IrDA Serial Infrared Physical Layer Link Specification for 16 Mb/s Addition (VFIR)

Errata To IrPHY Version 1.3

January 8, 1999

Authors:

This document is the result of collaboration between people of the *Hewlett-Packard Company*, *IBM Corporation*, and *Sharp Corporation*. Each of the authors listed below represents his company in the role of an editing author.

Walter Hirt
IBM Research Division, Zurich Research Laboratory, Rüschlikon, Switzerland

John Petrilla
Hewlett-Packard Company, Wireless Semiconductor Division, San Jose, CA, USA

Youichi Yuuki
Sharp Corporation, Optoelectronic Devices Division, Electronic Components Group, Nara, Japan

Contributors:

The following people contributed to the development and formulation of the concepts underlying this proposal and/or contributed to establish and demonstrate their technical and practical viability.

Ray Chock
(Calibre Inc. – Convenor, IrDA High-Speed Working Group)

Martin Beale, Dick Crawford, Takashi Hidai, John Petrilla, and *Tony Sorce*
(Hewlett-Packard Company)

Lee Hartley, *Martin Hassner*, *Nyles Heise*, *Walter Hirt*, *Dan McKay*, and *Beat Weiss*
(IBM Corporation)

Hitoshi Azuma, *Michita Katayama*, *Ikuo Nakaue*, *Hitoshi Naoe*, *Yoshihiro Ohtani*, and *Naruichi Yokogawa*
(Sharp Corporation)

Status of Document:

Version 0.42 (September 29, 1998)	–	Initial document (reached DIRECTIONAL/DRAFT status 10/98)
Version 0.90 (January 8, 1999)	–	Reworked and enhanced *Version 0.42* document (FINAL proposal)

Contacts:

Walter Hirt	τ	Tel.: +41-1-724-8477 ;	e-mail: hir@zurich.ibm.com
John Petrilla	τ	Tel.: +1- 408-435-6608 ;	e-mail: john_petrilla@hp.com
Youichi Yuuki	τ	Tel.: +81-745-63-3517 ;	e-mail: yuuki@ex.shinjo.sharp.co.jp

TABLE OF CONTENTS

1. Overview

This document describes proposed changes to the *IrDA Serial Infrared Physical Layer Link Specification*, Version 1.3, to support a 16 Mb/s data rate extension, also called **Very Fast IR (VFIR)**. While link distance, bit error ratio, field of view, and intensity levels all remain unchanged, the proposed system adds a **new data rate of 16 Mb/s** and a **new modulation code** to the IrDA specification [1].

The HHH(1, 13) code — a newly developed, low duty cycle, rate 2/3, (d, k) = (1, 13) run-length limited (RLL) code — is proposed as the modulation code to achieve the specified data rate. The HHH(1, 13) code guarantees for at least one empty chip and at most 13 empty chips between chips containing pulses in the transmitted IR signal.

The proposed packet frame structure is based on the current IrDA-FIR (4 Mb/s) frame format with modifications introduced where necessary to accommodate the requirements that are specific to the new modulation code. Furthermore, the proposed system includes a simple scrambling/descrambling scheme.

If this proposal is accepted into final status, the proposed changes will be incorporated into the physical layer document [1], resulting in Version 1.4.

[1] *Infrared Data Association Serial Infrared Physical Layer Link Specification*, Version 1.3, October 15, 1998.

2. New HHH(1, 13) Modulation Code

The HHH(1, 13) modulation code has the following salient features:

• Code Rate:	2/3 ,
• Maximal Duty Cycle:	1/3 (~33%) ,
• Average Duty Cycle:	~26% ,
• Minimal Duty Cycle:	1/12 (~8.3%) ,
• Run-Length Constraints:	(d, k) = (1, 13) ,
• Longest Run of ‘10’s:	yyy’000’101’010’101’000’yyy ,
• Chip Rate @ Data Rate 16 Mb/s:	24 Mchips/s ,
• System Clock @ Data Rate 16 Mb/s:	N×12 MHz (where N ≥ 4).

The run length constraints (d, k) = (1, 13) ensure an inactive chip after each active chip, i.e. only single-chip-width pulses occur. The details on the encoding and decoding functions and their basic implementations are defined and described in **Appendix A**. The information contained therein allows for the development of complete encoder and decoder circuitry.

To take full advantage of the d = 1 feature of HHH(1, 13) in strong signal conditions, clock and data recovery circuitry should be designed to ignore the level of the chip following an active chip and assume these chips are inactive.

The proposed modulation code is enhanced with simple frame-synchronized scrambler/descrambler mechanisms as defined and described in **Appendix B**. While such a scheme does not eliminate worst-case duty cycle signal patterns in all specific cases, the probabilities of their occurrence are reduced significantly on average. This leads to a better “eye” opening and reduced jitter in the recovered signal stream for typical payload data.

3. Adding the New Signaling Rate and Modulation Code

3.1 Changes to Add the New Signaling (Data) Rate of 16 Mb/s

To accommodate the 16 Mb/s signaling (data) rate, Table 2 in Section 4.1, Table 3 in Section 4.2, and Table 4 in Section 4.3, all require modifications. No modification is expected for Table 1 of Section 4.1. The SIP requirements of Section 4.1 and Section 5.2 do not change and continue to hold for this data rate.

Signaling Rate	Modulation	Rate Tolerance % of Rate	Pulse Duration Minimum	Pulse Duration Nominal	Pulse Duration Maximum
16 Mb/s	HHH(1, 13)	+/- 0.01	38.3 ns	41.7 ns	45.0 ns

Table 2. Signaling rate and pulse duration specifications (16 Mb/s Addition).

The rate tolerance of +/-100 ppm is the same as that for 4 Mb/s 4PPM. With the extension to 16 Mb/s and the new modulation code, the nominal chip rate increases from 8 Mchips/s (for 4 Mb/s 4PPM) to 24 Mchips/s (for 16 Mb/s HHH(1, 13) modulation).

SPECIFICATION	Data Rates	Type	Minimum	Maximum
Rise Time, 10-90%, ns	16 Mb/s	Std	-	19
Fall Time, 90-10%, ns	16 Mb/s	Std	-	19
Peak-to-Peak Edge Jitter, % of nominal chip duration	16 Mb/s	Std	-	8.0

Table 3. Active output specifications (16 Mb/s Addition).

To accommodate the most efficient LEDs possible, rise and fall time allocations of 45.6% of the nominal chip time will be used instead of the 32% used for 4 Mb/s 4PPM. The tolerance for peak-to-peak edge jitter, 8% of the nominal chip time, is the same as that for 4 Mb/s 4PPM.

SPECIFICATION	Data Rates	Type	Minimum	Maximum
Receiver Latency Allowance, ms	16 Mb/s	Std	-	0.10

Table 4. Active input specifications (16 Mb/s Addition).

A reduction in the maximum receiver latency allowance to 0.10 ms (= 100 μs) is required to reduce dead time between packets and enable a high effective date rate (packet throughput efficiency). The larger maximum values currently permitted at the lower data rates will continue to be available for those rates.

3.2 Changes to Accommodate the New Modulation Code

To accommodate the new modulation code, HHH(1, 13), a new sub-section (5.5) will be required in Section 5: *0.576, 1.152 and 4.0 Mb/s Modulation and Demodulation*. Within the new sub-section, data encoding and decoding for HHH(1, 13) and the packet/frame format for 16 Mb/s HHH(1, 13) will be described as follows.

For the purpose of this document, please refer to **Appendix A** for all relevant information required to implement the encoding and decoding circuits for the HHH(1, 13) modulation code.

