The Elixent D-Fabrix Reconfigurable Algorithm Processor as an Encryption Platform

Tony Stansfield
Elixent Limited
Castlemead, Lower Castle St.
Bristol, United Kingdom
+44 117 917 5770
tony.stansfield@elixent.com

Thomas Foxcroft
Elixent Limited
Castlemead, Lower Castle St.
Bristol, United Kingdom
+44 117 917 5770
thomas.foxcroft@elixent.com

ABSTRACT

This paper describes the benefits of using a high-performance reconfigurable platform (such as Elixents D-Fabrix Reconfigurable Algorithm Processor (RAP)) for encryption applications.

There are two principal benefits from using such a platform: flexibility and scalability. Flexibility means that the same hardware can support multiple different algorithms, and is inherent in the reconfigurable approach. Flexibility is of particular relevance to the encryption context, as it ensures that the hardware platform can be upgraded to future standards, and gives protection against obsolescence if a particular algorithm is found to be insecure.

Scalability of performance allows the size and throughput of an implementation to be tailored to the needs of the target application. In this paper we illustrate the scalability of D-Fabrix for such applications by reference to multiple implementations of both the Advanced Encryption Standard (AES) [1] and the Data Encryption Standard (DES) [2], and explain how the same implementation techniques can be applied to other encryption algorithms.

1. ENCRYPTION OVERVIEW

1.1 Encryption Applications

Encryption technologies are of increasing commercial importance in a range of high-volume applications. They provide a basis for privacy protection in both wired and wireless communications, and also for rights management and protection in media content distribution.

However, there is no single encryption standard used for all applications, nor is there a single performance level (i.e. bits per second of encrypted or decrypted data) that will cover all applications.

As an illustration of the range of performances, consider the following:

- Digital TV channels require 1-10MBit/s
- 802.11b wireless networking runs at up to 11MBit/s, and 11g at up to 54Mbit/s
- VPN over ethernet LAN can run at up to 100Mbit/s

i.e. there is a factor of 100 variation in required throughput across this set of applications (which is not exhaustive – there are application areas with a need for both higher and lower data rates). To create a useful family of encryption products therefore requires creating efficient designs for a wide range of data rates.

Similarly, support for multiple encryption algorithms is important for several reasons:

- Compliance with regional standards, e.g. to adapt a set-top box design to cope with regional variations in subscription TV broadcasts.
- To address situations where standards are in flux, or there is no single agreed standard that covers the whole of an application area. Media content distribution is the prime example of this, with multiple competing proposals at present.
- For situations where a standard itself requires flexibility. e.g. IPsec offers a choice of both encryption and hashing algorithms, with suggestions for extending it with further options.
- In the case of bidirectional communications, the change between encryption and decryption may require a change of algorithm.

1.2 Encryption on RAP

The desire for algorithmic flexibility outlined above makes encryption an interesting application area for reconfigurable technology. This is because although there are many encryption algorithms in common use, in any single application only a small number will be active at the same time. To construct a general-purpose encryption processor using ASIC technology would require dedicated hardware for multiple algorithms, a large part of which would be inactive at any given moment. With a reconfigurable approach, it is only necessary to have hardware sufficient to run

the active algorithms. There is of course an area penalty for the configurability, but this is offset by the removal of the hardware to support inactive algorithms to such an extent that the reconfigurable approach can ultimately provide higher silicon density.

The way in which reconfigurability is used will be different for the various situations outlined in section 1.1, above:

- In the case of geographic variation, an appropriate configuration is chosen on power-up, and loaded into the reconfigurable device. This configuration remains active until the device is switched off again.
- For the situation where multiple standards are in use, the configuration would be changed as required by the data – for instance, changing on a per-vendor or per-track basis when playing encrypted MP3s or other protected media content.
- Where the requirement for flexibility is embedded in the standard, the configuration may be changed in a data-dependent manner. e.g. as a communications channel is opened, or when a data packet arrives.

