FABRICACION Y SUMINISTRO DE TARJETAS SIN CONTACTO DE RENFE VIAJEROS CON DISEÑO EN EUSKERA

DOCUMENTACIÓN TÉCNICA PARA PRUEBAS DE CAPACITACIÓN

EXP. 2019-00080

Madrid, enero de 2019

CONTENIDO

El objeto del presente documento es recoger toda la información técnica necesaria para la fabricación de las muestras que deben de presentar los licitadores del expediente 2019-00080, de las matrículas siguientes 207, 208,209 y 210. Esta información, está referida fundamentalmente a, datos para impresión, datos para grabación, cálculo del CRC y datos que contendrá el fichero de retorno o fabricación.

1.- DATOS PARA IMPRESIÓN

1.1.- Nº DE CLIENTE

Cada tarjeta, de las matrículas 207,208 y 209, debe de llevar impreso en el anverso un nº de cliente.

El nº de cliente, está formado por 11 dígitos FFNNNNNNNNV, cuyo significado es el siguiente:

- FF: Conforman el nº de fabricante, que va grabado en el chip.
- NNNNNNNN: Conforman el código de viajero, correlativo, que va grabado en el chip.
- V: Es el dígito de verificación, que se calcula para cada tarjeta, siguiendo el procedimiento que se describe a continuación.

Dígito Verificador
Se calcula de la siguiente forma:

- Se obtiene el resto entre el número de código de viajero y 7.
- Se accede a una tabla que indica el dígito verificador en función del resto:

Resto	Dígito Verificador
1	6
2	5
3	4
4	3
5	2
6	1
0	0

Es decir que sigue el siguiente código en lenguaje C.

```
// doc contiene el número a verificar
int DigitoVerificador(char *doc)
{
        valorcontrol = 7;
        digitoverificador = atol(doc);
        digitoverificador = (digitoverificador % valorcontrol);

        if (digitoverificador > 0L)
                digitoverificador = (valorcontrol-digitoverificador);

        return digitoverificador;
}
```

Ejemplos:
Número: 08 00001378 X

Pasos:
1) Se obtiene RESTO(00001378 / 7) = 6.
2) Se busca el 6 en la tabla, obteniendo el dígito verificador 1.

Por lo tanto el número es:

08 00001378 1

Número: 08 90001378 X

Pasos:
3) Se obtiene RESTO(90001378 / 7) = 5.
4) Se busca el 6 en la tabla, obteniendo el dígito verificador 2.

Por lo tanto el número es:

08 90001378 2

Número: 02 20001378 X

Pasos:
5) Se obtiene RESTO(20001378 / 7) = 5.
6) Se busca el 5 en la tabla, obteniendo el dígito verificador 2.

Por lo tanto el número es:

02 20001378 2

Número: 10 10001378 X

Pasos:
7) Se obtiene RESTO(10001378 / 7) = 2.
8) Se busca el 2 en la tabla, obteniendo el dígito verificador 5.

Por lo tanto el número es:

10 10001378 5

1.2.- Nº DE CHIP

Todas las tarjetas, que componen cada una de las muestras de cada tipo de material, deben de llevar impreso en el reverso, y en el espacio reservado al efecto, en el diseño, el Nº de chip que viene grabado en el propio chip y, ésta impresión, se hará siguiendo **el orden físico de grabación de los Bytes correspondientes, en el mapa de memoria de la tarjeta.**

1.3.- CÓDIGO DATAMATRIX

Todas las tarjetas, excepto de la matrícula 210, deben de llevar impreso en el reverso, y en el espacio reservado al efecto, en el diseño, un código Datamatrix que debe de almacenar la información recogida en la siguiente tabla, y en el orden que también recoge la propia tabla:

Número de serie de la tarjeta	(8 caracteres)	(HEX). Grabado en la tarjeta
Tipo de tarjeta	(2 caracteres)	Grabado en la tarjeta
Fecha de fabricación	(6 caracteres)	(ddmmaa). Grabado en la tarjeta
Empresa fabricante	(2 caracteres)	Grabado en la tarjeta
Código de viajero	(8 caracteres)	Grabado en la tarjeta
Dígito de control	(1 carácter)	(Calculado)
Fecha de Caducidad	(6 caracteres)	(000000)

