Chapter 2
Hardware Architectures for Channel Encoding in Information Transmission Systems

Nowadays, digital representation of any kind of information source is common. Speech waveforms, image waveforms, text files and any other information sources are represented as binary sequences. In order to pass the information from source to destination, the source output, represented as a binary sequence, is further on converted into a form suitable for transmission over a particular physical media (e.g. cable, optical fiber etc.). In other words, information transmission systems are used to pass digital sequences from source to destination trough a particular physical media defined as the communication channel. Considering the aforementioned principle of digital communications, two fundamental ideas are distinguished: communication sources are viewed as digital sequences and communication channel demand the encoding of the digital sequences in a form suitable for reliable transmission [1]. Consequently, digital representation of the information sources is known as *source coding*, whereas the conversion of the source digital sequences in a form suitable for the communication channel is called *channel encoding*. The present chapter presents introductory sections for both information transmission systems and channel encoding, followed by hardware implementations of coder and decoder architectures in case of linear block codes (i.e. Hamming and cyclic codes).

2.1 Introduction to Information Transmission System

Accurate and timely data, presented in a context that gives it meaning and relevance can be defined as *information* [2]. The word information comes from ancient Greek, and is derived from the words "eidos" (idea) and "morphe" (shape, form). Joining the two terms suggests that, the mind interpretation of objects generates information.

With regards to the *Information and Communication Technologies* (ICT), information is embedded into a physical form (e.g. electromagnetic wave) in order

B. Belean, *Application-Specific Hardware Architecture Design with VHDL*,
Signals and Communication Technology,
https://doi.org/10.1007/978-3-319-65025-8_2

to be transmitted through a communication channel. Consequently, in ICT information involves an information source *S*, a destination *D* and a physical medium (i.e. transmission channel) that assures the information transmission from *S* to *D*. The information source can be discrete (digital source), or continuous (signal source). In our case, 0 and 1 symbols are used for information representation, thus information is discrete and specific for digital communications.

In digital communication, the ensemble of interdependent blocks used to transfer the information from source to destination is called *Information Transmission System* (ITS). As mentioned before, source coding and channel coding are independently performed using a binary interface between source and channel. Using digital sequences for information transmission is specific for digital communications systems.

The main advantages of digital communication systems are, on one hand, the possibility of flexible implementation and low cost considering the advent of digital logic circuits, and, on the other hand, the reliability and the possibility to ensure information confidentiality [2]. Digital communications come with the disadvantage, the increased bandwidth compared to analogue information transmission systems. Nevertheless, the disadvantage is diminished due to the existing computational power and the possibility of data compression.

2.1.1 Modelling an Information Transmission System

The schematic diagram depicted in Fig. 2.1 represent a digital ITS.

The *source encoder* converts the input signals into a sequence of bits *i*. For example, a number of *m* source messages to be transmitted are coded using *n* bits for each source message, where $m < 2^n - 1$. The main benefits of this source message conversion into a digital sequence are the inexpensive digital hardware and possibility of source/channel separation.

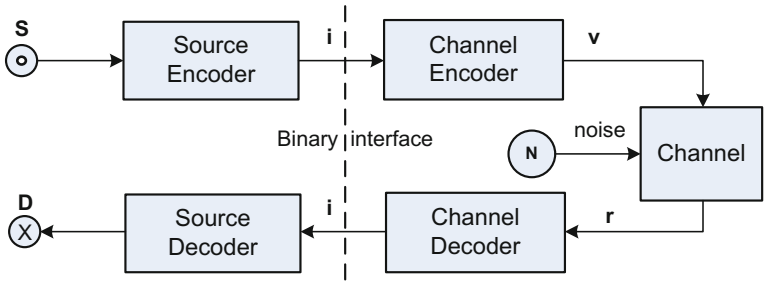

Fig. 2.1 Schematic model of an ITC for information transmission from sours S to destination D. The source messages are encoded in binary sequences i, and further on converted into a codeword *v* for channel transmission. The channel decoder performs error detection and correction on the received codeword v

The transmission channels, especially the electromagnetic field specific for wireless communication, are susceptible to noise which may alter the signals. The role of the channel encoder and decoder is presented next, considering noisy communication channels. To start with, the noise in communication channel is shortly described. Commonly, a channel model includes an input waveform *x(t)*, a noise waveform *n(t)* and the output waveform *y(t)*. In our case, the noise waveform is considered an additive white Gauss noise (AWGN). The input signal is characterized by its power *P*, whereas the communication channel is of bandwidth *B*. A characteristic of the communication channel is the signal per noise ratio SNR, given by the input signal power level compared to the background noise power, as denoted by ξ from Eq. (2.1). The SNR is often expressed in dB.

$$\xi = \frac{\overline{[y(t)]^2}}{\overline{[n(t)]^2}}, \xi_{[dB]} = 10 \log_{10} \xi \tag{2.1}$$

Considering channel noise levels, Shannon defined the maximum decision rate D_{max} [bits/s] that can be error free transmitted through an AWGN communication channel of bandwidth *B*. By decision rate we understand the number of bits per second delivered by the channel encoder block. Consequently, the channel capacity C is given by Eq. (2.2).

$$C = B \log_2(1 + \frac{P}{N}) = B \log_2(1 + \frac{P}{BN_0}) \tag{2.2}$$

where *N* is the noise power and N_0 is the noise power per unit bandwidth. The channel capacity as previously defined represents a theoretical limit impossible to be reached in real transmission systems. Nevertheless, state-of-the-art channel coding schemes together with remarkable computational power offered by digital hardware such as GPU or FPGA allow to closely approach this channel capacity, whereas high-throughput is also delivered. A classic channel coding scheme is detailed further on.

The channel coding, i.e., the *channel encoder* and the *channel decoder*, converts the information *i* into a codeword *v*, which is transmitted to the channel. At destination, the channel decoder block receives the codeword *r*, and decodes the sequence of information bits *i'*. These conversions are performed in order to ensure a high degree of accuracy at destination, in spite of the noisy transmission channels. In other words, redundant bits are added by the channel encoder to the information bits *i* and the codeword *v* is obtained. Based on the added redundant bits, the decoder performs error detection and correction of the information bits. A more detail description of error control using channel encoding is presented in the next section.

2.2 Introduction to Channel Encoding for Error Control

When information transmission is performed through a noisy communication channel, the signal carrying the information is susceptible to alterations due to the channel noise. Nowadays, considering the high impact of communication in economic and social life, data transmission requires increased reliability, high speed and increased throughput. Thus, protection against channel noise is considered in order to eliminate this unwanted effect. The error protection is performed through channel coding. The history of error control coding starts in mid 20th century and began with Hamming codes and reached to more powerful error correction codes such as low density parity check codes (LDPC) or turbo-codes, trying to limit technically errors effect in applications. The solution for achieving error protection while transmitting information was proposed by C. E. Shannon is known as Shannon second theorem namely noisy channels coding theorem [3].

