VLSI IMPLEMENTATION OF A NEW STANDARD FOR LOSSLESS COMPRESSION OF CONTINUOUS-TONE STILL IMAGES

by

A.Suat Aktürk

B.S. in E.E. Naval Academy, 1992

Submitted to the Institute for Graduate Studies in Science and Engineering in partial fulfillment of the requirements for the degree of Master of Science in Defence Technology Program

Boğaziçi University

1998

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to Turkish Naval Forces and Prof. Dr. Yorgo İstefanopulos for starting such a program, thus allowing me to have an educational and social opportunity in Boğaziçi University.

Then, I would like to thank my thesis supervisor Assoc. Prof. Dr. Sina Balkır and Assoc. Prof. Dr. Günhan Dündar for their continuous support during the thesis study. I would also like to thank to Prof. Dr. Avni Morgül and Assoc. Prof. Dr. Lale Akarun for their suggestions and participations in my thesis jury.

My special thanks are to the Electrical Engineering Department for all the facilities made available to me. In addition, I am grateful to the people in the BETA laboratory, particularly, to our leader Prof. Dr. Ömer Cerid and dear friends Selçuk Öğrenci, Faik Başkaya for their patience and support.

I would like to thank my friends, Nilüfen Çotuk, Levent Bektaş and Ümit Şirin for their support and Mustafa Sözer for his always being more than a friend.

Finally, I would like to appreciate my family for their invaluable, persistent support for all my life.

ABSTRACT

In this thesis, an integrated circuit realizing lossless compression of continuous-tone still images is designed.

Lossless compression has two main parts. In the encoding part, digital source image data are encoded in order to generate compressed image data. In the decoding part, this compressed image data are decompressed and digital reconstructed image data are obtained which must be identical to the digital source image data.

First, the circuit was created by drawing its schematic diagram, then it was designed, optimized, simulated and its IC layout was created and prepared for IC processing by using the Mentor Graphics v.8.4.1 tools running on a Sun Sparc2 workstation. In the design, Europractice Alcatel Mietec 0.7 μm CMOS target technology is used.

Then, by putting a select switch into the circuit, the mode of coding process, whether working as an encoder or decoder, can be chosen.

ÖZET

Bu tez çalışmasında, devamlı tondaki durağan görüntülerin kayıpsız sıkıştırılmasını gerçekleyecek bir tümleşik devre tasarlanmıştır.

Kayıpsız sıkıştırma iki ana kısımdan oluşur. Kodlama bölümünde, sayısal kaynak görüntü bilgileri, sıkıştırılmış görüntü bilgilerini oluşturabilmek için kodlanir. Kod çözücü bölümünde, sıkıştırılmış olan bu bilgiler açılır ve sayısal kaynak görüntü bilgileri ile aynı olması gereken sayısal yeniden oluşturulmuş görüntü bilgileri elde edilir.

İlk olarak, şematik görüntü çizilerek devre yaratılmıştır. Daha sonra, tasarımın yazılması, eniyilenmesi, benzetiminin yapılması ve tümleşik devre seriminin çıkarılması için Sun Sparc2 iş istasyonu üzerinde çalışan Mentor Graphics yazılım paketi, sürüm 8.4.1'den yararlanılmıştır. Ayrıca, tasarımda Europractice Alcatel Mietec 0.7 μm CMOS teknolojisi kullanılmıştır.

Daha sonra, devreye bir seçici anahtar yerleştirilerek, devrenin kodlama ya da kod çözücü olarak çalışması seçilebilir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a,b,c,d,e	Causal neighborhood for current sample
A[1..1092]	1092 counters for accumulated prediction error magnitude
B[1..1092]	1092 auxiliary counters for bias cancellation
C[1..1092]	1092 counters indicating bias correction value
COLUMNS	Number of columns in the image
D1,D2,D3	Most significant context gradients
D4	Least significant context gradient
EA	Auxiliary variable for Golomb coding in end of run
EA[0..1]	2 counters for accumulated prediction error for end of run
EM(error)	Error mapping at end of run mode
EN[0..1]	2 occurrence counters for end of run
ENn[0..1]	2 counters for negative prediction error for end of run
EORtype	Index for end of run coding context-type
error	Prediction error
J[0..31]	32 variables indicating order of run-length codes
k	Golomb coding parameter for regular mode
M(error)	Function that maps error to non-negative integers
map	Auxiliary variable for error mapping in end of run
Max	Maximal possible image value
MAX_C	Maximal allowed value of C[1092]
MIN_C	Minimal allowed value of C[1092]
MODULO	Range of prediction error representation
N[1..1092]	1092 counters for context-type occurrence
near	Difference bound for near-lossless mode
P	Probability distribution

Px	Predicted value for the sample x
Q	Context-type determined from Q1, Q2, Q3, Q4
Q1,Q2,Q3,Q4	Regions of quantized context gradients
Qi	One of the four quantized regions Q1, Q2, Q3, Q4
Ra,Rb,Rc,Rd,Re	Reconstructed values of samples in causal neighborhood
RESET	Interval, less than 256, at which A, B and N are halved
rk	Rm is the rk-th power of 2
rm	Golomb code order for run mode, a power of 2
ROWS	Number of rows in the image
RUNcnt	Repetitive sample count for run mode
RUNindex	Index for run mode order
Rx	Reconstructed value of current sample
sign	Sign associated with the context-type
T1,T2,T3,T4	Thresholds for context gradients
x	Current sample position

1. INTRODUCTION

1.1. Compression Techniques

Compression is used in image transmission and storage systems to reduce the number of bits to be transmitted or stored. Such a system includes two blocks: there is the compression algorithm that takes an input *X* and generates a representation *Xc* that requires fewer bits and there is a reconstruction algorithm that operates on the compressed representation *Xc* to generate the reconstruction *Y*. These operations are shown schematically in Figure 1.1.

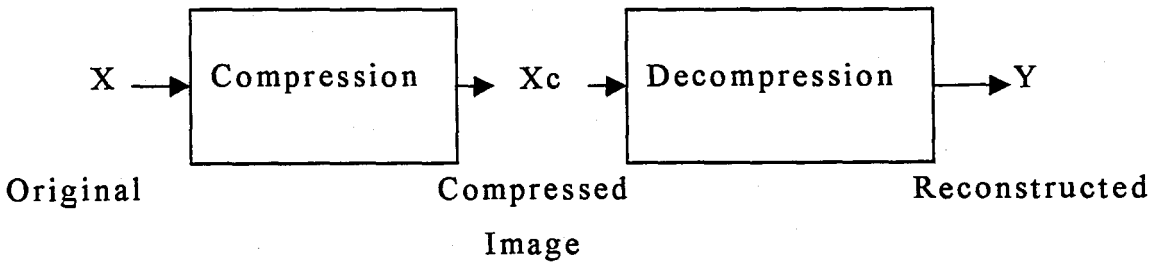

FIGURE 1.1. Compression and reconstruction.

Based on the requirements of reconstruction, data compression schemes can be divided into two broad classes: *lossless* compression schemes, in which *Y* is identical to *X* and *near-lossless* compression schemes, which generally provide much higher compression than lossless compression but allow *Y* to be different from *X* [1].

1.1.1. Lossless Compression

Lossless compression techniques involve no loss of information [2]. If the compression is losssless, the original data can be recovered exactly from the compressed data.

Lossless compression is generally used for "discrete" data, such as text, computer-generated data and some kinds of image and video information [3].

There are many situations that require compression where we want the reconstruction to be identical to the original. There are also a number of situations in which it is possible to relax this requirement in order to get more compression. In these situations, near-lossless compression techniques are used.

1.1.2. Near-lossless Compression

Near-lossless compression techniques involve some loss of information and data that have been compressed using near-lossless techniques generally cannot be recovered or reconstructed exactly. In return for accepting this distortion in the reconstruction, generally much higher compression ratios are obtained than is possible with lossless compression.

In many applications, this lack of exact reconstruction is not a problem. For example, when storing or transmitting speech, the exact value of each sample of speech is not necessary. Depending on the quality required of the reconstructed speech, varying amounts of loss of information about the value of each sample can be tolerated.

1.2. Measures of Performance

After developing a data compression scheme, its performance has to be measured. A compression algorithm can be evaluated in a number of different ways. It is possible to measure the relative complexity of the algorithm, the memory required to implement the algorithm, how fast the algorithm performs on a given machine.

Also it is important to know the amount of compression and how closely the reconstruction resembles the original.

A very logical way of measuring how well a compression algorithm compresses a given set of data is to look at the ratio of the number of bits required to represent the data before compression to the number of bits required to represent the data after compression [4]. This ratio is called the *compression ratio*. Suppose storing an image made up of a square array of 256 * 256 pixels requires 65 536 bytes. The image is compressed and the compressed version requires 16 384 bytes. So, the compression ratio is 4:1.

Another way of reporting compression performance is to provide the average number of bits required to represent a single sample. This is referred to as the *rate*. For example, in the case of the compressed image described above, if eight bits per byte (or pixel) is assumed, the average number of bits per pixel in the compressed representation is two. Thus, the rate is two bits per pixel.

When the quality of a reconstruction is high, the difference between the reconstruction and the original is small. That is, the variable *near* (difference between the reconstructed and original image data sample value) is small in near-lossless mode.

1.3. Still Image Compression Standards

Recent developments in electronic imaging technology are creating both a capability and a need for digital storage and transmission of binary and continuous-tone color images and video. Image compression is an essential enabling technology behind these developments, considering the raw data rates associated with digital images.

However, storage and transmission of digital images in compressed formats necessitates internationally accepted standards to ensure compatibility and interoperability of various products from different manufacturers.

Recognizing this need, several international organizations, including the International Standards Organization (ISO), the International Telecommunications Union (ITU) – formerly the International Consultative Committee for Telephone and Telegraph (CCITT) – and the International Electrotechnical Commission (IEC) have participated in programs to establish international standards for image compression in specific application areas. To this effect, expert groups, formed under the auspices of ISO, IEC and ITU (CCITT), solicited proposals from companies, universities and research laboratories. The best of those submitted have been selected on the basis of image quality, compression performance and practical constraints [5].

The international standards on bilevel and continuous-tone color still image compression are:

1. ITU (CCITT) : It is developed for bilevel image compression, primarily designed for fax transmission of documents.

2. JBIG (ITU-ISO / IEC) Bilevel Imaging Group: It is developed for improved bilevel image compression to handle progressive transmission and digital halftones.

3. JPEG (ITU-ISO / IEC) Photographic Expert Group: It is developed for color still image compression.

1.3.1. The JPEG Standard

The JPEG standard describes a family of image compression techniques for continuous-tone (gray-scale or color) still images. Because of the amount of data involved and the psychovisual redundancy in the images, JPEG employs a lossy compression scheme based on transform coding.

JPEG provides four modes of operation, sequential (baseline), progressive, hierarchical and lossless.

1. Baseline Algorithm: The image is first subdivided into 8 * 8 blocks. For 8-bit images, 128 is subtracted from each pixel. The coefficients are threshold coded using zigzag scanning to form a one-dimensional sequence of quantized coefficients.

2. JPEG Progressive: The progressive mode refers to encoding of the coefficients in multiple passes, where a portion of the quantized coefficients is transmitted in each pass. Two complementary procedures, corresponding to different grouping of the coefficients have been defined for progressive transmission; the spectral selection, where the coefficients are ordered into spectral bands and the lower-frequency bands are encoded and sent first, successive approximation, where the coefficients are first sent with lower precision and then refined in successive passes.

3. JPEG Hierarchical: The hierarchical mode of operation may be considered as a special case of the progressive transmission, with increasing spatial resolution between the progressive stages. The image quality at extremely low bit rates surpasses any of the other JPEG modes, but this is achieved at the expense of a slightly higher bit rate at the completion of the progression.

4. JPEG Lossless: Lossless image compression can be formulated as an inductive inference problem, in which an image is observed sample after sample in a pre-defined scan [1]. When coding the current sample, after having scanned past data, inferences can be made on the value of this sample by assigning a conditional probability distribution.

In a formulation where the image is scanned only once, this distribution is learned only from preceding samples. The decoder can reconstruct the conditional probability used to encode the current symbols, since it depends only on the already decoded data. The basic idea in lossless compression is then to assign shorter codes to more probable events.

1.3.2. Coding Process

The International Standard (ISO/IEC WD14495:1997(E)) [6], that is implemented, deals with lossless and near-lossless image compression of continuous tone still images. In lossless compression, the input digital source image data is identical to the digital reconstructed image data. In near-lossless compression, a pre-specified difference between the source and reconstructed image values is allowed per image sample.

The algorithms specified in this International Standard offer higher compression in lossless mode, as well as an additional near-lossless mode. Coding process is shown in Figure 1.2.

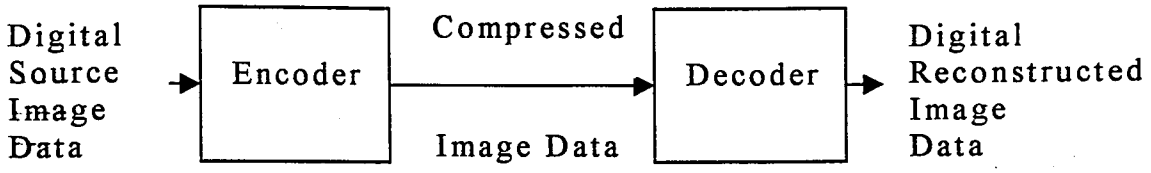

FIGURE 1.2. Coding process flow.

2. ENCODING PROCESSES

In this section, the encoding process of the International Standard [6] will be shown. The main procedures for the lossless (and near-lossless) encoding process are given in Figure 2.1.

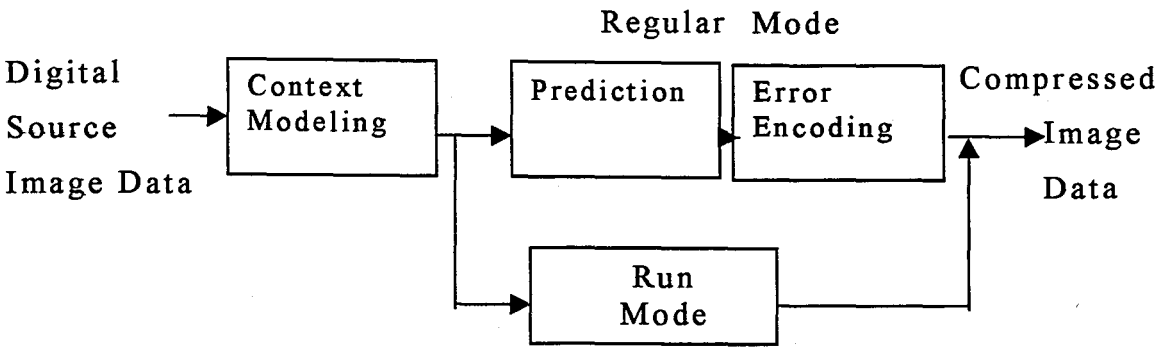

FIGURE 2.1. Lossless encoder simplified diagram.

In the regular mode, the context-type determination process is followed by a prediction process. The predictor combines the reconstructed values of the three neighborhood samples at positions a, b and c to form a prediction of the sample at position x as shown in Figure 2.2. The difference between the actual value at position x and the predicted one (the predicted error), corrected by a context-dependent bias cancellation term and quantized in the case of near-lossless coding, is the value to be encoded.

	c	a	d
e	b	x	

FIGURE 2.2. Current sample and its causal neighborhood for the coding process.

This corrected prediction error is encoded using a process derived from Golomb codes. These codes assume a geometric distribution and are a particular case of Huffmann codes. The parameters of the Golomb codes in this specification depend on the context-type defined by the values of the samples at positions a, b, c, d and e, as well as on prediction errors previously encoded on the same context-type.

If, based on the samples a, b, c and d, the context modeling unit detected a run mode, the encoding procedure skips the predictor and encoding of prediction error steps.

In this mode of operation, the encoder looks, starting at x, for a sequence of consecutive samples with value identical to the sample at b.

This run is ended by a sample of different value or by the end of the current row, whichever comes first. Its run-length is encoded using a procedure based on Golomb coding.

2.1. Default Parameters

The encoding of the sample x is based on its causal template composed by the samples a, b, c, d and e as shown in Figure 2.2.

The default parameters that are used in the applications are shown in Table 2.1.

TABLE 2.1. Default parameters.

Max	255
RESET	64
Near	0
T1	3 + 3 * near
T2	7 + 5 * near
T3	21 + 7 * near
T4	Max {5, near + 1}
MIN_C	- 128
MAX_C	127

If Max is not 255, that is, the image is not represented by 8 bits per symbol, then the default parameters T1, T2, T3 and T4 for the lossless mode are given by equation (2.1), where *bpp* stands for bits per pixel and the values on the right side of the equation refer to the default parameters for 8 bpp.

$$\begin{aligned} T1 &= 2^{bpp-8}(T1-2) + 2 \\ T2 &= 2^{bpp-8}(T2-3) + 3 \\ T3 &= 2^{bpp-8}(T3-4) + 4 \\ T4 &= \max\{5, 2^{bpp-9}\} \end{aligned} \tag{2.1}$$

If the coding is operating in the near-lossless mode, these parameters are further modified as in the case of 8 bits per symbol images described above.

After all the parameters are specified, they are written into the bit stream in the order specified in Table 2.2., from top to bottom and using the number of bits specified in the table.

TABLE 2.2. Order and size of parameters.

ROWS	16 bits
COLUMNS	16 bits
Max	8 bits
T1	8 bits
T2	8 bits
T3	8 bits
T4	8 bits
near	8 bits
RESET	8 bits
MAX_C	8 bits
MIN_C	8 bits

2.2. Initializations and Conventions

As a general convention, when encoding the first row, the values at the samples a, c and d are assumed to be zero [7]. When the samples b, d and e are not defined, their values are assumed to be equal to Ra, the value at the sample a.

A number of initializations should be carried out before starting the encoding process:

a. The default values of the encoding parameters are assigned and written into the bit stream following the order in Table 2.2.

b. For each one of the different context-types(total of 1092), four variables are initialized: N[1..1092], A[1..1092], B[1..1092] and C[1..1092], where the symbol [1..1092] indicates that there are 1092 copies of the variable. Each one of the entries of A is initialized with the value 4, those of N are initialized with the value 1 and those of B and C with the value 0.

c. The variables are allocated and initialized for the run mode: RUNindex=0, RUNcnt=0 and J[32] = {0,0,0,0,1,1,1,1, 2,2,2,2,3,3,3,3,4,4,5,5,6,6,7,7,8,9,10,11,12,13,14,15}.

d. End of run variables are allocated and initialized(2 of each): EA[0..1] = 4, EN[0..1] = 1 and ENn[0..1] = 0.

e. The variable MODULO is computed. If the lossless mode of operation is required, then MODULO = Max + 1. If the near-lossless mode of operation is required, then:

$$MODULO = ((Max + (2*near)) / ((2*near) + 1) + 1 \qquad (2.2)$$

2.3. Context-type Determination

After a number of samples have been processed in a sequential scan from left to right and from top to bottom, the sample x positioned as in Figure 2.2. should be encoded. The context-type at this sample is determined by the values Ra, Rb, Rc, Rd and Re corresponding to the samples a, b, c, d and e shown in Figure 2.2., respectively.

2.3.1. Gradient Computation

The first step in the context determination process is to compute the local gradient values of the neigborhood samples. This is done using equation (2.3).

These differences represent the local gradients which are used to estimate the statistical behavior of prediction errors, that are the values to be encoded.

$$D1 = Rd - Ra$$

$$D2 = Ra - Rc \qquad (2.3)$$

$$D3 = Rc - Rb$$

$$D4 = Rb - Re$$

Gradient computation step is shown in Figure A.1. The neighborhood samples Rd, Ra, Rc and Re are used to obtain local gradients D1, D2, D3 and D4. 8-bit subtractor is used for the subtraction process and is shown in Figure A.2. The parameter, m is used for determining the type of the operation. If m is equal to 1, the process performs subtraction, else addition of two 8-bit numbers.

2.3.2. Run Mode Check

If the most significant local gradients are all zero or their values are less than or equal to the allowed error for near-lossless coding, the encoder enters the run mode. The run mode detection process is specified in equation (2.4). In the case of lossless coding, this run mode detection process is equivalent to the mode shown in equation (2.5), where the coder is checking if Ra=Rb=Rc=Rd.

If (abs(D1) $\leq$ near and abs(D2) $\leq$ near and abs(D3) $\leq$ near) then

go into run mode else continue (2.4)

If (D1 = D2 = D3 = 0) then

go into run mode else continue (2.5)

In order to detect whether the coder enters the run mode or not, three comparators are used as shown in Figure A.3.

D1, D2 and D3 are the inputs of the comparators (each has 8-bit) and each comparator generates three outputs. The symbols (out1) and (out2) are binary output variables that are equal to one, when A<B or A>B, respectively. The binary variable (out3) is equal to one, if all pairs of digits of the two numbers are equal. The parameter, c1 is used for setting the inputs to zero. A comparator that compares two 8-bit numbers and generates the outputs, just described, is shown in Figure A.4.

If all the gradients are equal to zero, the output *run* will be one, else zero.

2.3.3. Gradient Quantization

The context merging process is based on quantizing the local gradients defined above. Since the probability of a local gradient taking the value v is assumed to be the same as the probability of its taking the value $-v$ and the local characteristics of the image for both cases can be considered to be the same, one of the quantized regions should be centered at the difference value 0 [8].

Based on this principle, a further reduction in the number of context-types is obtained by merging quantized contexts of opposite signs, meaning that the context-type obtained from (Q1, Q2, Q3, Q4) is merged with the context-type obtained from (-Q1, -Q2, -Q3, -Q4), where Qi, i=1, 2, 3, 4, stands for the quantized regions. This last merging process is compensated by changing the sign of the prediction error.

In the case of a regular mode, the context-type determination process continues and quantizes D1, D2, D3 according to the process specified in equation (2.6). For this, the non-negative thresholds T1, T2 and T3 are used. In equation (2.6), the entry Di to the process is one of the values D1, D2 or D3 from the local gradients computation step. According to their relation with the thresholds, a region number Qi is obtained (Q1, Q2 and Q3 respectively).

$$
\begin{aligned}
&\text{if } (Di \leq -T3) && \text{then } Qi = -4 \\
&\text{else if } (Di \leq -T2) && \text{then } Qi = -3 \\
&\text{else if } (Di \leq -T1) && \text{then } Qi = -2 \\
&\text{else if } (Di < -near) && \text{then } Qi = -1 \\
&\text{else if } (Di \leq near) && \text{then } Qi = 0 \\
&\text{else if } (Di < T1) && \text{then } Qi = 1 \\
&\text{else if } (Di < T2) && \text{then } Qi = 2 \\
&\text{else if } (Di < T3) && \text{then } Qi = 3
\end{aligned}
\qquad (2.6)
$$

$$\text{else} \quad Qi = 4$$

The process also quantizes D4 according to equation (2.7) using the non-negative threshold T4, to obtain Q4.

$$\begin{aligned} &\text{if } (D4 \leq -T4) \quad \text{then } Q4 = -1 \\ &\text{else if } (D4 < T4) \quad \text{then } Q4 = 0 \\ &\text{else} \quad Q4 = 1 \end{aligned} \tag{2.7}$$

This forms a vector (Q1, Q2, Q3, Q4) representing the context-type for the sample x. Since there are 9 quantization regions for each one of the most significant gradients and 3 for the least significant one, Q1, Q2 and Q3, all receive one of nine possible numbers between –4 and 4, while Q4 can have a value between –1 and 1. Then, a total of 9*9*9*3–3 = 2184 possible context-types are defined so far, where the 3 context-types corresponding to the run mode were subtracted.

2.3.4 Context-type Merging

If the first non-zero component of the vector (Q1, Q2, Q3) is negative, then all the signs of the vector (Q1, Q2, Q3, Q4) are reversed to obtain (-Q1, -Q2, -Q3, -Q4). In this case, the variable *sign* gets the value *negative*, otherwise, *positive*.

This can be achieved by checking the local gradients D1, D2 and D3, whether they are greater than zero or not and is shown in Figure A.5. If the first non-zero component is greater than zero, than the varibale *sign+* will be one, else zero.

This vector is mapped into an integer number Q representing the context-type for the sample x, after this possible merging.

Since the number of possible context-types is halved after this merging process, Q receives a number between 1 and 1092.

2.4. Prediction

In the context modeling process, the local gradients are quantized. In order to partially compensate for this information loss, the context determination process is followed by a prediction step. The idea behind this process is that the value at the current sample x can be estimated from the reconstructed values of the samples surrounding it. Then, instead of coding the value itself, the prediction error is encoded.

The prediction unit is based on the subset a, b and c of the causal template depicted in Figure 2.2, where x denotes the current sample to be encoded.

