A Hardware Implementation of the Nordstrom-Robinson Error Correction Code

CIS4930 Final Project

Charles R. Yates and Eric Sowers

January 13th, 1997 0:04

Abstract

In this project, we build a digital system for detecting all errors and correcting a limited subset of errors using the Nordstrom-Robinson error correction code. A glimpse of the theory behind the operations performed is presented, followed by a detailed explanation of the hardware theory and implementation. Finally, chip utilization for this design is examined and a performance analysis performed.

University Of South Florida

Typeset using PCTEX For Windows

Table of Contents

List of Figures

List of Tables

1 Introduction

Any real-world communication system is plagued by noise and thus susceptible to errors in the transmission of information. Theoretical work by pioneers such as Claude Shannon laid the foundation for methods of encoding signals such that errors can be reduced to arbitrarily small probability.

The Nordstrom-Robinson (NR) code is one such error correction code currently being investigated by Varanasi and Klein. In this project, we present a hardware design utilizing the Xilinx 4003 FPGA that decodes incoming NR codes. The design detects all "detectable" errors and corrects a convenient subset of correctable errors.

2 A Few Coding Theory Concepts and the Nordstrom-Robinson Code

One model of a communication system is shown in figure 1 (from [1]). In any real-world implementation, noise corrupts the signal, reducing the probability that the received message m* is identical to the transmitted message m. In one of his seminal papers on the subject [2], Claude Shannon showed that the probability of error in the received message m* can be made arbitrarily small by appropriate use of channel encoding.

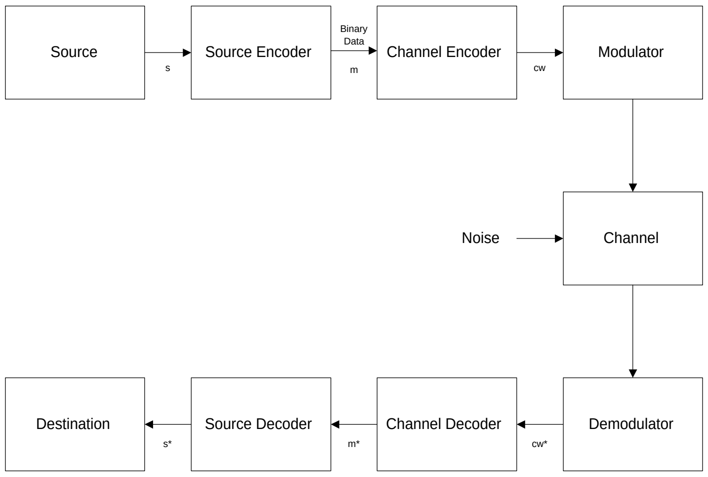


Figure 1. A Communication System Model

One such channel encoding method currently being researched by Dr. M.R. Varanasi (Professor in the Department of Computer Science and Engineering at the University of South Florida, Tampa) and Ron Klein (Ph.D. candidate in the Department of Computer Science and Engineering at the University of South Florida, Tampa) is the Nordstrom-Robinson (NR) code. NR encoding is a block code, which means that data from the source encoder is partitioned into blocks of n adjacent, non-overlapping symbols. In the NR code, $n = 4$. Symbols in the NR code are elements of the ring Z_4, (for an excellent introduction to Abstract Algebra, including groups, rings, and fields, see [3]). In Z_4 the normal mathematical operation of $+$ becomes $\oplus$ and $\times$ becomes $\otimes$. The Cayley tables for each of these operations are shown in tables 1 and 2. Subtraction is performed by adding the inverse (since, in a ring, addition forms an abelian group, and inverses exist in any group).

$\oplus$	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Table 1. Cayley Table for Z_4 Addition ($\oplus$)

$\otimes$	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Table 2. Cayley Table for Z_4 Multiplication ($\otimes$)

The input message M to the NR encoder is a 1×4 vector of Z_4 symbols. The resulting codeword CW is a 1×8 vector of Z_4 symbols. The codeword is calculated from a generator matrix, referred to as the G matrix, as follows:

$$CW = M \times G,$$

where

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 3 & 1 & 2 & 1 \\ 0 & 1 & 0 & 0 & 1 & 2 & 3 & 1 \\ 0 & 0 & 1 & 0 & 3 & 3 & 3 & 2 \\ 0 & 0 & 0 & 1 & 2 & 3 & 1 & 1 \end{bmatrix},$$

and where all operations are performed over Z_4.

We may partition the G matrix into two 4×4 matrices as follows:

$$G = [\, I \quad | \quad G' \,],$$

where I is the 4×4 identity matrix and G' is simply the right half of the original G matrix:

$$G' = \begin{bmatrix} 3 & 1 & 2 & 1 \\ 1 & 2 & 3 & 1 \\ 3 & 3 & 3 & 2 \\ 2 & 3 & 1 & 1 \end{bmatrix}.$$

The codeword then becomes

$$\begin{aligned} CW &= M \times G \\ &= M \times [\, I \quad | \quad G' \,] \\ &= [\, M \quad | \quad M \times G' \,], \end{aligned}$$

where both M and $M \times G'$ are 1×4 vectors. This shows that the codeword can be viewed as two parts: the left four symbols are simply the original message, while the right four symbols are a multiplication of the original message with the G' matrix. We denote the left four symbols of the codeword as CW_L and the right four symbols as CW_R.

In order to detect whether or not an error occurred, the received codeword CW' is multiplied by the transpose of an H matrix in order to generate the "syndrome" S, a 1×4 vector which provides information on the validity of the received codeword:

$$S = CW' \times H^T,$$

where

$$H = \begin{bmatrix} 1 & 3 & 1 & 2 & 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 3 & 0 & 0 & 1 & 0 \\ 3 & 3 & 2 & 3 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

If the syndrome evaluates to the $\mathbf{0}$ vector, then the received codeword is identical to the transmitted codeword. Non-zero syndrome vectors indicate an error has occurred.

