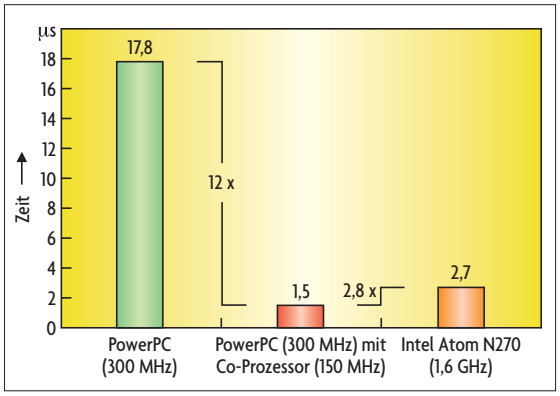
AES-Beschleunigung über Befehlssatzerweiterung:

Die eigene Crypto-Engine im Xilinx-FPGA

Die zunehmende Vernetzung von Embedded-Systemen stellt ganz neue Herausforderungen an solche Systeme. Um die Übertragungen mittels TCP/IP, WLAN oder VPNs abzusichern, findet oft der Advanced-Encryption-Standard (AES) Verwendung. Der hohe Rechenaufwand für diese Verschlüsselungsmethode bringt die Prozessoren allerdings schnell an ihre Leistungsgrenzen. Zum Glück kann bei FPGA-basierten System-on-Chip-Plattformen eine Steigerung der Leistungsfähigkeit über Co-Prozessoren im FPGA erfolgen.

Die PowerPC-Architektur in den Xilinx-FPGAs Virtex II-Pro, Virtex4-FX und Virtex5-FXT bietet für die Co-Prozessoren eine leistungsfähige Schnittstelle: Die Auxiliary Processor Unit (APU), an die Co-Prozessoren angeschlossen werden können, ohne dass der Systembus zusätzlich belastet wird. Für die AES-Verschlüsselung wurde solch ein Co-Prozessor als Crypto-Engine für eine FPGA-Plattform implementiert und analysiert. Die so beschleunigte AES-Verschlüsselung ist um den Faktor 12 schneller als die reine Software-Implementierung auf dem gleichen Prozessor. Obwohl der Co-Prozessor nur mit einer Taktfrequenz von 150 MHz betrieben wird, ist das Xilinx-basierte System doppelt so schnell wie Intels Atom-Prozessor mit 1,6 GHz Taktfrequenz.

Beim AES sind die erforderlichen Berechnungen nicht reduzierbar, was gerade Embedded-Systeme schnell an ihre Leistungsgrenzen stoßen lässt. Co-Prozessoren können durch parallele Operationen in der FPGA-Hardware Funktionen des Prozessors übernehmen und diesen deutlich entlasten. Der Entwickler hat dabei freie Hand über das Hardware/Software-Co-Design. In aller Regel wird ein solcher Co-Prozessor über den Systembus angekoppelt, mit dem auch der Prozessor und der Hauptspeicher verbunden sind. Leider ist der Systembus durch Zugriffe auf den Hauptspeicher häufig bereits voll ausgelastet und daher nicht geeignet, weitere Co-Prozessoren zu verbinden.



Vergleich der Rechenzeiten zur Verschlüsselung eines AES-Blocks.

Auxiliary Processor Unit

Eine andere, wesentlich leistungsfähigere Möglichkeit bietet die PowerPC-Architektur, die eine dedizierte Schnittstelle zur Befehlssatzerweiterung, die so genannte Auxiliary Processor Unit (APU) aufweist. Diese wurde ursprünglich von IBM entwickelt, um Gleitkomma-Prozessoren (FPUs) an die Pipeline des PowerPC anzuschließen. Damit stellt die APU eine vom Systembus unabhängige Schnittstelle mit hoher Bandbreite und geringer Latenz dar, die den Systembus nicht zusätzlich belastet. Bei den FPGA-Bausteinen Virtex II-Pro, Virtex4-FX und Virtex5-FXT von Xilinx sind PowerPC-Prozessoren samt APU als Hard-Macro integriert. An die APU lässt sich jetzt ein so genannter Fabric-Co-Processor-Bus (FCB) anschließen, über den der Anschluss von unterschiedlichen Co-Prozessoren, so genannten Fabric-Co-Processor-Modules (FCMs), möglich wird. Diese FCMs werden, wie auch andere Hardware-Blöcke im FPGA, zum Beispiel in VHDL oder in Verilog implementiert. In der Regel läuft der komplexere Co-Prozessor im FPGA mit geringerer Taktfrequenz als der PowerPC. Deshalb kann die APU ganzahlige Teilungsverhältnisse zwischen PowerPC-Takt und Co-Prozessortakt bewältigen. Für die Co-Prozessoren lassen sich neue Maschinencode-Befehle definieren. Die Konfigurationsmöglichkeiten sind vielfältig, neben so genannten User-Defined Instructions (UDIs) sind auch Load-/Store-Operationen möglich. Vor der Benutzung muss jeder Befehl softwaremäßig konfiguriert werden.

Bei der Ausführung dieser Befehle auf dem Prozessor, genauer gesagt in der „Decoding Stage“ der Pipeline, wird dies von der APU erkannt und decodiert, die erforderlichen Operatoren und Signale zur Kommunikation werden an die Co-Prozessor-Logik im FPGA weitergereicht. Je nach Konfiguration wartet der Prozessor auf das Ergebnis, welches in der „Writeback Stage“ der Pipeline zurückgegeben wird – genau wie bei jedem gewöhnlichen Befehl. Auf den ersten Blick scheint es komplizierter als es ist, denn die Xilinx-Toolchain bringt bereits alle erforderlichen Änderungen mit sich, um sofort mit diesen neuen Befehlen zu arbeiten. So wurde der C-Compiler von Xilinx angepasst, um direkt mit vordefinierten Befehlen arbeiten zu können. Mit Hilfe von C-Makros können die erweiterten Befehle direkt aus einem C-Programm eingesetzt werden.

