TSG-RAN Working Group1 meeting #6
Espoo, Finland 13-16, July 1999

TSGR1#6(99)928

Source: **NTT DoCoMo**

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

Document for: **Information**

Introduction

This document contains the ANSI C-language program capable of generating pattern of approved Turbo code internal interleaver [1] (the description will also be updated to make a few editorial changes [2]). This program could be used as a reference for the accurate interpretation of the Turbo code internal interleaver specification. Also, this program is provided for use only in standardization working procedure of 3GPP.

Program for interleaving pattern generation

- ANSI C-language program in **A.1**:
This program is capable of generating any Turbo code internal interleaver from 320-bit to 5120-bit frame-size with granularity of 1-bit (the version number of this program is 2.0 and this version is corresponding to the description in TSGR1#6(99)927). 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.

- Generated pattern example in **A.2**:
This is an example of Turbo-code internal interleaving pattern generated by the above C-language program. The pattern has a frame-size of 320-bit.

References

[1] TS 25.212 “Multiplexing and channel coding (FDD)” and TS 25.222 “Multiplexing and channel coding (TDD)”
[2] NTT DoCoMo, Nortel Networks, SAMSUNG Electronics Co., “Updated text proposal for Turbo code internal interleaver”, TSGR1#6(99)927

A.1 C-language program for interleaving pattern generation

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

PROGRAM NAME:
 Prime_InterLeaver.c

VIRSION:
 2.0

DEVELOPER:
 NTT Mobile Communications Network Inc. (NTT DoCoMo)
 3-5 Hikari-no-oka, Yokosuka-shi, Kanagawa, 239-8536 Japan
 Tel: +81 468 40 3190
 Fax: +81 468 40 3840

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

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

DESCRIPTION:
This program is capable of generating any Turbo code internal interleavers
from 320-bit to 5120-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.

Copyright(c)1999 NTT Mobile Communications Network 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 }; /*Pattern C, R{10}*/
int PIPforN1_2[20] = { 19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 16, 13, 17,
                       15, 3, 1,  6, 11,  8, 10 };  /* Pattern B */
int PIPforN1_3[20] = { 19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 10,  8, 13,
                       17, 3, 1, 16,  6, 15, 11 };  /* Pattern A */
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 < 320 || total_bit > 5120 ) {
            printf("(bit_frame) must be smaller than 5121 or larger than 319!\n");
            exit(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;
      }


      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 ) {
		N1 = 10;
		PIPforN1 = PIPforN1_1;
	} else {
		N1 = 20;
		PIPforN1 = PIPforN1_3;
	}
	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 );
}
```

A.2 Generated pattern example (320-bit interleaving size)

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

FILE NAME:
 Prime_InterLeaver_320.txt

DESCRIPTION:
This text is the example of Turbo-code internal interleaving pattern generated
by “Prime_InterLeaver.c”.  The pattern has a frame-size of 320-bit.

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

305             194             260             11              219             308
145             296             100             37              284             157
225             169             253             83              58              237
65              143             180             119             21              68
1               210             309             206             263             4
33              280             154             298             102             45
81              63              235             166             243             93
113             24              67              140             187             116
193             271             5               222             312             205
289             98              38              282             159             292
161             249             90              60              226             164
129             178             115             23              73              141
209             314             199             261             8               221
273             149             302             99              34              276
49              227             172             246             95              61
17              75              139             190             121             20
257             10              215             320             207             269
97              46              286             160             297             109
241             85              59              240             168             244
177             123             19              80              130             189
307             204             262             16              223             316
155             294             106             48              281             151
231             167             252             96              50              230
76              131             183             128             25              78
3               220             319             208             258             12
42              278             152             304             111             43
91              51              233             176             248             87
124             27              66              144             191             126
198             270             15              224             311             202
293             101             41              288             156             291
174             247             88              64              238             165
135             188             114             32              70              134
214             317             200             272             7               218
277             148             290             112             35              275
55              228             175             256             92              54
28              77              137             192             118             30
266             13              216             318             197             267
107             36              274             150             299             103
254             84              57              234             170             245
182             125             18              69              142             186
313             196             265             14              213             306
146             301             104             39              283             153
239             173             255             86              62              232
72              132             184             117             22              79
9               212             315             203             259             2
47              285             147             300             108             40
82              52              236             163             250             89
120             29              71              138             181             127
```

201
303
162
136
217
287
56
31
264
105
242
185
310
158
229
74
6
44
94
122
195
295
171
133
211
279
53
26
268
110
251
179