## Encoder Writeup:

Specification for Deep space

For deep space, the LDPC specification is AR4JA LDPC code. It has code size, N =1280 bits = Nb x Z, where Nb=40, Z=circulant size=32. A systematic form of G-matrix is followed for encoding. CCSDS document[1] provides the parity part of Punctured G matrix, which consists of Kb=32 block rows with each block row having (Nb-Kb)=8 circulants.

Here, I is 32x32 identity matrix and dm is a 32x32 dense circulant matrix. Each dense circulant is characterized by their first rows. By cyclic shifting operation on this row, other rows can be generated.

Specifications for Near Earth:

The generator matrix (G) for Near earth standard is of size 7154 x 1022. Having 14x2 array of 511x511 circulants.

The message vector to be multiplied with G is { 18 zeros, 7136 information bits} = 7154 bits.

The generated parity bits 1022. The transmitted Code is assumed to be **8160** = {7136 Systematic, 1022 Parity, 2zeros}, so that effectively, a shortened code (8160, 7136) is transmitted, as mentioned in [1].

The parity check (H) matrix for the Near earth Decoder is for a code standard (8176, 7154). It contains Mb=2 block row or Layers with each layer consisting of block columns Nb=16 corresponding to 16 Circulants of weight 2[1]. The row weight of the H-matrix is constant at Wc=32 for all layers. The circulant size, Z=511.

At the decoder, this codeword is expanded to 8176 codeword = {18 Max Values, 7136 Systematic symbols, 1022 parity symbols}, last 2 zeros are discarded.

### Architecture of encoder

Recursive Convolution Encoder (RCE) and Shift-Register-Adder-Accumulate (SRAA) are 2 types of encoder circuits, to perform multiplication of dense circulants of G-matrix and systematic bits to estimate parity bits. Compared to SRAA structure, the shifting of resultant parity bits is performed here for convolution operation, instead of shifting of circulant bits.

For the deep space LDPC, we found SRAA encoder[9] and Parallel Recursive convolution encoder (Parallel RCE) architectures[6] suitable. Out of this, parallel RCE offered better speed and flexible control over area. So we chose parallel RCE over SRAA.

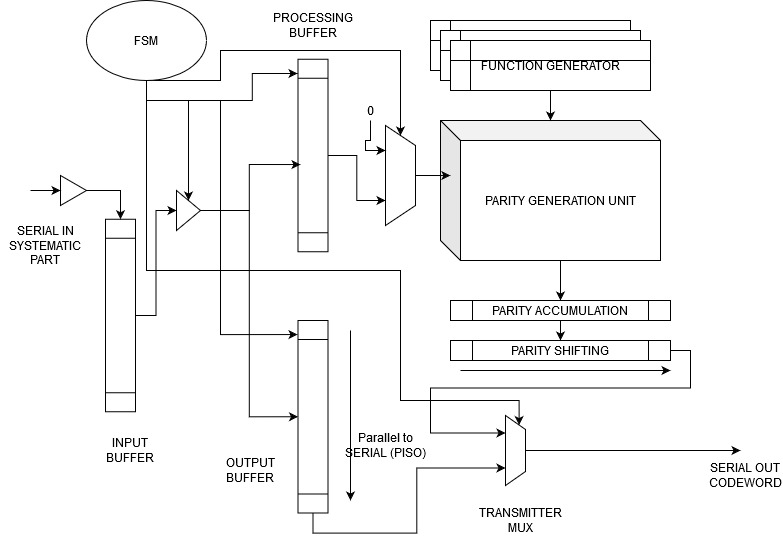


Fig. Parallel RCE architecture overall block diagram.

The parameter for performance is Lm, the systematic bits processed in parallel, and La, circulants of G-matrix processed in parallel, respectively. Based on the area and performance curves in [6], we chose La=8 and Lm=16, to attain smaller latency of 8 cycles.

In SRAA[9], by implementing all circulant rows using wiring, we can eliminate the shift register. For a comparison we had also implemented this parallel version from SRAA type encoder with Lm=Z, La=8. In this parallel version of SRAA:

* Parity accumulators is required, parity shifting operation is only used for output transmission.
* As LDPC code standard changes, Z changes, area of circuit changes. To adjust area by making Lm < Z, additional align/shift register circuit is necessary to get rotated rows of circulant similar to conventional SRAA in [9].
* Number of cycles is less.

For parallel RCE encoder

* Already existing parity shifting operation (for output transmission), is used for convolution, with slight modification.
* As LDPC code standard changes, Z changes, area of circuit can be adjusted by changing value of Lm.
* Number of cycles is more.

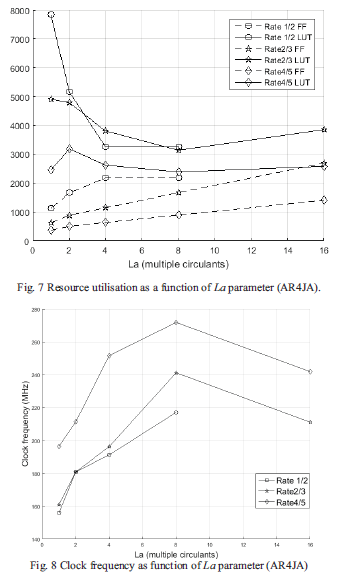


Fig: Area and Performance curves from [6]: The maximum performance is attained for La=8, where Lm x La=16.

We chose Lm=16, La=8, to reduce the latency.

Therefore,

Total systematic bits processed in parallel = Lm x La = 16 x 8 = 128

Total circulants processed in parallel = La x (Nb-Kb)= 8 x 8 =64

Each circulant multiplication completes in ( Z/Lm) = 32/16 = 2 cycles

Since 8 circulants are processed In parallel, in 2 cycles 8 circulant multiplication is completed.

Total cycles of computation= (Kb / La) x (Z/Lm) = (32/8) x (32/16) = 4 x 2 = 8 cycles.

### Pre-processing steps

For Parallel RCE, there are no pre-processing steps involved. The circuit for calculating parity is readily designed according to conventional matrix multiplication.

Here we analyse the intermediate product expressions obtained from normal matrix multiplication and from the parallel RCE network. This is done to identify any modifications to the order of input vector bits and circulant bits that are fed to the network.