Bei der Suche nach einem beschleunigbaren Algorithmus für den Co-Prozessor sollte man sorgfältig vorgehen. Ziel ist es, aus der Hardware- und aus der Software-Welt zu profitieren. Nicht jede Berechnung und Funktion eignet sich gleich gut, um sie in das FPGA auszulagern. Die Vorzüge des FPGA liegen in seiner hohen Parallelität. Bei sequenziellen Algorithmen mit vielen Verzweigungen oder auch unterschiedlichen Datenbreiten ist die Software häufig im Vorteil. Ein zu beschleunigender Algorithmus sollte daher deutlich von der Nebenläufigkeit im FPGA profitieren. Unsere Erfahrungen zeigen, dass außerdem die Verwendung

von mehreren Befehlen – anders als bei der klassischen Befehlssatzerweiterung mittels eines einzigen mächtigen Befehls – für komplexere Berechnungen besser ist, da sich dadurch größere Datenmengen zum Co-Prozessor und wieder zurück transferieren lassen.

Der AES-128-bit-Standard

Der AES-Standard zur symmetrischen Verschlüsselung hat sich in den letzten Jahren in vielen Bereichen als Standard-Verschlüsselungsmethode durchgesetzt. Insbesondere für die sichere Kommunikation in Netzwerken oder zur transparenten Absicherung von TCP/IP-basierten Anwendungen findet AES Verwendung. Darunter fallen Netzwerk-Protokolle wie Secure Shell (SSH), Transport-Layer-Security (TLS), Secure Sockets Layer (SSL) oder auch WiFi Protected Access 2 (WPA2). Darüber hinaus wird es für Virtual Private Networks (VPNs) oder zur Festplattenverschlüsselung eingesetzt – alles Gebiete, die zunehmend für Embedded-Systeme interessant werden. Das Protokoll AES mit einer Schlüssellänge von 128 bit ist durch seine vielen nebenläufigen Operationen auf Bit-Ebene sehr gut geeignet, um mittels Befehlssatzerweiterung beschleunigt zu werden. In der vorliegenden Implementierung wurden die Verschlüsselung von einem Block, bestehend aus 16 Byte, mit einem Schlüssel von ebenfalls 16 Byte und die Rückgabe der verschlüsselten Daten aus acht UDI-Befehlen aufgebaut (**Listing**).

Der zu Grunde liegende AES-IP-Core ist eine VHDL-Implementierung, die bei OpenCores unter www.opencores.org zu finden ist. Der IP-Core wurde portiert und um die Schnittstelle zur APU erweitert. Das gesamte System samt AES-IP-Core ist auf der MLE 1000 RPS von Missing Link Electronics lauffähig, oder auf anderen Systemen, die einen Xilinx Virtex4-FX oder Virtex5-FXT verwenden. Der Quellcode ist unter www.missinglink-electronics.com als Referenz zur Verfügung gestellt worden.

Beschleunigung um Faktor 12

Zur Performance-Analyse wurde die populäre AES-Implementierung von OpenSSL herangezogen. Zur Laufzeitanalyse wurden übliche Werkzeuge wie „valgrind“ verwendet. Die Ergebnisse der Rechenzeiten zur Verschlüsselung eines AES-Blocks bei dieser Software-Implementierung sind in der **Grafik** dargestellt. Ausgeführt als reines Software-Programm auf dem PowerPC 405 bei einer Taktfrequenz von 300 MHz, benötigt die AES-Verschlüsselung eines Blocks ca. 17,8 µs. Durch Verwendung des Co-Prozessors – und das, obwohl der Co-Prozessor nur mit einer Frequenz von 150 MHz betrieben wird – kann diese Ausführungszeit um den Faktor 12 auf 1,5 µs verringert werden. Zum Vergleich mit anderen leistungsfähigen Prozessoren ist die Ausführungszeit der Blockverschlüsselung auf einem Intel-Atom-Prozessor mit 1,6 GHz aufgetragen. Dieser benötigt zur Berechnung eines Blockes 2,7 µs und damit fast doppelt so lange wie der 300-MHz-PowerPC mit der APU-Crypto-Engine im Xilinx-FPGA. Das zeigt, wieviel Potential die Hardware-/Software-Partitionierung von Embedded-Systemen bietet und dass solche FPGA-basierten System-on-Chip-Plattformen durchaus mit modernen Prozessoren mithalten können – schlichtweg durch die hohe Nebenläufigkeit im FPGA.

Leo Santak, Dr. Endric Schubert/fr

```
//Schlüssel übertragen (128 bit)
UDI1FCM_GPR_GPR_GPR (dummy[0], key[0], key[1]);
UDI1FCM_GPR_GPR_GPR (dummy[1], key[2], key[3]);
//Klartext übertragen 8128 bit)
UDI1FCM_GPR_GPR_GPR (dummy[2], plain[0], plain[1]);
UDI1FCM_GPR_GPR_GPR (dummy[3], plain[2], plain[3]);
//Verschlüsselte Daten abholen (128 bit)
UDI1FCM_GPR_GPR_GPR (enc[0], dummy[0], dummy[1]);
UDI1FCM_GPR_GPR_GPR (enc[1], dummy[0], dummy[1]);
UDI1FCM_GPR_GPR_GPR (enc[2], dummy[0], dummy[1]);
UDI1FCM_GPR_GPR_GPR (enc[3], dummy[0], dummy[1]);
```

I Die Rückgabe der verschlüsselten Daten ist aus acht UDI-Befehlen aufgebaut.