Digital Devices and Techniques

Design of a Programmable Cyclic Encoder

Gavin Cameron

MSc/PGD Networks and Data Communication

June 1, 1999

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ABSTRACT

This report contains the theoretical design of a Programmable Cyclic Encoder.

It covers the algorithms used for two different codes, how they are implemented, how they are decoded and how they are used to corrected for errors.

For the theoretical calculations, modulo-2 polynomial algebra is used, namely multiplication and division

The practical design of the encoder shall be covered in a later report.

INTRODUCTION

Error correcting codes are used to correct errors when messages are transmitted through a noisy communications channel (referred to as the discrete communication channel).

Blocks of data become encoded with many redundant "parity" bits. These bits are usually calculated by a mathematical or logical equation. When these blocks are received, the parity bits are compared with the equation used to calculate the original bits to detect any errors. Errors detected can be corrected.

The encoding schemes used are chosen to detect a certain number of errors. As with all engineering tasks, there is a compromise between the amount of incorrect data that can be corrected and the bandwidth limitations of the transmission channel to allow transmission of the parity data.

HISTORY OF DATA ENCODING

In 1948, Claude Shannon published a paper showing that with any communications channel, a number *C* is associated (measured in bits per second), called the capacity of the channel. Whenever a transmission rate *R* (in bits per second) required of a communications system is less than *C*, it is possible to design a system to use the extra capacity (*C* - *R*) to encode the data such that the probability of output errors is as small as desired.

Shannon did not, however tell us how to find suitable codes for this purpose, rather, he is accredited with proving that they exist.

First take advantage of error correcting codes were Hamming codes. These are block encoding schemes in which a small blocks of data are transformed into another block. This usually means, for example, taking 4 bits and generating a 5 bit code for them. There are now 32 possible symbols, although only 16 of the symbols are valid. This scheme allowed for a single error detection and correction, disappointingly weak compared with the stronger codes promised by Shannon.

A major advance came when Bose and Ray-Chaudhuri (1960) and Hocquenghem (1959) found a large class of multiple error correcting codes (the BCH codes). Reed and Solomon (1960) found a related class of codes for non-binary communication channels.

The discovery of the BCH codes led to a search for practical methods of designing hardware or software for implementing the error encoder and decoder. The first good algorithm was found by Peterson. Later, a powerful algorithm for doing the calculations of Peterson was discovered by Berlekamp and Massey, and its implementation became practical as digital technology became available.

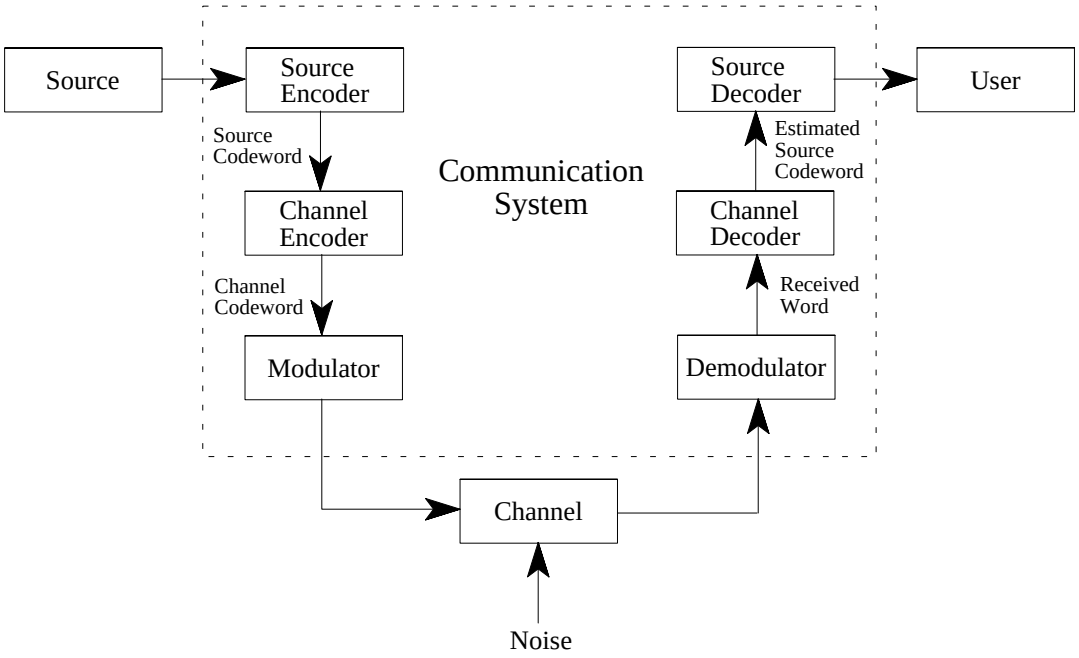
As well as block coding schemes there was also research into sequential encoding / decoding. This required the introduction of a class of non-block codes of indefinite length (which can be represented as a tree). These codes are referred to as convolutional codes. They can be implemented simply by a linear shift register circuit. Decoders were successfully designed in the 1950's for this task, but it wasn't until 1967 when Viterbi developed a much simpler algorithm. This algorithm is only practical for codes of moderate complexity.

