FPLD-Implementation of Computations over Finite Fields $GF(2^m)$ with Applications to Error Control Coding

André Klindworth

University of Hamburg, Dept. of Computer Science
Vogt-Koelln-Str. 30, 22527 Hamburg, Germany
E-mail: klindwor@informatik.uni-hamburg.de

Abstract. This paper investigates the implementation of computations over finite fields $GF(2^m)$ using field-programmable logic devices (FPLDs). Implementation details for addition/subtraction, multiplication, square, inversion, and division are given with mapping results for Xilinx LCAs, Altera CPLDs and Actel ACT FPGAs. As an application example, mapping results for complete encoders for error-correcting codes are also presented. Finally, new opportunities emerging from FPLD technology for data transmission systems with dynamic code adaption are discussed.

1 Introduction

Finite fields have seen an ongoing interest of the scientific community for more than three decades. This is due to the fact that computations over finite fields play a key role in many important applications such as cryptography [12] and error-control coding [3], [4], [9]. Efficient implementations of algorithms for encoding and decoding usually make use of a considerable amount of computations in finite fields $GF(2^m)$. In many cases, a hardware implementation is mandatory to meet strong performance requirements.

2 Construction of finite fields

This section gives a brief introduction to the construction of finite fields and their main properties. For details the reader may refer to [8], [2], [9], [6], or [4].

The finite field over a set F with q elements is denoted by $GF(q)$. In such a field, two operations are defined which are called *addition* (denoted by "$+$") and *multiplication* (denoted by "$\cdot$"), respectively. The smallest finite field is the field $GF(2)$ over the set $\{0, 1\}$. In this field, addition is defined as integer addition modulo-2 (logical XOR) and multiplication is defined as integer multiplication modulo-2 (logical AND).

With respect to hardware implementation, the most important class of finite fields are the fields $GF(2^m)$ for small m ($m \leq 16$). These fields consist of all polynomials over $GF(2)$ with degree smaller than m. To form a field, addition is

Element Number	Polynomial Representation	Polynomial Basis Rep.	Element Number	Polynomial Representation	Polynomial Basis Rep.
0	0	0000	8	x^2+x+1	0111
1	1	0001	9	x^3+x^2+x	1110
2	x	0010	10	x^2+1	0101
3	x^2	0100	11	x^3+x	1010
4	x^3	1000	12	x^3+x^2+1	1101
5	x^3+1	1001	13	$x+1$	0011
6	x^3+x+1	1011	14	x^2+x	0110
7	x^3+x^2+x+1	1111	15	x^3+x^2	1100

Table 1. Elements of $GF(16)$, constructed with $p(x) = x^4 + x^3 + 1$.

defined as polynomial addition and multiplication is defined as polynomial multiplication modulo an irreducible polynomial $p(x)$ with degree m and coefficients from $GF(2)$. $p(x)$ is called the *generator polynomial* of the field.

As an example, table 1 lists all elements of the field $GF(16)$ constructed with $p(x) = x^4 + x^3 + 1$.

The m coefficients of the polynomial representation of a field element form a vector of the m-dimensional vectorspace over $GF(2)$. An element of $GF(2^m)$ usually is represented with respect to a particular basis of this vectorspace. The second column in table 1 shows the vector representation of each element with respect to the basis $\{\alpha^3, \alpha^2, \alpha, 1\}$ with $\alpha = x$. This vector is called the *polynomial basis* (PB) representation of a field element.

3 Computations over $GF(2^m)$

3.1 Addition/Subtraction

In a finite field with characteristic 2, the additive inverse of β is β itself, that is, addition and subtraction are identical operations. Let $\beta, \gamma \in GF(2^m)$ be given in vector representation with respect to the basis $\{\alpha_0, \ldots, \alpha_{m-1}\}$. Then adding β and γ is done by modulo-2 addition of corresponding components.

Obviously, a parallel hardware realisation will use m XOR-gates while a single XOR-gate will be sufficient for bit-serial addition.

3.2 Multiplication

Multiplication in $GF(2^m)$ is much more complicated than addition. Several different hardware implementations have been proposed in the literature (e.g. [1], [13]). Their cost depends on the basis and generator polynomial chosen for the construction of the field.