2.- DATOS PARA GRABACION

2.1.- CLAVES Y CONDICIONES DE ACCESO

Las Claves de acceso, son iguales para todas las matrículas y vienen recogidas en la tabla insertada a continuación:

SECTOR	CLAVE A	CLAVE B
0	00000000000A	00000000000B
1	00000000001A	00000000001B
2	00000000002A	00000000002B
3	00000000003A	00000000003B
4	00000000004A	00000000004B
5	00000000005A	00000000005B

6	00000000006A	00000000006B
7	00000000007A	00000000007B
8	00000000008A	00000000008B
9	00000000009A	00000000009B
10	0000000000AA	0000000000AB
11	0000000000BA	0000000000BB
12	0000000000CA	0000000000CB
13	0000000000DA	0000000000DB
14	0000000000EA	0000000000EB
15	0000000000FA	0000000000FB

Asimismo las Condiciones de Acceso (CDA) se definen de la siguiente forma:

CDA: 78778869
Tipo de Sector: Datos

CDA: 08778F69
Tipo de Sector: Monedero
(Sector 7)

Estos CDA se corresponden a la Lectura con Clave A y la Escritura con Clave B.

En cuanto a las claves AES se grabarán a ceros.

Las tarjetas se suministrarán, por tanto en nivel de seguridad 1.

2.2.- RESTO DE DATOS

Hay que hacer constar aquí, que cuando nos referimos a que un determinado dato a grabar, debe de comenzar en una determinada posición del mapa de memoria, nos referimos a la posición lógica y no a la física, es decir hay que tener en cuenta cual es el byte menos

significativo dentro de un bloque y, a su vez, cual es el bit menos significativo dentro de cada byte.

A continuación se explica el Peso de los Bits en los Bloques, con un ejemplo.

Peso de los Bits en los Bloques.

A efectos de orden el primero del bloque será el bit 1 y el último el bit 128.

Dentro de cada campo o subcampo se considerara el bit menos significativo al primero según la organización señalada para el bloque.

Los bits de control de errores CRC (8bits), utilizados cuando estos bloques son de **datos** (Existen otros tipos de bloques de **valor** como el sector dedicado a monedero), estarán siempre situados en el último byte de cada bloque.

Ejemplo hipotético.

En la siguiente página se muestra un ejemplo de cómo se podría organizar un bloque **hipotético**, con una estructura no real y los siguientes campos:

Fecha y Hora de validación: 32 bits.

AAAAAA : 6 bits : Bit 1-6 (Comienza en año 2000)
MMMM : 4 bits : Bit 7-10
DDDDD: 5 bits : Bit 11-15
HHHHH: 5 bits : Bit 16-20
mmmmmm : 6 bits : Bit 21-26
SSSSSS : 6 bits : Bit 27-32

Saldo de Viajes: 8 bits.

Saldo de viajes disponibles.

Títulos activos: 4 bits.

Bit 1 : Título 1 activo.
Bit 2 : Título 2 activo.
Bit 3 : Título 3 activo.
Bit 4 : Monedero activo.

Pin de usuario: 16bits (4 caracteres BCD)

VALOR REPRESENTADO EN EL EJEMPLO PARA ESTOS CAMPOS:

Fecha : 28/4/2004 **Hora**: 17:15 **Segundos**: 32

Saldo de viajes : 4 **Títulos activos:** Título1 y 3 activos

Pin: 1 2 3

Byte	BYTE 15								BYTE 14								BYTE 13								BYTE 12							
Bit	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
Hex	F				F				F				F				F				F				F				F			
Valor Bit	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Calc. Valor	255																															
Mnemónico	CRC BLOQUE								LIBRES																							
Comentario	Control de errores (valor no refleja realidad)																															

Byte	BYTE 11								BYTE 10								BYTE 9								BYTE 8							
Bit	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
Hex	F				F				F				F				F				F				F				F			
Valor Bit	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Calc. Valor																																
Mnemónico	LIBRES																															
Comentario																																

