SecHack365

AESアクセラレータと自作CPU

学習駆動コース　砂場遊び開発ゼミ

伍藤豪

概要

FPGA上にSystem Verilogでフルスクラッチで実装している.

✓RISC-Vの32bit整数演算が可能なCPUを自作した.

✓AES-128の暗号化/復号化回路をアクセラレータとして実装

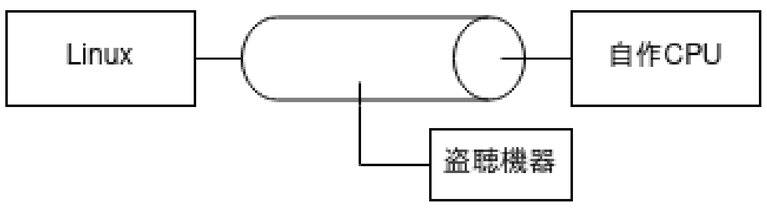
✓マルチリングオシレータによる真性乱数生成器を実装

✓Memory Mapped I/OによるUART通信を実装

→UART通信上でAES-CBCによる暗号化通信を実証

盗聴されても安全な通信を実現

ソースコードや拡張命令一覧: https://github.com/gotti/shrv32



GitHub

背景　ー　暗号化の負荷

- 高速な実装でもCPUでは暗号化に160命令以上必要である.
- 組み込み機器に計算させると遅い.

→そこでアクセラレータを搭載してハードウェアによる並列計算を行い，計算に要する時間を削減する.

FPGAはデジタル回路を構成/再構成可能なデバイス
自作CPUなどの論理回路を実装し現実世界で試すことができる.
並列化が可能な計算において理論的には専用の論理回路を作りそれを増やせば増やすほど並列して計算ができる.
しかし機種により回路を作れる面積が違い，この面積はLogic Elementなどの単位で表される．今回用いたCYC1000は4千円ほどで買えるボードであり，FPGAの中ではかなり安いものである.

動機

●CPUを自作してみたかった.

- CTFなどでアセンブリ言語には触れていたためCPUがどう動いているのか気になった.

●AESはハードウェアで高速化しやすく試してみたかった.

AESアクセラレータの設計

●右上に示すブロック図のようになった．処理をまとめて事前計算しテーブル化する最適化手法は今回利用していない.

●KeyExpandは暗号化/復号化で共用することで回路規模の削減を図った.

- AES暗号は10ラウンド掛けて平文と暗号鍵を混ぜていく.
- ラウンドごとにちょっと変えた暗号鍵を用いる．(ラウンド鍵)
- KeyExpandは前のラウンド鍵から次のラウンド鍵を求める.
- これにより暗号化ではラウンド鍵の計算と暗号化を並列で行える.
- 復号化の所要時間が暗号化の2倍の20クロック必要なのは，復号化ではラウンド鍵を逆順に用いるため事前にラウンド鍵を計算するから.

自作CPU全体としての設計

●RISC-VのRV32I (32bit整数演算命令)を採用

●アクセラレータ用の256bit拡張領域&拡張命令を搭載

●標準的なハーバードアーキテクチャで実装

●Intel CYC1000 (Cyclone 10 LP) Quartus Prime 20.1.1

●Total logic elements : 24,026/24,624(98%)

拡張領域の設計

●RISC-Vに定義されているカスタム命令に沿った拡張命令が実装されており，これを使うと拡張領域を利用できる.

●拡張命令のフォーマットはRV32IのR-typeもしくはI-typeのどちらかに準拠し簡素にした.

