の測定ハードウエアを用いて観測し，トレース情報から MPU の動作を再構成(reconstruction) することで，MPU の挙動が詳細に解析可能となる．

また，システムから出力される情報は，関数の遷移だけではなく，OS 等の情報，たとえばシステムで実行しているタスクの情報や，タスクスイッチ，割り込みといった情報も含まれる．また，組み込みソフトウエアのようなリアルタイム性の要求される対象は，これらの情報がいつどういったタイミングで発生したかといった時間に関する情報が高い精度で得られることも検証に必要である．

本論文では，システムの挙動を実時間で測定する方法として，システムの動作に関する情報を出力するコードを挿入するか，もしくはプロセッサの出力するトレース情報を外部の測定装置に出力して観測する方法をターゲットとする．それと同時に，その情報が発生した時間に関する情報も高い精度で付加して測定する．

2.2 実時間計測のデータ構造

外部の測定装置で観測される情報（以降，動作情報と表記）は，システムの動作情報と時間の情報で構成される．本論文が対象とする情報のデータ構造を図 1 のように定義する．時間情報(timestamp)のビット数を N_t ビット，動作情報(behavior info)を N_b とする．ここで時間情報は，時系列に出力される動作情報の絶対時間ではなく，直前の動作情報との差分の時間を表現する．差分の時間を用いる理由は，記録の際のビット数 N_t を少なくするためである．

timestamp (N_t-bit)	behavior info (N_b-bit)

図1 実時間計測のデータ構造

動作情報としては，どの関数が実行開始・終了したかの情報と，OS 等の情報（タスクの情報，タスクスイッチ，割り込み等）を扱う．関数の開始・終了を表現するために，動作情報の MSB を 0 と定義し，OS 等の情報は MSB を 1 として区別する．

まず関数の開始・終了を示すデータ構造を図 2に示す．Behavior info の MSB を 0 としてこのデータが関数の開始・終了を示すデータであることを示し，後述の OS 等の情報を示すデータと区別する．次の 1 ビットには関数の開始なのかそれとも終了なのかを区別するためのビットを設ける．それに続き，関数を区別するための関数 id(function id) を設ける．

次に，OS 等の情報を示すデータ構造を図 3に示す．OS 等の情報であることを示すために MSB を 1 に固定して区別する．それに続くビット列の info type には，このデータが何の情報であるかを示す情報種別を記録し，OS のタスクの情報なのか，割り込みなのか，それともその他の何らかの情報なのかを識別するための固有の値を格納する．そして，それに続くビット列 info value に，それらの情報が持つ実際の情報値を格納する．

†日本アイ・ビー・エム株式会社 東京基礎研究所
IBM Research, Tokyo Research Laboratory, IBM Japan Ltd.

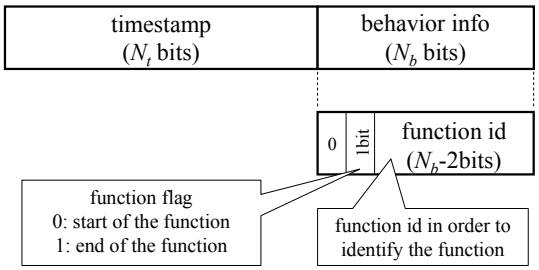

図2　関数の開始・終了を示すデータ構造

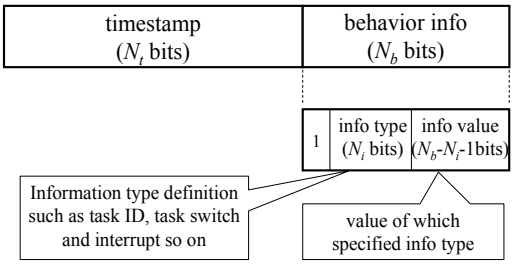

図3 OS 等の情報を示すデータ構造

2.3 実時間計測データの特徴

実時間測定において，測定される動作情報の頻度が高ければ高いほど，より多くの情報を外部の測定装置に記憶する必要がある．頻度が高いということは，言い換えるならば，動作情報が観測される時間間隔が狭いということであり，図 1～図 3に示した時間情報のフィールドの上位ビットの大部分は 0 になる．また，図 1 の関数情報には，関数の開始，終了といった情報が格納される．ある関数の開始に対し，その終了がペアとなって観測される特徴がある．

