TSG-RAN Working Group 1(Radio) meeting #6
Espoo, Finland 13-16th , July 1999

TSGR1#6(99) 946

Source: **LGIC**

Title: **Alternative Uplink Puncturing Algorithm**

---

1. Introduction

Samsung proposed a new universal puncturing algorithm in this meeting. But to our knowledge, the uplink part of this algorithm is ambiguous. Therefore, we propose an alternative puncturing algorithm for uplink. The basic idea is similar to that of the previously presented algorithm[7]. In other words, we can divide the column sequence into the ‘y’ sequence of 1st RSC encoder and ‘z’ sequence of 2nd RSC encoder then, apply the similar procedure to the previously presented algorithm to calculate the shifting parameter *S* for each column sequences.

2. Design of Alternative Uplink Puncturing Algorithm for Turbo Code

The design rules imposed on the uplink puncturing algorithm for turbo code are as below.

Preventing puncturing of systematic bits
Systematic bits of turbo code are more important than parity bits which means that puncturing of one systematic bit results in more performance degradation than a parity bit.

Equal amount and uniform puncturing of parity bits of two encoders
In order to maximise the BER performance of turbo code, the coding strength of each RSC code must be balanced. Balanced puncturing of parity bits between the two encoders means balanced puncturing of each RSC code.

Equal amount of puncturing for each 1st MIL interleaved column sequence
For the uplink, rate matching algorithm is performed over 1st MIL interleaved sequence and therefore, rate matching must be performed in a way that every column sequence has an equal amount of puncturing. The purpose of uplink rate matching for turbo code is to satisfy the original property of turbo puncturing algorithm in the view point of “before the 1st MIL original code sequence” while applying the equal amount of puncturing over each interleaved sequence.

Providing a unified rate matching algorithm for uplink and downlink
For the simplicity of implementation, it is desirable to use a unified rate matching algorithm for uplink and downlink.

3. The Description of Alternative Uplink Puncturing Algorithm

Figure 1 shows the example of writing the 1st MIL of K=8. In figure 1, In the figure, ‘x’ means the systematic code bit, ‘y’ the parity bit from 1st RSC encoder and ‘z’ from the 2nd RSC encoder. The subscript of each bit is the order of code symbol. This example is the case of totally 96 code symbol so 288 code bits.

The basic idea of this proposal is that we can use two rate matching algorithm in independent and parallel manner. In other words, rate matching algorithm 1 for ‘y’ sequences for each column and rate matching algorithm 2 for ‘z’ sequences for each column operate simultaneously.

For this purpose, we can divide the each column sequence of figure 1 into two groups. One is the group of ‘y’ bit sequence and the other is the group of ‘z’ bit sequence.

Then, we can obtain the virtual interleaver memory as shown in figure 2. Figure 2-(a) is an example of virtual interleaver for ‘y’ bit sequence and figure 2-(b) is an example of virtual interleaver for ‘z’ bit sequence.

The bold number of the 1st row for each virtual interleaver represents the actual column number of original interleaver. That is, the 1st column of the left interleaver represents the 1st column of the original MIL interleaver and the 2nd column represents the 7th column of the original MIL interleaver, and vice versa. The mapping of virtual interleaver column index to the original interleaver column index can be simply described by the next equation

$$Q(k) \quad = \quad (3k+1) \quad \text{mod} \quad K \ : \ k = 0,1,2,\cdots K-1 \quad \text{(mapping rule for ‘y’ bit sequence)}$$

$$Q(k) \quad = \quad (3k+2) \quad \text{mod} \quad K \ : \ k = 0,1,2,\cdots K-1 \quad \text{(mapping rule for ‘z’ bit sequence)}$$

