Praktikum 4, Teil 1 – AES-Algorithmus als Acceleratorblock im FPGA

Lernziele

- Sie können aus einem Programm unter Linux auf die Control- und Statusregister eines Peripherieblocks im FPGA zugreifen.
- Sie nutzen den AES-Block im FPGA erfolgreich für die Ver- und Entschlüsselung einer Nachricht.

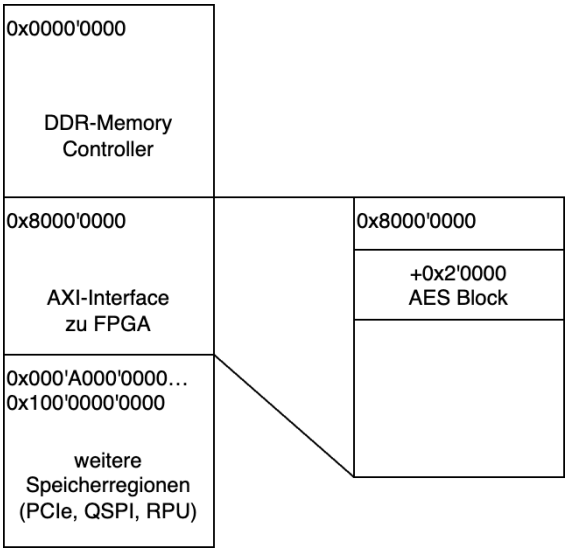
Grundlagen

Peripherie im FPGA

Der AES-Block wurde bereits im FPGA für Sie implementiert. Der FPGA-Teil wird bereits beim Bootvorgang konfiguriert. Ab dann ist die programmierbare Logik einsatzbereit. Der AES-Block steht Ihnen daher wie ein Peripherieblock zur Verfügung, wie Sie es von CT gewohnt sind: über Control- und Statusregister.

Da der Prozessor des Ultra96-Boards über eine Memory Protection Unit verfügt und unter Linux läuft, darf ein Programm nicht einfach auf jede Adresse zugreifen, wie das auf dem CT-Board möglich war. Sie müssen zuerst über den Kernel die Berechtigung erlangen, auf den entsprechenden Speicherbereich zuzugreifen. Sie erhalten dann vom Kernel einen Pointer für den Zugriff auf den gewünschten Adressbereich. Dieser Pointer zeigt auf eine virtuelle Adresse, weshalb der Wert des Pointers nicht der angefragten Adresse entspricht.

Die Anbindung des FPGAs über das AXI-Interface definiert die Basisadresse des Bereichs der Memory Map, wo die FPGA-Komponenten erreichbar sind: 0x8000'0000. Innerhalb dieses Bereichs werden die einzelnen Peripherieblöcke vom FPGA-Entwickler in einer Memory Map platziert. Der AES-Block hat den Offset 0x2'0000.



Registermap

Die Register des AES-Blocks werden im Folgenden beschrieben.

0x08: Control-Register

Write-only

Das Control-Register dient dazu, die Key-Expansion sowie die Block-Verarbeitung zu starten.
ACHTUNG: da das Register Write-Only ist, können einzelne Bits nicht via read-modify-write-Vorgang gesetzt werden:

```
*reg |= 0x01; // wird NICHT funktionieren
```

Es muss immer das ganze Register auf die gewünschte Einstellung geschrieben werden. Das Setzen eines Bits im Register entspricht einer Aufforderung, einen bestimmten Schritt in der FPGA-IP auszuführen. Führen Sie immer nur einen Schritt aufs Mal aus, damit die Reihenfolge klar ist.

Bit 0: 'INIT'
    schreiben einer '1' startet die Key-Expansion
Bit 1: 'NEXT'
    schreiben einer '1' startet die Block-Verarbeitung
Bits 2..31: unbenutzt
    '0' schreiben

0x09: Status-Register

Read-only

Bit 0: Ready-Bit.
    '0': not ready,
    '1': ready (Key expansion successful)
Bit 1: Valid-Bit.
    '0': Resultat im Result-Register ist nicht gültig,
    '1': Resultat ist gültig.