OS 等の情報を示すデータ構造に関しては，たとえば OS のタスクの切り替えを表現する場合を考えると，情報種別(info type)は OS のタスクの切り替えであることが判別可能な値となり，その時の情報値(info value)にはタスクを特定するためのプロセスＩＤ等が格納される．ここで，実際のシステムでは無秩序にプロセスＩＤが変化・発生するわけではなく，ある処理の間には特定のプロセスＩＤの頻度が高いといった状況となり，局所的にはある限られた識別情報が高い頻度で観測される傾向がある．

これらの特徴をまとめると，システムの動作の実時間計測で観測されるデータの特徴として，下記の点が挙げられる

- 時間情報は，頻度が高くなるにつれ，MSB 側に連続した 0 が増える
- 関数情報は，関数の呼び出しがスタック構造をとるため，開始と終了のペアとなることが大部分である
- 識別情報は，動作に応じて，ある程度局所的に同一の値が観測される傾向がある
- 測定時間に比例した情報量となる

3. 実時間計測情報の圧縮・伸張

実時間計測データの特徴をふまえ，本論文では，実時間計測で必要となるデータ構造の特徴に応じた圧縮・伸張の方法を提案し，これを実現するハードウエア実装に関して述べる．圧縮・伸張に関してはデータ構造の特徴のそれぞれに対し異なった３つの方法を適用する．以下詳細に説明する

3.1 時間情報の圧縮・伸張

時間情報は 0 以上の値をとるため，時間情報データフィールドのどこかに，必ず「1 のビット」が存在する．一方で，頻繁に実時間情報が観測される状況においては，時間情報の持つビットの値は，そのほとんどが 0 となり，MSB から連続した 0 が観測される．よって頻繁に実時間が観測される状況では，この MSB から連続する「値が 0 のビット」の数に注目して圧縮を行うことが効果的と考える．

時間情報のビット数が N_t の場合，連続する「0 のビット」の数を二進数で表記する場合，それに必要となるビット数 N_{zero} はガウス記号を用いて式(1)の様にあらわされる．

$$N_{zero} = \lfloor \log_2(N_t - 1) \rfloor + 1 \quad (1)$$

よって，時間情報の MSB 側の 0 のビットを，固定のビット長 N_{zero} での二進数表現に置換を行うことでビット数の圧縮を行う．そして，時間情報中の MSB 側から見た最初の「1 のビット」の位置は，前述の連続する 0 のビット数の次に存在するため省略し，以降のビットから LSB までは保持する．

この具体的な例を図 4に示す．図 4は観測された時間情報 N_t が 20bit の場合であり，その時の時間情報(timestamp)が二進数表現で 00000000000001011010b(以降，二進数の値の末尾に b を付記する) の場合の例である．圧縮は，まず時間情報の MSB から連続する「0 のビット」の数を検出する．図の例では「0 のビット」が 13 個連なっている．$N_{zero} = \lfloor \log_2(20-1) \rfloor + 1 = 5$ であるため，13 を 5 ビットの二進数で表現すると 01101b となり，これが圧縮後の値となる．　時間情報の最初の「1 のビット」以降の時間情報はそのまま複製する．結果として，図の例の 20bit の時間情報は，圧縮の結果 01101011010b と表現され，11 ビットとなる．

伸張は圧縮の逆の手順で行われる．圧縮されたデータの最初の N_{zero} ビットは，圧縮前データの MSB 側からの連続する「0 のビット」の数を表現した固定長のフィールドであるため，この N_{zero} ビットの値から，圧縮前に連続していた 0 の数が判断できる．この連続する「0 のビット」の次は必ず「1 のビット」であるためこの「1 のビット」も復元され，そして，圧縮後の残りのビットをそのまま複製すれば，伸張が完了する．

