TS 25.213 V2.1.~~0~~2 (1999-4)

Technical Specification

3rd Generation Partnership Project (3GPP);
Technical Specification Group (TSG)
Radio Access Network (RAN);
Working Group 1 (WG1);
Spreading and modulation (FDD)

Reference

<Workitem> (25_213-xxx.PDF)

Keywords

<keyword[, keyword]>

3GPP

Postal address

Office address

Internet

secretariat@3gpp.org

Individual copies of this deliverable
can be downloaded from
http://www.3gpp.org

Contents

Intellectual Property Rights

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project, Technical Specification Group Radio Access Network, Working Group 1.

The contents of this TS may be subject to continuing work within the 3GPP and may change following formal TSG approval. Should the TSG modify the contents of this TS, it will be re-released with an identifying change of release date and an increase in version number as follows:

Version m.t.e

where:

m	indicates [major version number]
x	the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
y	the third digit is incremented when editorial only changes have been incorporated into the specification.

1 Scope

The present document describes spreading and modulation for UTRA Physical Layer FDD mode.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[<seq>]	<doctype> <#>[([up to and including]{yyyy[-mm]|V<a[.b[.c]]>}[onwards])]: "<Title>".
[1]	EN 301 234 (V2.1 onwards): "Example 1, using sequence field".
[2]	EG 201 568 (V1.3.5): "Example 2, using fixed text".

<doctype> <#>[([up to and including]{yyyy[-mm]|V<a[.b[.c]]>}[onwards])]: "<Title>".

EN 301 234 (V2.1 onwards): "Example 1".

EG 201 568 (V1.3.5): "Example 2".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply.

3.2 Symbols

For the purposes of the present document, the following symbols apply:

<symbol>	<Explanation>

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AP	Access Preamble
BCH	Broadcast Control Channel
BER	Bit Error Rate
BS	Base Station
CCPCH	Common Control Physical Channel
CD	Collision Detection
CPCH	Common Packet Channel
DCH	Dedicated Channel

DL	Downlink
DPCH	Dedicated Physical Channel
DPCCH	Dedicated Physical Control Channel
DPDCH	Dedicated Physical Data Channel
DS-CDMA	Direct-Sequence Code Division Multiple Access
FACH	Forward Access Channel
FDD	Frequency Division Duplex
Mcps	Mega Chip Per Second
MS	Mobile Station
OVSF	Orthogonal Variable Spreading Factor (codes)
PCH	Paging Channel
PCPCH	Physical Common Packet Channel
PG	Processing Gain
PRACH	Physical Random Access Channel
RACH	Random Access Channel
RX	Receive
SCH	Synchronisation Channel
SF	Spreading Factor
SIR	Signal-to-Interference Ratio
TDD	Time Division Duplex
TFCI	Transport-Format Combination Indicator
TPC	Transmit Power Control
TX	Transmit
UE	User Equipment
UL	Uplink

4 Uplink spreading and modulation

4.1 Overview

Spreading is applied after modulation~~ and before pulse shaping~~. It consists of two operations. The first is the channelization operation, which transforms every data symbol into a number of chips, thus increasing the bandwidth of the signal. The number of chips per data symbol is called the Spreading Factor (SF). The second operation is the scrambling operation, where a scrambling code is applied to the spread signal.

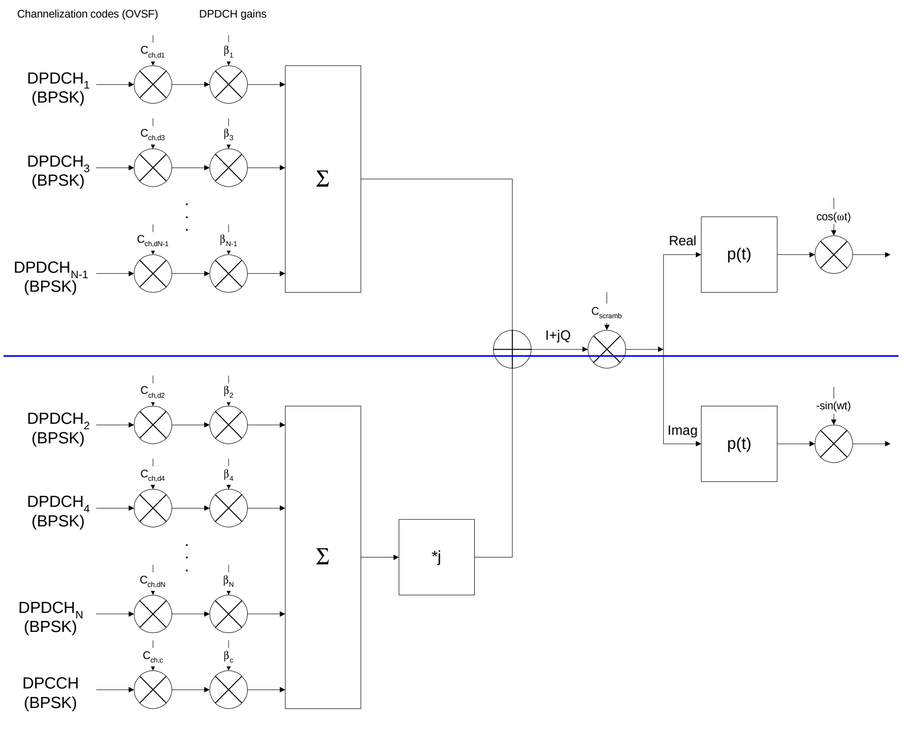
With the channelization, data symbol on so-called I- and Q-branches are independently multiplied with an OVSF code. With the scrambling operation, the resultant signals on the I- and Q-branches are further multiplied by complex-valued scrambling code, where I and Q denote real and imaginary parts, respectively. Note that before complex multiplication binary values 0 and 1 are mapped to +1 and -1, respectively.

4.2 Spreading

4.2.1 Uplink Dedicated Physical Channels (uplink DPDCH/DPCCH)

Figure 1 illustrates the spreading and modulation for the case of multiple uplink DPDCHs when total data rate is less than or equal to 1024kbps in the 5MHz band. Note that this figure only shows the principle, and does not necessarily describe an actual implementation. Figure 2 illustrates the case for data rate at 2048kbps in the 5 MHz band. Modulation is dual-channel QPSK (i.e.; separate BPSK on I- and Q-channel), where the uplink DPDCH and DPCCH are mapped to the I and Q branch respectively. The I and Q branches are then spread to the chip rate with two different channelization codes and subsequently complex scrambled by a UE specific complex scrambling code C_{scramb}.

<Editorš note: the data rates in this section should be reviewed following the chip rate change to 3.84Mcps.>



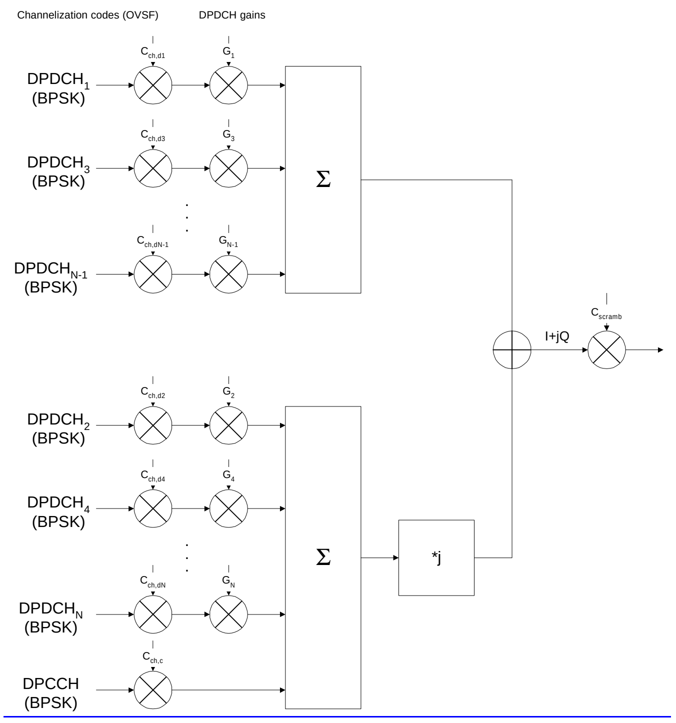
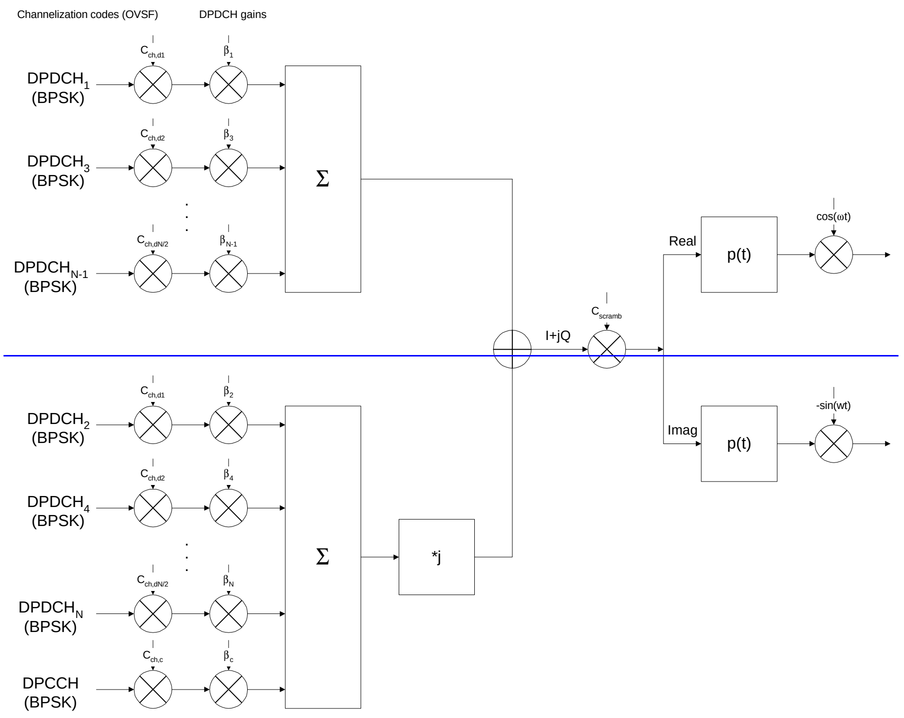


Figure 1 Spreading/modulation for uplink DPDCH/DPCCH for user services less than or equal to 1024kbps in the 5MHz band



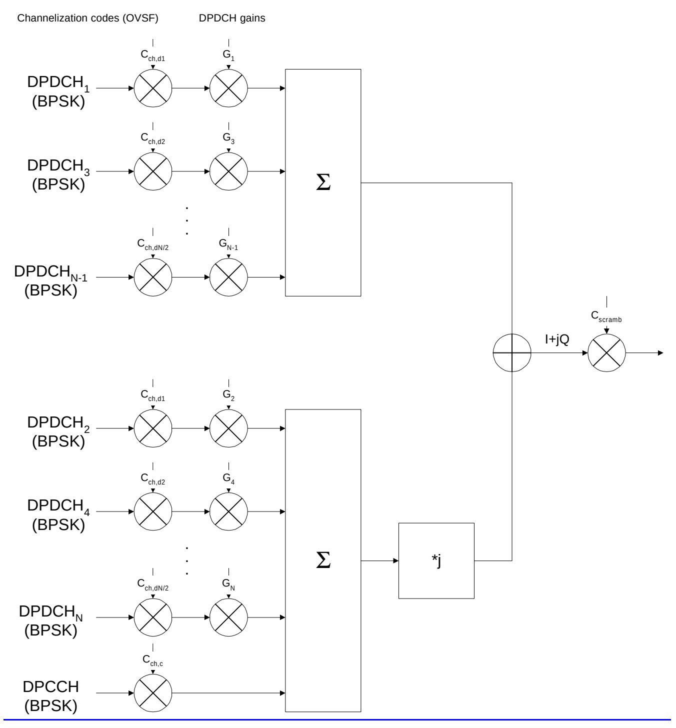


Figure 2. Spreading/modulation for uplink DPDCH/DPCCH for user services at 2048kbps in the 5MHz band

<Editor's note: pulse shaping will be moved to appropriate WG4 documentation.>

For a single uplink DPDCH transmission, only $DPDCH_1$ and DPCCH are transmitted.~~]~~

For services less than or equal to 1024kbps in the 5MHz band, the DPCCH is spread by the channelization code $C_{ch,c}$ and each $DPDCH_i$ is spread by a predefined individual channelization codes, $C_{ch,di}$ (di=1,2,).. For 2048kbps rate in the 5MHz band, the DPCCH is spread by the channelization code $C_{ch,c}$ and each pair of $DPDCH_{2di-1}$ and $DPDCH_{2di}$ is spread by a predefined individual channelization codes, $C_{ch,di}$. The data symbols of both the DPDCHs and the DPCCH are BPSK-modulated and the channelization codes are real-valued. The real-valued signals of the I- and Q-branches are then summed and treated as a complex signal. This complex signal is then scrambled by the complex-valued scrambling code, C_{scramb}. The powers of the DPDCHs may be adjusted by gain factors, β_c, β_{di} .