The packet format for 16 Mb/s HHH(1, 13) modulation has the following form:

PREAMBLE (PA)	START (STA)	IrLAP Frame	CRC	Flush Byte (FB)	STOP (STO)	NULL

The individual frame fields are defined as follows:

PREAMBLE (PA):

The transmitted PREAMBLE (PA) is constructed by concatenating ten times (10×) the 24-chip (1 µs) PREAMBLE PERIOD (PP), where

PP = ’100’010’010’001’001’001’000’100’,

to form the complete 240-chip (10 µs) preamble

PA = ’PP’PP’PP’PP’PP’PP’PP’PP’PP’PP’.

The left-most/right-most chip of PP and PA, respectively, is transmitted first/last and a ’1’ in PP means an active chip (pulse) and a ’0’ means an empty chip (no pulse).

START (STA):

The transmitted START (STA) delimiter is the 48-chip (2 µs) chip sequence

STA = ’100’101’010’100’100’010’000’001’001’010’101’001’000’001’010’000’.

The left-most/right-most chip of STA is transmitted first/last and a ’1’ in STA means an active chip (pulse) and a ’0’ means an empty chip (no pulse).

IRLAP FRAME:

The structure remains unchanged from that defined in the IrLAP Specification, Version 1.1. The content of the IrLAP frame is first scrambled with the scheme recommended in **Appendix B** of this document and then encoded with HHH(1, 13) as described in **Appendix A** of this document. Note that the 32 CRC bits for the IrLAP frame are calculated before the IrLAP frame is scrambled. For reference, the IrLAP frame has the following structure:

| Address (8 bits) | Control (8 bits) | Information (M times 8 bits) |

CRC:

Computation remains unchanged from the 32-bit CRC defined for the 4 Mb/s data rate. Please refer to the *IrDA Physical Layer Specification*, Version 1.3, for this CRC function. The content of the CRC field is first scrambled with the scheme recommended in **Appendix B** of this document and then encoded with HHH(1, 13) as described in **Appendix A** of this document. Note that the 32 CRC bits for the IrLAP frame are calculated before the IrLAP frame is scrambled. The transmitted CRC field is a 48-chip (2 µs) sequence.

FLUSH BYTE (FB):

The Flush Byte (FB) is the 8-bit sequence

FB = ’00’00’00’00’.

These 8 bits are not scrambled but directly sent to the HHH(1, 13) encoder described in **Appendix A** of this document. The transmitted FB field is a 12-chip (0.5 µs) sequence. Note that the FB field is required to enable complete decoding of the CRC field.

STOP (STO):

The transmitted STOP (STO) delimiter is the 48-chip (2 µs) sequence

STO = ’001’001’010’101’001’000’100’000’100’101’010’100’100’000’100’000’.

The left-most/right-most chip of STO is transmitted first/last and a ’1’ in STO means an active chip (pulse) and a ’0’ means an empty chip (no pulse).

NULL:

The transmitted NULL sequence is the 24-chip (1 µs) sequence

NULL = ’000’000’000’000’000’000’000’000’.

The NULL field is a new field for the purpose of providing an HHH(1, 13) code pattern violation that permits terminating reception of the packet in the event that the STO field is not recognized. The left-most/right-most chip in NULL is transmitted first/last and all chips of NULL are empty chips (no pulses).

4. Receiver Data and Calculated Performance

The following table will be added to Appendix B.4. The parameters in this table represent an example of a receiver design that is sufficient to implement a system for 16 Mb/s data rate with the HHH(1, 13) modulation code.

The analysis starts with a minimum irradiance of 10 µW/cm^2, the sunlight ambient requirement of 490 µW/cm^2, and determines a set of receiver characteristics that will provide 6.0 dB of link margin above the ideal SNR of 11.4 needed to support the maximum BER requirement of 10^{-8}.

PARAMETERS		MIN.	MAX.	How to calculate
SPECIFICATIONS				
Peak Wavelength, nm		850	900	
Half Angle, degree		15		
Minimum Link Length, m			0.01	
Maximum Link Length, m		1		
Intensity In Angular range, mW/sr		100	500	
Single Pulse Width, ns		38.3	45.0	± 8 % (% Chip Width)
Rise / Fall time, ns			19.0	45.6 % (% Chip Width)
Contributed Peak to Peak Jitter, ns			3.3	8% (% Chip Width)
Minimum Irradiance In Angular Range, µW/cm^2	a		10	
Maximum Irradiance In Angular Range, mW/cm^2	b	500		
Link Optical Attenuation, dB			40	
Optical Dynamic range, dB		47.0		= 10*log(b/a)
Sunlight Ambient Irradiance, µW/cm^2	c		490	
Bit Error Ratio (BER)			1.00E-08	
Required Signal-to-Noise (S/N) Ratio for BER	d	11.4		
Latency, µs			100	
RECEIVER DATA (NOT Specifications)				
Detector Responsivity, µA/(mW/cm^2)	e	100		
Receiver Input Noise Current Density, pA/ Hz	f	4.17		= 2.5nV/(Hz)^0.5/(600 Ohm)
Receiver 3dB Band [LPF cut-off freq.], MHz	g	12	14.5 *)	*) nominal Value
Receiver 3dB Band [HPF cut-off freq.], MHz	h	0.09	0.14	
Receiver 3dB Bandwidth, MHz	k		14.36	= g-h
CALCULATED PERFORMANCE				
Sunlight Photo Current, µA	m	49.0		= c*e
Sunlight noise Current density, pA/ Hz	n		3.96	= (2*1.6E-19*m)^0.5
Sunlight noise Current, nA	p		15.01	= n*k^0.5
Receiver Input Noise Current, nA	q		15.79	= f*k^0.5
Total Receiver Noise Current, nA	r		21.78	= (p^2+q^2)^0.5
Receiver Signal Current, nA	s	1000		= a*e
Comparator Threshold, nA		500		= 0.5*s
Receiver Signal-to-Noise (S/N) Ratio	t	45.9		= s/r
Margin (min. S/N)/(required S/N), dB		6.0		= 10*log(t/d)

Table 12. Receiver data and calculated performance for 16 Mb/s.

APPENDIX A

A1 – HHH(1, 13) Encoding Equations

Define the following encoder signal vectors where increasing indexes mean increasing time in the equivalent serial signal streams:

Data input *⁾:	$D = (d_1, d_2)$
*⁾ First data input to be encoded:	$D \equiv \tilde{D} = (\alpha,\ \beta)$
Present state:	$S = (s_1, s_2, s_3)$
Next state:	$N = (n_1, n_2, n_3)$
Internal data:	$B^1 = (B^1{}_1, B^1{}_2) = (b_1, b_2)$ $B^2 = (B^2{}_1, B^2{}_2) = (b_3, b_4)$ $B^3 = (B^3{}_1, B^3{}_2) = (b_5, b_6)$
Internal codeword:	$C = (c_1, c_2, c_3)$
Encoder output:	$Y = (Y_1, Y_2, Y_3)$
Initial conditions (start up):	$S = (s_1, s_2, s_3) = (1, 0, 0)$ when $B^1 = (b_1, b_2) \equiv \tilde{D} = (\alpha,\ \beta)$