時間情報の伸張の例を図 5に示す．例では $N_{zero} = 5$ であるため最初の 5 ビットである 01101b が圧縮前の時間情報の MSB 側「0 のビット」であることがわかる．よって，連続した 13 個の「0 のビット」復元し，次のビットは必ず 1 であるため，伸張後の「0 のビット」の後ろに「1 のビット」を追加する．そして，圧縮後ののこりのデータである 011010b をさらに追加することで，伸張が完了する．

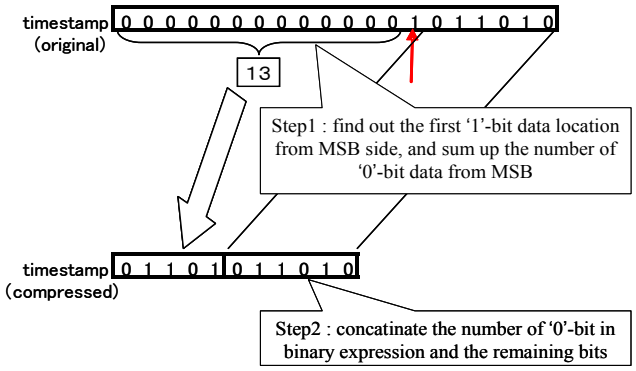

図4 時間情報の圧縮例

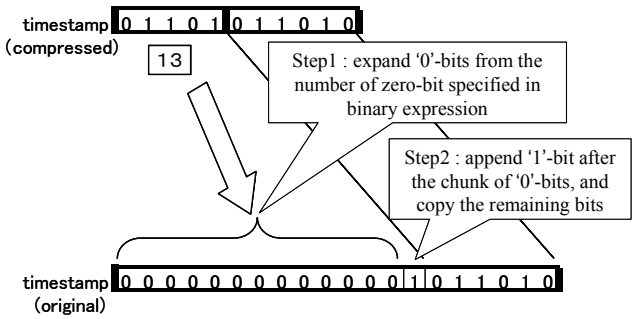

図5 時間情報の伸張例

このような連続するビットをその数に変換する圧縮伸張法は，広く知られているランレングス法に類似する．ランレングス法は，0 または 1 が連続する対象に対しては良い圧縮率を実現するが，頻繁に 0, 1 が変化する対象へ圧縮を試みる場合，ランレングスの表現のオーバーヘッドが大きくなってしまい圧縮率が悪化することが知られている．実時間測定では，測定する情報が多くなればなるほど時間情報の MSB 側の 0 の数が連続することになるため，時間情報の MSB 側だけにランレングス法を適用する本手法が効果的に機能する．

3.2 関数情報の圧縮・伸張

関数情報の圧縮伸張は，測定中に動的に変更される辞書を用いて行う．ここで，関数の開始・終了が対に発生することに注目して，辞書中に含まれる関数の開始・終了を表すビットを動的に変更することが特徴となる．関数情報の圧縮伸張に使われる辞書の構造を図 6に示す．辞書の要素数は事前に決定し，辞書のアドレスを N_d ビットとした場合その要素数は 2^{N_d-1} となる．辞書が小さいほど圧縮伸張の実装コストが減るため，設計時に適切な辞書のサイズを決定する．

辞書の値には，関数の開始か終了かを表す 1 ビットと共に，関数を特定する id を格納する N_b-1 ビットの領域が存在する．この辞書に，観測中に観測された関数 id が格納され，過去に観測された関数の場合は，その辞書のアドレスが圧縮後の符号として使われる．