The channel with maximum power has always $b_i \equiv 1.0$ and the others have $b_i \leq 1.0$, where i is in the range 1, 2, .. N, c The β-values are quantized into 4 bits, and the quantization steps are given in Table 1.

	Quantized amplitude ratio (b_{quant})
15	1.0
14	0.9375
13	0.875
12	0.8125
11	0.75
10	0.6875
9	0.625
8	0.5625
7	0.5
6	0.4375
5	0.375
4	0.3125
3	0.25
2	0.1875
1	0.125
0	Switch off

Table 1: The quantization of the gain parameters.

4.2.2 PRACH

The spreading and modulation of the message part of the Random-Access message part~~burst~~ is basically the same as for the uplink dedicated physical channels, see Figure 1, where the uplink DPDCH and uplink DPCCH are replaced by the data part and the control part respectively. The scrambling code for the message part is chosen based on the~~ base-station-specific~~ preamble code.

4.3 Code generation and allocation

4.3.1 Channelization codes

The channelization codes of Figure 1 are Orthogonal Variable Spreading Factor (OVSF) codes that preserve the orthogonality between a user's different physical channels. The OVSF codes can be defined using the code tree of Figure 3.

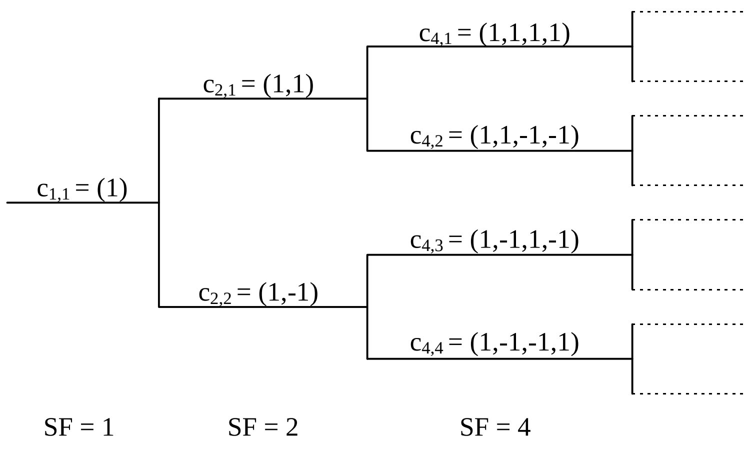


Figure 3. Code-tree for generation of Orthogonal Variable Spreading Factor (OVSF) codes.

In Figure 3, the OVSF code is described as $C_{SF,\text{code number}}$, where $SF_{d,n}$ represents the spreading factor of n^{th} DPDCH. Then the DPCCH is spread by code number 1 with a spreading factor of SF_c.

Each level in the code tree defines channelization codes of length SF, corresponding to a spreading factor of SF in Figure 3. All codes within the code tree cannot be used simultaneously by one mobile station. A code can be used by a UE if and only if no other code on the path from the specific code to the root of the tree or in the sub-tree below the specific code is used by the same mobile station. This means that the number of available channelization codes is not fixed but depends on the rate and spreading factor of each physical channel.

The generation method for the channelization code can also be explained in Figure 4.

$$C_{1,1} = 1$$

$$\begin{bmatrix} C_{2,1} \\ C_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,1} \\ C_{1,1} & \overline{C_{1,1}} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} C_{4,1} \\ C_{4,2} \\ C_{4,3} \\ C_{4,4} \end{bmatrix} = \begin{bmatrix} C_{2,1} & C_{2,1} \\ C_{2,1} & \overline{C_{2,1}} \\ C_{2,2} & C_{2,2} \\ C_{2,2} & \overline{C_{2,2}} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$: :$$

$$\begin{bmatrix} C_{2^{n+1},1} \\ C_{2^{n+1},2} \\ C_{2^{n+1},3} \\ C_{2^{n+1},4} \\ : \\ C_{2^{n+1},2^{n+1}-1} \\ C_{2^{n+1},2^{n+1}} \end{bmatrix} = \begin{bmatrix} C_{2^n,1} & C_{2^n,1} \\ C_{2^n,1} & \overline{C_{2^n,1}} \\ C_{2^n,2} & C_{2^n,2} \\ C_{2^n,2} & \overline{C_{2^n,2}} \\ : & : \\ C_{2^n,2^n} & C_{2^n,2^n} \\ C_{2^n,2^n} & \overline{C_{2^n,2^n}} \end{bmatrix}$$

Figure 4. Spreading Code Generation Method

Binary code words are equivalent to the real valued sequences by the transformation ‘0’-> ‘+1’, ‘1’-> ‘-1’.

The spreading code cycle is the symbol cycle. Thus, for a given chip rate, the spreading code cycle depends on the symbol rate. Furthermore, the number of codes that can be used also differs according to the symbol rate. The relations between symbol rate, spreading code types, spreading code cycle and number of spreading codes is listed in Table 2.

The spreading code phase synchronises with the modulation/demodulation symbols. In other words, the head chip of the symbol is spreading code phase=0.

Symbol rate (ksps)				spreading code cycle(chip) SF	No. of Spreading codes
Chip rate=[1.024 Mcps]	4.096 Mcps	[8.192 Mcps]	[16.384 Mcps]		
[240~~56~~]	960~~1024~~	[1920~~2048~~]	[3840~~4096~~]	4	4
[120~~8~~]	480~~512~~	[960~~1024~~]	[1920~~2048~~]	8	8
[60~~4~~]	240~~56~~	[480~~512~~]	[960~~1024~~]	16	16
[30~~2~~]	120~~8~~	[240~~56~~]	[480~~512~~]	32	32
[15~~6~~]	60~~4~~	[120~~8~~]	[240~~56~~]	64	64
[7.5~~8~~]	30~~2~~	[60~~4~~]	[120~~8~~]	128	128
-	15~~6~~	[30~~2~~]	[60~~4~~]	256	256
-	[7.5~~8~~]	[15~~6~~]	[30~~2~~]	512	512
-	-	[7.5~~8~~]	[15~~6~~]	1024	1024
			[7.5~~8~~]	2048	2048

Table 2. Correspondence between Symbol Rate and Spreading Code Types

The DPCCH is spread by code number 1 in any code tree as described in Section 4.3.1. The first DPDCH is spread by code number ($SF_{d,1}$ / 4 + 1). Subsequently added DPDCHs for multi-code transmission are spread by codes in ascending order starting from code number 2 excepting the one used for the first DPDCH. However to guarantee the orthogonality between channels, any subtree below the specified node is not used for the channelization code of a DPDCH.

<Editor's Note: The case of OVSF code allocation with multiple DPDCHs with different spreading factors is for further study

4.3.2 Scrambling codes

4.3.2.1 General

There are 2^{24} uplink scrambling codes. Either short or long scrambling codes should be used on the uplink. The short scrambling code is typically used in cells where the base station is equipped with an advanced receiver, such as a multi-user detector or interference canceller. With the short scrambling code the cross-correlation properties between different physical channels and users does not vary in time in the same way as when a long code is used. In cells where there is no gain in implementation complexity using the short scrambling code, the long code is used instead due to its better interference averaging properties. Both short and long scrambling codes are represented with complex-value.

The uplink scrambling generator (either short or long) shall be initialised by a 25 bit value. One bit shall indicate selection of short or long codes (short = 1, long = 0). Twenty four bits shall be loaded into the scrambling generators as shown in sections 4.3.2.2 and 4.3.2.3.

MSB — Initialisation Code Short/Long flag + Value *v* — LSB

short /long	v(23)	v(22)	v(21)	v(20)	v(19)	v(18)	v(17)	v(16)	v(15)	v(14)	v(13)	v(12)	v(11)	v(10)	v(9)	v(8)	v(7)	v(6)	v(5)	v(4)	v(3)	v(2)	v(1)	v(0)

Figure 5 - Initialisation Code for Uplink Scrambling generator

[Alternatively, if the system chooses, RSTS for uplink transmission, the scrambling code is the same as the downlink scrambling code described in ~~0~~5.2.2. In this case, the same scrambling code is allocated to all dedicated physical channels in the cell.]

Both short and long scrambling codes are formed as follows:

$$C_{scramb} = c_1(w_0 + jc_2'w_1)$$

where w_0 and w_1 are chip rate sequences defined as repetitions of:

w_0 = {1 1}

w_1 = {1 -1}

Also, c_1 is a real chip rate code, and c_2'is a decimated version of the real chip rate code c_2. The preferred decimation factor is 2, however other decimation factors should be possible in future evolutions of 3GPP if proved desirable.

With a decimation factor 2, c_2'is given as:

$$c_2(2k) = c_2(2k+1) = c_2(2k), \quad k=0,1,2\ldots$$

The constituent codes c_1 and c_2 are formed differently for the short and long scrambling codes as described in Sections 4.3.2.2 and 4.3.2.3.

4.3.2.2 Long scrambling code

The long scrambling codes are formed as described in Section 4.3.2, where c_1 and c_2 are constructed as the position wise modulo 2 sum of 38400~~40960~~ chip segments of two binary *m*-sequences generated by means of two generator polynomials of degree 25. Let *x*, and *y* be the two *m*-sequences respectively. The *x* sequence is constructed using the primitive (over GF(2)) polynomial $X^{25}+X^3+1$. The *y* sequence is constructed using the polynomial $X^{25}+X^3+X^2+X+1$. The resulting sequences thus constitute segments of a set of Gold sequences.

The code, c_2, used in generating the quadrature component of the complex spreading code is a 16,777,232 chip shifted version of the code, c_1, used in generating the in phase component.

The uplink scrambling code word has a period of one radio frame of 10 ms.

Let $n_{23} \dots n_0$ be the 24 bit binary representation of the scrambling code number *n* (decimal) with n_0 being the least significant bit. The *x* sequence depends on the chosen scrambling code number *n* and is denoted x_n, in the sequel. Furthermore, let $x_n(i)$ and $y(i)$ denote the *i*:th symbol of the sequence x_n and *y*, respectively

The *m*-sequences x_n and *y* are constructed as:

Initial conditions:

$x_n(0)=n_0$, $x_n(1)= n_1$, … $=x_n(22)= n_{22}$,$x_n(23)= n_{23}$, $x_n(24)=1$

$y(0)=y(1)= \dots =y(23)= y(24)=1$

Recursive definition of subsequent symbols:

$x_n(i+25) =x_n(i+3) + x_n(i)$ *modulo 2, i=0,…,* 2^{25}-~~43~~27,

$y(i+25) = y(i+3)+y(i+2) +y(i+1) +y(i)$ *modulo 2, i=0,…,* $2^{25}-27$.

The definition of the *n*:th scrambling code word for the in phase and quadrature components follows as (the left most index correspond to the chip scrambled first in each radio frame):

$c_{1,n} = < x_n(0)+y(0), x_n(1)+y(1), \dots,x_n(N-1)+y(N-1) >$,

$c_{2,n} = < x_n(M)+y(M), x_n(M+1)+y(M+1), \dots, x_n(M+N-1) + y(M+N-1) >$,

again all sums being modulo 2 additions. (Both N and M are defined in .)

These binary code words are converted to real valued sequences by the transformation ‘0’-> ‘+1’, ‘1’-> ‘-1’.