As an example, the codeword for the message $[1 \quad 3 \quad 2 \quad 0]$ is computed as follows:

$$\begin{aligned} CW &= M \times G \\ &= [1 \quad 3 \quad 2 \quad 0] \times \begin{bmatrix} 1 & 0 & 0 & 0 & 3 & 1 & 2 & 1 \\ 0 & 1 & 0 & 0 & 1 & 2 & 3 & 1 \\ 0 & 0 & 1 & 0 & 3 & 3 & 3 & 2 \\ 0 & 0 & 0 & 1 & 2 & 3 & 1 & 1 \end{bmatrix} \\ &= [1 \quad 3 \quad 2 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0]. \end{aligned}$$

Notice that CW_L is the original message. The syndrome for a perfectly-received codeword is then computed as:

$$\begin{aligned} S &= CW' \times H^T \\ &= [1 \quad 3 \quad 2 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0] \times \begin{bmatrix} 1 & 3 & 2 & 3 \\ 3 & 2 & 1 & 3 \\ 1 & 1 & 1 & 2 \\ 2 & 1 & 3 & 3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [0 \quad 0 \quad 0 \quad 0] \end{aligned}$$

For this example, let us introduce a single-symbol error into the received codeword such that it now becomes $CW' = [1 \quad 3 \quad 2 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]$. The resulting syndrome would then be

$$\begin{aligned} S &= CW' \times H^T \\ &= [1 \quad 3 \quad 2 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 1 & 3 & 2 & 3 \\ 3 & 2 & 1 & 3 \\ 1 & 1 & 1 & 2 \\ 2 & 1 & 3 & 3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [0 \quad 3 \quad 0 \quad 0]. \end{aligned}$$

2.1 A Convenient Class of Correctable Errors

If the syndrome evaluates to the **0** vector, then the received codeword is identical to the transmitted codeword, and the message can be extracted as CW_L. If it can somehow be determined that only symbols in CW_R are corrupted, then the original message can be recovered again by simply extracting CW_L. If it can somehow be determined that only symbols in CW_L are corrupted, then the original message can be recovered by multiplying CW_R by the inverse of the G' matrix. This is easily seen by remembering that $CW_R = M \times G'$, hence

$$\begin{aligned} CW_R \times G'^{-1} &= M \times G' \times G'^{-1} \\ &= M. \end{aligned}$$

Klein's research has shown that, within the subset of single-symbol errors, it can be easily distinguished whether the error occurred in CW_L or CW_R. Specifically, **assuming the error is a single-symbol error**, a symbol error occurring in CW_R will result in a syndrome pattern of $[\,x \quad 0 \quad 0 \quad 0\,]$, $[\,0 \quad x \quad 0 \quad 0\,]$, $[\,0 \quad 0 \quad x \quad 0\,]$, or $[\,0 \quad 0 \quad 0 \quad x\,]$, where x can be any Z_4 symbol. Conversely, any single-symbol error occuring in CW_L will never result in a syndrome containing three zeros. Hence we may state the following:

> ***If a single-symbol error has occurred, then the syndrome contains three zeros if and only if the error occurred in CW_R.***

Since single-symbol errors are conveniently classified into these two types, it is these types of errors that the hardware was designed to correct.

3 Hardware Theory of Operation

Figure 2 is a flowchart of the basic algorithm which the hardware uses to decode messages. In the flowchart, "C" designates the corrected message, calculated as $CW'_R \times G'^{-1}$.

The syndrome is computed. If the syndrome is zero, then the transmission is reported as error-free, otherwise an error in the transmission is reported. If the syndrome contains three or more zeros, then the corrected message C is chosen as the decoded message M. Note that if the syndrome is zero, then the corrected message C is still correct (although its computation is not necessary). If the syndrome contains less than three zeros, then CW'_L is chosen as the decoded message M.

4 Implementation

4.1 Overview

The design was entered using the OrCAD schematic editing system. The hierarchical capabilities of the tools were used to generate five pages of schematics at three levels. After schematic generation, the `inet`, `annotate`, `sdt2xnf`, `xnfmerge`, and `ppr` utilities were used to generate a Xilinx LCA file targeted for the 4003 series. Finally, the XACT editor was invoked to load the LCA, makebits, and download to the chip.

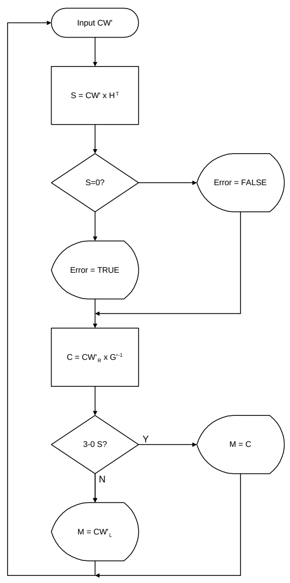


Figure 2. Hardware Algorithm Flowchart

Figure 3 illustrates a block diagram of the hardware, not including user I/O functions. The processing of the data follows the flowchart in figure 2 almost exactly. The timing circuit is designed such that the syndrome (S Reg) and corrected message (C Reg) are always calculated from the codeword. The result is chosen via the quad 2-1 MUX based on the “3 zeros” detector. Note that the base unit of information in the block diagram is a Z_4 symbol. Each symbol is equivalent to two binary bits.

A simple but important characteristic of this design is the mapping of Z_4 symbols to their binary representations. The mapping is done in the “natural” manner, as shown in table 3.

A single Z_4 multiplier-accumulator is used to perform the matrix multiplications. Both the H^T and G'^{-1} matrices are stored in the ROM. The timing circuit operates all signals to perform the appropriate calculations.