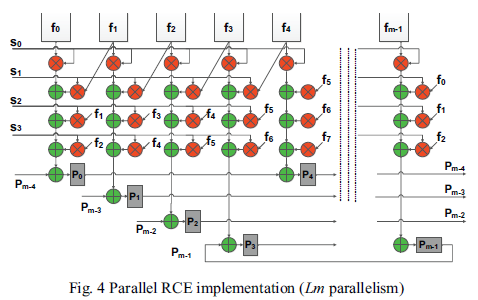
Consider the case of 4x4 dense circulant being multiplied by a 4-bit vector to get a 4-bit parity result ( r) :

Where vector variable **v**= {v1, v2, v3, v4} where v1, v2, v3, v4 are 32 bit vectors, and matrix m is a dense circulant of size 32x32, characterized by its first row, and product vector **r** = {r1, r2, r3, r4}, where r1, r2, r3, r4 are 32 bit vectors.

For the multiplication with 1 dense circulant:

The required expression for parity bits by normal matrix multiplication ( ):

In the (Fig.4) of [6], diagram of parallel RCE network is shown below,



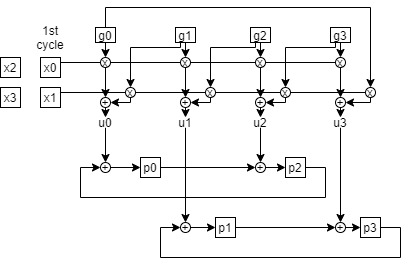


Fig. For Lm=2, circulant bits as g0,g1,g2,g3 and systematic bits as x0,x1,x2,x3 and parity registers p0,p1,p2,p3

Taking Lm=2, an arbitrary input vector and the first row vector of an arbitrary dense circulant as

According to the parallel RCE network, the row vector is left shifted by 1 place as shown below.

Each row should be multiplied with Lm (=2) vector bits in the following form in each cycle:

The result of each cycle (u0,u1,u2,u3) are accumulated with previous results in convolution manner to generate parity result.

From the network, parity register values at final cycle (cycle2) can be expressed as follows. (Note: the super script represents cycle number).

Let the parity result be denoted as ( = value of the parity registers at cycle 2) . Substituting u0,u1,u2,u3 in the above equations, the obtained mathematical expression of parity bits ( ) from parallel RCE network:

Compared with Eq(2):

1. Correcting the order in which the systematic bits are fed into the network:
2. By feeding the circulant row vector after right shifting by 1 place:

In our analysis, we noticed that, to match the expressions for parity for higher values of Lm and circulant size, we found 2 feeding criteria:

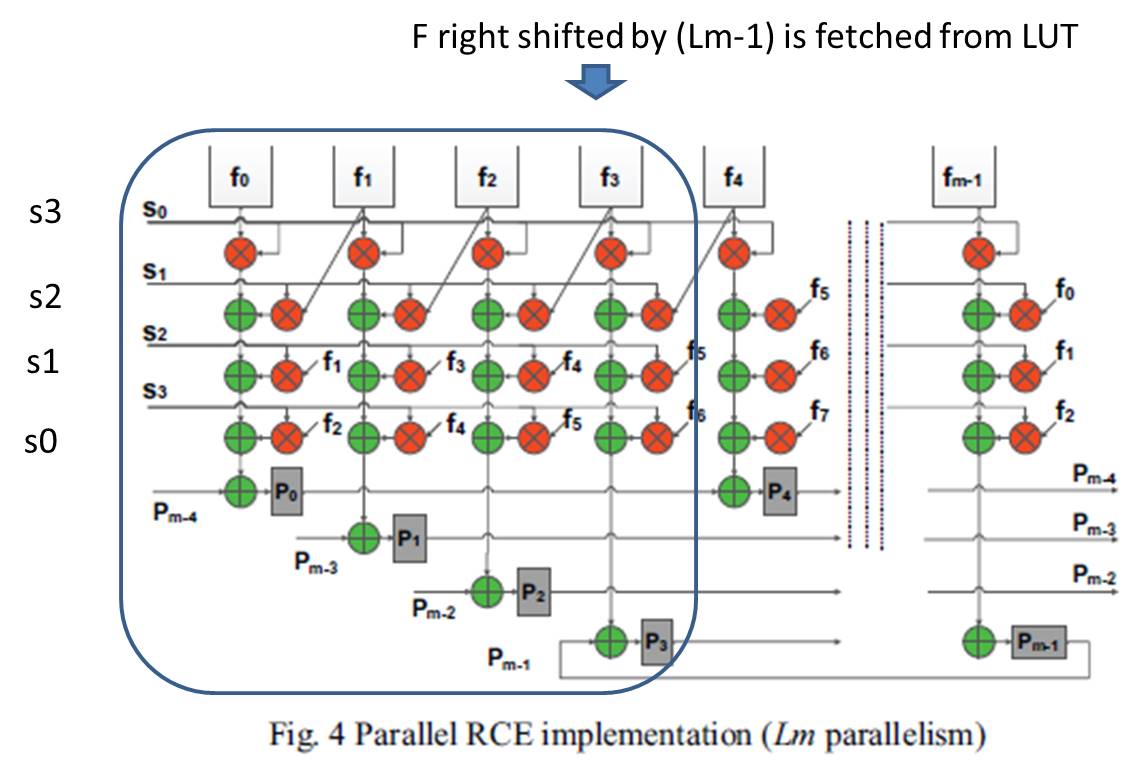
1. The exact order of feeding systematic bits: first step is to bit-reverse the entire 1024 systematic bits. Then feed each Lm bits from MSB of the reversed systematic bits.
2. The exact order of feeding the first row of circulant: the first row of circulant () is shifted right by (Lm-1) places

### Computation Units

The computation units perform multiplication of bit vector and circulant matrix to generate parity bits. In Parallel RCE, it is the Parity generation unit,

In Parallel RCE encoder, the parity generation unit realizes the multiplication of systematic bits with the shifted versions of first row of dense circulant in generator matrix.

The various shifted version are generated by appropriate wiring. In the Lm rows of AND and XOR gates, the multiplication of Lm bits of systematic part with Lm rows out of 32 rows of a dense circulant occurs in 1 cycle. Here we take Lm = 16, therefore the product is ready in 2 cycles.



The above shown is an example of parity generation unit core structure from [6], with Lm=4, and circulant size m=8, with the corrections obtained from pre-processing step. The circuit is partitioned based on Lm. The circuit elements in the box shown above is replicated M/Lm times, until the circulant size is reached.

The parity convolution buffer is formed by the registers { p0, to pm-1) along with the XORs in between. After multiplication of Lm bits and Lm rows, the Lm bits of product is accumulated with previous cycle’s product obtained for preceding Lm bits. For example, in the above figure, to obtain the result for {p4, p5, …, p7}, the corresponding product from the network of current cycle is XOR-ed with product obtained in previous cycle for preceding Lm bits residing in {p0,p1,…, p3}.