~~<*Editor’s note:* 2^{24} *– 1 is FFS.*>~~

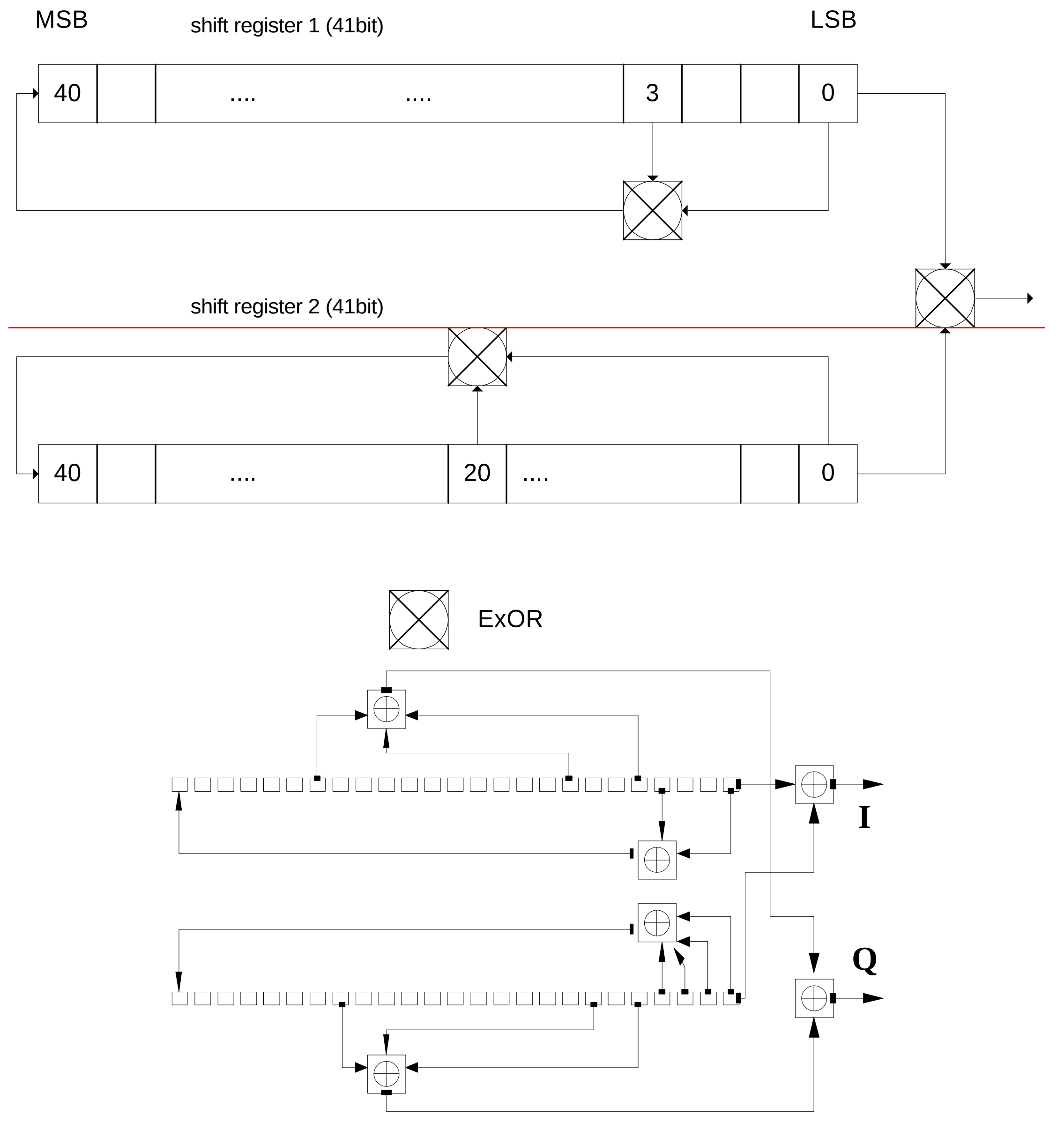


Figure 6. Configuration of uplink scrambling code generator

~~Chip rate (Mcps)~~	~~Period N (chips)~~	~~I/Q Offset M (chips)~~	~~Range of phase (chip)~~	
			~~(c_1)~~	~~(c_2)~~
~~[1.024~~	~~10240~~	~~896~~~~16777232~~	~~0 … N-1~~	~~M … N+(M-1)~~
~~4.096~~	~~40960~~	~~3584~~~~16777232~~		
~~[8.192~~	~~81920~~	~~7168~~~~16777232~~		
~~[16.384~~	~~163840~~	~~14336~~~~16777232~~		

~~**Table 3. Correspondence between chip rate and uplink scrambling code phase range**~~

4.3.2.3 Short scrambling code

The short scrambling codes are formed as described in Section 4.3.2.1,where c1 and c2 are the real and imaginary components of the complex spreading code from the family of periodically extended S(2) codes.

The uplink short codes $S_v(n)$, n=0,1,255, of length 256 chips are obtained by one chip periodic extension of S(2) sequences of length 255. It means that the first chip ($S_v(0)$) and the last chip ($S_v(255)$) of any uplink short scrambling code are the same.

The quaternary S(2) sequence $z_v(n)$, $0 \le v \le 16{,}777{,}216$, of length 255 is obtained by modulo 4 addition of three sequences, a quaternary sequence $a_r(n)$ and two binary sequences $b_s(n)$ and $c_t(n)$, according to the following relation:

$$z_v(n) = a_r(n) + 2 \cdot b_s(n) + 2 \cdot c_t(n) \quad (mod\ 4)\ , \quad n = 0, 1, , 254.$$

The user index v determines the indexes r, s, and t of the constituent sequences in the following way:

$$v = t \cdot 2^{16} + s \cdot 2^{8} + r\ ,$$

$r = 0, 1, 2, , 255,$

$s = 0, 1, 2, , 255,$

$t = 0, 1, 2, , 255.$

The quaternary sequence $a_r(n)$ is generated by the recursive generator G_0 defined by the polynomial

$g_0(x)= x^8+x^5+3x^3+x^2+2x+1$ as

$$a_r(n)= 3.a_r(n-3) +1.a_r(n-5) + 3.a_r(n-6) +2.a_r(n-7) + 3.a_r(n-8) \quad (mod\ 4).$$

n = ~~0, 1, 2, , 255~~8254.

The binary sequence $b_s(n)$ is generated by the recursive generator G_1 defined by the polynomial

$g_1(x)= x^8+x^7+x^5+x+1$ as

$$b_s(n)= b_s(n-1)+ b_s(n-3)+ b_s(n-7)+ b_s(n-8) \quad (mod\ 2).$$

The binary sequence $c_t(n)$ is generated by the recursive generator G_2 defined by the polynomial

$g_2(x)= x^8+x^7+x^5+x^4+1$ as

$$c_t(n)= c_t(n-1)+ c_t(n-3)+ c_t(n-4)+ c_t(n-8) \quad (mod\ 2).$$

An implementation of the short scrambling code generator is shown in Figure 7. The initial states for the binary generators G_1 and G_2 are the two 8-bit words representing the indexes s and t in the 24-bit binary representation of the user index v, as it is shown in Figure 8.

The initial state for the quaternary generator G_0 is according to Figure 8. obtained after the transformation of 8-bit word representing the index r. This transformation is given by

$$a_r(0) = 2v(0)+1 \quad (mod\ 4), \qquad a_r(n) = 2v(n) \quad (mod\ 4), \qquad n = 1,,7.$$

The complex quadriphase sequence $S_v(n)$ is obtained from quaternary sequence $z_v(n)$ by the mapping function given in Table 4.

The Re{Sv(n)} and Im{Sv(n)} of the S(2) code are the pair of two binary sequences corresponding to input binary sequences c_1 and c_2 respectively described in 4.3.2.

zv(n)	Sv(n)
0	+1 + j1
1	-1 + j1
2	-1 - j1
3	+1 - j1

Table 4. Mapping between $S_v(n)$ and $z_v(n)$

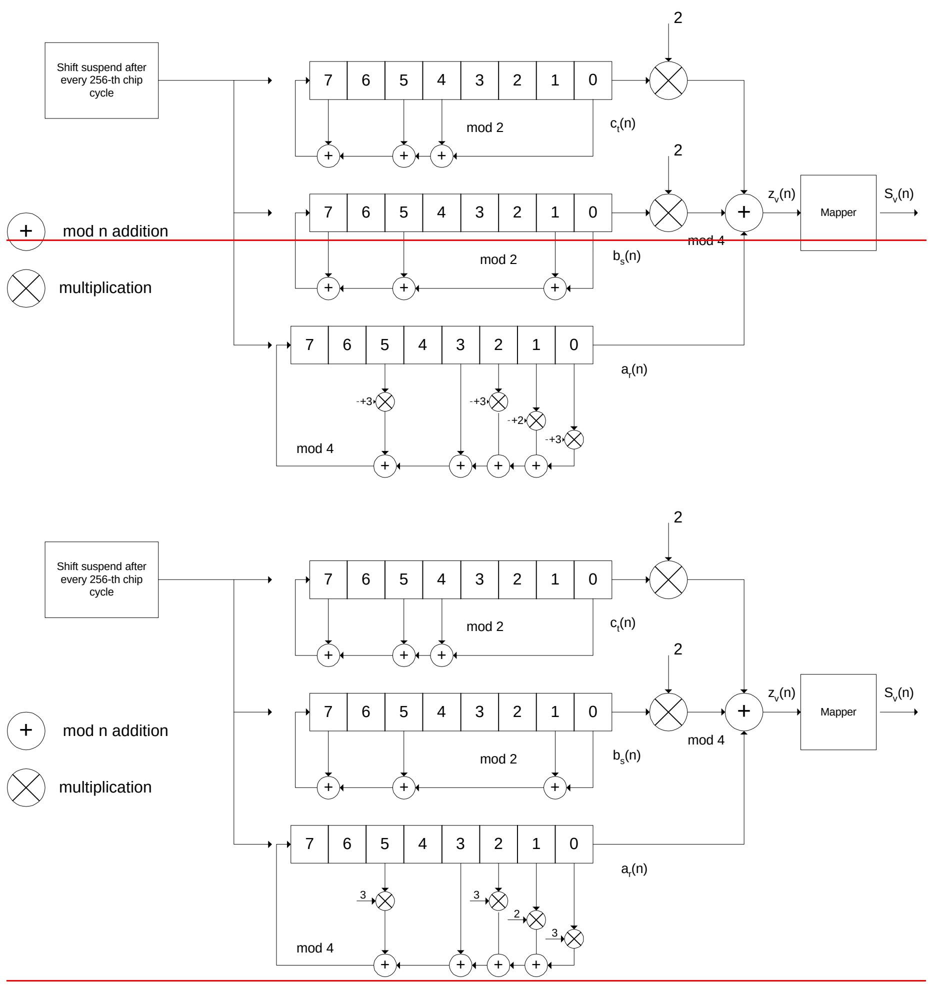


Figure 7. Uplink short scrambling code generator

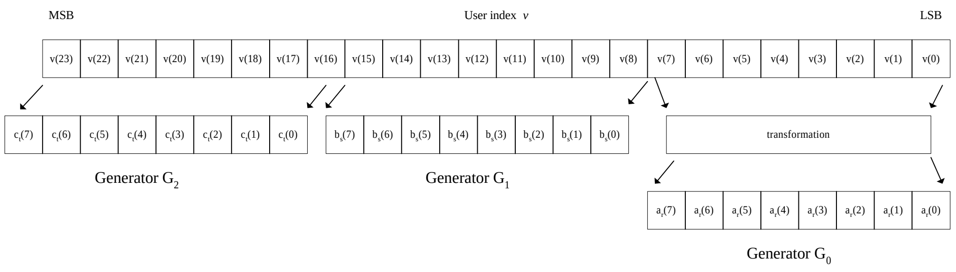

MSB
User index v
LSB
v(23) v(22) v(21) v(20) v(19) v(18) v(17) v(16) v(15) v(14) v(13) v(12) v(11) v(10) v(9) v(8) v(7) v(6) v(5) v(4) v(3) v(2) v(1) v(0)
c(7) c(6) c(5) c(4) c(3) c(2) c(1) c(0)
b(7) b(6) b(5) b(4) b(3) b(2) b(1) b(0)
transformation
Generator G2
Generator G1
a(7) a(6) a(5) a(4) a(3) a(2) a(1) a(0)
Generator G0

Figure 8. Uplink short scrambling code generator state initialisation

The short scrambling code may, in rare cases, be changed during a connection.

4.3.3 Random access codes

4.3.3.1 Preamble scrambling~~spreading~~ code

The scrambling~~spreading~~ code for the preamble part is as follows~~cell specific and is broadcast by the base station~~. ~~More than one preamble code can be used in a base station if the traffic load is high. The preamble codes must be code planned, since two neighbouring cells should not use the same preamble code.~~

The code generating method is the same as for the real part of the long codes on dedicated channels. Only the first 4096 chips of the code are used for preamble spreading with the chip rate of 3.84 Mchip/s. The long code c1 for the in-phase component is used directly on both in phase and quadrature branches without offset between branches. The preamble scrambling code is defined as the position wise modulo 2 sum of 4096 chips segments of two binary m-sequences generated by means of two generator polynomials of degree 25. Let x and y be the two m-sequences respectively. The x sequence is constructed using the primitive (over GF(2)) polynomial $X^{25}+X^3+1$. The y sequence is constructed using the polynomial $X^{25}+X^3+X^2+X+1$. The resulting sequences thus constitute segments of a set of Gold sequences.

~~The code used is a real-valued 256 chip Orthogonal Gold code. All 256 codes are used in the system.~~

~~The code sequences are constructed with the help of two binary *m*-sequences of length 255, *x*, and *y*, respectively. The *x* sequence is constructed using the polynomial $1+X^2+X^3+X^4+X^8$. The *y* sequence is constructed using the polynomial $1+X^3+X^5+X^6+X^8$.~~

Let $n_7 \ldots n_0$ be the binary representation of the code number *n* (decimal) with n_0 being the least significant bit. ~~The *x* sequence depends on the chosen code number *n* and is denoted x_n in the sequel. Furthermore, let $x_n(i)$ and $y(i)$ denote the *i*:th symbol of the sequence x_n and *y*, respectively~~

The *m*-sequences x_n and *y* are constructed as:

Initial conditions:

$x_n(0)=n_0$, $x_n(1)=n_1$, $\ldots=x_n(6)=n_6$, $x_n(7)=n_7$, $x_n(8)=0,\ldots,x_n(22)=0, x_n(23)=1, x_n(24)=0$

$y(0)=y(1)=\ldots=y(23$~~6~~$)=y(24$~~7~~$)=1$

Recursive definition of subsequent symbols:

$x_n(i+25)=x_n(i+3)+x_n(i)$ modulo 2, $i=0,\ldots,4070$,

$y(i+25)=y(i+3)+y(i+2)+y(i+1)+y(i)$ modulo 2, $i=0,\ldots,4070$.