Basically, the theorem proves that on a noisy channel having the capacity C, a real time transmission with D bit rate and an error probability $P(E)$ as small as wanted, using uniform codes of length n, such that: $P(E) \leq 2^{-n \cdot e(D)}$ ($e(D)$ is a positive function of D, completely determined by channel characteristics). This means an error probability $P(E)$ however small can be obtained in two ways: by increasing D which lead to inefficient channel usage and by using large codewords by adding redundant bits [2]. The last approach is used in practice for error control codes for data transmission over noisy channels. The strategies for error control in data transmission are classified as follows: (i) error detection which informs the transmitter about the need for retransmission of erroneous data (ARQ—automatic repeat request); (ii) forward error correction (FEC) in which case the channel coding approach automatically corrects an amount of errors at the receiver side; (iii) error correction and detection which involves error correction through coding according to error correction code capacity and repeat transmission for the rest of erroneous combinations (e.g. radio transmissions). To conclude, error detection and correction is achieved by channel encoding.

2.2.1 Representation of Error Control Codes

The representations used for binary code sequences can be classified as matrix, vector, polynomial and geometrical representations described as follows:

- matrix representation implies all the code words are stored in matrix, excepting the zero components; let m be the number of codewords and n be the length of one codeword, than the whole code may be stored in the next matrix ($a_{ij} \in \{0; 1\}$):

$$\mathbf{C} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{matrix} \Rightarrow \mathbf{v_1} \\ \Rightarrow \mathbf{v_2} \\ \dots \\ \Rightarrow \mathbf{v}_m \end{matrix} \tag{2.3}$$

- vector representation of codes considers each code word of length n as a n—length vector from the vector space $\boldsymbol{V_n}$. In other words each sequence of length n is represented by a vector as shown below:

$$\mathbf{v} = (\, a_1 \quad a_2 \quad \dots \quad a_n \,), \tag{2.4}$$

  where $a_i \in \{0; 1\}$ for binary codes.
- polynomial representation of a length n codeword $\boldsymbol{v}$ is made through a polynomial of degree n and unknown x, for which the coefficients are represented by the codeword bits a_i. Polynomial representation example:

$$v(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ \dots + a_1 x + a_0 \tag{2.5}$$

- geometric representation implies that the code words of length n are represented as points which defines the peaks of a geometrical shape, in a n-dimensional space; the main benefit is that the representation allows using a series of well known proprieties of geometric figures to the codes.

Within the current chapter, vector representation and polynomial representation of codes will be used in case of Hamming codes and cyclic codes respectively.

2.2.2 Classification of Error Control Codes

Considering the nature of the processed information, the error control codes used to detect, to correct or for both error correction and detection can be classified as:

- block codes in which case, the information at the encoder input is divided into blocks of m symbols/bits to which the encoder adds k control symbols resulting in an $n = m + k$ length codeword.
- convolutional codes in which case the information bits are processed in a continuous way; e.g. an n bits codeword is coded into another n bits codeword based on the coding relations.

Taking into account the types of the error propagated through the communication channel, the error control codes are:

- codes for independents error correction;
- codes for burst error correction;

Based on the possibility of having or not, the information and control symbols positions well defined in a codeword, the error control codes can be classified in:

- systematic codes for which both the information bits i and the control bits c positions are clearly defined;
- non-systematic codes in which case the information bits and control bits are not given in clear considering the codeword structure.

2.2.3 Error Control Codes Parameters

The performances and characteristics of error control codes are illustrated by a set of parameters which are further on detailed. Let us consider the codeword length given by parameter n and the number of information bits is m and the control bits k in case the information bits and control bits are separable.

Redundancy (R), as shown in Shannon second theorem, is used to achieve error protection by adding supplementary bits for error detection or correction. Depending on the type of error control codes (i.e. separable or non-separable corresponding to the possibility to separate the redundant bits from information bits or not) the redundancy is computed as presented bellow.

For separable codes, redundancy is given by:

$$\mathrm{R} = \frac{\mathrm{k}}{\mathrm{n}}, \tag{2.6}$$

whereas for non-separable codes,

$$\mathrm{R_r} = \frac{\log_2(\text{total no. of nlength sequences}) - \log_2(\text{no. of codewords})}{log_2(\text{total number of nlength sequences})} \tag{2.7}$$

Detection/correction capacity (C_d/C_s) represents the ration between the number of detectable/correctable erroneous sequences and the total number of erroneous sequences.

Code distance (d) is a parameter indicating the capacity of the code to detect and correct errors. The relation between the code distance and the error detection/correction capability are demonstrated in [4].

Let us first define the Hamming weight w of a codeword $\boldsymbol{v}$ by the number of non-zero symbols of the respective codeword. Moreover, let $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ be the two codewords for which the hamming distance is intended to be computed. The $\mathrm{d}(\boldsymbol{v}_i,\boldsymbol{v}_j)$ is defined by the number of positions for which $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ differ, and denotes the hamming distance: $\mathrm{d}(\mathbf{v_i}, \mathbf{v_j}) =: \sum_{k=1}^{n} a_{ki} \oplus a_{kj}$ where $\mathbf{v_i} = (a_{1i}, a_{2i}, \ldots, a_{ni})$, $\mathbf{v_j} = (a_{1j}, a_{2j}, \ldots, a_{nj})$ and $\oplus$ is modulo 2 summation.

As referred to the error correction capability related to the hamming distance, the necessary and sufficient condition for a code to correct maximum t errors is $d \geq 2t + 1$. Considering the error detection capacity, the necessary and sufficient condition for a coded to detect maximum *e* errors is $d \geq e + 1$. The necessary and sufficient condition for a code to simultaneously correct maximum t errors and detect maximum *e* errors is $d \geq t + e + 1,\ e > t$ [2].

2.3 Block Codes

Considering a block code, information from a binary source is divided into *m*-bits blocks representing the information bits $\boldsymbol{i}$. For each information symbols *m* we add *k* control symbols according to a coding rule. The result is a codeword $\boldsymbol{v}$ of length *n*, according to Eq. (2.8).

$$n = m + k \tag{2.8}$$

Considering such a structure, the number of resulted codewords is given by M, with $M = 2^m$. Let $\mathbf{V_n}$ (dimension n) denote the space of vectors containing all the codewords of *n* bits (with coefficients in Galois Field GF(2)). In case 2^m codewords form a vector space of dimension *m* which is a subspace *C* of the space $\boldsymbol{V_n}$, we call the block code linear, and the linear block code is denoted by the pair *(n, m)*. Such linear structures are very important for the practical applications for the block codes. Another interpretation of the block codes linearity is that, a block code is linear if and only if the modulo *2* sum of two code words is also a code word. It follows that the vector space of dimension *n*, $\boldsymbol{V_n}$ containing the set of distinct combinations that can be generated with *n* bits (2^n) is divided into two distinct sets:

C—the set of code-words,

$F = V_n - C$—the set of false code-words.