Also, there are multiple circulants in a block column j. In , we have 32 circulants in a block column.

So further parallelism is implemented to perform multiplication of La=8 circulants out of 32 circulants of a block column at a time, by replicating this structure La times. The 8 intermediate products in each cycle are accumulated (XOR-ed) before feeding to the parity convolution buffer. In 2 cycles, multiplication with 8 circulants completes to get the intermediate parity results. These are then Then next set of circulant data are loaded, and multiplication continues. After 4 passes, the parity results are ready. Therefore, the total encoding cycles taken is 8 cycles.

### Controllers

For the Parallel RCE architecture, the controller consists of 2 FSMs that consists of various conunters:

1. Receiver and Encoder FSM: This consists of counters to count incoming systematic bits to provide control signals for input buffer and starting the encoder. It also consists of counters to count the number of encoding cycles. Same is used as address to the ROMs. Based on the counter values, appropriate control signals are generated for enabling shift operations and transmission of codeword.
2. Transmit FSM: This FSM functions as the output interface FSM to send out the systematic code word bits. It is initiated once a frame of systematic bits is received. In its first state, it counts the systematic bits as it enables the transmission of systematic bits from input buffer. During this time, the encoding is completed. In the state transition, control signals are given to unload parity bits from convolution buffer. Then, in the next state, transmission from parity shifting buffer is enabled, and parity bits are transmitted.

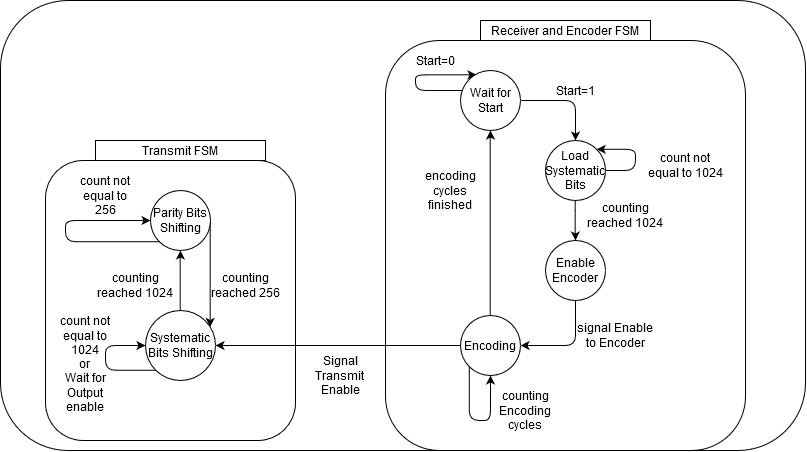


Fig. Parallel RCE Controller FSM state diagram

### ROMs

First rows of dense circulant: In Parallel RCE encoder, the function generators are ROMs that store the first rows of dense circulants of G-matrix.

## Near Earth Encoder

### Pre-processing/Analysis

#### Expected equations:

Consider the case of 4x4 dense circulant being multiplied by a 4-bit vector to get a 4-bit parity result ( r) :

For the multiplication with 1 dense circulant, additional column and row added with f5:

With s5 as an additional bit added to make even length.

Assume s5=f5=0.

The required expression for parity bits by normal matrix multiplication ( ):

#### Observed Equations

Deriving expressions for parallel rce network of (Z=5, Lm=2):

The parallel RCE network in shorthand form:

(Superscript represent cycle)

At cycle 0

At cycle 1

At cycle 2

Parity Register expressions:

At cycle 1

At cycle 2

At cycle 3

At cycle 4:

Assuming parity bits will align properly, here we are taking cycle 4 when we feed one more set of zero systematic messages:

Comparing with the normal matrix multiplication expressions, some corrections are made. The corrected expressions are shown:

By grouping terms: to get cycle2 parity register expressions

Ignoring P5.

Further group terms as: to get cycle 3 parity register expressions

Further add 0 and get cycle4 expressions:

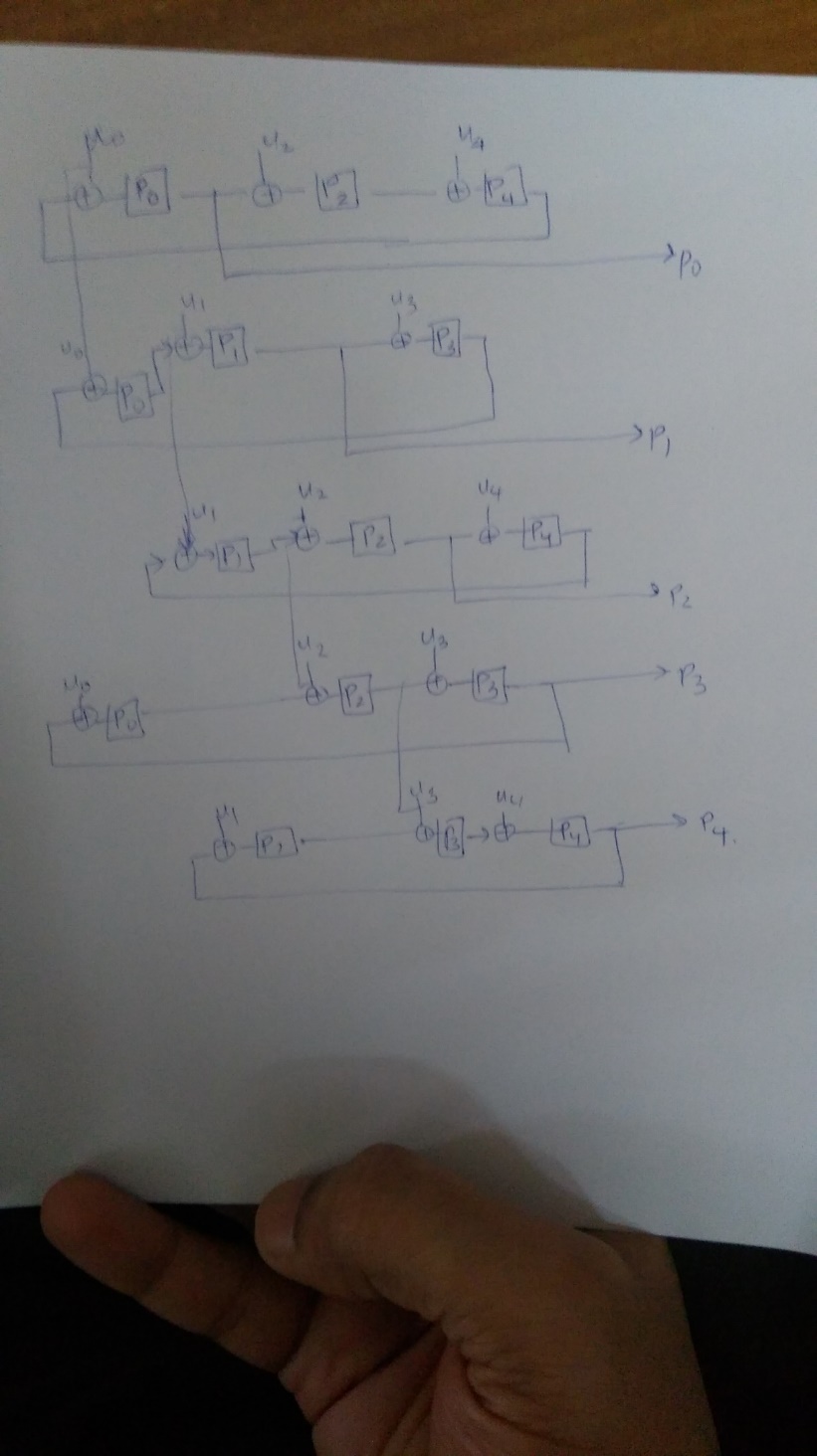


Fig.7. Parity shift networks based on above equations.

#### Observed Rule 1: Left shift Wiring of circulant bits

From the parallel RCE network, circulant bits are left shifted in each row of the Multiply-Accumulate (MAC) network.

#### Rule2: Add Rule, Parity Shift sequence:

Further Compressing above circuit (fig.7), we obtain a rule for generating parity shift sequence:

From Fig.6., if we analyze the subscripts alone of ‘u’, we get the following format

So we can combine these to form the following loop:

So parity shift loop can be formed using Registers:

i.e.

***Design Rule***: For arbitrary Lm value which is not a factor of Z, this sequence can be obtained by a modulo-Z arithmetic progression with a difference of Lm, stopping when first index occurs again.

**Addition Rule:**

*Note*: For Lm which is a factor of Z, such a sequence starting from 0, will not include all the indices (0 to Z-1). So for indices 1 to Lm-1, separate arithmetic progressions are to be written until we get sequences covering all indices. This results in Lm sequences each of length (Z/Lm).

#### Rule3: for output parity bits or circulant bits:

Case 1: First row of circulant bits fed as input to network:

Parity bits can be taken, **in parallel,** as output from R registers at 3rd cycle itself instead of 4th cycle as follows:

Here output parity vector **p** can be obtained from vector **R** of 3rd cycle as:

The right shift amount is same as the left shift amount of last line (i.e. ).

Case2: If last row of circulant is fed as input:

Then parity bits can be taken out as:

We can conclude that vector **p** can be obtained from vector **R** of 3rd cycle as:

The right shift amount is same as the left shift amount of last line (i.e. ).

***Design Rule***: If the circulant bits fed into the network is , where a is the left shift amount, then the last line will have a left shift of , then parity result vector is R at the cycle=ceil(Z/Lm), and right shifted by given as:

Instead of taking shifted version of R as parity, the circulant bits fed to the network can be shifted so that P=R.

**Case 2a:** To make Parity P= R, If second row of circulant is fed as input:

Then parity bits can be taken out as:

We can conclude that vector **p** can be obtained from vector **R** of 3rd cycle as:

***Note***: This is same as the rule obtained for deep space case, where we are feeding circulant bits after one right shift (i.e. second row of circulant).

Case 3: If some information bits are zero, say s0=s1=s2=0. This makes, u at cycle 3 to be zero.

Then we can avoid cycle 3 entirely, as the terms are zero, and the equations:

For Case1,

For Case 2,

For Case2a,

As per paper:

For Lm=16, Z=511 first circulant case:

If the circulant row fed to the network is last row (i.e. ), and for all other circulants the circulant row fed to the network is (i.e. ), then the parity P expression equations from registers R should be constant.?

But as per the observation from Lm=2, Z=5, it is not observed so.

**Case 3a**: To correct the order and make P=R in second cycle of Case2a,

Case 2a circuit:

F values for u3 should arrive at u0, f values for u4 should arrive at u1 and so on.

This change in arrangement of f values results in following, where the second last row of circulant (i.e. ), is fed as input:

***Rule***: From paper, rule followed is for first circulant, feed the last row of circulant (i.e. ), while for remaining circulants feed the first row of circulant (i.e. ).

From above analysis, rule to be followed for first circulant case is feed the second last row of circulant (i.e. ), while for remaining circulants feed the last row of circulant (i.e. ).

For an arbitrary case, we have cycles 0, 1, 2, 3, …, (Z/Lm)-1, (Z/Lm). Where we take output parity bits at cycle=ceil(Z/Lm), since at cycle0, parity bit values are assumed zero.

For Cycle ceil(Z/Lm), i.e 3rd cycle, to make P=R, feed circulant bits =

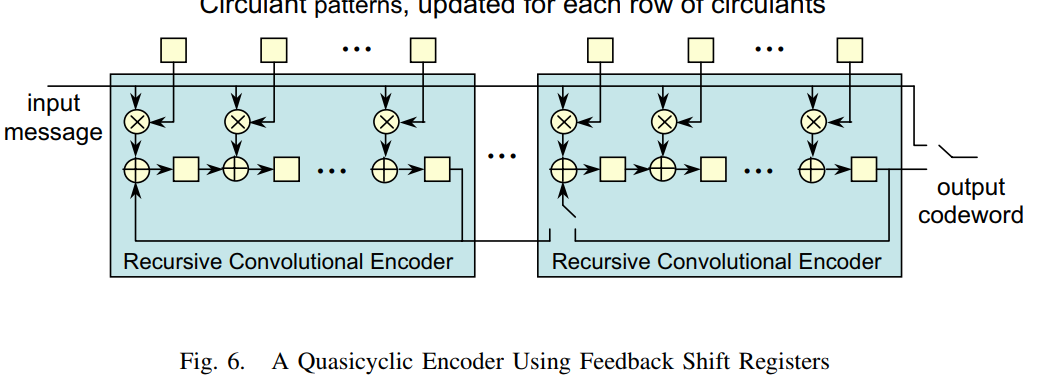
For cycle ceil(Z/Lm)-1, i.e 2nd cycle, to make P=R, feed circulant bits =

For cycle ceil(Z/Lm-2), i.e 1st cycle, to make P=R, feed circulant bits =

For arbitrary cycle b from last cycle i.e. at cycle (Z/Lm)-b, to make P=R, feed circulant bits =

### RCE parallelizing Method

#### Bit serial RCE encoder:



Bit serial RCE encoder in shorthand form:

Serially feed the information bits (s0,s1,s2,s3,s4) such that

Take parity bits serially out after first 5 cycles (cycles 0 to 4), i.e. at cycle 5 onwards

#### Unfolding:

Use unfolding DSP algorithm, we can convert bit serial structures to parallel structure. If the parity registers are represented as {R0,R1,R2,R3,R4}

Representing parity registers (R0,to R4) as Delay elements D:

Let (a,b,c,d,e) represent the nodes at the (+) operations, which represent the wole operation of (multiply and accumulate) at that point. Also there is one delay element (D) between each node connections, parameter w=1.