Byte	BYTE 7								BYTE 6								BYTE 5								BYTE 4							
Bit	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
Hex	F				4				3				2				1				5				0				4			
Valor Bit	1	1	1	1	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0
Calc. Valor					4							3				2				1				5								4
Mnemónico					UPIN																TTA				T1 CV							
Comentario					Pin 4				Pin 3				Pin 2				Pin 1				M	T3	T2	T1	Saldo de viajes							

Byte	BYTE 3								BYTE 2								BYTE 1								BYTE 0							
Bit	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Hex	8				0				F				8				F				1				0				4			
Valor Bit	1	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	0	0	0	0	0	1	0	0
Calc. Valor	32								15				17				28								4				4			
Mnemónico	VFH																															
Comentario	Segundo de validación								Minuto de validación				Hora de validación				Día de validación								Mes de validación				Año de validación			

2.2.1.- MATRICULA 207

DATOS A GRABAR POR EL FABRICANTE EN LAS TARJETAS DE LA MATRÍCULA 207

SECTOR	BLOQUE	P.INICIAL	P.FINAL	NOMBRE DE CAMPO	DATO
0	0	1	32	NUMERO DE SERIE DE LA TARJETA	Nº DE SERIE DEL CHIP
2	0	5	9	TIPO DE TARJETA	30-EN LAS 1.000 VÁLIDAS 31-EN LAS 10 DE FIN DE ROLLO
2	2	22	48	CÓDIGO DE VIAJERO	Del 90030001 al 90031000
4	0	1	7	EMPRESA QUE FABRICA LA TARJETA	54
4	0	8	13	AÑO DE FABRICACION DE LA TARJETA	17
4	0	14	17	MES DE FABRICACION DE LA TARJETA	10
4	0	18	22	DIA DE FABRICACION DE LA TARJETA	21

El resto de bloques de datos se suministra con 0 (en el caso de Monedero, con valor 0), **con su correspondiente CRC.**

Los bloques 0 y 1 del Sector 7, son utilizados como Bloques Valor. Por tal motivo, se almacenará el valor 0, utilizando como Byte final el número Absoluto de Bloque Negado y como valor, el 0. No obstante, como el valor del Sector 7, Bloque 1 es copia del Sector 7, Bloque 0, se utilizará únicamente la ubicación del Sector 7, Bloque 0 en ambos bloques.

Es decir que se almacenarán los siguientes Bytes tanto en el Sector 7, Bloque 0 como en el Bloque 1 del mismo Sector:

0x00 0x00 0x00 0x00 0xFF 0xFF 0xFF 0xFF 0x00 0x00 0x00 0x00 0x1C 0xE3 0x1C 0xE3**.**

2.2.2.- MATRICULA 208

DATOS A GRABAR POR EL FABRICANTE EN LAS TARJETAS DE LA MATRÍCULA 208

SECTOR	BLOQUE	P.INICIAL	P.FINAL	NOMBRE DE CAMPO	DATO
0	0	1	32	NUMERO DE SERIE DE LA TARJETA	Nº DE SERIE DEL CHIP
2	0	5	9	TIPO DE TARJETA	28-EN LAS 1.000 VÁLIDAS 31-EN LAS 10 DE FIN DE ROLLO
2	2	22	48	CÓDIGO DE VIAJERO	Del 90040001 al 90041000
4	0	1	7	EMPRESA QUE FABRICA LA TARJETA	55
4	0	8	13	AÑO DE FABRICACION DE LA TARJETA	17
4	0	14	17	MES DE FABRICACION DE LA TARJETA	10
4	0	18	22	DIA DE FABRICACION DE LA TARJETA	21

El resto de Bloques de datos se suministra con 0 (en el caso de Monedero, con valor 0), **con su correspondiente CRC.**

Los bloques 0 y 1 del Sector 7, son utilizados como Bloques Valor. Por tal motivo, se almacenará el valor 0, utilizando como Byte final el número Absoluto de Bloque Negado y como valor, el 0. No obstante, como el valor del Sector 7, Bloque 1 es copia del Sector 7, Bloque 0, se utilizará únicamente la ubicación del Sector 7, Bloque 0 en ambos bloques.

