Design Wave設計コンテスト関連企画

AES暗号回路の実現方式を検討する

――ハード・ワイヤード論理，汎用CPU，コンフィギャラブル・プロセッサの比較

東原朋成

デバイスの記事　ビギナーズ

LSIにCPUコアが組み込まれることがめずらしくなくなった．そこである機能をハードウェアで実現するか，ソフトウェアで実現するかといった検討が重要になる．ここでは，同じ機能の回路をハード・ワイヤード論理で実現するのがよいか，CPUを使ったソフトウェア処理がよいかについての評価事例を紹介する．最近注目を集めているコンフィギャラブル・プロセッサを活用する場合についても評価を行っている．Design Wave設計コンテスト2004の課題であるAES（Advanced Encription Standard）暗号を例題にする．　（編集部）

ハード・ワイヤード論理がどんどん組み込みCPUベースのソフトウェアに置き換えられています．設計が佳境に入った段階でもシステム仕様書があたりまえのように変更されたり，短期開発のプレッシャから検証もそこそこに設計の下流工程（いわゆるレイアウト）が始まってマスク出しに至るようなプロジェクトが跡を絶ちません．そんなASICができ上がってきても，評価ボードで動作させてみればものの5分もしないうちに設計の不ぐあいが見つかってしまいます．そこで，不ぐあいをソフトウェアで修正しようとするわけです．ただし，このアプローチであっても，必ずしも不ぐあいをなくせるわけではなく，結果としてコストが非常に高くついてしまうこともあります．

〔表1〕RTL（ハードウェア）の利点と問題点の整理

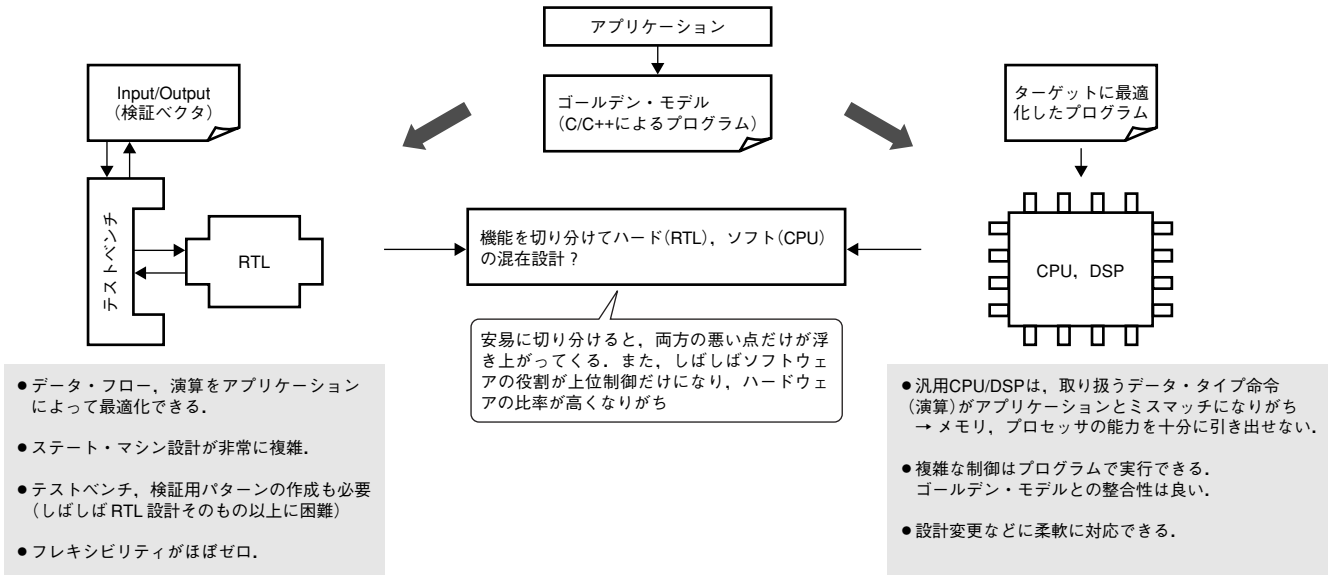
長所	ハード・ワイヤード論理は，レイテンシが最小のMIMDマシン． ●並列処理が基本．設計者の力量しだいで並列度を高められる．つまり，1クロック当たりの処理能力を非常に高められる． ●I/Oの制約が少ない．レイアウトが物理的に可能であれば，データ幅は256ビットでも512ビットでもよく，また一度に10でも20でもメモリをアクセスすることができる．さらに，さまざまなデータを並列で扱える．例えばビット単位のデータ，バイト単位のデータなどをミックスすることも可能． ●並列度を上げ，データ幅を増やすと，データ・レートを非常に高められる．基本的にレイテンシ1ということになる．ただし，クロック・レートの制約はある．
短所	論理の一つ一つまで設計者が責任を持たなくてはならず，設計がきわめて複雑になりがち． ●ステート・マシンの設計が非常に難しくなる．ステート数のみならず，いわゆる入出力信号の数が天文学的に膨れ上がりかねない． ●設計がリジッド．仕様変更などに対処するにはステート・マシンなどの変更を必要とするが，この設計変更はそのままコスト上昇につながる．また仕様変更の可能な範囲が小さく，それを超えてしまうと一から設計し直すことになりかねない．すなわちTAT（turn around time）が長くなる． ●検証が非常に困難．テストベンチ，検証用パターンの作成と評価が設計そのもの以上に困難，かつ長期化している．