x_{0}	y_{0}	z_{0}	x_{1}	y_{1}	z_{1}	x_{2}	y_{2}
z_{2}	x_{3}	y_{3}	z_{3}	x_{4}	y_{4}	z_{4}	x_{5}
y_{5}	z_{5}	x_{6}	y_{6}	z_{6}	x_{7}	y_{7}	z_{7}
x_{8}	y_{8}	z_{8}	x_{9}	y_{9}	z_{9}	x_{10}	y_{10}
z_{10}	x_{11}	y_{11}	z_{11}	x_{12}	y_{13}	z_{12}	x_{13}
y_{13}	z_{13}	x_{14}	y_{14}	z_{14}	x_{15}	y_{15}	z_{15}
x_{16}	y_{16}	z_{16}	x_{17}	y_{17}	z_{17}	x_{18}	y_{18}
z_{18}	x_{19}	y_{19}	z_{19}	x_{20}	y_{20}	z_{20}	x_{21}
y_{21}	z_{21}	x_{22}	y_{22}	z_{22}	x_{23}	y_{23}	z_{23}
x_{24}	y_{24}	z_{24}	x_{25}	y_{25}	z_{25}	x_{26}	y_{26}
z_{26}	x_{27}	y_{27}	z_{27}	x_{28}	y_{28}	z_{28}	x_{29}
y_{29}	z_{29}	x_{30}	y_{30}	z_{30}	x_{31}	y_{31}	z_{31}
x_{32}	y_{32}	z_{32}	x_{33}	y_{33}	z_{33}	x_{34}	y_{34}
z_{34}	x_{35}	y_{35}	z_{35}	x_{36}	y_{36}	z_{36}	x_{37}
y_{37}	z_{37}	x_{38}	y_{38}	z_{38}	x_{39}	y_{39}	z_{39}
x_{40}	y_{40}	z_{40}	x_{41}	y_{41}	z_{41}	x_{42}	y_{42}
z_{42}	x_{43}	y_{43}	z_{43}	x_{44}	y_{44}	z_{44}	x_{45}
y_{45}	z_{45}	x_{46}	y_{46}	z_{46}	x_{47}	y_{47}	z_{47}
x_{48}	y_{48}	z_{48}	x_{49}	y_{49}	z_{49}	x_{50}	y_{50}
z_{50}	x_{51}	y_{51}	z_{51}	x_{52}	y_{52}	z_{52}	x_{53}
y_{53}	z_{53}	x_{54}	y_{54}	z_{54}	x_{55}	y_{55}	z_{55}
x_{56}	y_{56}	z_{57}	x_{57}	y_{57}	z_{57}	x_{58}	y_{58}
z_{58}	x_{59}	y_{59}	z_{59}	x_{60}	y_{60}	z_{60}	x_{61}
y_{61}	z_{61}	x_{62}	y_{62}	z_{62}	x_{63}	y_{63}	z_{63}
x_{64}	y_{64}	z_{64}	x_{65}	y_{65}	z_{65}	x_{66}	y_{66}
z_{66}	x_{67}	y_{67}	z_{67}	x_{68}	y_{68}	z_{68}	x_{69}
y_{69}	z_{69}	x_{70}	y_{70}	z_{70}	x_{71}	y_{71}	z_{71}
x_{72}	y_{72}	z_{72}	x_{73}	y_{73}	z_{73}	x_{74}	y_{74}
z_{74}	x_{75}	y_{75}	z_{75}	x_{76}	y_{76}	z_{76}	x_{77}
y_{77}	z_{77}	x_{78}	y_{78}	z_{78}	x_{79}	y_{79}	z_{79}
x_{80}	y_{80}	z_{80}	x_{81}	y_{81}	z_{81}	x_{82}	y_{82}
z_{82}	x_{83}	y_{83}	z_{83}	x_{84}	y_{84}	z_{84}	x_{85}
y_{85}	z_{85}	x_{86}	y_{86}	z_{86}	x_{87}	y_{87}	z_{87}
x_{88}	y_{88}	z_{88}	x_{89}	y_{89}	z_{89}	x_{90}	y_{90}
z_{90}	x_{91}	y_{91}	z_{91}	x_{92}	y_{92}	z_{92}	x_{93}
y_{93}	z_{93}	x_{94}	y_{94}	z_{94}	x_{95}	y_{95}	z_{95}

