TSG-RAN Working Group1 meeting #11
San Diego, USA 29 February - 3 March 2000

TSGR1#11(00)0374

Source: **NTT DoCoMo**

Title: **Updated ANSI C-language program for Turbo code internal interleaver**

Document for: **Information**

Introduction

This document contains the updated ANSI C-language program capable of generating pattern of the latest approved Turbo code internal interleaver [1] of the specifications TS 25.212 and TS 25.222.

This program is capable of generating any Turbo code internal interleaving patterns with the size of 40-bit to 5114-bit with the granularity of 1-bit. The version number of this program is 3.0 and this version is corresponding to the approved specification text in [1]. The input of this program is block-size and the output is interleaving pattern. Each value of an interleaving pattern with a length of K, which is denoted by $A(k)$, $k = 0, 1, 2, ..., K - 1$, shows the input position $A(k)$ of the k-th output bit.

This program is provided for use only in standardisation working procedure of 3GPP. The relations between the approved specification texts of Turbo code internal interleaver and the versions of C-language program are shown in Table 1.

Table 1 Relation between approved specification text and program version

R1 documents		Version of C-program
Approved specification text	C-language program	
TSGR1#5(99)540	Distributed via R1-reflector on 9th April, 1999	Ver. 1.0
TSGR1#6(99)927	TSGR1#6(99)928	Ver. 2.0
TSGR1#6(99)a31	TSGR1#11(00)0374	Ver. 3.0
TSGR1#10(00)0160		

Reference

[1] NTT DoCoMo and Nortel Networks, “Modification of Turbo code internal interleaver”, TSGR1#11(00)0160.