ところで，暗号化処理やエラー訂正符号など，比較的制御（ステート・マシン）が単純で，かつデータ・レートの高いディジタル信号処理アプリケーションでは，ハード・ワイヤード論理で実現するRTLベースの設計が今でも主流のようです．例えば，AES（Advanced Encription Standard）の暗号化・復号化処理を取り上げると，RTLベースの設計に軍配が上がると言われています[1]．

一般にアルゴリズムを検討したとき，処理が比較的単純でステート・マシンが複雑にならず，かつ目標とするデータ・レートが比較的大きいような場合，RTLベースの設計が選択されます．ハードウェアというのは，極論すれば究極のMIMD（multiple instruction stream multiple data stream）マシンです．ある特定のアルゴリズムを徹底解析し，コストと性能の両面で最適な論理を組むことが可能です（右掲のコラム「ソフトウェア・ベース設計に対する疑問」を参照）．

●RTLベースの利点と問題点

筆者はフルカスタム設計の時代からという年季の入ったLSI設計エンジニアです．特にRTLベースの設計を長く行っています．そんな筆者の観点でRTL（ハードウェア）の利点



〔図1〕RTL（専用回路）vs. 汎用CPU/DSPベースのソフトウェア

と問題点を整理したのが**表1**です．また，RTLベースの設計とソフトウェア・ベースの設計の違いを**図1**に示します．

RTLベースの設計では，CPUにおける「アーキテクチャ（いわゆる命令セット・アーキテクチャ）」の概念がありません．筆者の友人である“Die-Hard”RTL設計エンジニアはこれをもって「自由だ！」と言うほどです．アーキテクチャに縛られないので，扱うデータ・タイプ，メモリ・インターフェース，演算器の構成などをアルゴリズムに従って自由に設計できるわけです．

ただし，「自由」なことが良いのか悪いのかは考えかたによります．例えば「自由」に代償は付き物です．法律を遵守しない自由というものが治安国家においてありえないように…．ハードウェア設計における「自由」の代償は，設計の複雑度が一気に膨れ上がる，設計変更や将来のアップグレードにおける自由度が少ない，検証が複雑になるなど，多岐にわたります．

最近，このような「代償」が特に問題になっています．設計の最終段階において，仕様変更が入ることが珍しくありません．しかし，残された少ない時間でそれに対応するには，非常に大きなリスクを伴います．例えば，LSI評価段階で不ぐあいが発見されると，大きな手直しが必要になります．最悪の場合，プロジェクトがキャンセルされるこ