The linear block codes were invented after Shannon gave his second theorem (1948). Thus, R. Hamming and Golay introduced the systematic liner codes (1950), whereas the unified theory for linear codes was documented in 1960. Many practical applications in information theory and coding were developed since, which make use of linear block codes [5]. Further on, Hamming codes and cyclic codes are exemplified, but first the coding equations are detailed.

2.3.1 Coding Equations

As mentioned before, the linear code *C(n,m)* is an *m*-dimensional subspace of V_n. Consequently, the entire set of codewords can be generated by the linear combinations of the *m* linear independent vectors belonging to *C*. Let the set of linear independent code vectors be denoted by $\mathbf{g_i},\ i = \overline{1, m}$. This set of vectors can be

written as the matrix from Eq. (2.9) and represents the *code generating matrix* $G_{[mxn]}$.

$$\mathbf{G} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ g_{m1} & g_{m2} & \cdots & g_{mn} \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{g_1} \\ \mathbf{g_2} \\ \\ \mathbf{g_m} \end{bmatrix} \tag{2.9}$$

where g_{ij} are binary symbols.

The encoding equation is given by:

$$\mathbf{v} = \mathbf{iG} = [i_1 i_2 \ldots i_m] \begin{bmatrix} \mathbf{g_1} \\ \mathbf{g_2} \\ \\ \mathbf{g_m} \end{bmatrix} = \mathbf{i}_1\mathbf{g}_1 + \mathbf{i}_2\mathbf{g}_2 + \ldots + \mathbf{i}_n\mathbf{g}_n \tag{2.10}$$

In order to obtain a systematic structure of the codeword $\mathbf{v}$, denoted by $\mathbf{v}' = [\mathbf{i} \quad \mathbf{c}]$ the generating matrix $\mathbf{G}$ must have the *canonical form*:

$$\mathbf{G}' = [\mathbf{I_m P}] \tag{2.11}$$

where $\mathbf{I_m}$, is the unit matrix of order m.

The m information symbols are found unchanged in the codeword $\mathbf{v}$ structure and that the k control symbols are linear combinations of information symbols. Let c_i be the k control symbols and i_j the m information symbols. The control symbols are computed based on the information bits, according to the f_i functions as shown by Eq. (2.12).

$$c_i = f_i(i_j),\ i = \overline{1,k},\ j = \overline{1,m} \tag{2.12}$$

Equation (2.12) are known as the parity check equations. Considering the binary vector spaces properties [6], if we have a space C of dimension m, then always exists an orthogonal space C* included in the vector space C such that a codeword $v^{\in}C$ is orthogonal in C*. The linear independent k vectors belonging to the orthogonal space C* can be put in a matrix $\mathbf{H_{[kxn]}}$ named the parity check matrix.

The spaces $\mathbf{C}$ and $\mathbf{C}^*$ being two orthogonal spaces means that the two matrices $\mathbf{G}$ and $\mathbf{H}$ are also orthogonal, as expressed by Eq. (2.13).

$$\mathbf{GH^T} = \mathbf{HG^T} = \mathbf{0} \tag{2.13}$$

The coding Eq. (2.10), relative to the vector space $\mathbf{C}^*$ becomes:

$$\mathbf{Hv^T} = \mathbf{0} \tag{2.14}$$

2.3.2 Decoding Equations

Let us consider the communication channel and from Fig. 2.2. The information bits ***i*** are coded using the channel coder and the codeword v is obtained which is transmitted through the communication channel. The binary codeword ***v*** is affected by channel noise, and, consequently the error vector ***e*** contains "1" values as additive noise for the bits position. The ***e*** vector is unknown at the receiver side. Thus, the vector ***r*** is received which contains errors.

The received codeword represented by the binary vector ***r*** is given by Eq. (2.15).

$$\mathbf{r} = \mathbf{v} + \mathbf{e} \tag{2.15}$$

The error vector ***e*** indicates the additive errors due to the noise within the communication channel. The sign + is the modulo 2 summation of the two vectors, codeword ***v*** and the error vector ***e***, respectively.

Considering the matrix representation, the error vector e may be written as:

$$\mathbf{e} = [\mathrm{e}_1 \quad \mathrm{e}_2 \quad \ldots \quad \mathrm{e}_n]$$

where an e_i value of "1" means an error at position I, whereas a value of "0" means an correct bit within the received codeword *r*. As far as for the *decoding equation* at the receiver side, it is based on the syndrome S, as denoted by Eq. (2.16). The syndrome represents a column matrix with *k* elements.

$$\mathbf{H}\mathbf{r}^{\mathbf{T}} = \mathbf{S} \tag{2.16}$$

By replacing **r** vector we obtain:

$$\mathbf{H}(\mathbf{v} + \mathbf{e})^{\mathbf{T}} = \mathbf{H}\mathbf{e}^{\mathbf{T}} = \mathbf{S} \tag{2.17}$$

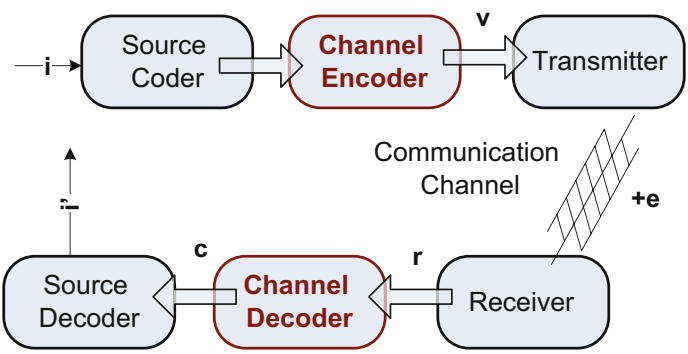

Fig. 2.2 Channel coder/decoder implementation scheme; blocks of ***i*** information bits are coded and transmitted through the communication channel; the receiver gets the codeword transmitted to the channel (***r***) and corrects the errors denoted by ***e***

Equation (2.17) denotes that the syndrome S depends only on the channel errors. Consequently, if there are no errors or the errors cannot be detected, the syndrome S = 0. Meanwhile, if the syndrome $\mathbf{S} \neq \mathbf{0}$ then the errors are detected. In case the error correction is wanted, it is necessary to determine the error position based on the resulted syndrome. This is the case for the Hamming codes. An example of a coder/decoder implementation for the Hamming codes is detailed further on.

2.4 Hamming Coder/Decoder Implementations

Hamming codes were introduced after Shannon second theorem by R. Hamming in 1950. In our case, we detail a Hamming group code for the correction of single errors.