Figure 1. example of writing the interleaver memory

1	**4**	**7**	**2**	**5**	**0**	**3**	**6**
y_{0}	y_{1}	y_{2}	y_{3}	y_{4}	y_{5}	y_{6}	y_{7}
y_{8}	y_{9}	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}
y_{16}	y_{17}	y_{18}	y_{19}	y_{20}	y_{21}	y_{22}	y_{23}
y_{24}	y_{25}	y_{26}	y_{27}	y_{28}	y_{29}	y_{30}	y_{31}
y_{32}	y_{33}	y_{34}	y_{35}	y_{36}	y_{37}	y_{38}	y_{39}
y_{40}	y_{41}	y_{42}	y_{43}	y_{44}	y_{45}	y_{46}	y_{47}
y_{48}	y_{49}	y_{50}	y_{51}	y_{52}	y_{53}	y_{54}	y_{55}
y_{56}	y_{57}	y_{58}	y_{59}	y_{60}	y_{61}	y_{62}	y_{63}
y_{64}	y_{65}	y_{66}	y_{67}	y_{68}	y_{69}	y_{70}	y_{71}
y_{72}	y_{73}	y_{74}	y_{75}	y_{76}	y_{77}	y_{78}	y_{79}
y_{80}	y_{81}	y_{82}	y_{83}	y_{84}	y_{85}	y_{86}	y_{87}
y_{88}	y_{89}	y_{90}	y_{91}	y_{92}	y_{93}	y_{94}	y_{95}

Figure 2-(a). virtual interleaver for ‘y’ sequence

2	**5**	**0**	**3**	**6**	**1**	**4**	**7**
z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}
z_{16}	z_{17}	z_{18}	z_{19}	z_{20}	z_{21}	z_{22}	z_{23}
z_{24}	z_{25}	z_{26}	z_{27}	z_{28}	z_{29}	z_{30}	z_{31}
z_{32}	z_{33}	z_{34}	z_{35}	z_{36}	z_{37}	z_{38}	z_{39}
z_{40}	z_{41}	z_{42}	z_{43}	z_{44}	z_{45}	z_{46}	z_{47}
z_{48}	z_{49}	z_{50}	z_{51}	z_{52}	z_{53}	z_{54}	z_{55}
z_{56}	z_{57}	z_{58}	z_{59}	z_{60}	z_{61}	z_{62}	z_{63}
z_{64}	z_{65}	z_{66}	z_{67}	z_{68}	z_{69}	z_{70}	z_{71}
z_{72}	z_{73}	z_{74}	z_{75}	z_{76}	z_{77}	z_{78}	z_{79}
z_{80}	z_{81}	z_{82}	z_{83}	z_{84}	z_{85}	z_{86}	z_{87}
z_{88}	z_{89}	z_{90}	z_{91}	z_{92}	z_{93}	z_{94}	z_{95}

Figure 2-(b). virtual interleaver for ‘z’ sequence

Let’s assume that among total number of 36 code bits for each column, 4 bits are to be punctured. In this case, N_c is 96 and P is 4. Then, we can use two rate matching procedure for each ‘y’ and ‘z’ sequence in figure 2-(a) and figure 2-(b).

All we need to do is to calculate the shifting parameter for each column of 1st interleaver for each rate matching algorithm.

The shifting parameter for each column is calculated in the similar procedure.

1) Calculation of Shifting Parameter for ‘y’ sequence

$N = \left\lfloor \frac{N_c}{3} \right\rfloor$: $\lfloor x \rfloor$ is the largest integer which does not exceed the value of ‘x’

$N_i = N - \left\lceil \frac{P}{2} \right\rceil$: $\lceil x \rceil$ is the smallest integer which exceeds the value of ‘x’

$q = \left\lfloor \frac{N}{|N_i - N|} \right\rfloor$: q means the average puncturing distance of ‘y’ sequence

From the value of q, we can find the shifting parameter S guaranteeing the overall uniformity over “before the 1st interleaved ‘y’ sequence” as follows.

```
if($q \le 2$){
        for($k$=0; $k$<$K$; $k$++) {
                if(($k$%2)=0)
                        $S[R[(3k+1) \bmod K]] = 0$;

                else
                        $S[R[(3k+1) \bmod K]] = 1$;
        }
}
else{
```

```
    if((q%2)=1)
        $q' = q - \frac{G.C.D(q,K)}{K}$ ;  to avoid hitting the same column
    else    $q' = q$
    for(i=0; i< K ; i++) {
        $k = \lceil i*q' \rceil \% K$ ;
        $S[R[(3k+1) \bmod K]] = \lceil i*q' \rceil \text{ div } K$ ;
    }
}
```

In the above procedure, R [] means the mapping pattern of the 1st MIL interleaver.