2.4.1. Edge-detecting Predictor

The approach consists first on performing a simple test to detect vertical or horizontal edges. If an edge is detected, then the predicted value Px, at the sample x, is Ra+Rb-Rc, as this would be the value at x if a plane is passed through the a, b and c sample locations, with respect to heights Ra, Rb, Rc and the constraint is imposed that the current sample belongs to the same plane.

This constraint expresses the expected smoothness of the image in the absence of edges [9].

If a vertical edge is detected, the value at a, Ra is predicted. If a horizontal edge is detected, the value at b, Rb is predicted. This process is shown in equation (2.8).

Three neighborhood sample values Ra, Rb and Rc are used to predict the current sample value and the process is shown in Figure A.6. The addition operation is performed by using 8-bit CLA(Carry look-ahead) adder and is shown in Figure A.7.

$$Px = \begin{cases} \min(Ra, Rb) & \text{if } Rc \geq \max(Ra, Rb) \\ \max(Ra, Rb) & \text{if } Rc \leq \min(Ra, Rb) \\ Ra + Rb - Rc & \text{otherwise} \end{cases} \qquad (2.8)$$

2.4.2. Prediction Correction

After Px is computed, the prediction is corrected according to the process depicted in equation (2.9), which depends on the sign detected on the context-type determination process [10]. The new value of Px is clamped in the range [0..Max], where Max is the maximal possible value of Rx(the range of the source digital image).

$$\begin{aligned} &\text{if (sign = positive)} && \text{then } Px = Px + C[Q] \\ & && \text{else } Px = Px - C[Q] \\ &\text{if (Px > Max)} && \text{then } Px = Max \\ &\text{else if (Px< 0)} && \text{then } Px = 0 \end{aligned} \qquad (2.9)$$

As shown in Figure A.8, sign+ is inverted because of the fact that, if *sign+* input is equal to zero, that is the parameter *sign* is negative, the subtractor will perform the subtraction process and generates the corrected prediction value by subtracting the Cq (bias correction value) from previously computed predicted sample value. The outputs of the Figure A.6 will be the inputs for Figure A.8 as indicated by parameter *i*.

If sign+ is equal to one, that is the parameter sign is positive, the process will perform addition operation. Finally, the corrected prediction value is computed.

2.4.3. Computation of Prediction Error

Having the value of Px, corrected by the bias correction process, the prediction error is computed. If the sign of the context-type was negative, the sign of the error is reversed. By using a subtractor, as in Figure A.9, error can be computed.

$$error = Rx - Px$$

$$\text{if (sign=negative)} \quad \text{then} \quad error = -error \tag{2.10}$$

2.4.4. Error Quantization for Near-lossless Mode

The processes in this section are executed if and only if the encoder is working in the near-lossless mode, that is, the variable *near* is not zero. First of all, the error is quantized.

After this quantization, the value Rx of the sample x, which will be used to encode further samples, should be replaced by the reconstructed value as the decoder will compute it.

$$\text{if (error} > 0) \quad \text{then} \quad \text{error} = ((\text{error} + \text{near}) / (2 * \text{near} + 1))$$

$$\text{else} \quad \text{error} = -((\text{near} - \text{error}) / (2 * \text{near} + 1))$$

$$Rx = Px + \text{sign} * \text{error} * (2 * \text{near} + 1) \qquad (2.11)$$

$$\text{if } (Rx < 0) \qquad \text{then } Rx = 0$$

$$\text{else if } (Rx > \text{Max}) \qquad \text{then } Rx = \text{Max}$$

2.4.5. Modulo Reduction of the Prediction Error

The error is reduced to the range relevant for coding process, (- [MODULO / 2] .. [MODULO / 2] – 1).

$$\text{if (error} < -(\text{MODULO} / 2))$$

$$\text{then error} = \text{error} + \text{MODULO} \qquad (2.12)$$

$$\text{else if (error} \geq ((\text{MODULO} + 1) / 2)$$

$$\text{then error} = \text{error} - \text{MODULO}$$

This can be performed by comparing the error value, shown by the parameters z1 to z8 in Figure A.10, with the range of allowed value (which is equal to 129) and if it is greater or equal than this value, subtraction is done for obtaining the proper range for the coding process. Outputs of the figure indicate the error value after the modulo reduction process and are denoted by k1 to k8.

2.5. Prediction Error Encoding

The next step of the regular mode is to encode the error. This is achieved by a Golomb encoding process. For this, the variables A[1..1092] and N[1..1092] are used to first compute the coding parameter *k*.

2.5.1. Golomb Parameter Estimation

One of the crucial steps in schemes using Golomb coding is the determination of the optimal value of the code parameter *k*.

The value yielding the shortest possible average code length for the mapped prediction errors is context-dependent and adaptive [11]. The value of *k* per context-type is updated each time a sample belonging to the same context-type is encoded. The update is based on the accumulated sum of absolute values of prediction errors that occurred in the same context-type. The parameter k, for the Golomb code G(k) is computed from the process in equation (2.13).

$$\text{for } (k=0 \ ; \ (N[Q] << k) < A[Q] \ ; \ k++) \qquad (2.13)$$

By comparing the value of N[Q] with the value of A[Q] and incrementing the value of k by one if N[Q] is greater than A[Q], as shown in Figure A.11, the parameter k is computed.

2.5.2. Error Mapping

Golomb codes were originally designed for non-negative integer values. Prediction errors can be negative as well and their distribution is in general two-sided geometric and symmetric, instead of just one-sided.

There is a need then to extend the codes for these types of distributions. One way of doing this extension is to map all possible error values into non-negative ones previously to the encoding [12].

For this, it is also crucial to have a good estimation of the centre of this two-sided distribution, which is closely related to the bias mentioned before. This mapping approximates the optimal solution for two-sided geometric distributions.

The error to be encoded is mapped to a non-negative value. If working in the lossless mode, the mapping process, presented in equation (2.14), checks for the value k of the Golomb parameter and according to it performs a regular mapping (k not zero) or a special mapping, k=0 and B[Q] less or equal than $-N$[Q] / 2, equivalent to encoding $-(error + 1)$. In the near-lossless mode, the mapping is independent of the value of k.

$$\text{if (near = 0 and } k = 0 \text{ and B[Q]} \leq \text{-N[Q] / 2)} \quad \text{then}$$

$$M(error) = \begin{cases} 2 * error + 1 & \text{if error} \geq 0 \\ -2 * (error + 1) & \text{if error} < 0 \end{cases}$$

$$\text{else} \qquad (2.14)$$

$$M(error) = \begin{cases} 2 * error & \text{if error} \geq 0 \\ -2 * error - 1 & \text{if error} < 0 \end{cases}$$

Depending on the value of the parameter k, the prediction error is multiplied by two and addition is performed, as shown in Figure A.12.

2.5.3. Mapped-error Encoding

Golomb codes were first introduced as means for encoding non-negative run lengths [13]. Given a positive integer parameter *m*, the Golomb code of order *m* encodes an integer *n* greater or equal to zero in two parts.

A *binary* representation of *n modulo m* and a *unary* representation of the integer part of *n/m*. Golomb codes are optimal for geometric probability distributions on the non-negative integers.

For every distribution of this form, there exists a value of the parameter *m* such that the code yields the shortest possible average code length over all uniquely decipherable codes for the non-negative integers.

When Golomb codes are used, only a small number of parameters need to be estimated per context-type. This allows for a large number of context-types, without paying an excessive penalty in code length due to the number of parameters modelled (model cost).

The special case of Golomb codes where *m* is a *k*-power of 2 leads to very simple encoding/decoding procedures: the code for *n* consists of the *k* least significant bits of *n*, followed by the number formed by the remaining higher order bits of *n*, in unary representation. This specific case is denoted by G(*k*). In the sequel, *k* stands for the Golomb parameter, where it is understood that *m* is equal to the *k*-th power of 2.

Thus, after the Golomb parameter k is computed and the map is performed, *M(error)* is encoded according to the following process:

a. The k least significant bits of *M(error)* are appended to the compressed data (bit stream) as they are.

b. The number formed by the remaining high order bits of *M(error)* is appended to the already compressed data (bit stream) in unary representation, that is, by as many zeros as the value of this remainder, followed by a one.

2.6. Parameters Update

The last step of the encoding of the sample x in the regular mode is the update of the parameters A, B, C, N and *sign*. It is important to note that this update is performed at the end of the coding process, after k and M(error) are computed.

2.6.1. Update

The variables B[Q], A[Q] and N[Q] are updated according to the new prediction error that is computed after modulo reduction.

$$B[Q] = B[Q] + sign * error * (2 * near + 1)$$

$$A[Q] = A[Q] + abs(error)$$

$$\text{if } (N[Q] = RESET) \quad \text{then} \qquad (2.15)$$

$$A[Q] = A[Q] / 2$$

$$B[Q] = B[Q] / 2$$

$$N[Q] = N[Q] / 2$$

$$N[Q] = N[Q] + 1$$

2.6.2. Context-dependent Bias Computation

The prediction *Px* was corrected by a context-dependent bias correction term, C[Q]. This correction value is updated at the end of the encoding of the sample x. Three variables are needed for this procedure: C[1..1092], B[1..1092] and N[1..1092].

The variable *C[Q]* corresponds to *the bias correction value* to be added to or subtracted from the value Px previously computed and the variable *N[Q]* is the *number of occurrences* of the context-type *Q*. The variable *B[Q]* is an *auxiliary* variable, since instead of correcting by the average value, a moderate correction is performed. This correction changes the value of C[Q] by one unit at most for every iteration. The variables are clamped to limit their range of possible values. The bias correction term C[Q] is computed according to the process in equation (2.16).

$$\begin{aligned}
&\text{if (} B[Q] \leq -N[Q] \text{)} \quad \text{then} \\
&\text{if (} C[Q] > MIN_C \text{)} \quad \text{then} \quad C[Q] = C[Q] - 1 \\
&B[Q] = B[Q] + N[Q] \\
&\text{if (} B[Q] \leq -N[Q] \text{)} \quad \text{then} \quad B[Q] = -N[Q] + 1 \\
&\text{else if (} B[Q] > 0 \text{)} \quad \text{then} \\
&\text{if (} C[Q] < MAX_C \text{)} \quad \text{then} \quad C[Q] = C[Q] + 1
\end{aligned} \tag{2.16}$$

$$B[Q] = B[Q] - N[Q]$$

$$\text{if } (B[Q] > 0) \qquad \text{then } B[Q] = 0$$

Depending on the value of the parameter sign, error is subtracted from B[Q], or not, as shown in Figure A.13. If B[Q] is greater than zero, it is set to zero by the process shown in Figure A.14.

A[Q] is updated by adding the error value to it and N[Q] is incremented by one, if it is not equal to 64, the value of the parameter "RESET". The bias correction value C[Q] is updated at the end of the process, by incrementing it by one.

2.7. Run Mode

In a pure Huffman coding process, at least one bit per sample is needed. In order to increase the compression in uniform image areas, a run mode coding process is added in this International Standard. The run mode process is especially useful for source images based on compound documents.

If the most significant gradients are all equal to zero or their absolute value is less than or equal to *near* in the near-lossless mode, then the coding enters a run mode. The encoder reads subsequent symbols as long as *Rx* = *Rb* or an end of the current image row is encountered. In the case of near-lossless encoding, if the difference between *Rx and Rb* is less than or equal to the allowed error, the scan continues as well. After the run is aborted, the length of the run is encoded.

This is followed by the encoding of the last scanned symbol if the run was interrupted other than by an end of row.

The run mode is composed of two main steps: *run-length scanning and encoding* and *end of run encoding*. The run-length encoding is performed with a technique derived from Golomb codes. Given a *code-order rm*, where *rm* is restricted to a *rk*-th power of 2, a one bit code word, "0", is used to encode runs of length *rm* and a *rk*+1 bits long code word is used to encode any other event.

The first case represents a *hit* situation, where a run of length *rm* is achieved, while the second case is a *miss* situation, where the run was interrupted before achieving the "maximal" length *rm*. In this miss situation, a prefix bit, "1", is sent, followed by the actual length of the run, which is encoded with *rk* bits. The value of *rm* is adapted, according to a pre-defined table *J* of 32 entries for values of *rk*, each time a run of length *rm* is scanned (the index to the table *J* increases) or a miss has occurred (the index to the table *J* decreases). If the run was aborted by an end of row, another "0" is appended to the bit stream.

The process of coding run lengths can be seen as an extension of Golomb coding. In the run mode, the coding works faster, since there is no need for context modeling and prediction.

2.7.1. Run-length Encoding

The first step in the run mode is to read the source image data until a sample of different from Rb is scanned or the end of the current image row is reached.

In the case of near-lossless encoding, if the difference between *Rx and Rb* is not larger than the allowed error, the scan continues as well.

$$\text{RUNcnt} = 0$$

$$\text{while } ((\text{Rx} = \text{Rb}) \text{ or } (\text{near} > 0 \text{ and } \text{abs}(\text{Rx} - \text{Rb}) \leq \text{near }))$$

$$\text{if } (\text{end of current row }) \text{ then stop}$$

$$\text{else } \text{RUNcnt} = \text{RUNcnt} + 1 \qquad (2.17)$$

$$\text{if } (\text{near} > 0) \quad \text{then } \text{Rx} = \text{Rb}$$

The subsequent sample values Rx and Rb are compared so as to increment RUNcnt by one unit, if they are equal. This process is shown in Figure A.15. At the beginning of the run mode, RUNcnt is reset.

The variable *RUNcnt* computed following the procedure in equation (2.17) represents the run-length. A "0" is appended to the bit stream for each run of length *rm*, where *rm* is obtained from the 32-entries table *J*. The index, *RUNindex*, to the table *J* is increased, to a maximum value of 31, each time a run of length *rm* was reached.

The table *J* contains values for *rk*, not *rm*. The complete procedure for this part is specified in equation (2.18). If the run was aborted at *an end of row* and the remaining length is greater than zero, an extra "0" is appended to the bit stream.

If the run was aborted by a sample of a different value, the remaining length is coded by a code word of length $rk+1$ (a prefix bit, "1", followed by rk bits to encode the remaining run length) and the index *RUNindex* is decreased (not to less than 0). This is detailed in equation (2.19).

$$\begin{aligned}
&\text{while (RUNcnt} \geq 2^{J[RUNindex]} \text{)} \quad \text{do} \\
&\quad \text{Appending "0" to the bit stream} \\
&\quad \text{RUNcnt = RUNcnt} - 2^{J[RUNindex]} \qquad (2.18) \\
&\quad \text{if (RUNindex < 31)} \\
&\quad \text{then RUNindex = RUNindex + 1} \\
&\quad \text{if (not end of current row) then} \\
&\quad \text{Appending "1" to the bit stream} \\
&\text{Appending RUNcnt in binary representation to the bit stream} \\
&\quad \text{if (RUNindex > 0)} \\
&\quad \text{then RUNindex = RUNindex} - 1 \qquad (2.19) \\
&\text{else if} \quad \text{RUNcnt > 0} \quad \text{then} \\
&\quad \text{Appending "0" to the bit stream}
\end{aligned}$$

According the value of run-length, a zero or one is taken as output and is shown in Figure A.16. If RUNcnt is greater or equal than zero, the output app will be zero. At the end of the run-length encoding process, RUNcnt is decremented by one.

2.7.2. Run Interruption Sample Encoding

If the run was aborted other than by the end of the image row, the new sample that caused the run interruption is to be encoded. This is done by encoding the difference between the value at the current sample, *Rx* and the value at *a* or *b* (both positions corresponding to the current symbol to be encoded).

In this mode of operation, two different context-types are used: One is when *Ra* is up to *near* from *Rb* and the second when their difference is larger than *near*. The basic concepts in the end of run encoding are the same as those used to encode a new sample in the regular encoding mode, with the additional information that *Rx* must differ from *Rb* more than *near*, otherwise the run would have continued.

Equation (2.20) shows computing of the index EORtype, which defines the context-type.

$$\text{if (abs (Ra} - \text{Rb)} \leq \text{near)} \quad \text{then} \quad \text{EORtype} = 1 \qquad \text{else} \quad \text{EORtype} = 0 \tag{2.20}$$

The prediction error is computed by the equation (2.21).

$$\text{if (EORtype} = 1) \quad \text{then} \quad \text{error} = \text{Rx} - \text{Rb} \qquad \text{else} \quad \text{error} = \text{Rx} - \text{Ra} \tag{2.21}$$

The sign of the *error* is corrected if necessary. This step is analogous to the context-merging process in the regular coding mode. If near-lossless mode, then *error* is quantized, as done in equation (2.11). The error is then reduced using the variable *MODULO*.

$$\text{if (EORtype = 1) and (Ra < Rb)} \quad \text{then error = - error}$$

$$\text{if (near > 0)} \quad \text{then error = quantize (error)} \qquad (2.22)$$

$$\text{error = error mod MODULO}$$

By a subtraction process, as shown in Figure A.17, first error is computed, then it is reduced to a relevant range.

The auxiliary variable *EA* is computed as in equation (2.23). This variable is used for the computation of the Golomb parameter *k*.

$$\text{if (EORtype = 0)} \quad \text{then EA = EA[0]}$$

$$\text{else EA = EA[1] + EN[1] / 2} \qquad (2.23)$$

The Golomb parameter *k* is computed following the same procedure as in the regular mode, shown in equation (2.13), based now on *EN[EORtype]* and *EA* instead of *N[Q]* and *A[Q]*. Figure A.18 shows the process of obtaining the parameter, k.

$$\text{for (k=0 ; (EN[EORtype] << k) < EA ; k++)} \qquad (2.24)$$

The flag *map* is computed as in equation (2.25). This variable will influence the mapping of *error* to non-negative values.

$$\text{if (k = 0 and error > 0 and ENn[EORtype] < EN[EORtype] / 2)}$$

then map = 1

else if (error < 0 and ENn[EORtype] ≥ EN[EORtype] / 2)

then map = 1

(2.25)

else if (error < 0 and k ≠ 0) then map = 1

else map = 0

Error mapping can be performed as:

$$EM(error) = 2 * abs(error) - EORtype - map \qquad (2.26)$$

The outputs of the Figure A.17 (rem1 to rem8) are also the outputs of the encoding process.

Golomb encode EM(error) following the same procedures as in the regular mode.

a. The *k* least significant bits of *EM*(*error*) are appended to the compressed data (bit stream) as they are.

b. The number formed by the remaining high order bits of *EM*(*error*) is appended to the already compressed data (bit stream) in unary representation, that is, by as many zeros as the value of this remainder, followed by a one.

For the end of run encoding, the variables can be updated as:

if (error < 0) then ENn[EORtype] = ENn[EORtype] + 1

else EA[EORtype] = EA[EORtype] + (EM(error) + 1 – EORtype) / 2

if (EN[EORtype] = RESET) then

(2.27)

EA[EORtype] = EA[EORtype] / 2

EN[EORtype] = EN[EORtype] / 2

ENn(EORtype] = ENn[EORtype] / 2

else EN[EORtype] = EN[EORtype] + 1

The update process is similar with the regular mode and is shown in Figure A.19.

2.8. End of Run Processing

At the end of run mode RUNcnt is reset to zero.

$$RUNcnt \doteq 0$$

2.9. End of Encoding

After creating each individual block, they are combined together to obtain the top-level schematic of encoding process, which has two modes of operation, regular and run mode. The regular mode is shown in Figure A.24. Five neighborhood samples are the inputs of the local gradients block. Most significant gradients, D1, D2 and D3 are fed into mode check block in order to check whether the encoder enters run mode or not. These local gradients are also used for determining the sign.

Then prediction computation and correction steps are performed. This corrected prediction value and current sample value, Rx are used for finding the prediction error value, which will be reduced to a relevant range.

Golomb parameter, k is computed and used for mapping the error value. At the end of the regular mode, the parameters are updated.

If the encoder enters in run mode, run mode process is performed as shown in Figure A.25. Current sample value, Rx is compared with the previous sample value, Rb for determining the run-length and encoding of this variable.

Depending on the value of Golomb parameter, k, error value is mapped. Parameters are updated as a last step in run mode.

By combining regular and run mode blocks, top-level encoding schematic is obtained, as shown in Figure A.26. The only connection between these blocks is used for determining the mode of operation (regular or run mode).

After the last step (for both the regular and run mode) is completed, next sample at the digital source image data should be taken in order to encode it and this will continue up to encoding of the last sample.

If all the samples have been processed, the encoding part will be ended and the compressed image data have been obtained. To get the reconstructed image data, which must be identical to the digital source image data, compressed image data should be decompressed by the decoding process.

3. DECODING PROCESSES

The coding processes specified in this International Standard [6] are fairly symmetric , meaning that both encoding and decoding processes use the same basic procedures and follow almost the same process flow (besides a few sign changes and some parts in reverse order).

The main procedures for the lossless (and near-lossless) decoding process are given in Figure 3.1.

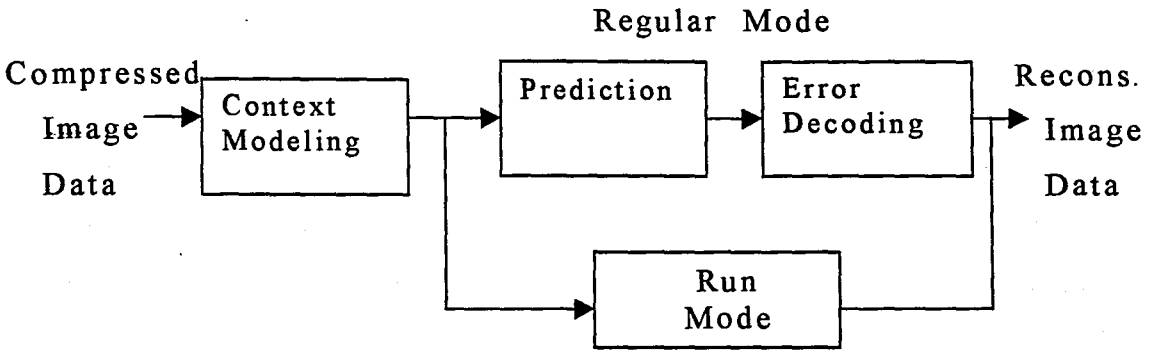

FIGURE 3.1. Lossless decoder simplified diagram.

3.1. Initializations and Conventions

All the initializations that have specified in Section 2.2 are performed at the beginning of the decoding process. The coding parameters that have been written into the bit stream, as specified in Table 2.2, are taken.

3.2. Context-type Determination

The decoding of the sample x is also based on its causal template composed of the samples a, b, c, d and e as shown in Figure 2.2.

3.2.1. Gradient Computation

The first step in the context determination process is to compute the local gradient values of the neigborhood samples. This is done using equation (2.3).

3.2.2. Run Mode Check

If the most significant local gradients are all zero or their values are less than or equal to the allowed error for near-lossless coding, the decoder enters the run mode. The run mode detection process is specified in equation (2.4). In the case of lossless coding, this run mode detection process is equivalent to the mode shown in equation (2.5), where the coder is checking if Ra=Rb=Rc=Rd.

3.2.3. Gradient Quantization

In the case of a regular mode, the context-type determination process continues and quantizes D1, D2 and D3 according to the process specified in equation (2.6). For this, the non-negative thresholds T1, T2 and T3 are used.

The process also quantizes D4 according to equation (2.7) using the non-negative threshold T4, to obtain Q4.

3.2.4. Context-type Merging

If the first non-zero component of the vector (Q1, Q2, Q3) is negative, then all the signs of the vector (Q1, Q2, Q3, Q4) are reversed to obtain (-Q1, -Q2, -Q3, -Q4). In this case, the variable *sign* gets the value *negative*, otherwise, *positive*.

This vector is mapped into an integer number Q representing the context-type for the sample x, after this possible merging.

3.3. Prediction

The prediction unit is based on the subset a, b and c of the causal template depicted in Figure 2.2, where x denotes the current sample to be decoded. The prediction process at decoding part is same as shown in equation (2.8).

After Px is computed, the prediction is corrected according to the process depicted in equation (2.9), which depends on the sign detected on the context-type determination process. The new value of Px is clamped in the range [0..Max], where Max is the maximal possible value of Rx(the range of the source digital image).

3.4. Prediction Error Decoding

The next step of the regular mode is to decode the error. This is achieved by a Golomb decoding process. For this, the variables A[1..1092] and N[1..1092] are used to first compute the coding parameter k.

3.4.1. Golomb Parameter Estimation

The parameter k, for the Golomb code G(k) is computed from the process in equation (2.13).

3.4.2. Mapped-error Decoding

Golomb decode the mapped error value *M(error)*, which is computed at Section 2.5.3, as follows.

a. Reading k bits to compose the k least significant bits of *M(error)*.

b. Reading the unary code to form the remainder bits of *M(error)*.

3.4.3. Inverse of Error Mapping

Performing the inverse of the error mapping, where now M(error) is known and error is computed.

$$\begin{aligned} &\text{if (near = 0 and } k \text{ = 0 and B[Q]} \leq \text{-N[Q] / 2)} \quad \text{then} \\ &\qquad \text{error = ((M(error) -1) / 2)} \\ &\text{else} \\ &\qquad \text{error = (M(error) / 2)} \end{aligned} \tag{3.1}$$

Depending on the value of the parameter k, error value is computed, as shown in Figure A.20. For the near-lossless mode, this error value is multiplied by (2*near+1).