The characteristics of this code are:

- the code length can be determined by:

$$n = 2^k - 1 \tag{2.18}$$

- the codeword structure is given by the Eq. (2.19), where a_i represent the information symbols and c_i represent the control (parity check) symbols;

$$\mathbf{v} = [c_1 c_2 a_3 c_4 a_5 a_6 a_7 c_8 a_9 \ldots a_n] \tag{2.19}$$

- the control symbols are placed at positions 2^i within the codeword, with $i = \overline{0, k-1}$;
- the control matrix **H** is given by Eq. (2.20), where each column $\boldsymbol{h_i}$ expresses in binary natural code (BN) its position with the less significant bit LSB on the k^{th} line;

$$\mathbf{H}_{[k \times n]} = [\mathbf{h_1} \quad \ldots \quad \mathbf{h_i} \quad \ldots \quad \mathbf{h_n}] \tag{2.20}$$

The coding relationships are determined using Eq. (2.14). Consequently, the control symbols are expressed as a linear combination of information symbols, as expressed by Eq. (2.12).

Regarding the decoding process, having the received codeword $\boldsymbol{r}$, it verifies the decoding Eq. (2.16). Thus the **S** syndrome is determined by Eq. (2.21).

$$\mathbf{S} = [\mathbf{h_1} \quad \ldots \quad \mathbf{h_n}] \begin{bmatrix} 0 \\ \ldots \\ e_i \\ \ldots \\ 0 \end{bmatrix} = \mathbf{h_i} \tag{2.21}$$

When only one error occurs, the syndrome indicates a binary number h_i, corresponding to the error position within the codeword $\boldsymbol{r}$. Thus, using a binary-decimal conversion, we can determine from the S syndrome the erroneous position. In case more than one error occurs, a major disadvantage comes up. The supplementary errors are not detected, whereas the decoder introduces supplementary errors. To deal with the aforementioned disadvantage, modified Hamming codes are available (extended or shortened Hamming codes) which allow supplementary errors detection or correction.

Following the aforementioned descriptions for coding and decoding processes, hardware architectures are built both for the coder and the decoder, in case of a Hamming (7, 4) group code.

2.4.1 Encoder Implementation

The Hamming (7, 4) codeword structure is presented as follows:

$$\mathbf{v} = [\mathrm{c}_1 \ \mathrm{c}_2 \ \mathrm{a}_3 \ \mathrm{c}_4 \ \mathrm{a}_5 \ \mathrm{a}_6 \ \mathrm{a}_7] \tag{2.22}$$

The parity check matrix $\mathbf{H}$ and the encoding equations are denoted by the Eq. (2.23).

$$\begin{aligned} \mathbf{H} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \\ \mathbf{H} \cdot \mathbf{v}^{\mathbf{T}} &= \mathbf{0} \end{aligned} \tag{2.23}$$

The control symbols are expressed as a linear combination of information symbols according to the following set of equations:

$$\begin{aligned} \mathrm{c}_1 &= \mathrm{a}_3 + \mathrm{a}_5 + \mathrm{a}_7 \\ \mathrm{c}_2 &= \mathrm{a}_3 + \mathrm{a}_6 + \mathrm{a}_7 \\ \mathrm{c}_4 &= \mathrm{a}_5 + \mathrm{a}_6 + \mathrm{a}_7 \end{aligned} \tag{2.24}$$

Example 2.1 Hardware architecture for the Hamming encoder
The hardware architecture corresponding to the Hamming encoder is described in Fig. 2.3a. The main components are: a shift register with parallel load for each codeword; three adders for calculating the parity check bits c_1, c_2 and c_4. According to the equations set (21), the control bits are calculated and loaded into the shift register *RD* in the corresponding positions: *1*, *2* and *4*. In this manner, the code word $\boldsymbol{v}$ is formed using the shift register *RD*.

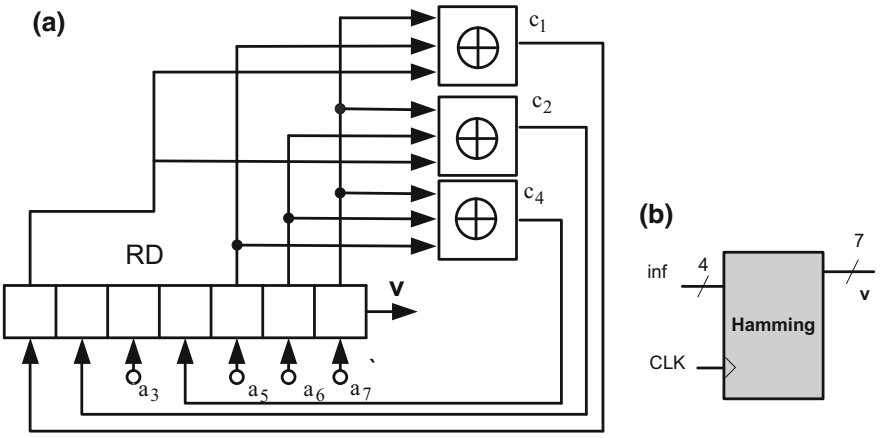

Fig. 2.3 **a** Hamming coder implementation using shift register RD, **b** logic block for the Hamming encoder (inputs/outputs)

The logic block for the Hamming encoder intuitively called Hamming is illustrated in Fig. 2.3b. The Hamming logic block has a 4 bits input representing the information bits and an output of 7 bits for the codeword resulted after encoding. The corresponding VHDL code for the entity associated with the Hamming encoder is presented next:

```
---------- Entity declaration ----------
entity Hamming is
Port ( Clk: IN std_logic;
       inf: IN std_logic_vector(3 downto 0);
       data_paralel:   OUT std_logic_vector (6 downto 0));
END Hamming;
---------- End of entity declaration ----------
```

The corresponding VHDL code which describes the Hamming encoder functionality is presented next. A shift register *RD* with parallel load is used to store the received codeword. The register is represented by the *temp* internal signal, whereas the parallel load operation is performed through the concurrent statement *temp (j) = inf(i)*. Note that *j = {6, 5 4, 2}*, the position of information bits, whereas *i = 0 to 3*. The 7 bits output is also assigned in a concurrent manner to the *temp* register output (e.g. *data_parallel <= temp*). In order for the *temp* signal to represent a register within the behavioral architecture, the control bits are computed by *xor* operator and assigned on the rising edge of the *clk* input, on the positions 0, 1, and 3 (e.g. `temp(0) <=inf(3) xor inf(0) xor inf(1);`).

```
---------- Behavioral description of the Hamming encoder ----------
architecture Behavioral of Hamming is
```

```
signal temp: std_logic_vector (6 downto 0);
begin
    process (clk)
    begin
        if clk'event and clk = '1' then
            temp(3) <=inf(1) xor inf(2) xor inf(3);
            temp(1) <=inf(0) xor inf(2) xor inf(3);
            temp(0) <=inf(3) xor inf(0) xor inf(1);
         end if;
      end process;
   data_paralel <= temp;
   temp(6) <=inf(3);   temp(5) <=inf(2);
   temp(4) <=inf(1);   temp(2) <=inf(0);
end Behavioral;
-------------End of behavioral description of the Hamming encoder ------------
```

Figure 2.4 illustrates the parallel load of the input (i.e. the information bits *inf*) into the shift register RD at simulation time t_0 and t_2. The resulted codewords after adding the control bits are underlined at simulation time t_1 and t_3 as underlined in Fig. 2.4.