Es decir que se almacenarán los siguientes Bytes tanto en el Sector 7, Bloque 0 como en el Bloque 1 del mismo Sector:

0x00 0x00 0x00 0x00 0xFF 0xFF 0xFF 0xFF 0x00 0x00 0x00 0x00 0x1C 0xE3 0x1C 0xE3**.**

2.2.3.- MATRICULA 209

DATOS A GRABAR POR EL FABRICANTE EN LAS TARJETAS DE LA MATRÍCULA 209

SECTOR	BLOQUE	P.INICIAL	P.FINAL	NOMBRE DE CAMPO	DATO
0	0	1	32	NUMERO DE SERIE DE LA TARJETA	Nº DE SERIE DEL CHIP
2	0	5	9	TIPO DE TARJETA	27
2	2	22	48	CÓDIGO DE VIAJERO	Del 90050001 al 90050050
4	0	1	7	EMPRESA QUE FABRICA LA TARJETA	56
4	0	8	13	AÑO DE FABRICACION DE LA TARJETA	17
4	0	14	17	MES DE FABRICACION DE LA TARJETA	10
4	0	18	22	DIA DE FABRICACION DE LA TARJETA	21

El resto de Bloques de datos se suministra con 0 (en el caso de Monedero, con valor 0), **con su correspondiente CRC.**

Los bloques 0 y 1 del Sector 7, son utilizados como Bloques Valor. Por tal motivo, se almacenará el valor 0, utilizando como Byte final el número Absoluto de Bloque Negado y como valor, el 0. No obstante, como el valor del Sector 7, Bloque 1 es copia del Sector 7, Bloque 0, se utilizará únicamente la ubicación del Sector 7, Bloque 0 en ambos bloques.

Es decir que se almacenarán los siguientes Bytes tanto en el Sector 7, Bloque 0 como en el Bloque 1 del mismo Sector:

0x00 0x00 0x00 0x00 0xFF 0xFF 0xFF 0xFF 0x00 0x00 0x00 0x00 0x1C 0xE3 0x1C 0xE3**.**

2.2.4.- MATRICULA 210.

DATOS A GRABAR POR EL FABRICANTE EN LAS TARJETAS DE LA MATRÍCULA 210

SECTOR	BLOQUE	P.INICIAL	P.FINAL	NOMBRE DE CAMPO	DATO
0	0	1	32	NUMERO DE SERIE DE LA TARJETA	Nº DE SERIE DEL CHIP
2	0	5	9	TIPO DE TARJETA	26
4	0	1	7	EMPRESA QUE FABRICA LA TARJETA	57
4	0	8	13	AÑO DE FABRICACION DE LA TARJETA	17
4	0	14	17	MES DE FABRICACION DE LA TARJETA	10
4	0	18	22	DIA DE FABRICACION DE LA TARJETA	21

El resto de bloques de datos se suministra con 0 (en el caso de Monedero, con valor 0), con su correspondiente CRC.

Los bloques 0 y 1 del Sector 7, son utilizados como Bloques Valor. Por tal motivo, se almacenará el valor 0, utilizando como Byte final el número Absoluto de Bloque Negado y como valor, el 0. No obstante, como el valor del Sector 7, Bloque 1 es copia del Sector 7, Bloque 0, se utilizará únicamente la ubicación del Sector 7, Bloque 0 en ambos bloques.

Es decir que se almacenarán los siguientes Bytes tanto en el Sector 7, Bloque 0 como en el Bloque 1 del mismo Sector:

0x00 0x00 0x00 0x00 0xFF 0xFF 0xFF 0xFF 0x00 0x00 0x00 0x00 0x1C 0xE3 0x1C 0xE3.

3.- CALCULO DEL CRC

Todos los bloques de datos, independientemente de su contenido, deben de llevar grabado su CRC.

El CRC previsto es un CRC-8 Obtenido por el siguiente procedimiento:

1. Obtención del CRC-16 CCITT, aplicado a los 15 bytes (120 bits)

primeros del bloque, dando como resultado un CRC de 16 bits. En el Anexo A, se encuentra documentación complementaria para éste cálculo.

2. Obtención de un CRC-8 (8 bits) haciendo un XOR entre los dos bytes del CRC-16, Colocación de este byte del CRC resultante en el último byte del bloque (byte 16).

Puesto que bajo estos mismos requisitos parecen coexistir dos versiones de CRC-16 CCITT, la versión a utilizar es aquella que utiliza el polinomio X16+X12+X5+1.