These different ways of exploiting configurability involve changing the configuration at different rates, and place different constraints on the acceptable configuration time of the reconfigurable device. In the case of a device that loads its configuration once on power-up, a configuration time of several seconds may be acceptable, but if the configuration changes in response to data the configuration time must be small relative to the lifetime of the data and the response time expected by a user to avoid becoming a significant overhead. For communications data this implies reconfiguration times of much less than a millisecond. Announced reconfigurable devices have a range of reconfiguration times, from around a second (for large FPGAs)[3], down to a few microseconds (D-Fabrix)[4], and even a few 10s of nanoseconds in the case of devices targeted at network processing[5].

1.3 Other Benefits of Reconfigurability

Reconfigurability offers other benefits to an encryption processor, beyond the area benefit described above:

- Silicon development time depends on the properties of the reconfigurable component, not on the number of algorithms to be supported
- New algorithms (and algorithm variants) can be easily added
- There is no risk of hardware obsolescence if an algorithm is found to be insecure – the hardware can be reprogrammed to use a new, more secure algorithm.

Reconfigurability therefore provides a useful approach to the desire for flexibility in encryption products referred to above, but does not directly address the need for scalability across a range of performance levels. The remainder of this paper is principally about scalability, and is illustrated with scalable versions of the AES and DES algorithms. Firstly though we have included an outline description of D-Fabrix as relevant background to the discussions of AES and DES that follow.

2. DESCRIPTION OF D-FABRIX

Elixents D-Fabrix Reconfigurable Algorithm Processor (RAP) is a reconfigurable computing platform, available as a scalable, embeddable macro core for System-on-Chip devices that require both high performance and flexibility.

D-Fabrix consists of a regular array of Tiles and Memories. Each tile contains two 4-bit ALUs, six 4-bit registers, and four 4-bit multiplexers. These processing elements are linked together via a configurable routing network to create high-performance application-specific datapaths. A typical array can be reconfigured in a few microseconds. The remainder of this section describes the properties of the individual elements in more detail.

2.1 The ALUs

Each ALU has two 4-bit data inputs (A and B), and generates a 4-bit data output (F).The output is a simple arithmetic or logical function of the inputs, such as add, subtract, bitwise AND, etc.

The ALU has a further input and output (Cin and Cout) that may be used to create a carry chain linking multiple ALUs together – Cout of one ALU driving Cin of the next. This allows the arithmetic functions to operate on words that are greater than 4-bits wide. Cin is also used as the select input for the ALU when used as a multiplexer, and Cout is used as a 1-bit output for comparison and bit reduction operations that process words to produce a 1-bit result.

2.2 The Multiplexers

Multiplexers have two 4-bit data inputs (A and B) and a 4-bit data output (F). A further input (Select) is used to determine which of the inputs is passed to the output. This is directly analogous to the multiplexer function of the ALU referred to above. A dedicated multiplexer is significantly smaller than an ALU, and multiplexers are sufficiently common in applications that including dedicated multiplexers in D-Fabrix results in better overall performance.

2.3 The Registers

Each register has a 4-bit data input and a 4-bit data output. All registers in the array share the same clock. There are no separate Reset or Enable inputs. Such functions are implemented with a register and a multiplexer, and the routing network supports the necessary connections in an efficient manner. All registers can be initialized during configuration.

2.4 The Memories

The memories (commonly referred to as block RAMs) are 256x8-bit SRAMs (i.e. 8-bit address and 8-bit data), with separate data in and data out buses, and a write enable input. Memory access is synchronous (each access takes one clock cycle) and uses the same clock as the registers. Like the registers, all memories can be initialized during configuration.

2.5 The Routing Network

The routing network propagates data between the inputs and outputs of the various elements in the D-Fabrix array. All data is carried across the network as a 4-bit nibble value, with 1-bit values (such as carries and multiplexer selects) being padded with 0s.

3. ADVANCED ENCRYPTION STANDARD

3.1 Description of the Algorithm

AES is a block cipher operating on 128-bit (16 byte) blocks. It is an iterative algorithm, consisting of multiple rounds of the same basic function – between 10 and 14 rounds, depending on key length. The round function contains 4 stages:

- **SubBytes** – Each of the 16 bytes is replaced with a different byte. This can be implemented as a set of byte-addressed table lookups. The same transformation is used for all 16 bytes.
- **ShiftRows** – The 16 bytes are arranged in a 4x4 grid, and the rows are rotated by 0, 1, 2, or 3 bytes. Different rows are rotated different amounts. This operation transforms the relative order of the bytes within a block.
- **MixColumns** – The most complex part of the function, equivalent to a matrix multiplication using finite-field arithmetic. The result is to make each output byte a function of the 4 bytes in the equivalent column of the 4x4 input grid.
- **AddRoundKey** – an XOR of the data with a subset of the encryption key. The subset is different for each round.