For the hardware realisation of parallel multipliers, the coefficients of the product each have to be generated by a considerable amount of combinatorial logic. Fig.1 shows as an example a parallel multiplier circuit for the field $GF(16)$.

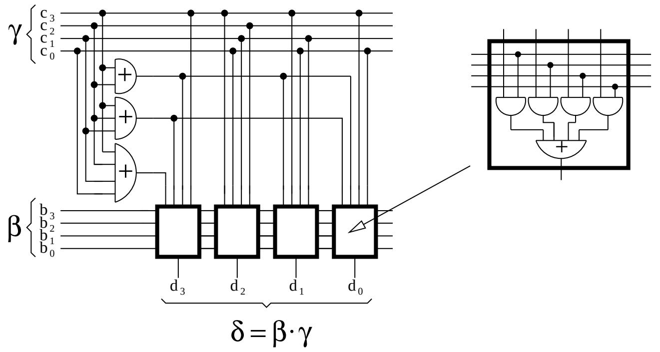

Fig. 1. A PB parallel multiplier over $GF(16)$ with $p(x) = x^4 + x^3 + 1$

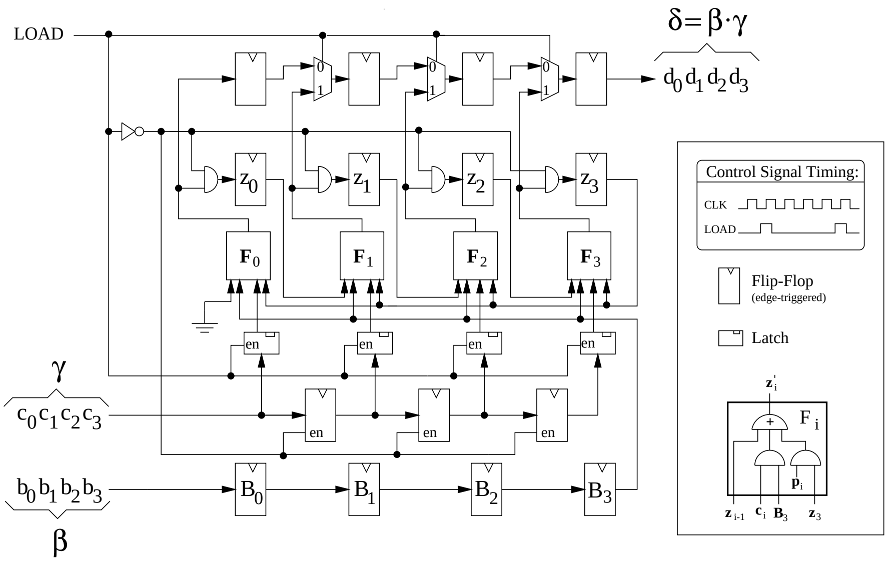

Fig. 2. A classical shift register multiplier over $GF(16)$.

This multiplier uses the PB representation and may be designed according to the procedure described in [7].

Mastrovito has shown that the classical shift register multiplier of Bartee and Schneider [1] indeed is an area- and time-efficient implementation of a bit-serial multiplier [7]. The remainder of this section briefly describes this kind of multiplier and presents its hardware implementation.

With PB representation, multiplying a field element β by α can be done by shifting the components of $beta$ to the left and conditionally adding the coefficents of $\alpha^m = \sum_{i=0}^{m-1} a_i\, \alpha^i$:

$$\begin{aligned}
\alpha \cdot \beta &= \alpha \cdot \sum_{i=0}^{m-1} b_i\, \alpha^i = b_{m-1}\, \alpha^m + \sum_{i=1}^{m-1} b_{i-1}\, \alpha^i \\
&= \sum_{i=0}^{m-1} b_{m-1}\, a_i\, \alpha^i + \sum_{i=1}^{m-1} b_{i-1}\, \alpha^i \\
&= \sum_{i=1}^{m-1} (b_{m-1}\, a_i + b_{i-1})\, \alpha^i + b_{m-1}\, a_0 \quad .
\end{aligned}$$

Bit-serial multiplication of two arbitrary field elements β and γ processes each component of β sequentially. If the current component is 1, γ is added in parallel to an accumulator, which is initially set to zero. After that, the accumulator is multiplied by α and the next component of β is processed. After the m components of β have been encountered, the accumulator contains the product $\beta \cdot \gamma$.

