Application exploration regarding a DPC like architecture

Application Exploration Regarding a DPC Like Architecture

Rolf Enzler, Thomas Sailer
Swiss Federal Institute of Technology (ETH)
Electronics Laboratory
CH-8092 Zürich, Switzerland
{enzler,sailer}@ife.ee.ethz.ch

Technical Report

May 2000

Abstract

This report explores applications in the fields of multimedia, cryptography and telecommunication with regard to the Dynamically Programmable Cache (DPC) architecture proposed by Nakkar [38]. A set of applications representing future systems' requirements are investigated in order to obtain an application base, which allows deriving a suitable DPC like architecture. The considered evaluation criteria are operations, granularity, parallelism and data access patterns. Our investigations show that a DPC like architecture has the potential to significantly speed-up the targeted application classes. We believe, however, that the proposed DPC architecture is too restrictive to fully exploit its potential, and that it is therefore worth to investigate several basic systems parameters more in depth.

1 Introduction

With reconfigurable computing a new computation paradigm has emerged in the last decade, which intents to fill the gap between conventional microprocessors and application-specific integrated circuits (ASICs) [22, 29, 52]. All reconfigurable systems share the same basic idea: to benefit from reprogrammable logic, which allows to dynamically adapt the system's functionality to the requirements of the running application. The most popular devices, which actually enabled reconfigurable computing, are field-programmable gate arrays (FPGAs), which were introduced in the mid eighties [23].

Many approaches of reconfigurable systems have been proposed in the recent years. Amongst them are so called hybrids, which combine reconfigurable hardware with a processor core. One of these approaches is the Dynamically Programmable Cache (DPC) architecture [37, 38], which intends to integrate reconfigurable resources into a processor's data cache.

These report summarises our effort of exploring applications suited for a DPC like architecture. We have concentrated on the three application areas

- multimedia,
- cryptography, and
- telecommunication,

which will play an important role in future systems like e. g. handhelds, as we believe one of the most interesting target fields for reconfigurable systems in the near future.

The evaluation criteria that underlie our investigations are

- operations,
- granularity (bit width),
- parallelism, and
- data access patterns (memory, cache, register).

The goal of the application exploration was to gain insight of the potential that a DPC like architecture offers for the considered application fields. Furthermore, the investigations should serve as a base for future improvements of the DPC architecture.

Sections 2, 3 and 4 describe and analyse the most important building blocks in the fields of multimedia, cryptography and telecommunication, respectively. Section 5 summarises the results and outlines the gained insights.

2 Multimedia

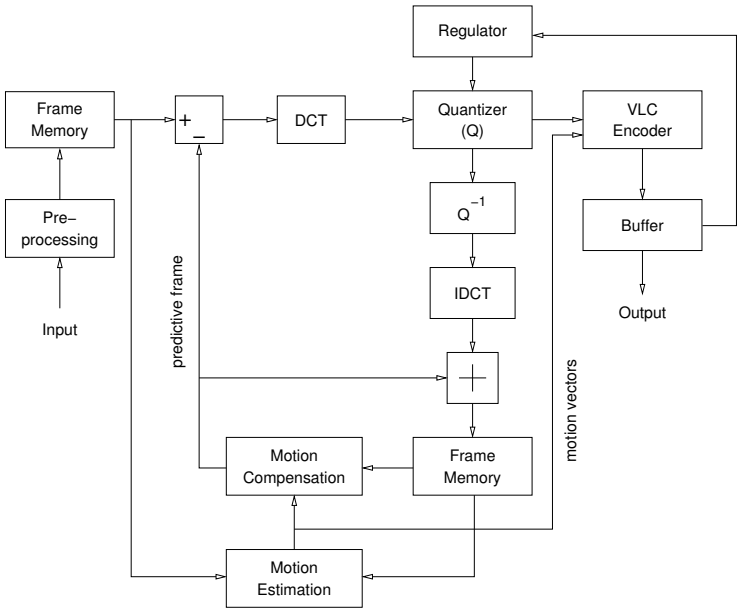
This section investigates algorithms that are widely used in multimedia applications, like video and audio compression. An important representative is MPEG [6, 9, 24, 36]. Figure 1 shows the block diagrams of an MPEG video encoder and decoder, respectively.

2.1 Motion Estimation

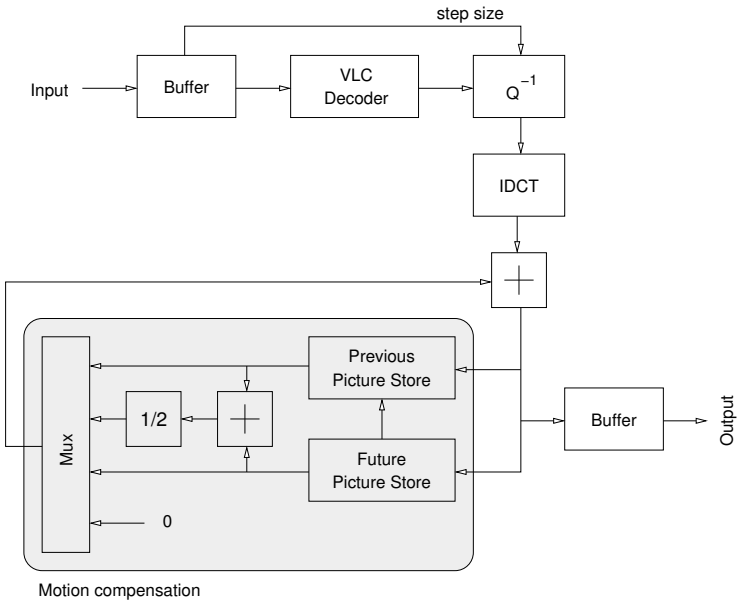
Typically, two consecutive frames of a video sequence are very similar. This observation is the basis of motion-compensated coding, where a frame is coded based on its difference from another frame. In practice, frames are divided into blocks (called *macroblocks*). For a block in the current frame (called *reference* or *source* block), motion estimation is the process of finding a block in another frame that best matches its characteristics (according to a given criterion). The *motion vector* identifies the position of the best block relative to the reference block. The search for the best matching block is done in a rectangular area (called *search window*) around the relative position of the reference block (Fig. 2). In most video applications, the reference block is 16×16 pixels, and the search window is 31×31 pixels. For broadcast TV, good performance is obtained at $p = 15$ for head-and-shoulders-type video scenes, and at $p = 63$ for sporting events [6].

The common matching criteria are the mean absolute error (MAE) or the mean square error (MSE). Under the MAE criterion, for a displacement vector (i, j), the distortion between two macroblocks is defined as

$$D(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |r_{m,n} - s_{m+i,n+j}| \ , \qquad i,j \in [-p, p-1] \ , \tag{1}$$

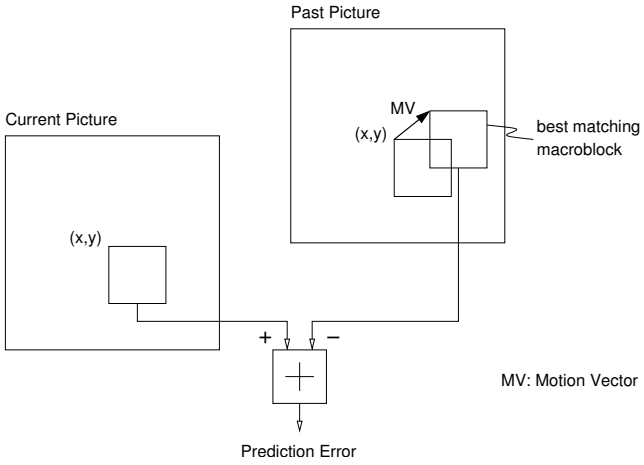


(a)

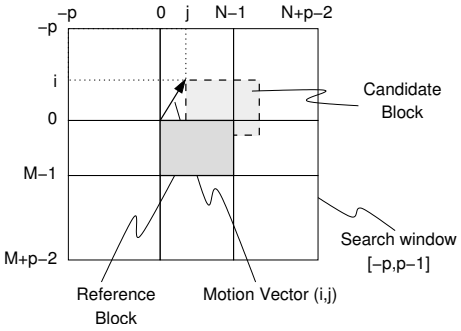


(b)

Figure 1: Block diagram of an MPEG video encoder (a) and decoder (b).



(a)



(b)

Figure 2: Motion estimation (a) with search window of range [-p,p-1] (b).

where r is the reference block and s is a candidate block. The motion vector is the displacement vector (i, j) which minimises $D(i, j)$. For each macroblock of the reference picture a motion vector has to be calculated.

Figures 3 and 4 show the dataflow on different levels according (1).