ソフトウェア・ベース設計に対する疑問

筆者がLSI設計を始めたころは，LSI設計と言えばフルカスタムでした．トランジスタの一つ一つにまでエンジニアの目が届く，いや届かせなくてはなりませんでした．例えば，筆者がかつて設計に参加したMPEG-1 CODECチップ（0.75 μmプロセス）は，当時としては世界最小かつ消費電力も最小だったはずです（とにかくそれを目標としていた）．符号化アルゴリズムをすべてハード・ワイヤード論理に置き換えました．論理合成も自動レイアウトも行わない，まさしく純粋な手作りでした．その後，筆者もEDAツールを活用してASIC設計を行うようになりましたが，しばらくの間は論理合成ツール能力や既存のライブラリ群の性能がはがゆくてならなかったことを覚えています．

こういった目で見ると，設計に自信のあるハードウェア・エンジニアなら暗号化処理のような論理を，なぜソフトウェア・ベースで設計しなくてはならないのか疑問になります．コーディング・スタイルによってコンパイラがどういった最適化を行うかわからないとか，思ったように性能が出ないと悩むこともあります．

RISC CPUでは，データのロードやストアが頻繁に起こりがちです．そのロードやストアが演算と並列に行えない！などというのは，ハードウェア設計を専門とするエンジニアにとって理解に苦しむものかもしれません．しかもデータやアドレスの計算も必要です．レジスタ（GPRとかARなどと呼ばれる）数やアドレッシング・モードにも制限が付き物です．DSPではこういったことは並行実行されるのが普通ですが，それとてかなり制限されたものです．RTLベースの設計であれば，こういったことをすべて並列実行させることができるのですから．

ともあります．LSI評価段階で不ぐあいが発見されず，市場に出た製品が回収されるような事態になると最悪です．

いちばんの原因は，検証がおろそかになってしまうことです．そこで筆者がしごとをしているシリコンバレーでは，検証エンジニアの需要がよりいっそう高まっています．そして検証エンジニアがプロジェクト・リーダになっていくケースが増えています．

最近では，アルゴリズム（システム）をC，C++などで記述することが多くなっています．いわゆるゴールデン・モデルです．テストベンチを介してゴールデン・モデルとインターフェースしたり（筆者はPerl，Veraを多用する），検証用パターンを作成するといったベーシックなことからリグレッション環境の構築，デバッグ環境（いわゆるモニタ・モジュール．最近はアサーション・チェッカが話題になっている）の構築など，検証システムは非常に複雑になっています．EDAベンダが争ってこの分野でイニシアチブを取ろうとしています．

●CPUベースの利点と問題点

H.264，IEEE 802.11といった標準規格が完全に承認される前に設計を完了させ，市場で主導権を握ろうとアグレッシブに設計を進めるケースがあります．しかし，最終仕様において大きな変更がある場合は，再設計が必要になり，当初のもくろみが果たせなくなってしまいます．このような理由が，ハード・ワイヤード論理からソフトウェアへと機能の実現方法をシフトさせているわけです．

しかし，筆者は設計エンジニアとして，いわゆるDSPやCPUのアーキテクチャにはかなりの不満があります．例えばマルチメディア関連の処理では効率が悪く，必然的にコード・サイズが大きくなるためです（下掲のコラム「汎用アーキテクチャで良いのか？」を参照）．

アルゴリズムが複雑すぎてRTLで設計できないからソフ

汎用アーキテクチャで良いのか？

筆者は，汎用CPU（Alpha，MIPSなど）以外にも，“専用”RISC CPUやDSPの設計に数多くかかわってきました（ただし，いわゆるCISC CPUは設計したことがない）．なぜそのような専用CPUをいくつも設計してきたのでしょうか？

以前，SIMDタイプのコプロセッサ（「COP2」と呼んでいた）をインプリメントしたことがあります．このとき，命令セットを決めたアーキテクトの気持ちが全然理解できませんでした．COP2は，米国Intel社のMMXのようにマルチメディア関係のアプリケーションを意識したアーキテクチャでした．しかし命令セットを見ると，どうしても納得がいきません．そこで，このCPUのアーキテクトとかなりやりとりしたのです．