If the variable *sign* is negative, sign of the *error* is inverted.

3.5. Computation of Reconstructed Value

The reconstructed image data is obtained by computing the current sample value using the equation (3.2).

$$Rx = (error + Px) \bmod MODULO * (2 * near + 1)$$

$$\text{if } (Rx < 0) \quad \text{then } Rx = 0 \qquad (3.2)$$

$$\text{else if } (Rx > Max) \quad \text{then } Rx = Max$$

At the end of this step, the reconstructed sample values are obtained, as shown in Figure A.21. They have to be same with the digital source image data sample values.

3.6. Parameters Update

The last step of the decoding of the sample x in the regular mode is the update of the parameters A, B, C, N and *sign*.

3.6.1. Update

The variables B[Q], A[Q] and N[Q] are updated according to equation (2.15). The error value, used in this equation, is the one computed in inverse of error mapping section.

3.6.2. Context-dependent Bias Computation

The bias correction term C[Q] is computed according to the process in equation (2.16).

3.7. Run Mode of Decoding

3.7.1. Run-length Decoding

The run mode in the decoding process has an order of almost the reverse of the encoding process in order to obtain the digital reconstructed image data.

In the encoding part, current sample value (Rx) was compared with Rb value, then run-length was computed and a value of zero or one was taken as output. Now, the inverse operation has to be done, as shown in Figure A.22.

a. Reading a bit R, from the bit stream.

b. If R=0 then;

i. Filling the image with (1 << J[RUNindex]) samples of value Rb or until an end of row is achieved.

ii. If not an end of row and RUNindex < 31, RUNindex is increased by one and more bits are taken from the bit stream. If end of row, it is necessary to go on decoding the next sample starting from the gradient computation step.

c. If R=1 then;

i. Reading J[RUNindex] bits from the bit stream and filling the image with the value Rb for as many samples as this number formed by these bits.

ii. If RUNindex > 0, RUNindex is decremented.

3.7.2. Run Interruption Sample Decoding

End of run decoding is performed by reversing the run interruption sample encoding, as shown in Figure A.23.

Equation (3.3) shows computing of the index EORtype, which defines the context-type.

$$\text{if (abs (Ra – Rb) } \leq \text{ near)} \quad \text{then} \quad \text{EORtype} = 1 \qquad \text{else} \quad \text{EORtype} = 0 \tag{3.3}$$

The auxiliary variable *EA* is computed as in equation (3.4). This variable is used for the computation of the Golomb parameter *k*.

$$\text{if (EORtype = 0)} \quad \text{then} \quad \text{EA} = \text{EA[0]} \qquad \text{else} \quad \text{EA} = \text{EA[1]} + \text{EN[1]} / 2 \tag{3.4}$$

The Golomb parameter *k* is computed following the same procedure as in the regular mode, based now on *EN[EORtype]* and *EA* instead of *N[Q]* and *A[Q]*.

$$\text{for (k=0 ; (EN[EORtype] << k) < EA ; k++)} \tag{3.5}$$

Golomb decode the mapped error value *EM(error)*, which is computed at Section 2.7.2, as follows.

a. Reading *k* bits to compose the *k* least significant bits of *EM(error)*.

b. Reading the unary code to form the remainder bits of *EM(error)*.

After *EM(error)* is found, error can be calculated as:

$$\text{error} = (EM(error) + \text{EORtype})/\ 2 \qquad (3.6)$$

The reconstructed image sample value can be computed by the following equation.

$$\begin{aligned} &\text{if (EORtype = 1)} \quad \text{then } Rx = \text{error} + Rb \\ &\qquad\qquad\qquad\qquad \text{else } Rx = \text{error} + Ra \end{aligned} \qquad (3.7)$$

For the end of run decoding, the variables can be updated as:

$$\begin{aligned} &\text{if (error} < 0) \quad \text{then } ENn[EORtype] = ENn[EORtype] + 1 \\ &\text{else } EA[EORtype] = EA[EORtype] + (EM(error) + 1 - EORtype) / 2 \\ &\text{if (} EN[EORtype] = RESET) \quad \text{then} \\ &\qquad EA[EORtype] = EA[EORtype] / 2 \\ &\qquad EN[EORtype] = EN[EORtype] / 2 \\ &\qquad ENn(EORtype] = ENn[EORtype] / 2 \\ &\text{else } EN[EORtype] = EN[EORtype] + 1 \end{aligned} \qquad (3.8)$$

3.8. End of Decoding

Similar with the encoding process, after creating each individual block, they are combined together to obtain the top-level schematic of decoding process, which has two modes of operation, regular and run mode. Regular mode is shown in Figure A.27. Almost all blocks are common with encoding process, except for two blocks, that are used for computation of error and reconstructed sample value, Rx.

If the decoder enters in run mode, run mode process is performed as shown in Figure A.28. Mapped error value, that was computed in encoding process, is used for determining the reconstructed sample value.

The neighborhood sample value Rb and the variable app, which was appended in the encoding process, are used for determining the reconstructed sample value. Parameters are updated at the end of the decoding process.

By combining regular and run mode blocks, top-level decoding schematic is obtained, as shown in Figure A.29. The only connection between these blocks is used for determining the mode of operation (regular or run mode).

After the last step (for both the regular and run mode) is completed, next sample should be taken in order to decode it and this will continue up to decoding of the last sample.

If all the samples have been processed, the decoding part will be ended and the reconstructed image data have been obtained which must be identical to the digital source image data.

3.9. End of Coding Process

Top-level schematics of encoding and decoding processes are fairly symmetric, meaning that both encoding and decoding processes use the same basic procedures and follow almost the same process flow, only a few blocks are different. Therefore, there is no need to use duplicated blocks.

By putting a mode select switch to the top-level circuit, it will be possible to choose whether the coder works as an encoder or a decoder. This can be accomplished by a simple 2-input multiplexer, as shown in Figure 3.2.

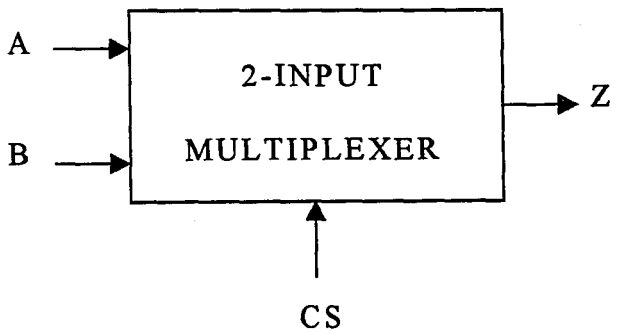

FIGURE 3.2. 2-Input multiplexer.

The function table of a 2-input multiplexer is shown in Table 3.1. If CS(mode select) is equal to zero, the coder works as an encoder, else if it is one, it works as a decoder.

TABLE 3.1. Function table of a 2-input multiplexer.

A	B	S	Z
0	X	0	0
1	X	0	1
X	0	1	0
X	1	1	1

The sample values are represented by 8 bits, so when it is necessary to choose one of the two 8-bit samples, depending on the process, a circuit, as shown in Figure A.30, has to be used.

After combining the encoding and decoding blocks together by deleting the duplicated blocks and putting a mode select switch, the top-level schematic of the coding process can be created as shown in Figure A.31.

The main algorithm in coding process is that, in order to encode or decode the current sample, the neighborhood samples are used.

Therefore, all the samples at the previous row and two previous samples of the current row have to be held by using a register. This register can be created by connecting the 259 D-latches in a series form, 256 of them belonging the previous row and 3 of them belonging the current row, as shown in Figure 3.3.

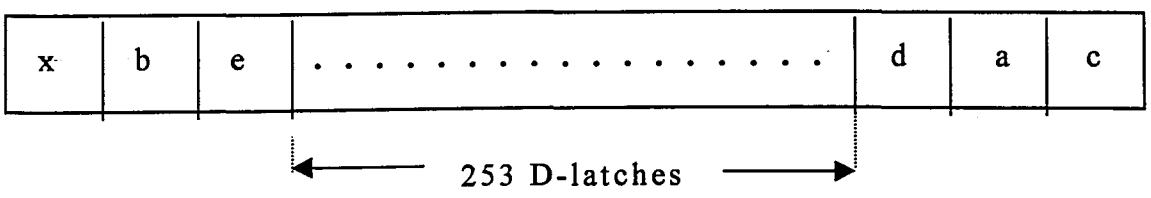

FIGURE 3.3. Register formed by 259 D-latches.

Each block, that is shown in Figure 3.3, is created by connecting eight D flip-flops in parallel, meaning that clock pulses are applied to all inputs at the same time. At every clock pulse the value of the current block is shifted to right.

Top-level schematic of the overall circuit can be obtained by connecting the register block and top-level coding block, as shown in Figure A.32. 8-bit current sample value is fed into the circuit at every clock pulse (symbolized by In(8:1)) and output is taken from Out(8:1).

Mode select switch, denoted by cs, is used for determining the process mode (encoding or decoding). All other symbols are used for initializing and updating the process.

4. SIMULATION

After each resulting schematic circuit, a simulation is done to check the functions of the blocks. Then, top-level schematic, as shown in Figure A.32 is obtained and simulated.

In simulation of the coding process, digital sample values are given as 8 bits/pixel to the input of the encoding process and at the output of the decoding process, reconstructed sample values are taken without any loss.

The inputs and outputs of the coding process are shown in Figure 4.1., 4.2 and 4.3, for regular and run mode.

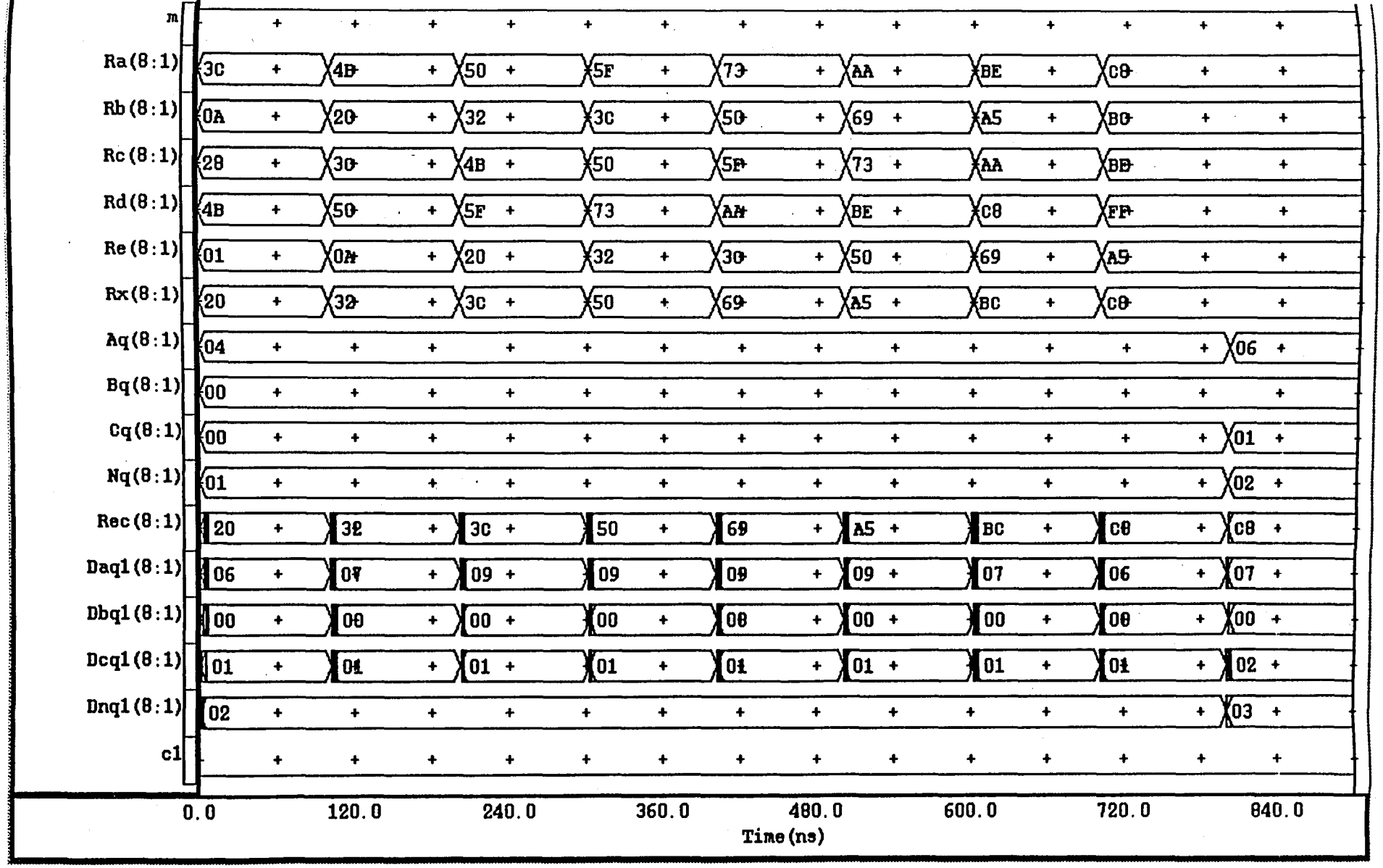

FIGURE 4.1. Timing of the coding process in regular mode.

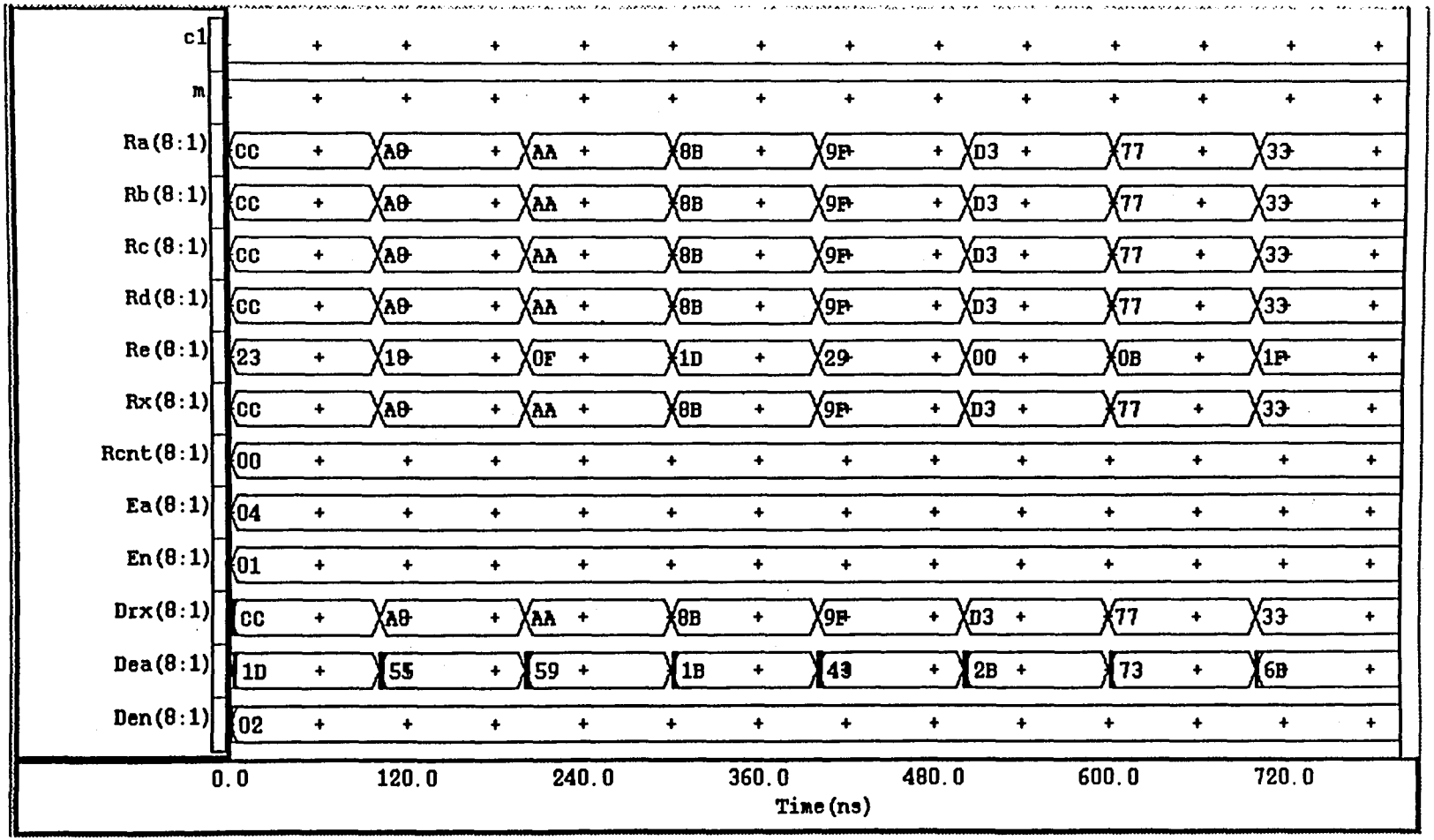

FIGURE 4.2. Timing of the coding process in run mode.

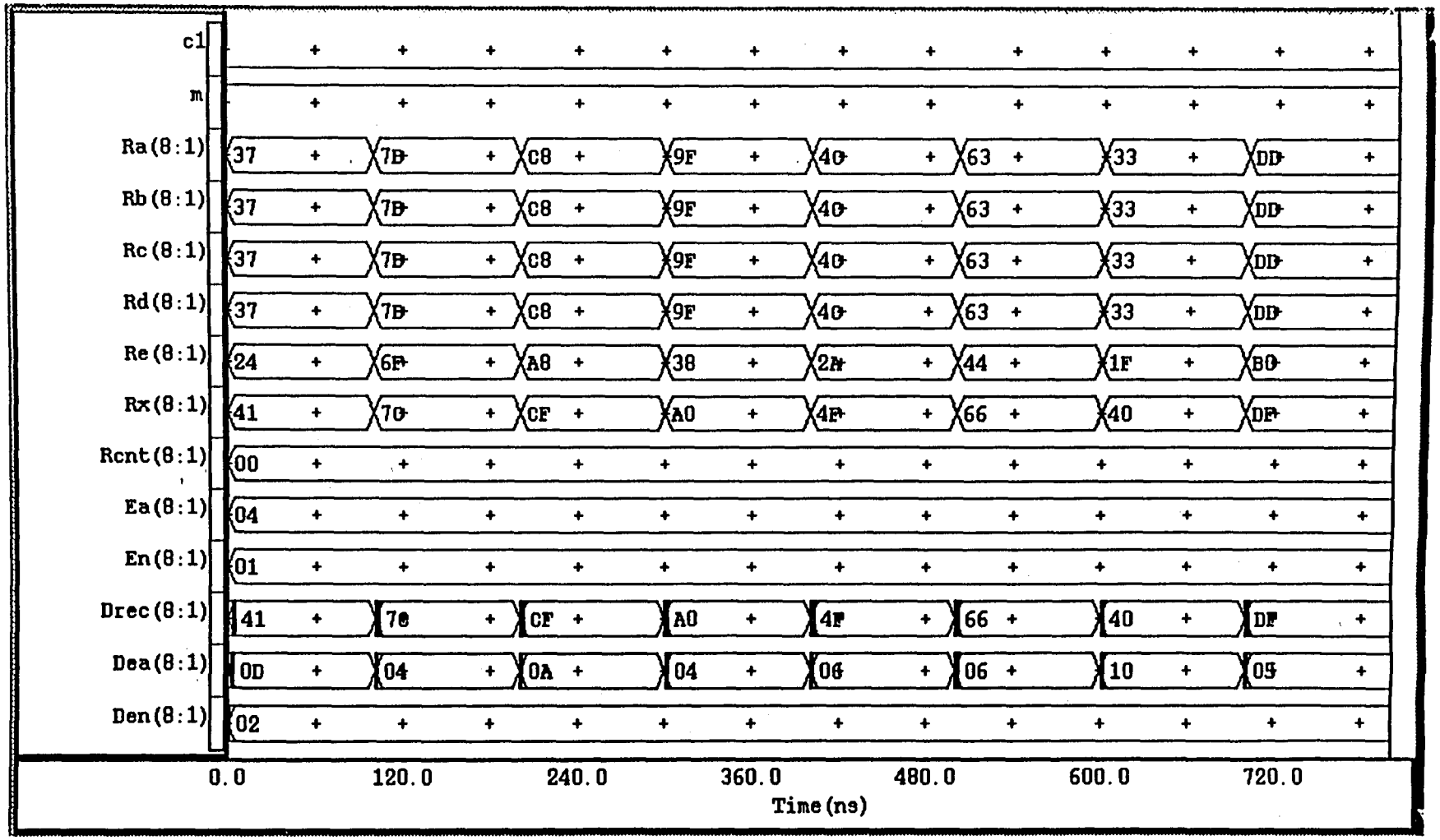

FIGURE 4.3. Timing of the run interruption sample coding process.

In these simulations, it is observed that, if both circuits are functioning properly and perfectly, reconstruction can be achieved at the output of the decoding process.

In the regular mode, current image sample value Rx(8:1) is first compressed by encoding process and then the identical sample value, Rec(8:1) is obtained by decoding process which can be seen in Figure 4.1.

In the run mode, process works faster because of the fact that, the encoding procedure skips the predictor and encoding of the prediction error steps. In Figure 4.2, it can be seen that if the sample values are identical (i.e. Ra=Rb=Rc=Rd), run mode is performed and reconstructed image sample value, Drx(8:1), that is identical with the digital source image sample value, Rx(8:1), is obtained.

Figure 4.3 shows the run interruption sample coding process, that is, the run was aborted other than by the end of the image row. The sample value that caused the run interruption is obtained at the output without any loss (Drec(8:1) = Rx(8:1))

After success in these steps, top level circuit is analyzed , as shown in Figure 4.4, which has cs(mode select) and clk(clock) inputs, additionally. Operations in excess of 40 Mhz. have been observed.

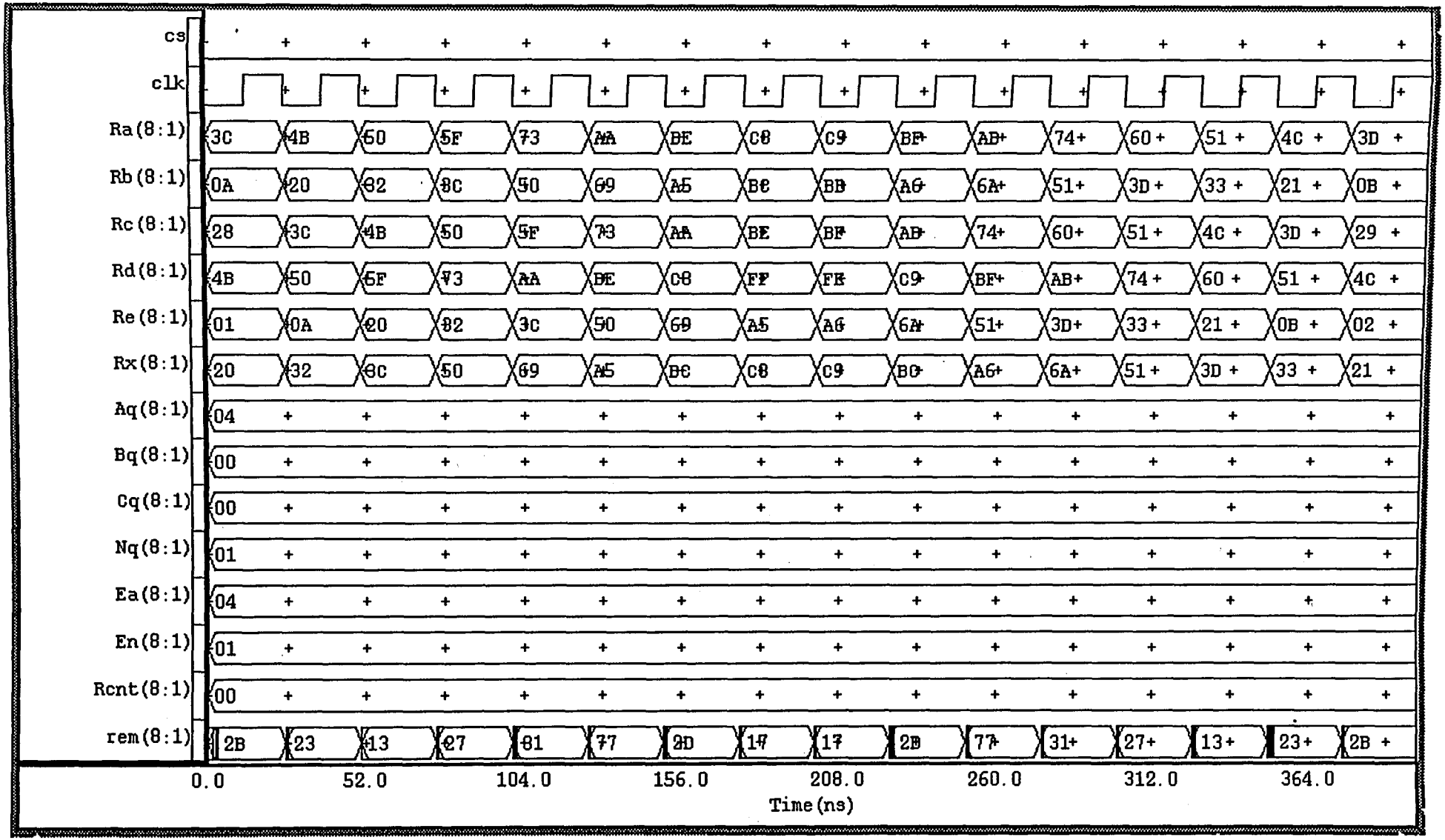

FIGURE 4.4. Timing of the encoding process with chip select and clock.