Para obtener una mayor aclaración sobre este asunto a continuación se adjuntan los documentos siguientes, necesarios para aclarar este particular y definir en detalle este CRC:

- Frame Check Sequences (FCS) in HDLC frame for PPP: Documento que describe el procedimiento de cálculo.
- Documento RFC1662 de donde se extrae el "look up-table" que aparece en el documento anterior.
- Documento ejemplo de código de implementación en C.

Frame Check Sequence (FCS) in High-level Data Link Control (HDLC) frame used for the Point-to-Point Protocol (PPP).

Introduction

There are a 16-bit and a 32-bit Frame Check Sequence methods used in the High-level Link Control frame as used in the Point-to-Point Protocol. The 16-bit FCS is the default, 32-bit FCS can be negotiated between the participating devices. Here, the 16-bit FCS is discussed only, the 32-bit one can be infered from this discussion, however.

> Note that the FCS is sometimes called CRC (Cyclic Redundancy Check) for other applications than HDLC and PPP. They function in the same way, although the generator polynomial may be different.

What makes it somewhat hard to understand the FCS is the fact, that it was originally designed for a hardware implementation. The bits drip along the serial telephone line. To speed up a purely software implementation using bytes instead of bits, a precalculated table is used.

Generator Polynomial

This generator polynomial applies to CCITT X.25 and UIT V.41 as well.

$$x^{16} + x^{12} + x^5 + 1$$

Here is the table for this polynomial copied from RFC 1662, page 19:

	+ 0	+ 1	+ 2	+ 3	¦	+ 4	+ 5	+ 6	+ 7	
00	0000	1189	2312	329b	¦	4624	57ad	6536	74bf	**07**
08	8c48	9dc1	af5a	bed3	¦	ca6c	dbe5	e97e	f8f7	**0f**
10	1081	0108	3393	221a	¦	56a5	472c	75b7	643e	**17**
18	9cc9	8d40	bfdb	ae52	¦	daed	cb64	f9ff	e876	**1f**
20	2102	308b	0210	1399	¦	6726	76af	4434	55bd	**27**
28	ad4a	bcc3	8e58	9fd1	¦	eb6e	fae7	c87c	d9f5	**2f**
30	3183	200a	1291	0318	¦	77a7	662e	54b5	453c	**37**
38	bdcb	ac42	9ed9	8f50	¦	fbef	ea66	d8fd	c974	**3f**
40	4204	538d	6116	709f	¦	0420	15a9	2732	36bb	**47**
48	ce4c	dfc5	ed5e	fcd7	¦	8868	99e1	ab7a	baf3	**4f**
50	5285	430c	7197	601e	¦	14a1	0528	37b3	263a	**57**
58	decd	cf44	fddf	ec56	¦	98e9	8960	bbfb	aa72	**5f**
60	6306	728f	4014	519d	¦	2522	34ab	0630	17b9	**67**
68	ef4e	fec7	cc5c	ddd5	¦	a96a	b8e3	8a78	9bf1	**6f**

70	7387	620e	5095	411c	¦	35a3	242a	16b1	0738	**77**
78	ffcf	ee46	dcdd	cd54	¦	b9eb	a862	9af9	8b70	**7f**
80	8408	9581	a71a	b693	¦	c22c	d3a5	e13e	f0b7	**87**
88	0840	19c9	2b52	3adb	¦	4e64	5fed	6d76	7cff	**8f**
90	9489	8500	b79b	a612	¦	d2ad	c324	f1bf	e036	**97**
98	18c1	0948	3bd3	2a5a	¦	5ee5	4f6c	7df7	6c7e	**9f**
a0	a50a	b483	8618	9791	¦	e32e	f2a7	c03c	d1b5	**a7**
a8	2942	38cb	0a50	1bd9	¦	6f66	7eef	4c74	5dfd	**af**
b0	b58b	a402	9699	8710	¦	f3af	e226	d0bd	c134	**b7**
b8	39c3	284a	1ad1	0b58	¦	7fe7	6e6e	5cf5	4d7c	**bf**
c0	c60c	d785	e51e	f497	¦	8028	91a1	a33a	b2b3	**c7**
c8	4a44	5bcd	6956	78df	¦	0c60	1de9	2f72	3efb	**cf**
d0	d68d	c704	f59f	e416	¦	90a9	8120	b3bb	a232	**d7**
d8	5ac5	4b4c	79d7	685e	¦	1ce1	0d68	3ff3	2e7a	**df**
e0	e70e	f687	c41c	d595	¦	a12a	b0a3	8238	93b1	**e7**
e8	6b46	7acf	4854	59dd	¦	2d62	3ceb	0e70	1ff9	**ef**
f0	f78f	e606	d49d	c514	¦	b1ab	a022	92b9	8330	**f7**
f8	7bc7	6a4e	58d5	495c	¦	3de3	2c6a	1ef1	0f78	**ff**
	+ 0	**+ 1**	**+ 2**	**+ 3**	¦	**+ 4**	**+ 5**	**+ 6**	**+ 7**	