With the Boolean operator notation

$$\overline{m} = \text{INVERSE}(m),$$

$$m + n = m \text{ OR } n,$$

$$mn = m \text{ AND } n,$$

the components of N and C are computed in terms of the components of S, B^1, B^2, and B^3 with the following Boolean expressions:

$$n_1 = (s_1 s_3) + (s_3 b_1) + (\overline{s_1} b_1 b_2 \overline{b_3}) + (\overline{s_1} b_1 b_2 \overline{b_4} b_5 b_6),$$
$$n_2 = (\overline{s_3} b_1) + (s_1 s_2 b_1 \overline{b_2}),$$
$$n_3 = (\overline{s_3} b_2) + (\overline{s_1}\ \overline{b_1} b_2) + (s_1 s_2 b_1 \overline{b_2}),$$

$$c_1 = \overline{s_1} s_2,$$
$$c_2 = \overline{s_1}\ \overline{s_2}\ \overline{c_3},$$
$$c_3 = \overline{s_1} s_3 (\overline{b_1} + \overline{b_2}) + (\overline{s_1}\ \overline{s_3} b_1 b_2 \overline{b_3}\ b_4).$$

The vectors B^1, B^2, B^3, S, and Y are outputs of latches; in every encoding cycle, they are updated as follows:

$$B^1 \leftarrow B^2 \leftarrow B^3 \leftarrow D ,$$

$$\leftarrow N , \qquad Y \leftarrow C .$$

Table A1 shows the state transition/output table that corresponds to the HHH(1, 13) code defined by the above equations. The particular HHH(1, 13) code construction and implementation methods require the following interpretation of the table entries with respect to the mapping of *Internal Inputs* and *Present State* into *Next State* and *Internal Output*, respectively:

- A specific data pair D ≡ D* = (□1, □2) arriving at the encoder input is first associated with a corresponding next state N ≡ N*. This occurs as soon as the data D* have advanced into the positions of the internal data bits B^1 = (b1, b2), i.e., when (b1, b2, b3, b4, b5, b6) ≡ (□1, □2, x, x, x, x). In a second step, during the next encoding cycle, the state S takes on the value of N*, i.e., S ≡ S* ← N* so that S is now associated with (□1, □2). In the same cycle, the inner codeword C ≡ C* now carrying the information of D* is computed. Thus, referring to Table A1, a given internal input vector (b1, b2, b3, b4, b5, b6) associates the bits (b1, b2) with the next state N and a given state S associates the data pair ahead of (b1, b2) to the output C. In other words, the pair-wise values for N and C as listed in Table A1 are *not* associated with the same input data pair.

- Encoder initialization: The state S = (s1, s2, s3) = (1, 0, 0) is also used as the initial state of the encoder, i.e., denoting with (α, β) the first pair of data bits to be encoded, the state S is forced to take on the value (1, 0, 0) when the bits (α, β) have advanced into the encoding circuits such that the internal inputs B^1 = (b1, b2) ≡ (α, β). Examples of HHH(1, 13) encoding/decoding can be found in Section A7 of this appendix.

Present State: $S=(s_1,s_2,s_3)$	Next State / Internal Output: $N=(n_1,n_2,n_3)$ / $C=(c_1,c_2,c_3)$							
	Internal Inputs: $(b_1,b_2,b_3,b_4,b_5,b_6)$							
	00xxxx	01xxxx	10xxxx	1100xx	1101xx	111011	1110(~~11~~)	1111xx
0 0 0	000/010	001/010	010/010	111/010	111/001	111/010	011/010	011/010
0 0 1	000/001	001/001	100/001	100/010	100/010	100/010	100/010	100/010
0 1 0	000/100	001/100	010/100	111/100	111/101	111/100	011/100	011/100
0 1 1	000/101	001/101	100/101	100/100	100/100	100/100	100/100	100/100
1 0 0 [1)]	000/000	001/000	010/000	011/000	011/000	011/000	011/000	011/000
1 1 1	100/000	100/000	111/000	100/000	100/000	100/000	100/000	100/000

Table A1: State transition/output table for the HHH(1, 13) code (Note: [1)] the state (s1, s2, s3) = (1, 0, 0) is the required initial state during the one encoding cycle where the internal input pair B^1 = (b1, b2) represents the first data pair to be encoded; ’x’ signifies *don’t care*).

A2 – Reference Implementation of the HHH(1, 13) Encoder

Figure A1 shows the reference implementation of the HHH(1, 13) encoder specified by the equations in Section A1. The purpose of this figure and Table A2 is to illustrate how on the time scale the encoder’s data inputs (d1, d2) are related to the encoder’s output triplets (Y1, Y2, Y3); each of these output triplets, also called codewords, carries the information of a specific pair of input bits. Note that, throughout this document, increasing indexes in the signal vectors mean increasing time in the respective serial signal streams. Correct interpretation and implementation of the HHH(1, 13) code requires that a pair of specific input bits D = (d1, d2) ≡ (□1, □2) arriving at the encoder’s input in the time interval nT (1/T = 24 MHz is the chip frequency) must first be “absorbed” into the next state N ≡ (η1, η2, η3) and then into the state S ≡ (‗1, ‗2, ‗3) before the internal codeword C ≡ (⇳1, ⇳2, ⇳3) associated with (□1, □2) can be computed. In Fig. A1, the next state N ≡ (η1, η2, η3) associated with the data bits (□1, □2) occurs in the time interval (n+9)T, i.e, three encoding cycles (one encoding cycle has duration 3T) after the data bits (□1, □2) have arrived at the encoder input. In the next cycle, (n+12)T, S ≡ (₋1, ₋2, ₋3) takes on the value of N and the inner codeword C ≡ (⇳1, ⇳2, ⇳3) now associated with (□1, □2) is being computed; it takes one further encoding cycle before this codeword C becomes available as the encoder's output codeword Y ≡ (□1, □2, □3) associated with (□1, □2). The encoding process yields therefore a delay of five encoding cycles or, equivalently, of 5×3T = 15T seconds.

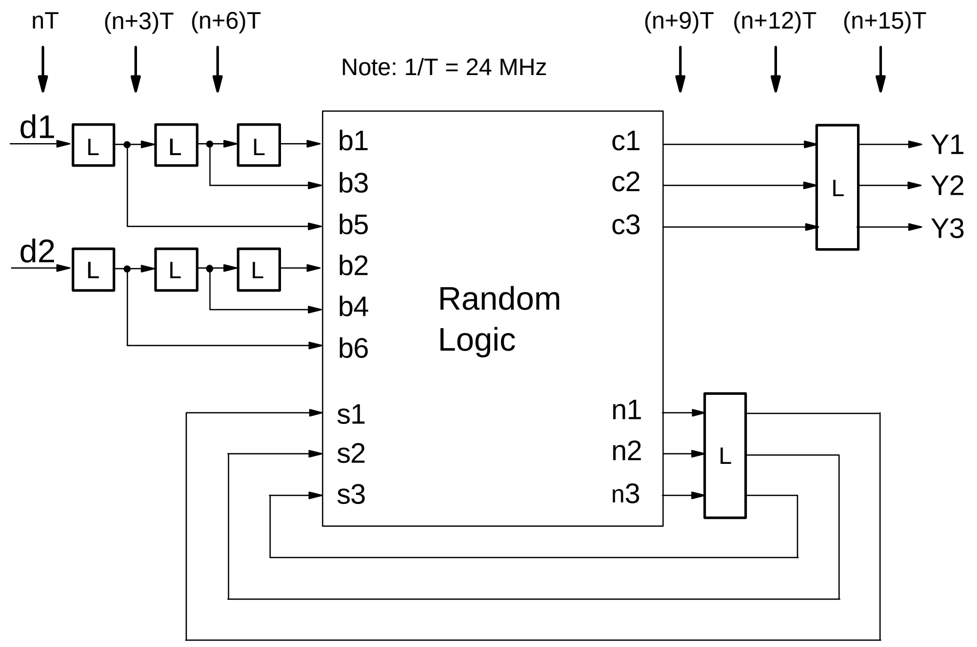

Fig. A1: The reference implementation of the HHH(1, 13) encoder indicating the inherent pipelining of codeword generation. The equations for the random logic that computes the next state N = (n1, n2, n3) and the inner codeword C = (c1, c2, c3), respectively, are defined in Section A1 of this appendix. Note that the delay of HHH(1, 13) encoding is five encoding cycles or, equivalently, 15 chips each of length T = 41.7 ns (see also Table A2).