2) Calculation of Shifting Parameter for 'z' sequence

$N = \left\lfloor \frac{N_c}{3} \right\rfloor$: $\lfloor x \rfloor$ is the largest integer which does not exceed the value of 'x'

$N_i = N - \left\lceil \frac{P}{2} \right\rceil$: $\lceil x \rceil$ is the smallest integer which exceeds the value of 'x'

$q = \left\lfloor \frac{N}{|N_i - N|} \right\rfloor$: q means the average puncturing distance of 'z' sequence

From the value of q, we can find the shifting parameter S guaranteeing the overall uniformity over "before the 1st interleaved 'z' sequence" as follows.

```
if($q \le 2$){
    for(k=0; k<K; k++) {
        if((k%2)=0)
            $S[R[(3k+2) \bmod K]] = 0$;
        else
            $S[R[(3k+2) \bmod K]] = 1$;
    }
}
else{
    if((q%2)=1)
        $q' = q - \frac{G.C.D(q,K)}{K}$ ;  to avoid hitting the same column
    else    $q' = q$
    for(i=0; i< K ; i++) {
        $k = \lceil i*q' \rceil \% K$ ;
        $S[R[(3k+2) \bmod K]] = \lceil i*q' \rceil \text{ div } K$ ;
```

```
    }
}
```

Then using the shifting parameter obtained through the procedure of (1) and (2), the two rate matching block operates simultaneously. The rate matching procedure can be described as follows.

(3) Rate Matching Procedure

```
    $S_0 = \{d_{N_1}, d_{N_2}, \cdots d_{N_c}\}$ : set of $N_C$ data bits for each column
    $N_i$ : symbol number after puncturing
    $N = \lfloor N_c / 3 \rfloor$
    k : column index  k=0,1,2,3,….,K-1 (K : Column number of 1st MIL)
    if puncturing is to be performed
        $y = N - N_i$
        $e = (2*S(k) * y + N) \bmod 2N$        -- initial error  $e_{offset}$
        – $S(k)$ from procedure 1 if ‘y’ sequence puncturing and from procedure 2 if ‘z’ sequence puncturing
        if($e$=0)  $e = 2N$
        $m = 1$                                -- index for current symbol
        do while $m <= N$
            $e = e – 2 * y$                    -- update error
            if $e <= 0$ then                   -- check if symbol number m should be punctured
                puncture bit $m$ from set $S_0$
                $e = e + 2*N$                  -- update error
            end if
            $m = m + 1$                        --  index for next symbol
        end do
    end if
    else if repetition is to be performed
        $y = N - N_i$
        $e = (2*S(k) * y + N) \bmod 2N$        -- initial error  $e_{offset}$
        – $S(k)$ from procedure 1 if ‘y’ sequence repetition and from procedure 2 if ‘z’ sequence repetition
        if($e$=0)  $e = 2N$
        $m = 1$                                -- index for current symbol
        do while $m <= N$
            $e = e – 2 * y$                    -- update error
            if $e <= 0$ then                   -- check if symbol number m should be repeated
                repeat bit $m$ from set $S_0$
                $e = e + 2*N$                  -- update error
            end if
            $m = m + 1$                        --  index for next symbol
        end do
```

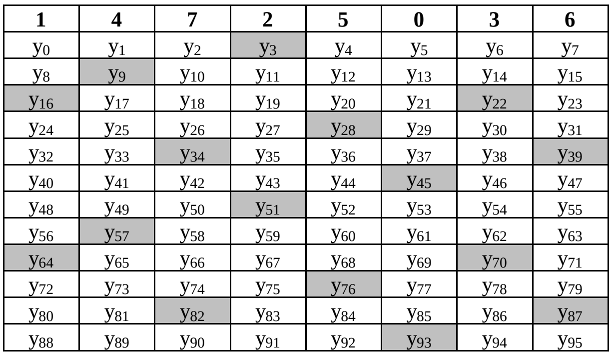
end if

Using the above procedure, we can obtain the puncturing pattern which satisfies almost all of the requirements for turbo code puncturing.