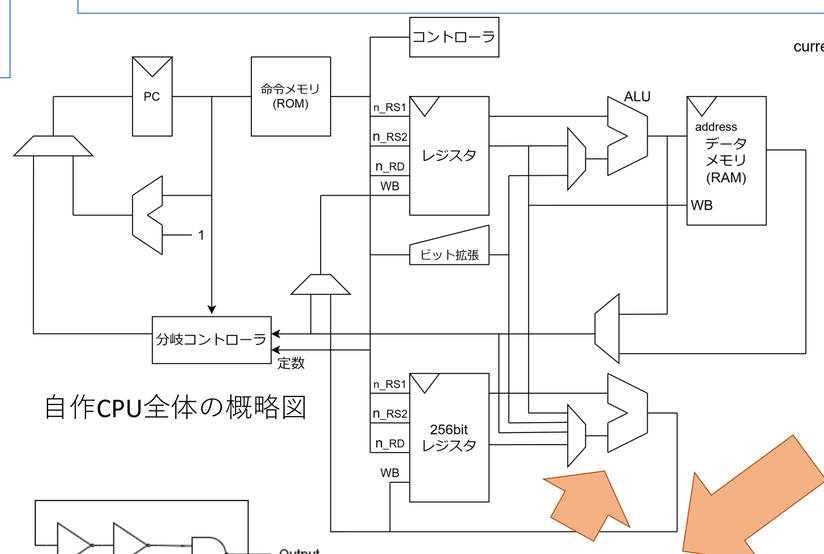
●256bitレジスタを6本搭載

●専用のALUにおいて256bitでの基本的な演算が可能

- 加算やXOR，シフト命令などを搭載

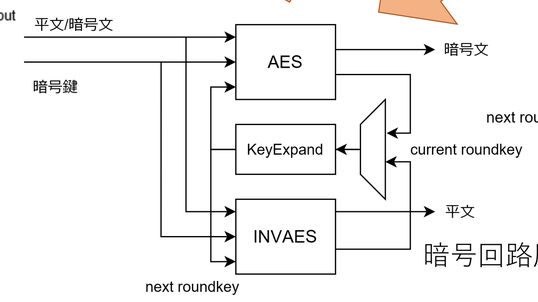
●AESアクセラレータはこのALU内部に搭載している．平文と暗号鍵をALUに渡すことで暗号文がALUから出力される.

●CPU本体とはいくつかのバスで接続しており，複数ある拡張命令でデータのやりとりが行える.

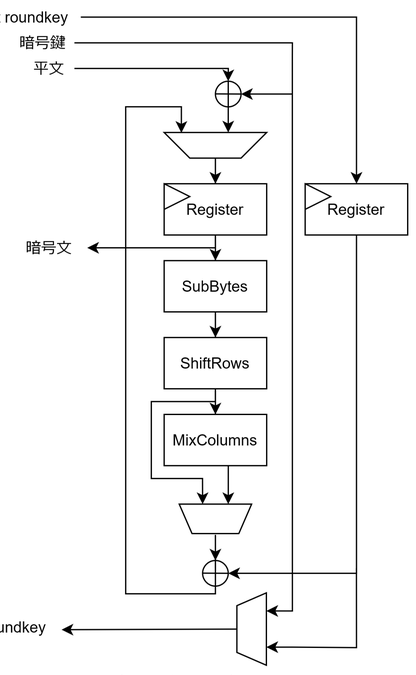


自作CPU全体の概略図

リングオシレータ
周辺の環境などによる誤差を持ちつつ発振し乱数源となる.
これを百個ほど並べて乱数を生成する.



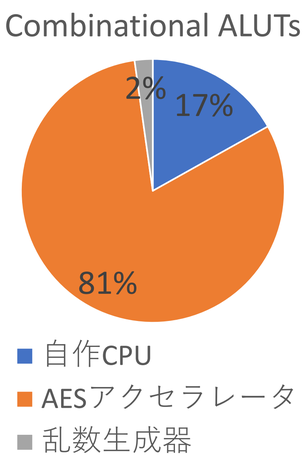
暗号回路周辺の関係



AES暗号化回路

今後の課題

- パイプライン化などCPU本体の高速化
- 現状LEの大半を占めるAESアクセラレータの省スペース化
- 冗長になっている拡張命令の最適化
- AES以外のアクセラレータの搭載
- サイドチャネル攻撃に対する検証
- 乱数の検証



国立研究開発法人 情報通信研究機構