THE DISCRETE COMMUNICATION CHANNEL

A communication system connects a source transmitter and a destination receiver (the user) through some form of channel. This channel may take any of a number forms, for example, a co-axial cable, a radio link, a fibre optic cable, or even magnetic storage can be thought of as a channel of communication.

The major functions of a communications system can be illustrated as shown in the diagram below:

Figure 1 - Discrete Communication Channel Block Diagram



Data, from the source, is first processed by the source encoder (for example an A/D convertor) designed to represent the data more compactly. This interim representation is a sequence of symbols called the *source codeword*. Then the data is processed by the channel encoder which transforms the sequence of codeword symbols into another sequence called the *channel codeword*. The channel codeword contains all the redundance required to provide error correction capabilities. The modulator converts each symbol into whichever form of analogue symbol is required for the given channel (for example, light or an electrical signal).

As the symbol is transmitted through the channel, it is subjected to noise which may distort the data with the end result that the data received by the demodulator is not what was transmitted.

The demodulator converts the received symbol back into a channel codeword. Because of the noise in the channel, the demodulated sequence of symbols is referred to as the *received word*. The channel decoder uses the redundancy in the channel codeword to correct the errors in the received word and then produces an *estimated source codeword*. The source decoder performs the inverse operation to the encoder (for example a D/A convertor) and delivers its output to the user.

METHODOLOGY

A block of *k* message bits is encoded to form a code of *n* bits. This requires *n-k* redundancy (or parity) bits to be added to the message.

Let the input polynomial be *P(x)* and the generator polynomial (to generate the redundancy bits) be *G(x)*. The method of encoding a message is as follows:

1. Prescale the input polynomial *P(x)* by x^{n-k} to create *P'(x)*.

   This has the effect of shifting the data *n-k* bits to the left, hence creating space for *n-k* parity bits.

2. Select a generator polynomial *G(x)* of degree *n-k*, e.g. for (15,7) code, use degree 8.

3. Divide the prescaled input polynomial *P'(x)* by the generator polynomial *G(x)*. The remainder *R(x)* contains *n-k* parity bits.

4. Add (modulo-2 addition[1]) the prescaled input polynomial *P'(x)* to the remainder polynomial *R(x)*.

5. The resultant polynomial to be transmitted *T(x)* is now exactly divisible by the generator function *G(x)*. This is the *channel codeword*.

At the receiver, the *received word* is divided by the same generator polynomial *G(x)*. If the remainder *R(x)* is zero, then there are no detectable errors in the *received word*.

However, if the remainder *R(x)* is non-zero, then the *channel codeword* has become corrupted during transit. The remainder *R(x)* is used to determine the bit position that is at fault in the message. The remainder *R(x)* will have the same value as the remainder from dividing x^j by the generator polynomial *G(x)* where *j* is the bit position (0 to 14 for a 15 bit codeword).

The faulty bit is inverted and the *estimated source codeword* is passed out of the decoder.

---

[1] modulo-2 addition has the same function of modulo-2 subtraction. When adding / subtracting polynomials, an XOR function is performed on all terms, i.e. two like terms cancel each other out.