Fig. 2 shows a circuit implementing a bit-serial multiplier for $GF(16)$. Besides the accumulator described above, the multiplier uses three buffer registers which realize a serial-in serial-out data interface.

In many applications where multiplication in $GF(2^m)$ is needed, one of the factors is a constant and thus the circuits presented in this subsection can be simplified.

3.3 Square

A circuit for the multiplication of field element β by itself is comparatively simple. A parallel implementation only has to permute some of the bits b_i and determine the remaining bits by a few (less than m) XOR-operations [7]. As an example, fig. 3 shows a parallel squarer for $GF(2^m)$.

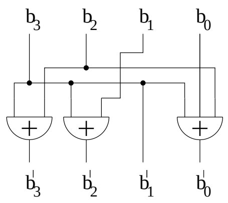

Fig. 3. A parallel squaring circuit for $GF(16)$.

3.4 Inversion/Division

To compute the multiplicative inverse of an element is much more complicated than multiplication. In many hardware implementations, simply tabulating all inverses in a ROM proved to be the most effective solution (e.g. in [11]).

As an alternative to the look-up table approach, bit-serial inversion circuits have been proposed in the literature ([13],[7],[1]) which use serial-to-parallel operand conversion and a parallel multiplier. Such an implementation makes use of the equation $\forall \beta \in GF(2^m): \quad \beta^{2^m} = \beta$

from which follows $\beta^{-1} = \beta^{2^m-2} = \beta^2 \cdot \beta^4 \cdot \beta^8 \cdots \beta^{2^{m-1}}$.

From this relationship, an iterative algorithm for calculating the multiplicative inverse of β can be derived and is shown along with its hardware implementation in fig. 4. The circuit uses two buffer registers for serial-to-parallel and parallel-to-serial conversion, a parallel multiplier, and a parallel squaring circuit.

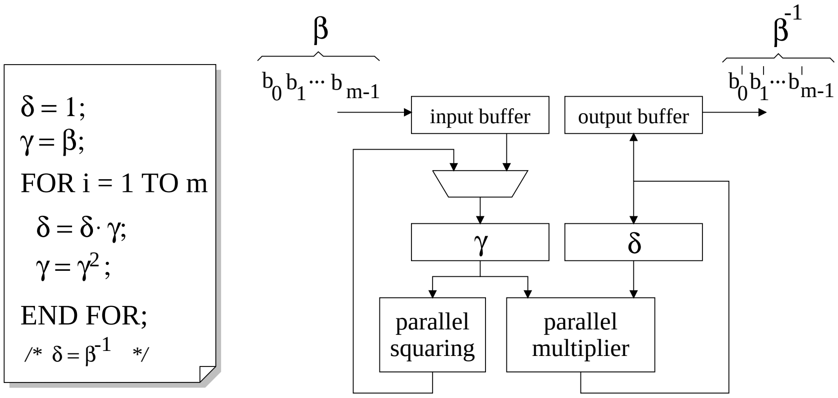

Fig. 4. Bit-serial inversion over $GF(2^m)$ by iterative calculation.

In the case of division, multiplication of the inverted divisor by the dividend can be easily integrated in the circuit of fig. 4 by initializing δ with the dividend instead of 1.

4 Implementation results

4.1 FPLD Architectures

This section presents mapping results for three major FPLD architectures, namely for the PAL-like Altera MAX7000 devices, for the multiplexor-based architecture of the Actel ACT-3 series, and for two representatives of look-up table (LUT) FPLDs: the Xilinx XC4000 LCAs (mixed 3- and 4-input LUTs) and Alteras FLEX8000 series (4-input LUT only). Table 2 gives the number of blocks available in the largest device of each architecture. This allows a normalized comparison of the architectures' respective capacities in terms of *utilization rate*, which is defined as the number of functional blocks used for the implementation of a given function, divided by the number of available blocks in the largest device of the device series.