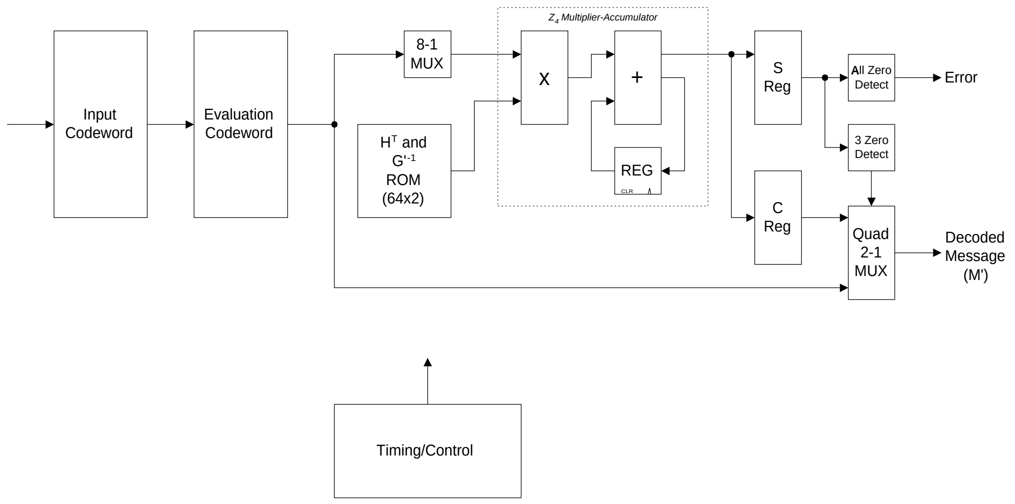


Figure 3. Hardware Block Diagram

Z_4 **Symbol**	**Binary Representation**
0	00
1	01
2	10
3	11

Table 3. Z_4 to Binary Mapping

4.2 User I/O Interface

Figure 4 illustrates the user I/O interfaces for the circuit. The Xilinx 4003 Evaluation Board was utilized for this implementation. Since the number of switches and display devices are limited on this board, techniques such as input switch addressing and output multiplexing were necessary.

Codeword symbols are entered by setting the desired codeword symbol data on S1 and S0, the desired codeword symbol address (0-7, 0 being the least significant symbol) on S2-S4, and then pulsing the Codeword Symbol Load switch S7. Symbols from the input codeword are constantly multiplexed on the 7-segment LED display labeled “Codeword Symbol” at a rate of approximately 1 Hz. The decimal point on the “Codeword Symbol” display flashes on the “0th” symbol so the sequence can be read. The input codeword is not evaluated (i.e., decoded) until the “Evaluate Codeword” switch (S6) is pulsed.

4.3 HT ROM

The HT ROM was generated using the `memgen` utility. Figure 5 shows the file that was used as input to `memgen` in order to generate the `HT ROM` device. In addition, several steps are necessary in order to add the device to the schematic library. Once it is added, one simply selects it from the library and places

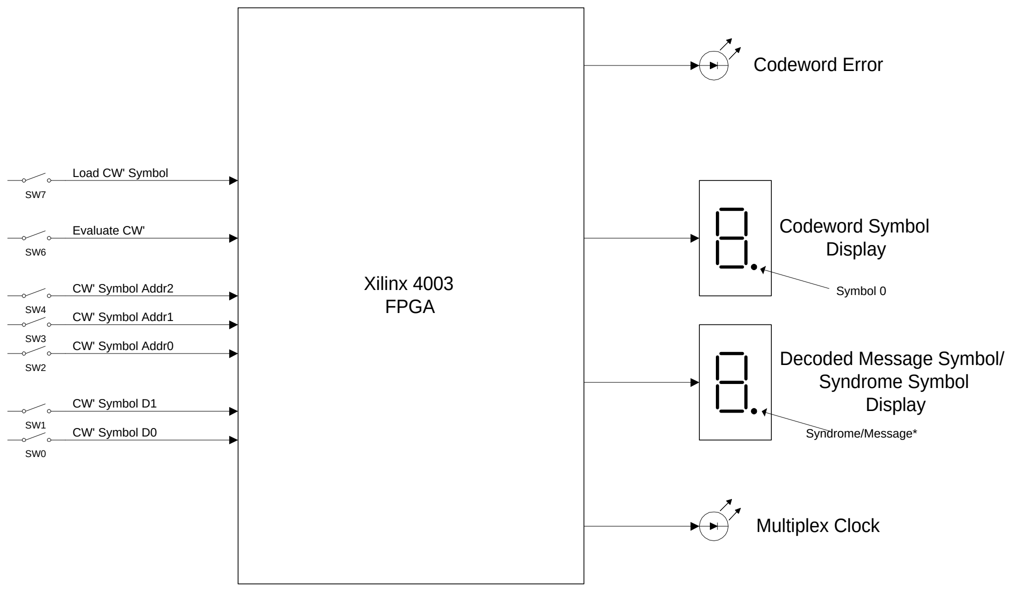


Figure 4. I/O Configuration

it on the schematic, just as you would any other device in the schematic libraries.