A complete *N*-round encryption operation consists of an initial AddRoundKey operation, followed by *N*-1 repeats of the round described above, and a final simplified round that omits the MixColumns stage.

The decryption process has the same structure, with the individual functions replaced with their inverses. The order remains the same, since the properties of the different stages allow some swapping of their order. Further, the inverse functions have the same form as the originals:

- The inverse of SubBytes is a byte substitution.
- The inverse of ShiftRows is a permutation.
- The inverse of MixColumns is a further matrix multiplication.
- The inverse of AddRoundKeys is an XOR with a value derived from the key.

Thus the same principles required for an efficient encryption implementation also apply to decryption. For the remainder of this paper we will therefore concentrate only on the encryption process.

3.2 Implementation on D-Fabrix

3.2.1 Baseline Implementation

In this section we describe an initial implementation of AES, that is then used as the basis for those in following sections.[1]

All the operations in AES manipulate bytes, and therefore map well onto the nibble-based architecture of D-Fabrix. Furthermore, a D-Fabrix array has 256x8 RAMs embedded within it, which are the appropriate size for storing the necessary lookup tables for AES.

SubBytes, ShiftRows, and AddRoundKey all have simple implementations on D-Fabrix: SubBytes is a table lookup using a block RAM, ShiftRows a routing operation, and AddRoundKey an XOR (one of the primitive operations of the ALU).

The only operation with significant complexity is MixColumns. As mentioned above, this is equivalent to a matrix multiplication, so that each output byte is a weighted sum of the bytes in the relevant input column. The weights are constants, so the results of the finite-field multiplication can be pre-computed and stored in a lookup table (which will again fit in a 256x8 block RAM). For the finite field the 'add' operation is equivalent to an XOR, so MixColumns can be implemented as a set of table lookups and XORs. Fortunately, there is only a restricted set of weights used in AES, so there is scope for significant sharing of the RAMs.

Figure 1 is a block diagram of a single round for a highly parallel AES implementation. It uses a large

[1] Note that this implementation is intended to be illustrative, not optimal. It therefore omits some common optimizations, such as the merger of SubBytes and MixColumns lookup tables.

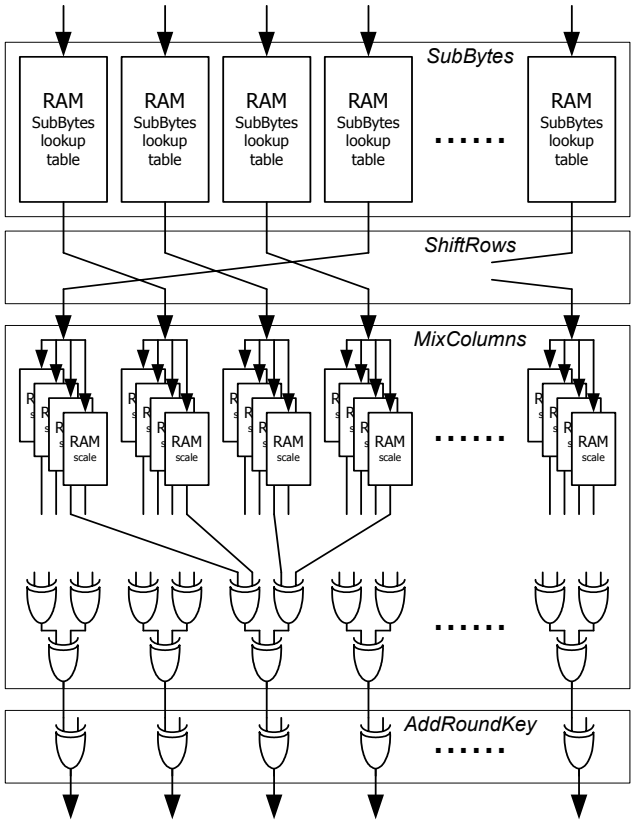

Figure 1: Illustration of Baseline Implementation of Single AES Round

number of RAMs and ALUs (for the XORs), but is capable of very high performance – this single stage can operate in 2 or 3 clock cycles,[2] so repeating it 14 times, with one extra cycle for the initial AddRoundKey stage means that 128 bits will be encrypted in 43 cycles – approximately 3 bits per cycle, or 300Mbits/s at 100MHz.[3]