Device Series	Largest Device	number of functional blocks
MAX7000	EPM7256	256 LCs
FLEX8000	EPF81500	1296 LEs
XC4000	XC4013	576 CLBs
ACT-3	A14100A	1377 LMs

Table 2. Device capacities

4.2 Design method

As has been shown in section 3, multiplication plays a key-role in the implementation of computations over $GF(2^m)$. Hence, the presentation of mapping results focusses on the implementation of multipliers. Delays of mapped circuits are always given for the fastest available device into which the respective design completely fits.

For the Altera devices, mapping results have been produced by automatic synthesis using the vendor's MAX+plusII tool, version 5.2. Mapping for the PAL-based MAX7000 CPLDs had to be guided by manual intervention to find an acceptable realization. The Xilinx FPGAs have been mapped using the Synopsys FPGA Compiler, version 3.2 and Xilinx XACT 5.1. Mapping to Actel's ACT-3 FPGAs has been done manually, since neither libraries for the Synopsys FPGA Compiler nor Actel's ALS development tool had been available at our institute. As all circuits exhibit a highly regular structure, hand-mapping supported by a general-purpose logic synthesis tool (`MIS-II` [5]) proved feasible and results should be comparable to those for the other devices. Reported delays are for the physically mapped circuits, except for ACT-3 devices for which delays have been estimated according to the vendor's rules-of-thumb considering logic depth and signal fanouts.

4.3 Mapping Results

Table 3 gives the mapping results for parallel multipliers for the fields $GF(2^m)$ with $m \in \{4, 6, 7, 8, 9, 10\}$. For each architecture and each field, the number of functional blocks used, the resulting device utilization, the number of block levels in the critical path, and the maximum clock frequency for a register-to-register multiplication are given. For comparative reasons, the rightmost column of table 3 lists the data for a mapping on 2-input NAND/NOR gates. This data has been obtained using `MIS-II` [5].

From this table it can be seen that the PAL-like architecture of the MAX7000 series falls well behind the other devices. Its macrocell structure does not match the XOR-rich logic functions that have to be generated and hence much of the logic resources are wasted. When compared to the respective maximum device capacity, it implements the multiplier five times less efficiently than the FLEX devices, while it is at most 30 % faster. Difficulties in implementing XOR-rich logic functions proved to be a drawback for the MUX-based devices too. Although they give significantly better utilization than the PAL-like architecture,

m	Altera MAX7000	Altera FLEX8000	Xilinx XC4000	Actel ACT-3	2-input NAND/NOR
4	15 LCs 5.8 % 3 levels 42 MHz	14 LEs 1.0 % 2 levels 65 MHz	5 CLBs 0.8 % 2 levels 43 MHz	21 LMs 1.7 % 4 levels ca. 52 MHz	97 gates - 10 levels -
6	27 LCs 10.5 % 3 levels 42 MHz	27 LEs 2.0 % 3 levels 40 MHz	11 CLBs 1.9 % 1 level 52 MHz	45 LMs 3.3 % 5 levels ca. 48 MHz	227 gates - 17 levels -
7	41 LCs 16.0 % 3 levels 42 MHz	40 LEs 3.1 % 3 levels 37 MHz	17 CLBs 3.0 % 2 levels 42 MHz	64 LMs 4.7 % 5 levels ca. 45 MHz	381 gates - 20 levels -
8	55 LCs 21.5 % 3 levels 42 MHz	55 LEs 4.2 % 3 levels 33 MHz	28 CLBs 4.9 % 3 levels 31 MHz	84 LMs 6.1 % 6 levels ca. 38 MHz	569 gates - 16 levels -
9	62 LCs 24.2 % 4 levels 35 MHz	61 LEs 4.7 % 4 levels 28 MHz	27 CLBs 4.7 % 3 levels 34 MHz	105 LMs 7.6 % 6 levels ca. 38 MHz	602 gates - 16 levels -
10	76 LCs 29.7 % 4 levels 35 MHz	76 LEs 5.9 % 4 levels 28 MHz	34 CLBs 5.9 % 3 levels 30 MHz	132 LMs 9.6 % 6 levels ca. 35 MHz	773 gates - 16 levels -

Table 3. Mapping results for parallel multipliers over $GF(2^m)$

Device	resource usage	levels	max. freq.
MAX7000	$5\,m$ MCs	1	67 MHz
FLEX8000	$6\,m$ LCs	2	49 MHz
XC4000	$3\,m$ CLBs	1	58 MHz
ACT-3	$6\,m$ LMs	2	ca. 70 MHz

Table 4. Mapping results for bit-serial multiplier over $GF(2^m)$

MUX-based FPLDs do not reach the capacity of LUT-based architectures. But ACT-3 FPGAs tend to be faster with increasing field dimension m.