最後のアドレスである 1...111b には値を格納しない．これは，観測した関数情報がどの辞書にも属さない場合を識別するための特別な符号として用いるために予約される．関数情報が観測されるたびに辞書に記録済みかどうかがチェックされ，辞書に該当する関数情報があれば，そのアドレスを圧縮・伸張の符号として用いる．観測された関数情報が辞書にない場合は，辞書にその最新の関数情報を格納し，最も古い辞書の情報を破棄する．以下で圧縮・伸張のながれを具体的な例とともに示す．

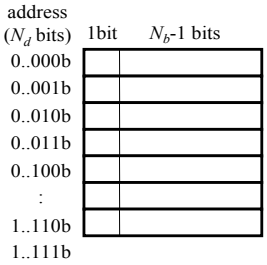

図6 関数情報の圧縮伸張に使われる辞書の構造

関数情報が観測されるたびに，辞書にその関数情報と合致するものが格納済みかどうかをチェックする．図 7の例は，観測した関数情報に該当するものが辞書にあった場合の例である．図 7では，観測データの関数 id が 00010001b の開始である例であり，すでに同じものが辞書のアドレス 0..010b に格納されている例である．この場合，圧縮後のデータはそのアドレスである 0..010b が圧縮後のデータとなる．また，関数の開始，終了がペアとなって観測される特徴から，次回に観測される可能性があるのは，開始・終了フラグが反転した値である．この例では，観測された関数が開始であるため，該当する辞書の開始・終了フラグを反転して，辞書には終了がくるものとして準備しておく．

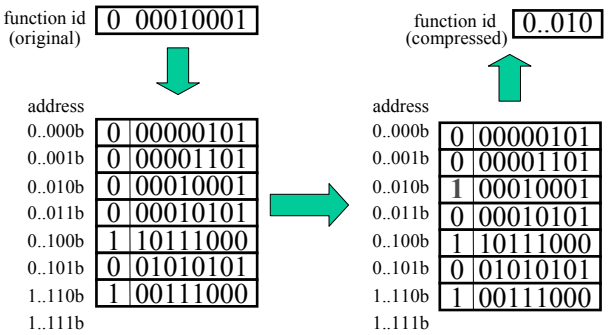

図7 関数情報の圧縮例（辞書に存在する場合）

観測した関数情報に該当するものが辞書にない場合は，辞書に観測した関数情報を追加する操作を行う．その例を図 8に示す．観測されたデータは，辞書の最も古いものを捨て，その領域に追加される．実装方法としては，図 8のように辞書の先頭（アドレス 0..000b）を最新のデータとなるように管理する方法でもよいし，辞書にライトポインタを設けて最新の情報の格納先を管理する方法でも良い．また，その際には，関数の開始・終了を示すフラグは反転させ，次に観測された関数の開始・終了の関係が保たれるようにする．

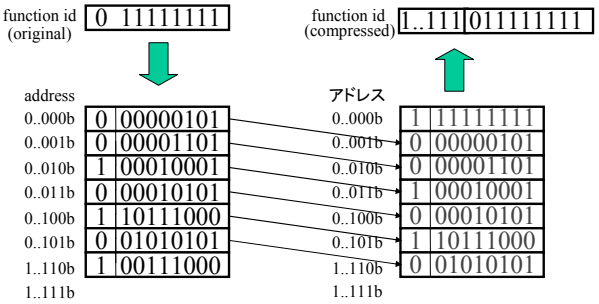

図8 関数情報の圧縮例（辞書に存在しない場合）

辞書に新たな関数情報が追加された際の圧縮後のデータは，まず，辞書に値が追加されたことを示す予約の符号 1..111b が先頭に配置され，それに続いて観測されたデータをそのまま出力する．これによって，辞書に無い関数 id が観測されて辞書に新規に登録されたことが伸張の際に検出でき，伸張側の辞書も同様に操作することで圧縮伸張が実現される．