驚かされたのは「お前の提案を受け入れたらRISCマシンではなくなる…」というセリフです．RISCアーキテクチャの専門家だったがゆえに，彼は「使えない」アーキテクチャを考え付いてしまったようです．例えばCOP2は，2入力8ビット・データの平均を取ることを1命令として実行できませんでした．RISCの考えかたでは，これをADD命令と算術右シフト命令の二つに分解するので，2命令で実行します．ということは実行に2サイクルかかります（パイプラインを無視した場合）．また，プログラム・サイズも必然的に大きくなるのです．

結果的に，COP2は日の目を一度も見ないまま捨てられました（拡張アーキテクチャから消えてなくなってしまった）．インプリメントを担当した筆者と筆者の同僚以外，たぶんだれも覚えていないでしょう．

この例は極端かもしれません．しかし，筆者はMPEGをはじめとするいわゆるマルチメディア関連のLSIをいくつも設計した経験があります．したがって，どのようなデータ・タイプを扱うのか，またどのような命令セットが有効なのかを理解しています．そういった目で見ると，汎用CPUでは目的を達成できないことが予測できたのです．そこで，「専用CPUを自分たちで作ってしまおう」ということになるのです．

ところで，CPUの設計は難しくありませんが，とてもたいへんです．簡単なRISCコアなら大学のプロジェクトでも設計できそうですから，難しいものではありません．それでは何がたいへんなのでしょうか．

まず検証がたいへんです．例えば，例外機能やメモリ保護機能がないような簡単なアーキテクチャのプロセッサですら，検証することはたいへんです．

ソフトウェア開発環境（OSなども含む）の開発はさらにたいへんです．筆者が以前開発した専用組み込みRISCコアやDSPは，簡単なアセンブラを開発するのがやっとでした．また，非常に簡単なリアルタイムOSを開発することで精いっぱいという状況でした．コンパイラのように高度なものを開発するとなると非常にたいへんです．開発したものには保守がつきまといますが，これはもっとたいへんなことです．

結局，巨大企業ならともかく，CPUの内製というのはコストの見合わないものになりがちです．

汎用DSPや汎用CPUではシステム設計で思ったように使えない，だからといって自分たちでDSP/CPUを開発するというのもコスト面で割が合わない…．となると残るはやはり，RTLベースで設計ということになるのでしょうか．

トウェア・ベースに移行している（移行する必要がある）という意見もあります．しかし，ハードウェアだろうと，ソフトウェアだろうと，同じ処理であれば複雑度という点ではなんら変わりませんから，これは「単純すぎる見解」と言わざるを得ません．

●コア・プロセッサを拡張するコンフィギャラブル

コンフィギャラブル・プロセッサIP（例えば米国Tensilica社の「Xtensa」，そして英国ARC International社の「ARC Tangent」）が最近知られるようになっています．これらに刺激される形なのでしょうか，「コンフィギャラブル」ということばが数々の記事，論文に登場しています．

コンフィギャラブル・プロセッサのイメージを**表2**に示します．汎用CPU/DSPとは異なり，メモリ構成，外部インターフェースなどをコンフィグレーションします（**図2**）．メモリの構成などは，ISS（instruction set simulator）を使って解析したキャッシュのヒット率などをもとに決定していきます．アルゴリズムの検討によって，新しいデータ・タイプや命令語などが拡張されることになります．Xtensaの場合，プロセッサ・コアのほかに，拡張命令作成のための言語，ソフトウェア開発環境（コンパイラ，デバッガ，ISSなど），各種リアルタイムOSへのHAL（hardware abstraction layer），システム・シミュレーションのためのバス・インターフェース・モデル，FPGA評価ボードといった開発のツールがすべてそろっています[2]．