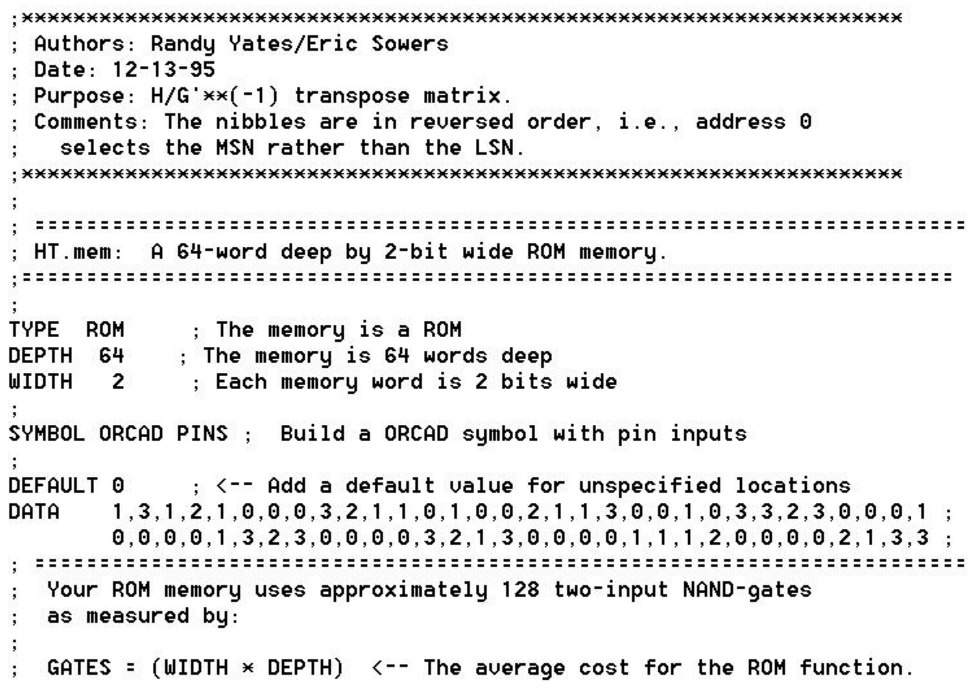
The HT ROM is a 64x2 ROM containing both the H transpose matrix and the G'^{-1} matrix. The data is indexed first by column, then by row, with the top left corner corresponding to address 0. The H^T matrix is in the first 32 locations, followed by the G'^{-1} matrix is in the lower 32 locations. The first four symbols of each column in the G'^{-1} matrix are set to zero since this matrix only requires half the space as the H^T matrix. This layout was simply convenient—a more efficient design can be implemented.

4.4 Timing Generator

The timing generator is shown in figure 9, along with the ROM and S and C registers. The design is fully synchronous, with all clocks driven by the `PCLK` signal from the calling schematic. A 64-state counter is the basis of the circuit. The syndrome is calculated in the first 32 states, while the corrected message is calculated in the last 32 states.

For both syndrome and corrected message calculations, the codeword symbols are addressed using an inverted version of the 3 LSBs of the counter. This causes the symbols to be addressed in reverse order, i.e., symbol 7 is addressed at time 0, symbol 6 at time 1, etc.

During the syndrome calculation, the `ACCCLR` (accumulator clear) signal is asserted on every eighth `PCLK` clock pulse, causing the accumulator latch to clear on the beginning of the next PCLK pulse. Thus the codeword symbols are multiplied by the data in the ROM and accumulated for each column of the matrix data in the ROM, just as a human normally does when computing matrix multiplications.



```
;**********************************************************************
; Authors: Randy Yates/Eric Sowers
; Date: 12-13-95
; Purpose: H/G'**(-1) transpose matrix.
; Comments: The nibbles are in reversed order, i.e., address 0
;   selects the MSN rather than the LSN.
;**********************************************************************
;
; ======================================================================
; HT.mem:  A 64-word deep by 2-bit wide ROM memory.
;======================================================================
;
TYPE  ROM     ; The memory is a ROM
DEPTH  64    ; The memory is 64 words deep
WIDTH   2     ; Each memory word is 2 bits wide
;
SYMBOL ORCAD PINS ;  Build a ORCAD symbol with pin inputs
;
DEFAULT 0     ; <-- Add a default value for unspecified locations
DATA    1,3,1,2,1,0,0,0,3,2,1,1,0,1,0,0,2,1,1,3,0,0,1,0,3,3,2,3,0,0,0,1 ;
        0,0,0,0,1,3,2,3,0,0,0,0,3,2,1,3,0,0,0,0,1,1,1,2,0,0,0,0,2,1,3,3 ;
; ======================================================================
;  Your ROM memory uses approximately 128 two-input NAND-gates
;  as measured by:
;
;  GATES = (WIDTH * DEPTH)  <-- The average cost for the ROM function.
```

Figure 5. HT ROM `.MEM` File

The `Sn_SELECT` signals cycle with the counter and are enabled every eighth `PCLK` pulse, causing the accumulation to be latched into symbol n of the S register on the first accumulation cycle of the next symbol. Note that the decoder signals are reversed since the addressing results in the most significant symbol being calculated first.

Calculation of the corrected message proceeds almost identically, except that the `ACCCLR` signal is asserted every fourth `PCLK` pulse. This results in two accumulations being performed for every symbol. Since the `Cn_SELECT` signal only goes active on the eighth `PCLK` pulse, only the second calculation is latched into the C register. The ROM data is arranged so this results in the proper calculations. Since the first four symbols of each group of eight in the G'^{-1} ROM data are zero, it really wasn't necessary to use a different `ACCCLR` signal. Originally, the ROM was going to be half as deep since the G'^{-1} matrix is actually part of the H^T matrix and can be derived from the H^T matrix ROM using special addressing techniques. However, it was determined that it would be easier and more straightforward to simply enter in the G'^{-1} matrix into the ROM.

bmp

4.5 Multiplier/Accumulator

The multiplier/accumulator is shown in figure 10. It is a two-bit device which operates in Z_4, thus the $\oplus$ and $\otimes$ operations are performed using tables 1 and 2, respectively.

The Z_4 addition is indentical to standard two-bit binary addition with the carry ignored, therefore the two-bit adder library component was utilized. The logic equations were derived for $\otimes$ and simple combinational logic implements the result.