上記のように，本圧縮伸張の方法では，観測される情報がスタックの特徴（Call の後には高い確率で return がある）となることに注目した結果，関数情報の辞書の内部で自動的に次に高い可能性で発生するであろうデータに辞書内部で自動的に切り替えを行っている．頻繁に呼び出される関数は，辞書のエントリとしてずっと存在しヒットし続けるため，効率的に圧縮されることとなる．

一方，再起呼び出しや，OS によるコンテクストスイッチ，割り込み，longjmp 等のスタックの構造が不規則に変化する場合は，辞書にヒットしないため圧縮率は悪化する．しかし，これは観測情報が単に辞書にヒットしないだけであって，圧縮伸張の動作が破綻する事態とはならず，また，実時間測定全体からみた，これらのスタックが保持されない動作は割合が低い[7]ため，全体の圧縮率に対する影響は少ない．

伸張は，圧縮とまったく逆のプロセスによって処理され，伸張側にも圧縮側と同様の辞書を持つ．いま，伸張を図 8 に示す辞書の構造で行っているとすると，伸張側が受け取った圧縮後のデータの最初が 1..111b であれば，それ以降に続く関数情報のビット数分の情報が，そのまま圧縮前の関数情報である．

伸張側が受け取った圧縮データの最初が，1..111b 以外であれば，それは辞書のそのアドレスに格納されている値が，圧縮前の関数情報であるため，伸張が可能となる．なお，この該当する関数情報を検出した際には，圧縮のときと同様に関数の開始・終了フラグを辞書内部で反転させる．

関数の開始・終了フラグを用いることで，関数 id が同じ場合に開始と終了で 2 つのエントリが辞書にできることを防ぐ効果が生まれ，辞書のエントリの数を半分にすることができる．辞書は CAM 等のハードウエア量の多い回路を用いるため，辞書が小さく構成できることは圧縮伸張の回路の小型化へ寄与が大きい．

3.3 3. OS 等の情報の圧縮・伸張

OS 等の情報の圧縮伸張は，時間部分に関しては，前述の時間情報の圧縮と全く同様に処理する．関数情報の圧縮と同様に処理を行う．

情報種別(info type)およびその値(info value) は，関数情報同様に辞書を用いて圧縮伸張を行う．ただし，情報種別とその値には，関数情報のような開始・終了の構造は無いため，辞書にも開始・終了を管理するビットは設けない通常の辞書で構成する．

4. 実装と実験

4.1 ハードウエア実装

前記の手法を用いた実時間測定のための圧縮伸張方法に従い，実際に圧縮器を実装した．ハードウエア実装例として図 9にブロック図を示す．回路へ入力されたデータに応じて，各フィールドが対応する時間情報，関数情報，識別ＩＤ，識別情報のそれぞれが，本手法の処理を行う圧縮部に入力される．その結果，圧縮された結果がそれぞれの圧縮部から出力される．この出力されたデータは可変長であるため，それらを出力 I/F 経由で所望のビット幅となるように変形し出力する．

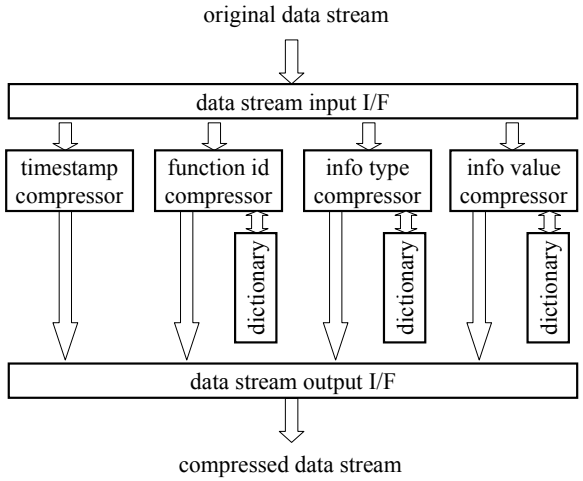

図 9 ハードウエア実装例のブロック図

FPGA へ実装し，実際に動作する組み込みシステムの実時間測定に適用し，そのハードウエア量，動作速度，実行時のバンド幅について求めた．

今回の実装において，データの時間情報のビット数 N_t は 23 ビット，動作情報のビット数 N_b は関数 id では 16 ビット，情報種別のビット数 N_i は 7 ビット，その値を 16 ビットとした．また，辞書のアドレスビット数 N_d を 3 ビットとした．