3.2.2 Sharing of MixColumns

The parallel implementation is significantly faster than required for the applications listed in section 1.1. It is therefore worth considering the options for reducing size and throughput, to provide a more cost-effective version. This can be done by removing some of the repeated copies of blocks of logic, and replacing them with sequential accesses to shared blocks.

The baseline implementation uses 4 copies of the MixColumn circuit, to process all columns in parallel. This can be replaced with sequential use of a single MixColumn stage, as illustrated in figure 2. One columns worth of data is presented at a time to the MixColumn circuit, and the data therefore flows through the circuit in column order. This requires a change of implementation for ShiftRows – the simple wiring network becomes a collection of registers and multiplexers, as the required reordering is now in temporal order, not simply in space. (Figure 2 omits the circuit needed to control the multiplexers in the ShiftRows circuit, but it is not a significant overhead)

This new circuit uses one quarter as many RAMs and XORs, plus some additional multiplexers and registers. It now takes 9 or 10 clock cycles to operate (plus extra cycles to empty the pipeline at the end of processing). 14 iterations will therefore take around 145 cycles, so will process 88Mbits/s at 100MHz.[4] [5]

An intermediate case is also possible, with two copies of the MixColumn circuit processing two columns at a time. This requires a different register and multiplexer network to implement ShiftRows, has a 6-stage pipeline delay, and processes around 150Mbits/s.

3.2.3 Sharing within MixColumns

The implementations described above cover the 80-to-300Mbit/s throughput range, and are therefore appropriate for the wired and wireless (802.11g) LAN applications mentioned in section 1.1. They are significantly faster than required for 802.11b, and for broadcast media though, so it is worth considering the options for further reductions in area and throughput.

There remains a set of duplicated blocks in figure 2 – the RAMs that store scaled versions of the MixColumn inputs. These can also be replaced with sequential access to shared RAMs. This requires a further change in the ShiftRows circuit, increasing the number of registers and multiplexers required. Alternatively, these registers and multiplexers can be replaced with a pair of RAMs (or a single dual-port RAM), used to store the byte stream and then read it back in a different order. With this approach the round

[2] RAM accesses are synchronous in D-Fabrix, and an extra pipeline stage can be inserted in the XOR gates if required.

[3] This data rate assumes that the encryption or decryption of one block must finish before the next can start. In situations where this is not the case (e.g. for Electronic Code Book (ECB) operation, or to encrypt separate streams of data) the unused pipeline cycles can be used to process other blocks. If used in such a way the peak data rate could be 3 times higher – 900Mbits/sec

[4] i.e. ¼ the area, but better than ¼ the throughput. The new circuit uses 4 of the 10 pipeline stages to perform useful processing, while the original only used 1 of 3. The higher activity rate in part compensates for the reduced area.

[5] For ECB operation the unused pipeline cycles could be used to double the data rate – 176Mbits/sec.