Table 3 also shows that the additional 3-input LUT in the XC4000 can be used effectively to reduce the number of CLBs and/or the logic depth in comparison with the pure 4-input LUT structure of the FLEX8000 devices. On average, both device series exhibit the same utilization and speed. But in contrast to Altera's FLEX8000 devices, unused registers of CLBs that implement combinatorial logic functions may still be used for other circuitry in a XC4000 device, e.g. for serial-to-parallel and parallel-to-serial conversion in the bit-serial inversion circuit of fig. 4. This feature would increase the capacity of XC4000 devices when compared to FLEX8000. A similar feature in Altera's new MAX9000 devices will improve the utilization of the MAX7000 series in the same way.

Mapping results for bit-serial multipliers are given in table 4. The complexity of the circuit is proportional to m for all architectures. The PAL-like architecture is almost as fast as the MUX-based architecture, but the latter has a much lower utilization rate. When speed is the main concern, both devices outperform the LUT-FPLDs which again exhibit the highest capacity. Here, the H-block of the Xilinx XC4000 CLB can be used effectively to reduce the logic depth to a single CLB. In addition, since buffering can be overlayed with combinatorial logic in the Xilinx device, XC4000 LCAs also implement serial multiplication more area-efficient than FLEX8000 CPLDs. The latter use approximately one half of their logic resources for buffering purposes.

5 Error Control Coding

5.1 Error-correcting Block Codes

One of the most important application areas of computations in finite fields $GF(2^m)$ is error control coding, namely the implementation of algorithms for encoding and decoding of linear block codes. The general idea of error-correcting block codes (ECBC) is to protect a sequence of k m-bit symbols by $(n-k)$ *parity* or *check symbols* so that the Hamming distance between any two distinct codewords of length n is not less than a selected lower bound d. Then a decoder may reconstruct the original codeword even in the presence of up to $\lfloor (d-1)/2 \rfloor$ errornous symbols. *Cyclic linear block codes* over $GF(2^m)$ proved to be especially well suited for hardware implementation of encoders and decoders. Within this class, the check bits are chosen so that the resulting codeword $(c_{n-1}, c_{n-2}, ...c_1, c_0)$, when interpreted as a polynomial $c(x) = \sum_{i=0}^{n-1} c_i x^i$, is divisable by a polynomial $g(x)$ of degree $(n-k)$. $g(x)$ is called the generator polynomial of the code.

5.2 Cyclic Linear Block Encoders

In order to guarantee that $c(x)$ generated from an information polynomial $i(x)$ is a multiple of $g(x)$, a systematic encoder computes $c(x) = i(x)x^{n-k} - r(x)$, where $r(x) = (i(x)x^{n-k}) \bmod g(x)$ [3]. Such an encoder is sketched in fig. 5. It can be seen from this figure that one should search for a generator polynomial $g(x)$ with the least number of distinct non-zero coefficients since as many $GF(2^m)$-multipliers are needed.

Table 5 lists mapping results for the implementation of encoders for a popular class of ECBCs: the Reed-Solomon codes. For each of the three codes, results for an encoder which uses either a parallel or a serial multiplier circuit are given.