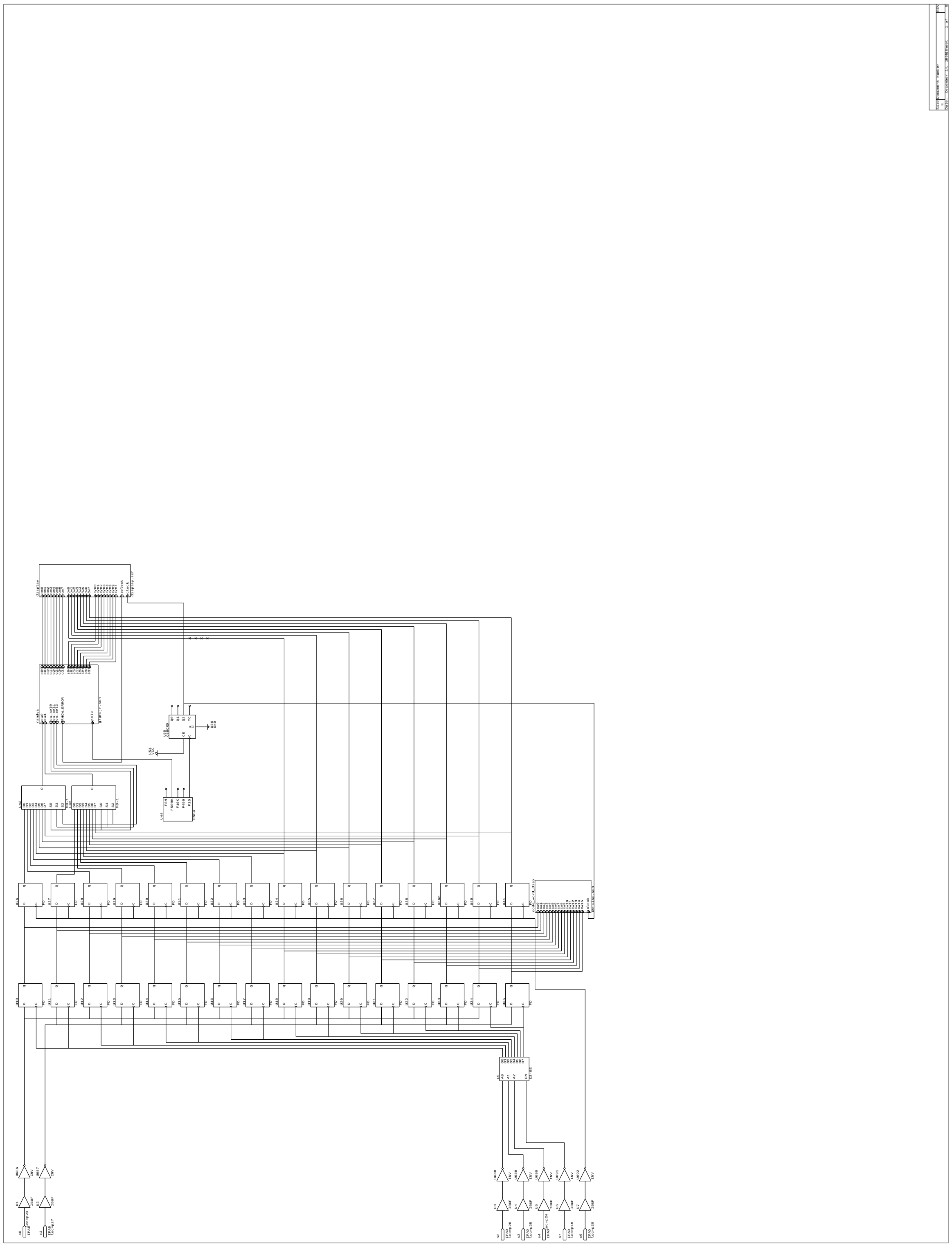


Figure 6. Schematic Sheet 1: Codeword Input Registers, Top Hierarchical Level

The key component in accumulating is the latch. The latch clock and clear are part of the external interface signals, thus providing a completely general Z_4 multiplier/accumulator component.

4.6 I/O Circuitry

Figures 6 through 8 consist mainly of the addressing and multiplexing necessary to implement the user interface functions. The operation of these are relatively straightforward and follow the description in user interface section.

5 Chip Usage

The text below are usage statistics reported by the PPR utility:

```
Preliminary evaluation of your selected part, 4003PC84:
51% utilization of io pins.  ( 31 of 61)
67% utilization of function generators.  (133 of 200)
45% utilization of clb flip-flops.  ( 90 of 200)
This includes 24 function generators and 24 flip-flops inside hard macros,
which cannot be used for other purposes.
This includes 1 CLB flip-flop that must be left unoccupied, because 1 net
sources an odd number of DFF C pins.
```

The remaining 33 percent of function generators implies that a second multiplier/accumulator might be added, enabling more operations to be performed in parallel and thus increasing the operating speed of the design.

6 Performance Analysis

The design is capable of decoding a message every 64 `PCLK`s. Since each message consists of 8 bits, this is equivalent to decoding 8 bits every 64 `PCLK`s. This series of FPGA has typical maximum clock speed of around 60 MHz, thus the design is capable of decoding at a maximum data transfer rate $\mathrm{F_M}$ of

$$\begin{aligned}\mathrm{F_M} &= \frac{\mathrm{F_{CLKM}}\ \text{cycles}}{\text{second}} \times \frac{1\ \text{symbol}}{64\ \text{cycles}} \times \frac{8\ \text{bits}}{1\ \text{symbol}} \\ &= 7.5\ \text{Mbps.}\end{aligned}$$

An immediate improvement of a factor of 2 can be made by simply placing the G'^{-1} matrix in a separate ROM and introducing another copy of the multiplier/accumulator—probably achievable even with a 4003.