4. Example of Puncturing Pattern of the Alternative Algorithm

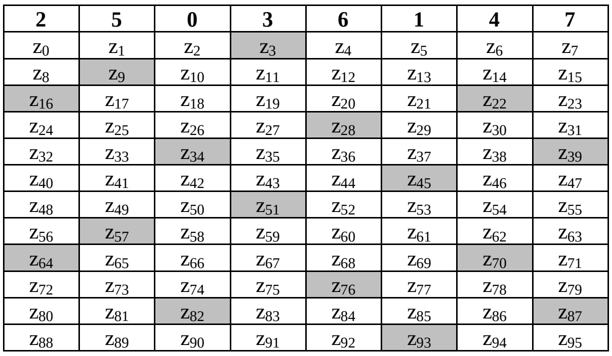
Let’s assume the case when among the 36 code bits for each column, 4 bits are to be punctured.

Then, by applying the above procedure, we can obtain the puncturing pattern for each virtual interleaver as shown in figure 3-(a) and figure 3-(b)



1	4	7	2	5	0	3	6
y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}
y_{16}	y_{17}	y_{18}	y_{19}	y_{20}	y_{21}	y_{22}	y_{23}
y_{24}	y_{25}	y_{26}	y_{27}	y_{28}	y_{29}	y_{30}	y_{31}
y_{32}	y_{33}	y_{34}	y_{35}	y_{36}	y_{37}	y_{38}	y_{39}
y_{40}	y_{41}	y_{42}	y_{43}	y_{44}	y_{45}	y_{46}	y_{47}
y_{48}	y_{49}	y_{50}	y_{51}	y_{52}	y_{53}	y_{54}	y_{55}
y_{56}	y_{57}	y_{58}	y_{59}	y_{60}	y_{61}	y_{62}	y_{63}
y_{64}	y_{65}	y_{66}	y_{67}	y_{68}	y_{69}	y_{70}	y_{71}
y_{72}	y_{73}	y_{74}	y_{75}	y_{76}	y_{77}	y_{78}	y_{79}
y_{80}	y_{81}	y_{82}	y_{83}	y_{84}	y_{85}	y_{86}	y_{87}
y_{88}	y_{89}	y_{90}	y_{91}	y_{92}	y_{93}	y_{94}	y_{95}