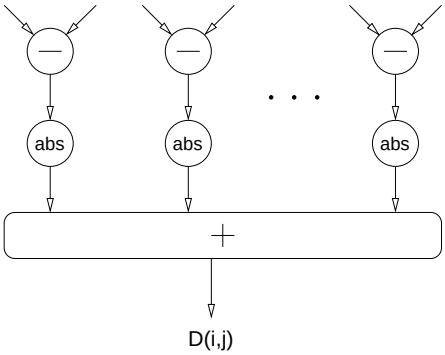


Figure 3: Distortion computation between a reference block and a candidate block (inner loop)

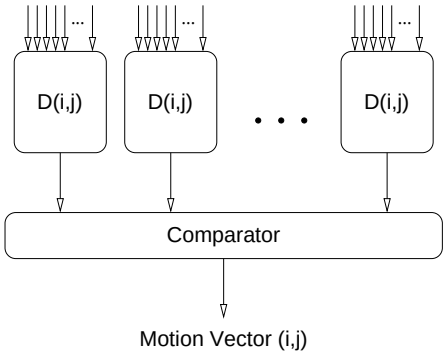


Figure 4: Motion vector computation (outer loop)

Operations

- Subtraction (8-bit operands)
- Absolute-value computation (8-bit operands); i. e. some sort of control structure is needed.
- Addition (8-bit to 16-bit operands)
- Comparison (16-bit operands)

Granularity

- Input (pixel data): 8 bit
- Distortions: 16 bit
- Motion vectors: 2×16 bit

Parallelism

There exists parallelism on different levels of the algorithm:

0. The basic block is the computation of the difference between two pels (pixel elements), i. e. a "sub-abs-add" operation.
1. The inner loop is the computation of the distortion $D(i, j)$ between the reference block and a candidate block (Fig. 3).
2. The outer loop is the computation of all distortions $D(i, j)$ between a reference block and its according candidate blocks merged with the comparison of the distortions (Fig. 4).
3. The ultimate loop is the computation of the motion vectors for all macroblocks of the reference picture.

Data Access

On the different levels the following accesses occur:

0. A pel of the reference block and the according pel of the candidate block are needed.
1. All pels of the reference block and all pels of the candidate block are needed. In subsequent executions of the inner loop nearly the same pels of the candidate block are used again. Therefore, the computation is sensitive to the data accesses (order of pels in the memory and cache parameters).
2. The pels of the reference block stay the same. All pels of the search window are needed. In subsequent computations of the outer loop many pels of the search window are the same. Therefore, the computation is sensitive to the data accesses.
3. All pels of the reference and the current frame are needed.

Enhancements

- Sub-pel motion estimation is possible. The additional required operations in the basic block are additions and right shifts.
- If instead of the MAE criterion the MSE criterion is used, the absolute-value operation in the basic block is substituted by the squaring operation.
- To reduce the computational requirements, 1-bit motion estimation seems to be valuable [6]. In that case, the operations for the MAE criterion are reduced to the EXOR of a bit sequence and summing up the number of 1's in the result.

2.2 DCT/IDCT

DCT-based image coding is the basis for all the image and video compression standards. The basic computation in a DCT-based system is the transformation of an 8×8 image block from the spatial domain to the DCT domain. An important property of the 2D DCT and IDCT transforms is separability, i. e. the 8×8 2D DCT can be obtained by first performing eight 1D DCTs on the rows followed by a matrix transposition and eight 1D DCTs on the columns. An 8-point 1D DCT is given by

$$y_i = \sum_{k=1}^{8} c_{ik} x_k \ , \qquad i = 1, 2, \ldots, 8 \ , \tag{2}$$

where y_i denotes the output elements, x_k denotes the input data, and c_{ik} denotes the DCT coefficients [6].

Operations

There exist several DCT algorithms [5, 6, 28, 47], which highly depend on the target architecture. Therefore, the operations vary between the different algorithms.

- Multiplication (maybe bit-serial)

- Addition (maybe bit-serial)
- Subtraction (maybe bit-serial)
- Shift
- Maybe 4-product MAC (using conventional arithmetic or serial distributed arithmetic [47])
- Maybe look-up table (LUT)
- Matrix transposition

Granularity

From an ASIC implementation example [47]:

- Input: 8 bit
- Internal: 12 bit
- DCT coefficients: 10 bit
- Output: 12 bit

Parallelism

- MAC operation
- 8-point 1D DCT (e. g. four adders, four subtracters, eight 4-product MACs)
- 8×8 2D DCT is composed of (i) eight 8-point 1D DCTs, (ii) matrix transposition, (iii) eight 8-point 1D DCTs.

Data Access

- The eight coefficients stay always the same and are continuously used.
- For an 8-point 1D DCT eight data values are needed.
- For the 2D DCT all data values of the 8×8 matrix are needed.

2.3 Entropy Coding (RLC, VLC)

Run-length coding (RLC) and variable-length coding (VLC) are both lossless compression techniques.

The *RLC encoder* compresses an input stream by representing consecutive zeros by their run-length. The RLC decoder reverses the process by generating the appropriate number of zeros between two nonzero data. In hardware, an RLC coder can easily be implemented using a counter, registers, and some logic [6]. In a software implementation, the most common operations are compare and accumulates [37].

Operations

- 1-bit compare
- Counter

Granularity

- 1 bit

Parallelism

- None

Data Access

- Forward data flow

The *VLC encoder* maps input data of fixed length into codewords of variable length, concatenates them together, and segments them into 16-bit words. Compression is achieved by assigning short codewords to input symbols of high probability and long codewords to input symbols of low probability. The mapping process can be performed either through table-look ups or bit-serially by tracing a Huffman encoding tree. The Huffman coding tables are designed based on the statistics of the input source [6].

VLC decoding is much harder, because codewords have variable length, and the receiver has no prior knowledge of the boundaries between two consecutive codewords.

Operations

- Bit-wise merging of output data
- Maybe barrel shifter
- Maybe 4-bit adder
- Maybe MUX, i. e. control structures
- Maybe shift/compares

Granularity

- Input: 8 - 16 bit
- Output: 16 bit

Parallelism

- None

Data Access

- VLC tables (codeword table, code-length table)
- Data is continuously processed.

2.4 Audio compression

This subsection investigates MPEG audio encoding/decoding as described in [6]. Figure 5 shows the block diagram of an MPEG audio encoder.

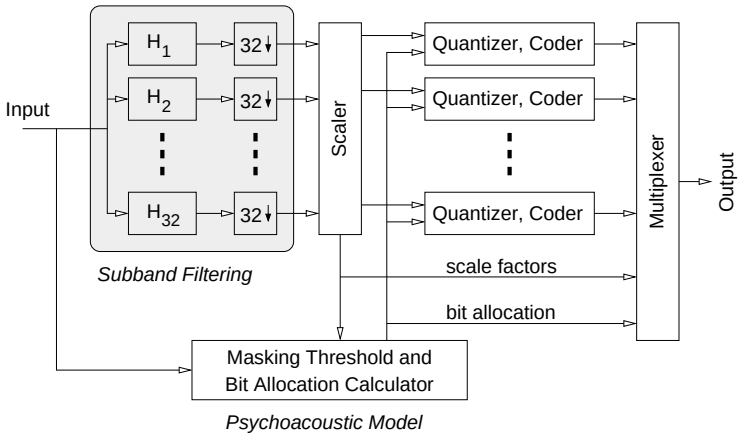


Figure 5: Block diagram of an MPEG audio encoder.

Operations

- Subband analysis/synthesis filtering:
  - Windowing (multiplications with coefficients)
  - Partial summations (additions)
  - Matrixing (MACs)
  - Maybe DCT
- Psychoacoustic model:
  - 1024-point FFT

Granularity

- Audio samples: 16 - 20 bit
- Internal: up to 24 bit, or floating-point

Parallelism

- Windowing
- Partial summations
- Matrixing

Data Access

- Coefficients
- Matrixing values
- Data: 512 input audio samples for 32 output subband samples (encoding)

2.5 Video and Audio Compression – Summary

Table 1 shows an overview of generic operations in *video* processing. Based on the content of Table 1 and an analysis of the processing pipeline in video compression, [6] concludes the following:

- Input data and coefficients have usually 8- to 16-bit precision.
- There is no need for floating-point operations.
- The multiply-accumulate (MAC) operation is very common. However, most of the multiplications are with constants.
- Saturation arithmetic, where a result is clipped (operation *clip*()) to the maximum or minimum value of a predefined range, is common in many operations (such as colour transformation).

Function	Operations
Colour transformation, preprocessing, and post-processing	$\sum c_i x_i$, $clip()$, $\frac{1}{2}(x_i + x_j)$, $\frac{1}{4}\sum_{i=1}^{4} x_i$
DCT, IDCT	$ax + b$, $\sum c_i x_i$
Quantisation	$\frac{x_i}{c_i}$
Dequantisation	$x_i c_i$
Huffman coding (VLC)	data shifts, comparisons
Motion estimation/compensation	$\sum \lvert x_i - y_i \rvert$ or $\sum (x_i - y_i)^2$, $min(a, b)$, $x_i + cx_j$

Table 1: Generic operations in video compression [6]

Table 2 shows the most common arithmetic operations in Layer II MPEG *audio* decoding. [6] outlines that similar analysis of other audio coding algorithms show that multiplication and addition are the most common operations and concludes that general-purpose DSPs are ideally suited for audio processing. Since audio samples have 16-bit precision, encoders should use either floating-point or 24-bit precision.