Time Interval	…	nT	(n+3)T	(n+6)T	(n+9)T	(n+12)T	(n+15)T	…
D = (d1, d2)	…	**(□1, □2)**	(x , x)	(x , x)	(x , x)	(x , x)	(x , x)	…
N = (n1, n2, n3)	…	(x, x, x)	(x, x, x)	(x, x, x)	**(η1, η2, η3)**	(x, x, x)	(x, x, x)	…
S = (s1, s2, s3)	…	(x, x, x)	(x, x, x)	(x, x, x)	(x, x, x)	**(₋1, ₋2, ₋3)**	(x, x, x)	…
C = (c1, c2, c3)	…	(x, x, x)	(x, x, x)	(x, x, x)	(x, x, x)	**(γ1, γ2, γ3)**	(x, x, x)	…
Y = (Y1, Y2, Y3)	…	(x, x, x)	(x, x, x)	(x, x, x)	(x, x, x)	(x, x, x)	**(□1, □2, □3)**	…

Table A2: This table illustrates that the delay of HHH(1, 13) encoding is five encoding cycles, or 15 chips. Referring to Fig. A1, a specific data pair D ≡ (□1, □2) arriving at the encoder input in the interval nT is first associated with the next state N ≡ (η1, η2, η3) during time interval (n+9)T, when (b1, b2) ≡ (□1, □2). During the next time interval, (n+12)T, the state S takes on the value of N and – based on this state – the inner codeword C ≡ (γ1, γ2, γ3) is computed which now carries the information of (□1, □2). In the time interval (n+15)T, the encoder output associated with the data pair (□1, □2), Y ≡ (□1, □2, □3), leaves the encoder (Note: 1/T = 24 MHz is the chip frequency and 'x' signifies *don't care*).

A3 – Gate-Level Implementation of the HHH(1, 13) Encoder

Figure A2 shows the basic recommended gate-level implementation of the HHH(1, 13) encoder as specified by the equations in Section A1. The required initialization circuits for the state S = (s1, s2, s3) are not shown.

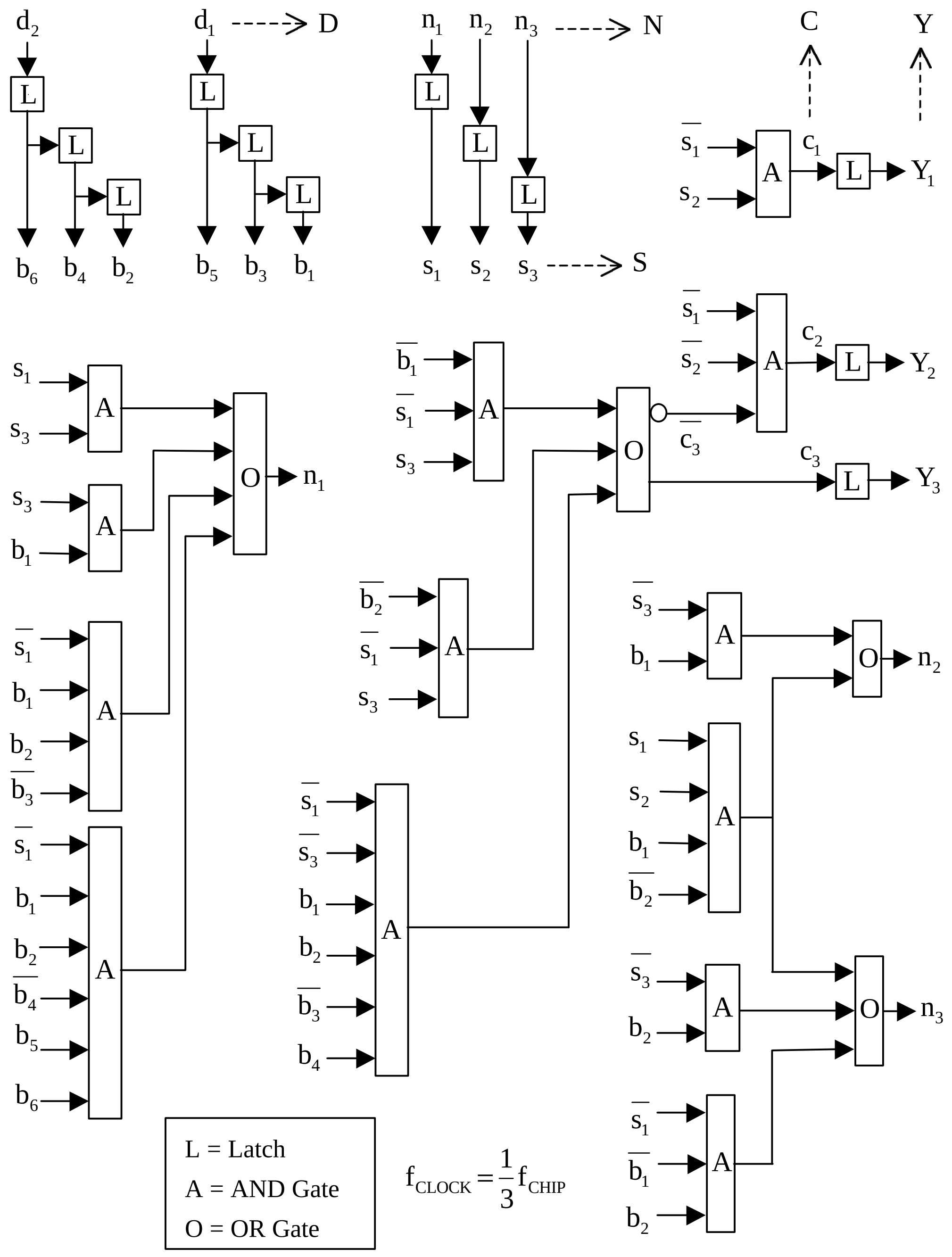

Fig. A2. Basic recommended gate-level implementation of the HHH(1, 13) encoder.

A4 – HHH(1, 13) Decoding Equations

Define the following decoder signal vectors where increasing indexes mean increasing time in the equivalent serial signal streams:

Received codeword:	$R = (r_1, r_2, r_3)$
Internal codewords:	$Y^4 = (y_{10}, y_{11}, y_{12})$
	$Y^3 = (y_7, y_8, y_9)$
	$Y^2 = (y_4, y_5, y_6)$
	$Y^1 = (y_1, y_2, y_3)$
Internal variables:	$Z_B = \overline{y_4 + y_5 + y_6}$
	$Z_C = \overline{y_7 + y_8 + y_9}$
	$Z_D = \overline{y_{10} + y_{11} + y_{12}}$
	$X^1 = (X^1{}_1, X^1{}_2) = (x_1, x_2)$
	$X^2 = (X^2{}_1, X^2{}_2) = (x_3, x_4)$
	$X^3 = (X^3{}_1, X^3{}_2) = (x_5, x_6)$
	$W = (w_1, w_2)$
	$V = (v_1, v_2)$
Decoder output:	$U = (u_1, u_2)$
Initial conditions (start up):	*None*