The original (uncompressed) image consists of 256 rows of 256 pixels and each pixel is represented by 8 bits. So the uncompressed representation uses 65,536 bytes. A number of images are fed to the coding process as a stream, since this design is 1D (one dimension). Then, the compression ratios are computed. The compression ratio is simply the ratio of the number of bytes in the uncompressed representation to the number of bytes in the compressed representation. This is shown in Table 4.1.

TABLE 4.1. Compression ratios for different images.

Image Name	Total Size (Bytes)	Compression Ratio
Lena	49,275	1.33
Sena	37,449	1.75
Sensin	41,742	1.57
Earth	36,612	1.79
Omaha	52,012	1.26

A compression ratio of range from 1.26 to 1.79 is obtained after applying different image sample values in simulation steps. If the image contains similar sample values, this causes higher compression ratios.

As can be seen easily from the Table 4.1, the compression ratio may differ from image to image. This can cause some problems in certain applications where it is necessary to know in advance how many bytes will be needed to represent a particular data set [1].

After all these simulation steps, IC layout creation begins. The topmost level circuit's chip layout is created and is shown in Figure 4.5.

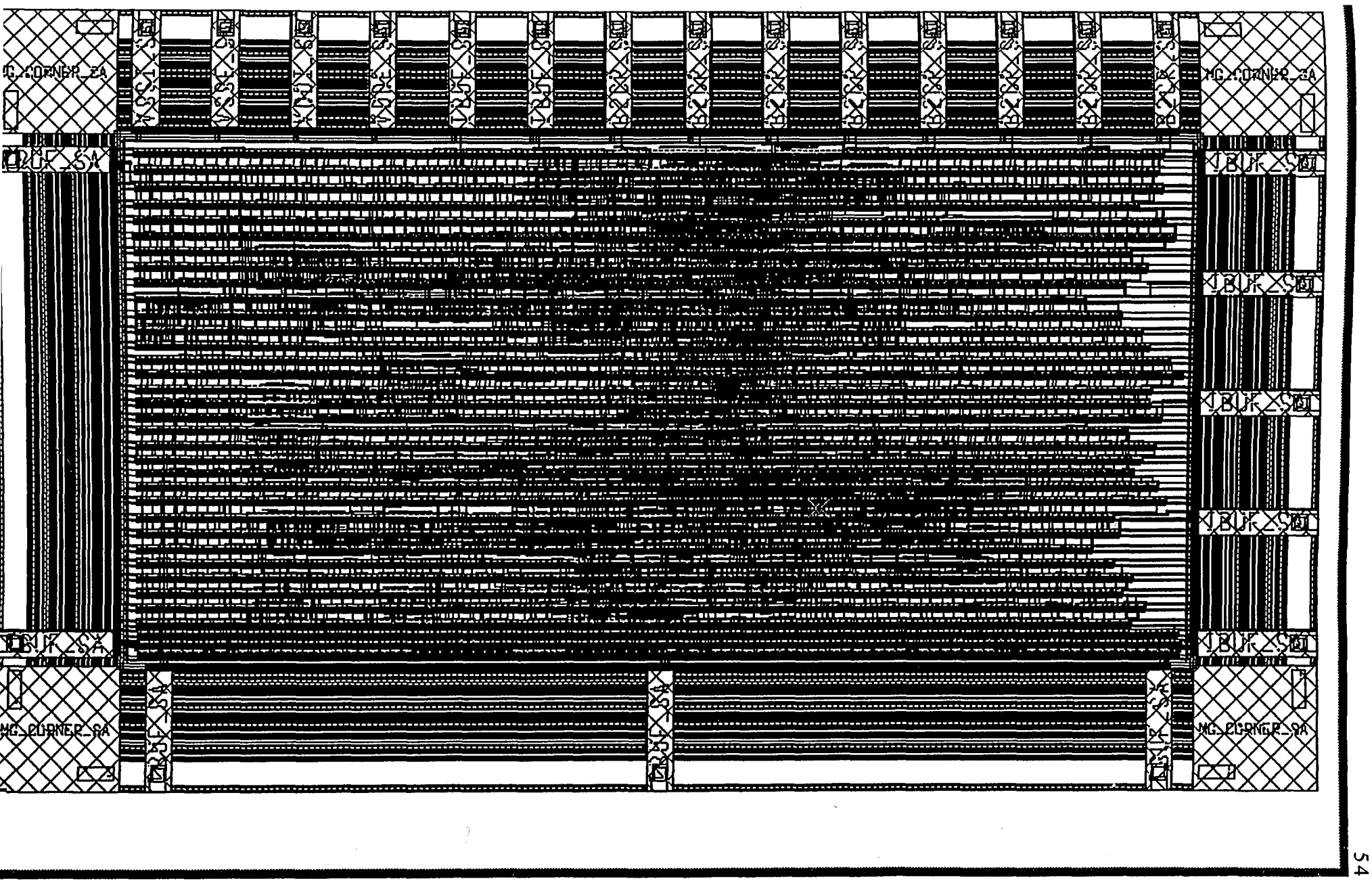

FIGURE 4.5. Layout of the chip.

The overall chip dimension has a cell height of 4386.1μm and a cell width of 7381.7μm. Therefore, the chip area is 32.37 mm^2 in which 67,295 transistors reside.

IC has 12 pins for the inputs, 8 for the outputs and 4 for the power supply, so totally 24 pins.

5. CONCLUSION

In this thesis, a chip which performs lossless compression of continuous-tone still images is designed. In this design, a new International Standard [6] is used for compression.

The results that were taken after simulation ensures a compression ratio of about 1.5. Making image files smaller is an advantage for transmitting files across networks and for archiving libraries of images. If thousands of images were saved in an archive, a reduction of one bit per pixel saves many megabytes in disk space [14].

Lossless compression procedures, generally do not offer high compression ratios. Therefore, this procedure does not have so many applications and can be used in medical instruments and some special projects [15].

In this design, after creating the chip layout, the number of transistors used in the top-level circuit was estimated and the transistor count for the overall chip layout was obtained as 67,295.

This design has been made for 8 bits per pixel. To use this design for more bits per pixel operations, some changes have to be done at the initialization part. And also, for the lossless mode, the parameter "near" is taken to be zero. This can be changed to a different value for the near-lossless procedures depending on the applications.

The future work for this design will include design of more than 8 bits per pixel version for color images and in order to get high compression ratios, applying near-lossless mode for image compression applications.

For higher resolution, image size may be greater than 256*256, which necessitates a register of having higher capacity, so increasing the chip size.

APPENDIX A

SCHEMATIC DIAGRAMS

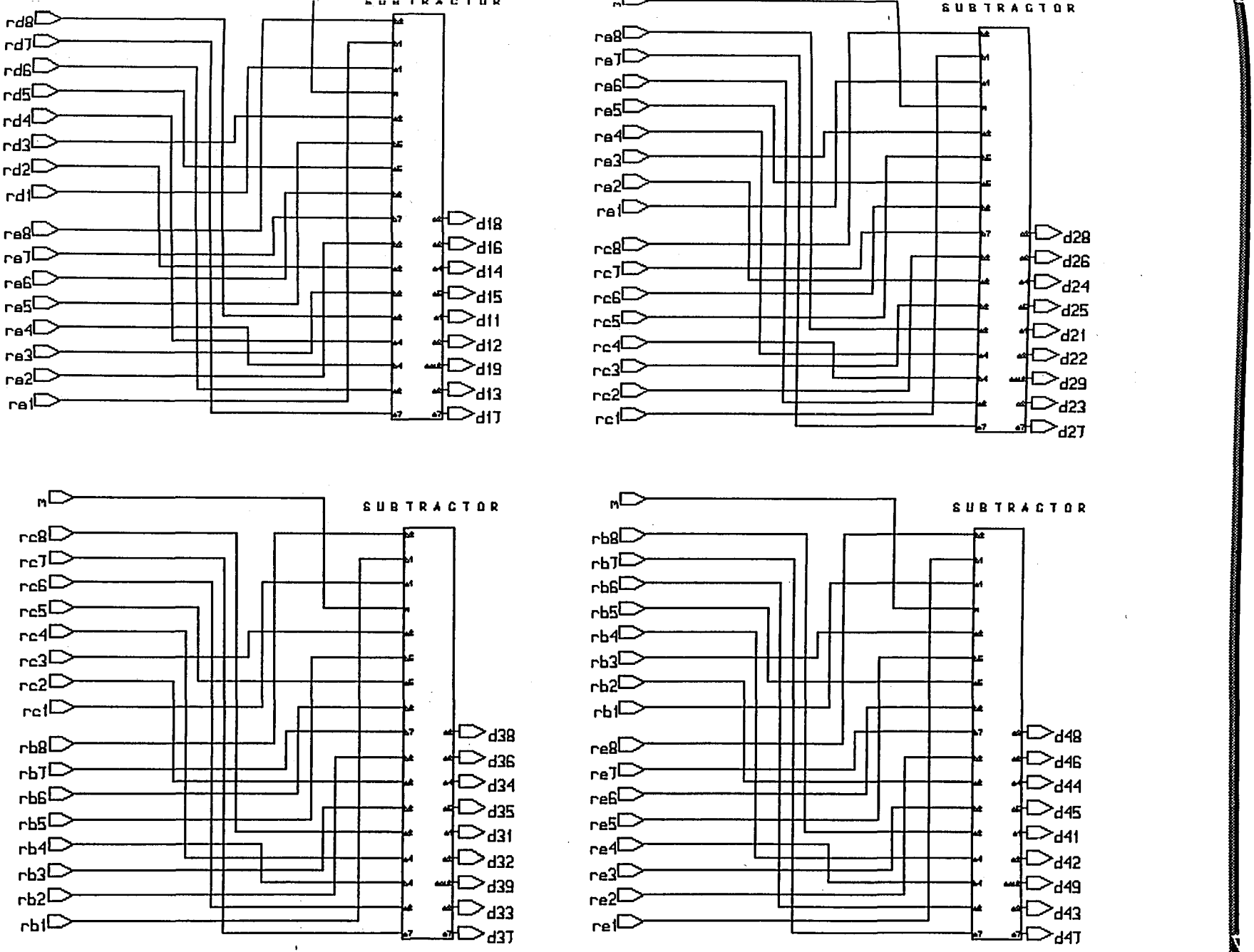

FIGURE A.1. Block diagram of gradient computation circuit.

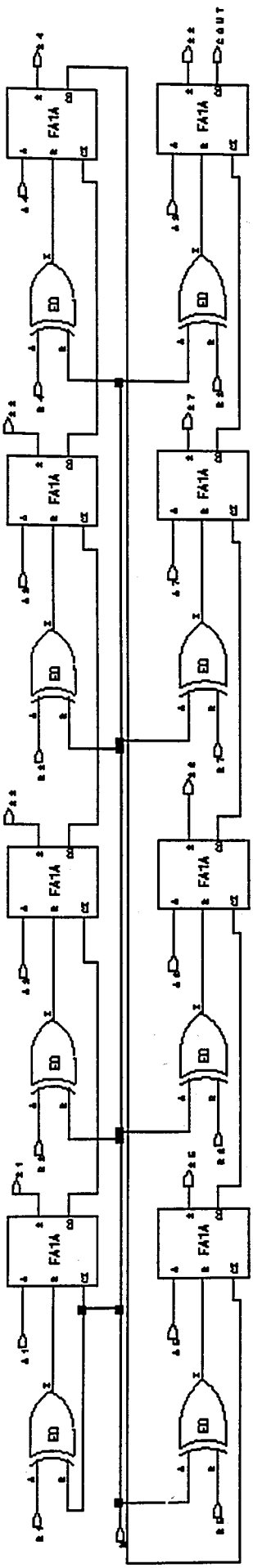

FIGURE A.2. 8-bit subtractor.

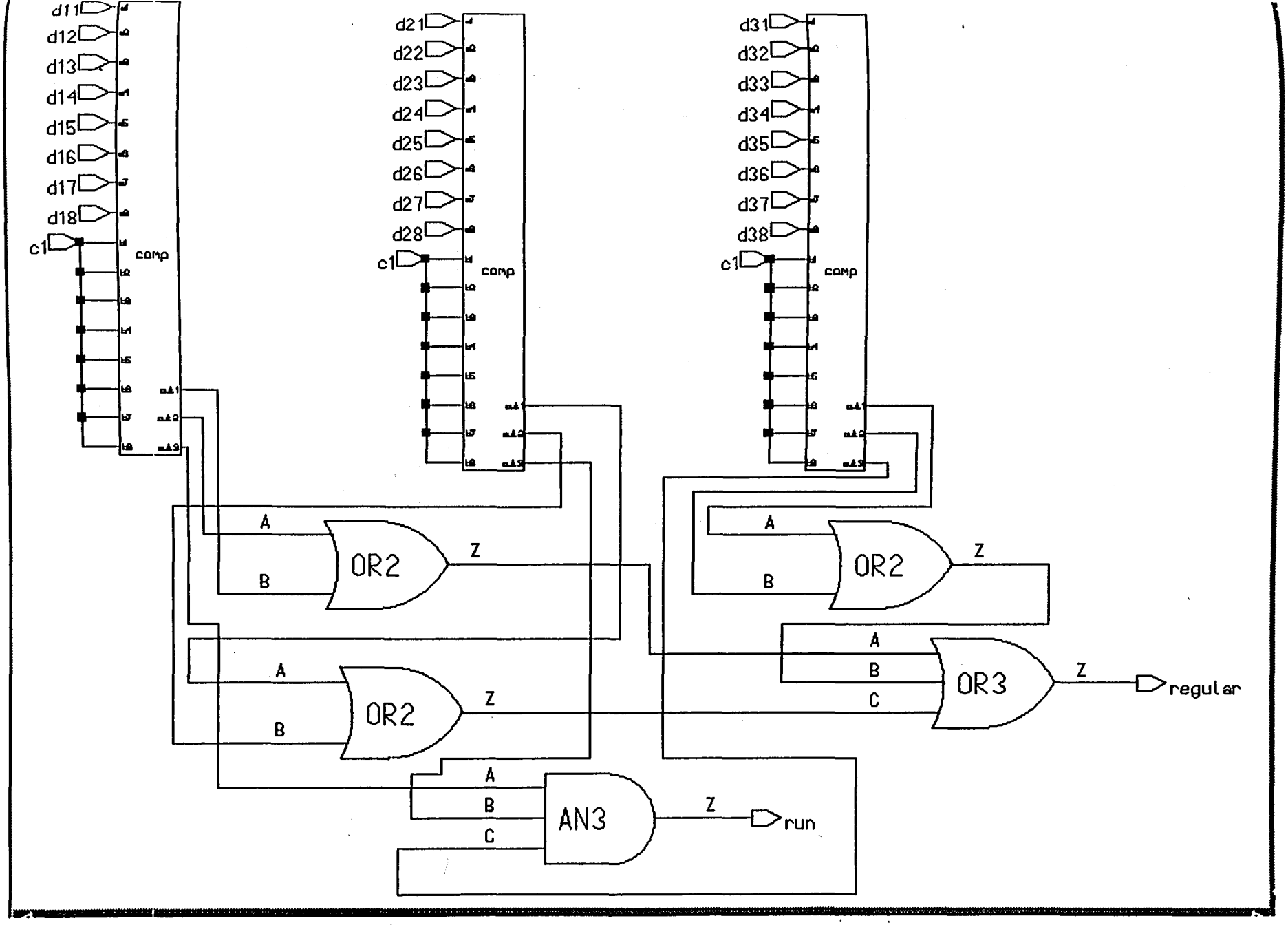

FIGURE A.3. Block diagram of run mode check circuit.

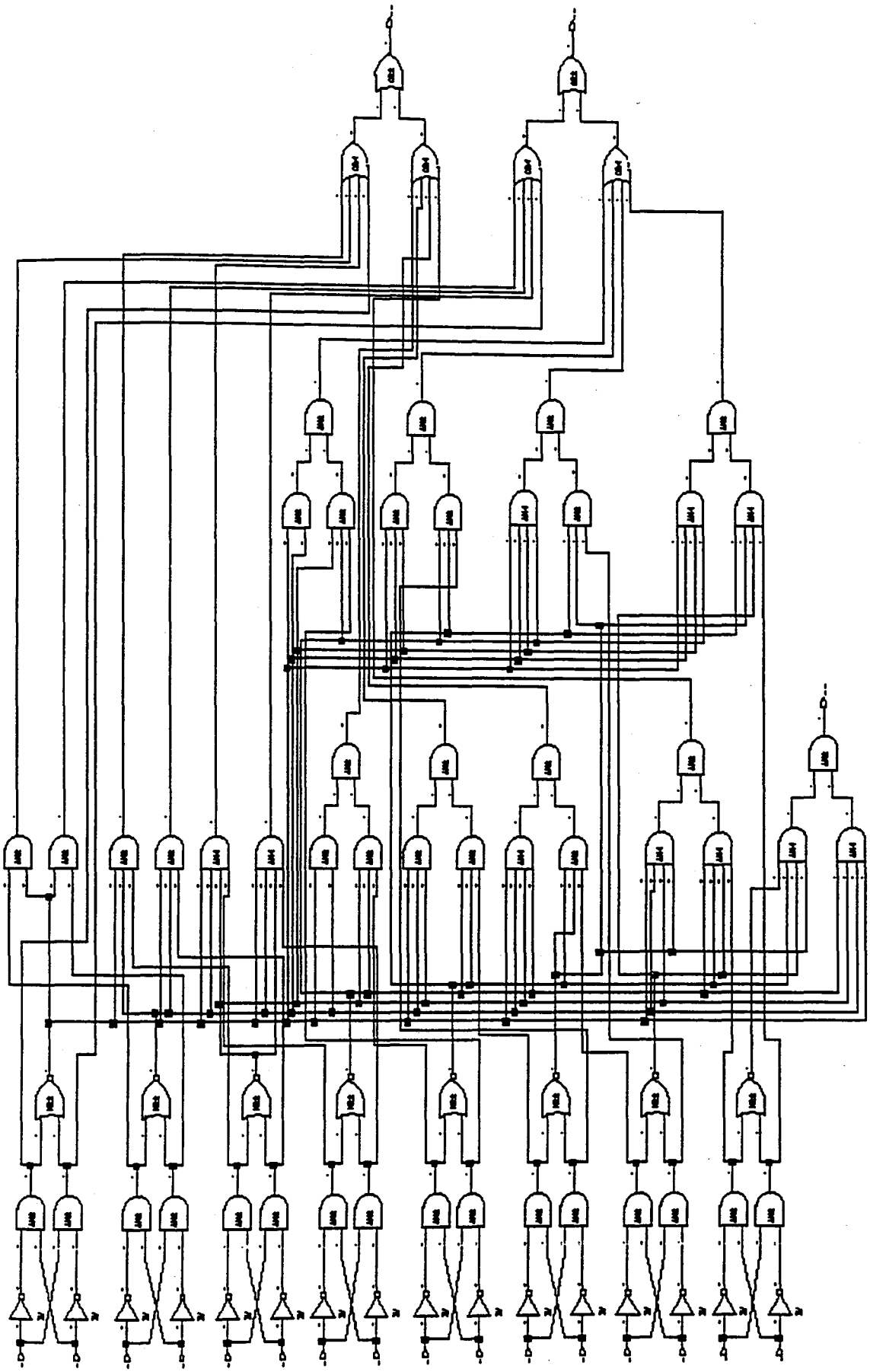

FIGURE A.4. 8-bit comparator.

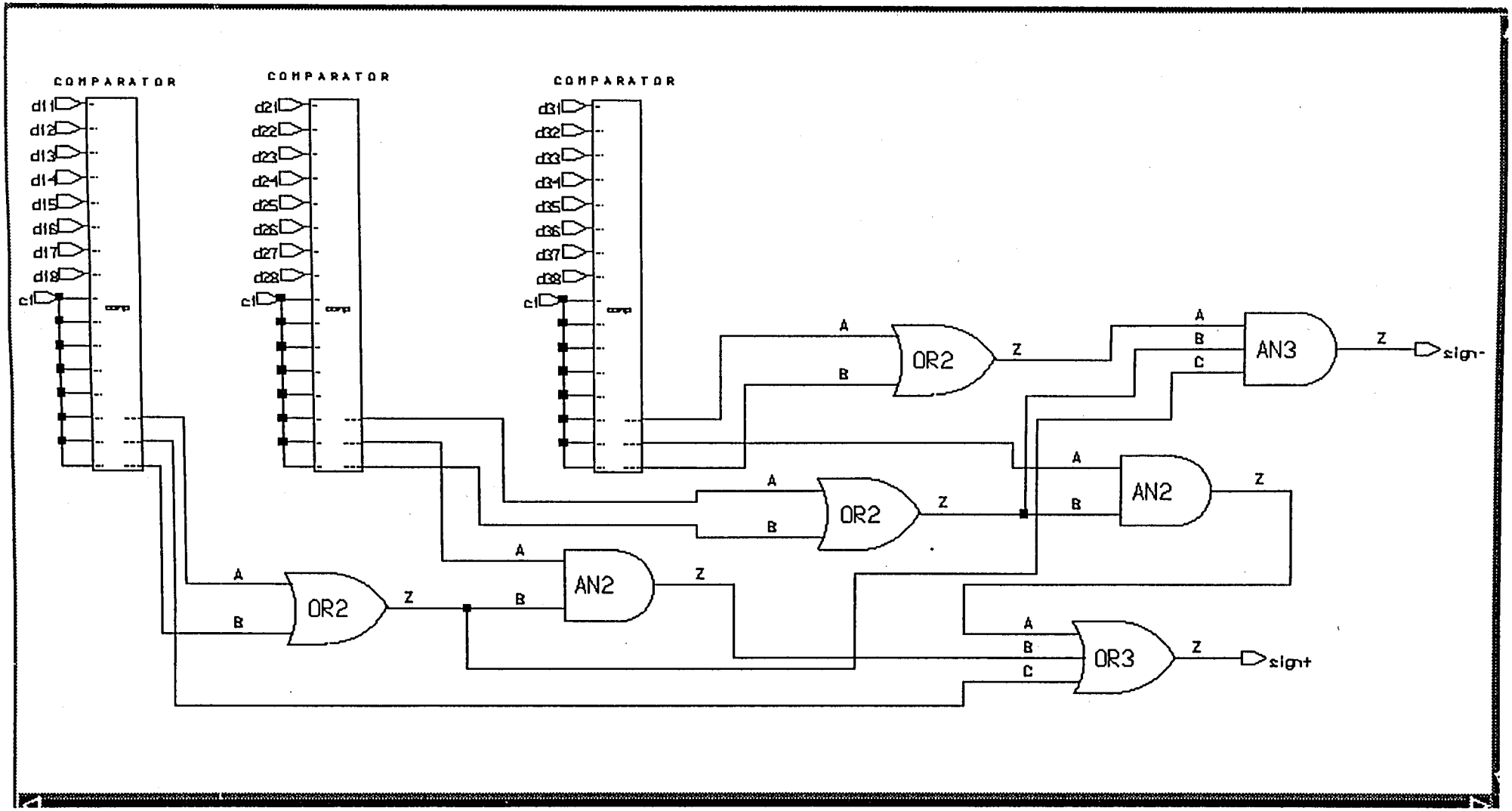

FIGURE A.5. Block diagram of sign check circuit.