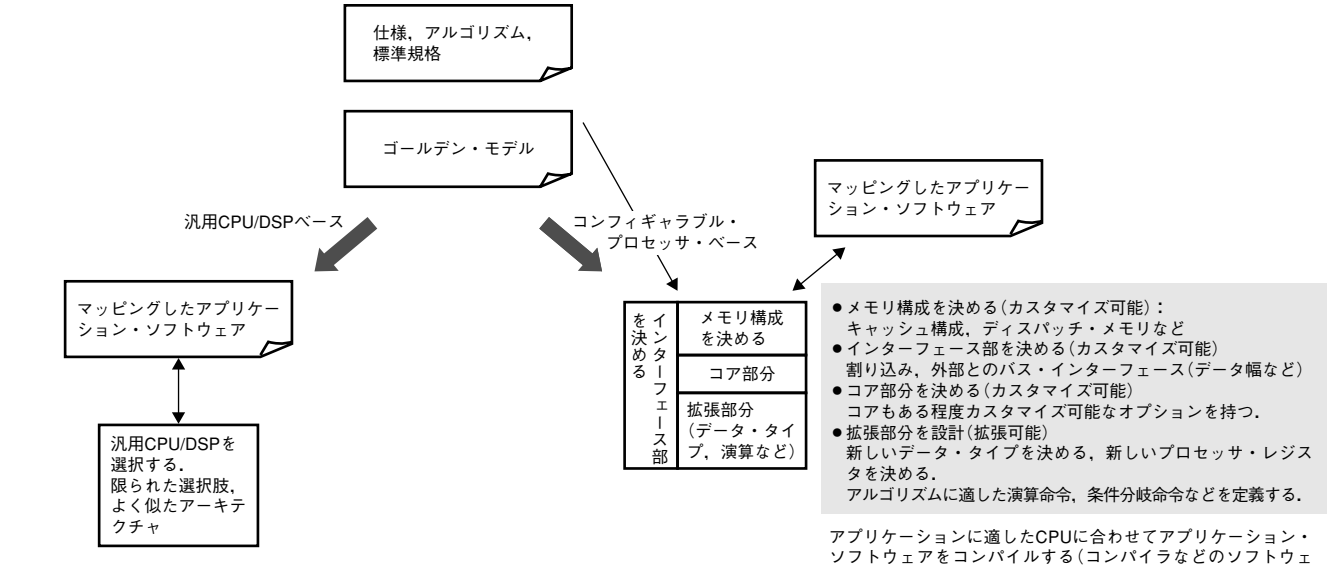
AES暗号回路で比較してみる

AES暗号化回路をRTLベース，汎用CPU（x86）ベース，コンフィギャラブル・プロセッサ（Tensilica社のXtensa）ベースで設計してみました．筆者の設計結果を**表3**にまとめます．

ここでは話を簡単にするため，かぎの長さを128ビット固定にしました．AESを例に取り上げたのは，AESが一見してRTLベースの設計に適しているからです．もし，MPEG-4ビデオCODECを例にすれば，デコード処理方式

〔表2〕コンフィギャラブル・プロセッサのイメージ

基本構成から変更可能な機能	●メモリ容量 ●メモリ構成（キャッシュなど） ●バス（I/O）のデータ幅 ●汎用レジスタ（GPR）の数 ●外部，内部割り込みの構成 ●コプロセッサ ●デバッグ用ハードウェアの追加 ●MMUの追加，構成 ●コア・アーキテクチャのオプション命令セットの選択 ●プロセッサ・ローカル・データ・メモリへのDMA
拡張機能	●アプリケーションに最適な命令の追加 （追加できる命令数はコア・アーキテクチャの命令数以上） ●命令語長（制約はある） ●汎用レジスタ（GPR）の追加 ●プロセッサ・ステート・レジスタの追加 ●新しいデータ・タイプの定義