Function	Operations
Degrouping	$y = c \bmod a$, $c = \frac{c}{d}$
Dequantisation	$y = (x + a)b$
Denormalisation	$y = ax$
Matrixing	$y = ax + b$, $y = \sum_i x_i c_i$
Windowing	$y = xa$, $y = \sum_i w_i$

Table 2: Arithmetic operations in MPEG audio decoding [6]

3 Cryptography

3.1 DES (Data Encryption Standard)

The DES algorithm possesses an iterative structure. Data is passed through the Feistel network 16 times, each time with a different subkey from the key

transformation (Fig. 6 and 7) [42].

Operations

- Permutations (32-, 48-bit)
- Expansion (32→48-bit)
- EXORs (32-, 48-bit)
- Combinational logic (S-boxes, 6→4-bit)
- Left-rotations (28-bit) for key transformation

Granularity

- Operands: 4, 6, 28, 48, 56, 64 bit
- Permutations and expansion require access to single bits.

Parallelism

- The DES computation sequence for encoding a 64-bit data value does not have much inherent parallelism, unless the computation of the eight S-boxes. What is anyway interesting for a DPC like architecture is the mix of operations that are needed (permutations, EXOR, rotations, etc.), that do not map well onto a general-purpose processor.

Data Access

- The key is needed during the whole encryption process. However, during the 16 rounds of the encoding process for one data value the key is continuously left-shifted.
- The encoded data value is processed straightforward.
- If tables (e. g. the S-boxes) are stored in the memory instead of being realized in combinational logic, they are accessed very frequently.

3.2 RSA

RSA (named after its inventors Rivest, Shamir and Adleman) gets its security from the difficulty of factoring large numbers. The public and private keys are functions of a pair of large prime numbers (100 to 200 digits or even larger). Recovering the plaintext from the public key and the ciphertext is conjectured to be equivalent to factoring the product of the two numbers. The RSA algorithm is a de facto standard in much of the world. RSA is used in PGP (Pretty Good Privacy) for key management and digital signatures (with keys up to 2047 bits) [42].

Encryption is performed by computing

$$C = M^e \pmod{n} \ , \tag{3}$$

where M is the plaintext such that $0 \leq M < n$. The number C is the ciphertext from which the plaintext M can be computed by

$$M = C^d \pmod{n} \ , \tag{4}$$

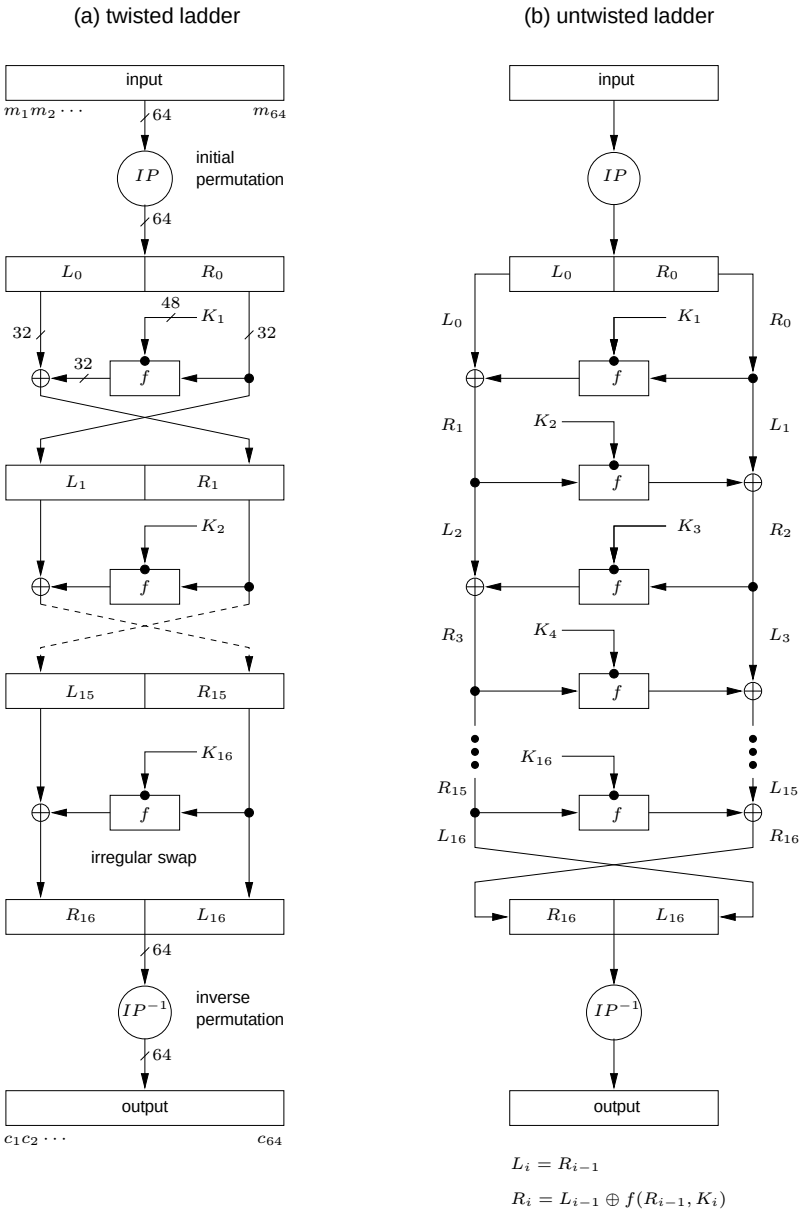


Figure 6: DES computation structure

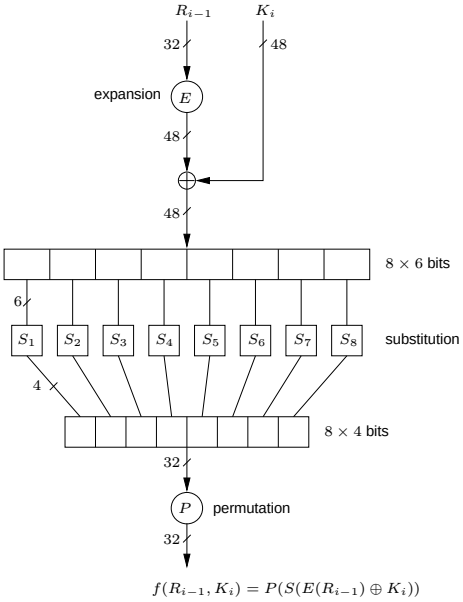


Figure 7: DES inner f function

where e and n are the public key, and d is the private decryption key (see Table 3).

Public Key:	
n	product of the two primes p and q
e	relatively prime to $(p-1)(q-1)$
Private Key:	
d	$e^{-1} \bmod ((p-1)(q-1))$

Table 3: RSA keys

Operations

- Modular exponentiation
  - Bit scanning of the exponent e
  - Modular multiplication
    - MAC
    - Addition
    - Shift
    - Maybe EXOR and AND
    - Maybe subtraction
    - Some sort of control
  - Modular squaring

    - Addition
    - Shift
    - Maybe EXOR and AND
    - Maybe subtraction
    - Some sort of control
- Modular inverse (Euclidean algorithm), just once

However, several algorithms have been proposed for the modular exponentiation and modular multiplication operations [26, 27, 42].

Granularity

- p, q: arbitrary, but large (see modulus n)
- n: 512 – 2048 bit
- e: rather "small", usually not more than 32 bit (3, 17, and $2^{16} + 1$ are common choices)
- d: large, approximately like n
- Granularity of the operations is arbitrary; e. g. word-width of the processor.

Parallelism

- Inherent parallelism of the operations
- Multiplication and squaring operations (in the RL binary method)

Data Access

- n, e or d, respectively, are needed throughout the computation.
- Data is processed straightforward.

3.3 IDEA (International Data Encryption Algorithm)

IDEA is a block-cipher that operates on 64-bit plaintext blocks [11, 42, 57, 58]. The key is 128 bits long. The same algorithm is used for both encryption and decryption (Fig. 8). IDEA is used in PGP for data encryption.

Operations

- Bit-wise EXOR (16-bit)
- Addition modulo 2^{16}
- Multiplication modulo $2^{16} + 1$ [57]
- Shift

Granularity

- All operations work on 16-bit sub-blocks
- 16-bit subkeys

Parallelism

- Some parallelism: 2 to 4 operations (see Fig. 8), but partly sequential execution