Figure 3-(a). puncturing pattern for ‘y’ sequences

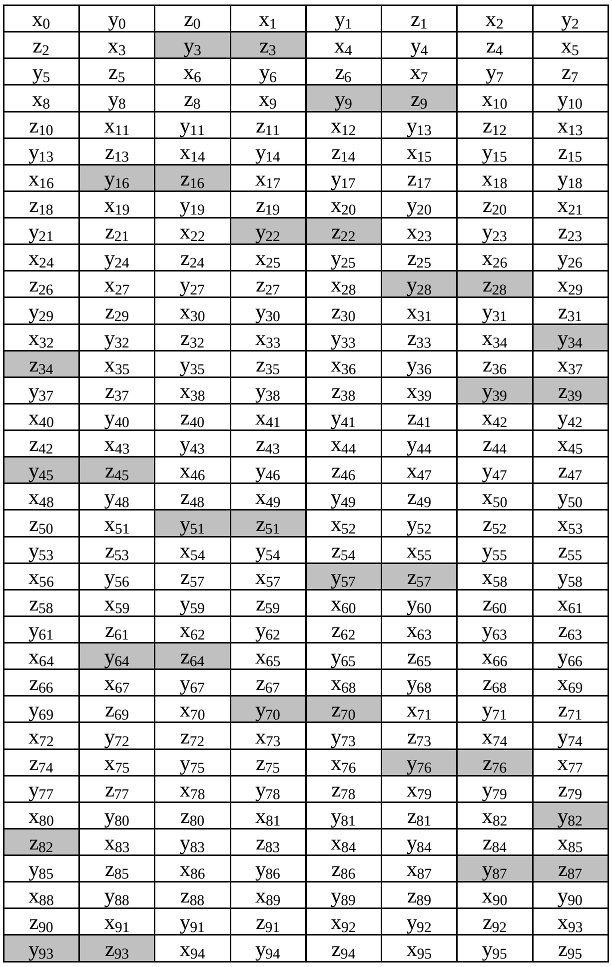


2	5	0	3	6	1	4	7
z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}
z_{16}	z_{17}	z_{18}	z_{19}	z_{20}	z_{21}	z_{22}	z_{23}
z_{24}	z_{25}	z_{26}	z_{27}	z_{28}	z_{29}	z_{30}	z_{31}
z_{32}	z_{33}	z_{34}	z_{35}	z_{36}	z_{37}	z_{38}	z_{39}
z_{40}	z_{41}	z_{42}	z_{43}	z_{44}	z_{45}	z_{46}	z_{47}
z_{48}	z_{49}	z_{50}	z_{51}	z_{52}	z_{53}	z_{54}	z_{55}
z_{56}	z_{57}	z_{58}	z_{59}	z_{60}	z_{61}	z_{62}	z_{63}
z_{64}	z_{65}	z_{66}	z_{67}	z_{68}	z_{69}	z_{70}	z_{71}
z_{72}	z_{73}	z_{74}	z_{75}	z_{76}	z_{77}	z_{78}	z_{79}
z_{80}	z_{81}	z_{82}	z_{83}	z_{84}	z_{85}	z_{86}	z_{87}
z_{88}	z_{89}	z_{90}	z_{91}	z_{92}	z_{93}	z_{94}	z_{95}

Figure 3-(b). puncturing pattern for ‘z’ sequences

Then we can obtain the resulting puncturing pattern as shown in figure 4. As can be seen in figure 4, puncturing of ‘y’ sequence and ‘z’ sequence occurs simultaneously. But if the initial offsets for each sequences are calculated in a different manner, then another pattern can be obtained.

The problem occurs if the puncturing number for each column is an odd number. Then, in the extreme case, 8 more ‘y’ bits can be punctured than ‘z’ bits or 8 more ‘z’ bits than ‘y’ bits. If it is preferred to avoid this problem, the number of puncturing for each sequence can be calculated by $\left\lceil \frac{P}{2} \right\rceil$. Using this number, shifting parameters for each sequence are calculated. Then two rate matching algorithm operates simultaneously, knowing that for example, for odd numbered column in the 1st interleaver, the last puncturing of ‘y’ must be avoided and for even numbered column, the last puncturing of ‘z’ must be avoided.