(15,7) BCH CODE

It can be shown (see reference 1) that the BCH generator polynomial for a 15,7 code is as follows:

$$G(x) = x^8 \oplus x^7 \oplus x^6 \oplus x^4 \oplus 1$$

Let the input polynomial *P(x)* be, as an example:

$$P(x) = x^5 \oplus x^4 \oplus x \oplus 1$$

The encoding for the polynomial *P(x)* is as follows:

1. Prescale *P(x)* by x^{n-k} to get *P'(x)*.

$$P'(x) = x^8(x^5 \oplus x^4 \oplus x \oplus 1)$$

$$= x^{13} \oplus x^{12} \oplus x^9 \oplus x^8$$

2. Divide *P'(x)* by *G(x)*.

$$\begin{array}{r|l} & x^5 \oplus x^3 \oplus x^2 \\ \hline x^8 \oplus x^7 \oplus x^6 \oplus x^4 \oplus 1 & x^{13} \oplus x^{12} \oplus x^9 \oplus x^8 \\ & x^{13} \oplus x^{12} \oplus x^{11} \oplus x^9 \oplus x^5 \\ \hline & x^{11} \oplus x^8 \oplus x^5 \\ & x^{11} \oplus x^{10} \oplus x^9 \oplus x^7 \oplus x^3 \\ \hline & x^{10} \oplus x^9 \oplus x^8 \oplus x^7 \oplus x^5 \oplus x^3 \\ & x^{10} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^2 \\ \hline \textit{remainder } R(x) = & x^7 \oplus x^6 \oplus x^5 \oplus x^3 \oplus x^2 \end{array}$$

3. Add *P'(x)* to *R(x)*.

$$T(x) = P'(x) \oplus R(x)$$

$$= x^{13} \oplus x^{12} \oplus x^9 \oplus x^8 \oplus x^7 \oplus x^6 \oplus x^5 \oplus x^3 \oplus x^2$$

This is the polynomial that will be transmitted through the discrete channel to the receiver. If *T(x)* is exactly divisible by *G(x)* when it is received then there are no detectable errors in the *codeword*. This is calculated as follows:

$$
\begin{array}{r|l}
 & x^5 \oplus x^3 \oplus x^2 \\
\hline
x^8 \oplus x^7 \oplus x^6 \oplus x^4 \oplus 1 & x^{13} \oplus x^{12} \oplus x^9 \oplus x^8 \oplus x^7 \oplus x^6 \oplus x^5 \oplus x^3 \oplus x^2 \\
 & x^{13} \oplus x^{12} \oplus x^{11} \oplus x^9 \oplus x^5 \\
\hline
 & x^{11} \oplus x^8 \oplus x^7 \oplus x^6 \oplus x^3 \oplus x^2 \\
 & x^{11} \oplus x^{10} \oplus x^9 \oplus x^7 \oplus x^3 \\
\hline
 & x^{10} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^2 \\
 & x^{10} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^2 \\
\hline
\end{array}
$$

$$remainder\ R(x) = 0$$

If this was the case then the *received* word would be passed out of the decoder as the *estimated source codeword*.

However, as explained earlier, any reminder is the same remainder from dividing x^j by $G(x)$. All possible single error remainders have been calculated and are shown in the table below:

Table 1 - (15,7) Bit Position Error Code Remainders

		Bit Position in Error														
		x^{14}	x^{13}	x^{12}	x^{11}	x^{10}	x^9	x^8	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
R(x)	x^7	1				1		1	1							
	x^6	1	1			1	1	1		1						
	x^5	1	1	1		1	1				1					
	x^4		1	1	1		1	1				1				
	x^3	1		1	1								1			
	x^2		1		1	1								1		
	x^1			1		1									1	
	x^0				1		1	1								1