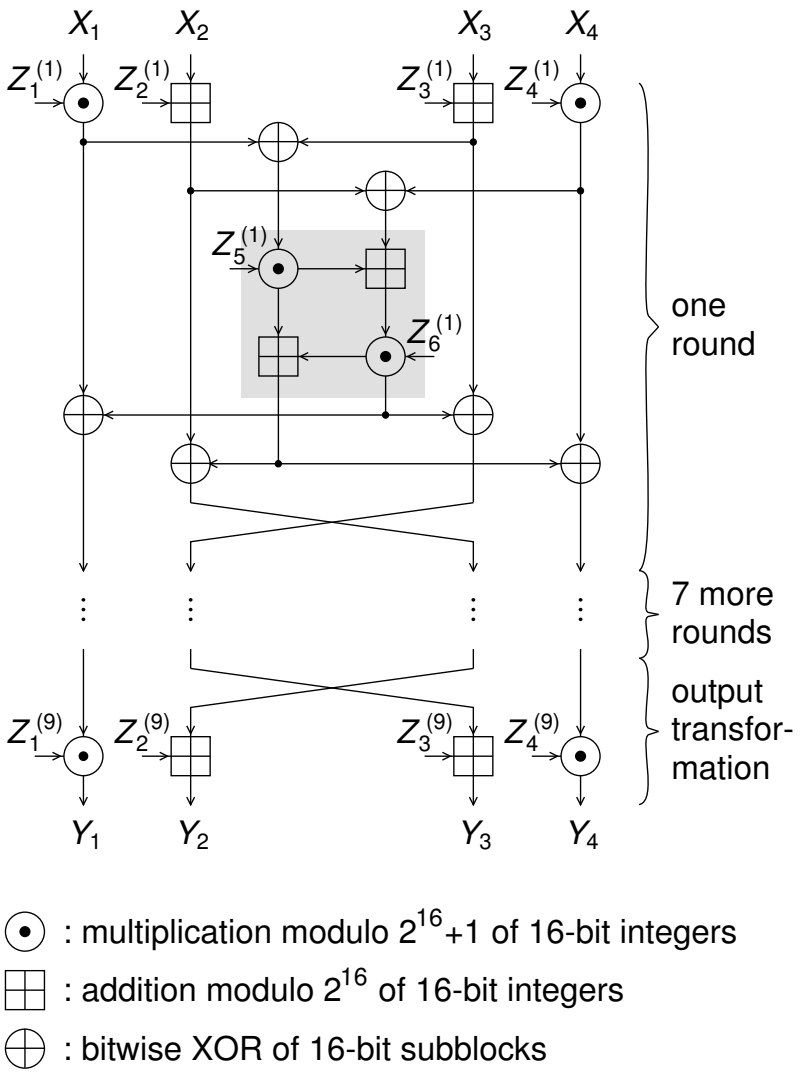


Figure 8: IDEA computation structure

Data Access

- 128-bit key
- 8 identical (but sequential) rounds
- Data is continuously processed.

3.4 Twofish

Twofish is one of the five final candidates[1] for the new Advanced Encryption Standard (AES)[2], which is planned to be completed in summer 2001.

Twofish is a 128-bit block cipher that accepts a variable-length key up to 256 bits [39, 43]. The cipher is a 16-round Feistel network with a bijective F function made up of four key-dependent 8-by-8-bit S-boxes, a fixed 4-by-4 maximum distance separable matrix over GF(2^8), a pseudo-Hadamard transform (PHT), bit-wise rotations, and a carefully designed key schedule (Fig. 9 and 10).

Operations

- EXOR (32, 8, 4 bit)
- Left rotation (1, 8, 9 bit)
- Right rotation (1 bit) on 32-, 4-bit values
- Addition modulo 2^{32}
- Multiplication in $GF(2^8)$
- 8-bit permutation
- Swap

Granularity

- 128, 64, 32, 8, 4 bit
- Defined key length: 128, 192 or 256 bit

Parallelism

- 16 sequential rounds
- A single F function consists of two g and two h functions in parallel.
- Permutations q_0, q_1 (maybe combinational logic)
- Input and output whitening

Data Access

- 4 8×8-bit S-boxes (key-dependent),
- 4×4 MDS matrix
- 0kB, 1kB or 4kB table for S-boxes and MDS matrix (depending on keying option)
- Key (derived are 40 subkeys), 160 bytes

[1] The four other candidates are MARS, RC6, Rijndael and Serpent.

[2] An effort by the U.S. National Institute of Standards and Technology (NIST). For further information see http://www.nist.gov/aes/.

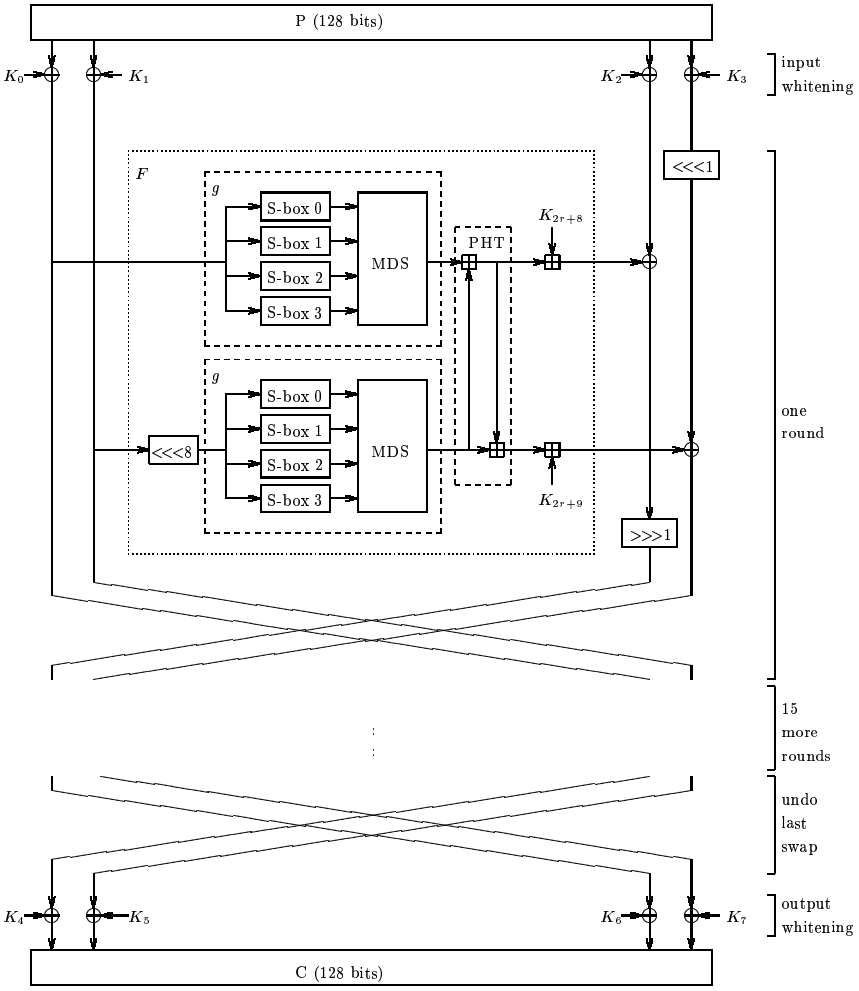


Figure 9: Twofish computation structure

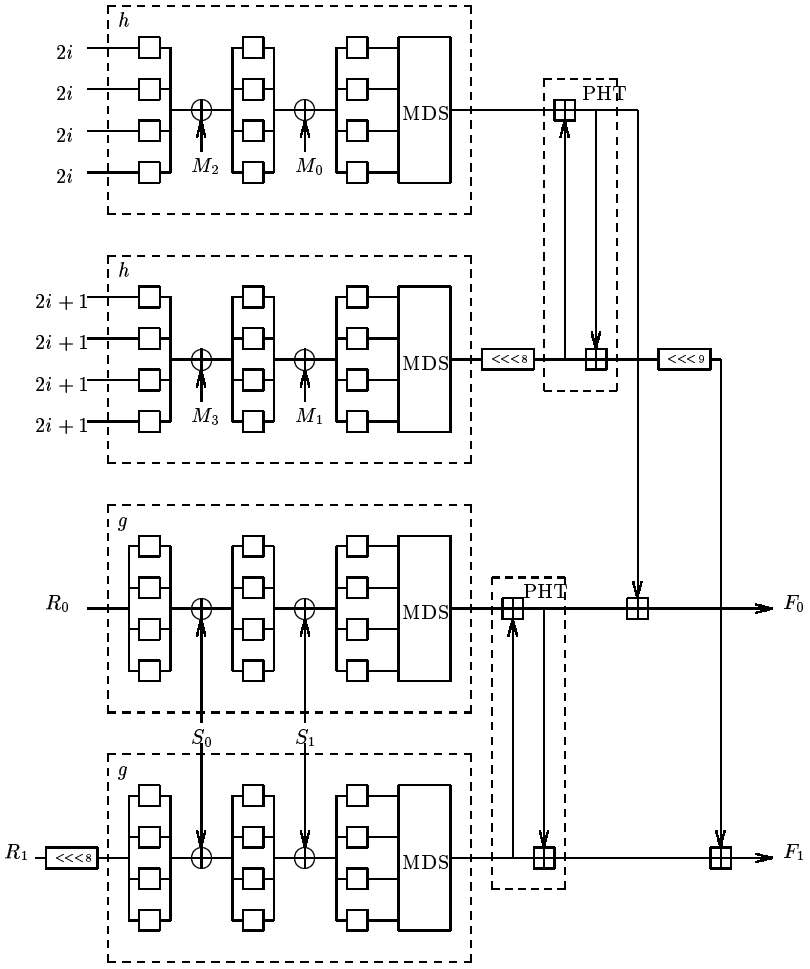


Figure 10: Twofish inner F function (128-bit key)

3.5 Elliptic Curve Cryptosystems

At a high level, elliptic curve cryptosystems are analogs of existing public-key cryptosystems in which modular arithmetic is replaced by operations defined over elliptic curves [2, 3, 8, 30, 31, 32, 33, 46]. In the literature, elliptic curve cryptosystems have appeared that are analogs either to RSA or to discrete logarithm based systems.