7 Conclusion

This design is capable of correcting only a few of the total number of correctable errors provided by the Nordstrom-Robinson code, which in general can correct for any one- and two-symbol errors. The question that has not yet been answered is, “How does one efficiently generate corrections for all types of errors?” A possible brute force method would be to simply use a 256x8 ROM to generate an XOR

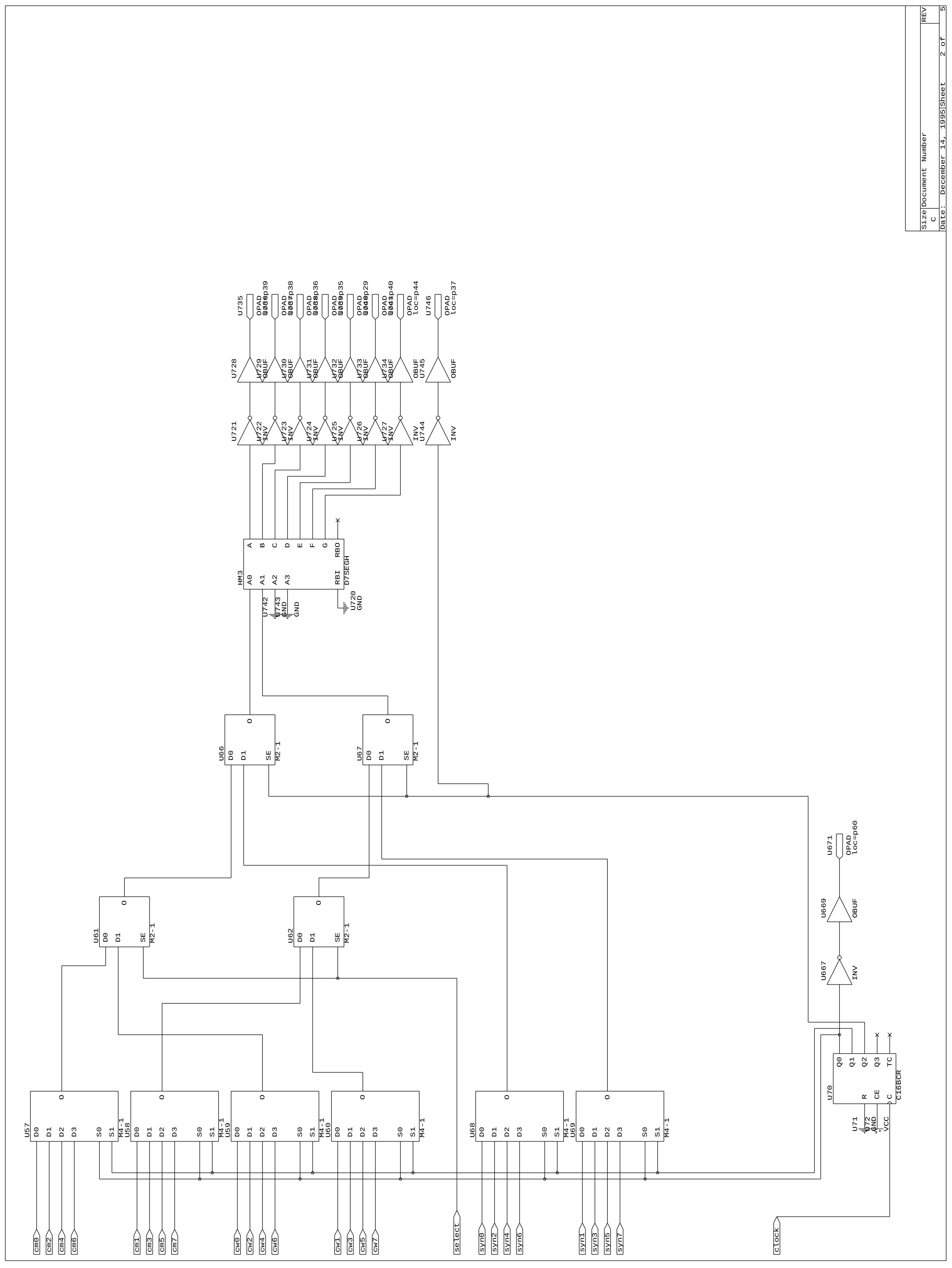


Figure 7. Schematic Sheet 2: Decoded Message MUX and Syndrome/Decoded Message Display

mask that corrects the data. The input to the ROM would be the 8 bit syndrome, and the output the XOR mask to be used on CW_L. However, according to the `memgen` output, such a ROM would require about 2048 of the 3000 available gates in the 4003. The discovery of an efficient method of classifying and correcting errors probably lies in gleaning insights into the theory on which the NR code is based.

Another problem is that of partitioning errors into two sets: correctable and uncorrectable. That is to say, it would be desirable to be able to determine when an error is not correctable, thus enabling the receiver to take appropriate action such as sending a request to the transmitter to resend the data. To date, no such partition is known to the authors.

The chip utilization indicates that perhaps some further parallel processing is possible (e.g., by adding a second multiplier/accumulator) while remaining within the 4003 family. It is significant to note that the amount of chip area required by the user interface is not negligible. If this were a typical decoding device, such functions would not be required and further parallelization may be possible. Since the corrected message is not always required, a small average speedup could be achieved even with the current implementation by only calculating the corrected message when it is required.

Of course, there are much denser FPGAs available today, and a full custom solution would provide the maximum real estate and speed. A very rough estimate is that a factor of 16 speedup (8 times the chip area, double the maximum clock speed) is possible with these alternatives, putting the maximum transfer rate at over 100 Mbps.

8 References

[1] Shu Lin, *An Introduction to Error-Correcting Codes* (Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970).

[2] C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, vol. 27, pp.379-423 (July 1948), and pp.623-656 (October 1948).

[3] John R. Durbin, *Modern Algebra: An Introduction* (third edition, John Wiley and Sons, 1992).