~~$x_n(i+8)=x_n(i+4)+x_n(i+3)+x_n(i+2)+x_n(i)$ modulo 2, $i=0,\ldots,246$,~~

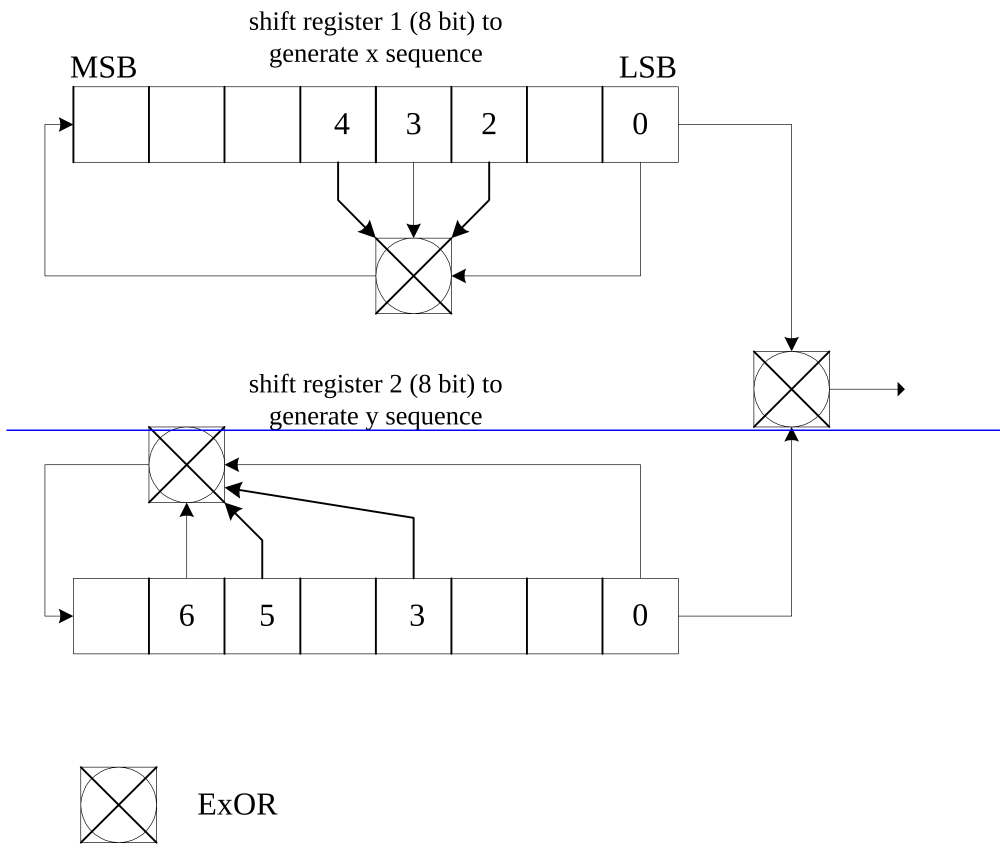
~~$y(i+8)=y(i+6)+y(i+5)+y(i+3)+y(i)$ modulo 2, $i=0,\ldots,246$.~~

The definition of the *n*:th code word follows (the left most index correspond to the chip transmitted first in each slot):

$C_{RACH,n}=<$~~0,~~$x_n(0)+y(0), x_n(1)+y(1),\ldots,x_n(4095$~~254~~$)+y(4095$~~254~~$)>$,

All sums of symbols are taken modulo 2.

The preamble spreading code is described in Figure 9.


shift register 1 (8 bit) to generate x sequence
MSB
LSB
4
3
2
0
shift register 2 (8 bit) to generate y sequence
6
5
3
0
ExOR

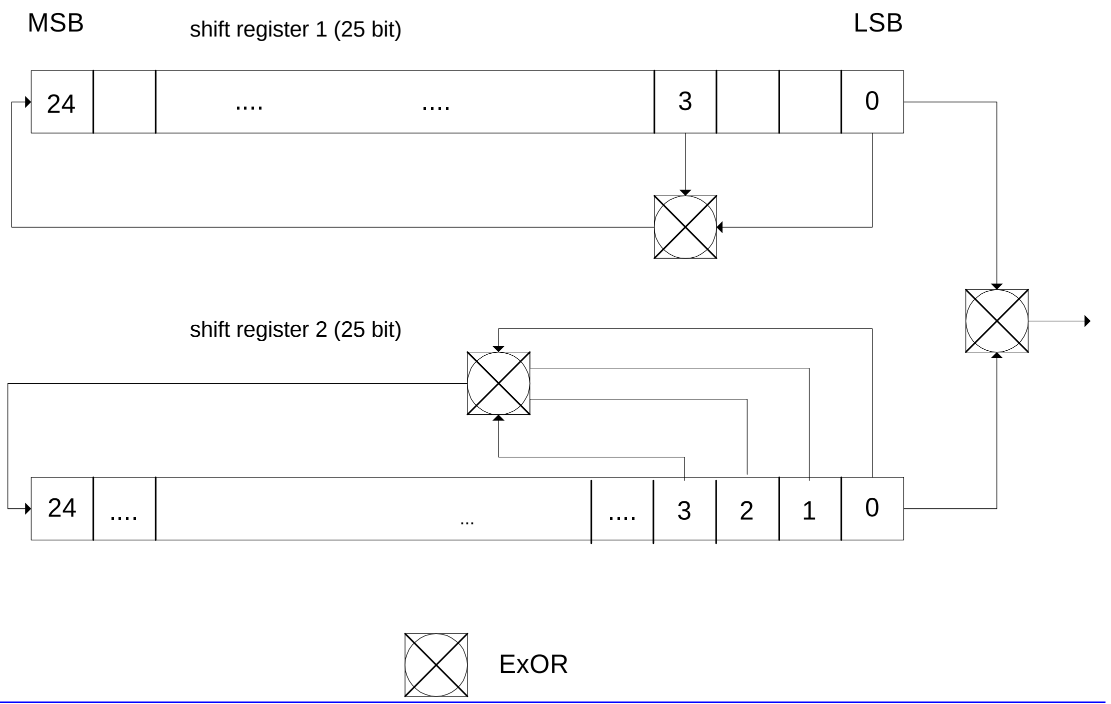


Figure 9. Preamble scrambling ~~spreading~~code generator

~~Note that the code words always start with a constant '0' symbol.~~

Before ~~modulation and~~ transmission these binary code words are converted to real valued sequences by the transformation '0'-> '+1', '1'-> '-1'.

Note: WG1 has accepted the 4096 chip long code scrambling~~spreading~~ as a working assumption.

4.3.3.2 Preamble signature

The preamble part consists of 256 repetitions of a length 16 signature,<P_0,P_1,,P_{15}>. Before scrambling the preamble is therefore

$$P_0, P_1, \cdots, P_{15}, P_0, P_1, \cdots, P_{15}, \cdots\cdots, P_0, P_1, \cdots, P_{15}$$

The signature is from the set of 16 Hadamard codes of length 16. These are listed in Table 5

~~The preamble part carries one of 16 different orthogonal complex signatures of length 16, <P_0, P_1, ..., P_{15}>. The signatures are based on a set of Orthogonal Gold codes of length 16 and are specified in Table 5.~~

~~Note: WG1 has accepted differential preambles additionally as a working assumption.~~

	~~Preamble symbols~~															
~~**Signature**~~	~~P_0~~	~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	~~P_6~~	~~P_7~~	~~P_8~~	~~P_9~~	~~P_{10}~~	~~P_{11}~~	~~P_{12}~~	~~P_{13}~~	~~P_{14}~~	~~P_{15}~~
~~**1**~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~A~~
~~**2**~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~A~~
~~**3**~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~A~~
~~**4**~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~A~~
~~**5**~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~
~~**6**~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~A~~
~~**7**~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~
~~**8**~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~A~~
~~**9**~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~A~~
~~**10**~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~A~~
~~**11**~~	~~A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~A~~
~~**12**~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~A~~
~~**13**~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~A~~
~~**14**~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~A~~
~~**15**~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~A~~
~~**16**~~	~~-A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~A~~	~~-A~~	~~-A~~	~~-A~~	~~-A~~	~~A~~	~~-A~~	~~A~~	~~A~~	~~-A~~	~~A~~

	Preamble symbols															
Signature	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
2	A	-A	A	-A	A	-A	A	-A	A	-A	A	-A	A	-A	A	-A
3	A	A	-A	-A	A	A	-A	-A	A	A	-A	-A	A	A	-A	-A
4	A	-A	-A	A	A	-A	-A	A	A	-A	-A	A	A	-A	-A	A
5	A	A	A	A	-A	-A	-A	-A	A	A	A	A	-A	-A	-A	-A
6	A	-A	A	-A	-A	A	-A	A	A	-A	A	-A	-A	A	-A	A
7	A	A	-A	-A	-A	-A	A	A	A	A	-A	-A	-A	-A	A	A
8	A	-A	-A	A	-A	A	A	-A	A	-A	-A	A	-A	A	A	-A
9	A	A	A	A	A	A	A	A	-A	-A	-A	-A	-A	-A	-A	-A
10	A	-A	A	-A	A	-A	A	-A	-A	A	-A	A	-A	A	-A	A
11	A	A	-A	-A	A	A	-A	-A	-A	-A	A	A	-A	-A	A	A
12	A	-A	-A	A	A	-A	-A	A	-A	A	A	-A	-A	A	A	-A
13	A	A	A	A	-A	-A	-A	-A	-A	-A	-A	-A	A	A	A	A
14	A	-A	A	-A	-A	A	-A	A	-A	A	-A	A	A	-A	A	-A
15	A	A	-A	-A	-A	-A	A	A	-A	-A	A	A	A	A	-A	-A
16	A	-A	-A	A	-A	A	A	-A	-A	A	A	-A	A	-A	-A	A

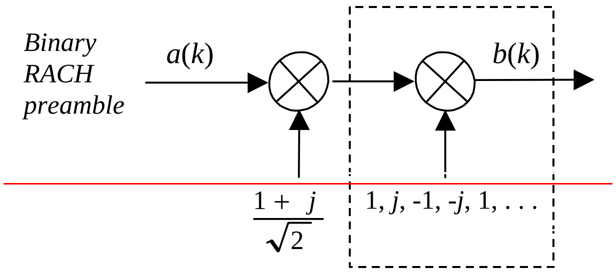
Table 5. Preamble signatures

The value of~~.~~ A = +1~~+j~~ in bipolar representation which is equivalent to 0 in boolean representation.

Note: the Hadamard signatures are a working assumption.

4.3.3.3 Preamble PAPR reduction

In order to reduce the PAPR during RACH preamble transmission the following technique is used.



Modulation for PAPR reduction

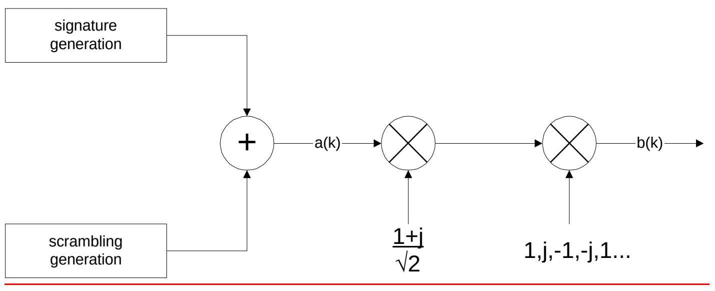


Figure 10 - Baseband modulator for RACH preamble.

The binary preamble $a(k)$ is modulated to get the complex valued preamble $b(k)$,

$$b(k) = a(k)\, e^{j(\frac{\pi}{4}+\frac{\pi}{2}k)}, \quad k = 0, 1, 2, 3, \ldots, 4095.$$

Note: this is a working assumption.

4.3.3.4~~3~~ Channelization codes for the message part

The signature in the preamble specifies one of the 16 nodes in the code-tree that corresponds to channelization codes of length 16, as shown in Figure 11. The sub-tree below the specified node is used for spreading of the message part. The control (Q-branch) is spread with the channelization code of spreading factor 256 in the lowest branch of the sub-tree. The data part (I-branch) can use any of the channelization codes from spreading factor 32 to 256 in the upper-most branch of the sub-tree. ~~However, the system may restrict the set of codes (spreading factors) actually allowed in the cell, through the use of a BCH message.~~

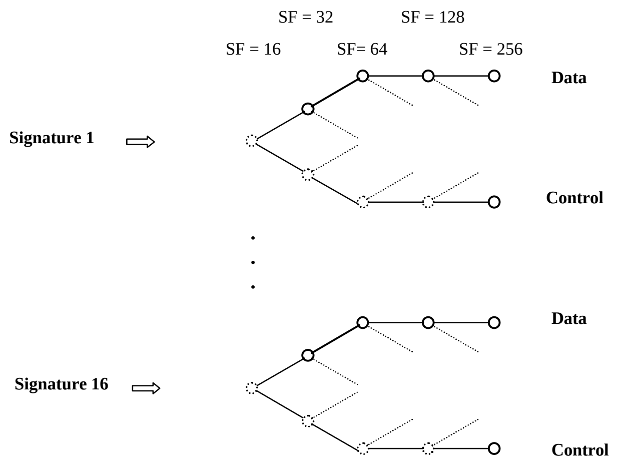


Figure 11. Channelization codes for the random access message part.