To make Lm=j=2 parallelism, we replicate the structure Lm times, so that we get Lm versions of the input node s[n] and output node p[n] and internal nodes (a,b,c,d,e).

Input and output node becomes 2:

s1[k]= s[2k+1] ,s0[k]=s[2k]: here we are assuming to feed s1[0]=s5, assuming s5 as 0.

p1[k]= p[2k+1], p0[k]= p[2k]

Where k=0,1,…, ceil(5/Lm)-1

To generalize: for arbitrary Lm value we have inputs: s[Lm\*k], s[Lm\*k +1], s[Lm\*k + 2], …, s[Lm\*k + Lm-1

Replicated unconnected structure :

The connections have to preserved in the following order (as it was in serial circuit):

|  |  |  |  |
| --- | --- | --- | --- |
| Starting node (index=u) | Connect to ( u+w)%j | Delay elements=Floor((u+w)/j) | Connection precedence |
| a0 | (0+1)%2=1 => b1 | (0+1)/2=0=>0 delay | a->b |
| b0 | c1 | 0 | b->c |
| c0 | d1 | 0 | c->d |
| d0 | e1 | 0 | d->e |
| e0 | a1 | 0 | e->a |
| a1 | (1+1)%2=0 => b0 | (1+1)/2=1=>1 delay | a->b |
| b1 | c0 | 1 delay | b->c |
| c1 | d0 | 1 delay | c->d |
| d1 | e0 | 1 delay | d->e |
| e1 | a0 | 1 delay | e->a |
| e0 | (0+0)%2=0=>p0 | (0+0)/2=0=>0 delay | e->p[n] |
| e1 | (1+0)%2=1=>p1 | (1+0)/2=0=>0 delay | e->p[n] |

Applying above results on the Replicated unconnected structure, we get:

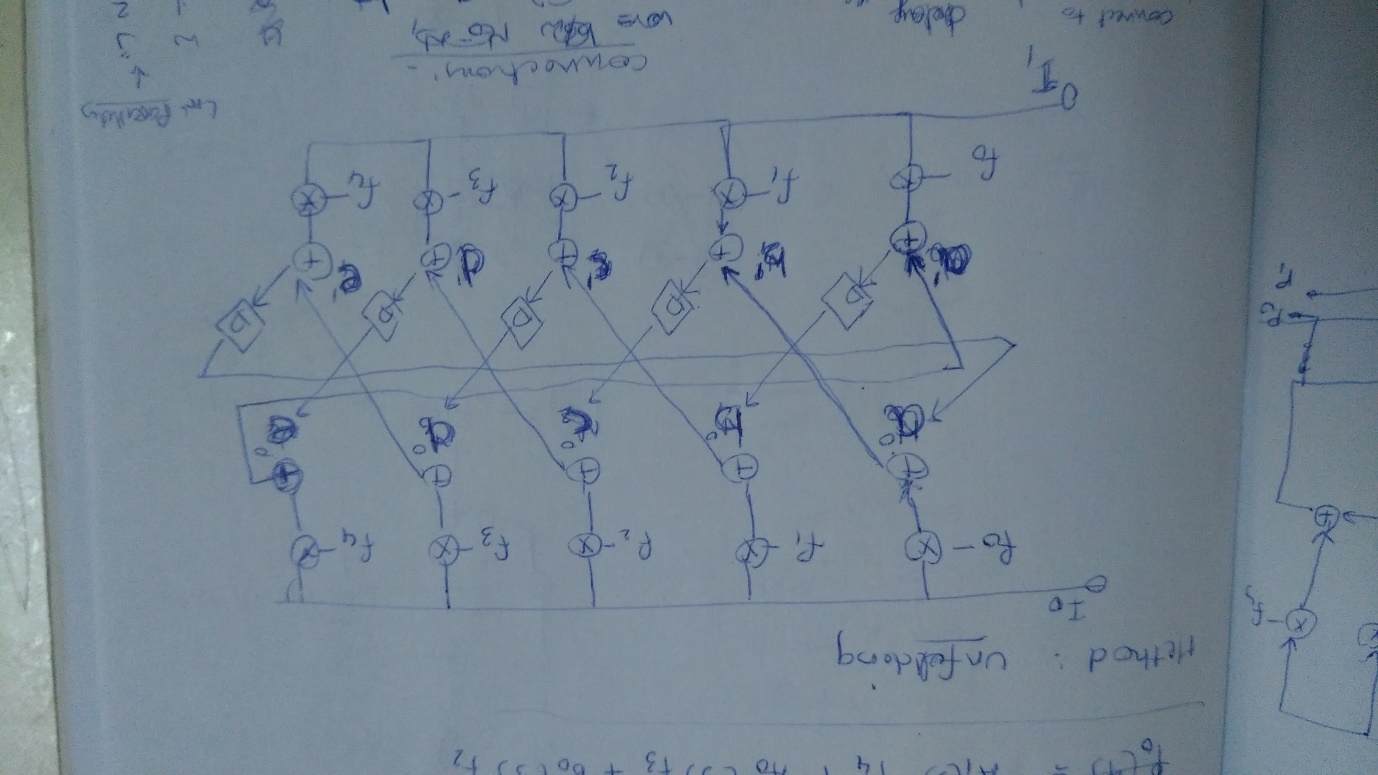


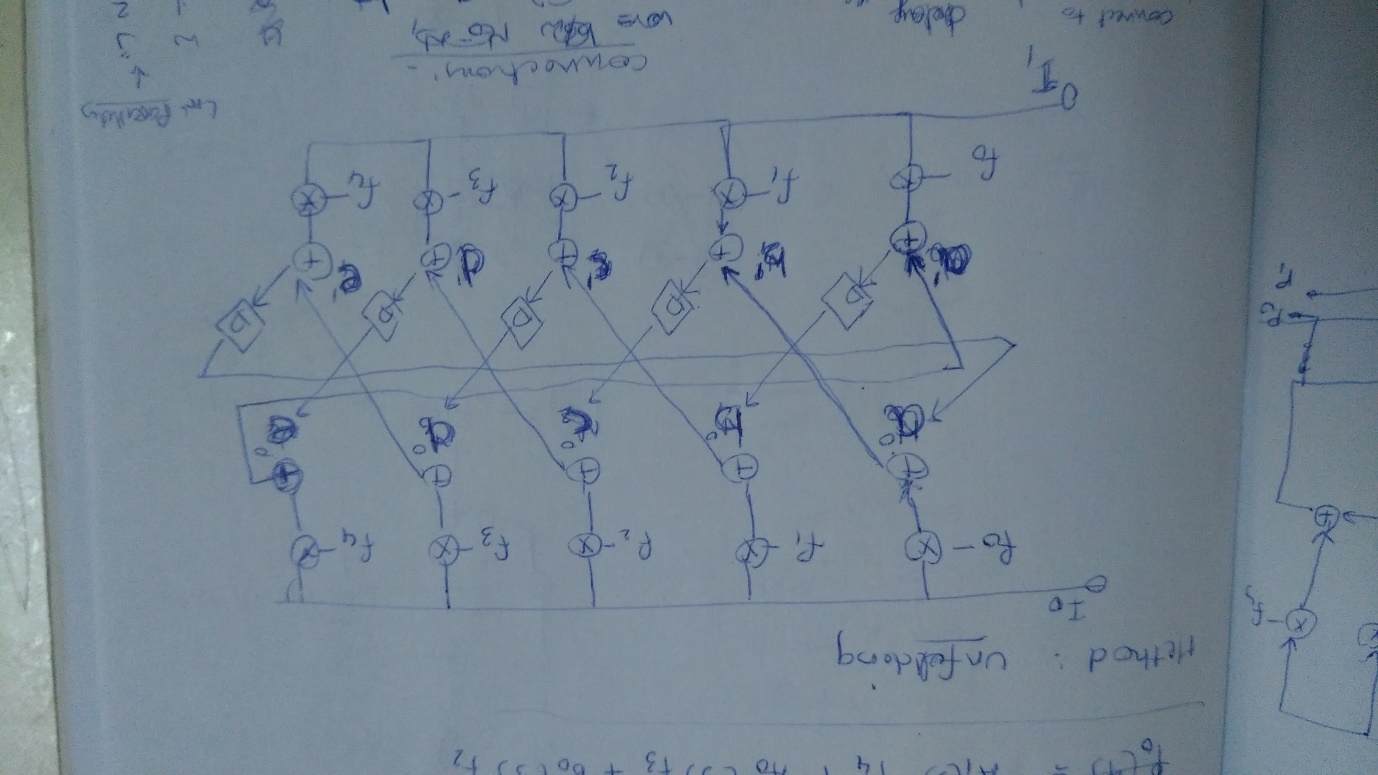
Fig. unfolded structure.

#### Observed Rule1: Inherent left shift of circulant bits:

When we connect based on the table, we can see that the 0-delay connections will align f1 under f0, f2 under f1, and so on. For arbitrary Lm, each line is fed with circulant bits where j = 0 to Lm-1.

Further compressing, we get the structure (as initially assumed parallel rce structure):

#### Observed Rule 2: Add Rule, Register indices and Parity Shift Sequence



Initial in serial architecture, register indices and nodes were related as:

Let the nodes {a,b,c,d,e} be represented as {a0,b0,c0,d0,e0}. Then we get the following naming convention, where register that comes after a0 is R0, register that comes after b0 is R1 and so on.

For parallel structure, we have the connections obtained based on the table as:

Combining above rules and substituting registers in place of delay elements by following the naming convention as in naming1:

To generalize, to find the register indices from unfolded structure,

If nodes are , then after replication and node connections,

For arbitrary node connection pair in the table, having a delay element in the connection sequence, the register index for the delay element is:

The Shift sequence of registers R:

Based on above rule:

**Addition Rule:**

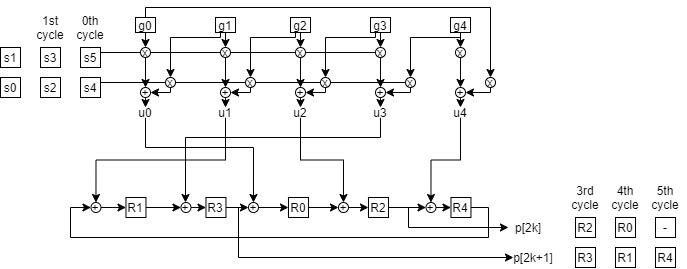
#### Observed Rule 3: Output Parity bits:

To take parity bits output correctly **to satisfy equations**, the mapping of actual parity bits to register values is as shown:

Take parity bits serially out after first 3 cycles, i.e. at cycle 3 onwards

We should expect at k=3rd cycle, register values of R3 and R2, and at 4th cycle , register values of R1 and R0 and at cycle 5th cycle, register values of R4 as shown:

Take outputs p[2k+1], p[2k] in parallel from register output R3 and R2 and the output parity bits will be shifted out in order from these points (nodes e and d).



To take parity bits in parallel, the relation is to be considered. Appropriate alignment in circulant bits can make P=R, as discussed in previous section (Case2a under the Rule for output parity bits or circulant Bits).

For arbitrary Lm, and nodes

Precedence rule with w=1 delay elements ( :

Replication by j=Lm gives:

|  |  |  |  |
| --- | --- | --- | --- |
| Starting node index=u | Connect to node index ( u+w)%j | Delay elements=Floor((u+w)/j) | Connection precedence |
|  | (0+1)%Lm=1 => | (0+1)/Lm=0=>0 delay |  |
|  |  | 0 |  |
|  |  | 0 | … |
| … | … | 0 | … |
|  |  | 0 |  |
|  |  | (1+1)/Lm delay : |  |
|  |  | (1+1)/Lm delay |  |
|  | d0 | 1 delay | … |
| … | e0 | 1 delay | … |
|  | a0 | 1 delay |  |
| e0 | (0+0)%2=0=>p0 | (0+0)/2=0=>0 delay | e->p[n] |
| e1 | (1+0)%2=1=>p1 | (1+0)/2=0=>0 delay | e->p[n] |

## Verilog Design

Msg input order (top to bottom):

The verilog variable reg[Lm-1:0] msg\_inp should be arranged from 0 to Lm-1.