Example 2.2—Hardware architecture for the Hamming decoder
The Hamming decoder receives the codeword r through the communication channel. In case of the Hamming (7, 4) code, the error prone received codeword is denoted by Eq. (2.25).

$$r = [r_1 r_2 r_3 r_4 r_5 r_6 r_7] \tag{2.25}$$

Taking into account Eq. (2.14), the Hamming decoder corrects or detects the transmission errors based on the syndrome S. The following cases are distinguished based on the S syndrome values:

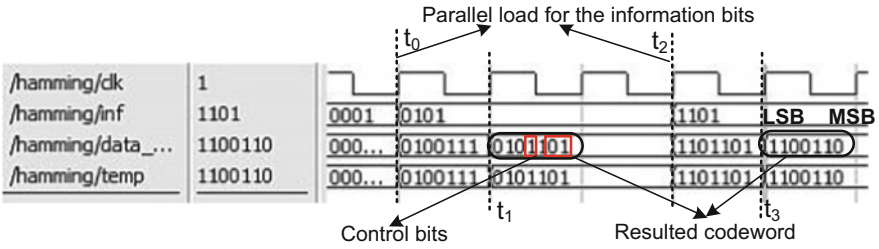

Fig. 2.4 **a** Simulation of the Hamming encoder; the resulted codewords corresponding to the information bits *bx0101* and *bx1101*are underlined

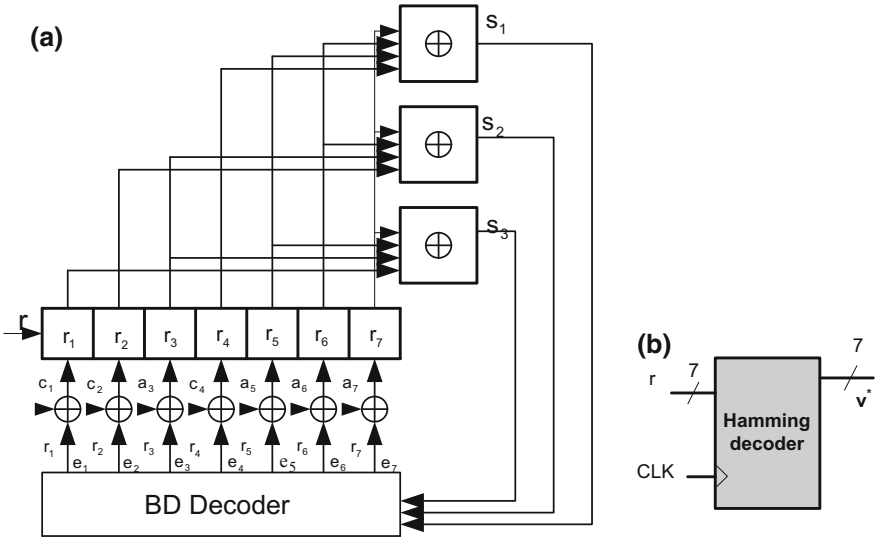

Fig. 2.5 **a** Block diagram for the Hamming decoder composed of a shift register, adders and binary to decimal decoder (BD decoder), **b** the black-bocx corresponding to the Hamming decoder

- $S = 0$ meaning there are no errors or the errors are not detected;
- $S \neq 0$ meaning the error is found at the position given by the binary representation of the syndrome S (e.g. the syndrome $S = [011]$ denotes an error on the 3rd position within the received codeword r).

The hardware architecture for the Hamming decoder based on syndrome decoding is described in Fig. 2.5a. The main components are: a shift register where the received codeword is loaded in parallel, three adders for computing the syndrome bits ($S = [s_1, s_2, s_3]$), a binary to decimal decoder (i.e. BD decoder) for computing the position of the erroneous bit. Once the received codeword r is loaded in the shift register, the syndrome S is computed. Further on, the computed binary value of the S syndrome is decoded into a 7 bits binary vector e, which marks the position of the erroneous bit. The received codeword r is corrected by adding "1" logic value to the position specified by the BD decoder. Thus, the corrected codeword is denoted by Eq. (2.26).

$$\mathbf{v}^* = \mathbf{r} + \mathbf{e} \tag{2.26}$$

The logic block for the Hamming decoder is illustrated in Fig. 2.5b. It has a 7 bits input *y_in* representing the received codeword through the communication channel and an output of 7 bits (*y_out*) for the corrected codeword resulted after decoding process. There are also the clock signal *clk* and the reset input *reset*

available for the Hamming decoder. The corresponding VHDL code for the entity associated with the Hamming decoder is presented next:

```
———————— Entity declaration————————————————————————————————
      entity decoder is
      Port ( clk:          in std_logic;
                reset :  in std_logic;
                v_in:     in std_logic_vector(0 to 6);
                v_out:   out std_logic_vector(0 to 6)
                );
         end decoder;
———————— End of entity declaration————————————————————————————
```

The functionality of the Hamming decoder is described within its behavioral description detailed in the next VHDL code section. There are two processes describing the functionality of the decoder (lines 5 and 13) together with two concurrent assignments (lines 33 and 34). The signal *temp2* is used within the first process to build a register which stores the syndrome $S = [s_1\ s_2\ s_3]$. The values for the syndrome bits s_1, s_2 and s_3 are computed and stored in *temp2(i)* register cells, according to the code lines 8, 9 and 10. The second sequential process, computes the error vector *err* which specifies the position of the erroneous bit, based on the syndrome *S*. Further on, the concurrent assignments corresponding to the code lines 33 and 34 perform the correction of the codeword *v_in* by adding the error vector *err* to the *temp1* register. It is to be mentioned that, the *temp1* registers store the input codeword *v_in*. Consequently, the last assignment (line 34) delivers the corrected codeword to the decoder output *v_out*.

```
1:   architecture Behavioral of decoder is
2:   signal temp1: std_logic_vector(0 to 6);
3:   signal temp2: std_logic_vector(0 to 2);
4:   signal err: std_logic_vector(0 to 6);



4:   begin
5:   process (clk)
6:             begin
7:             if clk'event and clk = '1' then
8:                  temp2(0) <=v_in(3) xor v_in(4) xor v_in(5) xor v_in(6);
9:                  temp2(1) <=v_in(1) xor v_in(2) xor v_in(5) xor v_in(6);
10:                 temp2(2) <=v_in(6) xor v_in(4) xor v_in(2) xor v_in(0);
11:            end if;
12:  end process;
```

```
13:   process (clk)
14:   begin
15:     if ( clk'event and clk = '1') then
16:        if ( reset = '1') then
17:           err <= "0000000";
18:        else
19:            case temp2 is
20:              when "000" => err <= "0000000";
21:              when "001" => err <= "0000001";
22:              when "010" => err <= "0000010";
23:              when "011" => err <= "0000100";
24:              when "100" => err <= "0001000";
25:              when "101" => err <= "0010000";
26:              when "110" => err <= "0100000";
27:              when "111" => err <= "1000000";
28:              when others => err <= "0000000";
29:           end case;
30:        end if;
31:     end if;
32:  end process;
33:  temp1 <= v_in;
34:  v_out <= err XOR temp1;
35:  end Behavioral;
```

The simulation of the Hamming decoder implementation is presented in Fig. 2.6, where three different situations of the decoding operation mode are underlined. At the simulation time t_0, the decoder input got no errors and consequently, the resulted *err* signal is err = '0000000'. At the simulation time t_1, there is a unique error in the *v_in* codeword, which is corrected using the *err* vector err = '0001000' at the simulation time t_2. The last simulation time t_3 corresponds to an input code word *v_in* having two errors; in this case the decoder introduces an additional error to the output signal *v_out*.