C-language program for interleaving pattern generation

```
/*****************************************************************************

PROGRAM NAME:
 Prime_InterLeaver.c

VIRSION:
 3.0

DEVELOPER:
 NTT Mobile Communications Network Inc. / NTT DoCoMo Inc.
 3-5 Hikari-no-oka, Yokosuka-shi, Kanagawa, 239-8536 Japan

CONTACT POINT:
 Yukihiko OKUMURA
 E-mail: okumura@mlab.yrp.nttdocomo.co.jp

PURPOSE:
This program is provided for use only in standardisation working procedure of
3GPP.

DESCRIPTION:
This program is capable of generating any Turbo code internal interleaver patterns
with block size of from 40-bit to 5114-bit frame-size, with granularity of 1-bit.
The input of this program is frame-size and the output is interleaving pattern.
Each value of an interleaving pattern with a length of K, which is denoted by A(n),
n=0, 1, 2, ..., K-1, shows the input position A(n) of the n-th output bit.

REVISION HISTORY:
 Ver. 1.0: 8th, April 1999: Initial version created.
 Ver. 2.0: 9th, July 1999: Revised second stage and changed maximum block size.
 Ver. 3.0: 2nd, March 2000: Added pattern generation for smaller blocks with size
           of 40-bit to 319-bit.

Copyright(c)1999-2000 NTT Mobile Communications Network Inc., All rights reserved.
Copyright(c)2000 NTT DoCoMo Inc., All rights reserved.

*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

static void Get_pattern_primenumber ( int *pattern_primenumber, int M1 );
static void Init (void);
static void Read_ROM (void);
void malloc_error( char *message );

int M1, N1;
int plus1=0,minus1=0;
int root_of_primenumber[259];
int rotate_pattern[20];
int *PIPforN1;
int PIPforN1_1[10] = { 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };
int PIPforN1_2[20] = { 19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 16, 13, 17,
                       15, 3, 1,  6, 11,  8, 10 };
int PIPforN1_3[20] = { 19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 10,  8, 13,
                       17, 3, 1, 16,  6, 15, 11 };
int PIPforN1_4[5] = { 4, 3, 2, 1, 0 }; /* V3.0 */
int total_bit,total_bit2;
```

```
int main( int argc, char *argv[] ){
  int i,j;
  int *pattern_mil, *pattern_primenumber;
  int row_j[20];

  if ( argc == 2 ) {
    total_bit = atoi(argv[1]);
  } else {
    printf ("Usage: %s (bit_frame)\n", argv[0]);
    exit(0);
  }

  if (total_bit < 40 || total_bit > 5114 ) {
    printf("(bit_frame) must be smaller than 5115 or larger than 39!\n");
    exit(0);
  } /* V3.0 */

  Read_ROM();
  Init();
  if ( ( pattern_mil = (int *)malloc( total_bit * sizeof(int) ) )
       == NULL ) malloc_error( "pattern_mil" );
  if ( ( pattern_primenumber = (int *)malloc( M1 * sizeof(int) ) )
       == NULL ) malloc_error( "pattern_primenumber" );

  Get_pattern_primenumber ( pattern_primenumber, M1 );

  for ( j = 0 ; j < N1 ; j++) row_j[j]=0;

  for ( i = 0 ; i < M1 - minus1 + plus1; i++) {
    for ( j = 0 ; j < N1 ; j++) {
      if ( i == M1 ) {
      pattern_mil[j+i*N1] = M1 + (M1 + 1) * PIPforN1[j];
      } else if ( i == M1 - 1 ) {
      pattern_mil[j+i*N1] = 0 + (M1 + plus1) * PIPforN1[j];
      } else {
      pattern_mil[j+i*N1] = pattern_primenumber[row_j[j]]
        + (M1 - minus1 + plus1) * PIPforN1[j];
      row_j[j] = (row_j[j] + rotate_pattern[j])%(M1-1) ;
      }
    }
  }

  if ( total_bit == total_bit2 && plus1 == 1 ) {
    i = pattern_mil[total_bit-N1];
    pattern_mil[total_bit-N1] = pattern_mil[0];
    pattern_mil[0] = i;
  } /* V2.0 */


  for ( i = 0 ; i < total_bit ; i++) {
    if ( total_bit2 > pattern_mil[i] )
      printf ( "%d\n" , pattern_mil[i]/*+1*/);
  }

  free ( pattern_mil );
  free ( pattern_primenumber );
  exit(0);
}

static void Get_pattern_primenumber ( int *pattern_primenumber, int M1 )
{
```

```
  int i,j;

  for ( i=0, j=1; i< M1-1 ; i++){
    pattern_primenumber[i]=j-minus1;
    j = (j * root_of_primenumber[M1] )%M1;
  }
}

static void Init( void )
{
  int i,j;

  plus1=0; minus1=0;
  if ( (total_bit >= 2281 && total_bit <= 2480 ) ||
       (total_bit >= 3161 && total_bit <= 3210 ) ) {
    N1 = 20;
    PIPforN1 = PIPforN1_2;
  } else if  ( (total_bit >= 481 && total_bit <= 530) ||
             (total_bit >= 160 && total_bit <= 200) ) {
    N1 = 10;
    PIPforN1 = PIPforN1_1;
  } else if  (total_bit >= 40 && total_bit <= 159){
    N1 = 5;
    PIPforN1 = PIPforN1_4;
  } else {
    N1 = 20;
    PIPforN1 = PIPforN1_3;
  } /* V3.0 */

  total_bit2=total_bit;

  M1 = total_bit / N1;

  if ( total_bit % N1 > 0 ) {
    M1+=1;
    total_bit     = M1 * N1;
  }
  if ( root_of_primenumber[M1] == 0 ) {
    if ( root_of_primenumber[M1 - 1] > 0) {
      M1 -= 1;
      plus1 = 1;
    } else {
      for ( i = 1 ; i < 20 ; i++ ) {
      if ( root_of_primenumber[M1 + i] > 0 ) {
        if ( M1 + i >=410 ) {
          printf ( "M1 must be less than 410" );
          exit(0);
        }
        M1 = M1 + i ;
        minus1 = 1;
        break;
      }
      }
      total_bit   = (M1-1) * N1;
    }
  }

  if ( minus1 == 1 && total_bit2 >= 481 && total_bit2 <= 530 ) {
    minus1 = 0;
    total_bit     = M1 * N1;
  }
```

```
  rotate_pattern[0]=1;
  for ( i = 7,j = 1 ; i < 100 ; i++) {
    if ( root_of_primenumber[i] > 0 ) {
      if ( (M1 - 1) % i > 0 ) {
      rotate_pattern[j++]=i;
      }
    }
    if ( j >= N1 ) break;
  }
}

static void Read_ROM( void )
{
  int i;
  for (i = 0 ; i < 259 ; i++) root_of_primenumber[i]=0;
  root_of_primenumber[2] = 1;
  root_of_primenumber[3] = 2;
  root_of_primenumber[5] = 2;
  root_of_primenumber[7] = 3;
  root_of_primenumber[11] = 2;
  root_of_primenumber[13] = 2;
  root_of_primenumber[17] = 3;
  root_of_primenumber[19] = 2;
  root_of_primenumber[23] = 5;
  root_of_primenumber[29] = 2;
  root_of_primenumber[31] = 3;
  root_of_primenumber[37] = 2;
  root_of_primenumber[41] = 6;
  root_of_primenumber[43] = 3;
  root_of_primenumber[47] = 5;
  root_of_primenumber[53] = 2;
  root_of_primenumber[59] = 2;
  root_of_primenumber[61] = 2;
  root_of_primenumber[67] = 2;
  root_of_primenumber[71] = 7;
  root_of_primenumber[73] = 5;
  root_of_primenumber[79] = 3;
  root_of_primenumber[83] = 2;
  root_of_primenumber[89] = 3;
  root_of_primenumber[97] = 5;
  root_of_primenumber[101] = 2;
  root_of_primenumber[103] = 5;
  root_of_primenumber[107] = 2;
  root_of_primenumber[109] = 6;
  root_of_primenumber[113] = 3;
  root_of_primenumber[127] = 3;
  root_of_primenumber[131] = 2;
  root_of_primenumber[137] = 3;
  root_of_primenumber[139] = 2;
  root_of_primenumber[149] = 2;
  root_of_primenumber[151] = 6;
  root_of_primenumber[157] = 5;
  root_of_primenumber[163] = 2;
  root_of_primenumber[167] = 5;
  root_of_primenumber[173] = 2;
  root_of_primenumber[179] = 2;
  root_of_primenumber[181] = 2;
  root_of_primenumber[191] = 19;
  root_of_primenumber[193] = 5;
  root_of_primenumber[197] = 2;
  root_of_primenumber[199] = 3;
  root_of_primenumber[211] = 2;
```

```
  root_of_primenumber[223] = 3;
  root_of_primenumber[227] = 2;
  root_of_primenumber[229] = 6;
  root_of_primenumber[233] = 3;
  root_of_primenumber[239] = 7;
  root_of_primenumber[241] = 7;
  root_of_primenumber[251] = 6;
  root_of_primenumber[257] = 3;
}

void malloc_error( char *message )
{
      fprintf( stderr, "malloc %s failed\n", message );
      exit( 1 );
}
```