Msg 0 = s5, msg Lm-1 : s4

Circulant bits order (left to right):

0 to M-1

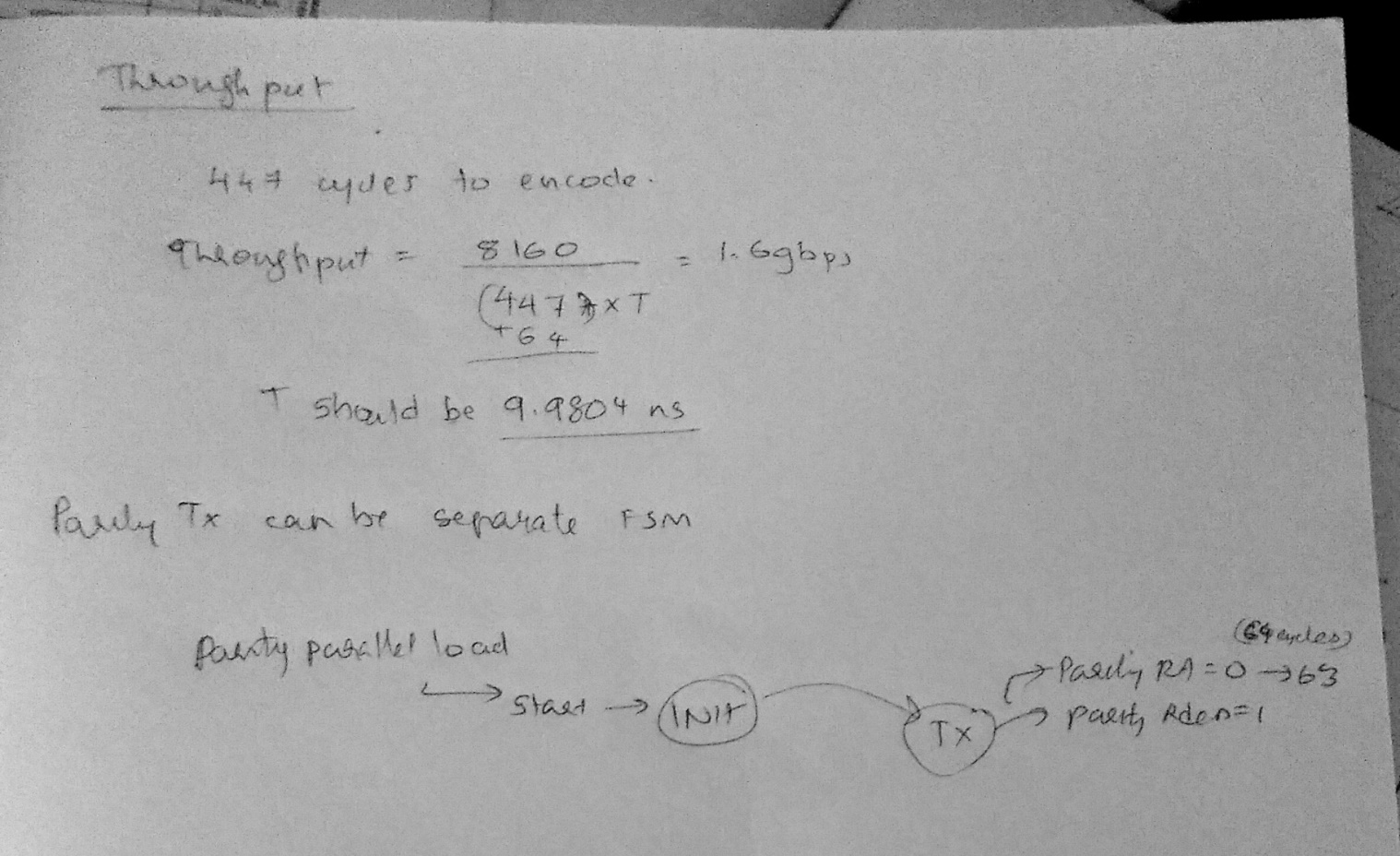
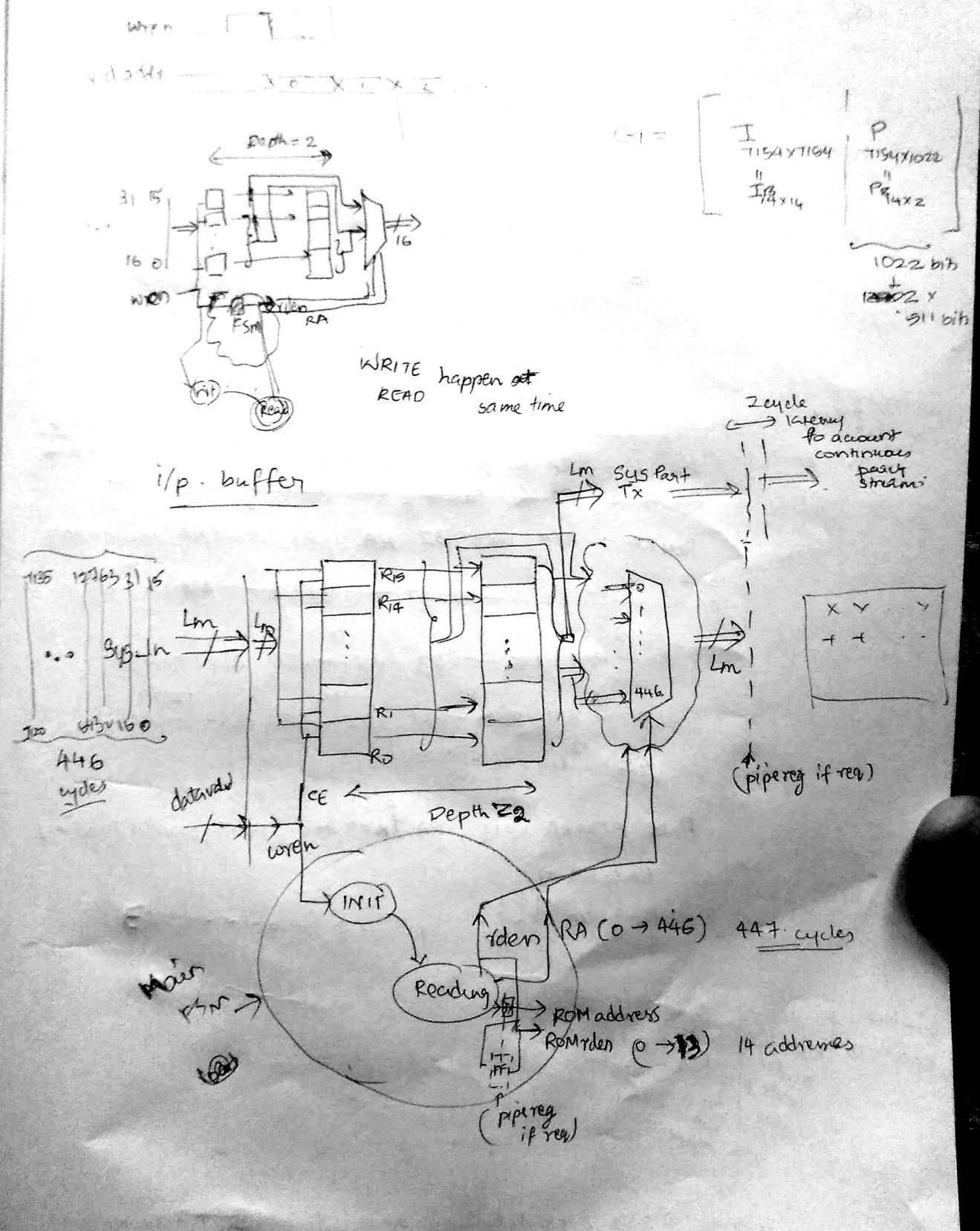
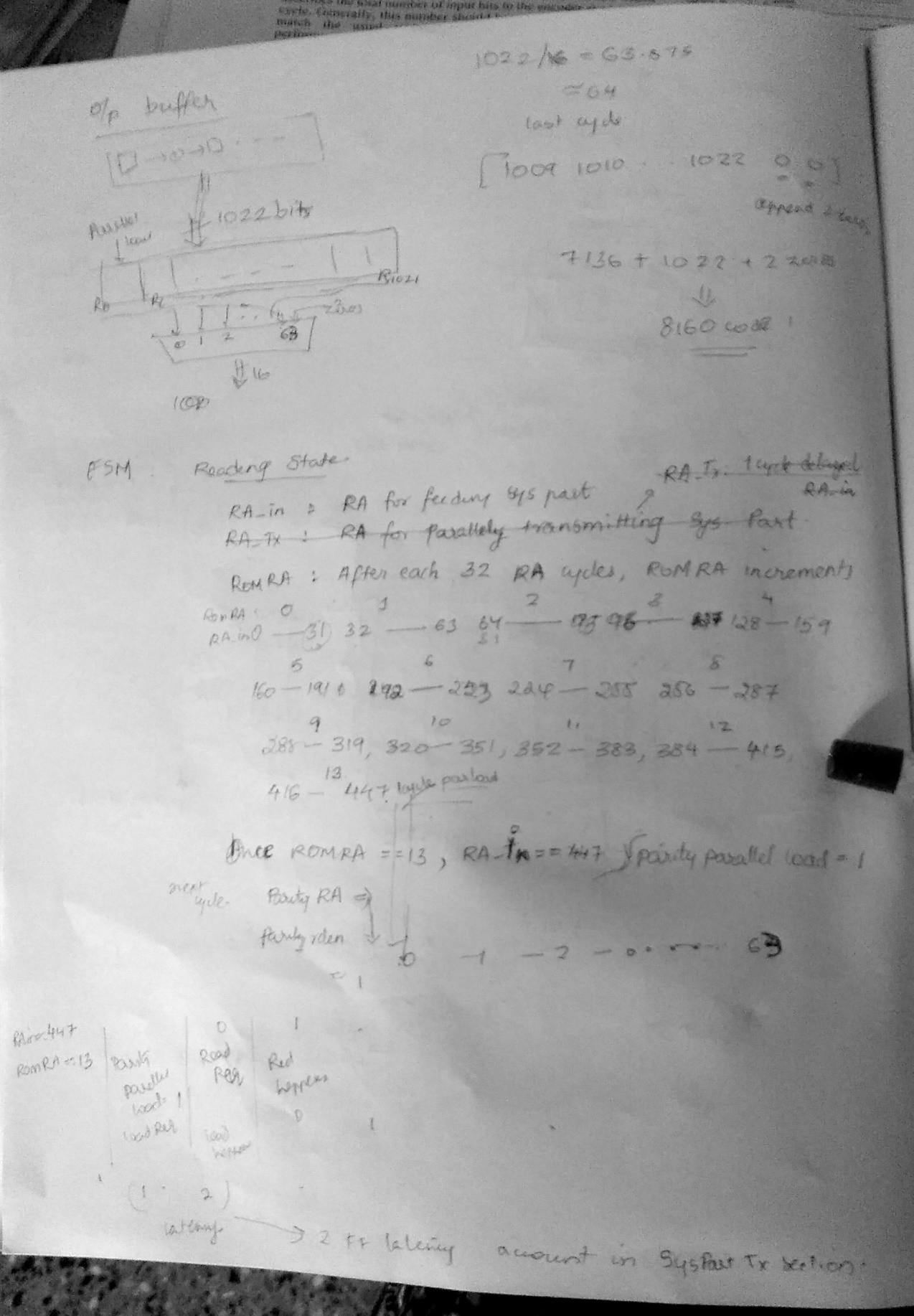
U order ( left to right)