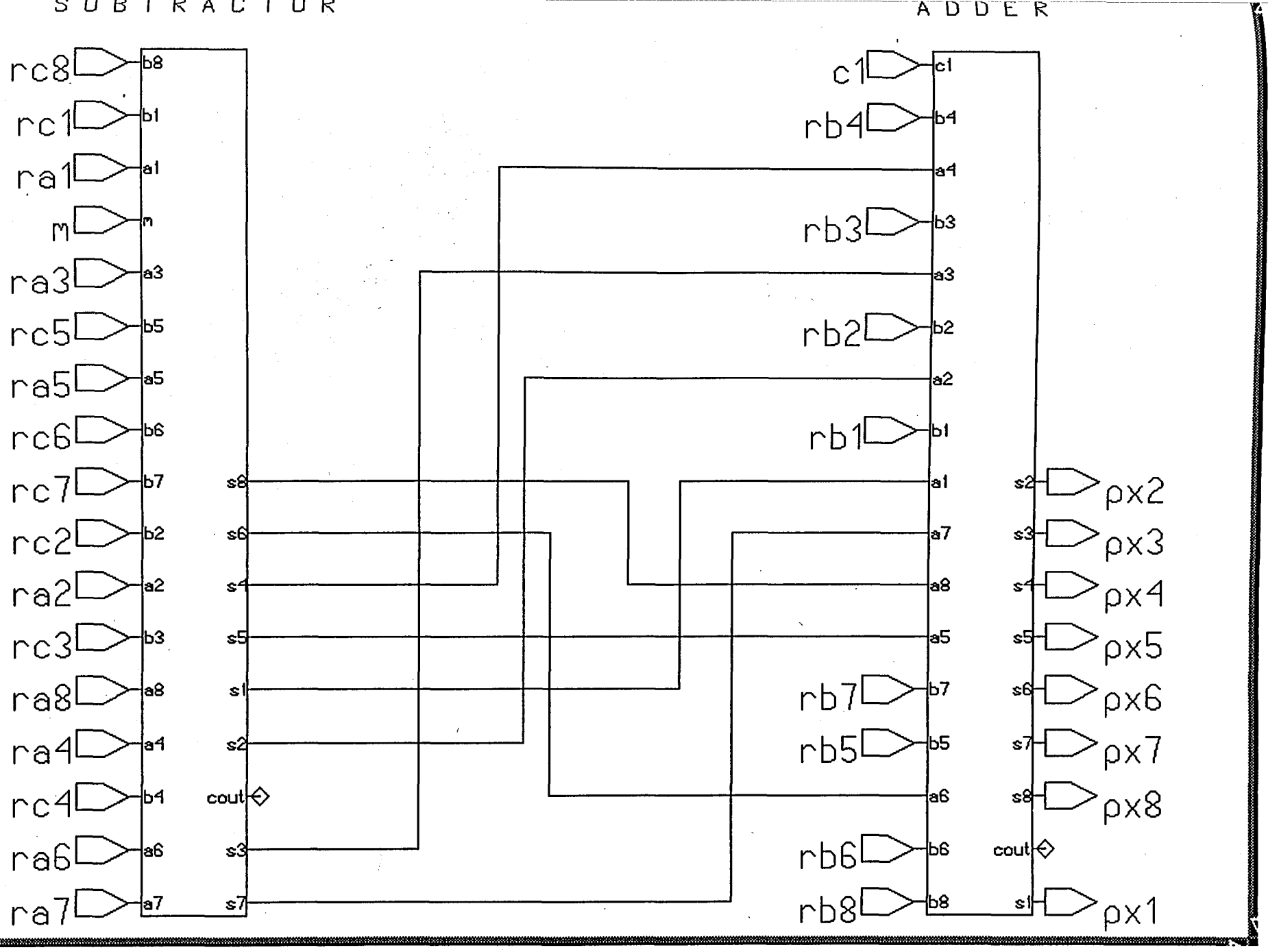

FIGURE A.6. Block diagram of prediction circuit.

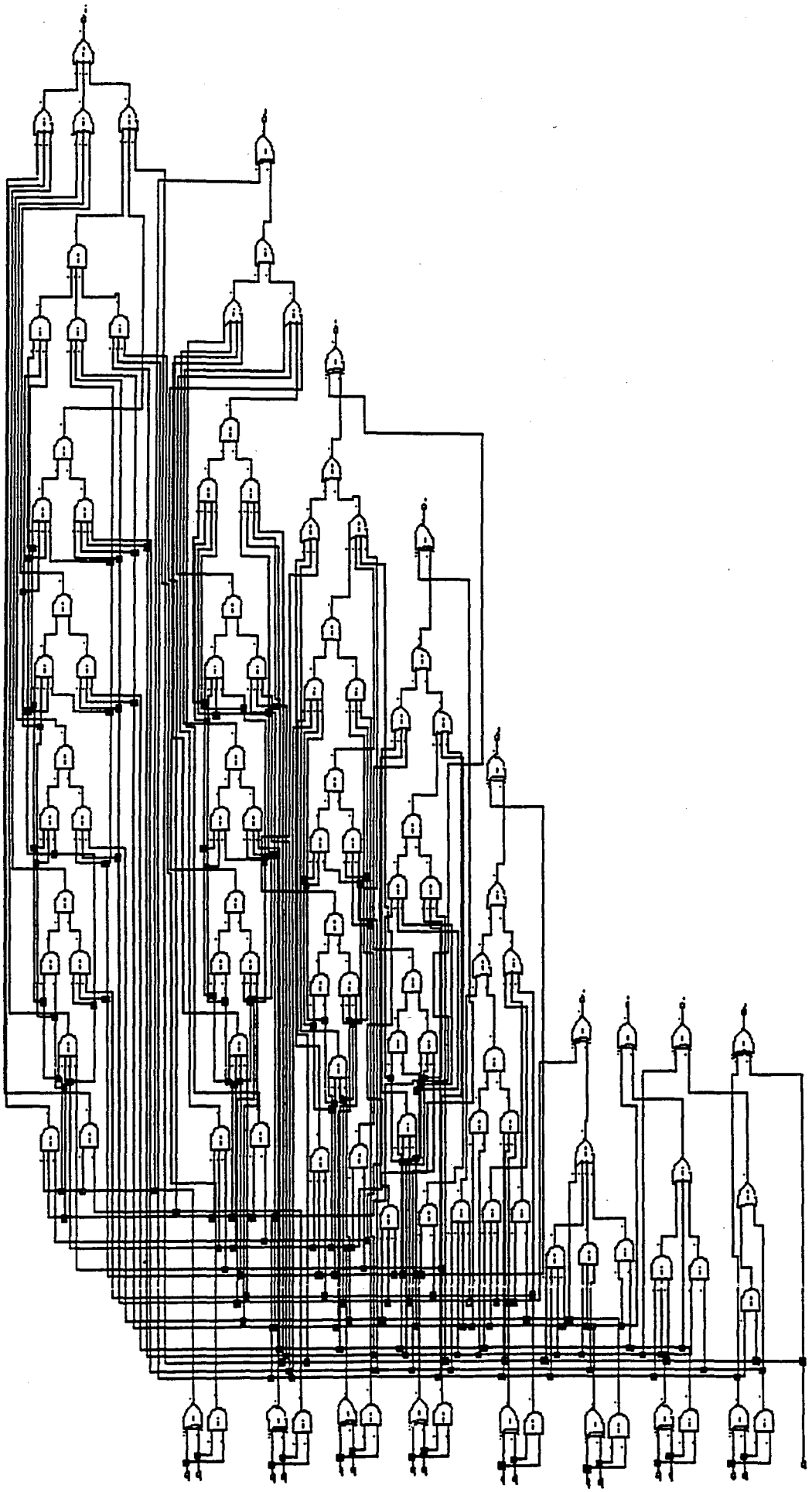

FIGURE A.7. 8-bit CLA(Carry-Look-Ahead) adder.

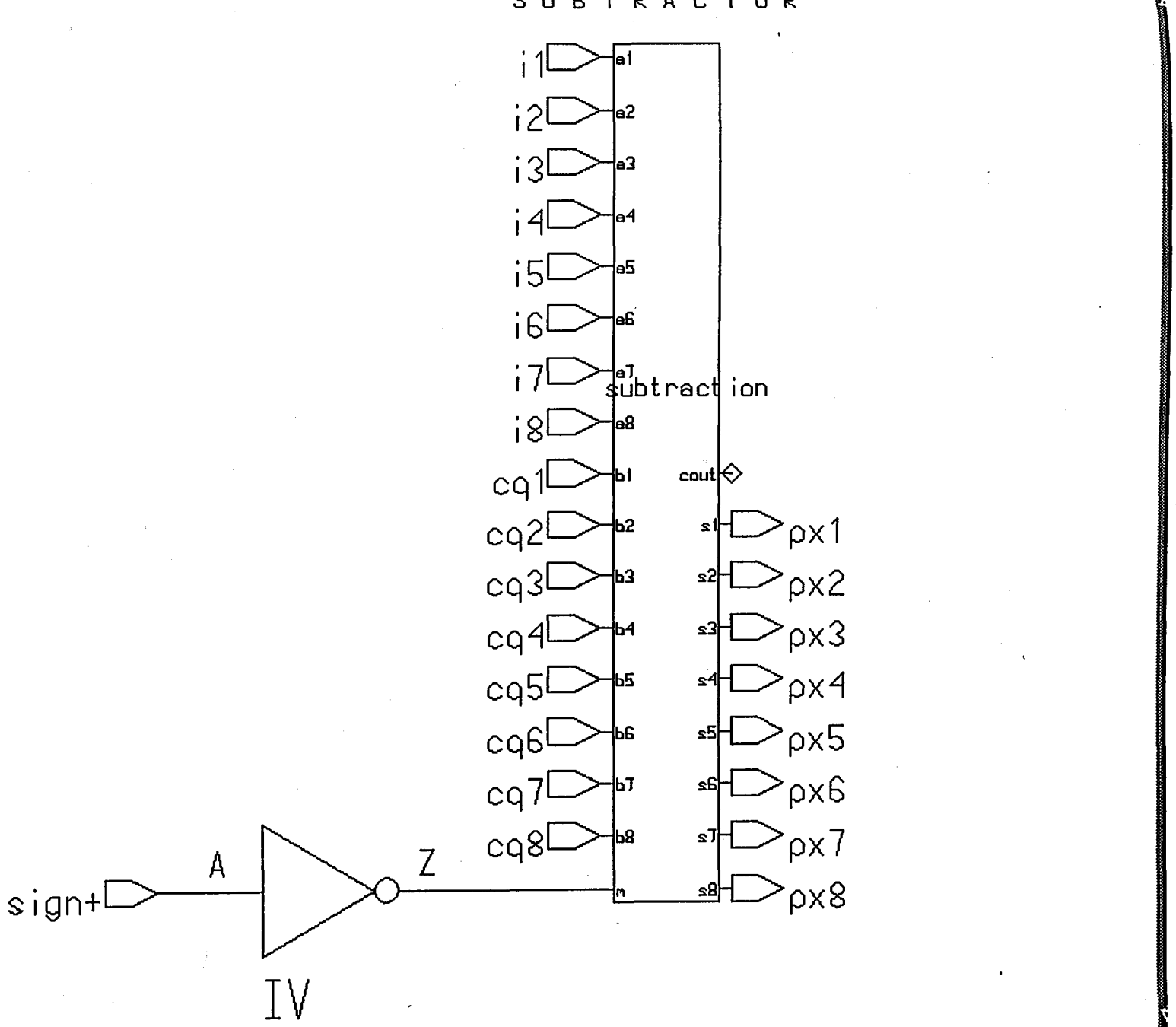

FIGURE A.8. Block diagram of prediction correction circuit.

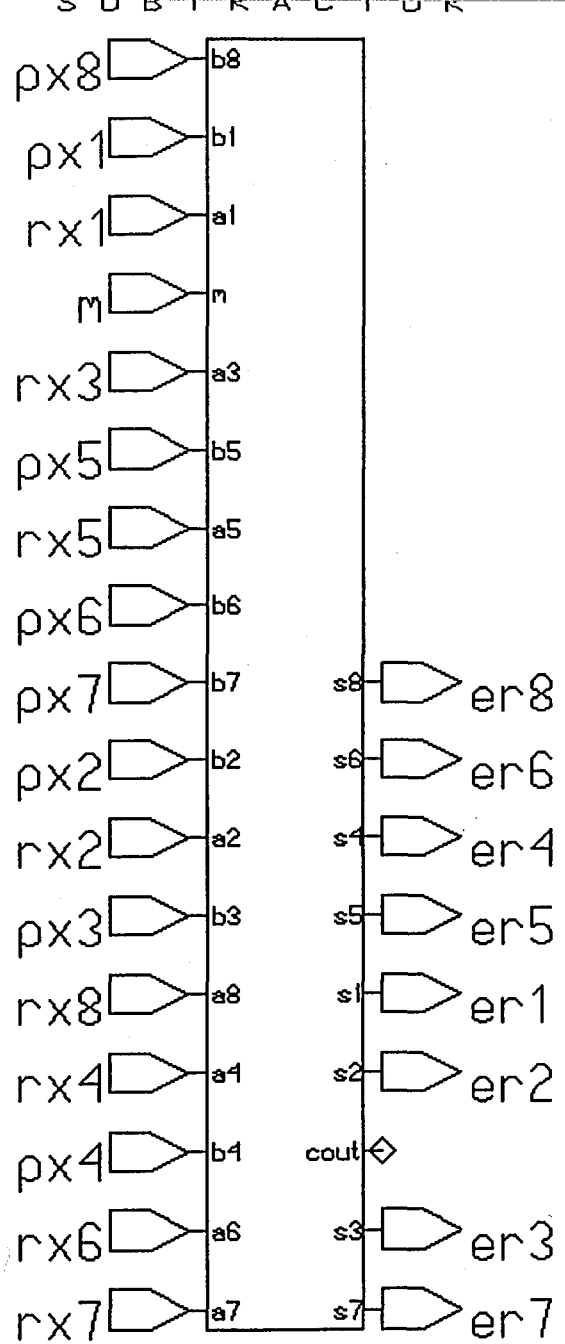

FIGURE A.9. Block diagram of computation of prediction error circuit.

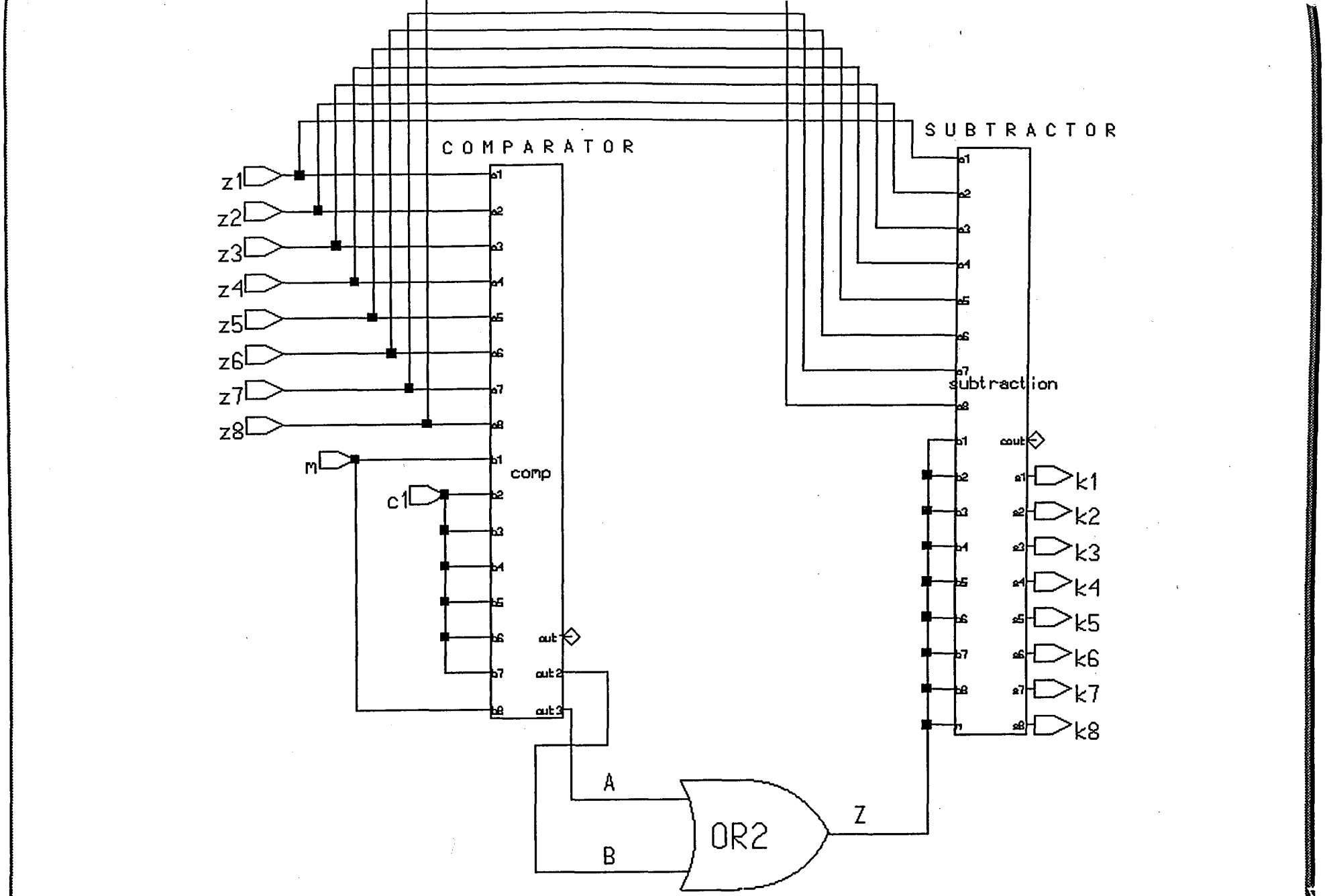

FIGURE A.10. Block diagram of modulo reduction circuit.

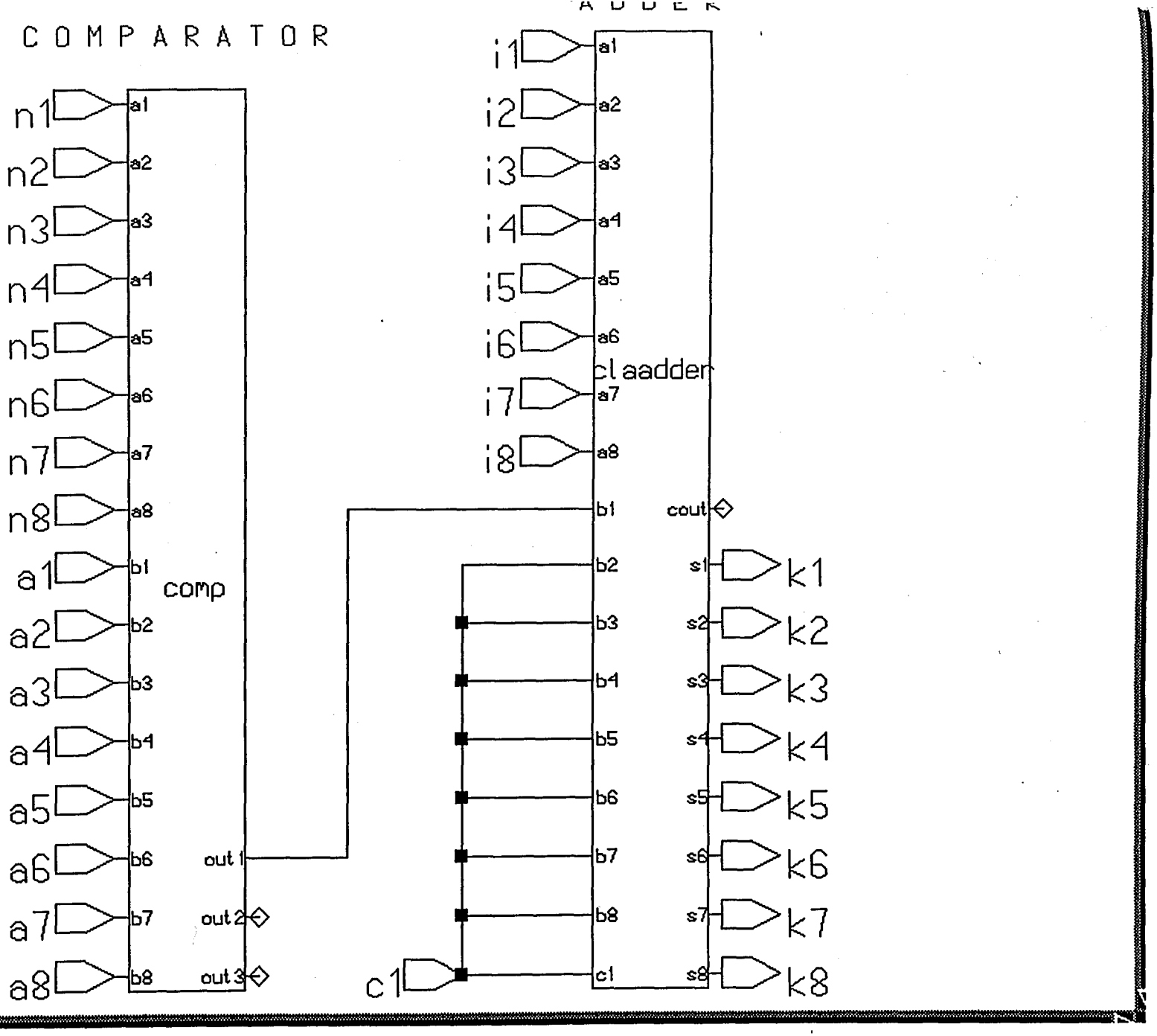

FIGURE A.11. Block diagram of estimation of Golomb parameter circuit.

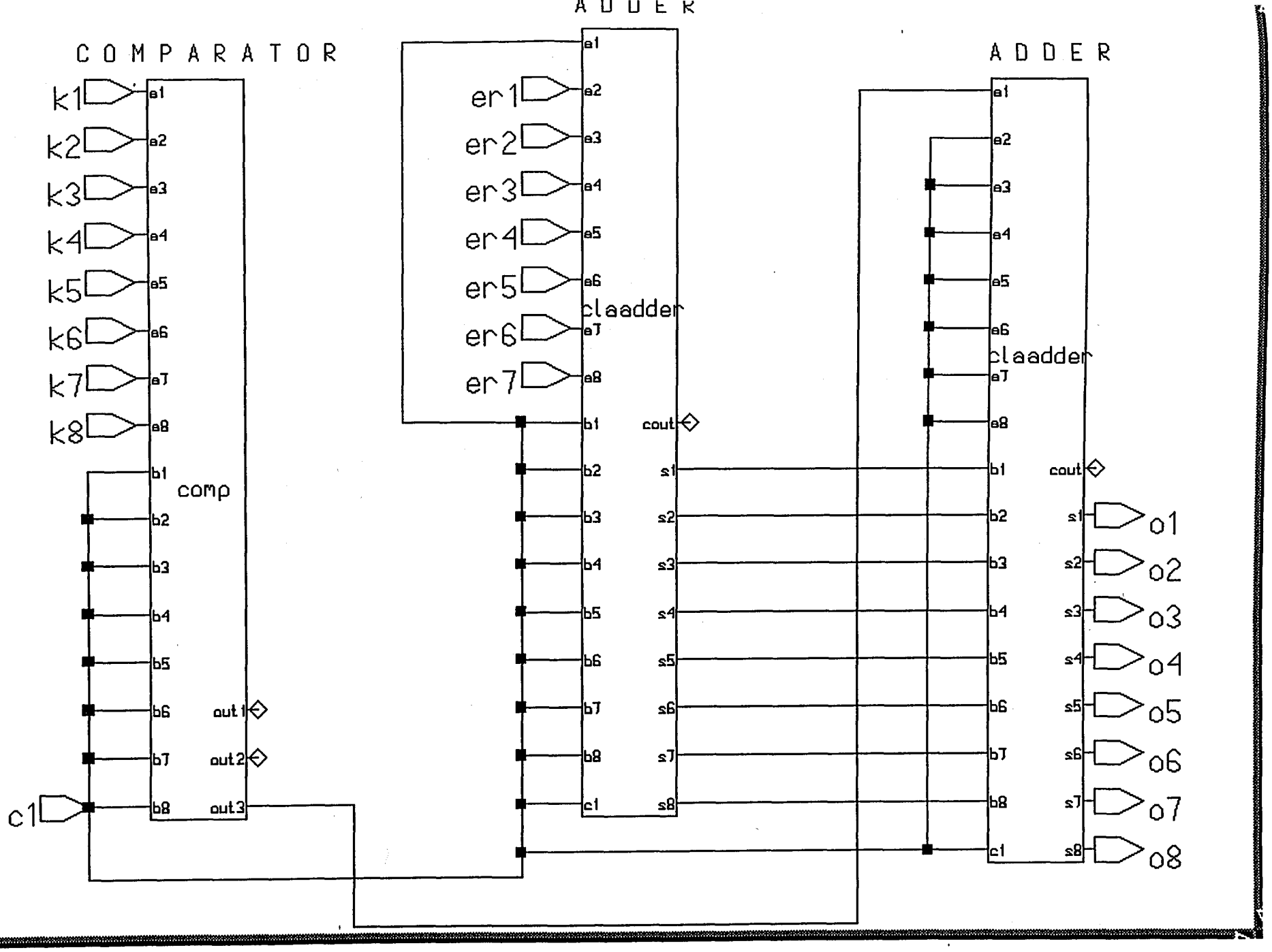

FIGURE A.12. Block diagram of error mapping circuit.