Hence, if the *codeword* is received with an error in position x^6, then the remainder [from $P'(x)$ / $G(x)$] would be as follows:

$$
\begin{array}{r|l}
 & x^5 \oplus x^3 \oplus x^2 \\
\hline
x^8 \oplus x^7 \oplus x^6 \oplus x^4 \oplus 1 & x^{13} \oplus x^{12} \oplus x^9 \oplus x^8 \oplus x^7 \oplus x^5 \oplus x^3 \oplus x^2 \\
 & x^{13} \oplus x^{12} \oplus x^{11} \oplus x^9 \oplus x^5 \\
\hline
 & x^{11} \oplus x^8 \oplus x^7 \oplus x^3 \oplus x^2 \\
 & x^{11} \oplus x^{10} \oplus x^9 \oplus x^7 \oplus x^3 \\
\hline
 & x^{10} \oplus x^9 \oplus x^8 \oplus x^2 \\
 & x^{10} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^2 \\
\hline
\end{array}
$$

$$remainder\ R(x) = x^6$$

From table 1, it can be seen that a remainder of x^6 indicates an error in position x^6.

Similarly, if an error is induced in position x^{10}, then the remainder of $P'(x) / G(x)$ is:

$$
\begin{array}{r|l}
 & x^5 \oplus x^3 \oplus x \\
\hline
x^8 \oplus x^7 \oplus x^6 \oplus x^4 \oplus 1 & x^{13} \oplus x^{12} \oplus x^{10} \oplus x^9 \oplus x^8 \oplus x^7 \oplus x^6 \oplus x^5 \oplus x^3 \oplus x^2 \\
 & x^{13} \oplus x^{12} \oplus x^{11} \oplus x^9 \oplus x^5 \\
\hline
 & x^{11} \oplus x^{10} \oplus x^8 \oplus x^7 \oplus x^6 \oplus x^3 \oplus x^2 \\
 & x^{11} \oplus x^{10} \oplus x^9 \oplus x^7 \oplus x^3 \\
\hline
 & x^9 \oplus x^8 \oplus x^6 \oplus x^2 \\
 & x^9 \oplus x^8 \oplus x^7 \oplus x^5 \oplus x \\
\hline
\textit{remainder } R(x) = & x^7 \oplus x^6 \oplus x^5 \oplus x^2 \oplus x
\end{array}
$$

Again, from table 1, it can be seen that the remainder matches the remainder of $x^{10} / G(x)$.

On both occasions, the offending bit position would be inverted and passed out of the decoder as the *estimated source codeword*.

It should be noted that multiple errors (3 or more) could cause the same remainder which would cause a bad estimate as the (15,7) code is only designed to correct 2 errors.

(15,5) BCH CODE

It can be shown (see reference 1) that the BCH generator polynomial for a 15,5 code is as follows:

$$G(x) = x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1$$

Let the input polynomial *P(x)* be, as an example:

$$P(x) = x^4 \oplus x^2 \oplus 1$$

The encoding for the polynomial *P(x)* is as follows:

1. Prescale *P(x)* by x^{n-k} to get *P'(x)*.

$$P'(x) = x^{10}\left(x^4 \oplus x^2 \oplus 1\right)$$

$$= x^{14} \oplus x^{12} \oplus x^{10}$$

2. Divide *P'(x)* by *G(x)*.

$$\begin{array}{r|l} & x^4 \oplus 1 \\ \hline x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1 & x^{14} \oplus x^{12} \oplus x^{10} \\ & x^{14} \oplus x^{12} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^5 \oplus x^4 \\ \hline & x^{10} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^5 \oplus x^4 \\ & x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1 \\ \hline \textit{remainder } R(x) = & x^9 \oplus x^6 \oplus x^2 \oplus x \oplus 1 \end{array}$$

3. Add *P'(x)* to *R(x)*.

$$T(x) = P'(x) \oplus R(x)$$

$$= x^{14} \oplus x^{12} \oplus x^{10} \oplus x^9 \oplus x^6 \oplus x^2 \oplus x \oplus 1$$

If this *codeword* is received correctly then the remainder of *P'(x)* / *G(x)* should equal zero as follows:

$$
\begin{array}{r|l}
 & x^4 \oplus 1 \\
\hline
x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1 & x^{14} \oplus x^{12} \oplus x^{10} \oplus x^9 \oplus x^6 \oplus x^2 \oplus x \oplus 1 \\
 & x^{14} \oplus x^{12} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^5 \oplus x^4 \\
\hline
 & x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1 \\
 & x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1 \\
\hline
\end{array}
$$

$$remainder\ R(x) = 0$$

The bit position error code reminders can be calculated the same way as before from dividing x^j by $G(x)$. All possible single error remainders have been calculated and are shown in the table below:

Table 2 - (15,5) Bit Position Error Code Remainders

		Bit Position in Error														
		x^{14}	x^{13}	x^{12}	x^{11}	x^{10}	x^9	x^8	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
R(x)	x^9	1	1		1		1									
	x^8		1	1		1		1								
	x^7	1	1	1					1							
	x^6		1	1	1					1						
	x^5			1	1	1					1					
	x^4	1	1			1						1				
	x^3	1		1	1								1			
	x^2		1		1	1								1		
	x^1	1	1	1	1	1									1	
	x^0	1		1		1										1

If an error is induced in bit x^{12} then the remainder of $P'(x)$ / $G(x)$ is as follows:

$$
\begin{array}{r|l}
 & x^4 \oplus x^2 \\
\hline
x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1 & x^{14} \oplus x^{10} \oplus x^9 \oplus x^6 \oplus x^2 \oplus x \oplus 1 \\
 & x^{14} \oplus x^{12} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^5 \oplus x^4 \\
\hline
 & x^{12} \oplus x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1 \\
 & x^{12} \oplus x^{10} \oplus x^7 \oplus x^6 \oplus x^4 \oplus x^3 \oplus x^2 \\
\hline
\end{array}
$$

$$remainder\ R(x) = x^8 \oplus x^7 \oplus x^6 \oplus x^5 \oplus x^3 \oplus x \oplus 1$$

From table 2, it can be seen that the remainder $R(x)$ is the same remainder as for bit x^{12}.

The (15,5) BCH code is designed to detect up to 3 bits in error. Any more than this and the data will be estimated incorrectly.

IMPLEMENTATION OF CODES

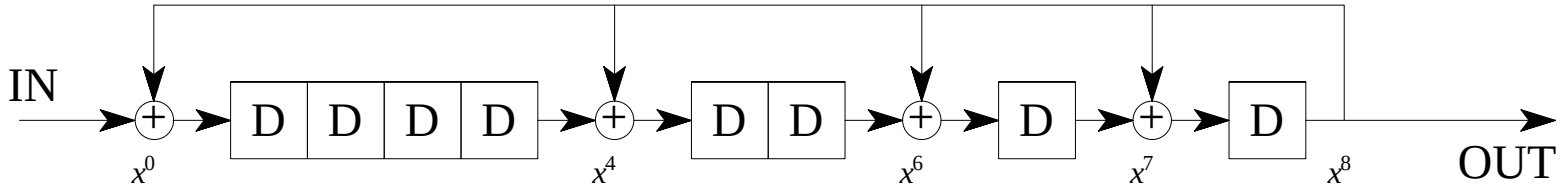
This type of error encoder is easily constructed from D-type registers and XOR gates. The circuit used to divide polynomials is a negative feedback linear shift register.

From before, for the (15,7) code,

$$G(x) = x^8 \oplus x^7 \oplus x^6 \oplus x^4 \oplus 1$$

This can be implemented as follows:

Figure 2 - (15,7) Code Circuit

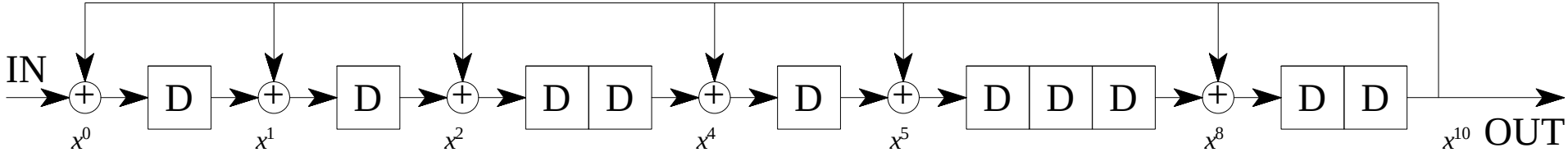


Similarly, the (15,5) code,

$$G(x) = x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus x \oplus 1$$

can be implemented as follows:

Figure 3 - (15,5) Code Circuit

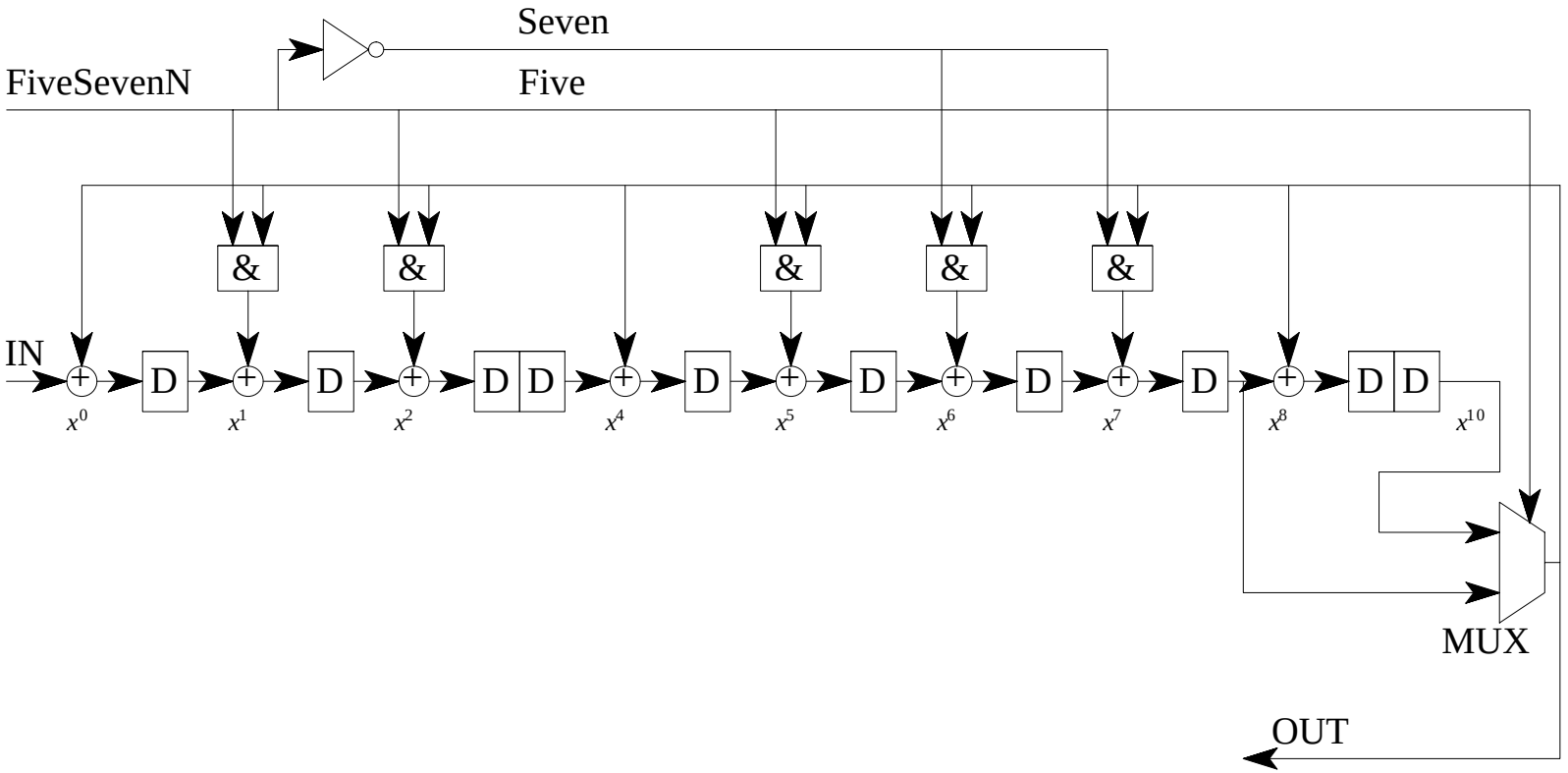


By combining these two circuits to create a programmable encoder, an extra signal shall be required called FiveSevenN. If this signal is low then the encoder will be in (15,7) mode, and high will put it into (15,5) mode.