0 to M-1.

## NE Encoder work To do:

### Design of interfaces and Memory

Initial design: 1 bit to 7136 bits output (SIPO) followed by a local memory for all bits



#### Local memory:

Shift register type hardwired (scripted) memory circuit.

Input 16 bits, output 16 bits.

~~Input 7136 bits parallel to 16 bits output~~

~~For each address, provide those bits in the order:~~

~~Address0 : Take first circulant’s 511 bits (assuming it includes 18 zeros) ={0,1,2,.., 17, 18,…,511}, in verilog it is stored in Big-endian= msg511\_1 = {511, …, 18, 17, …, 2, 1, 0}.~~

~~take MSB Lm bits = {511, .., 511-16+1} = msg16\_1 = {Lm-1, …, 0}. Then bit rev it : msg16rev\_1 = {511-16+1, …, 511} = {Lm-1, …, 0}. Feed msg16rev\_1 to encoding network.~~

### Design of Randomizer and ASM sync Marker attacher

### Verification of Encoder:

Without interface or memory:

Testcases:

1. Feed circulant bits for Z=5
2. Feed the 5 message bits in parts of Lm=2
3. Ex; f = {5’b11001}, msg={5’b10001} = 6’b010001, msg\_0 = {2’b01}, msg\_1={2’b00}, msg\_2={2’b01}.
4. See if matlab reference for product of f and msg is obtained in TB simulation.

without interface:

reference model in MATLAB -> reference codeword bits(8176 bits) and sample input message bits(7154), and TB in vivado.

test cases:

1. Feed only first 511 message bits and first circulant bits. Check the output matching with the MAtlab output

2. Feed (7154 + 14zeros - first 16zeros) message bits

with interface:

refertence model in MATLAB-> reference codeword bits(8160 bits) and sample input message bits (7136) and TB in vivado