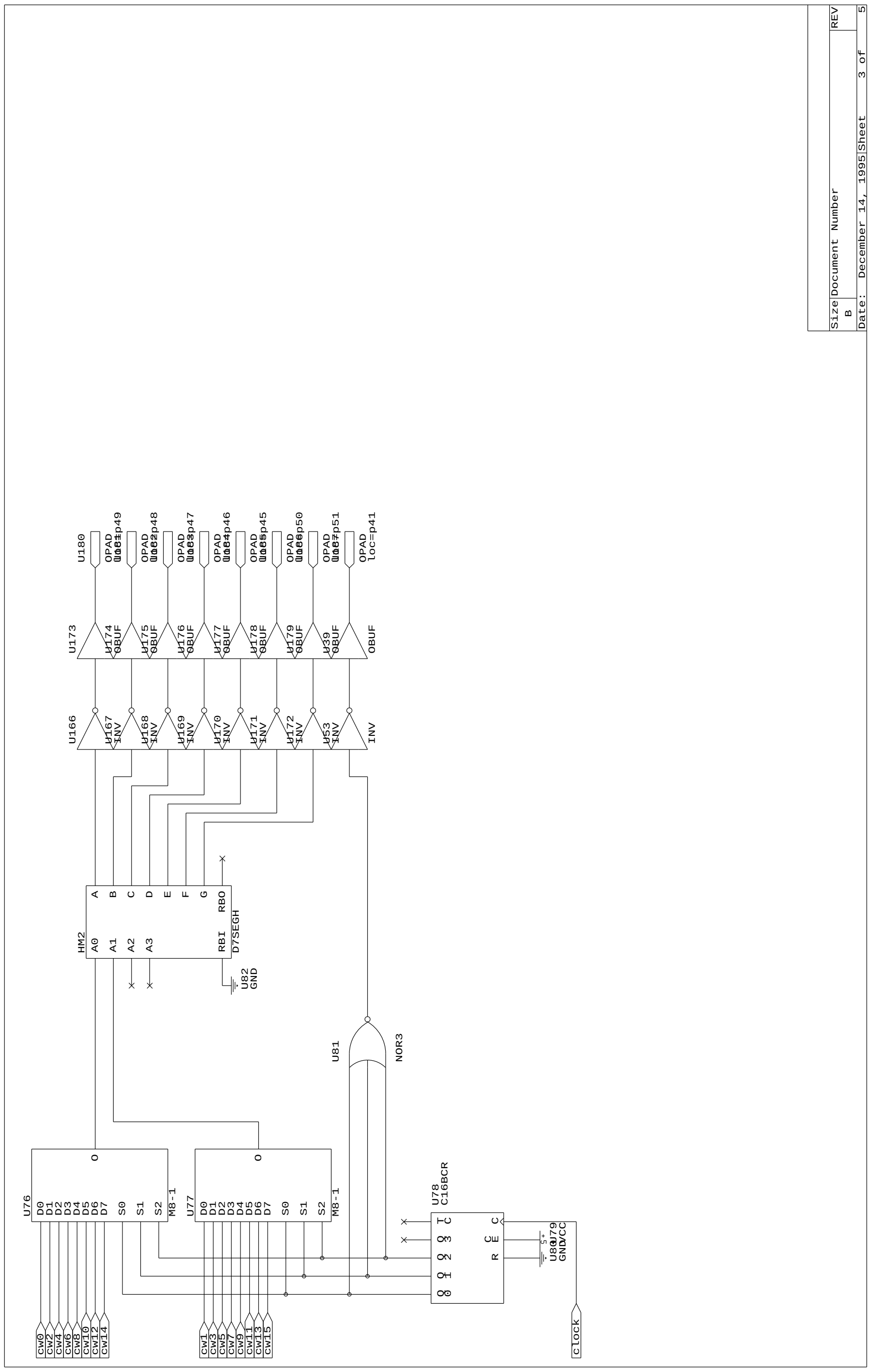


Figure . Schematic Sheet 3: Input Codeword Display

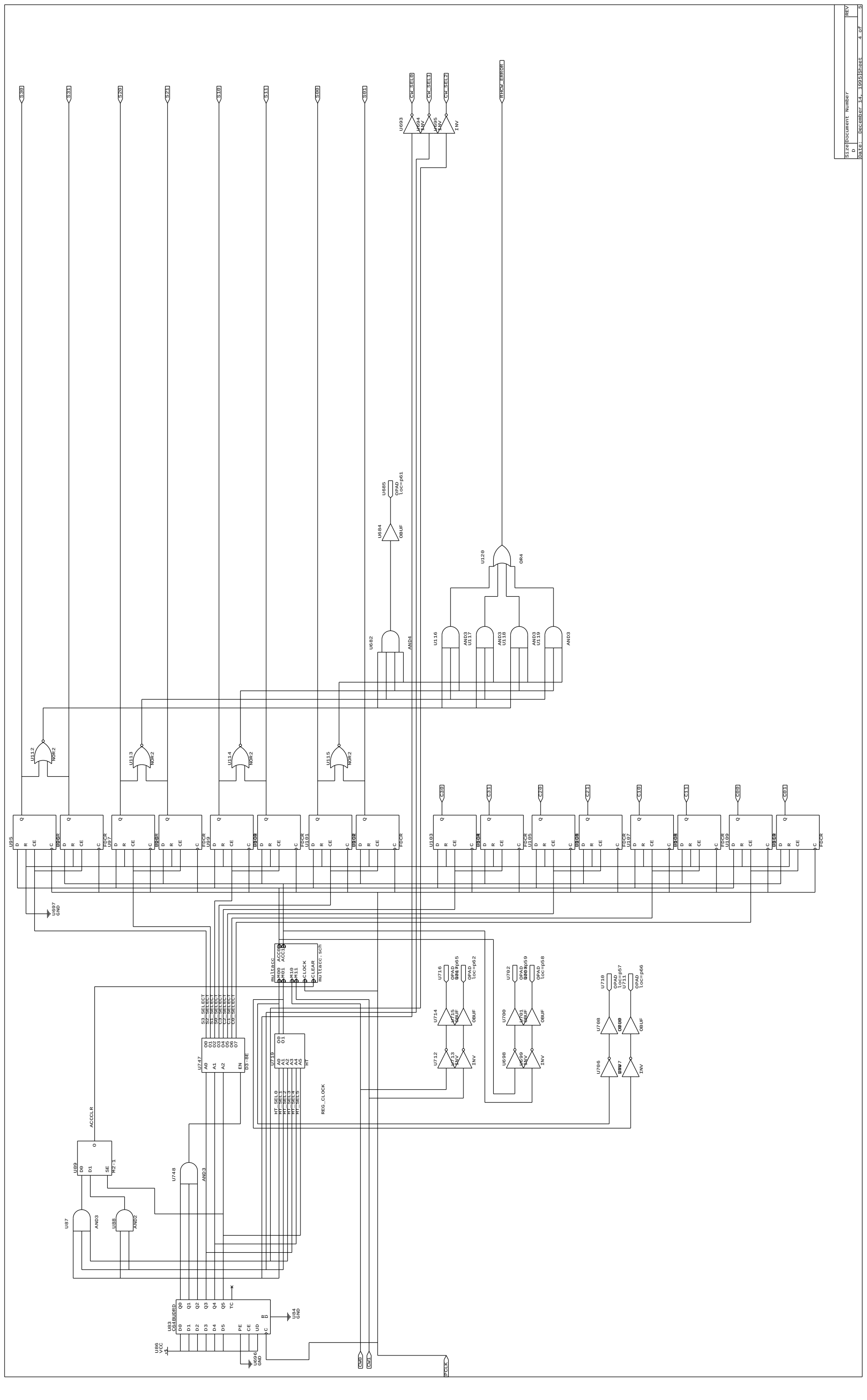


Figure 9. Schematic Sheet 4: Timing Generator, ROM, and S and C Registers