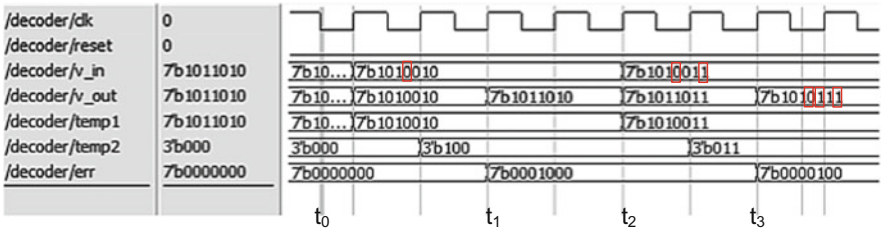

Fig. 2.6 Simulation of the Hamming decoder implementation; the erroneous positions are marked with rectangular areas

2.5 Cyclic Codes Principles

Cyclic codes are a well known subset of the linear block codes commonly used in practice, considering the simplicity of their implementation. The implementation is performed using linear feedback shift registers.

Cyclic codes were first studied by E. Prange in 1957, whereas in 1960 multiple error correction cyclic codes were introduced as BCH codes by R. Rose and P. Chauduri. Moreover, non-binary cyclic codes known as Reed—Solomon codes were proposed in 1960.

Definition
A cyclic code is a linear block code if any cyclic permutation of a codeword is also a codeword. In other words, if a codeword $v = (a_0\ a_1\ \ldots\ a_n)$ is included in the set of codewords C, than any cyclic permutation of *v* (i.e. $v^{(1)} = (a_n\ a_0\ \ldots\ a_{n-1})\ \ldots$ $v^{(i)} = (a_{n-i}\ a_{n-i-1}\ \ldots a_{n-1}\ a_0\ a_1\ \ldots\ a_{n-i+1})$) is still a codeword.

Further on, the polynomial representation of the codes is used for describing the cyclic codes coding and decoding processes. Let $i = (i_0 \ldots i_{m-1})$ be the information bits. Thus, the information polynomial *i(x)* of degree *m-1* detailed in Eq. (2.27) describes the information bits.

$$i(x) = i_0 + i_1 x + \ldots + i_{m-1} x^{m-1} \tag{2.27}$$

The codewords of size *n* are chosen as polynomials multiples of a k = n-m degree polynomial known as the code generator polynomial *g(x)* (see Eq. 2.28).

$$g(x) = g_0 + g_1 x + \ldots + g_k x^k, g_k = g_0 = 1 \tag{2.28}$$

Consequently the coding equation is given by:

$$v(x) = i(x)g(x) \tag{2.29}$$

In the current chapter, we will focus on the binary cyclic codes. For binary codes, considering the *c(x)* polynomial of degree *n*, the set of modulo 2 residue classes of c(x) has 2^n elements, out of which, 2^m elements are considered the codeword set that are multiples of the generator polynomial *g(x)*.

The codeword formed with the relation (2.29) leads to a non-systematic codeword structure. In order for the codeword to have a systematic structure, the next steps are followed as presented in [2].

$$\begin{aligned} x^k i(x) &= i_0 x^k + i_1 x^{k+1} + \ldots + i_{m-1} x^{n-1} \\ \frac{x^k i(x)}{g(x)} &= q(x) + \frac{r(x)}{g(x)} \\ v(x) &= x^k i(x) + r(x) = q(x)g(x) \\ &= a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} + a_k x^k + \ldots + a_{n-1} x^{n-1} \end{aligned} \tag{2.30}$$

Thus, a cyclic codeword multiple of *g(x)* is obtained, with the information bits placed on the first m significant positions whereas the control bits are given by the *r (x)* polynomial. The *r(x)* polynomial represents the reminder after the division of $x^k i(x)$ with *g(x)*. In [2] the reader may find how an equation similar with $Hv^T = 0$ is obtained for the coding process. Considering $h(x) = (x^n + 1)/g(x)$, the H matrix is defined as denoted by Eq. (2.31).

$$\mathbf{H}_{[k\times n]} = \begin{bmatrix} 0 & 0 & \cdots & 0 & h_m & h_{m-1} & \cdots & h_1 & h_0 \\ 0 & 0 & \cdots & h_m & h_{m-1} & h_{m-2} & \cdots & h_0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_m & h_{m-1} & \cdots & h_0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{2.31}$$

To sum up, we present next how the cyclic code size is determined, and how to choose the generator polynomial of degree k in order to correct a number of t errors.

First, the code size is $n = 2^k - 1$, as the cyclic codes are a particular case of linear block codes described in Sect. 3.1.

Secondly, the *g(x)* is chosen as a primitive polynomial of degree *k,* according to the Table 1 from Annex 2. Moreover, a table with generator polynomials for different codeword sizes *n* and errors to be corrected *t* are detailed in Annex 2, Table 2. Further on, digital circuits for cyclic encoding and decoding are presented and implemented using VHDL code.

2.6 Cyclic Codes Encoder and Decoder Implementations

The coding and decoding process for the cyclic codes is performed through the division of $x^k i(x)$ and *r(x)* respectively to the *g(x)* polynomial. *Linear feedback shift registers* with external modulo 2 adders are used for the polynomials division implementation.

The register cells *Cj* connections to the external adders depend on the characteristic of the generator polynomial $g(x) = g_0 + g_1 x + \ldots + g_k x^k, \quad g_k = g_0 = 1$. The block scheme is presented in the Fig. 2.7.