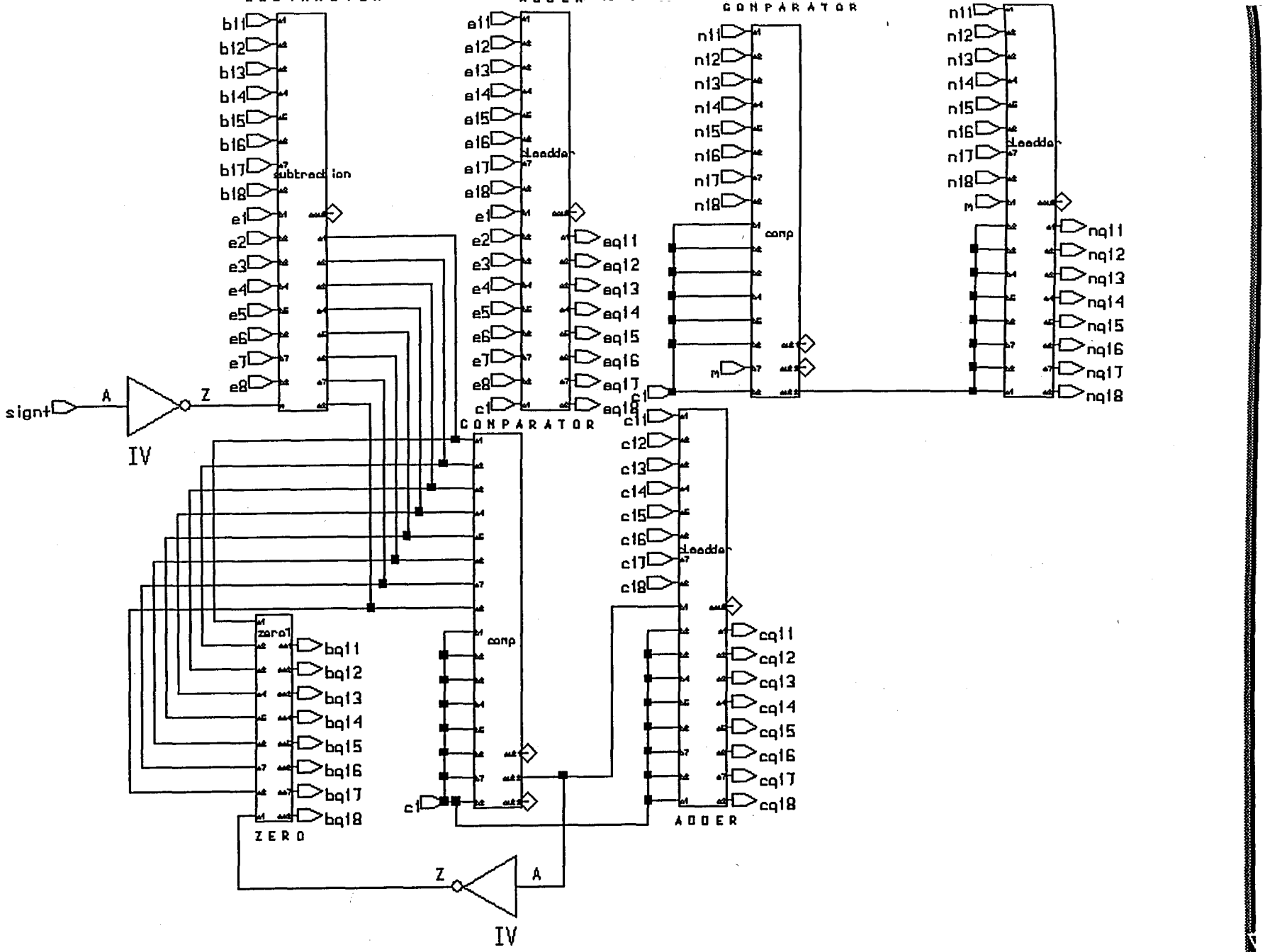

FIGURE A.13. Block diagram of parameters update circuit.

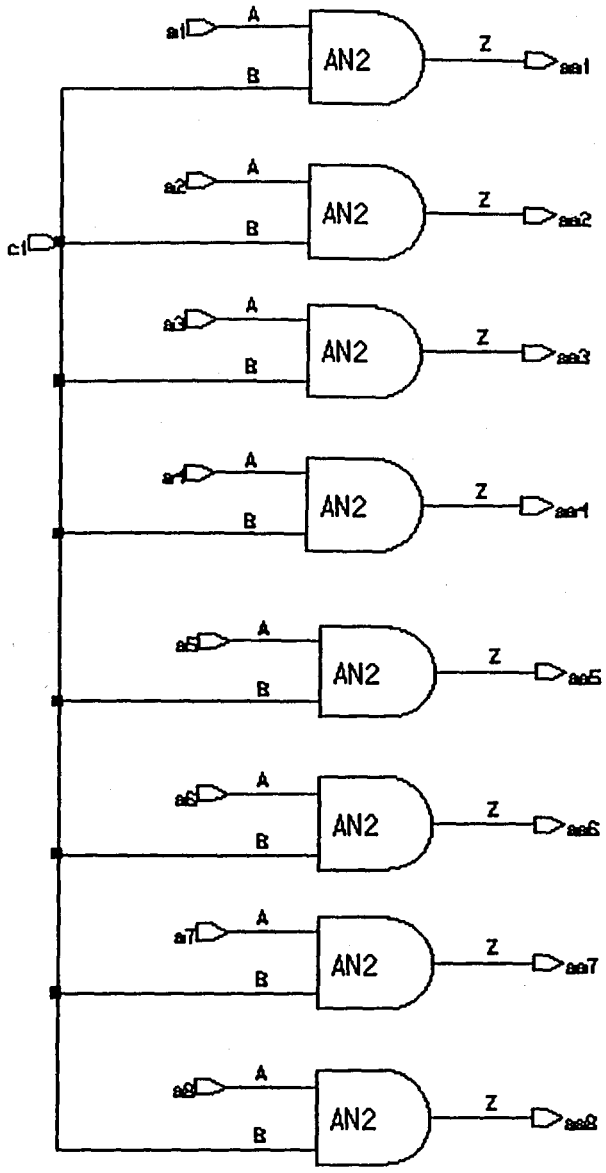

FIGURE A.14. Block diagram of setting all inputs to zero circuit.

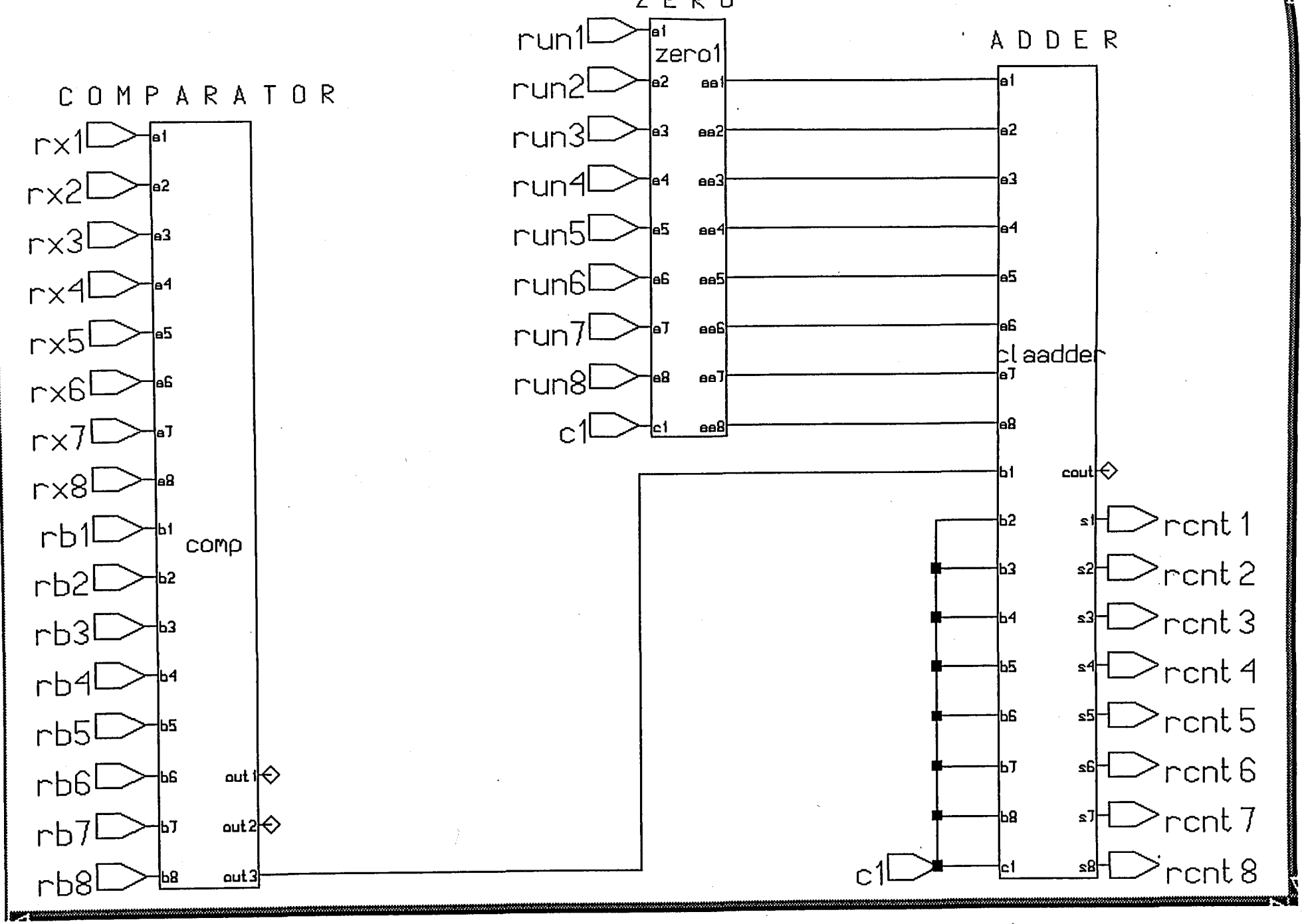

FIGURE A.15. Block diagram of run-length scanning circuit.

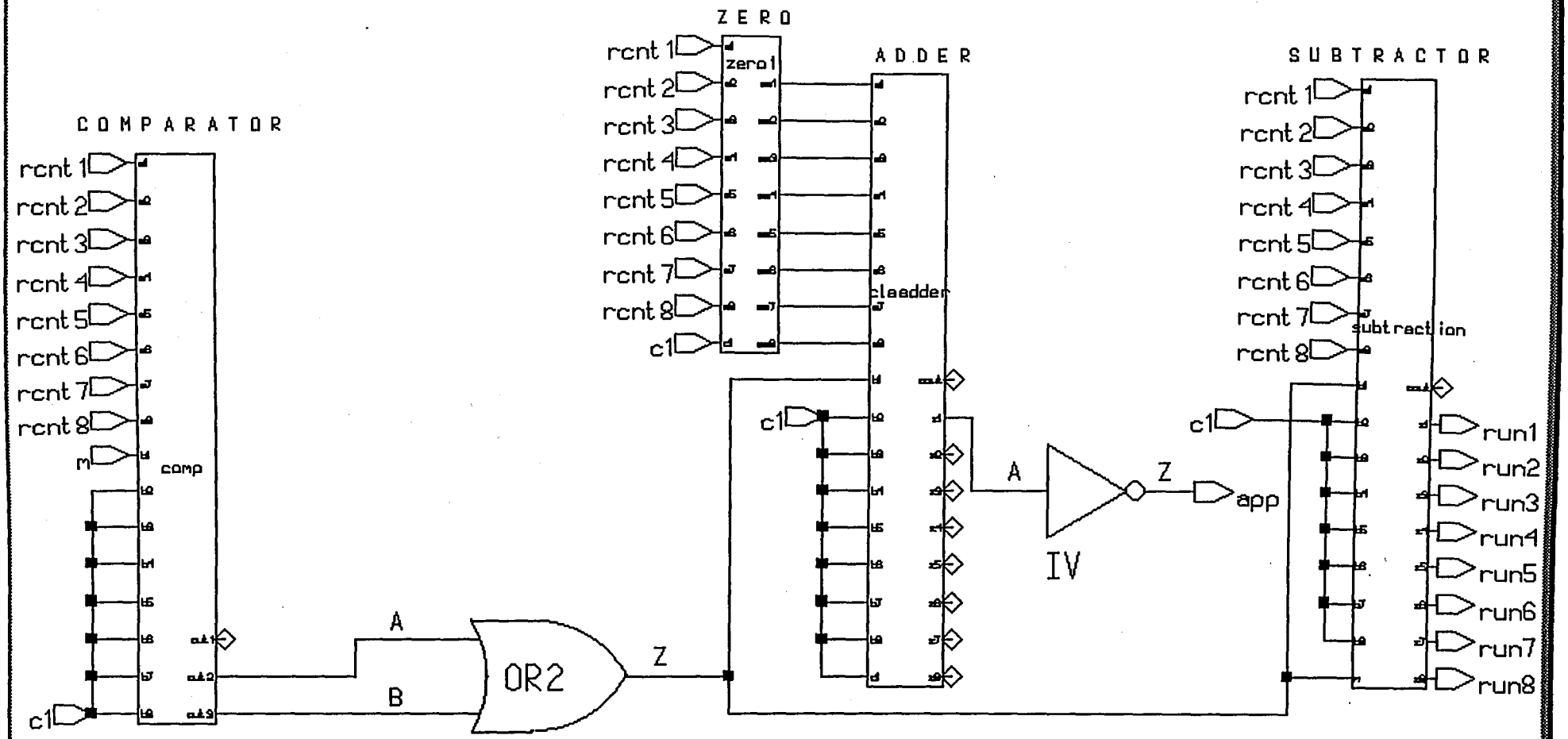

FIGURE A.16. Block diagram of run-length encoding circuit.

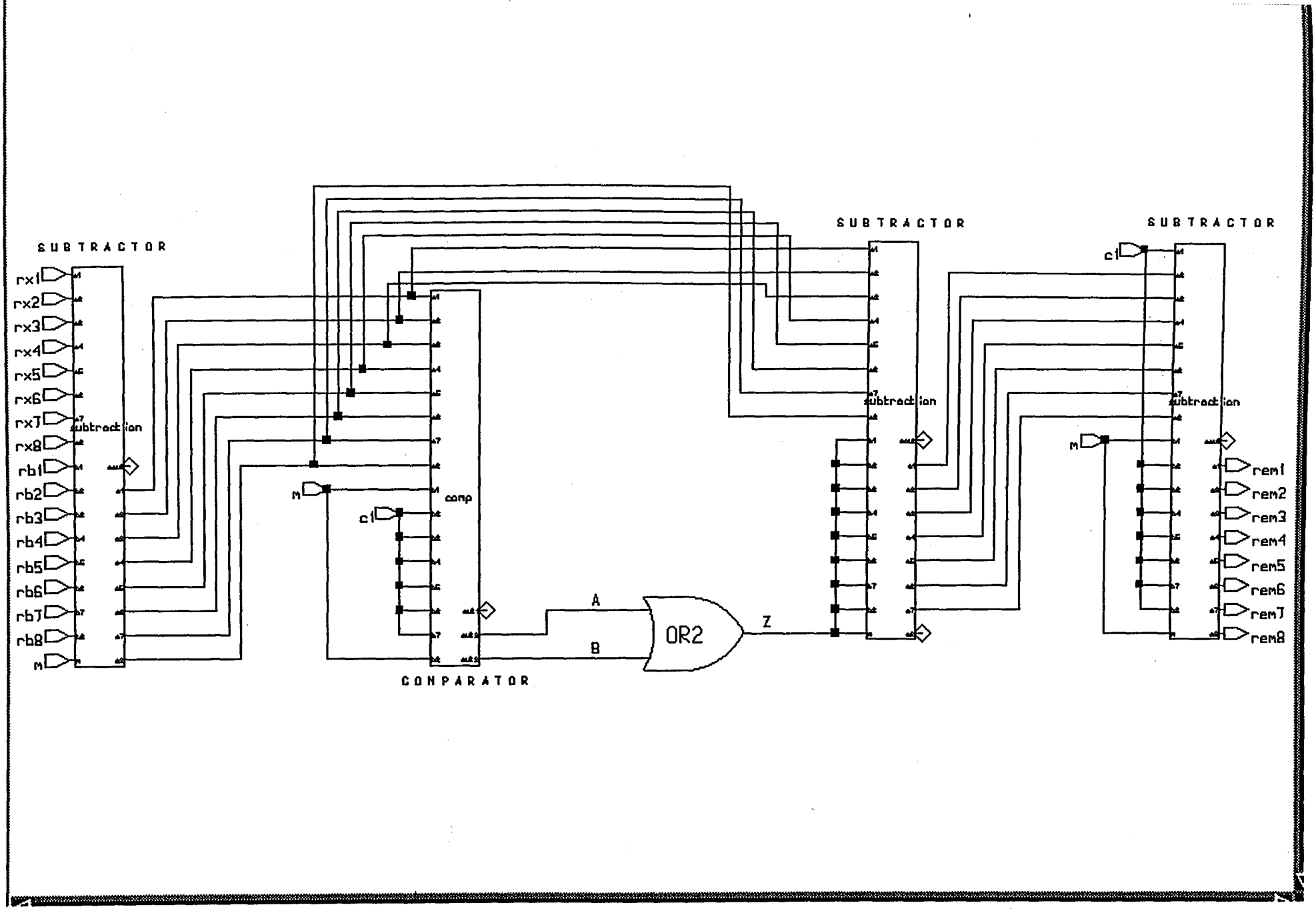

FIGURE A.17. Block diagram of run interruption sample encoding circuit.

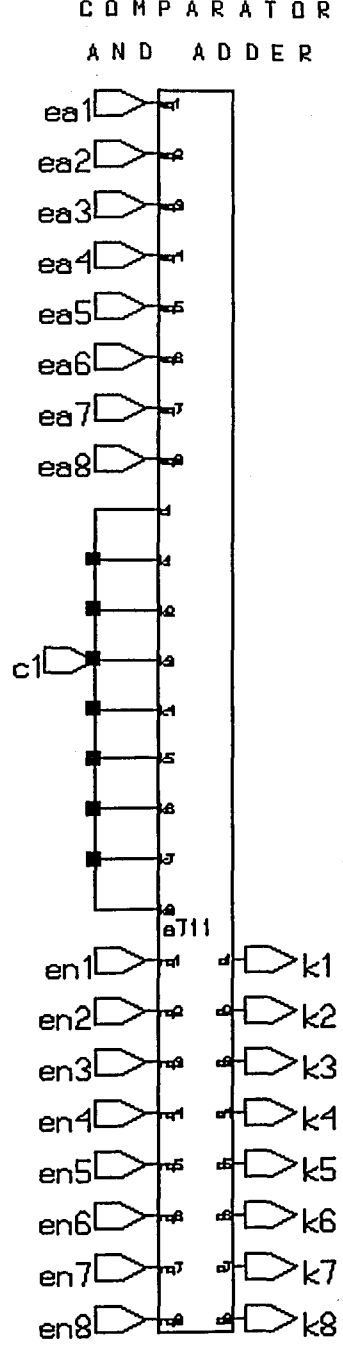

FIGURE A.18. Block diagram of estimation of Golomb parameter circuit in run mode.

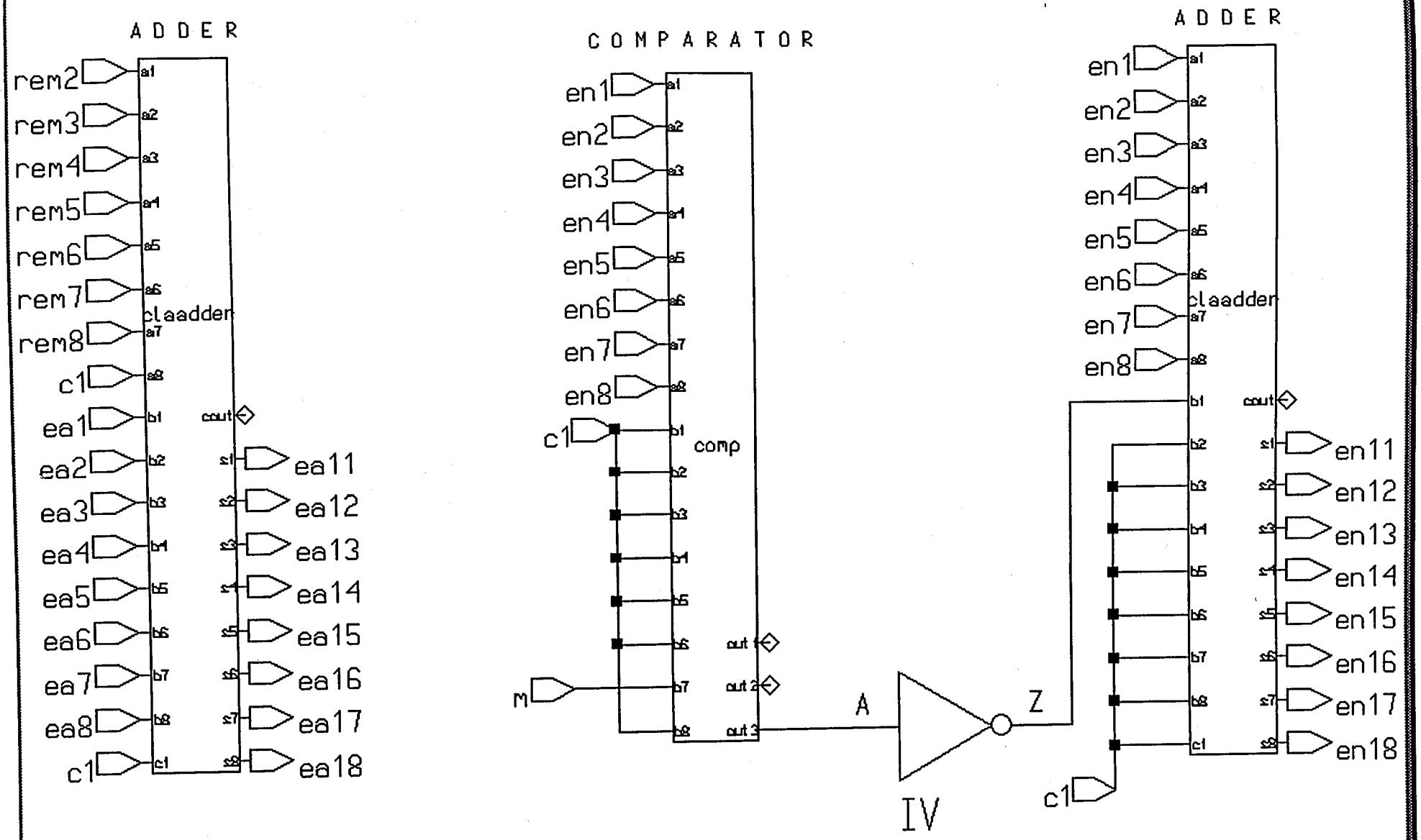

FIGURE A.19. Block diagram of parameters update circuit in run mode.