Just as in all public-key cryptosystems, the security of elliptic curve cryptosystems relies on the underlying hard mathematical problems. It turns out that elliptic curve analogs of RSA are mainly of academic interest and offer no practical advantage over ordinary RSA, since their security is based on the same underlying problem as RSA, namely integer factorisation. The situation is quite different with elliptic curve variants of discrete logarithm based systems. The security of such systems depends on the following hard problem: Given two points G and Y on an elliptic curve such that Y = kG, i. e. Y is G added to itself k times, find the integer k. This problem is commonly referred to as the "elliptic curve discrete logarithm problem."

Currently, it appears that elliptic curve cryptosystems with a 160-bit key offer the same security as RSA and discrete logarithm based systems with a 1024-bit key. The smaller key sizes result in smaller system parameters and hence in improved performance and eased memory requirements. Therefore, elliptic curve cryptosystems are especially useful in applications with limited memory, bandwidth, or computational power.

Establishing the system parameters of a cryptosystem involves selecting an underlying finite field and a representation for the elements in the finite field. Then an elliptic curve has to be chosen together with a point on the curve called the generator.

The addition operation in an elliptic curve is the counterpart to modular multiplication in common public-key cryptosystems (e. g. RSA), and multiple addition is the counterpart to modular exponentiation. However, the elliptic curve operations are quite complicated, more complicated in fact than the operations required for RSA.

Table 4 shows an example of an elliptic curve in the finite field $\mathbb{F}_{2^m}$ with the according addition formula.

For the following considerations we assume that the field $\mathbb{F}_{2^m}$ is represented in terms of a *normal basis*, i. e. a basis over $\mathbb{F}_2$ of the form

$$\left\{\theta, \theta^2, \theta^{2^2}, \ldots, \theta^{2^{m-1}}\right\} \quad . \tag{5}$$

Operations

- Elliptic addition/subtraction[3]: $Q = O + P$
  - Field inversion (several field multiplications)
  - Field multiplication (rather complex operation)
  - Field addition (EXOR)
  - Field squaring (one-bit cyclic shift)
- Elliptic doubling: $Q = 2P$

[3] to subtract the point $P = (x, y)$ one adds the point $-P = (x, x + y)$

Finite field	$\mathbb{F}_{2^m}$
Elliptic curve E	$y^2 + xy = x^3 + ax^2 + b, \quad b \neq 0$
Addition formula	$P = (x_0, y_0)$, $Q = (x_1, y_1)$, $P + Q = (x_2, y_2)$
	if $P \neq Q$ (elliptic addition): $\lambda = \left(\frac{y_0 + y_1}{x_0 + x_1}\right)$ $x_2 = \lambda^2 + \lambda + x_0 + x_1 + a$ $y_2 = (x_1 + x_2)\,\lambda + x_2 + y_1$
	if $P = Q$ (elliptic doubling): $\lambda = \left(x_1 + \frac{y_1}{x_1}\right)$ $x_2 = \lambda^2 + \lambda + a$ $y_2 = x_1^2 + (\lambda + 1)\,x_2$
	some special cases have to be considered.

Table 4: Standard elliptic curve over $\mathbb{F}_{2^m}$ with the according addition formula. Actually, there exists a faster algorithm [33], but it does not fit well with hardware implementations of normal bases [46].

  - Field inversion (several field multiplications)
  - Field multiplication (rather complex operation)
  - Field addition (EXOR)
  - Field squaring (one-bit cyclic shift)
- Elliptic scalar multiplication: $Q = nP$ (*addition-subtraction method*)
  - Nonadjacent form (NAF) of the coefficient n (bit operations, control)
  - Elliptic doubling
  - Elliptic addition, subtraction

Granularity

- Operands: ≥ 160 bit

Parallelism

- Inherent parallelism of the operations
- Implementation depends heavily on the chosen parameters (field, basis, elliptic curve).

4 Telecommunication

Software Defined Radio is a hot topic for third generation (3G) mobile systems [7, 10, 34, 35, 48, 49, 50, 51, 53, 54]. On one hand, the industry seems unable

to agree on a single standard, on the other hand, service providers want to customise the air interface to their needs.

In this section, we identify the most computationally demanding building blocks for receivers for wireless communication systems. Transmitters are not considered; they usually are much less computationally demanding, and often their building blocks are the same or very similar to those of the receiver (e. g. IFFT/FFT in OFDM transmitters/receivers).

There are two modulation schemes that are popular in emerging digital wireless communication schemes:

- OFDM (Orthogonal Frequency Division Multiplex)
- DSSS (Direct Sequence Spread Spectrum)

The former is used in broadcast type applications (e. g. DAB[4], DVB-T[5]) and very high speed indoor wireless data networks with limited coverage (e. g. WAND[6]). The latter is used extensively in cellular voice and data networks (e. g. IS-95[7] [17], UMTS UTRA[8] [1, 12, 13, 40, 41]).

4.1 FFT

The Fast Fourier Transform is by far the computationally most expensive task for OFDM demodulation. Table 5 shows the FFT sizes of important air interfaces [16, 20].

Standard	FFT Size
WAND	16
DAB	2048
DVB-T	2048 or 8192

Table 5: FFT sizes for various wireless standards

The basic radix-2 FFT algorithms perform $\frac{n}{2}\log2(n)$ "butterfly" calculations. Each butterfly consists of two complex adds and one complex multiplication. Higher radix FFT algorithms exist, their advantage is lower memory bandwidth.

Synchronisation can be achieved either by periodically transmitting a sequence of synchronisation symbols, or by transmitting "pilot carriers" and hypothesis testing.

The former method is used when synchronisation has to be achieved fast, such as in high speed data networks. Synchronisation can be detected using a matched filter to the sync symbols (i. e. convolution).

The latter method is used when synchronisation time does not matter, but overhead due to synchronisation should be kept low. Synchronisation works by running the demodulator as if it was synchronised for some time. After that, it

[4] Digital Audio Broadcast
[5] Terrestrial Digital Video Broadcast
[6] Wireless ATM Network Demonstrator
[7] also known as N-CDMA
[8] Universal Mobile Telecommunications System UMTS Terrestrial Radio Access, also known as 3G mobile telecommunications, IMT-2000

decides if it really is synchronised and if not, advances one time-step and repeats that procedure.

Operations

- $n \log2(n)$ complex additions
- $\frac{n}{2} \log2(n)$ complex multiplications

Granularity

- 6 – 16 bit

Parallelism

- There are $\frac{n}{2}$ butterflies per stage; in the extreme, they can be executed in parallel.

Data Access

- Input/output data array (the computation is usually performed in place)
- LUT for the twiddle factors ($e^{i2\pi \frac{k}{N}}$)

4.2 Despreading

Despreading is done in DSSS receivers to reduce the large signal bandwidth to approximately the user data rate. The key problem here is to achieve synchronisation as quickly as possible.

Parameter	IS-95	UMTS UTRA	GPS (C/A)
Chip rate	1.2288M	4.096M (opt. 8.192M or 16.384M)	1.023M
Spreading code	M-Sequence	Gold (opt. Kasami)	Gold
Channelisation code	Walsh (H_{64})	OVSF	not applicable
Typ. # of Despreaders (RAKE fingers)	4	4–5	
Synch code length	2^{15}	2^8	$2^{10} - 1$
Inner code	Convolutional, K=9, rate $\frac{1}{2}$	Convolutional, K=9, rate $\frac{1}{3}$	not applicable
Interleaver	48–384	$\surd$	not applicable
Outer code	none	Reed Solomon	not applicable

Table 6: DSSS parameters of important air interfaces

One basic despreading unit (one RAKE finger) computes the following operation:

$$d_k = \sum_{m=0}^{n-1} c_{kn+m} \times s_{kn+m} \ , \tag{6}$$

where d_k denotes the despreaded samples, s_k the chip samples, and c_k the scrambling code samples. c_k can be ± 1.

Each despreader usually calculates (6) twice with chip samples taken at slightly different times (e. g. $\frac{1}{4}$ chips apart) to obtain time tracking information.

The scrambling code c_k may be the combination of a scrambling code and a channelisation code [14]. The following algorithms are used to generate the scrambling and channelisation codes:

- M-Sequences
- Gold codes
- Kasami codes
- Walsh codes
- Orthogonal Variable Spreading Factor (OVSF) codes

M-Sequences can be generated using a linear feedback shift register (LFSR). It consists of flip-flops and EXOR gates. Both Gold codes and Kasami codes are families of codes constructed by exoring two LFSRs. Walsh codes are the rows of a Hadamard matrix and can be constructed recursively: $H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and $H_{2N} = \begin{pmatrix} H_N & H_N \\ H_N & -H_N \end{pmatrix}$. OVSF is a variation on the Walsh theme.

In the GPS case, there is usually one despreader per satellite tracked, and a couple of despreaders to find new satellites. In the cellular terminal case, about three despreaders receive and combine the three strongest propagation components, and one despreader looks for new propagation components and new base stations.

During initial synchronisation search, worst case about four times the synchronisation code length hypotheses need to be tested. Testing one such hypothesis means running one RAKE finger for some time and then compare its output against a threshold. The number of samples to be processed before making the decision determines the false alarm/miss probabilities. So it is desirable to have a high number of RAKE fingers to speed up the synchronisation process.