~~Since the control part is always spread with a known channelization code of length 256, it can be detected by the NodeB. The rate information field of the control part informs the base station about the spreading factor used on the data part. With knowledge of the sub-tree (obtained from the preamble signature) and the spreading factor (obtained from the rate information), the NodeB knows which channelization code is used for the data part.~~

~~<Editor's note: possibly the replacement term for BS should be cell.>~~

4.3.3.5~~4~~ Scrambling code for the message part

In addition to spreading, the message part is also subject to scrambling with a 10 ms complex code. The scrambling code is cell-specific and has a one-to-one correspondence to the spreading code used for the preamble part.

The scrambling codes used are from the same set of codes as is used for the other dedicated uplink channels when the long scrambling codes are used for these channels. The first 256 of the long scrambling codes are used for the random access channel. The phases 4096..42496 of the codes are used for the message part (phases 0..4095 of c_1 are used in preamble spreading) with the chip rate of 3.84 Mchips/s.

The generation of these codes is explained in Section 4.3.2.2. The mapping of these codes to provide a complex scrambling code is also the same as for the other dedicated uplink channels and is described in Section 4.3.2.

Note: the 4096 long code scrambling is a working assumption.

4.3.4 Common packet channel codes

<to be defined>

4.3.4.1 Access Preamble scrambling code

<to be defined>

4.3.4.2 CD preamble spreading code

<to be defined>

4.3.4.3 CPCH preamble signatures

<to be defined>

4.3.4.4 Channelization codes for the CD message part

<to be defined>

4.3.4.5 Scrambling code for the CD message part

<to be defined>

4.4 Modulation

4.4.1 Modulating chip rate

The modulating chip rate is 3.84~~4.096~~ Mcps. This basic chip rate can be extended to [0.96~~1.024~~,] 7.68~~8.192~~ or 15.36~~16.384~~ Mcps.

~~4.4.2 Pulse shaping~~

~~The pulse-shaping filters are root-raised cosine (RRC) with roll-off α=0.22 in the frequency domain.~~

~~<Editor's note: pulse shaping will be moved to appropriate WG4 documentation.>~~

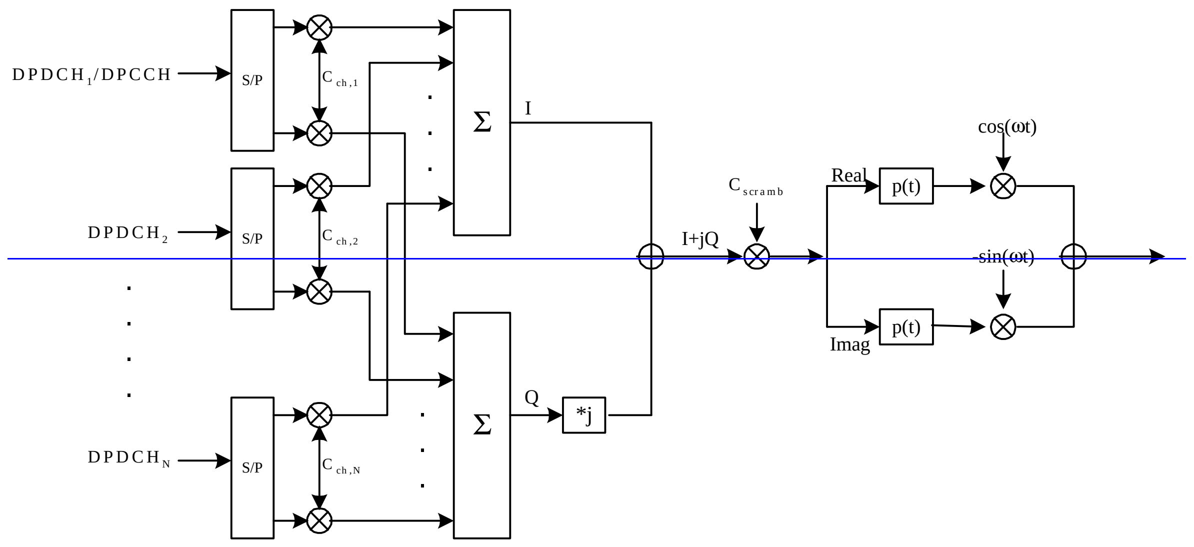
4.4.2~~3~~ Modulation

In the uplink, the modulation of both DPCCH and DPDCH is BPSK. The modulated DPCCH is mapped to the Q-branch, while the first DPDCH is mapped to the I-branch. Subsequently added DPDCHs are mapped alternatively to the I or Q-branches.

5 Downlink spreading and modulation

5.1 Spreading

Figure 12 illustrates the spreading and modulation for the downlink DPCH. Data modulation is QPSK where each pair of two bits are serial-to-parallel converted and mapped to the I and Q branch respectively. The I and Q branch are then spread to the chip rate with the same channelization code c_{ch} (real spreading) and subsequently scrambled by the scrambling code C_{scramb} (complex scrambling).



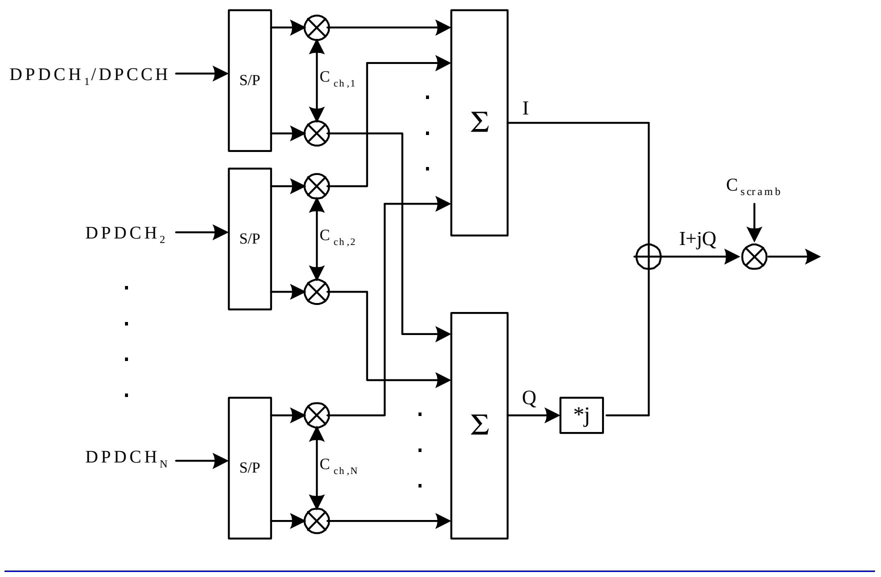


Figure 12. Spreading/modulation for downlink DPCH.

Spreading/modulation of the Secondary CCPCH, PSCCCH, PDSCH, PICH and AICH is done in an identical way as for the downlink DPCH.

Spreading/modulation of the Primary CCPCH is done in an identical way as for the downlink DPCH, except that the Primary CCPCH is time multiplexed after spreading.

~~a~~As illustrated in Figure 13. Primary SCH and Secondary SCH are code multiplexed and transmitted simultaneously during the 1st 256 chips of each slot. The transmission power of SCH can be adjusted by a gain factor $G_{P\text{-}SCH}$ and $G_{S\text{-}SCH}$, respectively, independent of transmission power of P-CCPCH. The SCH is *non-orthogonal* to the other downlink physical channels.

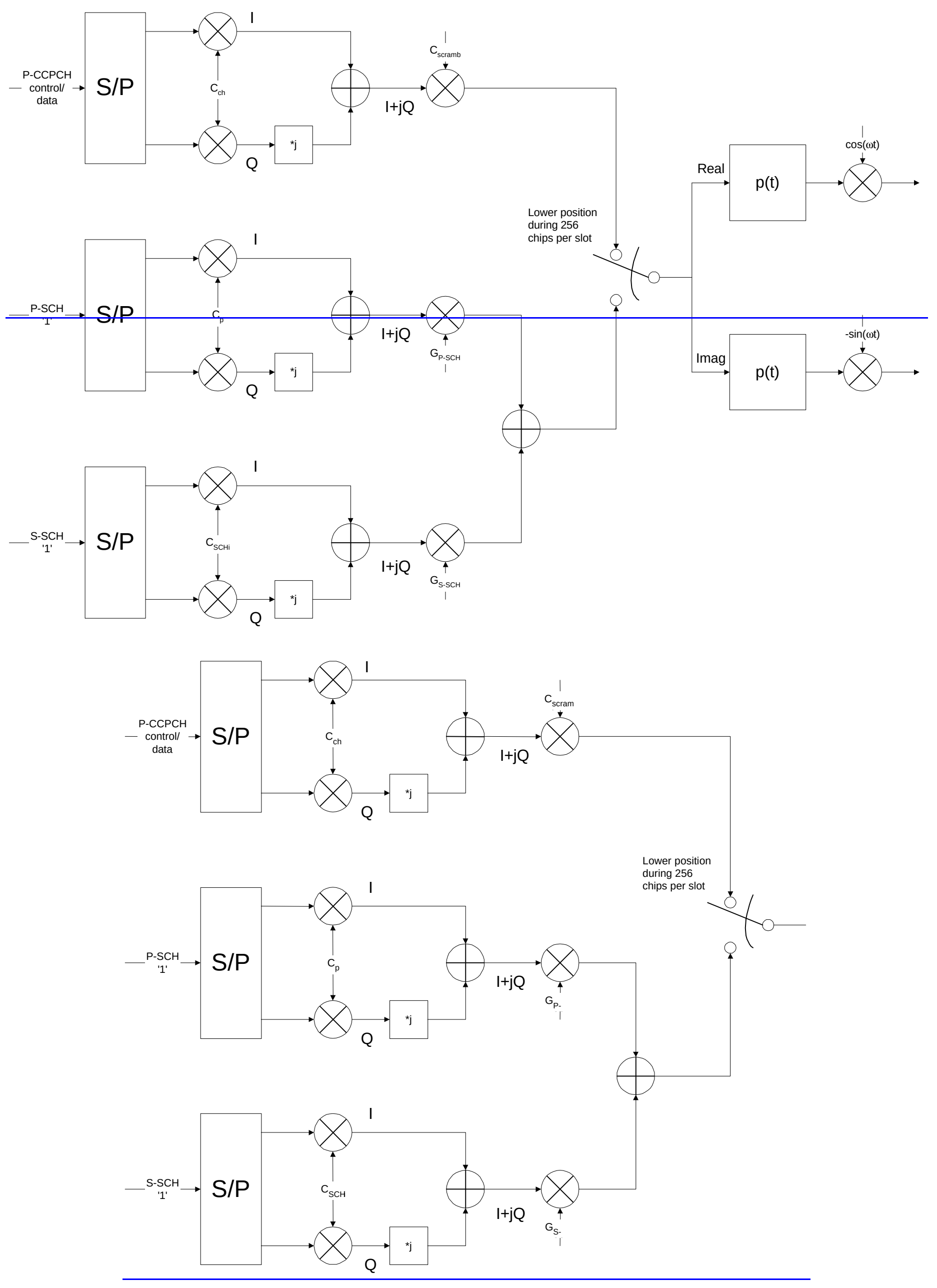


Figure 13. Spreading and modulation for SCH and P-CCPCH

5.2 Code generation and allocation

5.2.1 Channelization codes

The channelization codes of Figure 12 and Figure 13 are the same codes as used in the uplink, namely Orthogonal Variable Spreading Factor (OVSF) codes that preserve the orthogonality between downlink channels of different rates and spreading factors. The OVSF codes are defined in Figure 3 in Section 4.3.1. The same restriction on code allocation applies as for the uplink, but for a cell and not a UE as in the uplink. Hence, in the downlink, a specific combination of channelization code and scrambling code can be used in a cell if and only if no other channelization code on the path from the specific code to the root of the tree or in the sub-tree below the specific code is used in the same cell with the same scrambling code.

The channelization code for the BCH is a predefined code which is the same for all cells within the system.

The channelization code(s) used for the Secondary Common Control Physical Channel is broadcast on the BCH.

<Editors note: the above sentence may not be within the scope of this document.>

5.2.2 Scrambling code

There are a total 512*512 = 262,144 scrambling codes, numbered 0…262,143. The scrambling codes are divided into 512 sets each of a primary scrambling code and 511 secondary scrambling codes.

The primary scrambling codes consist of scrambling codes i=0…511. The i:th set of secondary scrambling codes consists

There is a one-to-one mapping between each primary scrambling code and 511 secondary scrambling codes in a set such that i:th primary scrambling code corresponds to i:th set of scrambling codes.