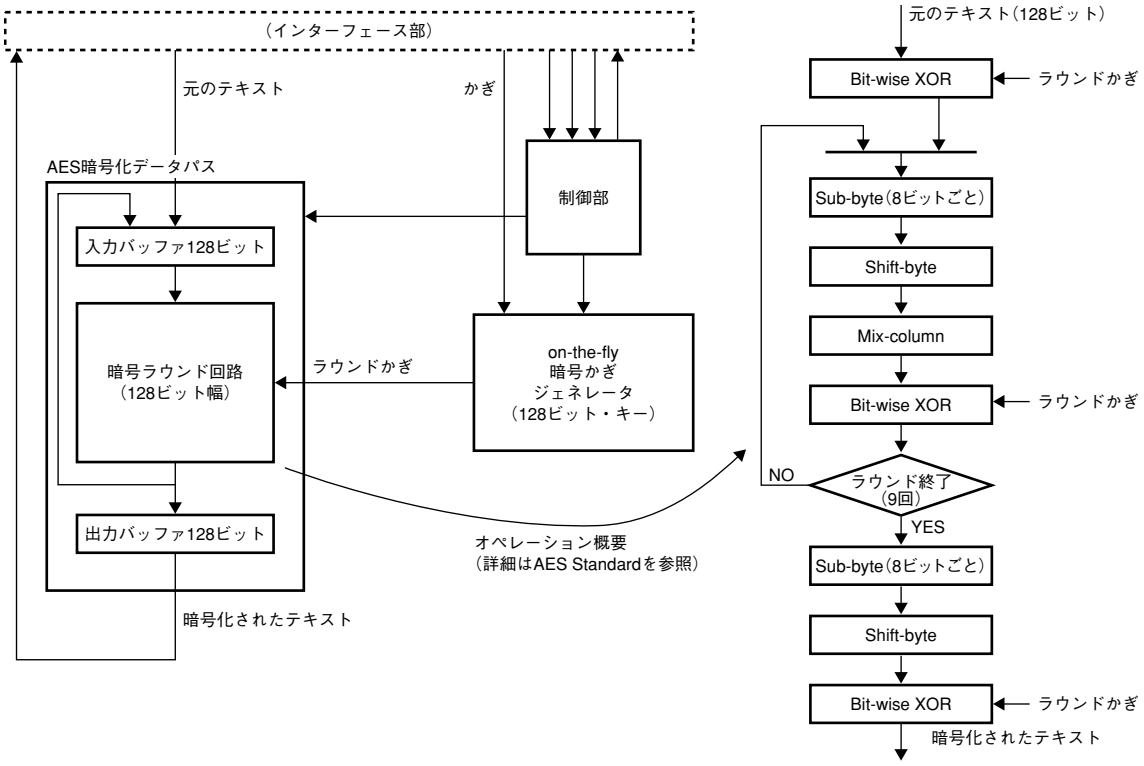


〔図2〕汎用CPU/DSP vs. 拡張可能なCPU

〔表3〕AES暗号化システムの設計比較（かぎ長が128ビット，テキスト長が128ビットの場合）

	RLTコード（コア）	市販のデバイス・データから（チップ）	コンフィギャラブル・プロセッサ			IA64	TMS 320C6201	Pentium Ⅲ
			かぎの同時生成も実行	中間的な拡張	簡単な拡張			
クロック数（サイクル）	11	11	11 Cコンパイラ	88 Cコンパイラ	154 Cコンパイラ	124 IA64 アセンブリ言語	228 アセンブリ言語	352 Gnu C（Linux）
クロック周波数（MHz）	200	200	200	400	450	2000	600	800
データ・レート（Gbps）	2.91	2.91	2.91	0.916	0.525	2.58	0.422	0.364
回路規模（kゲート）	13.5（エンコーダ）	14（エンコーダ）	35.1（エンコーダとデコーダ）	22（エンコーダとデコーダ）	10（エンコーダとデコーダ）			

注1：IA64，TMS320C6201のデータはインターネットの検索による．
注2：Xtensaの場合の回路規模は拡張部のみ．ここではエンコーダとデコーダの拡張命令を同時に論理合成した結果．



〔図3〕AES暗号化のRTLブロック図

から考えて，ソフトウェア・ベースの設計に分があるでしょう．ここではあえてRTLベースの設計に分がありそうなアプリケーションを使って設計結果を検討してみようと思います．

●RTLベースの設計

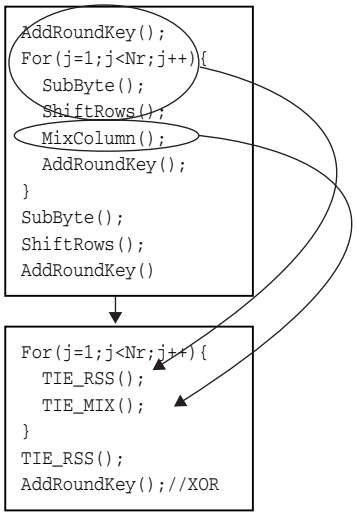
RTLベースのブロック図を図3に示します．データパスを128ビットとし，かぎの更新と暗号データの繰り返しループを同時に実行します．