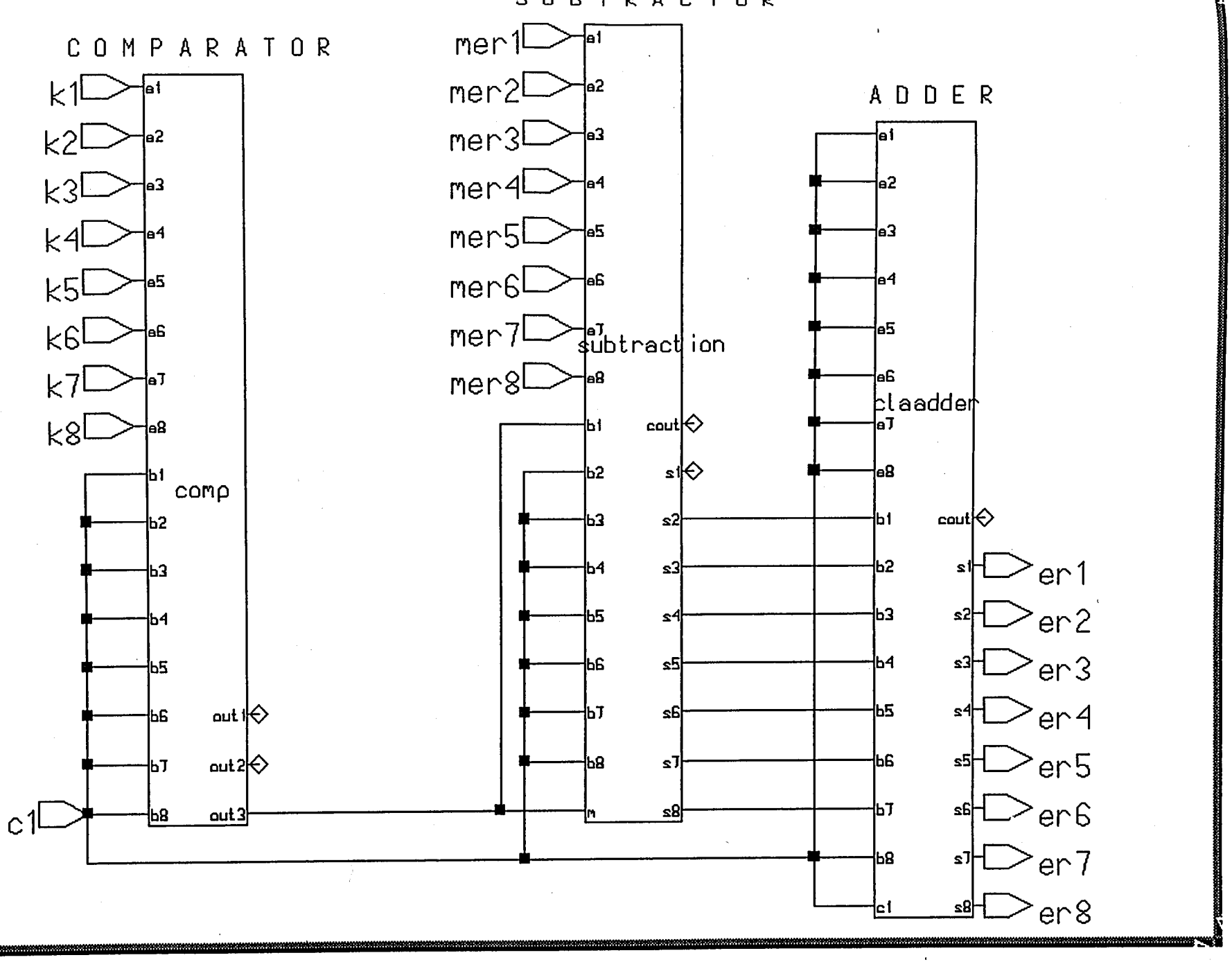

FIGURE A.20. Block diagram of inverse of error mapping circuit.

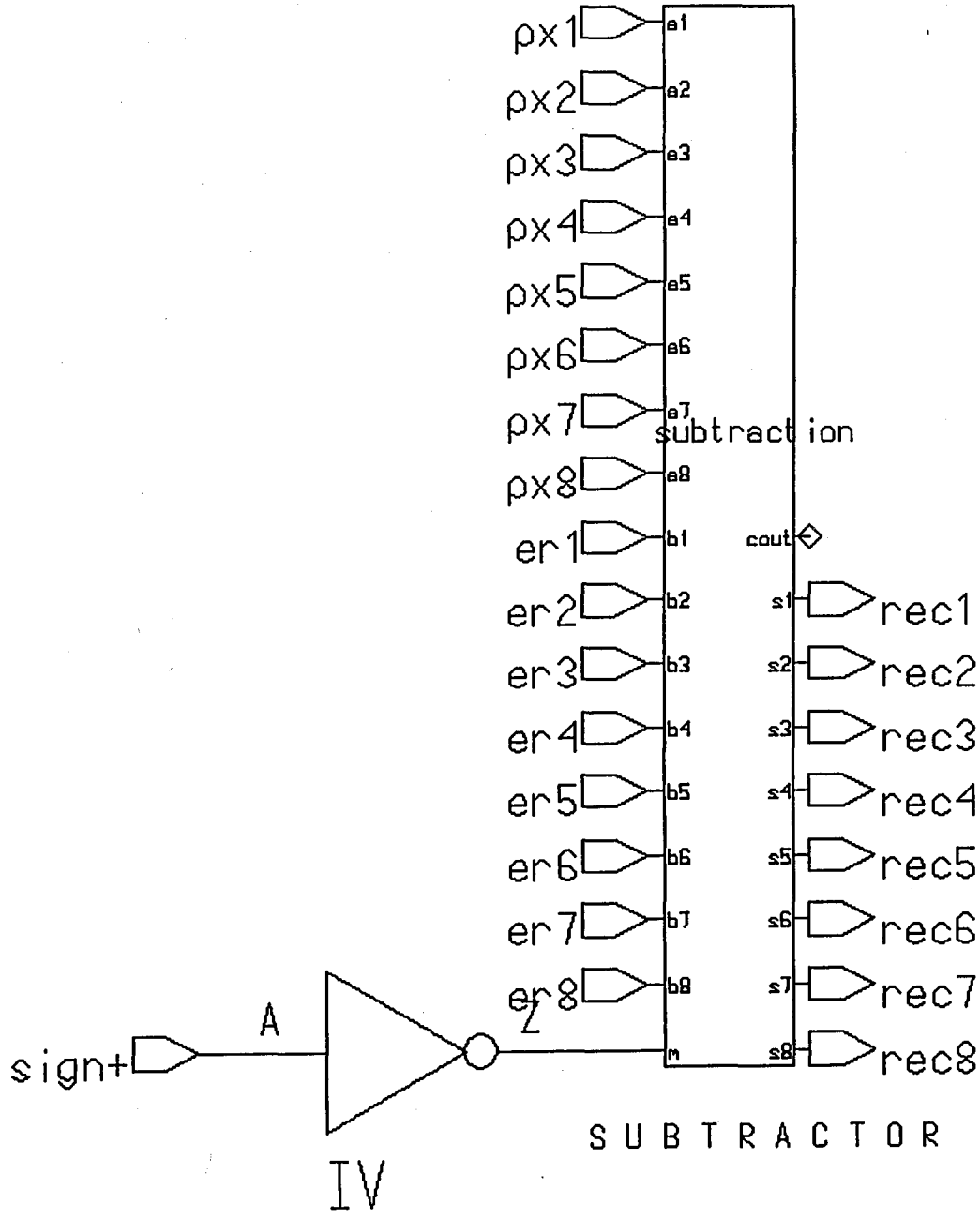

FIGURE A.21. Block diagram of computation of reconstructed value circuit.

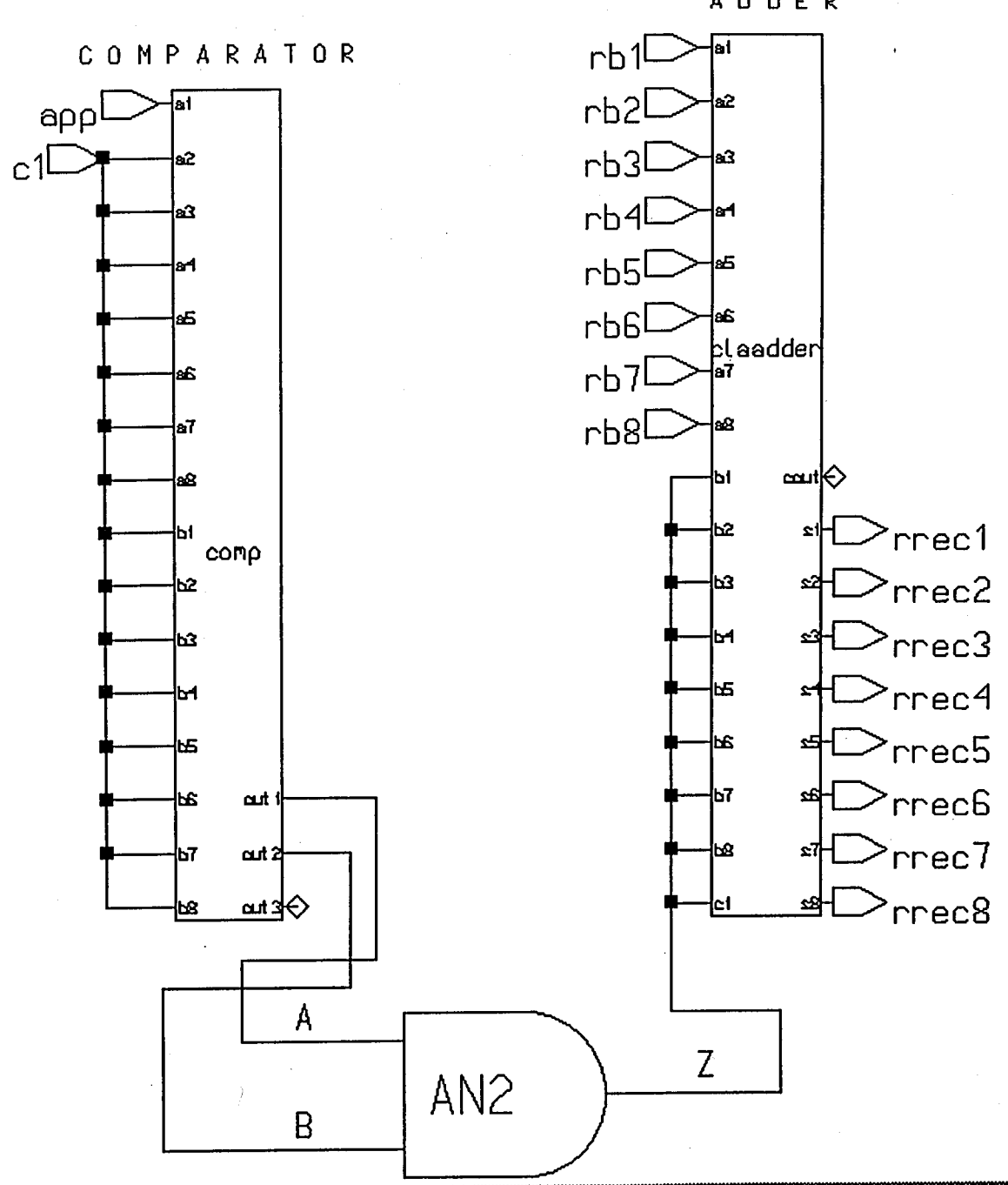

FIGURE A.22. Block diagram of run-length decoding circuit.

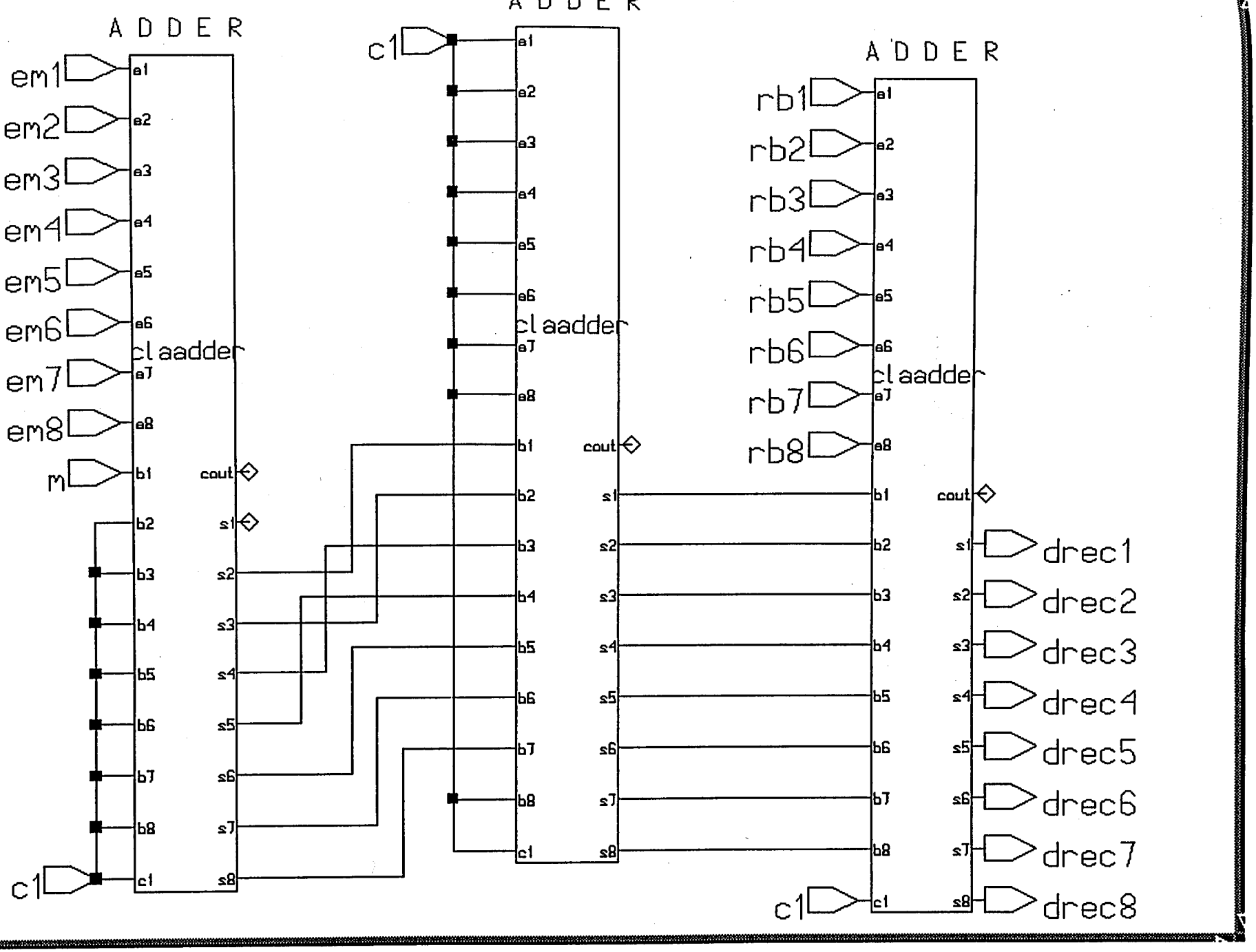

FIGURE A.23. Block diagram of run interruption sample decoding circuit.

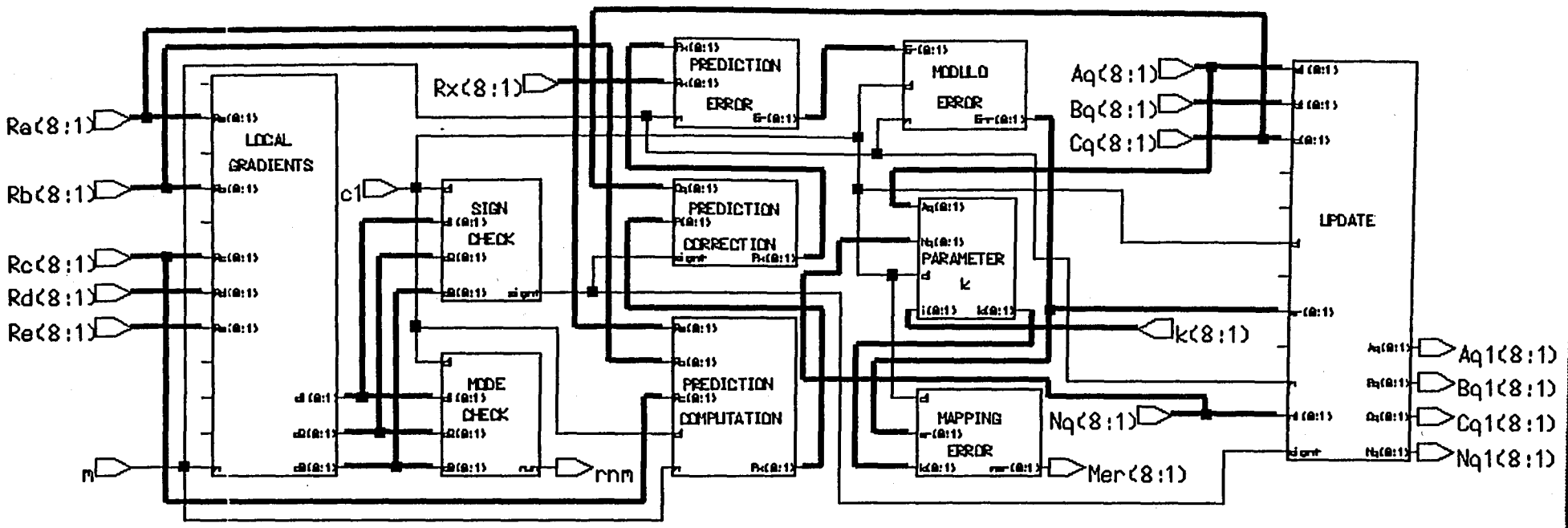

FIGURE A.24. Block diagram of encoding process in regular mode.

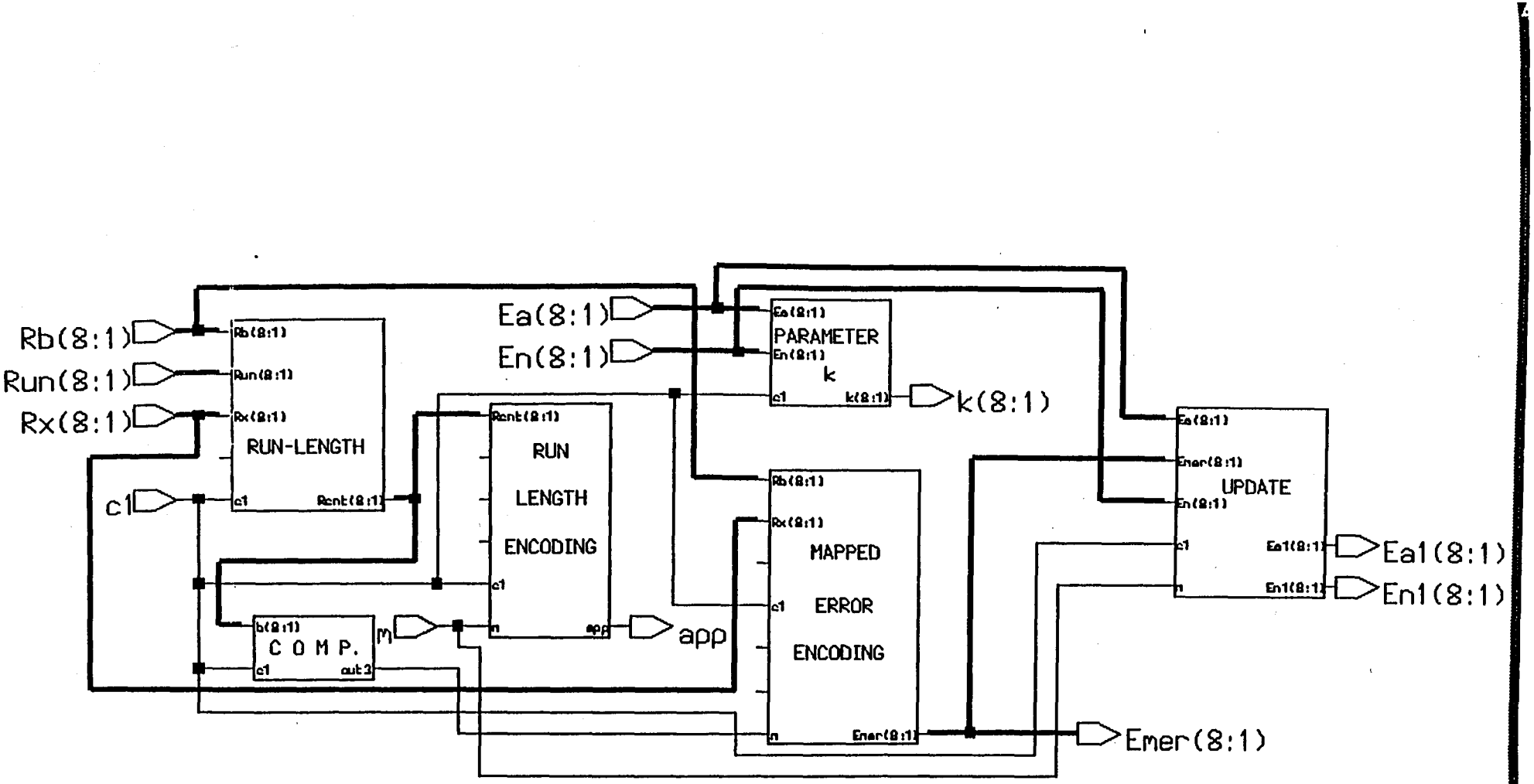

FIGURE A.25. Block diagram of encoding process in run mode.

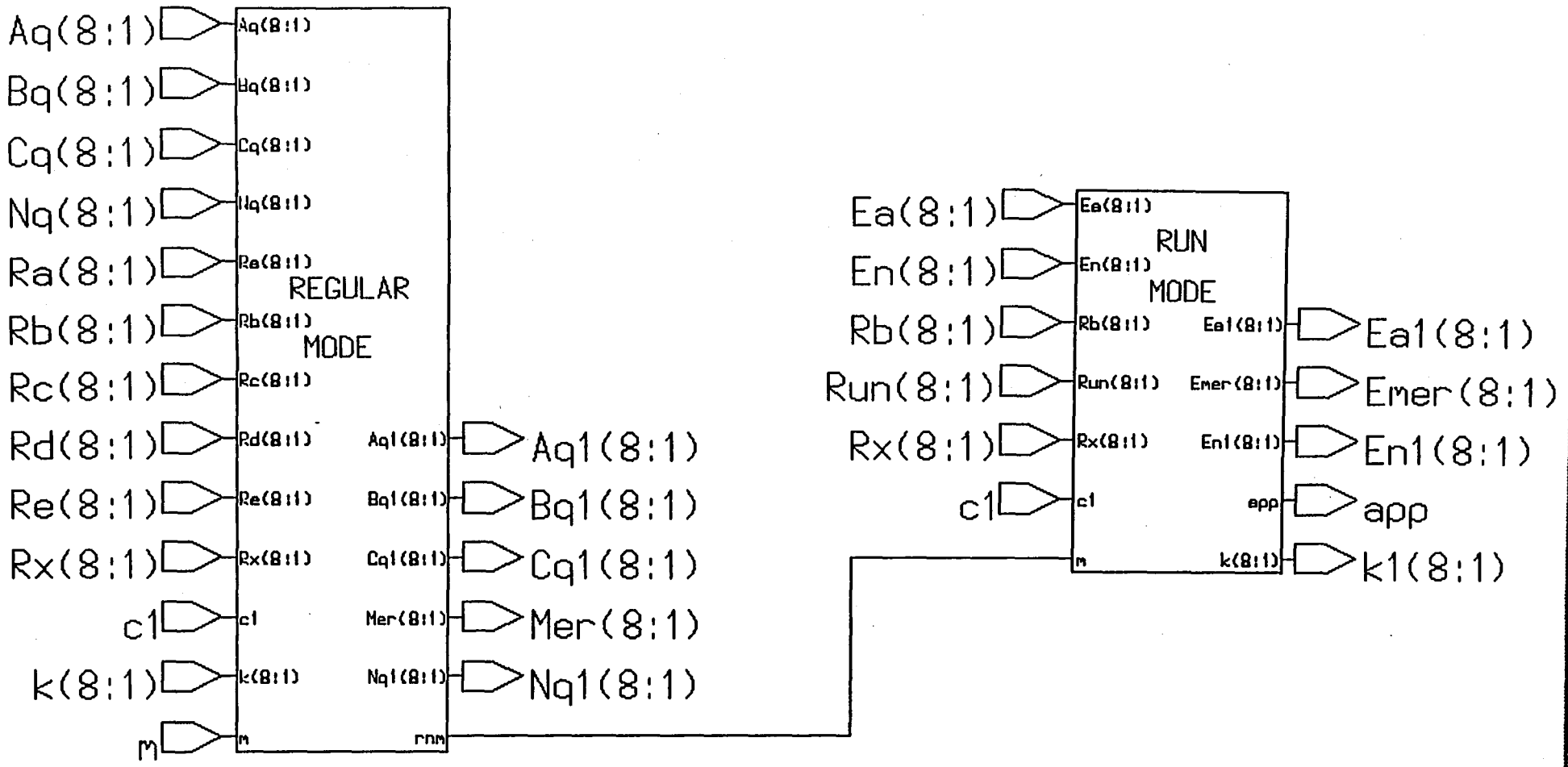

FIGURE A.26. Top level encoding schematic diagram.