x_0	y_0	z_0	x_1	y_1	z_1	x_2	y_2
z_2	x_3	y_3	z_3	x_4	y_4	z_4	x_5
y_5	z_5	x_6	y_6	z_6	x_7	y_7	z_7
x_8	y_8	z_8	x_9	y_9	z_9	x_{10}	y_{10}
z_{10}	x_{11}	y_{11}	z_{11}	x_{12}	y_{13}	z_{12}	x_{13}
y_{13}	z_{13}	x_{14}	y_{14}	z_{14}	x_{15}	y_{15}	z_{15}
x_{16}	y_{16}	z_{16}	x_{17}	y_{17}	z_{17}	x_{18}	y_{18}
z_{18}	x_{19}	y_{19}	z_{19}	x_{20}	y_{20}	z_{20}	x_{21}
y_{21}	z_{21}	x_{22}	y_{22}	z_{22}	x_{23}	y_{23}	z_{23}
x_{24}	y_{24}	z_{24}	x_{25}	y_{25}	z_{25}	x_{26}	y_{26}
z_{26}	x_{27}	y_{27}	z_{27}	x_{28}	y_{28}	z_{28}	x_{29}
y_{29}	z_{29}	x_{30}	y_{30}	z_{30}	x_{31}	y_{31}	z_{31}
x_{32}	y_{32}	z_{32}	x_{33}	y_{33}	z_{33}	x_{34}	y_{34}
z_{34}	x_{35}	y_{35}	z_{35}	x_{36}	y_{36}	z_{36}	x_{37}
y_{37}	z_{37}	x_{38}	y_{38}	z_{38}	x_{39}	y_{39}	z_{39}
x_{40}	y_{40}	z_{40}	x_{41}	y_{41}	z_{41}	x_{42}	y_{42}
z_{42}	x_{43}	y_{43}	z_{43}	x_{44}	y_{44}	z_{44}	x_{45}
y_{45}	z_{45}	x_{46}	y_{46}	z_{46}	x_{47}	y_{47}	z_{47}
x_{48}	y_{48}	z_{48}	x_{49}	y_{49}	z_{49}	x_{50}	y_{50}
z_{50}	x_{51}	y_{51}	z_{51}	x_{52}	y_{52}	z_{52}	x_{53}
y_{53}	z_{53}	x_{54}	y_{54}	z_{54}	x_{55}	y_{55}	z_{55}
x_{56}	y_{56}	z_{57}	x_{57}	y_{57}	z_{57}	x_{58}	y_{58}
z_{58}	x_{59}	y_{59}	z_{59}	x_{60}	y_{60}	z_{60}	x_{61}
y_{61}	z_{61}	x_{62}	y_{62}	z_{62}	x_{63}	y_{63}	z_{63}
x_{64}	y_{64}	z_{64}	x_{65}	y_{65}	z_{65}	x_{66}	y_{66}
z_{66}	x_{67}	y_{67}	z_{67}	x_{68}	y_{68}	z_{68}	x_{69}
y_{69}	z_{69}	x_{70}	y_{70}	z_{70}	x_{71}	y_{71}	z_{71}
x_{72}	y_{72}	z_{72}	x_{73}	y_{73}	z_{73}	x_{74}	y_{74}
z_{74}	x_{75}	y_{75}	z_{75}	x_{76}	y_{76}	z_{76}	x_{77}
y_{77}	z_{77}	x_{78}	y_{78}	z_{78}	x_{79}	y_{79}	z_{79}
x_{80}	y_{80}	z_{80}	x_{81}	y_{81}	z_{81}	x_{82}	y_{82}
z_{82}	x_{83}	y_{83}	z_{83}	x_{84}	y_{84}	z_{84}	x_{85}
y_{85}	z_{85}	x_{86}	y_{86}	z_{86}	x_{87}	y_{87}	z_{87}
x_{88}	y_{88}	z_{88}	x_{89}	y_{89}	z_{89}	x_{90}	y_{90}
z_{90}	x_{91}	y_{91}	z_{91}	x_{92}	y_{92}	z_{92}	x_{93}
y_{93}	z_{93}	x_{94}	y_{94}	z_{94}	x_{95}	y_{95}	z_{95}

Figure 4. resulting puncturing pattern

5. Conclusion

In this contribution, we propose an alternative puncturing algorithm for uplink using similar idea of [7]. This algorithm can satisfy all the requirements for turbo code puncturing without changing the conventional rate matching appraoch.

6. Reference

[1] 3GPP TSG RAN WG1 Multiplexing and Channel Coding(FDD) TS 25.212 V1.1.0 (1996. 06)

[2] 3GPP TSG RAN WG1 R1-99203 Optimised Rate Matching After Interleaving, Siemens.

[3] 3GPP TSG RAN WG1 R1-99703 Text Proposal for Optimised Puncturing, Siemens.

[4] 3GPP TSG RAN WG1 R1-99338 Puncturing Algorithm for Turbo Code, LGIC.

[5] 3GPP TSG RAN WG1 R1-99654 Comparison of Downlink Puncturing Algorithms, LGIC.

[6] 3GPP TSG RAN WG1 R1-99388 Optimised Puncturing Scheme for Turbo Coding, Fujitsu.

[7] 3GPP TSG RAN WG1 R1-99908 Code Symbol Based Uplink Puncturing Algorithm, LGIC