This table is available here as pure ascii-text files (2 KB) for

- 80x86 Assembler: fcs6asm.txt,
- BASIC: fcs16bas.txt, (PowerBASIC dialect)
- C and C++: fcs16c.txt and
- REBOL: fcs16reb.txt.

ready to copy into your source file.

Calculation Loop

In the HDLC frame of the PPP, all data bytes are submitted to the FCS, except the frame start and end bytes (FE hex). The calculated FCS is inserted by the sender just prior the end of frame byte, low byte first, high byte last. The receiver includes the two FCS bytes in the HDLC frame received to the calculation. The final result must yield the «good value» of F0B8 (ef-zero-be-eight).

Initially, the table entry is undetermined. The calculation always starts with the «initial value» of FFFF. The sender submits the data bytes to the FCS prior to encapsulation. Consequently, the receiver must dump the escape byte (FD hex) and substract 20 hex from the encapsulated byte to regain the true data byte which is then submitted to the FCS calculation.

The calculation loop step by step

1. Set «initial table value» to FFFF hex.
2. Get a «data byte».
3. Exclusive-OR «data byte» with low byte of «initial table value», this gives the «pointer» into the table.
4. Read two-byte «table value» pointed to by «pointer» in step 3.
5. Exclusive-OR low byte of «table value» with high byte of «initial table value».
6. Store value formed by high byte of «table value» and low byte of result from the XOR in step 5 as new «initial table value».
7. Go to step 2 and repeat for all data bytes.

Transmitter

8. Form one's complement of final «initial table value».
9. Store low byte of complemented «initial table value» at the end of all «data bytes», then store the high byte.