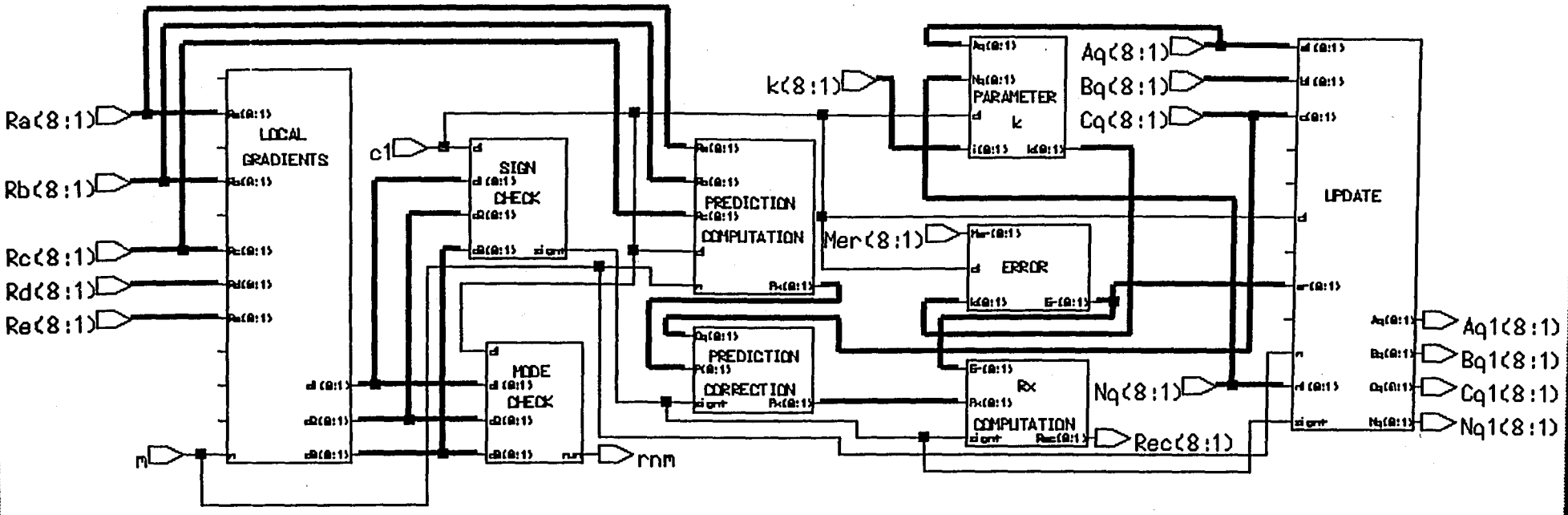

FIGURE A.27. Block diagram of decoding process in regular mode.

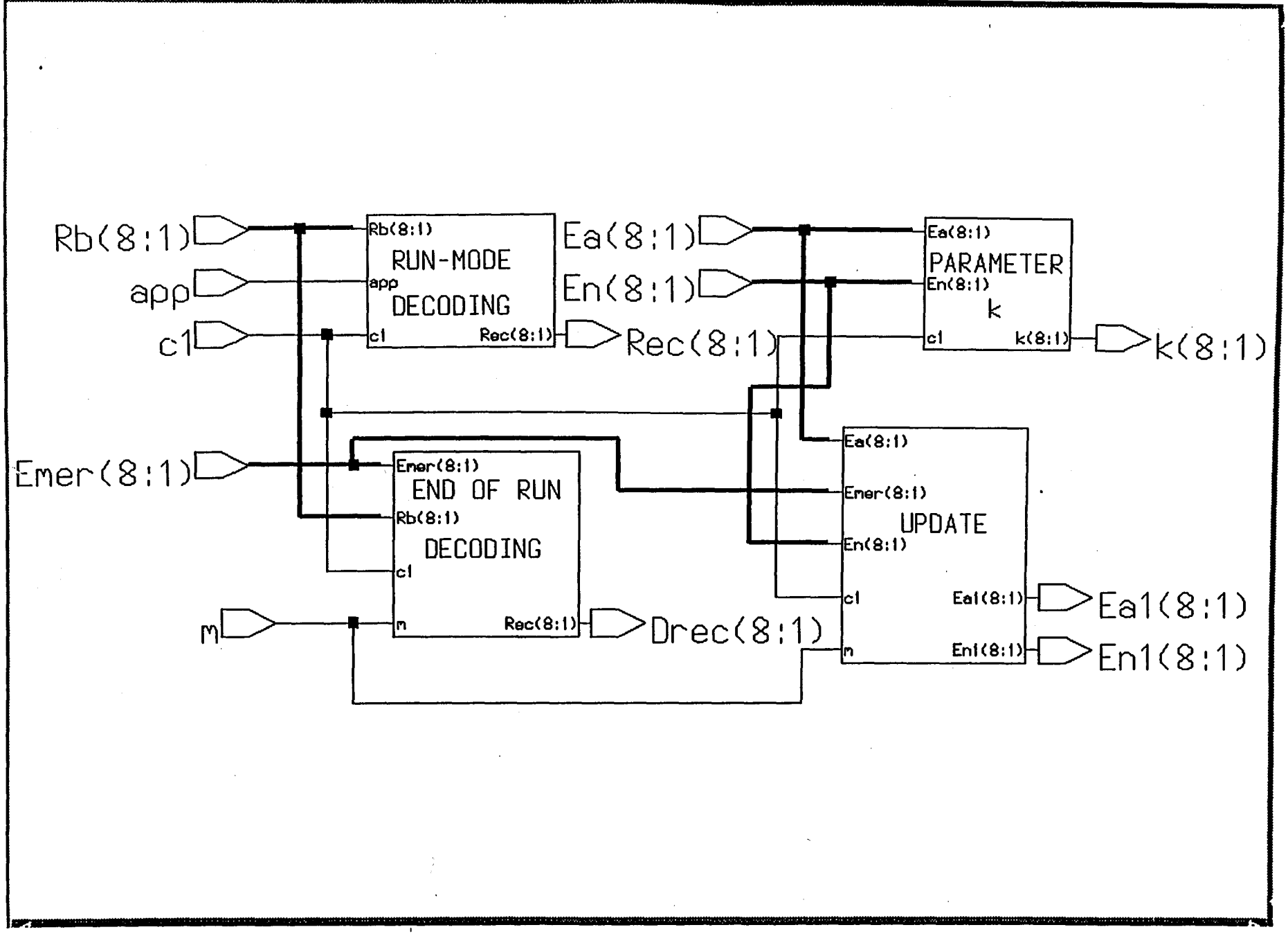

FIGURE A.28. Block diagram of decoding process in run mode.

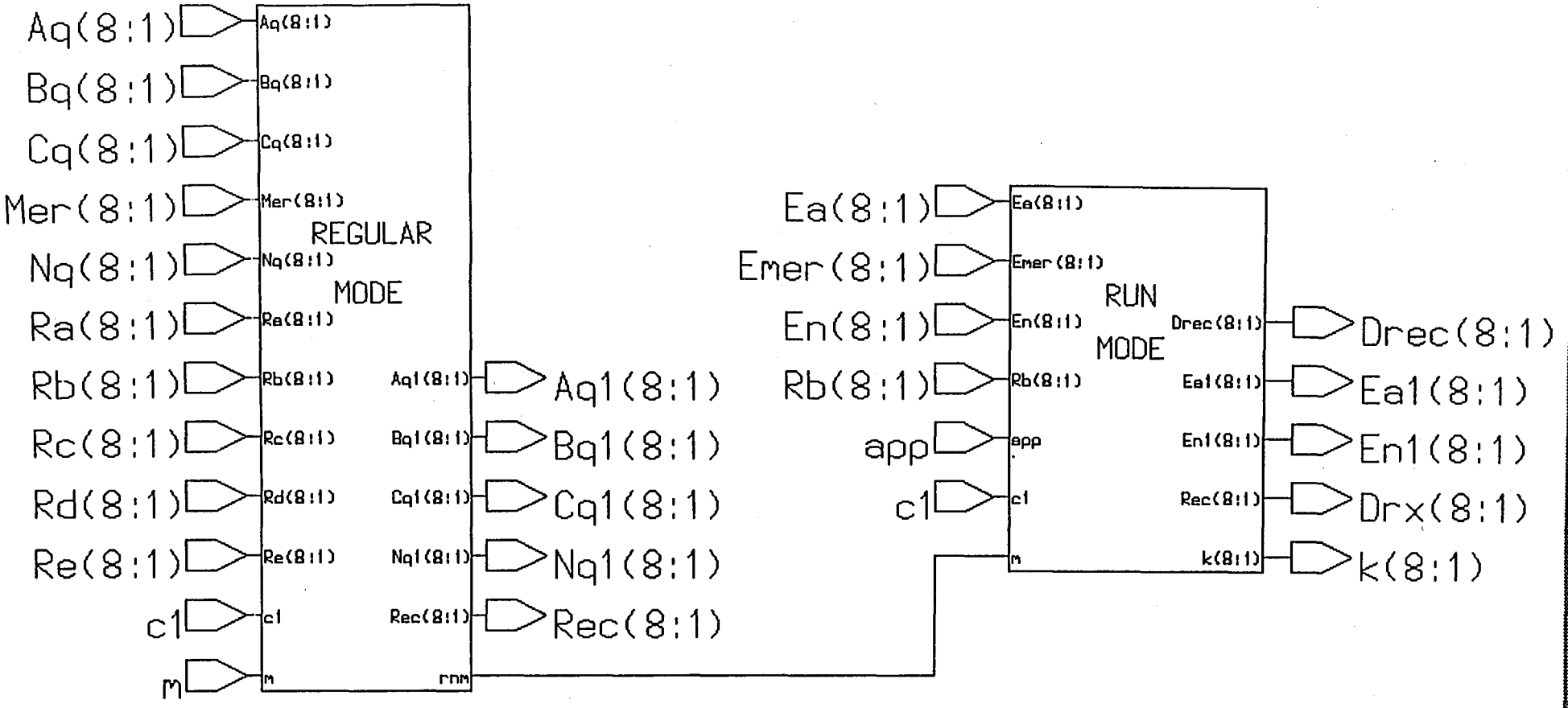

FIGURE A.29. Top level decoding schematic diagram.

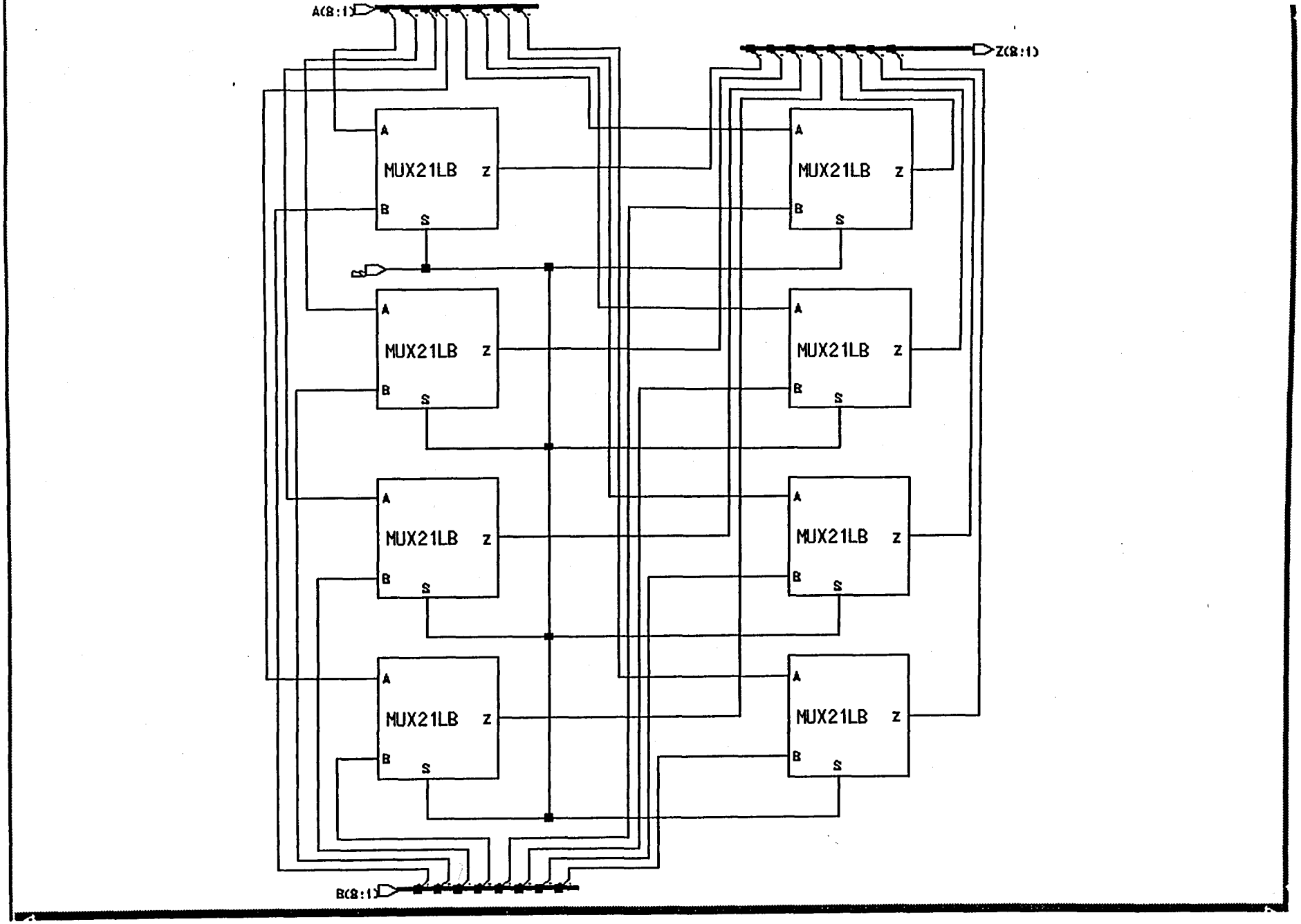

FIGURE A.30. Block diagram of choosing one of the two 8-bit samples circuit.

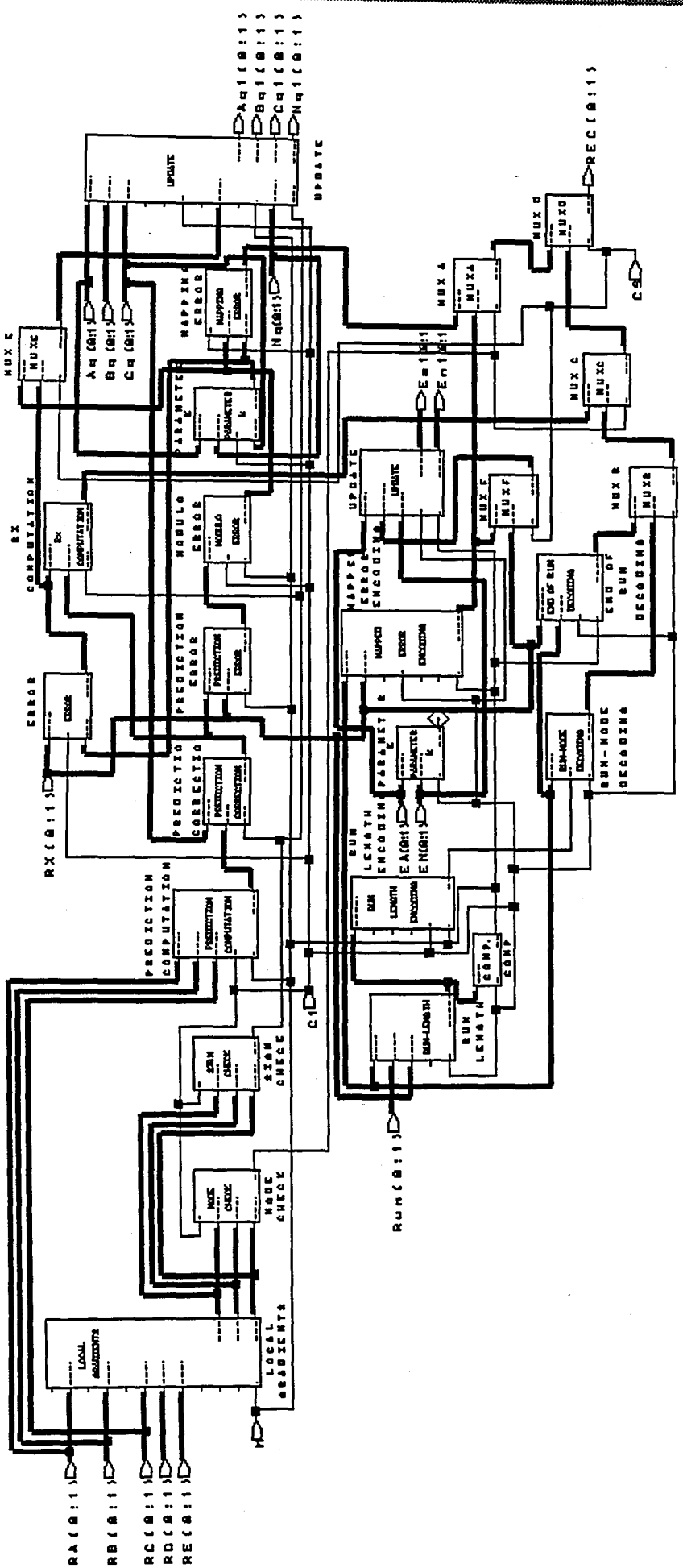

FIGURE A.31. Top level coding schematic diagram.

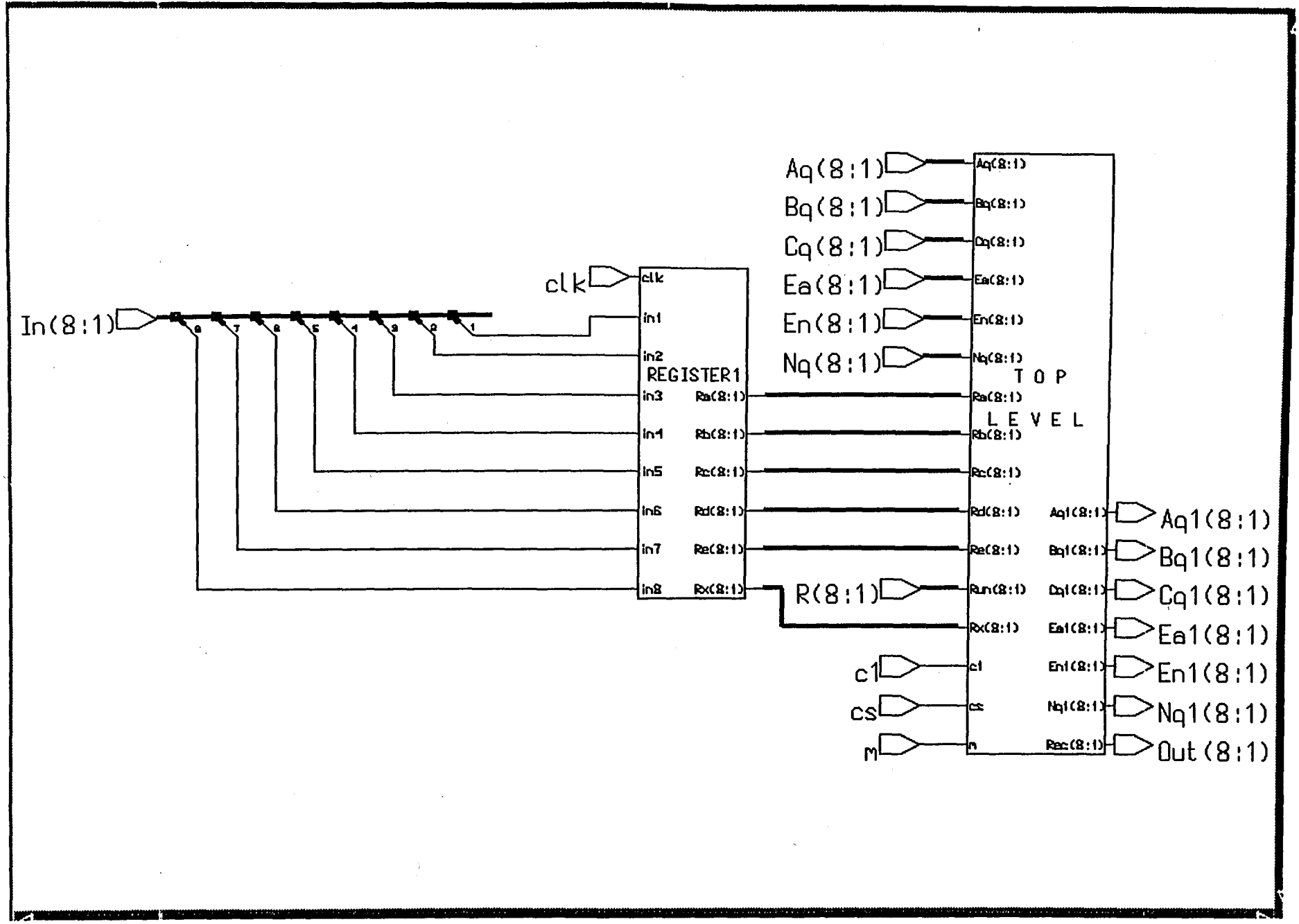

FIGURE A.32. Top level schematic diagram with register.

APPENDIX B

DEFINITIONS

Bias:

Deviation from zero of average accumulated prediction errors.

Bit stream:

Partially encoded or decoded sequence of bits comprising a coded segment.

Causal template:

Fixed set of previously coded values according to a pre-specified scan sequence.

Color image:

A continuous-tone image that has more than one component.

Component:

One of the two-dimensional arrays which comprise an image.

Compressed image data:

A coded representation of an image.

Compression:

Reduction in the number of bits used to represent source image data.

Context:

Fixed set of previously coded samples used to code the present sample.

Context modeling:

Process to determine coding distribution from the context.

Context-type:

One of the 1092 possible different quantized contexts.

Continuous-tone image:

An image whose components have more than one bit per sample.

Decoder:

An embodiment of decoding process.

Decoding process:

A process which takes as its input compressed image data and outputs a continuous-tone image.

Digital reconstructed image:

An image which is the output of the decoder.

Digital source image:

An image which is the input to the encoder.

Encoder:

An embodiment of encoding process.

Encoding process:

A process which takes as its input a source image and outputs compressed image data.

End of run mode:

Mode of operation when a run is term at other than by the end of the current image row.

Golomb coding:

A procedure which assumes a geometric distribution and is a particular case of Huffmann coding.

Huffmann coding:

A procedure which assigns a variable length code to each input sample, so that the total code length is minimized.

Image value:

A non-negative integer number indicating the image information in an image sample.

Lossless:

A description term for encoding and decoding processes and procedures in which the output of the decoding procedure is identical to the input to the encoding procedure.

Near-lossless:

A description term for encoding and decoding processes and procedures in which the output of the decoding procedure is such that each image sample differs from the corresponding one in the input to the encoding procedure by not more than a pre-specified value.

Prediction:

A function of previously reconstructed values.

Prediction error:

Difference between the actual value of the current sample and the predicted one.

Regular mode:

Mode of operation while coding samples in a non-run mode.

Run:

A sequence of consecutive samples with identical value.

Run-length:

Number of samples in a run.

Run mode:

Mode of operation while coding runs of identical values.

Sample:

One element in the two-dimensional array which comprises a component.

Scan:

A single pass through the data for one or more of the components in the image.

Sequential coding:

One of the lossless or near-lossless coding processes in which each component of the image is encoded within a single scan.

Unary code:

The unary code of an integer number m is composed of m zeros followed by a one.

REFERENCES

1. Sayood,K., *Introduction to Data Compression*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

2. Pennebaker,W. B., and J. L. Mitchell, *JPEG: Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, 1993.

3. Memon,N. D., and K. Sayood, "Lossless image compression: A comparative study," *Proceedings SPIE*, Vol. 2418, pp. 8-20, February 1995.

4. Rabbani,M., and P. Jones, *Digital Image Compression Techniques*, Tutorial Texts in Optical Engineering, Vol. TT7, SPIE Press, 1991.

5. Tekalp,A.M., *Digital Video Processing*, Prentice-Hall, New York, 1995.

6. Wallace,G.K., *The JPEG Still Picture Compression Standard*, Communications of the ACM, 34(4) : 31-44, 1997.

7. Rissanen,J., "A universal data compression system," *IEEE Transactions on Information Theory*, Vol. 29, pp. 656-664, 1983.

8. Weinberger,M. J., J. Rissanen and R. Arps, "Applications of universal context modeling to lossless compression of gray-scale images," *IEEE Transactions on Image Processing*, Vol. 5, pp. 575-586, April 1996.

9. Langdon,G.G., "Sunset: A hardware-oriented algorithm for lossless compression of gray-scale images," *Proceedings SPIE Medical Imaging V: Image Capture, Formatting, Display*, Vol. 1444, pp. 272-282, May 1991.

10. Weinberger,M. J., G. Seroussi and G. Sapiro, "LOCO-I: A low complexity, context-based, lossless image compression algorithm," *Proceedings Data Compression Conference*, Snowbird, Utah, April 1996.

11. Todd,S., G. G. Langdon, and J. Rissanen, "Parameter reduction and context selection for compression of gray-scale images," *IBM Journal of Research and Development*, Vol. 29 (2), pp. 188-193, 1985.

12. Gallager,D., and D.V. Voorhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Transactions on Information Theory*, Vol. 21, pp. 228-230, March 1975.

13. Golomb,W.S., *Digital Communications with Space Applications*, Prentice-Hall Series, New York, 1964.

14. Merhav,N., G. Seroussi and M. J. Weinberger, Proceedings *IEEE Internatinal Conference on Image Procesing* 96, Laussanne, Switzerland, September 1996.

15. Morgül,A. and A. Ataman, Televizyon Tekniği, Boğaziçi Üniversitesi Yayınları, 1997.