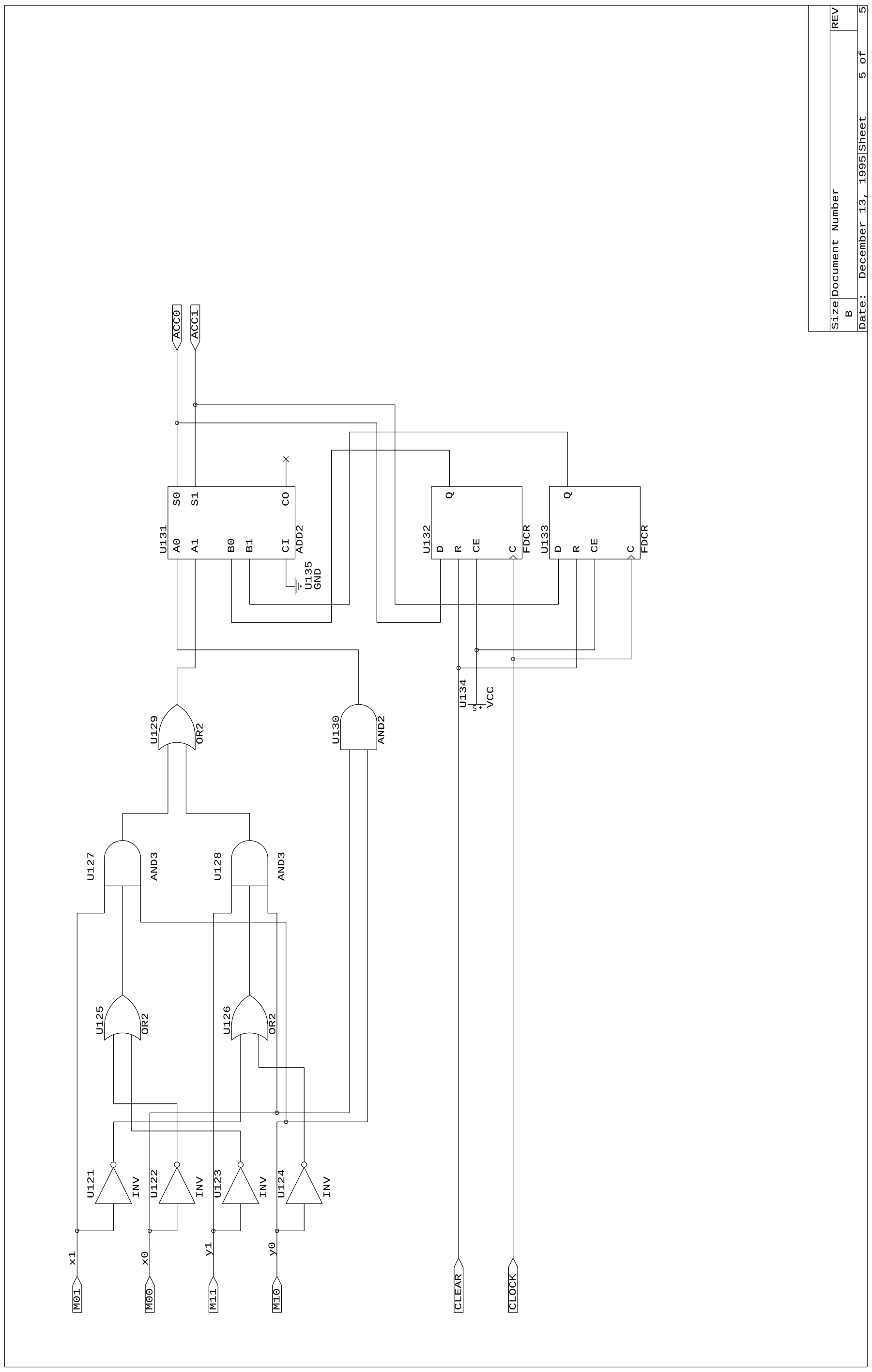


Figure 10. Schematic Sheet 5: Z_4 Multiplier/Accumulator

A Hardware Implementation of the Nordstrom-Robinson Error Correction CodeCharles R. Yates and Eric Sowers