アルゴリズムをすなおにそのまま回路にマッピングしたといえます．

●汎用CPUベースの設計

汎用CPUベースのソフトウェア開発では，演算はできる

〔リスト1〕AES暗号化ステップを検討して拡張命令を定義



TIE_RSS，TIE_MIXCOLという命令を定義した．
ここでは一度に32ビットのデータを扱うことにしたが，128ビットを一度に扱うように拡張することもできる

暗号化アルゴリズムを解析すると，

1）演算は比較的簡単．
2）演算はループを伴う．

というわけで二つの命令を定義することにした．
データパス幅を32ビットに決めた（128ビットでもよい）

演算がループを伴うので演算途中のデータはすべてレジスタに保持するようにした．演算を効率良く実行するために新たにステート・レジスタをいくつか定義した．

ラウンドかぎはあらかじめ拡張しておいてメモリにセーブすることにした．この点，ハードウェアで実現したon-the-flyかぎ拡張と異なる．
ラウンドかぎをレジスタにロードする必要があるが，32ビット幅のデータパスでは1回のラウンドにロード命令が4回必要になり，性能を下げてしまう・
そこでTIE_MIX命令を，このkey loadを演算と同時に実行するように定義した．

〔リスト2〕AESに適した命令語を追加したXtensaによるAES暗号化ルーチン例

```
132i
132i
132i
132i        } 128ビット・データをレジスタにセットする
aes_rss     a2, 3, 0
aes_rss     a4, 2, 1
aes_rss     a2, 1, 2
aes_rss     a4, 0, 3
Aes_mix     3, 3  //平行してキーのローディングが行われる．
Aes_mix     2, 2
Aes_mix     1, 1
Aes_mix     0, 0
    繰り返し（全部で9回）．ここで繰り返し命令を使うとコード・サイズを抑えられるが，ここではループを展開してもよい．

aes_rss     a2, 3, 0
aes_rss     a4, 2, 1
aes_rss     a2, 1, 2
aes_rss     a4, 0, 3    } 最終

xor
xor
xor
xor         } 最後にラウンドかぎとXORを取る．

s32i
s32i
s32i
s32i        } 暗号化されたテキストをメモリに戻す．
```

128ビットのテキストを暗号化するのに必要なサイクルは4＋9×8＋4＋4＋4クロック・サイクル（88クロック）となる．
データパスを128ビットに拡張すればクロック数は1＋9×2＋1＋1＋1クロック・サイクル（22クロック）となり，クロック数がRTL設計の暗号化回路の2倍になる．さらにAES_RSSとAES_MIXを1命令AES_UNITと定義し直すと，クロック数をさらに少なくできる．
拡張データパスの回路規模，プロセッサのクロック周波数と暗号化レート（目標とする性能，コスト）の検討が必要．

かぎりルックアップ・テーブルで行うことにしました．例えば，最近のx86プロセッサは非常に大きなキャッシュ・メモリを持っていますが，128ビット・データをそのまま扱うことはできません．また，体（field）演算（特に掛け算）命令を備えていません．そこで，あらかじめ演算した結果をキャッシュ・メモリに置いておき，メモリを参照することで演算を実行します．AESでは128ビット・テキストをバイト単位で扱うので，ルックアップ・テーブルが大きくなります．しかし組み込みCPUと比べると非常に大きなメモリを使用できるx86プロセッサでは問題ありません．

こういったことを「ソフトウェアのアーキテクチャへのマッピング」と呼びます．ゴールデン・モデルがアルゴリズム（システム）を忠実に記述しているのに対して，マッピングされたソフトウェアは設計目標（性能，コストなど）を満たさなければなりません．