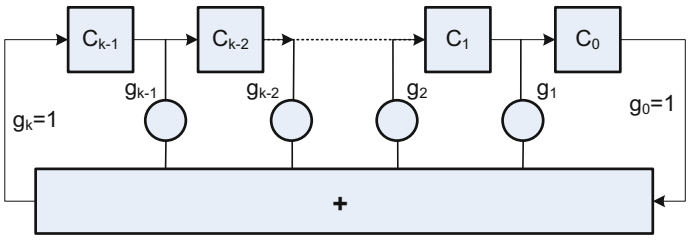

Fig. 2.7 Liner feedback shift registers with external adders for the polynomial division with g(x)

Let the st_iC_j be the state of the register cell C_j at the moment i. Considering a matrix description, the functionality of the linear feedback shift registers is described by Eq. (2.32):

$$\mathbf{S}_i = \mathbf{T}\mathbf{S}_{i-1} \tag{2.32}$$

where

$$\mathbf{S}_i = \begin{bmatrix} st_iC_0 \\ \vdots \\ st_iC_{k-1} \end{bmatrix}, \quad \mathbf{S}_{i-1} = \begin{bmatrix} st_{i-1}C_0 \\ \vdots \\ st_{i-1}C_{k-1} \end{bmatrix}, \text{ and } \quad \mathbf{T} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ g_0 & g_1 & g_2 & \cdots & g_{k-1} \end{bmatrix}. \tag{2.33}$$

The cyclic encoder can be build based on the previously described linear feedback shift register. Thus, another modulo 2 adder S_2 is introduced together with a switch K. This leads to the cyclic encoder from Fig. 2.8.

Concerning the decoder implementation, the information bits $i = [a_{n-1}\ a_{n-2}\ \dots\ a_{n-m}]$ are delivered to the input sequentially during m clock cycle, whereas the switch K is found at position 1. The encoder output during these first m clock cycles is the same as the input. After m clock cycles, the switch K is in position 2 for the next k clock cycles, while the control bits are computed by dividing $x^k i(x)$ to $g(x)$. At the output we can find along the $m + k$ clock cycles the $v(x)$ polynomial corresponding to the codeword associated with the information symbols from the input (i.e. $v(x) = x^k i(x) + r(x)$). Note that at the end of the n clock cycles, the registers cells are all 0. The Eq. (2.34) expressing the register functionality is illustrated next.

$$\mathbf{S}_i = \mathbf{T}\mathbf{S}_{i-1} + a_i\mathbf{U} \tag{2.34}$$

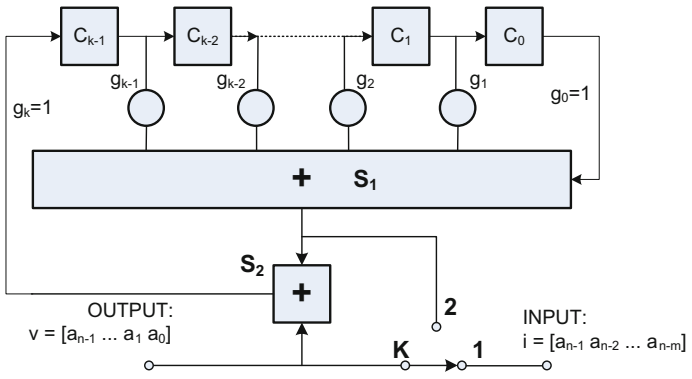

Fig. 2.8 Cyclic encoder with linear feedback shift register and external adders

where a_i is the input signal at the moment i and the U matrix is $U = [0\ 0\ \ldots\ 1]^T$ of size k. Note that during the k clock cycles while the switch K is on position 2, the input is 0 logic. This leads to the value 0 for all the register cells at the end of the n clock cycles (i.e. $S_n = 0$).

The equality $S_n = 0$ and the Eq. (2.31) are used to compute the control symbols based on the information bits (i.e. the encoding equations).

Example 2.3—Hardware architecture for the cyclic encoder implementation We will present how the encoding relation are determined for a cyclic code C(7,4) using the generator polynomial $g(x) = x^3 + x+1$. Further on, the relation will be verified through simulation, using the VHDL implementation of the cyclic encoder having the same size and the same generator polynomial.

The block scheme for the C(7,4) cyclic encoder is detailed in Fig. 2.9.

The encoder operates according to the Table 2.1. Thus, the initial state of the register is $[C_1\ C_2\ C_3] = [0\ 0\ 0]$. For the first $m = 4$ clock cycles, the switch is on position 1, meaning the output is the same as the input (i.e. $i = [a_6\ a_5\ a_4\ a_3]$), whereas the register cells evolve according to the generator polynomial $g(x)$. For the next $k = 3$ clock cycles, the switch is on position 2, meaning the *output* v at

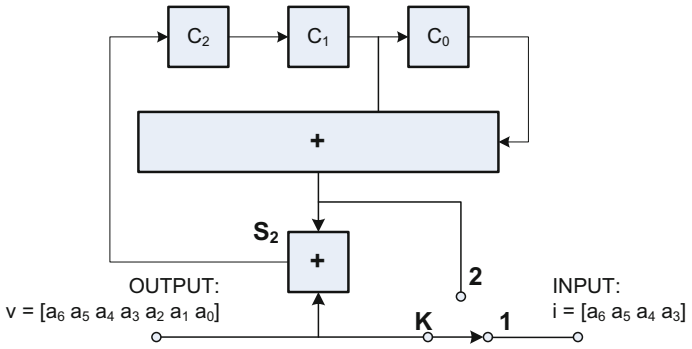

Fig. 2.9 Cyclic encoder for the C(7,4) cyclc code with $g(x) = x^3 + x + 1$

Table 2.1 Hardware resource usage for the implementation of 10 hardware architectures for Canny edge detector aiming parallel microarray spot processing

t_n			t_{n+1}			t_n
T	K	Input i	$C_2^{(0)}$	$C_1^{(0)}$	$C_0^{(0)}$	Output v
1	1	a_6	a_6	0	0	a_6
2		a_5	a_5	a_6	0	a_5
3		a_4	$a_4 + a_6$	a_5	a_6	a_4
4		a_3	$a_3 + a_5 + a_6$	$a_4 + a_6$	a_5	a_3
5	2		0	$a_3 + a_5 + a_6$	$a_4 + a_6$	$a_2 = a_4 + a_5 + a_6$
6			0	0	$a_3 + a_5 + a_6$	$a_2 = a_3 + a_4 + a_5$
7			0	0	0	$a_2 = a_3 + a_5 + a_6$

simulation time t_n is given by the register cells C_1 + C_0 from the simulation time t_n. This leads to the coding equations described by Eq. (2.35).

$$\begin{cases} a_2 = a_4 + a_5 + a_6 \\ a_1 = a_3 + a_4 + a_5 \\ a_0 = a_3 + a_5 + a_6 \end{cases} \tag{2.35}$$

The VHDL code for the cyclic encoder is described next. The input output ports are detailed in the entity description as presented next.

```
------------ Entity declaration------------------------------------------
1:      entity RD1 is
2:      Port (  clk:    in std_logic;
3:              inf:    in std_logic;
4:              reset:  in std_logic;
5:              k_switch: in std_logic;
6:              data:   out std_logic) ;
7:      end RD1;
------------ End of entity declaration-----------------------------------
```