The components of X^1, X^2, and X^3 are computed with the following Boolean expressions (for the definition of the Boolean operator notation see Section A1 of this appendix):

$$x_1 = v_1$$
$$x_2 = (y_6 \overline{Z_C}) + (\overline{Z_B} Z_C \overline{Z_D}) + v_2$$
$$x_3 = (Z_B Z_C Z_D) + (\overline{Z_B} Z_C) + w_1 + w_2$$
$$x_4 = (Z_B Z_C \overline{Z_D}\, y_3) + [\overline{Z_B} Z_C (Z_D + \overline{y_6})] + w_2$$
$$x_5 = y_{10}$$
$$x_6 = Z_B Z_C Z_D$$

The vectors Y^1, Y^2, Y^3, Y^4, U, V, and W are outputs of latches; in every decoding cycle, they are updated as follows:

$$Y^1 \leftarrow Y^2 \leftarrow Y^3 \leftarrow Y^4 \leftarrow R \ ,$$

$$W \leftarrow X^3 \ , \quad V \leftarrow X^2 \ , \quad U \leftarrow X^1 \ ,$$

where U represents the decoded data bit pair. Note that both Z_B and Z_C can be directly obtained from delayed versions of Z_D (see also Figs. A3 and A4):

$$Z_B \leftarrow Z_C \leftarrow Z_D \ .$$

A5 – Reference Implementation of the HHH(1, 13) Decoder

Figure A3 shows the reference implementation of the HHH(1, 13) decoder specified by the equations in Section A4. The decoding delay of this decoder is four decoding cycles or 12T seconds where T = 41.7 ns.

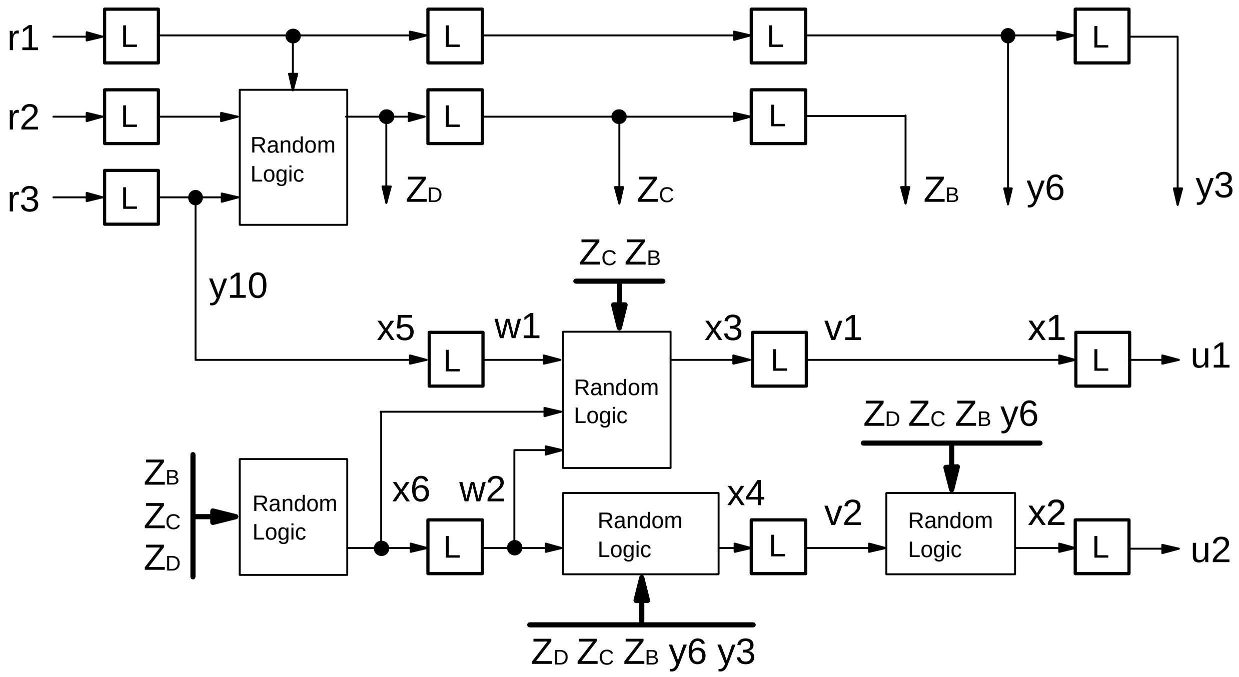

Fig. A3: The reference implementation of the HHH(1, 13) decoder. The equations for the random logic circuits that compute Z_D , x6, x4, x3, and x2, respectively, are listed in Section A4 of this appendix. This form of implementation makes use of the fact that Z_B and Z_C are delayed versions of Z_D. The delay of HHH(1, 13) decoding is four decoding cycles or, equivalently, 12 chips each of length T = 41.7 ns.

A6 – Gate-Level Implementation of the HHH(1, 13) Decoder

Figure A4 shows the basic recommended gate-level implementation of the HHH(1, 13) decoder as specified by the equations in Section A4. This implementation makes use of the fact that Z_B and Z_C are delayed versions of Z_D.