The set of primary scrambling codes is further divided into 32 scrambling code groups, each consisting of 16 primary scrambling codes. The j:th scrambling code group consists of scrambling codes j*16, …, j*16+15, where j=0, …, 31.

Each cell is allocated one and only one primary scrambling code. The primary CCPCH is always transmitted using the primary scrambling code. The other downlink physical channels can be transmitted with either the primary scrambling code or a secondary scrambling code from the set associated with the primary scrambling code of the cell.

<Editor's note: There may be a need to limit the actual number of codes used in each set of secondary scrambling codes, in order to limit the signalling requriements. >

<Editor's note: it is not standardised how many scrambling codes a UE must decode in parallel.>

The scrambling code sequences are constructed by combining two real sequences into a complex sequence. Each of the two real sequences are constructed as the position wise modulo 2 sum of [38400~~40960~~ chip segments of] two binary *m*-sequences generated by means of two generator polynomials of degree 18. The resulting sequences thus constitute segments of a set of Gold sequences. The scrambling codes are repeated for every 10 ms radio frame. Let *x* and *y* be the two sequences respectively. The *x* sequence is constructed using the primitive (over GF(2)) polynomial $1+X^7+X^{18}$. The y sequence is constructed using the polynomial $1+X^5+X^7+ X^{10}+X^{18}$.

<Editor's note: [ ] is due to the fact that only 3.84~~4.096~~Mcps is an agreement~~ working assumptions~~. 0.96~~1.024~~, 7.68~~8.192~~, and 15.36~~16.384~~Mcps are ffs.>

Let $n_{17} \ldots n_0$ be the binary representation of the scrambling code number *n* (decimal) with n_0 being the least significant bit. The *x* sequence depends on the chosen scrambling code number *n* and is denoted x_n, in the sequel. Furthermore, let $x_n(i)$ and $y(i)$ denote the *i*:th symbol of the sequence x_n and *y*, respectively

The *m*-sequences x_n and *y* are constructed as:

Initial conditions:

$x_n(0)=n_0 , x_n(1)= n_1 , \ldots =x_n(16)= n_{16} , x_n(17)= n_{17}$

$y(0)=y(1)= \ldots =y(16)= y(17)=1$

Recursive definition of subsequent symbols:

$x_n(i+18) =x_n(i+7) + x_n(i) \text{ modulo } 2, i=0,\ldots,2^{18}-20,$

$y(i+18) = y(i+10)+y(i+7)+y(i+5)+y(i) \text{ modulo } 2, i=0,\ldots, 2^{18}-20.$

The n:th Gold code sequence z_n is then defined as

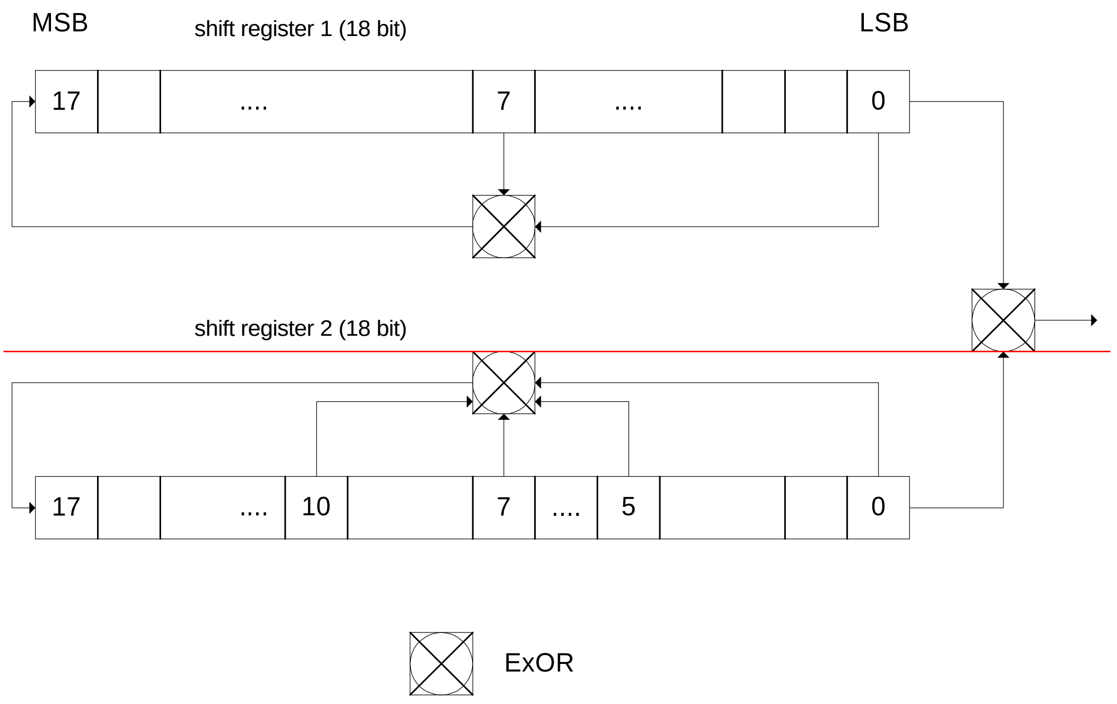
$z_n(i) = x_n(i) + y(i)$ *modulo 2*, $i=0,\ldots, 2^{18}-2$.

These binary code words are converted to real valued sequences by the transformation '0'-> '+1', '1'-> '-1'.

Finally, the n:th complex scrambling code sequence C_{scramb} is defined as (the lowest index corresponding to the chip scrambled first in each radio frame): (~~see Table 6 for definition of~~ where N is the period in chips and M is 131,072)

$C_{scramb}(i) = z\cancel{'}_n(i) + j\, z\cancel{'}_n(i+M)$, $i=0,1,\ldots,N-1$.
<Editor's note: the values 38400~~40960~~ is based on an assumption of a chip rate of 3.84~~4.096~~ Mcps.>

Note that the pattern from phase 0 up to the phase of 38399~~10 msec~~ is repeated.



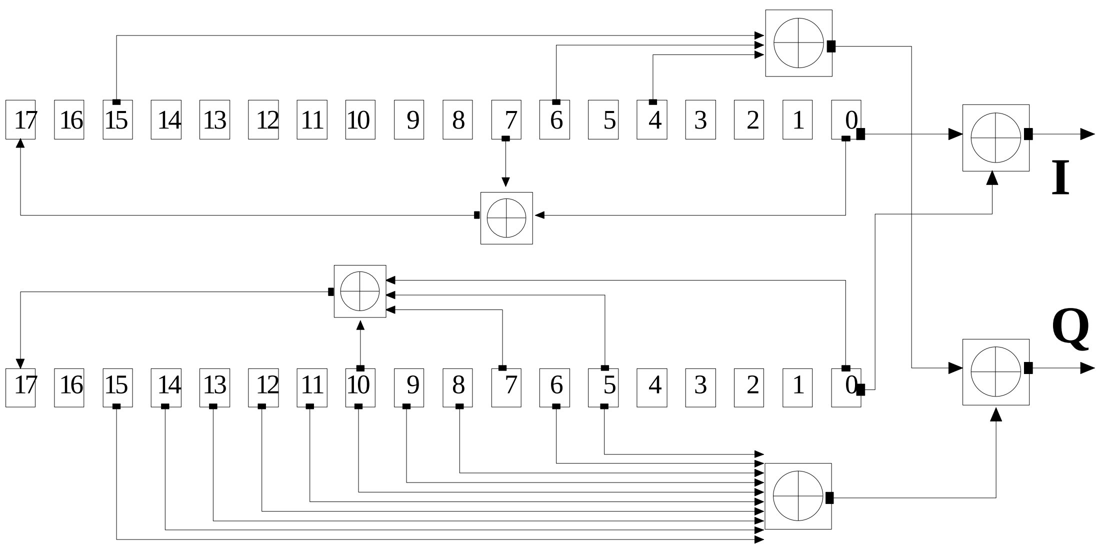


Figure 14. Configuration of downlink scrambling code generator

~~<Editor's note: a replacement figure for the above is to be prepared showing both I & Q generation.>~~

~~chip rate (Mcps)~~	~~Period~~ ~~N~~	~~IQ Offset~~ ~~M~~	~~Range of phase (chip)~~	
			~~for in-phase component~~	~~for quadrature component~~
~~[1.024]~~	~~[10240]~~	~~[131072]~~	~~0 N-1~~	~~M N+M-1~~
~~4.096~~	~~40960~~	~~131072~~		
~~[8.192]~~	~~[81920]~~	~~[131072]~~		
~~[16.384]~~	~~[163840]~~	~~[131072]~~		

~~**Table 6. Correspondence between chip rate and downlink scrambling code phase range**~~

5.2.3 Synchronisation codes

5.2.3.1 Code Generation

The Primary code sequence, C$_p$ is constructed as a so-called generalised hierarchical Golay sequence. The Primary SCH is furthermore chosen to have good aperiodic auto correlation properties.

Letting a = <x$_1$, x$_2$, x$_3$, , .x$_{16}$> = <0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0> and

~~b = <x$_1$, x$_2$, x$_3$, , .x$_8$, x$_1$, x$_2$, x$_3$, , .x$_8$ >.~~

$$b =< x_1, x_2, .., x_8, \bar{x}_9, \bar{x}_{10}, .., \bar{x}_{16} >$$

The PSC code is generated by repeating sequence a'modulated by a Golay complementary sequence.

Letting $y =< a, a, a, \bar{a}, \bar{a}, a, \bar{a}, \bar{a}, a, a, a, \bar{a}, a, \bar{a}, a, a >$

The definition of the PSC code word C$_p$ follows (the left most index corresponds to the chip transmitted first in each time slot):

C$_p$=<y(0),y(1),y(2),...,y(255)>.

Let the sequence $Z = \{b,b,b,\bar{b},b,b,\bar{b},\bar{b},b,\bar{b},b,\bar{b},\bar{b},\bar{b},\bar{b},\bar{b}\}$ ~~z = <b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b>~~. Then the Secondary Synchronization code words, {C$_1$,,C$_{16}$~~$_7$~~} are constructed as the position wise addition modulo 2 of a Hadamard sequence and the sequence *z*.
The Hadamard sequences are obtained as the rows in a matrix H_8 constructed recursively by:

$$H_0 = (0)$$

$$H_k = \begin{pmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & \overline{H}_{k-1} \end{pmatrix}, \quad k \geq 1$$

The rows are numbered from the top starting with row *0* (the all zeros sequence).

The Hadamard sequence *h* depends on the chosen code number *n* and is denoted h_n in the sequel.

This code word is chosen from every 16~~8~~th row of the matrix H_8.*implying* ~~Therefore, there are~~ 16~~32~~ possible code words given by n =0,16,32,48,64,80,96,112,128,144,160,176,192,208,224,240~~out of which *n* = 1, 2, .., 17 are used~~.

Furthermore, let $h_n(i)$ and $z(i)$ denote the *i*:th symbol of the sequence h_n and *z*, respectively.

~~Then h_n is equal to the row of H_8 numbered by the bit reverse of the 8 bit binary representation of *n*.~~

~~<Editor's note: the above line should be checked for correctness and removed if necessary.>~~

The definition of the *n*:th SCH code word follows (the left most index correspond to the chip transmitted first in each slot):

C$_{SCH,n}$ = < $h_n(0)+z(0)$, $h_n(1)+z(1)$, $h_n(2)+z(2)$, …,$h_n(255)+z(255)$ >, All sums of symbols are taken modulo 2.

These PSC and SSC binary code words are converted to real valued sequences by the transformation ‘0’-> ‘+1’, ‘1’-> ‘-1’.