Operations

- Add/subtract
- Shift registers/EXOR gates

Granularity

- PN code generation: 1 bit
- Input samples: ≈ 2 – 8 bit

Parallelism

- RAKE fingers are independent.
- One RAKE finger could be split into multiple parts that could be executed in parallel. There is some additional complexity in determining where to start the PN generator for each part.

Data Access

- Input sample memory

4.3 Interleaver

Interleaving is used to spread burst errors over a long interval to make them more tractable for the error correcting codes.

The interleaver is often regular; it writes data to a matrix row-wise and reads the data back column-wise.

Operations

- Memory writes/reads
- Address generators; since one matrix dimension is often a power of 2, address generators can usually simply be counters with some permutation of their output lines.

Granularity

- 1 – 8 bit

4.4 Convolutional Codes

Convolutional codes are often used just after the demodulator, because they can easily use soft decisions.

There are two main algorithms for decoding convolutional codes, namely the Viterbi Algorithm and the Sequential Decoding Algorithm.

The latter follows the most promising path first. If the metric increase starts to fall short of the expectation, it backs off and tries other paths. This algorithm can deal with long constraint length codes easily, but the main disadvantage is that its execution time is not predictable. It is therefore very seldom used in practice.

The Viterbi Algorithm (VA) on the other hand is very popular. It has constant runtime independent on the received symbols. Its complexity (number of nodes) is 2^{k-1}, where k is the constraint length.

Two paths emanate from each node at time n to two nodes at time $n+1$. For each of those paths, the metric has to be updated, i. e. a value has to be added to the metric accumulator. The value is a function of the received symbol and the expected symbol along the path. At each node at time $n+1$, two paths merge. The VA has to select the one with the higher metric, and it has to record this decision. The primitive operation of the VA is therefore the update of two metric accumulators and the selection of the one with the higher number. This operation is called "Add Compare Select" (ACS). Many DSPs have instruction

set extensions containing an ACS primitive. The VA executes an ACS operation for each receiver symbol for each of the 2^{k-1} nodes.

The output of the VA is the ACS decisions along the surviving path. The surviving path is the path ending at the toor node (node 0) at the end of the message.

In continuous mode, every few received symbols the highest metric node is determined, and the decisions along its path up to about 5 constraint lengths in the past are output.

For every path, the VA needs to somehow store the past decisions. One possibility is to keep a shift register containing the past decisions at every node. Another possibility is to store a "back pointer" to a node at time k at every note at time $k+1$.

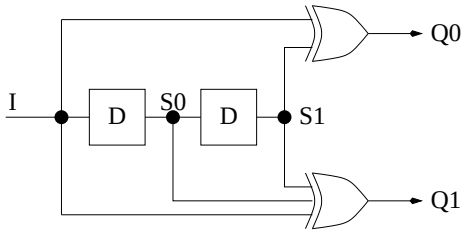


Figure 11: Convolutional Code Encoder

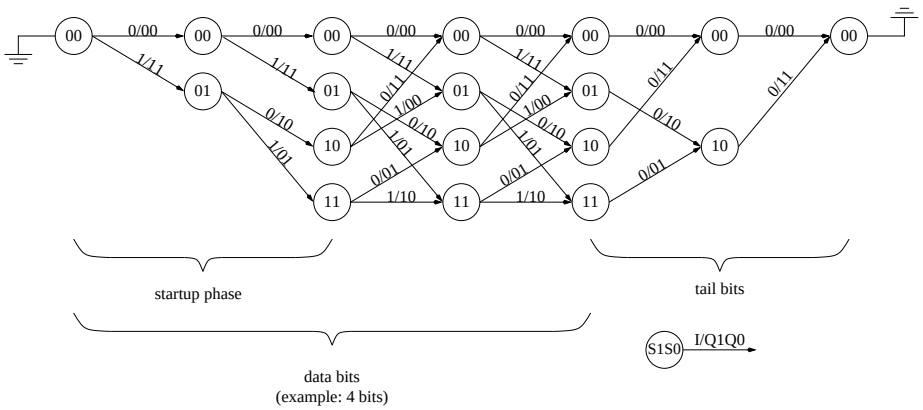


Figure 12: Trellis Diagram

Figure 11 shows an example Convolutional Code Encoder. It is a rate $R = \frac{1}{2}$ constraint length $K = 3$ code with x^2+1 and x^2+x+1 as generating polynomials. Figure 12 shows the corresponding Trellis diagram. From the Trellis diagram one can see that the smallest possible detour from the all zero path has 1 nonzero information bit and 5 nonzero channel bits along its way, this is a measure for the error correcting capabilities of the code.

Operations

- "AddCompareSelect"
- Path backtracking

Granularity

- Path metric for every symbol: up to about 4 bit
- Path metric accumulator: 8 – 16 bit

Parallelism

- Nodes at time k are independent and can be executed in parallel.

Data Access

- Metric accumulators
- Path storage
- Input symbol storage

4.5 Block Codes

To decode block codes, the received symbols (bits, as block codes are almost always fed with hard decision data) are written as a vector. This received signal vector is then multiplied with a matrix to produce the syndrome vector. If the syndrome vector is the zero vector, the received codeword was error free. If not, a mapping procedure is carried out to map the syndrome vector to the most likely error vector. The error vector is then subtracted from the received codeword vector to obtain the most likely correct codeword.

The mapping procedure may be carried out algorithmically (e. g. Berlekamp-Massey for Reed Solomon codes) or using a look-up table (e. g. Meggitt Decoder for BCH codes).

In the Reed Solomon case, the syndrome vector can be computed by an FFT. The syndrome vector is then a subvector of the FFT output vector.

Note however that Block codes use arithmetic structures other than the ring of the natural numbers $\langle \mathbb{Z}, +, \cdot \rangle$ or the ring of the natural numbers modulo m $\langle \mathbb{Z}_m, \oplus, \odot \rangle$. A field is required, and a popular choice is the Galois Field $GF(2^8)$.

Operations

- Additions in GF(q)
- Multiplications in GF(q)

Granularity

- $\approx$ 4 – 8 bit

Parallelism

- Matrix vector multiplication may be executed row-wise.

Data Access

- Input vector
- Parity check matrix

4.6 Fractional Sample Delays

Fractional sample delays are required to synchronise the receiver precisely to the transmitter clock.

One solution for this problem is to have a FIR (lowpass) filter with a set of m coefficient arrays spaced at $1/m$ samples, and then choosing the coefficient array that produces the nearest sample.

Another solution is to have multiple FIR filters and then sum their outputs, each filter output weighted with a different power of μ, the fractional sample delay. This is called "Farrow Structure".

Operations

- Convolution (multiplication and addition)
- Polynomial (multiplication and addition)

Granularity

- 4 – 12 bit

Parallelism

- Farrow structure: every constituent convolver is independent.

Data Access

- Past samples
- Filter coefficients

4.7 Finite Field Arithmetic

Finite Field Arithmetic is used heavily in cryptography and forward error correcting (FEC) codes. By far the most important finite fields are the GF(2^m) extension fields of GF(2). GF($2^{\leq 8}$) are the most used Galois fields in practice.

Given a primitive polynomial $p(x)$ in GF(2^m), and α a root of $p(x)$, i. e. $p(\alpha) = 0$, numbers in GF(2^m) (i. e. members of the set) may be represented in different ways [25, 44].

$$GF(2^m) = \{0, \alpha^0, \alpha^1, \cdots, \alpha^{2^m-2}\} \tag{7}$$

Equation (7) shows the *exponential form* or *power representation*. Multiplication, division and inversion is simple using this representation (addition, subtraction and negation modulo $2^m - 1$, since $\alpha^{2^m-1} = 1$), but addition is not possible. 0 needs to be special cased. Conversions to other bases (see below) usually require a "log table" and an "antilog table", which can be quite big (255×8 bits for GF(2^8), each).

$$GF(2^m) = \{A | A = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \cdots + a_1\alpha + a_0, \quad \text{where} \quad a_i \in GF(2), 0 \leq i \leq m-1\} \tag{8}$$

$$\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\} \tag{9}$$

Since the primitive polynomial $p(x)$ is monic, of degree m, and $p(\alpha) = 0$, $\alpha^m = \sum_{i=0}^{m-1} a_i \alpha^i$. Therefore, every element of GF(2^m) can be represented by a polynomial with a degree less than m and coefficients in GF(2) (8). Equation (9) is called the *standard base representation* (SBR). Addition using the SBR is easy (digit-wise using an EXOR gate, no "carry chain"), but multiplication is more tedious. Formally, it is a multiplication of two polynomials modulo $p(x)$, i. e. $c(x) = a(x)b(x) \mod p(x)$. In principle, it is similar to integer multiplication, but requires an additional modulo reduction step.

$$\{1, \alpha^2, \alpha^4, \alpha^8, \cdots, \alpha^{2^{m-1}}\} \tag{10}$$

Another base used is the *normal base representation* (NBR), Equation (10). Squaring using the NBR is easy (cyclic left shift, since squaring in GF(2^m) is a linear operation, $(a \oplus b)^2 = a^2 \oplus a \otimes b \oplus a \otimes b \oplus b^2 = a^2 \oplus b^2$), but multiplication is more tedious than using the SBR. Some proposed multipliers use two representations and switch back and forth between the two representations, but that is only feasible if α can be chosen suitably. In the general case, base conversion is not worth the effort.