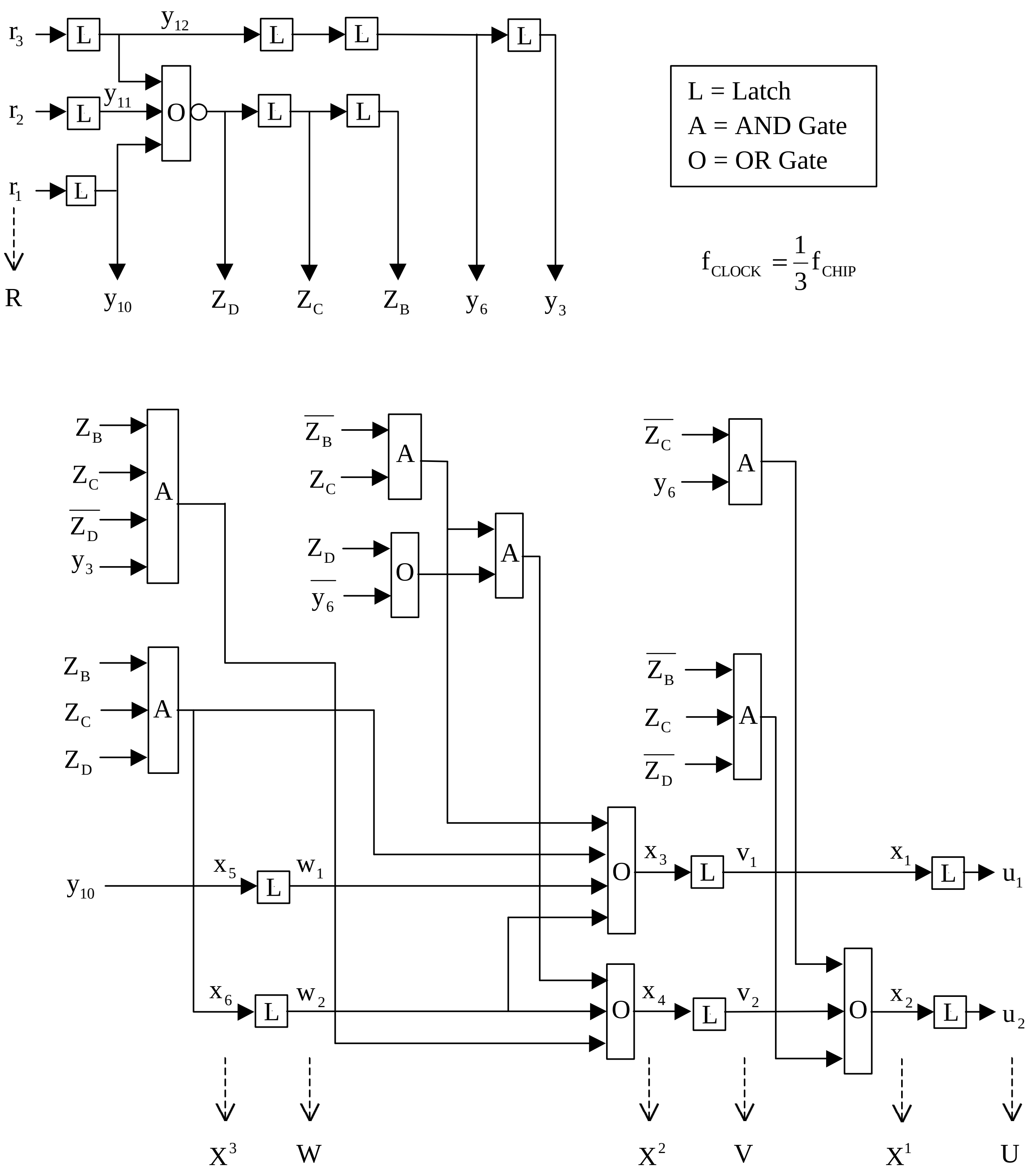

Fig. A4. Basic recommended gate-level implementation of the HHH(1, 13) decoder.

A7 – Encoding/Decoding Examples

EXAMPLE 1:

Scrambled payload:	{(d1, d2)}	=	(1, 1)	(0, 0)	(0, 0)	(0, 0)	(1, 1)	(0, 0)	(0, 0)	(0, 0)
Encoder output:	{(Y1, Y2, Y3)}	=	(1, 0, 1)	(0, 1, 0)	(0, 1, 0)	(0, 1, 0)	(0, 0, 0)	(0, 0, 0)	(0, 1, 0)	(0, 1, 0)
Decoded payload:	{(u1, u2)}	=	(1, 1)	(0, 0)	(0, 0)	(0, 0)	(1, 1)	(0, 0)	(0, 0)	(0, 0)

Legend:

a	=	time index nT, n = 0, 1, … (a = *: reset latches to logic 0)
bc	=	data input, D = (d1, d2)
d	=	control signal: d = 1 enforces N = (n1, n2, n3) = (1, 0, 0)
efghij	=	internal data, (b1, b2, b3, b4, b5, b6)
klm	=	state, S = (s1, s2, s3)
nop	=	next state, N = (n1, n2, n3)
qrs	=	internal codeword, C = (c1, c2, c3)
tuv	=	encoder output, Y = (Y1, Y2, Y3)
wx	=	data bits carried by Y
y	=	control signal: y = 1 signals valid encoder output Y
z	=	count of encoding cycles

a	bc	d	efghij	klm	nop	qrs	tuv	wx	y	z	– **Notes:**
*	00	1	000000	000	100	010	000	00	0	*	– Reset state / set N = (1, 0, 0)
0	**11**	1	000000	100	100	000	010	00	0	0	– First data at input, (d1, d2) ≡ (α, β) = (1, 1)
3	**00**	1	000011	100	100	000	000	00	0	1	
6	**00**	1	001100	100	100	000	000	00	0	2	
9	**00**	0	**11**0000	**100**	011	000	000	00	0	3	– **S = (1, 0, 0) when (b1, b2) ≡ (α, β) = (1, 1)**
12	**11**	0	000000	011	000	101	000	00	0	4	
15	**00**	0	000011	000	000	010	**101**	11	1	5	- First valid output Y / carries (α, β) = (1, 1)
18	**00**	0	001100	000	000	010	**010**	00	1	6	
21	**00**	0	110000	000	111	010	**010**	00	1	7	- Last data at input, (d1, d2) = (0, 0)
24	00	0	000000	111	100	000	**010**	00	1	8	– First flush bits at input
27	00	0	000000	100	000	000	**000**	11	1	9	
30	00	0	000000	000	000	010	**000**	00	1	10	
33	00	0	000000	000	000	010	**010**	00	1	11	- Last flush bits at input
36	[00]	0	000000	000	000	010	**010**	00	1	12	– Last output Y carrying data
39	[00]	0	000000	000	000	010	010	00	1	13	– First output Y carrying flush bits
42	[00]	0	000000	000	000	010	010	00	1	14	
45	[00]	0	000000	000	000	010	010	00	1	15	
48	[00]	0	000000	000	000	010	010	00	1	16	– Last output Y carrying flush bits

Table A3: Encoder states for payload sequence of Example 1. After the last flush bits have appeared at the encoder's input (time index 33), all-0 dummy data [00] is fed to the encoder during the last five encoding cycles, until the output Y carrying the last pair of flush bits becomes available (time index 48).

Legend:

`a`	=	time index nT, n = 0, 1, … (a = `*`: reset latches to logic 0)
`bcd`	=	received codeword, R = (r1, r2, r3)
`efg`	=	internal codeword, Y^4 = (y10, y11, y12)
`hij`	=	internal variables, (Z_D, Z_C, Z_B), where Z_C and Z_B are outputs of latches as shown in Fig. A4
`kl`	=	internal variables, W =(w1, w2)
`mn`	=	internal variables, V = (v1, v2)
`op`	=	decoder output, U = (u1, u2)
`q`	=	control signal: q = 1 signals valid decoder output
`r`	=	count of decoding cycles

a	bcd	efg	hij	kl	mn	op	q	r	– **Notes:**
*	000	000	100	00	00	00	0	*	– Reset state (all latches logic 0)
0	**101**	000	110	00	00	00	0	0	– First valid received input R
3	**010**	101	011	00	11	00	0	1	
6	**010**	010	001	10	00	11	0	2	
9	**010**	010	000	00	10	00	0	3	
12	**000**	010	000	00	00	**11**	1	4	– **First valid decoded data pair U at output**
15	**000**	000	100	00	00	**00**	1	5	
18	**010**	000	110	00	00	**00**	1	6	
21	**010**	010	011	00	11	**00**	1	7	– Last input R carrying data
24	010	010	001	00	00	**11**	1	8	– First input R carrying flush bits
27	010	010	000	00	00	**00**	1	9	
30	010	010	000	00	00	**00**	1	10	
33	010	010	000	00	00	**00**	1	11	– Last valid decoded data pair U at output

Table A4: Operation of the HHH(1, 13) decoder shown in Fig. A4 for the payload of Example 1.

EXAMPLE 2:

Scrambled payload:	{(d1, d2)}	=	(1, 1)	(0, 1)	(0, 0)	(0, 0)	(1, 1)	(0, 1)	(0, 0)	(0, 0)
Encoder output:	{(Y1, Y2, Y3)}	=	(1, 0, 1)	(0, 0, 1)	(0, 1, 0)	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)	(0, 1, 0)	(0, 1, 0)
Decoded payload:	{(u1, u2)}	=	(1, 1)	(0, 1)	(0, 0)	(0, 0)	(1, 1)	(0, 1)	(0, 0)	(0, 0)

EXAMPLE 3:

Scrambled payload:	{(d1, d2)}	=	(0, 1)	(0, 0)	(1, 1)	(0, 0)	(0, 0)	(1, 1)	(0, 1)	(1, 0)
Encoder output:	{(Y1, Y2, Y3)}	=	(0, 0, 1)	(0, 1, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 1)	(0, 0, 0)	(0, 0, 0)	(1, 0, 0)
Decoded payload:	{(u1, u2)}	=	(0, 1)	(0, 0)	(1, 1)	(0, 0)	(0, 0)	(1, 1)	(0, 1)	(1, 0)

APPENDIX B

B1 – Scrambling/Descrambling Functions

It is advantageous to enhance the encoder/decoder system with simple scrambler/descrambler functions. The primitive polynomial

$$x^8 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1 ,$$

where $\oplus$ indicates a modulo-2 addition or, equivalently, a logic exclusive OR (XOR) operation, is proposed for implementing these functions. The operations of the proposed scrambling and descrambling functions are performed according to the principles of <u>frame synchronized scrambling/descrambling (FSS) mechanisms</u>. Note that FSS does not introduce memory into the signal path, i.e., FSS does not increase the encoding/decoding delay and it does not aggravate error propagation in the decoded data stream. The hardware used for scrambling during transmission can mostly be reused during the descrambling process in reception mode.