The Secondary SCH code words are defined in terms of $C_{SCH,n}$ and the definition of {C_1,,C_{16}~~$_7$~~} now follows as:

$C_i = C_{SCH,i}$, i=1,,16~~7~~

5.2.3.2 Code Allocation

The 32 sequences are constructed such that their cyclic-shifts are unique, i.e., a non-zero cyclic shift less than 15~~6~~ of any of the 32 sequences is not equivalent to some cyclic shift of any other of the 32 sequences. Also, a non-zero cyclic shift less than 15~~6~~ of any of the sequences is not equivalent to itself with any other cyclic shift less than 15~~6~~. The following sequences are used to encode the 32 different scrambling code groups (note that c_i indicates the *i*th Secondary Short code of the 16~~7~~ codes). Note that a Secondary Short code can be different from one time slot to another and that the sequence pattern can be different from one cell to another, depending on Scrambling Code Group the cell uses

~~Scrambling Code Groups~~	~~Slot Number~~															
	~~#1~~	~~#2~~	~~#3~~	~~#4~~	~~#5~~	~~#6~~	~~#7~~	~~#8~~	~~#9~~	~~#10~~	~~#11~~	~~#12~~	~~#13~~	~~#14~~	~~#15~~	~~#16~~
~~Group1~~	~~C_{1}~~	~~C_{1}~~	~~C_{2}~~	~~C_{11}~~	~~C_{6}~~	~~C_{3}~~	~~C_{15}~~	~~C_{7}~~	~~C_{8}~~	~~C_{8}~~	~~C_{7}~~	~~C_{15}~~	~~C_{3}~~	~~C_{6}~~	~~C_{11}~~	~~C_{2}~~
~~Group2~~	~~C_{1}~~	~~C_{2}~~	~~C_{9}~~	~~C_{3}~~	~~C_{10}~~	~~C_{11}~~	~~C_{13}~~	~~C_{13}~~	~~C_{11}~~	~~C_{10}~~	~~C_{3}~~	~~C_{9}~~	~~C_{2}~~	~~C_{1}~~	~~C_{16}~~	~~C_{16}~~
~~Group 3~~	~~C_{1}~~	~~C_{3}~~	~~C_{16}~~	~~C_{12}~~	~~C_{14}~~	~~C_{2}~~	~~C_{11}~~	~~C_{2}~~	~~C_{14}~~	~~C_{12}~~	~~C_{16}~~	~~C_{3}~~	~~C_{1}~~	~~C_{13}~~	~~C_{4}~~	~~C_{13}~~
~~Group 4~~	~~C_{1}~~	~~C_{4}~~	~~C_{6}~~	~~C_{4}~~	~~C_{1}~~	~~C_{10}~~	~~C_{9}~~	~~C_{8}~~	~~C_{17}~~	~~C_{14}~~	~~C_{12}~~	~~C_{14}~~	~~C_{17}~~	~~C_{8}~~	~~C_{9}~~	~~C_{10}~~
~~Group 5~~	~~C_{1}~~	~~C_{5}~~	~~C_{13}~~	~~C_{13}~~	~~C_{5}~~	~~C_{1}~~	~~C_{7}~~	~~C_{14}~~	~~C_{3}~~	~~C_{16}~~	~~C_{8}~~	~~C_{8}~~	~~C_{16}~~	~~C_{3}~~	~~C_{14}~~	~~C_{7}~~
~~Group 6~~	~~C_{1}~~	~~C_{6}~~	~~C_{3}~~	~~C_{5}~~	~~C_{9}~~	~~C_{9}~~	~~C_{5}~~	~~C_{3}~~	~~C_{6}~~	~~C_{1}~~	~~C_{4}~~	~~C_{2}~~	~~C_{15}~~	~~C_{15}~~	~~C_{2}~~	~~C_{4}~~
~~Group 7~~	~~C_{1}~~	~~C_{7}~~	~~C_{10}~~	~~C_{14}~~	~~C_{13}~~	~~C_{17}~~	~~C_{3}~~	~~C_{9}~~	~~C_{9}~~	~~C_{3}~~	~~C_{17}~~	~~C_{13}~~	~~C_{14}~~	~~C_{10}~~	~~C_{7}~~	~~C_{1}~~
~~Group 8~~	~~C_{1}~~	~~C_{8}~~	~~C_{17}~~	~~C_{6}~~	~~C_{17}~~	~~C_{8}~~	~~C_{1}~~	~~C_{15}~~	~~C_{12}~~	~~C_{5}~~	~~C_{1}~~	~~C_{7}~~	~~C_{13}~~	~~C_{5}~~	~~C_{12}~~	~~C_{15}~~
~~Group 9~~	~~C_{1}~~	~~C_{9}~~	~~C_{7}~~	~~C_{15}~~	~~C_{4}~~	~~C_{16}~~	~~C_{16}~~	~~C_{4}~~	~~C_{15}~~	~~C_{7}~~	~~C_{9}~~	~~C_{1}~~	~~C_{12}~~	~~C_{17}~~	~~C_{17}~~	~~C_{12}~~
~~Group 10~~	~~C_{1}~~	~~C_{10}~~	~~C_{14}~~	~~C_{7}~~	~~C_{8}~~	~~C_{7}~~	~~C_{14}~~	~~C_{10}~~	~~C_{1}~~	~~C_{9}~~	~~C_{5}~~	~~C_{12}~~	~~C_{11}~~	~~C_{12}~~	~~C_{5}~~	~~C_{9}~~
~~Group 11~~	~~C_{1}~~	~~C_{11}~~	~~C_{4}~~	~~C_{16}~~	~~C_{12}~~	~~C_{15}~~	~~C_{12}~~	~~C_{16}~~	~~C_{4}~~	~~C_{11}~~	~~C_{1}~~	~~C_{6}~~	~~C_{10}~~	~~C_{7}~~	~~C_{10}~~	~~C_{6}~~
~~Group 12~~	~~C_{1}~~	~~C_{12}~~	~~C_{11}~~	~~C_{8}~~	~~C_{16}~~	~~C_{6}~~	~~C_{10}~~	~~C_{5}~~	~~C_{7}~~	~~C_{13}~~	~~C_{14}~~	~~C_{17}~~	~~C_{9}~~	~~C_{2}~~	~~C_{15}~~	~~C_{3}~~
~~Group 13~~	~~C_{1}~~	~~C_{13}~~	~~C_{1}~~	~~C_{17}~~	~~C_{3}~~	~~C_{14}~~	~~C_{8}~~	~~C_{11}~~	~~C_{10}~~	~~C_{15}~~	~~C_{10}~~	~~C_{11}~~	~~C_{8}~~	~~C_{14}~~	~~C_{3}~~	~~C_{17}~~
~~Group 14~~	~~C_{1}~~	~~C_{14}~~	~~C_{8}~~	~~C_{9}~~	~~C_{7}~~	~~C_{5}~~	~~C_{6}~~	~~C_{17}~~	~~C_{13}~~	~~C_{17}~~	~~C_{6}~~	~~C_{5}~~	~~C_{7}~~	~~C_{9}~~	~~C_{8}~~	~~C_{14}~~
~~Group 15~~	~~C_{1}~~	~~C_{15}~~	~~C_{15}~~	~~C_{1}~~	~~C_{11}~~	~~C_{13}~~	~~C_{4}~~	~~C_{6}~~	~~C_{16}~~	~~C_{2}~~	~~C_{2}~~	~~C_{16}~~	~~C_{6}~~	~~C_{4}~~	~~C_{13}~~	~~C_{11}~~
~~Group 16~~	~~C_{1}~~	~~C_{16}~~	~~C_{5}~~	~~C_{10}~~	~~C_{15}~~	~~C_{4}~~	~~C_{2}~~	~~C_{12}~~	~~C_{2}~~	~~C_{4}~~	~~C_{15}~~	~~C_{10}~~	~~C_{5}~~	~~C_{16}~~	~~C_{1}~~	~~C_{8}~~
~~Group 17~~	~~C_{1}~~	~~C_{17}~~	~~C_{12}~~	~~C_{2}~~	~~C_{2}~~	~~C_{12}~~	~~C_{17}~~	~~C_{1}~~	~~C_{5}~~	~~C_{6}~~	~~C_{11}~~	~~C_{4}~~	~~C_{4}~~	~~C_{11}~~	~~C_{6}~~	~~C_{5}~~
~~Group 18~~	~~C_{2}~~	~~C_{8}~~	~~C_{11}~~	~~C_{15}~~	~~C_{14}~~	~~C_{1}~~	~~C_{4}~~	~~C_{10}~~	~~C_{10}~~	~~C_{4}~~	~~C_{1}~~	~~C_{14}~~	~~C_{15}~~	~~C_{11}~~	~~C_{8}~~	~~C_{2}~~
~~Group 19~~	~~C_{2}~~	~~C_{9}~~	~~C_{1}~~	~~C_{7}~~	~~C_{1}~~	~~C_{9}~~	~~C_{2}~~	~~C_{16}~~	~~C_{13}~~	~~C_{6}~~	~~C_{14}~~	~~C8~~	~~C_{14}~~	~~C_{6}~~	~~C_{13}~~	~~C_{16}~~
~~Group 20~~	~~C_{2}~~	~~C_{10}~~	~~C_{8}~~	~~C_{16}~~	~~C_{5}~~	~~C_{17}~~	~~C_{17}~~	~~C_{5}~~	~~C_{16}~~	~~C_{8}~~	~~C_{10}~~	~~C_{2}~~	~~C_{13}~~	~~C_{1}~~	~~C_{1}~~	~~C_{13}~~
~~Group 21~~	~~C_{2}~~	~~C_{11}~~	~~C_{15}~~	~~C_{8}~~	~~C_{9}~~	~~C_{8}~~	~~C_{15}~~	~~C_{11}~~	~~C_{2}~~	~~C_{10}~~	~~C_{6}~~	~~C_{13}~~	~~C_{12}~~	~~C_{13}~~	~~C_{6}~~	~~C_{10}~~
~~Group 22~~	~~C_{2}~~	~~C_{12}~~	~~C_{5}~~	~~C_{17}~~	~~C_{13}~~	~~C_{16}~~	~~C_{13}~~	~~C_{17}~~	~~C_{5}~~	~~C_{12}~~	~~C_{2}~~	~~C_{7}~~	~~C_{11}~~	~~C_{8}~~	~~C_{11}~~	~~C_{7}~~
~~Group 23~~	~~C_{2}~~	~~C_{13}~~	~~C_{12}~~	~~C_{9}~~	~~C_{17}~~	~~C_{7}~~	~~C_{11}~~	~~C_{6}~~	~~C_{8}~~	~~C_{14}~~	~~C_{15}~~	~~C_{1}~~	~~C_{10}~~	~~C_{3}~~	~~C_{16}~~	~~C_{4}~~
~~Group 24~~	~~C_{2}~~	~~C_{14}~~	~~C_{2}~~	~~C_{1}~~	~~C_{4}~~	~~C_{15}~~	~~C_{9}~~	~~C_{12}~~	~~C_{11}~~	~~C_{16}~~	~~C_{11}~~	~~C_{12}~~	~~C_{9}~~	~~C_{15}~~	~~C_{4}~~	~~C_{1}~~
~~Group 25~~	~~C_{2}~~	~~C_{15}~~	~~C_{9}~~	~~C_{10}~~	~~C_{8}~~	~~C_{6}~~	~~C_{7}~~	~~C_{1}~~	~~C_{14}~~	~~C_{1}~~	~~C_{7}~~	~~C_{6}~~	~~C_{8}~~	~~C_{10}~~	~~C_{9}~~	~~C_{15}~~
~~Group 26~~	~~C_{2}~~	~~C_{16}~~	~~C_{16}~~	~~C_{2}~~	~~C_{12}~~	~~C_{14}~~	~~C_{5}~~	~~C_{7}~~	~~C_{17}~~	~~C_{3}~~	~~C_{3}~~	~~C_{17}~~	~~C_{7}~~	~~C_{5}~~	~~C_{14}~~	~~C_{12}~~
~~Group 27~~	~~C_{2}~~	~~C_{17}~~	~~C_{6}~~	~~C_{11}~~	~~C_{16}~~	~~C_{5}~~	~~C_{3}~~	~~C_{13}~~	~~C_{3}~~	~~C_{5}~~	~~C_{16}~~	~~C_{11}~~	~~C_{6}~~	~~C_{17}~~	~~C_{2}~~	~~C_{9}~~
~~Group 28~~	~~C_{2}~~	~~C_{1}~~	~~C_{13}~~	~~C_{3}~~	~~C_{3}~~	~~C_{13}~~	~~C_{1}~~	~~C_{2}~~	~~C_{6}~~	~~C_{7}~~	~~C_{12}~~	~~C_{5}~~	~~C_{5}~~	~~C_{12}~~	~~C_{7}~~	~~C_{6}~~

~~Group 29~~	~~C_2~~	~~C_2~~	~~C_3~~	~~C_{12}~~	~~C_7~~	~~C_4~~	~~C_{16}~~	~~C_8~~	~~C_9~~	~~C_9~~	~~C8~~	~~C_{16}~~	~~C_4~~	~~C_7~~	~~C_{12}~~	~~C_3~~
~~Group 30~~	~~C_2~~	~~C_3~~	~~C_{10}~~	~~C_4~~	~~C_{11}~~	~~C_{12}~~	~~C_{14}~~	~~C_{14}~~	~~C_{12}~~	~~C_{11}~~	~~C_4~~	~~C_{10}~~	~~C_3~~	~~C_2~~	~~C_{17}~~	~~C_{17}~~
~~Group 31~~	~~C_2~~	~~C_4~~	~~C_{17}~~	~~C_{13}~~	~~C_{15}~~	~~C_3~~	~~C_{12}~~	~~C_3~~	~~C_{15}~~	~~C_{13}~~	~~C_{17}~~	~~C_4~~	~~C_2~~	~~C_{14}~~	~~C_5~~	~~C_{14}~~
~~Group 32~~	~~C_2~~	~~C_5~~	~~C_7~~	~~C_5~~	~~C_2~~	~~C_{11}~~	~~C_{10}~~	~~C_9~~	~~C_1~~	~~C_{15}~~	~~C_{13}~~	~~C_{15}~~	~~C_1~~	~~C_9~~	~~C_{10}~~	~~C_{11}~~
~~[SyncBTS]~~	~~C_2~~	~~C_6~~	~~C_{14}~~	~~C_{14}~~	~~C_6~~	~~C_2~~	~~C_8~~	~~C_{15}~~	~~C_4~~	~~C_{17}~~	~~C_9~~	~~C_9~~	~~C_{17}~~	~~C_4~~	~~C_{15}~~	~~C_8~~