0x0A: Config-Register

Write-only
Das Config-Register dient der Konfiguration der Schlüssellänge und der Wahl der Ver- oder Entschlüsselung.
ACHTUNG: da das Register Write-Only ist, können einzelne Bits nicht via read-modify-write-Vorgang gesetzt werden:

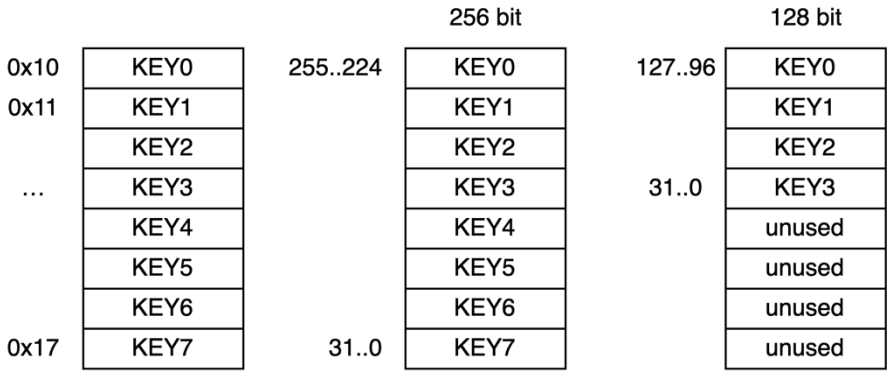
```
*reg |= 0x01; // wird NICHT funktionieren
```

Es muss immer das ganze Register auf die gewünschte Einstellung geschrieben werden.

Bit 0: Encrypt/Decrypt-Modus:
'0' Decrypt
'1' Encrypt
Bit 1: Schlüssellänge:
'0' 128-Bit Key
'1' 256-Bit Key

0x10-0x17: Key-Register KEY0 bis KEY7

Write-only
Das Key-Register nimmt den Schlüssel auf, verteilt auf vier resp. acht 32-Bit Wörter, wie in der folgenden Grafik gezeigt:



0x20-0x23: Block-Register

Write-only
Das Block-Register nimmt den zu verarbeitenden, 16 Byte grossen Block der Nachricht in vier 32-Bit Wörtern auf.

0x20	BLOCK0	127..96
0x21	BLOCK1	95..64
0x22	BLOCK2	63..32
0x23	BLOCK3	31..0

0x30-0x33: Result-Register

Read-only
Das Result-Register nimmt das 16 Byte grosse Resultat der Operation in vier 32-Bit Wörtern auf.

0x30	RES0	127..96
0x31	RES1	95..64
0x32	RES2	63..32
0x33	RES3	31..0

0x40-0x43: IV-Register

Write-only

Das IV-Register nimmt den 16 Byte grossen Initialisierungsvektor der Nachricht in vier 32-Bit Wörtern auf.

0x40	IV0	127..96
0x41	IV1	95..64
0x42	IV2	63..32
0x43	IV3	31..0

Benutzung des AES-Blocks

Mit folgendem Vorgehen können Sie den AES-Block nutzen:

1. Key laden, Key-Länge und Encrypt/Decrypt setzen.
2. Key Expansion starten (INIT), warten auf Ready-Bit im Status-Register.
3. Plaintext-Block und IV in die entsprechenden Register schreiben.
4. Block-Verarbeitung starten (NEXT).
5. Warte auf Valid-Bit im Status-Register, Resultat auslesen.

Aufgabenstellung

Die Source Files für den Versuch liegen im Git-Repository ESE_HS25_stud, im Verzeichnis P4. Sie benötigen den Cross-Compiler für den A53-Prozessor.

Im vorbereiteten Programmrahmen finden Sie Macros für die Registeroffsets und die Control- und Config-Bits, sowie Kommentare für die nötigen Schritte einer xcrypt-Funktion. Teile der Funktion AES_FPGA_xcrypt_buffer(…) sind bereits implementiert.

1. Mit welchem Vorgehen wird der Zugriff auf den Speicherbereich beantragt?
   _____

2. Stellen Sie die Implementierung der Funktion AES_FPGA_xcrypt_buffer(…) fertig, sodass die Funktionsaufrufe in main.c erfolgreich durchlaufen.

   Hinweis: Das Programm muss mit Root-Berechtigungen laufen, damit wir auf die Speicheradresse der AES-Einheit im FPGA zugreifen dürfen. Starten Sie das Programm deshalb mit *sudo ./fpga.elf*.

3. Messen Sie die Ausführungszeiten für die Verschlüsselung und Entschlüsselung einer 16 Byte langen Nachricht.

4. Optional: Falls Sie der Ehrgeiz packt, ändern Sie die Implementierung so ab, dass die Software auch längere Nachrichten verarbeiten kann. Bauen Sie dazu eine Schleife, um die Nachricht in 16-Byte-Blöcken zu verarbeiten. Die Initialisierung führen Sie nur vor dem ersten Block aus. Messen Sie auch dafür die Ausführungszeiten.

Bewertungskriterien:

- Ihre Software konfiguriert und initialisiert den AES-Block im FPGA erfolgreich.
- Ihre Software nutzt den AES-Block korrekt für Ver- und Entschlüsselung.
- Sie wählen sinnvolle Punkte im Programm für Ihre Zeitmessungen.
- Sie haben vertrauenswürdige Messwerte für die Ausführungszeiten des AES-Algorithmus im FPGA protokolliert.
- Sie können den Ablauf der Software erklären