The reference hardware implementation of the proposed scrambling/descramblng scheme is shown in Fig. B1. The linear feedback shift register (LFSR) produces a maximum-length pseudo-random sequence with period 255. It is important to note that the proposed scrambling/descrambling functions are implemented with an LFSR where the feedback taps are configured according to the so-called <u>one-to-many implementation</u>; for reasons of compatibility, implementations should adhere to this type of LFSR. Furthermore, it is assumed that <u>the output of register cell x6 shown in Fig. B1 is defined to be the equivalent serial output of the LFSR</u>.

The modulo-2 adders shown in Fig. B1 correspond to logic XOR (exclusive OR) gates. During transmission, each new pair of source bits (d1', d2') is XOR-ed with a new pair of scrambling bits (s1, s2) to produce the scrambled data bit pair (d1, d2) entering the encoder. Similarly, during reception, each new pair of decoded bits (u1, u2) is XOR-ed with a new pair of descrambling bits (s1, s2) to produce the descrambled user bit pair (u1', u2') that is sent to the data sink. A scrambling/descrambling cycle has duration 3T seconds where T = 41.7 ns is the chip period.

Effects and Limits of Scrambling/Descrambling:

By enhancing the system with scrambling/descrambling functions during data transmission/reception, one achieves generally better duty cycle statistics in the HHH(1, 13) coded channel chip stream; the resulting duty cycle converges towards the average duty cycle of the code (≈26%) for typical payload data. It is important to note that <u>scrambling cannot entirely eliminate possible worst-case duty cycle patterns</u> in the transmitted signal stream that can result from certain specific input data sequences. However, scrambling can greatly reduce the probability of occurrence of such worst-case patterns.

Scrambler/Descrambler Initialization:

<u>Transmit mode:</u> The scrambler's LFSR is initialized with the all-1 state, that is (x8, x7, x6, x5, x4, x3, x2, x1) = (1, 1, 1, 1, 1, 1, 1, 1), such that (s1, s2) = (x6, x5) = (1, 1) at the arrival of the first pair of source bits (d1', d2'), ready to be scrambled. Note that <u>the LFSR must be advanced *twice* per scrambling cycle</u> to produce a new pair of scrambling bits (s1, s2) for each new pair of data bits (d1', d2').

<u>Receive mode:</u> The descrambler's LFSR is initialized with the all-1 state, that is (x8, x7, x6, x5, x4, x3, x2, x1) = (1, 1, 1, 1, 1, 1, 1, 1), such that (s1, s2) = (x6, x5) = (1, 1) when the decoder produces the first pair of decoded bits (u1, u2), ready to be descrambled. Note that <u>the LFSR must be advanced *twice* per descrambling cycle</u> to produce a new pair of descrambling bits (s1, s2) for each new pair of decoded data bits (u1, u2).

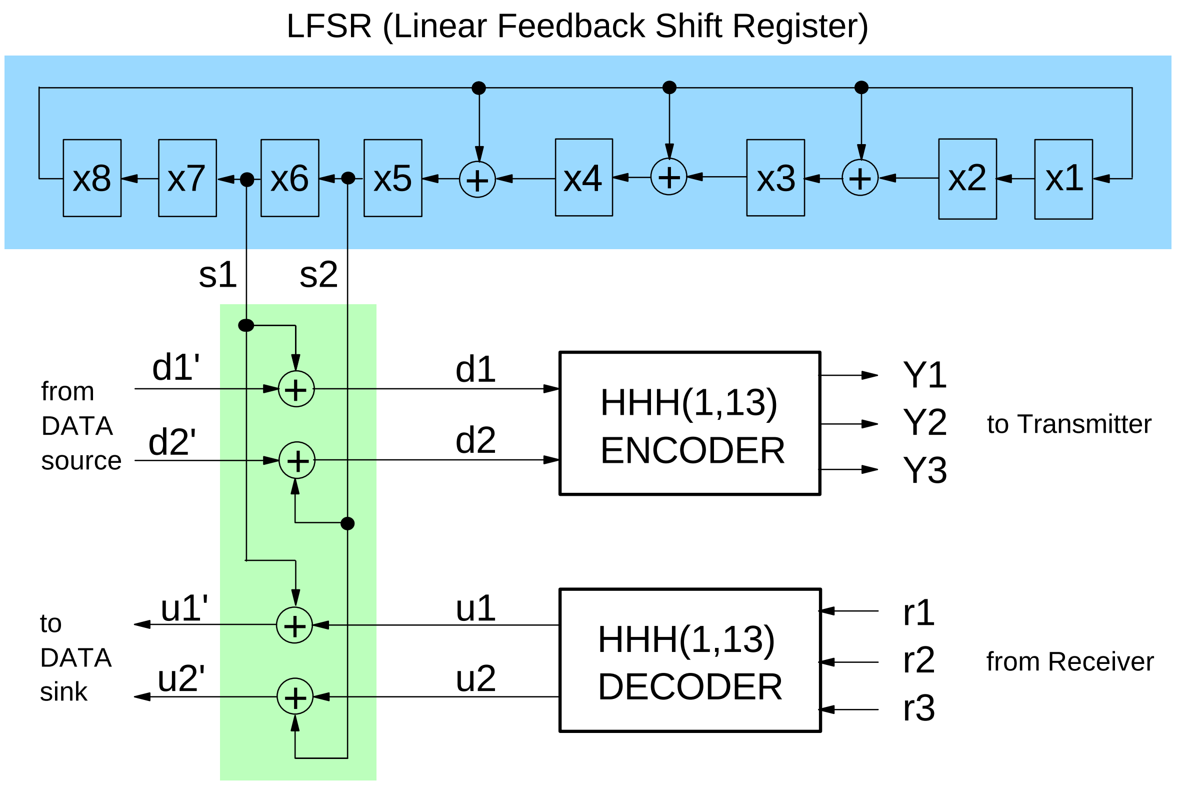

Fig. B1. Reference hardware to implement the scrambling/descrambling functions. The LFSR is implemented in the one-to-many form.

B2 – State Table of Scrambler/Descrambler Reference Hardware

Table B1 represents the complete state table of the scrambler/descrambler hardware shown in Fig. B1 where we have defined that a new state is reached after the LFSR has been clocked *twice*. The LFSR's state is represented by its contents, i.e., (x8, x7, x6, x5, x4, x3, x2, x1), x*i* χ [0, 1], …*i*. The table lists also the scrambling/descrambling bit pairs (s1, s2) = (x6, x5) that are valid in each state. The state sequence {(x8, x7, x6, x5, x4, x3, x2, x1)} and thus the sequence of scrambling/descrambling bit pairs {(s1, s2)} have period 255, i.e., they both repeat after 255 scrambling/descrambling cycles. The all-0 state (x8, x7, x6, x5, x4, x3, x2, x1) = (0, 0, 0, 0, 0, 0, 0, 0) does not occur (it is not allowed at any time). The table also indicates that the equivalent serial sequence formed from the pair sequence {(s1, s2)} consists of two periods of the maximum-length pseudo-random sequence (MLPRS) of length 255 bits, as determined by the scrambling/descrambling polynomial shown above.