Useful operation primitives are $D = A \otimes B \oplus C$, which can be used to compose matrix multiplications, and $D = A \otimes B^2 \oplus C$, which can be used for exponentiations, divisions and inversions.

To decompose these operations ($D = A \otimes B \oplus C$ and $D = A \otimes B^2 \oplus C$) further, $a(\alpha)\alpha \mod p(\alpha)$, $a(\alpha)\alpha^2 \mod p(\alpha)$ and $a(\alpha) + b_i c(\alpha)$ need to be computed. The latter is simply a combination of an AND and an XOR gate for every digit. $a(\alpha)\alpha$ and $a(\alpha)\alpha^2$ is simply a left by one or two digits, respectively. $\mod p(\alpha)$ is also a combination of AND and XOR gates.

Various regular structures can be found in literature [25, 44, 55, 56] for computing multiplications, divisions and exponentiations in GF(2^m) in hardware.

[15] proposes a combined 17×17 bit integer multiplier/GF($2^{\leq 8}$) multiplier, where the GF(2^m) part does not significantly slow down nor increase in size the integer part.

As an example we consider arithmetic operations the Galois field GF(2^4) using standard basis representation and the primitive polynomial $p(x) = x^4 + x + 1$. Figure 13 shows the addition circuitry. Figure 15 shows the $\alpha a(\alpha) \mod p(\alpha)$ building block, and Fig. 14 the conditional sum building block. These two building blocks may be combined to form a circuit that calculates $S = A \otimes B \oplus C$. Note that A is fed into the circuit in parallel while B is fed serially. If the $\alpha a(\alpha) \mod p(\alpha)$ blocks are replaced with $\alpha^2 a(\alpha) \mod p(\alpha)$ blocks, the circuit calculates $S = A \otimes B^2 \oplus C$.

Operations

- $D = A \otimes B \oplus C$ and $D = A \otimes B^2 \oplus C$ in GF($2^{\leq 8}$)
- AND, XOR gates

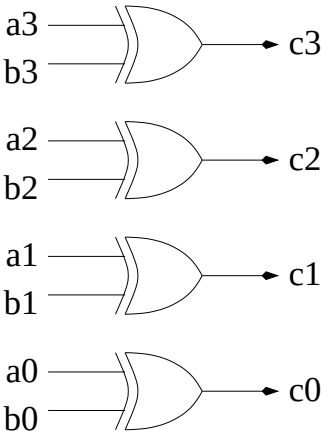


Figure 13: Addition in GF(2^4) using standard base representation

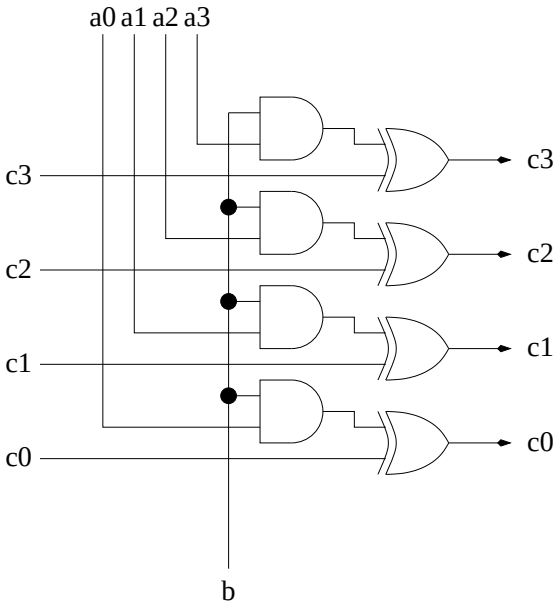


Figure 14: Conditional Sum circuitry in GF(2^4) using standard base representation

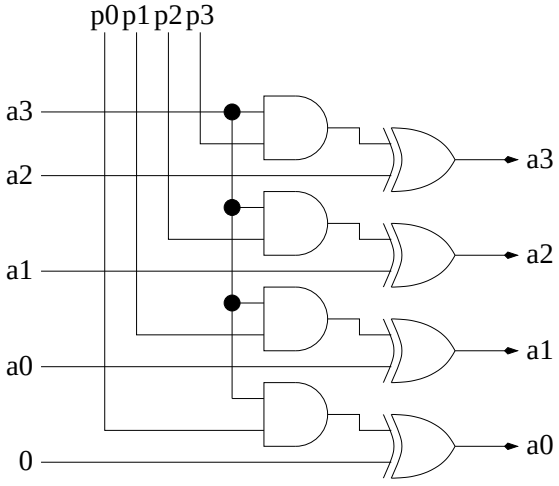


Figure 15: Multiplication by α in GF(2^4) using standard base representation

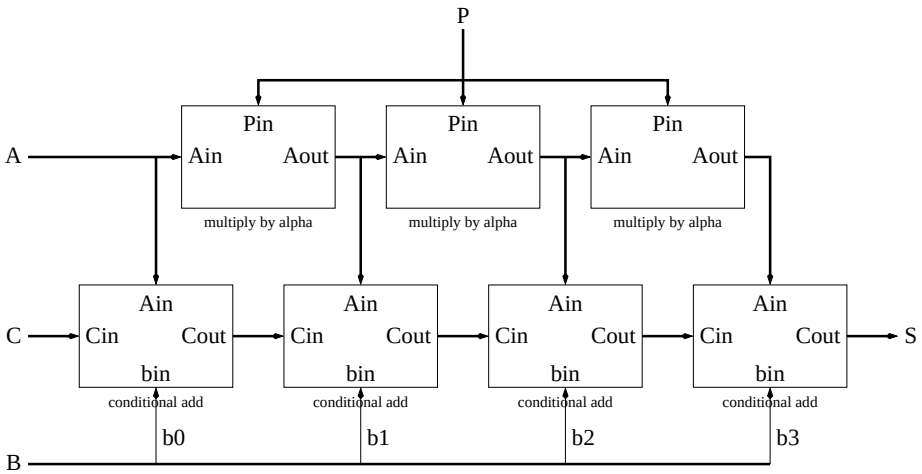


Figure 16: Multiply-Add in GF(2^4) using standard base representation

Granularity

- Up to 8 bit

5 Summary and Conclusions

Based on our application exploration we derive the following conclusions:

Operations Besides the common addition, subtraction and multiplication operations especially EXOR, shift, rotation, MAC, permutations, and operations in Galois fields are widely used. Many of these operations are not well supported by general-purpose processors.

Granularity The granularity, i. e. the operands' bit widths, varies between the different algorithms. Surprisingly, less algorithms than we expected require 1-bit operations, especially if we do not count e. g. EXORs as 1-bit operations (EXORs need access to single bits of the operands, but work mostly on words, e. g. on 8-bit data). Therefore, a 1-bit computation architecture does not seem mandatory. Multi-bit computation units seem to be an interesting trade-off (e. g. 4-bit or 8-bit).
With the connection structure between the computation units, the situation is different. If we consider multi-bit computation units, it would be the most natural just to connect the units with buses of the same bitwidth. We state that especially shift and permutation operations do not map well on such an architecture. Therefore, we believe that a connection scheme that allows an arbitrary bit access between the computation units is better suited for the targeted applications. From this observation we conclude furthermore that architectures like for example MorphoSys [21, 45], which is composed of an 8-bit ALU array, do not suite many of our targeted applications. However, since the requirements of the applications are quite diverse, the architecture will always be a trade-off and

therefore different approaches are conceivable.

Parallelism The applications show inherent parallelism on various levels, fine grain (operation level) and course grain (function level). That observation lets us conclude that there is potential to significantly speed-up the targeted application classes. To exploit parallelism, often a sort of control structure is required.

Data access Data access seems to be one of the most critical parts of the architecture. Some of the investigated algorithms access the data values quite irregularly (e. g. FFT). We are therefore concerned that an architecture like DPC, whose concept targets a forward direction computation flow like it is typical for systolic arrays, could significantly suffer.

Recapitulating, we believe that there is potential to significantly speed-up the targeted application classes. The proposed DPC architecture seems to be too restrictive, we believe that some very basic system parameters have to be investigated more deeply:

- Composition of the DPC unit
  - Logic block structure
  - Interconnect
  - Memory interface (between logic blocks and memory lines)
  - System interface (to the processor core; memory hierarchy)
- Location of the DPC unit in the system's memory hierarchy
  - Cache (as proposed by the DPC architecture)
  - On-chip memory (as many DSP have it)
  - Off-chip memory
  - Co-processor
- Embedding the DPC unit into the system

References

[1] Fumiyuki Adachi, Mamoru Sawahashi, and Hirohito Suda. Wideband DS-CDMA for next-generation mobile communications systems. *IEEE Communications Magazine*, 36(9):56ff, September 1998.

[2] G. B. Agnew, T. Beth, R. C. Mullin, and S. A. Vanstone. Arithmetic operations in GF(2^m). *Journal of Cryptology*, 6(1):3–13, 1993.

[3] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone. An implementation for a fast public-key cryptosystem. *Journal of Cryptology*, 3(2):63–79, 1991.