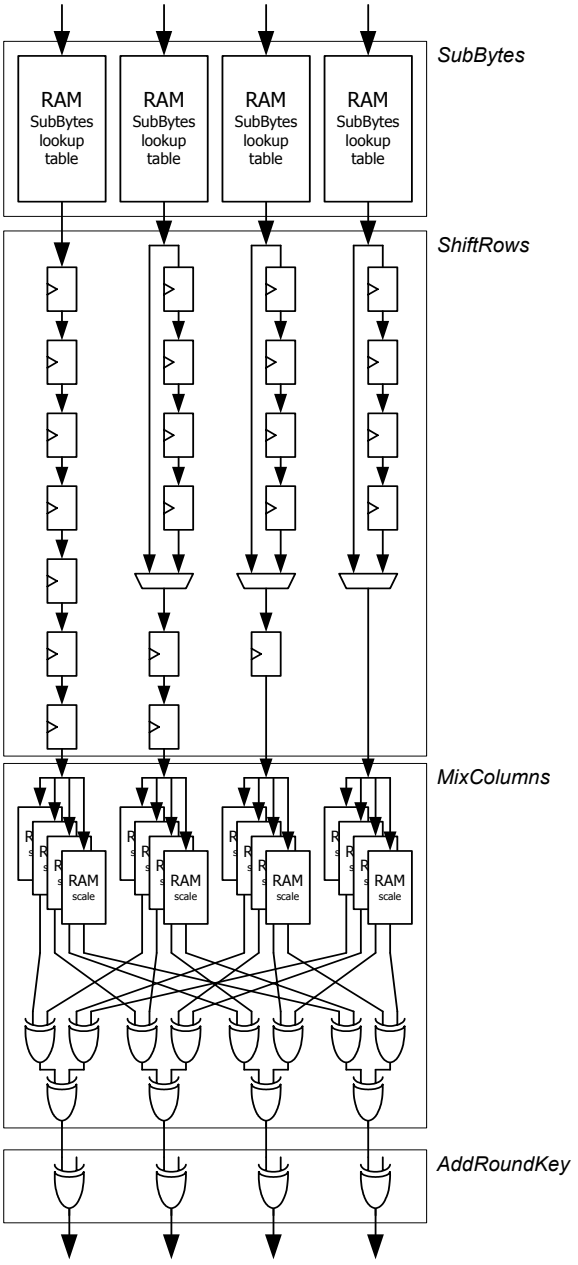

Figure 2 AES Implementation with Shared MixColumns

takes 18 cycles, giving an encryption rate of approximately half a bit per cycle, or 50Mbits/s[6]

The size of all the implementations so far discussed is ultimately limited by the number of RAMs required – D-Fabrix has 1 block RAM per 16 ALUs, whereas figure 2 shows 20 RAMs and 32 nibble-wide XORs.

[6] Throughput reduction is again lower than the area reduction due to the increased pipeline efficiency – 16 of the 18 cycles are processing useful data. There is though no longer any scope for higher throughput in ECB mode.

Even the version with sharing within MixColumns has 7 RAMs and only 26 nibble-wide XORs, each using 1 ALU. Further size reductions can therefore be achieved by replacing some of the RAMs with equivalent logic. This increases the logic depth, (i.e. further pipelining is required), and the throughput is further reduced.

A finite-field multiply-by-constant (to replace the MixColumns RAMs) can be constructed from shifts and XORs, both of which are primitive functions of the ALU. The size and logic depth depend on the constant. In AES, decryption needs larger multipliers than encryption, and adds an extra 8 stages to the pipeline delay so that the data rate falls to 35Mbits/s.

4. DATA ENCRYPTION STANDARD

4.1 Description of the Algorithm

DES is also an iterative block cipher, with 16 identical rounds used to encrypt or decrypt a 64-bit block. The operations used in the round function are as follows:

- XOR of the data with a key
- Bit substitution (or S-box lookup) – overlapping groups of 6 bits in the input are used to select a 4-bit output. 8 such substitutions are used to transform 32 bits of input into a 32-bit result. The 8 substitution tables are different.
- Bit permutation – the order of the input bits is changed
- XOR of one half of the data with the other half

In addition to the round function there are two other bit permutation stages, one applied to the input data before the first round, and the other to the result of the final round.

4.2 Implementation on D-Fabrix

The strategy for creating multiple implementations of DES on D-Fabrix is very similar to that used for AES – create a high-performance, parallel circuit, from which a series of lower performance but smaller versions can be derived.

The basic components of the round function can themselves also be implemented in a similar way to that used for AES:

- The XORs map directly to ALUs.
- The bit substitution is implemented as a table lookup, using a 6-bit address to a block RAM.
- A bit permutation network is created to link the other stages together.