The information bits *i* are sequentially delivered to the *inf* input port, which is connected internally to the first register cell C_2. The *clk* and *reset* inputs represent the clock signal input port and the reset port, respectively. The reset port initializes the register cell C_2, C_1 and C_0 with 0 logic. The *data* output ports delivers sequentially each bit of the v codeword. Note that the size of v is 7 in our example (i.e. C(7, 4) cyclic code). The input port *k_switch,* as its name reveals, represents the switch K. This input port is 1 logic for the first 4 clock cycles, meaning the switch K is on position 1 and the information bits are loaded in the register cells. The next 3 clock cycles the *k_switch* is 0 logic, meaning the input into the shift register is 0 and the control symbols are computed and delivered at the *data* output port.

```
---------- Behavioral description of the cyclic encoder ----------
8:      architecture Behavioral of RD1 is
9:      signal temp:STD_LOGIC_VECTOR(2 downto 0);
10:     signal outt:std_logic;
11:     signal outt2:std_logic;
12:     begin
13:     process (clk)
14:     variable RDin : std_logic;
15:     begin
16:             if k_switch = '1' then
17:                     RDin : = inf;
18:             else
19:                     RDin : = '0';
20:             end if;
```

```
21:    if reset = '1' then temp <= "000";
22:    elsif clk'event and clk = '1' then
23:    temp <= RDin & temp(2 downto 1);
24:    end if;
25:    end process;
26:    outt <=inf;
27:    outt2 <=temp(0) xor temp(1);
28:    data <= outt when k_switch = '1' else
29:       outt2 ;
30:    end Behavioral;
```

The end of the behavioral description of the cyclic encoder

The functionality of the decoder is described by the aforementioned behavioral description. There is a *temp* signal declared, which represent the shift register. The register is instantiated through the sequential process description where register cell values are instantiated on clock event (code line 22 and 23). On *reset* (code line 21), the *temp* register in initialized with 0 logic values. The variable *RDin* represents the input into the first register cell, which is a multiplexed input. In case the *k_swith* is 1, the input into the register is the information bits, whereas the *k_switch* is 0, the input is 0. This multiplexed input is described by the code lines 16 to 20. Outside the process there are 3 other concurrent statements, code lines 26, 27 and 28. These code lines connect the output port either to the input through the *outt* wires in case the *k_switch* is 1, or to the *temp(0)* xor *temp(1)* in case the control bits are computed and the k_switch is 0 logic. Note that *k_switch* 1 logic means the switch K is on position 1 and *k_switch* 0 logic means that the switch K is on position 2, as referred to Fig. 2.9.

Figure 2.10 details the functionality of the cyclic encoder through two examples. Let us consider two information sequences given by the following information bits: $i_1 = [1010]$ and $i_2 = [0001]$ for each of the previously mentioned examples. The information bits i_1 and i_2 are sequentially delivered to the encoder starting with

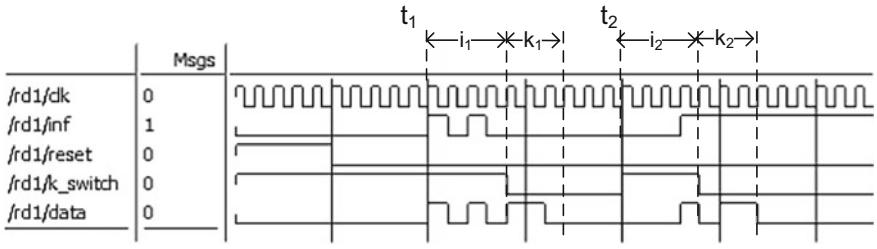

Fig. 2.10 Simulation results for the cyclic encoder for the C(7,4) cyclc code with g (x) = $x^3 + x + 1$

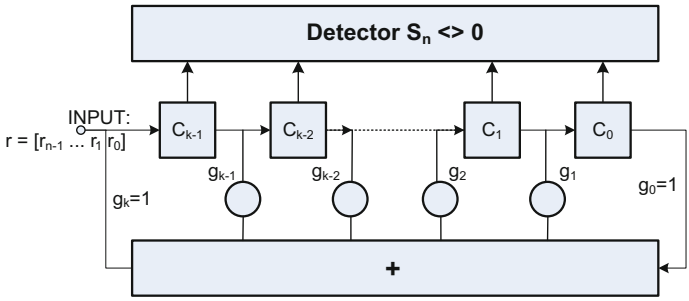

Fig. 2.11 Logic block for the cyclic decoder

simulation time t_1 and t_2 respectively, according to Fig. 2.10. After 4 clock cycles, the number of $k = 3$ control bits are delivered to the output for the next 3 clock cycles. Consequently, the codewords delivered by the encoder at the output corresponding to the two information bits sequences are v1 = [1010110] and v2 = [0001011].

2.6.1 Cyclic Decoder Architectures

Let us consider r to be de received codeword at the decoder side. For error detection, the non-zero state of the shift register shows that an error occurred during information transmission. Consequently, the architecture for the decoder is build based on the encoder, by adding a detector block which specifies if the shift register state S_n is non-zero. This leads to the logic block for the cyclic decoder illustrated in Fig. 2.11. During n clock cycles, the decoder detects based on the register state $S_n = [C_{k-1} \ldots C_0]$ if there are errors during transmission.

In case error correction is desired, based on the S_n syndrome value, the positions of the errors within the received codeword r are computed using supplementary n clock cycles. Thus, during 2n clock cycles the error can be corrected. Detailed architectures for error correction cyclic decoders are presented in [2].

2.7 Conclusions

The current chapter illustrates how channel coding is used for error protection through error detection or correction during data transmission. Basic notions for channel coding such as error control codes representation and parameters are presented. The chapter continues with the description of linear block codes, namely hamming and cyclic codes. Once all the necessary information on error control codes are provided, coder and decoder architectures are presented together with

VHDL codes sequences for their implementation. The functionality of the proposed architectures is shown through simulation. The examples of VHDL code for encoder and decoder implementation represent the starting point for any other implementation of coder/decoder architectures.

References

1. R.G. Gallager, Principles of Digital Communications I, Course materials 2006. MIT OpenCourseWare (http://ocw.mit.edu/), Massachusetts Institute of Technology
2. M. Borda, *Fundamentals in Information Theory and Coding* (Springer, Berlin, 2011)
3. C.E. Shannon, A mathematical theory of communication. Bell Syst. Techn. J. **27**, 379–423 (1948)
4. S. Lin, D. Costello, Error control coding, Prentice-Hall, (1983)
5. G. Cullmann, *Codage et transmission de l'information* (Editions Eyrolles, Paris, 1972)
6. J. Liesen, M. Volker, *Linear Algebra* (Springer, Berlin, 2015)

http://www.springer.com/978-3-319-65023-4

Application-Specific Hardware Architecture Design with VHDL

Belean, B.

2018, XI, 182 p. 75 illus., 8 illus. in color., Hardcover
ISBN: 978-3-319-65023-4