Table 5 also summarizes the main parameters of the codes, namely the type of the code, the code's blocklength n, the number of information symbols per block

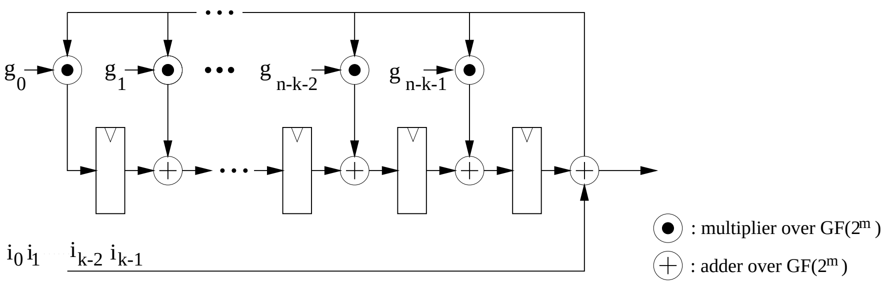

Fig. 5. Principle structure of an encoder for a cyclic linear block code

k, the number of distinct non-zero coefficients of $g(x)$, g, the minimum distance d, and the number of correctable symbol errors per block t. According to Berlekamp [3], the first type of code named *rs31* has been adopted by the Joint Tactical Information Distribution System (JTIDS) of the U.S. Armed Forces. The type of code named *rs255* has widely been used in space communication and therefore is sometimes called the NASA-code. Finally, the shortened Reed-Solomon code *rs32* is one of the types of Reed-Solomon codes used in the cross-interleaved code for the Compact Disk [10].

It can be seen that except for the PAL-like architecture, Reed-Solomon encoders can be realized efficiently with field-programmable logic. Bit-serial encoders exhibit an encoding rate of approximately 30 Mbit/sec. In most cases, parallel encoders are as area-efficient as their serial counterparts or are even smaller. Thus, for bit-serial encoding it would be best to use a parallel encoder with serial-to-parallel and parallel-to-serial conversion at the interface. This would reduce the apparent logic depth to a single functional block and should allow encoding rates near 100 Mbit/sec.

5.3 Dynamic Code Adaption

In case an in-circuit reconfigurable device is used for encoder implementation, FPLDs give rise to a new interesting feature for data transmission over noisy channels. With FPLDs it becomes possible to build encoders with dynamic code adaption based on measurements of the channel quality. Fig. 6 shows a system for data transmission that uses this feature. At the side of the sender, an encoder adds parity symbols to the stream of information symbols, sending codewords to the modulator. The receiver first demodulates the incoming signals from the *data channel* and than decodes a received word $v(x)$ to restore the original information $i(x)$. Decoder and demodulator at the receiving end collect information on the number of detected/corrected errors and on the signal noise. This data is handed over to an unit that evaluates the channel quality. If the ECC currently used on the data channel does no longer match the channel's actual noise level, coding may be switched to another code by reconfiguring the encoder and the decoder. Reconfiguration of the encoder is induced over a second low bandwidth channel

Code Acronym	Code Description	Code Parameter n	k	g	d	t
rs31	Reed-Solomon code over $GF(2^5)$	31	15	8	17	8
rs255	Reed-Solomon code over $GF(2^8)$	255	223	16	33	16
rs32	shortened RS code over $GF(2^8)$	32	28	3	5	2

Code	Altera MAX7000	Altera FLEX8000	Xilinx XC4000	Actel ACT-3
rs31 parallel	90 MCs 45.5 MHz	90 LEs 29 MHz	45 CLBs 28 MHz	116 LMs ca. 45 MHz
rs31 bit-serial	167 MCs 40 MHz	172 LEs 31 MHz	86 CLBs 35 MHz	211 LMs ca. 40 MHz
rs255 parallel	no fit	314 LEs 22 MHz	145 CLBs 27 MHz	478 LMs ca. 40 MHz
rs255 bit-serial	no fit	547 LEs 27 MHz	264 CLBs 28 MHz	645 LMs ca. 42 MHz
rs32 parallel	78 MCs 40 MHz	55 LEs 28 MHz	26 CLBs 34 MHz	79 LMs ca. 42 MHz
rs32 bit-serial	74 MCs 45 MHz	77 LEs 33 MHz	38 CLBs 40 MHz	91 LMs ca. 50 MHz

Table 5. Code parameters and mapping results for some Reed-Solomon encoders.

which may carry the configuration data as well. Of course, the configuration channel has to be protected against errors, too. If enough energy is available at the sender, this may be easily accomplished by a high signal-to-noise ratio (SNR). Such a situation is assumed in fig. 6 where the sender may be earthbound while a satellite receiving the configuration data has very limited power. The configuration channel needs to sustain a much smaller bandwidth than the data channel. This eases error protection and allows the use of an inefficient but simple ECC that is trivial to decode (e.g. a repetition code).