Example of Scrambled/Descrambled Data Sequences:

The scrambled sequence, {(d1, d2)}, in this example corresponds to Example 1 in Section A5 of Appendix A.

Payload sequence:	{(d1', d2')}	=	(0, 0) (0, 1) (0, 0) (1, 1) (1, 1) (1, 1) (0, 1) (0, 1)
Scrambling sequence:	{(s1, s2)}	=	(1, 1) (0, 1) (0, 0) (1, 1) (0, 0) (1, 1) (0, 1) (0, 1)
Scrambled sequence:	{(d1, d2)}	=	(1, 1) (0, 0) (0, 0) (0, 0) (1, 1) (0, 0) (0, 0) (0, 0)

Decoded scrambled sequence:	{(u1, u2)}	=	(1, 1) (0, 0) (0, 0) (0, 0) (1, 1) (0, 0) (0, 0) (0, 0)
Descrambling sequence:	{(s1, s2)}	=	(1, 1) (0, 1) (0, 0) (1, 1) (0, 0) (1, 1) (0, 1) (0, 1)
Descrambled payload sequence:	{(u1', u2')}	=	(0, 0) (0, 1) (0, 0) (1, 1) (1, 1) (1, 1) (0, 1) (0, 1)

Legend:

a = count index for scrambling/descrambling cycle
bcdefghi = x8, x7, x6, x5, x4, x3, x2, x1 (LFSR contents = state)
jk = s1, s2 (pair of scrambling/descrambling bits)

*) First cycle of the first scrambling/descrambling period (a = 1)

First bit of first serial MLPRS (a =1: j = s1 = x6)
↓
Second bit of first serial MLPRS (a = 1: k = s2 = x5)
↓

a	bcdefghi	jk	a	bcdefghi	jk	a	bcdefghi	jk	a	bcdefghi	jk
*) 1	11111111	11	65	01000110	00	129	11100011	10	193	10001100	00
2	11011011	01	66	00000101	00	130	10101011	10	194	00001010	00
3	01001011	00	67	00010100	01	131	10010110	01	195	00101000	10
4	00110001	11	68	01010000	01	132	01100010	10	196	10100000	10
5	11000100	00	69	01011101	01	133	10010101	01	197	10111010	11
6	00110111	11	70	01101001	10	134	01101110	10	198	11010010	01
7	11011100	01	71	10111001	11	135	10100101	10	199	01101111	10
8	01010111	01	72	11011110	01	136	10101110	10	200	10100001	10
9	01000001	00	73	01011111	01	137	10000010	00	201	10111110	11
10	00011001	01	74	01100001	10	138	00110010	11	202	11000010	00
11	01100100	10	75	10011001	01	139	11001000	00	203	00101111	10
12	10001101	00	76	01011110	01	140	00000111	00	204	10111100	11
13	00001110	00	77	01100101	10	141	00011100	01	205	11001010	00
14	00111000	11	78	10001001	00	142	01110000	11	206	00001111	00
15	11100000	10	79	00011110	01	143	11011101	01	207	00111100	11
16	10100111	10	80	01111000	11	144	01010011	01	208	11110000	11
17	10100110	10	81	11111101	11	145	01010001	01	209	11100111	10
18	10100010	10	82	11010011	01	146	01011001	01	210	10111011	11
19	10110010	11	83	01101011	10	147	01111001	11	211	11010110	01
20	11110010	11	84	10110001	11	148	11111001	11	212	01111111	11
21	11101111	10	85	11111110	11	149	11000011	00	213	11100001	10
22	10011011	01	86	11011111	01	150	00101011	10	214	10100011	10
23	01010110	01	87	01011011	01	151	10101100	10	215	10110110	11
24	01000101	00	88	01110001	11	152	10001010	00	216	11100010	10
25	00001001	00	89	11011001	01	153	00010010	01	217	10101111	10
26	00100100	10	90	01000011	00	154	01001000	00	218	10000110	00
27	10010000	01	91	00010001	01	155	00111101	11	219	00100010	10
28	01111010	11	92	01000100	00	156	11110100	11	220	10001000	00
29	11110101	11	93	00001101	00	157	11110111	11	221	00011010	01
30	11110011	11	94	00110100	11	158	11111011	11	222	01101000	10
31	11101011	10	95	11010000	01	159	11001011	00	223	10111101	11
32	10001011	00	96	01100111	10	160	00001011	00	224	11001110	00
33	00010110	01	97	10000001	00	161	00101100	10	225	00011111	01
34	01011000	01	98	00111110	11	162	10110000	11	226	01111100	11
35	01111101	11	99	11111000	11	163	11111010	11	227	11101101	10
36	11101001	10	100	11000111	00	164	11001111	00	228	10010011	01
37	10000011	00	101	00111011	11	165	00011011	01	229	01110110	11
38	00110110	11	102	11101100	10	166	01101100	10	230	11000101	00
39	11011000	01	103	10010111	01	167	10101101	10	231	00110011	11
40	01000111	00	104	01100110	10	168	10001110	00	232	11001100	00
41	(next page)		105	(next page)		169	(next page)		233	(next page)	

(continued from previous page)

a	bcdefghi	jk	a	bcdefghi	jk	a	bcdefghi	jk	a	bcdefghi	jk	
41	00000001	00	105	10000101	00	169	00000010	00	233	00010111	01	
42	00000100	00	106	00101110	10	170	00001000	00	234	01011100	01	
43	00010000	01	107	10111000	11	171	00100000	10	235	01101101	10	
44	01000000	00	108	11011010	01	172	10000000	00	236	10101001	10	
45	00011101	01	109	01001111	00	173	00111010	11	237	10011110	01	
46	01110100	11	110	00100001	10	174	11101000	10	238	01000010	00	
47	11001101	00	111	10000100	00	175	10000111	00	239	00010101	01	
48	00010011	01	112	00101010	10	176	00100110	10	240	01010100	01	
49	01001100	00	113	10101000	10	177	10011000	01	241	01001101	00	
50	00101101	10	114	10011010	01	178	01011010	01	242	00101001	10	
51	10110100	11	115	01010010	01	179	01110101	11	243	10100100	10	
52	11101010	10	116	01010101	01	180	11001001	00	244	10101010	10	
53	10001111	00	117	01001001	00	181	00000011	00	245	10010010	01	
54	00000110	00	118	00111001	11	182	00001100	00	246	01110010	11	
55	00011000	01	119	11100100	10	183	00110000	11	247	11010101	01	
56	01100000	10	120	10110111	11	184	11000000	00	248	01110011	11	
57	10011101	01	121	11100110	10	185	00100111	10	249	11010001	01	
58	01001110	00	122	10111111	11	186	10011100	01	250	01100011	10	
59	00100101	10	123	11000110	00	187	01001010	00	251	10010001	01	
60	10010100	01	124	00111111	11	188	00110101	11	252	01111110	11	
61	01101010	10	125	11111100	11	189	11010100	01	253	11100101	10	
62	10110101	11	126	11010111	01	190	01110111	11	254	10110011	11	
63	11101110	10	127	01111011	11	191	11000001	00	255	11110110	11	^)
64	10011111	01	128	11110001	11	192	00100011	10	[256	11111111	11]	″)

↑ First bit of second serial MLPRS (a = 128: k = s2 = x5)

↑ Last (255th) bit of first serial MLPRS (a = 128: j = s1 = x6)

^) End of the first scrambling/descrambling period (a = 255)

″) Start of the second scrambling/descrambling period (a = 256 ≡ 1)

Table B1: The complete state table of the scrambler/descrambler reference hardware shown in Fig. B1.