The details of the permutation network are different for DES. In particular, it has to be able to permute bits rather than bytes, and therefore does not map so well to the nibble-based D-Fabrix architecture. D-Fabrix

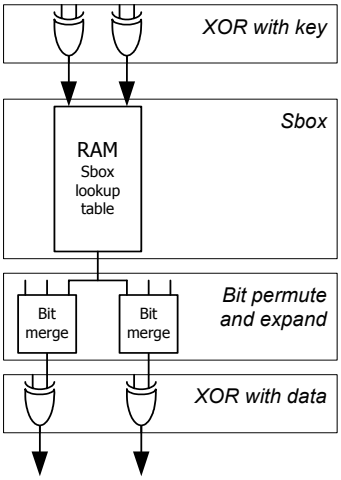

Figure 3: Part of DES implementation showing single Sbox

includes some ALU operations for selecting bits from separate nibbles to merge into the output, but this means that bit permutation is a logic function, not purely a routing function as was byte permutation.

Figure 3 shows the logic associated with a single S-box in a fully parallel implementation. A fully parallel implementation will use 8 copies of this circuit. It can operate with a 2-stage pipeline, so requires 32 cycles to process each 64-bit data block – 2 bits per cycle, or 200Mbits/s at 100MHz.[7]

More compact, but lower throughput, versions can be derived from this implementation in the same way as for AES – by serializing access to shared resources. With DES, there is a complication in that the 8 S-boxes are all different, and so a simple sharing is not possible. However, each individual S-box occupies only one-eighth of a block RAM,[8] so multiple S-boxes can be stored in the same RAM, with the spare address bits used to indicate which S-box is required for each memory access.[9] The ultimate version of this approach is to serialize all S-box accesses, and store all S-Boxes in a single block RAM.[10] This results in a 12-stage pipeline, so requires 192 cycles for 16 iterations. This is equivalent to 1/3 bit per cycle, or 33Mbits/s at 100MHz.

5. OTHER ALGORITHMS

Iterative block cipher algorithms are relatively common, and often have a round function that combines permutation, substitution and/or arithmetic. There are good theoretical reasons for this, it is not simply a result of arbitrary decisions by the designers of these algorithms. The implementation process used above for AES and DES is then generally applicable to any such algorithm:

- Define a parallel implementation of the round function, and:
- Transform it into a lower data rate version by sharing blocks, and replacing the permutation stage with a circuit that can permute temporal order as well as position.

Such general applicability provides a degree of confidence that it is possible to create scalable and flexible encryption platforms that can handle a wide variety of algorithms – both known and unknown. As mentioned in section 1.3 this is a useful feature, since encryption is always vulnerable to the discovery of a weakness in any given algorithm.

6. CONCLUSIONS

In this paper we have described the benefits of using a reconfigurable algorithm processor for encryption, and a simple way of creating applications with a range of throughputs. This therefore meets our original aim of having an encryption platform that is both flexible (achieved with reconfigurability) and has scalable performance.

7. REFERENCES

[1] Federal Information Processing Standard (FIPS PUB) 197. "Advanced Encryption Standard (AES)" see e.g. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] Federal Information Processing Standard (FIPS PUB) 46-2. "Data Encryption Standard (DES)" see e.g. http://www.itl.nist.gov/fipspubs/fip46-2.htm

[3] Xilinx Inc. Virtex II datasheet, for download from: http://direct.xilinx.com/bvdocs/publications/ds031.pdf Module 2 table 25 and Module 3 tables 31 – 33.

[4] T.Stansfield & T.Foxcroft "*JPEG on the Elixent D-Fabrix Reconfigurable Algorithm Processor*" ISPC, April 2003.

[5] M.Motomura "*A Dynamically Reconfigurable Processor Architecture*" Microprocessor Forum, Oct.2002

---

[7] Using the spare pipeline cycle could give 400Mbits/sec ECB operation.

[8] Each S-box only requires 6 of the available 8 address bits, and 4 of the available 8 data output bits.

[9] The spare address bits can be set via the key stream, and inserted into the address by the XOR shown at the top of figure 3. This approach works to insert 2 bits into the stream –i.e. to select one of 4 S-boxes.

[10] This requires an extra multiplexer at the RAM output to choose which of the 2 nibble outputs corresponds to the active S-box.