AND gates can be used along with control signals (from FiveSevenN) to control the feedback paths such that only the XOR gates that require feedback for each mode are enabled.

The circuit then becomes as follows:

Figure 4 - Programmable (15,5) / (15,7) Code Circuit



The output / feedback path also differs for each mode, hence a multiplexor selects which output to use. The MUX is controlled by FiveSevenN.

What is not shown in the circuit is a method of outputting the *k* data bits and then switching over to output the *n-k* parity bits. That shall be dealt with in the second part of the assignment.

It is proposed to have a 4 bit counter with a programmable output which is low for the first 5 bits of a (15,5) code and then high for the rest. Similarly, low for the first 7 bits of a (15,7) code and high for the rest. This signal will control a multiplexor to allow the first *n* bits of data straight through and then allow the *n-k* bits of parity from the CRC generator through.

TIMING AND TESTING

If this circuit was designed using discrete components, care would have to be taken to ensure that the circuit would function at it's required clock frequency. This would be done by taking into account the propagation delays through all gates and registers. The longest path (as far as number of devices is concerned) is either x^8 or x^{10}, through the multiplexor, through the AND gates and through the XOR gates.

This time would be compared to the setup and hold times of the registers, i.e. the longest path must reach the register before the clock reaches the register (setup) and it must be static for a length of time after the clock (hold). If both these criteria are met, then the circuit will function.

The actual propagation delays, and hence maximum clock frequency are determined by the logic family used in the design.

There are tools available to analyse the circuit and provide the maximum operating frequency, giving it the requirements, otherwise a manual calculation is required.

Another alternative, is to "fit" the circuit into a programmable logic device (PLD). This will increase the maximum operating frequency (generally) as delays between gates will be very small compared to traversing over a PCB. Again, most modern PLD tools can calculate the timing requirements of the circuit.

The circuit can be tested before manufacture by using a simulation package. This involves designing the circuit on a schematic capture tool (or VHDL editor), stimulating the input and monitoring the output using the simulator. ECAD simulators can save time and money for complicated designs. Generalized figures state that if it costs *£n* to fix a design fault at design stage, it will cost *£10n* at PCB stage and *£100n* if a fault is found with the customer.

When the circuit is manufactured, there are a couple of way of testing it.

The best is to have designed the circuit with testing in mind. In that case any devices used would have a JTAG interface on them. JTAG (or Joint Test Access Group) interfaces are an industry standard method for accessing internal nodes of a device. They allow the user to load and read values from the pins of devices using *boundary scan chains*[2]. This can be used to test internal functionality, or inter-device connections. This method of testing can be achieved with the addition of only 4 extra pins (for the JTAG interface).

The alternative, is to add monitor points at various points in the circuit. However, unless extra test logic is added, the circuit can only be stimulated by supplying a test input to the actual data input. Since this is a sequential circuit, the probability is that a large number of test vectors would have to be supplied in order to fully test the circuit.

---

[2]boundary scan chain - all I/O pins on a device have a boundary scan cell which allows data to be loaded in, read out or sampled from either side of the cell, these are chained together. The operation of such is controlled via the JTAG command interface.

CONCLUSION

Cyclic Redundancy Checking is used on almost all forms of communication, from mobile phones to computer networking.

As can be seen, it can be a very powerful and quick method for detecting and correcting codes. The larger the ratio of n / k, the more bits can be corrected, i.e. a (15,7) code can detect up to 2 errors, a (15,5) code can detect up to 3 errors and a (15,1) code can detect up to 7 errors.

There is, of course, a trade off between speed, reliability and bandwidth of a communication system.

This type of encoding scheme is very simple to implement in hardware (or software) as it just involves registers (or delay elements) and XOR functions. However, to allow this circuit to be programmable, extra components are required - AND gates and multiplexors.

This circuit will be entered into a Mentor Graphics system and simulated to verify it’s operation.

REFERENCES

1. Theory and Practice of Error Control Codes, Richard E. Blahut, Addison Wesley Publishing Company, 1984.