時間情報を 23 ビットとした理由は，時間情報の最小解像度に対して十分長い差分時間を得られるよう 23 ビットとした．関数 id は 16 ビット，および，動作情報として 23 ビットとした理由は，十分な量の関数の種類を測定可能にすることと，OS 等の情報を取得する際には，OS のタスクスイッチの際のプロセス識別子が 16bit であるためである．

辞書のアドレスビット数 N_d を 3 としたのは，実時間測定中に短時間に大量に観測データが観測されるのは，たとえば 最も内側に存在する for 文の内部であり，その場合，ある特定の関数が大量に呼び出され辞書のヒット率が非常に高くなるためである．

4.2 ハードウエア量の比較

上記の条件のもと回路実装を行った．実装に用いた FPGA は Virtex4lvx15-12[11]であり，論理合成・配置配線は Xilinx 社 ISE[12]を用いた．実装結果の回路量，動作周波数，およびスループットを

表 1に示す．また，比較対照として市販のストリーム型圧縮伸張の IP コアでの結果を合わせて示す．

結果より，比較対象の市販 IP と本手法を回路量の観点で比較すると，スループットは市販の圧縮伸張と変わらない性能を維持しつつ，スライス単体で比較した場合に約 1/3 で実装できていることがわかる．更に，本手法は blockRAM といった大きなメモリセルを用いない．18Kbit をスライス数で換算して比較を考えると，1 スライス[11]には 16bit のＬＵＴが 2 つあることから，1 スライスが 32 ビットの情報量に相当するとして 1 blockRAM は 562 スライスと見積もれる．その場合，比較対象の実装は 3000 スライスを超えることとなり，本手法の回路両派は約 1/10 程度となる．

このように非常に小さな実装が可能となったのは，本手法は実時間測定で必要となるデータ構造に特化した圧縮を，データの内部構造単位で変えるのと同時に，さらに関数の開始・終了といった性質を辞書中に組み込んだ効果と考えられる．

表 1 ハードウエア量およびスループットの比較

	提案手法	市販圧縮 IP[10]（LZRW3 Data Compression ）
回路量	242 Slices 0 x blockRAM(18Kbit RAM)	783 Slices 4 x blockRAM (18Kbit RAM)
動作周波数	300MHz	175 MHz
スループット	1.16Gbps	1.13 Gbps

表 2 圧縮率の評価

	圧縮前の情報量 (bit)	圧縮後の情報量 (bit)	圧縮率
時間情報(timestamp)	49,229,614	23,558,629	47%
関数情報(function id)	33,835,248	18,157,028	53%
識別ＩD(info type)	180,005	77,166	42%
識別情報(info value)	411,440	140,745	34%
全体	83,656,307	41,933,568	50%

4.3 圧縮率と動作情報の評価

実際に動作する組み込みシステム（印刷システムであり，印刷対象の画像をレンダリングするシステム）に本手法を適用し，約 1 秒間測定を行った．その際の圧縮の効果を表 2に示す．

動作情報には，ユーザープログラム中の関数の遷移に起因する関数情報と，OS の動作に起因する識別 ID，識別情報が含まれる．今回の実験では，関数情報が毎秒約 210 万回，OS に起因する情報が 毎秒約 2 万 5 千回発生した．ユーザープログラムの関数の遷移に関する情報の頻度は平均で約 480 ns 程度の間隔で発生していたこととなる．今回の測定における時間の解像度を 10ns としたため，時間情報の下位 7 ビット程度で表現される時間である．今回は時間情報のビット数 N_t は 23 ビットであるため，上位の 16 ビットはほとんどの場合 0 であり，ランレングスを使って上位の 0 のビットを効果的に圧縮できる．