[4] Arinc Research Corporation. *Navstar GPS Space Segment/Navigation User Interfaces*, 1993.

[5] N. W. Bergmann, Y. Y. Chung, and B. K. Gunther. Efficient implementation of the DCT on custom computers. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'97)*, pages 244–245, 1997.

[6] V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures*. Kluwer Academic Publishers, 2nd edition, 1997.

[7] David B. Chester. Digital IF filter technology for 3G systems: An introduction. *IEEE Communications Magazine*, 37(2):102ff, February 1999.

[8] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology - ASIACRYPT'98, Proceedings*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 1998.

[9] R. Cook, J. Jean, and J.-S. Chen. Accelerating an MPEG-2 encoder utilizing reconfigurable computing. In *CERC/VIUF/IEEE Computer Society Workshop on 21st Century Electronic Systems Design: Breakthroughs in Quality and Productivity*, 1997.

[10] Mark Cummings and Shinichiro Haruyama. FPGA in the software radio. *IEEE Communications Magazine*, 37(2):108ff, February 1999.

[11] A. Curiger, H. Bonnenberg, R. Zimmermann, N. Felber, H. Kaeslin, and W. Fichtner. VINCI: VLSI implementation of the new secret-key block cipher IDEA. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'93)*, pages 15.5.1–15.5.4, 1993.

[12] E. Dahlman, P. Beming, J. Knutsson, F. Ovesjö, M. Persson, and C. Roobol. WCDMA—the radio interface for future mobile multimedia communications. *IEEE Transactions on Vehicular Technology*, 47(4):1105ff, November 1998.

[13] Erik Dahlman, Björn Gudmundson, Mats Nilsson, and Johan Sköld. UMTS/IMT-2000 based on wideband CDMA. *IEEE Communications Magazine*, 36(9):70ff, September 1998.

[14] Esmael H. Dinan and Bijan Jabbari. Spreading codes for direct sequence CDMA and wideband CDMA cellular network. *IEEE Communications Magazine*, 36(9):48ff, September 1998.

[15] Wolfram Drescher and Gerhard Fettweis. VLSI architectures for multiplication in gf(2^m) for application tailored digital signal processing. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, 1997.

[16] EBU/CENELEC/ETSI JTC. *ETS 300 401: Radio broadcasting systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*. European Telecommunication Standards Institute (ETSI), 2nd edition, May 1997.

[17] Electronics Industry Association (EIA)/Telephone Industry Association (TIA). *Interim Standard 95*, 1992.

[18] European Telecommunication Standards Institute (ETSI). *ETS 300 421: Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services*, v1.1.2 edition, August 1997.

[19] European Telecommunication Standards Institute (ETSI). *ETS 300 429: Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for cable systems*, v1.2.1 edition, April 1997.

[20] European Telecommunication Standards Institute (ETSI). *ETS 300 744: Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, v1.1.2 edition, August 1997.

[21] Guangming Lu, H. Singh, Ming-hau Lee, N. Bagherzadeh, F. Kurdahi, and E. M. C. Filho. The MorphoSys parallel reconfigurable system. In *Euro-Par'99, Parallel Processing*, volume 1685 of *Lecture Notes in Computer Science*. Springer, 1999.

[22] S. Hauck. The future of reconfigurable systems. In *Canadian Conference on Field Programmable Devices (FPD'98)*, 1998.

[23] S. Hauck. The roles of FPGA's in reprogrammable systems. *Proceedings of the IEEE*, 86(4):615–638, April 1998.

[24] E. Iwata and K. Olukotun. Exploiting coarse-grain parallelism in the MPEG-2 algorithm. Technical Report CSL-TR-98-771, Stanford University, Computer Systems Lab, September 1998.

[25] Surendra K. Jain, Leilei Song, and Keshab K. Parhi. Efficient semisystolic architectures for finite-field arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(1):101–113, March 1998.

[26] Ç. K. Koç. High-speed RSA implementation. Technical Report 201, Version 2.0, RSA Laboratories, November 1994.

[27] Ç. K. Koç. RSA hardware implementation. Technical Report 801, Version 1.0, RSA Laboratories, August 1995.

[28] D. Lau, A. Schneider, M. D. Ercegovac, and J. Villasenor. FPGA-based structures for on-line FFT and DCT. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'99)*, pages 266–267, 1999.

[29] W. H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V. K. Prasanna, and H. A. E. Spaanenburg. Seeking solutions in configurable computing. *IEEE Computer*, 30(12):38–43, December 1997.

[30] A. Menezes. Elliptic curve cryptosystems. *CryptoBytes*, 1(2):1–4, 1995.

[31] A. Menezes and S. Vanstone. The implementation of elliptic curve cryptosystems. In *Advances in Cryptology – AUSCRYPT'90*, volume 453 of *Lecture Notes in Computer Science*, pages 2–13, 1990.

[32] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[33] A. J. Menezes and S. A. Vanstone. Elliptic curve cryptosystems and their implementation. *Journal of Cryptology*, 6(4):209–224, 1993.

[34] J. Mitola III. Software radio architecture: A mathematical perspective. *IEEE Journal on Selected Areas in Communications*, 17(4):514ff, April 1999.

[35] Joseph Mitola III. Technical challenges in the globalization of software radio. *IEEE Communications Magazine*, 37(2):84ff, February 1999.

[36] Moving Picture Experts Group (MPEG). `http://www.cselt.it/mpeg/`.

[37] M. Nakkar. *Evaluation of Dynamically Programmable Cache Machine with Low Power Field Programmable Gate Arrays (FPGAs) and 3-Dimensional Multi-Chip-Module Package*. PhD thesis, North Carolina State University, Department of Electrical and Computer Engineering, 1999.

[38] M. Nakkar, J. Harding, D. Schwartz, P. Franzon, and T. Conte. Dynamically programmable cache. In *Configurable Computing: Technology and Applications*, volume 3526 of *Proceedings of SPIE*, pages 218–226. SPIE, 1998.

[39] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, and E. Roback. Status report on the first round of the development of the Advanced Encryption Standard. Technical report, National Institute of Standards and Technology (NIST), August 1999.

[40] Tero Ojanperä and Ramjee Prasad. An overview of air interface multiple access for IMT-2000/UMTS. *IEEE Communications Magazine*, 36(9):82ff, September 1998.

[41] A. Samukic. UMTS universal mobile telecommunications system: Development of standards for the third generation. *IEEE Transactions on Vehicular Technology*, 47(4):1099ff, November 1998.

[42] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 2nd edition, 1996.

[43] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall. Twofish: A 128-bit block cipher. Technical report, Counterpane Systems, June 1998. `http://www.counterpane.com/twofish.html`.

[44] Shyue-Win Wei. VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in GF(2^m). *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, 44(10):847–855, October 1997.

[45] H. Singh, Ming-hau Lee, Guangming Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho. MorphoSys: A reconfigurable architecture for multimedia applications. In *Proceedings of the PACT'98 Workshop on Reconfigurable Computing*, pages 34–39, 1998.

[46] J. A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 1997.

[47] D. W. Trainor, J. P. Heron, and R. F. Woods. Implementation of the 2D DCT using a Xilinx XC6264 FPGA. In *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS'97)*, pages 541–550, 1997.

[48] Hiroshi Tsurumi and Yasuo Suzuki. Broadband RF stage architecture for software defined radio in handheld terminal applications. *IEEE Communications Magazine*, 37(2):90ff, February 1999.

[49] Thierry Turletti and David Tennenhouse. Complexity of a software GSM base station. *IEEE Communications Magazine*, 37(2):113ff, February 1999.

[50] Walter H. W. Tuttlebee. Software defined radio: Facets of a developing technology. *IEEE Personal Communications*, 6(2):38ff, April 1999.

[51] Walter H. W. Tuttlebee. Software radio technology: A european perspective. *IEEE Communications Magazine*, 37(2):118ff, February 1999.

[52] J. Villasenor and W. H. Mangione-Smith. Configurable computing. *Scientific American*, pages 66–71, June 1997.

[53] R. H. Walden. Analog-to-digital converter survey and analysis. *IEEE Journal on Selected Areas in Communications*, 17(4):539ff, April 1999.

[54] Robert H. Walden. Performance trends for analog-to-digital converters. *IEEE Communications Magazine*, 37(2):96ff, February 1999.

[55] Charles C. Wang, T. K. Truong, Howard M. Shao, Leslie J. Deutsch, Jim K. Omura, and Irving S. Reed. VLSI architectures for computing multiplications and inverses in GF(2^m). *IEEE Transactions on Computers*, C-34(8):709–717, August 1985.

[56] C.-S. Yeh, Irving S. Reed, and T. K. Truong. Systolic multipliers for finite fields GF(2^m). *IEEE Transactions on Computers*, C-33(4):357–360, April 1984.

[57] R. Zimmermann. Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication. In *Proceedings of the IEEE Symposium on Computer Arithmetic*, pages 158–166, 1999.

[58] R. Zimmermann, A. Curiger, H. Bonnenberg, H. Kaeslin, N. Felber, and W. Fichtner. A 177 Mbit/s VLSI implementation of the International Data Encryption Algorithm. *IEEE Journal of Solid-State Circuits*, 29(3):303–307, March 1994.