●コンフィギャラブル・プロセッサ・ベースの設計

コンフィギャラブル・プロセッサによる実現方法をリスト1とリスト2に示します．Xtensaでは，性能を重視する拡張，簡単な拡張，中間的な規模の拡張の3種類を評価し

ましたが，ここでは中間的な方法を説明します．

Xtensaは32ビットCPUですが，128ビット長のデータを直接扱えるように拡張可能です．しかし今回は，拡張部も32ビット構成としました．AESではバイト単位でデータを扱いますが，ローテート（AESではshift-rowと呼んでいる）は32ビット・データをユニットと考えるとよさそうです．AES_RSSという命令ではデータパスを8ビット，32ビットの単位で扱うようにしました．

この命令拡張により，128ビット・データを88サイクルで暗号化できます（Cプログラム）．データパス幅を128ビットにすれば，容易に22サイクルにすることができます．

●性能の評価

さて，**表3**のようにクロック数で比較するとRTLベースでは128ビット・テキストを暗号化するのにきっちり11クロックかかっています．パイプライン化していないので，11クロックごとに新しいテキスト・データを取り込むことができます．クロック・レートが200MHzとして，2.91Gbpsのデータ処理量となります．パイプライン化も容易ですが，回路規模はかなり大きくなります．

汎用CPU/DSPベースの設計例（x86など）では128ビット・データの暗号化に数百サイクルかかっています．処理を最適化するためにアセンブリ言語で記述した例もありますが，これはAESのように比較的単純なアルゴリズムだからできたワザといえそうです．

ところで組み込み機器に高価な高性能汎用CPUを採用することはあまりありません．かりに採用しようものなら，AES暗号化といった単純な処理のみならずシステム・コントロールなどを受け持たされるでしょうから，実質，暗号化レートは**表1**のデータ・レートの数%にとどまるでしょう．

コンフィギャラブル・プロセッサ・ベースの場合，RTLベースとほぼ同等の性能を達成させることができました．しかも暗号化キーのon-the-fly生成も同時に実行できます[3]．しかし，RTLベースと比較すると回路規模的には大きくなります．

コンフィギャラブル・プロセッサによって実現する場合の回路規模は，RTLベースと比較すると同等かもしくはまだ大きいといえるかもしれません．ただし，この命令拡張は暗号化だけでなく復号化にも対応できるものです．暗号化だけの命令にすればより少ないゲート数ですむでしょう．

●暗号化だけならRTLベースが優れているが…

暗号化データ・レートとゲート数を比較すると，RTLベースの設計がいちばん優れていることがわかります．ということは，AES暗号化回路のようにステート・マシンが比較的単純でかつ高データ・レートを目標とするような場合，RTL（ハードウェア）による設計を取るべきだということになるのでしょうか？

もしAES暗号（復号）化のみを行うASICを設計するというのであれば，RTLベースで設計するべきと言えるかもしれません．しかし，暗号化するデータ，そして暗号化されたデータはどこから来てどこへ行くのでしょうか？　AESの機能ブロックをそれらとどのようにインターフェースさせるのでしょうか？　システム全体を十分に検討することなしに安易にハードウェア化してしまうと，システム全体の設計，検証の困難を招くことにもなりかねません．システム全体で考えると，AESのようにRTLベースの設計が比較的単純で，高い性能の得やすいブロックでも，ソフトウェアで設計するほうが好ましいケースもあります．

ソフトウェア化する場合には，汎用CPUを選択するのか，コンフィギャラブル・プロセッサを選択するのかという検討も必要です．

参考文献

1）佐藤証，森岡澄夫，「暗号処理のソフト vs. ハード」，『Design Wave Magazine』，pp.72-79，2003年9月号．
2）Chris Rowen，「HDL記述からCコンパイラまで応用に合わせて自動生成――プロセッサ・ジェネレータ“Xtensa”」，『Design Wave Magazine』，pp.92-101，1999年12月号．
3）Xtensaのアプリケーション・ドキュメント，http://www.tensilica.com/．
4）東原朋成，「コンフィギャラブル・プロセッサの大規模PLDへの実装――Xtensaによる VoIP設計事例」，『Design Wave Magazine』，pp.67-77，2000年10月号．

とうはら・ともなり