Scrambling Code Groups	Slot Number														
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
Group1	C_1	C_1	C_2	C_8	C_9	C_{10}	C_{15}	C_8	C_{10}	C_{16}	C_2	C_7	C_{15}	C_7	C_{16}
Group2	C_1	C_2	C_5	C_2	C_3	C_7	C_7	C_1	C_8	C_4	C_6	C_5	C_8	C_6	C_3
Group 3	C_1	C_3	C_{12}	C_{12}	C_{16}	C_5	C_{14}	C_{10}	C_7	C_5	C_{10}	C_3	C_{16}	C_1	C_7
Group 4	C_1	C_5	C_7	C_{13}	C_7	C_1	C_9	C_9	C_5	C_{11}	C_3	C_{15}	C_{13}	C_{11}	C_{15}
Group 5	**C_1**	**C_9**	**C_{16}**	**C_3**	**C_8**	**C_9**	**C_3**	**C_{11}**	**C_1**	**C_6**	**C_8**	**C_6**	**C_{11}**	**C_{14}**	**C_{14}**
Group 6	C_1	C_4	C_{15}	C_{14}	C_6	C_{12}	C_6	C_{15}	C_9	C_9	C_{14}	C_1	C_7	C_4	C_{12}
Group 7	C_1	C_7	C_{13}	C_1	C_2	C_{14}	C_{12}	C_7	C_{12}	C_2	C_{11}	C_{11}	C_{14}	C_{13}	C_8
Group 8	C_1	C_{13}	C_9	C_{10}	C_{10}	C_2	C_5	C_6	C_{14}	C_1	C_5	C_{14}	C_9	C_2	C_{13}
Group 9	C_1	C_{12}	C_1	C_9	C_{11}	C_{11}	C_{10}	C_4	C_2	C_3	C_{12}	C_4	C_3	C_9	C_{10}
Group 10	C_1	C_6	C_4	C_{11}	C_{13}	C_{16}	C_1	C_{16}	C_{11}	C_7	C_7	C_{13}	C_6	C_{10}	C_4
Group 11	C_1	C_{11}	C_6	C_{15}	C_1	C_6	C_2	C_5	C_{16}	C_{15}	C_{16}	C_2	C_{12}	C_{12}	C_5
Group 12	C_1	C_8	C_{10}	C_7	C_{12}	C_3	C_4	C_2	C_6	C_{14}	C_{15}	C_9	C_5	C_{16}	C_{11}
Group 13	C_1	C_{15}	C_3	C_6	C_{15}	C_{13}	C_8	C_{12}	C_3	C_{12}	C_{13}	C_{10}	C_{10}	C_8	C_6
Group 14	C_1	C_{16}	C_8	C_4	C_5	C_4	C_{16}	C_{13}	C_{13}	C_8	C_9	C_{12}	C_1	C_5	C_9
Group 15	C_1	C_{14}	C_{14}	C_{16}	C_4	C_{15}	C_{13}	C_3	C_4	C_{13}	C_1	C_{16}	C_2	C_3	C_2
Group 16	C_1	C_{10}	C_{11}	C_5	C_{14}	C_8	C_{11}	C_{14}	C_{15}	C_{10}	C_4	C_8	C_4	C_{15}	C_1
Group 17	C_2	C_6	C_8	C_{14}	C_8	C_2	C_{10}	C_{10}	C_6	C_{12}	C_4	C_{16}	C_{14}	C_{12}	C_{16}
Group 18	C_2	C_5	C_3	C_{12}	C_{14}	C_{15}	C_2	C_{15}	C_{12}	C_8	C_8	C_{14}	C_5	C_9	C_3
Group 19	C_2	C_8	C_{14}	C_2	C_1	C_{13}	C_{11}	C_8	C_{11}	C_1	C_{12}	C_{12}	C_{13}	C_{14}	C_7
Group 20	C_2	C_2	C_1	C_7	C_{10}	C_9	C_{16}	C_7	C_9	C_{15}	C_1	C_8	C_{16}	C_8	C_{15}
Group 21	C_2	C_{14}	C_{10}	C_9	C_9	C_1	C_6	C_5	C_{13}	C_2	C_6	C_{13}	C_{10}	C_1	C_{14}
Group 22	C_2	C_7	C_9	C_8	C_{11}	C_4	C_3	C_1	C_5	C_{13}	C_{16}	C_{10}	C_6	C_{15}	C_{12}
Group 23	C_2	C_4	C_{11}	C_{11}	C_{15}	C_6	C_{13}	C_9	C_8	C_6	C_9	C_4	C_{15}	C_2	C_8
Group 24	C_2	C_{10}	C_{15}	C_4	C_7	C_{10}	C_4	C_{12}	C_2	C_5	C_7	C_5	C_{12}	C_{13}	C_{13}
Group 25	C_2	C_{15}	C_7	C_3	C_6	C_3	C_{15}	C_{14}	C_{14}	C_7	C_{10}	C_{11}	C_2	C_6	C_{10}
Group 26	C_2	C_1	C_6	C_1	C_4	C_8	C_8	C_2	C_7	C_3	C_5	C_6	C_7	C_5	C_4
Group 27	C_2	C_{16}	C_4	C_5	C_{16}	C_{14}	C_7	C_{11}	C_4	C_{11}	C_{14}	C_9	C_9	C_7	C_5
Group 28	C_2	C_3	C_{16}	C_{13}	C_5	C_{11}	C_5	C_{16}	C_{10}	C_{10}	C_{13}	C_2	C_8	C_3	C_{11}
Group 29	C_2	C_{12}	C_5	C_{16}	C_2	C_5	C_1	C_6	C_{15}	C_{16}	C_{15}	C_1	C_{11}	C_{11}	C_6
Group 30	C_2	C_{11}	C_2	C_{10}	C_{12}	C_{12}	C_9	C_3	C_1	C_4	C_{11}	C_3	C_4	C_{10}	C_9
Group 31	C_2	C_9	C_{12}	C_6	C_{13}	C_7	C_{12}	C_{13}	C_{16}	C_9	C_3	C_7	C_3	C_{16}	C_2
Group 32	C_2	C_{13}	C_{13}	C_{15}	C_3	C_{16}	C_{14}	C_4	C_3	C_{14}	C_2	C_{15}	C_1	C_4	C_1
[SyncBTS]	C_3	C_{11}	C_{14}	C_1	C_6	C_{11}	C_1	C_9	C_3	C_8	C_6	C_8	C_9	C_{16}	C_{16}

Table 9 Spreading Code allocation for Secondary SCH Code

5.3 Modulation

5.3.1 Modulating chip rate

The modulating chip rate is 3.84~~4.096~~ Mcps. This basic chip rate can be extended to [0.96~~1.024~~,] 7.68~~8.192~~ or 15.36~~16.384~~ Mcps.

~~5.3.2 Pulse shaping~~

~~The pulse-shaping filters are root raised cosine (RRC) with roll-off α=0.22 in the frequency domain.~~

~~<Editor's note: pulse shaping will be moved to appropriate WG4 documentation.>~~

5.3.2~~3~~ Modulation

QPSK modulation is used.

Annex A Generalised Hierarchical Golay Sequences

A.1 Alternative generation

The generalised hierarchical Golay sequences for the PSC described in 5.2.3.1 may be also viewed as generated (in real valued representation) by the following methods:

Method 1.

The sequence y is constructed from two constituent sequences x_1 and x_2 of length n_1 and n_2 respectively using the following formula:

$y(i) = x_2(i \bmod n_2) * x_1(i \text{ div } n_2),\ i = 0 \ldots (n_1 * n_2) - 1$

The constituent sequences x_1 and x_2 are chosen to be the following length 16 (i.e. $n_1 = n_2 = 16$) sequences:

- x_1 is defined to be the length 16 ($N^{(1)}=4$) Golay complementary sequence obtained by the delay matrix $D^{(1)} = [8, 4, 1,2]$ and weight matrix $W^{(1)} = [1, -1, 1,1]$.

- x_2 is a generalised hierarchical sequence using the following formula, selecting s=2 and using the two Golay complementary sequences x_3 and x_4 as constituent sequences. The length of the sequence x_3 and x_4 is called n_3 respectively n_4.

  $x_2(i) = x_4(i \bmod s + s*(i \text{ div } sn_3)) * x_3((i \text{ div } s) \bmod n_3),\ i = 0 \ldots (n_3 * n_4) - 1$

  x_3 and x_4 are defined to be identical and the length 4 ($N^{(3)}= N^{(4)}=2$) Golay complementary sequence obtained by the delay matrix $D^{(3)} = D^{(4)} = [1, 2]$ and weight matrix $W^{(3)} = W^{(4)} = [1, 1]$.

The Golay complementary sequences x_1,x_3 and x_4 are defined using the following recursive relation:

$a_0(k) = \delta(k)$ and $b_0(k) = \delta(k)$

$a_n(k) = a_{n-1}(k) + W^{(j)}{}_n \cdot b_{n-1}(k-D^{(j)}{}_n)$,

$b_n(k) = a_{n-1}(k) - W^{(j)}{}_n \cdot b_{n-1}(k-D^{(j)}{}_n)$,

$k = 0, 1, 2, , 2^{**}N^{(j)} -1,$

$n = 1, 2, , N^{(j)}.$

The wanted Golay complementary sequence x_j is defined by a_n assuming $n=N^{(j)}$. The Kronecker delta function is described by δ, k,j and n are integers.

Method 2

The sequence y can be viewed as a pruned Golay ~~code~~complementary sequence and generated using the following parameters which apply to the generator equations for a and b above:

(a) Let $j = 0, N^{(0)} = 8$

(b) $[D_1^0, D_2^0, D_3^0, D_4^0, D_5^0, D_6^0, D_7^0, D_8^0] = [128, 64, 16, 32, 8, 1, 4, 2]$

(c) $[W_1^0, W_2^0, W_3^0, W_4^0, W_5^0, W_6^0, W_7^0, W_8^0] = [1, -1, 1, 1, 1, 1, 1, 1]$

(d) For $n = 4, 6$, set $b_4(k) = a_4(k)$, $b_6(k) = a_6(k)$.

6 History

Document history		
draft	1999-02-12	New document merged from ETSI XX.05 and ARIB 3.2.4 sources.
0.0.1	1999-02-12	Corrected typo in table2.
0.0.2	1999-02-16	Added sec. SCH code table, option for HPSK on S(2) codes, scale on SCH.
0.0.3	1999-02-18	Reflected decision made on SCH multiplexing (see document titled Report from Ad Hoc #2 SCH multiplexing:) and additional description on the use of S(2) for uplink short scrambling code.
0.1.0	1999-02-28	Raised to 0.1.0 after TSG RAN WG1#2 meeting (Yokohama).
1.0.0	1999-03-12	Raised to 1.0.0 when presented to TSG RAN.
1.0.1	1999-03-17	Raised to 1.0.1 incorporated Ad Hoc changes and errata from e-mail.
1.0.2	1999-03-23	Raised to 1.0.2 incorporated reports from Ad Hocs plus editorial matters.
1.0.3	1999-03-24	Raised to 1.0.3 incorporated actions from WG1#3 plenary..
1.1.0	1999-03-26	Raised to 1.1.0 changed as result of text proposal, Tdoc 298.
1.1.1	1999-04-12	Raised to 1.1.1 by incorporating 3GPP template and adding editors note.
1.1.2	1999-04-12	Raised to 1.1.2 by entering editorial changes with revision marks.
1.1.3	1999-04-19	Rasied to 1.1.3 by Tdocs 347, 385 at WG1#4 meeting (Yokohama)
1.1.4	1999-04-20	Raised to 1.1.4 by Tdoc 397 at WG1#4 meeting (Yokohama)
2.0.0	1999-04-20	Raised to 2.0.0 at WG1#4 (Yokohama) for presentation to RSG RAN.
2.0.1	1999-04-27	Raised to 2.0.1 fixing references in section 4.3.2.3, fixed figures 10, 11.
2.0.2	1999-06-04	Raised to 2.0.2 at WG1#5 (Cheju) plenary.
2.1.0	1999-06-04	Raised to 2.1.0 at WG1#5 plenary for presentation to TSG RAN.
2.1.1	1999-06-22	Raised to 2.1.1 due to editorial changes noted after WG1#5.
2.1.2	1999-07-20	Raised to 2.1.2 due to editorial changes noted offline and proposals at WG1#6.

Editor for 25.213, spreading and modulation specification, is:

Peter Chambers
Siemens Roke Manor Research
Email: peter.chambers@roke.co.uk

This document is written in Microsoft Word 97.

ISBN <G-EEEE-NNNN-K>
Dépôt légal : <Mois AAAA>