一方，OS の動作情報は，数十 ms の間隔で発生するタイマ割り込みと，ハードウエアによる割り込み等で構成されるため，ユーザープログラム中の関数の遷移より頻度が低い．今回の実験で OS 動作情報は平均 40us の間隔で発生した．この時間間隔は 13 ビットあれば表現可能である．実際には，これらの動作情報は一定間隔ではなくばらつきがあるが，今回用いた時間情報のビット数 N_t =23 に比べ十分に短い間隔でイベントが発生するため，23 ビットで足りなくなることはない．一方，システムがある特殊な状況，たとえば，測定対象のソフトウエアも外部割込みもすべてとまってしまい，OS のタイマ割り込み以外何も発生しないような状況では，OS のタイマ割り込みが数十 ms に 1 度程度しか発生しない．数十 ms の時間を表現するためには，20 数ビットが必要になるため，今回は時間情報のビット数 N_t =23 を選択した．

結果から，各フィールドの特徴に応じた圧縮を行うことで，全体として 50%の圧縮が行われていることがわかる．圧縮は市販 IP においても同程度の圧縮率であり，本手法を用いることでも同程度の圧縮が可能であることが実験からわかった．これより，本手法を用いても実時間測定の際に外部へ約 2 倍の情報を送出可能である．

5. おわりに

本論文では，組み込み機器やマルチプロセッサシステムのリアルタイム検証で必要となる，実時間の測定データをリアルタイムで lossless データ圧縮伸張する方式について提案し，実際に実装と実験を行い，圧縮伸張の結果について報告した，

本手法は実時間測定で必要となるデータ構造に特化した圧縮を，データの内部構造単位で変更して適用することで通常のストリーム型圧縮伸張器よりも 1/10 程度の小規模な回路量で同等の圧縮率を実現した．

より複雑でリアルタイム性を持つ組み込みシステムが実現するにつれ，問題の発見にはより大量の情報を取得し，その中から問題を判定する必要があり，検証は難度を増すと考えられる．今後の課題としては，膨大な情報から検証や問題究明に必要な情報をいかに取得するかを研究する必要がある．

参考文献

[1] J. O. Hamblen, “Rapid Prototyping Using Field-Programmable Logic Devices,” IEEE Micro, pp.29-37 (2000).
[2] N. Ohba and K. Takano, “An SoC Design Methodology Using FPGAs and Embedded Microprocessors,” DAC, pp.747-752, (2004).
[3] Susan L. Graham, et al., “gprof: a Call Graph Execution Profiler,” SIGPLAN, (1982).
[4] IBM Rational Test RealTime™ ‡, http://www.ibm.com/software/rational, (2007)..
[5] ARM Limited, "Embedded Trace Macrocell Architecture Specification" Feb., (2006).
[6] IBM Global Engineering Solutions, “Debugger for PowerPC Processors,” (2007).
[7] K. Takano, N. Ohba, Y. Kohda, and Y. Sakaguchi, “A low-intrusive real-time observation method for tracing function transitions of concurrent programs in embedded systems,” ESS2007, Oct. (2007)
[8] D. Urting and et al., “A Tool for Component Based Design of Embedded Software,” Proceedings of the 40th International Conference on Tools Pacific, pp. 159-168 (2002).
[9] P. Graf and K. D. Muller-Glaser, “ModelScope: Inspecting Executable Models during Run-Time,” ACM International Conference on Software Engineering, pp. 935-936, May (2008).
[10] Helion, “LZRW3 Data compression core”, http://www.heliontech.com/compression.htm, (2009).
[11] Xilinx Inc., "Virtex-4 User Guide," Sep., 2005
[12] Xilinx Inc., “Xilinx ISE Design Suite 10.1 Software Manuals,” (2008).

‡ IBM, Rational は International business Machines Corporation の米国およびその他の国における商標