6 Conclusion

The implementation results presented show that FPLDs are suitable for the implementation of finite field arithmetic. In this application area, PAL-like devices are significantly inferior to MUX-based FPLD architectures. FPLDs which use LUTs in their functional blocks give the highest capacity at an acceptable speed. The mapping results in section 5 show that even complex and powerful error-correcting codes may be encoded with FPLDs. It remains to be seen whether FPLD-based *decoding* is feasible as well. Besides facilitating rapid prototyping and low-volume production in the field of error control coding, in-circuit reconfigurable FPLDs give rise to data transmission systems with dynamic code adaption based on channel measurement information.

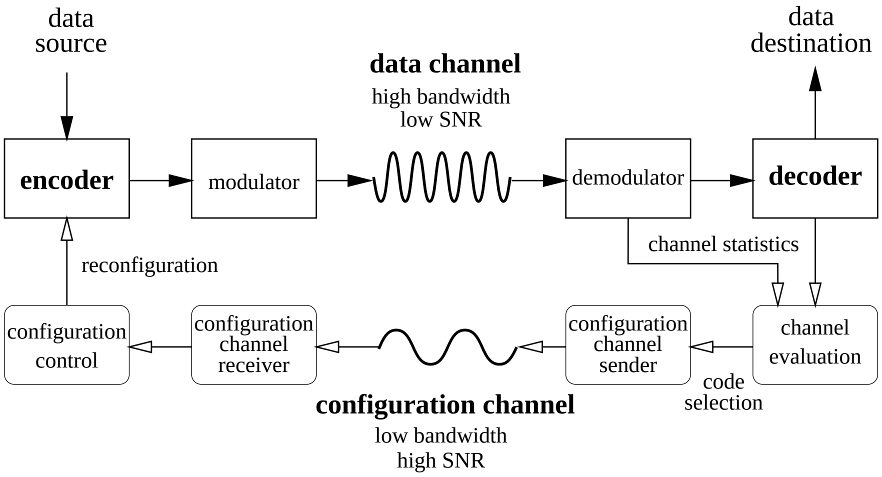

Fig. 6. A data transmission system with dynamic code adaption.

References

1. T. C. Bartee and D. I. Schneider, *Computation with finite fields*, Inform. Contr., vol. 6, pp.79-98, Mar. 1963.
2. E. R. Berlekamp, *Algebraic Coding Theory*, New York: McGraw-Hill, 1968.
3. E. R. Berlekamp, *The Technology of Error-Correcting Codes*, Proc. IEEE, vol. 68, pp.564-593, May 1980.
4. R. E. Blahut, *Theory and practice of error control codes*, Addison-Wesley, 1983.
5. R. K. Brayton et al., *MIS: A Multiple-Level Logic Optimization System*, IEEE Trans. on Computer Aided-Design, CAD-6(6), pp. 1062 - 1081, Nov. 1987
6. R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, London, New York: Cambridge University Press, 1986
7. E. D. Mastrovito, *VLSI designs for computations over finite fields $GF(2^m)$*, Ph.D. Thesis No.159, Dept. of Electr. Engineering, Linköping Univ., Sweden, Dec. 1988.
8. R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Boston,MA: Kluwer Academic Publishers, 1987.
9. P. J. McWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Amsterdam: North-Holland, 1978.
10. H. Hoeve, J. Timmermans and L. B. Vries, *Error correction and concealment in the Compact Disk system*, Philips tech. Rev. 40, No. 6, 1982.
11. J.-L. Politano and D. Deprey, *A 30 Mbits/s (255,223) Reed-Solomon Decoder*, EUROCODE '90, Int't Symp. on Coding Theory and Applications, Udine, Italy, Nov. 1990.
12. H. C. A. van Tilborg, *An Introduction to Cryptology*; Boston,MA; Kluwer Academic Publishers, 1988.
13. C. C. Wang et. al., *VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$*, IEEE Trans. Comput., vol. C-34, pp.709-717, Aug. 1985.