Dokumentation: Aes Erweiterung für die RISCV-Pipeline

In diesem Projekte wurde die RISCV Pipeline um ein Aes Modul erweitert.

Die Erweiterung beschränkt sich dabei auf AES-128 Ver- und Entschlüsselung

Umsetzung

Das Aes Block Diagram

> Das Aes Modul wurde als eigenes Block Diagram umgesetzt. Im folgenden werden die einzelnen Komponenten des Block-Diagramms kurz erläutert.

- *AESKey*
    - Dieses Modul gibt den fixen Schlüssel zurück, welcher für die Ver- und Entschlüsselung benutzt wird.
    - Da der AES-Schlüssel fix im Modul ist und die einzelnen Rundenschlüssel, welche für jede Ver- und Entschlüsselungs Runde benötigt werden, nur von diesem Schlüssel abhängen, wurde der Algorithmus zur Schlüsselexpansion (welcher die einzelnen Rundenschlüssel generiert) nicht implementiert. Das AESKey Modul gibt also auch die fixen (im vorraus berechneten) Rundenschlüssel zurück.
- *AesAddRoundKey*
    - Dieser Block wird sowohl für die Verschlüsselung (im ersten Schritt), als auch für die Entschlüsselung (im letzten Schritt) benutzt und entspricht der AddRoundKey-Operation der AES-Verschlüsselung
    - In der Praxis handelt es sich um eine XOR-Operation der aktuellen Cypher und dem zugehörigen Rundenschlüssel
- *AesEncryptionRound*
    - Dieses Modul spiegelt eine einzelne Verschlüsselungsrunde im Aes-Algorithmus wieder. Die Verschlüsselungsrunde besteht aus folgenden 4 Funktionen
        - SubBytes
        - ShiftRows
        - MixColumns
        - AddRoundKey
    - Die einzelnen Funktionen die in dem Modul benutzt werden, sind im *AesEncryptionOperations* Package bzw. im *AesGeneralOperations* Package implementiert.
- *AedEncryptionLastRound*

    - Enspricht dem Module *AesEncryptionRound* bis auf die fehlende MixColumns Operation und spiegelt damit die letzte Verschlüsselungsrunde des AES-Algorithmus wider

- *AesDecryptionRound*

    - Implementiert die inversen zur *AesEncryptionRound*, d.h. es besteht aus den Methoden
        - AddRoundKey
        - InvMixColumns
        - InvShiftRows
        - InvSubBytes
    - Die Methoden sind im *AesDecryptionOperations* Package, bzw im *AesGeneralOperations* Package implementiert

- *AesDecryptionFirstRound*

    - Enspricht der inversen *AesEncryptionLastRound*

Das AesModul ist Rundenweise gepipelined. Es dauert also 10 Takte bis nach dem Start einer Ver- oder Entschlüsselung das Ergebnis zur Verfügung steht.

Einbindung in die RISCV Pipeline

Das Aes Modul ist neben der Execution-Stage in die RISCV-Pipeline integriert. Eine davorgeschatelte *AesStageStart* sorgt dafür, dass alle benötigten Signale an das AES Moduls weitergegeben wird. Eine *AesStage* hinter dem Aes Modul sorgt dann wiederum dafür, dass die Ver- bzw. Entschlüsselten Daten in die entsprechenden Register geschrieben werden. Dafür kommuniziert diese Stage auch mit dem RegisterSet Für das schreiben in das Register wird die komplette Pipeline um eine Takt verzögert, um sicherzustellen dass nicht auf dasselbe Register in einem Takt doppelte geschrieben wird.

Für die korrekte Anbindung wurde das *Decode* Modul erweitert um die unten definierten Befehle nun zusätzlich zu dekodieren.

Desweiteren wurde das *RegsiterSet* sowie das *Forward* Modul angepasst, sodass sie für die aus dem *Decode* Modul kommende *AesSrcRegNo* die entsprechenden Daten zurück geben und diese auch geforwarded werden.

> Note: Die Anbindung der *AesStage* an den Speicher wurde leider nicht vollständig fertiggestellt.

Implementierte AES Befehle

Das *Decode* Modul decoded folgende zusätliche Befehle:

```
    csrw 0x1, rs
```

Dieser Befehl verschlüsselt die Register rs, rs+1, rs+2, rs+3. Da Aes-128 auf 128-Bit großen Datenworten arbeitet, werden diese einfach hintereinander zusammengesetzt und als 128 Bit Wort dem Aes Modul

übergeben.

```
    csrw 0x2, rs
```

Dieser Befehl entschlüsselt die Register rs, rs+1, rs+2, rs+3. Es bildet also die Umkehroperation zum oberen Befehl ab.

> Note: Bei der Programmierung ist darauf zu achten, dass beide Befehle nicht blockierend implementiert sind. Das heißt nachdem die Ver- bzw. Entschlüsselung fertiggestellt ist, wird das Ergebnis in die Register geschrieben, unabhängig davon ob sich die Register danach schon geändert haben.
> Die Entscheidung diese Befehle nicht blockierend zu machen beruht darauf, dass auf diese Weise die Pipelining funktionalität des Aes-Moduls getestet werden kann.

Angefangene Erweiterungen (bisher nur auf Branch master verfügbar und noch nicht vollständig fuktional)

```
    csrs rs1, rs2

Beispiel:
    csrs 0x01, x10
```

Dieser Befehl enspricht einem *store encrypted*. Er soll den Inhalt der Register rs1, rs1+1, rs1+2, rs1+3 verschlüsseln und dann den verschlüsselten Inhalt an der Speicheraddress die in rs2 steht schreiben.

```
    csrc rs1, rs2

Beispiel:
    csrc 0xf, x13
```

Dieser Befehl enspricht einem *load decrypted*. Er stellt wieder die Umkehroperation zum oberen *store encrypted dar*

Beide Befehle werden von der Decode Stage korrekt dekodiert. Für den *store encrypted* Befehl hat die *AesStage* am Ende des Aes Moduls die Logik um das Ergebnis im Ram zu speichern. Für *load decrypted* übernimmt diese Logik die *AesStageStart* am Anfang des Aes Moduls