Receiver

8. Cycle twice longer through the loop to include both FCS bytes.
9. Compare final result to the «good value» of F0B8.

The calculation loop in 80x86 assembler

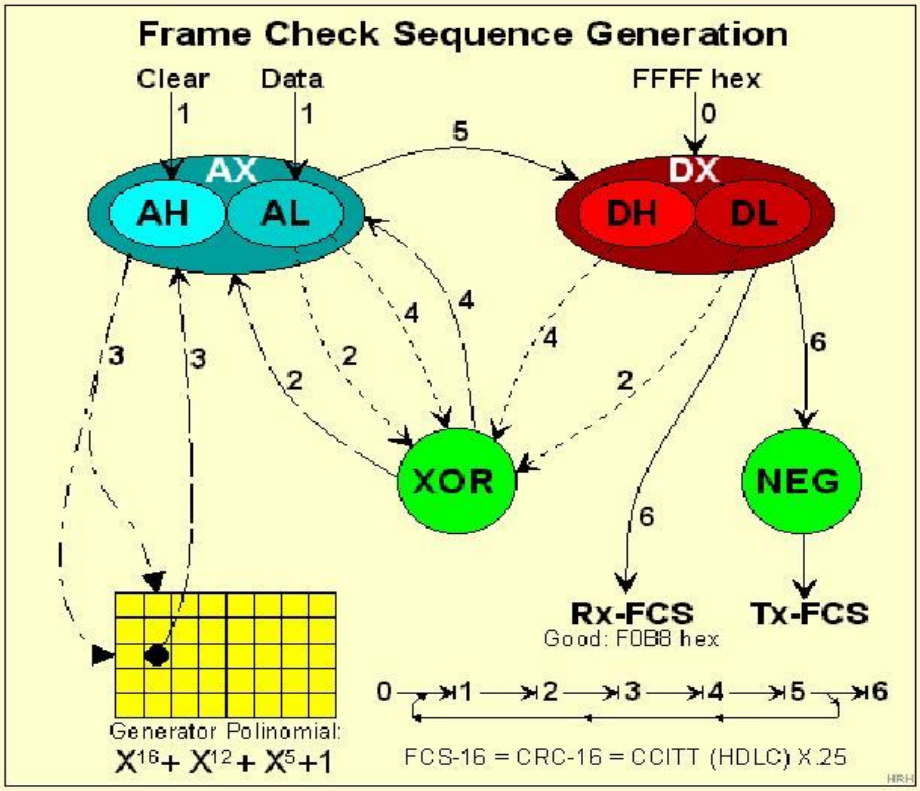
Personally, I find a listing of an assembler fragment the most enlightning piece of documentation.

```
fcs:  mov  ds:si,DataBuf      ;start of data buffer
      mov  cx,BufLen          ;length of buffer
      mov  dx,0ffffh          ;initial table value
;
fcs1: xor  ax,ax              ;clear register
      mov  al,[si]            ;get data byte
      xor  al,dl              ;xor data byte with low byte of
                              ;initial table value
      shl  ax,1               ;pointer x 2 since table entries
                              ;are 2 bytes (16 bits)
      mov  bx,ax              ;save pointer into table
      mov  ax,[DataBuf + bx]  ;read table value (word)
      xor  al,dh              ;xor low byte of table entry with
                              ;high byte of initial table value
      mov  dx,ax              ;store result as new initial table
                              ;value
      inc  si                 ;point to next byte in DataBuf
      dec  cx                 ;decrement length counter
      jnz  fcs1               ;repeat until all bytes done (cx=0)
;
;
xmtr: xor  dx,0ffffh          ;one's complement
      mov  [si],dl            ;store low byte of result after last
                              ;data byte
      inc  si                 ;point to next byte in DataBuf
      mov [si],dh             ;store high byte of result
;
;
rcvr: mov  ax,0f0b8h          ;load good value into register
      cmp  ax,dx              ;compare against FCS
```

```
        jnz   error                ;if no match, error handling
;
```

The calculation loop graphically

Here is an uncommon representation of the FCS or any CRC calculation loop. The numbers do not follow those of «The calculation loop step by step» above.



Example HDLC Frame for PPP

Here is a HDLC frame I captured. I have two computers, each connected to a modem, the modems are connected to my small in-house telephone exchange. Computer 1 running Windows called computer 2 with the RAS service (PPP) just as if it would call up an ISP (Internet Service provider). The modem of computer 2 answered the call, on the computer run a small packet capture program I wrote. The initiating computer sent 10 PPP packets until it gave up with the message, that it would get no answer. Of course, I had not programmed computer 2 to answer the PPP request. I could analyze the captured PPP HDLC frame.

ESC	Enc.	Data	Remarks
		7e	Start of HDLC frame
		ff	Address field: address all

7d	23	**03**	Escaped encapsuled control byte
		c0	Start of LCP (Link Control Packet)
		21	
7d	21	**01**	Type 1 = Configuration request
7d	21	**01**	Packet ID = 1 (1st packet sent)
7d	20	**00**	Length of packet = 0017 hex = 23 including Type. Note, high byte preceeds low byte
7d	37	**17**	
7d	22	**02**	Modem Control Characters (ACCM) follow Length = 6 including ACCM
7d	26	**06**	
7d	20	**00**	
7d	2a	**0a**	
7d	20	**00**	
7d	20	**00**	
7d	25	**05**	Magic Number (refer to RFC 1662) Length = 6 including Magic Number
7d	26	**06**	
7d	20	**00**	
		2a	
		2b	
		78	
7d	27	**07**	Address & Control Field Compression = ON Length = 2 including ACF
7d	22	**02**	
7d	28	**08**	Protocol Field Compression = ON (for IP) Length = 2 including PFC
7d	22	**02**	
7d	2d	**0d**	Data (?)
7d	23	**03**	Data (?)
7d	26	**06**	Data (?)
		a5	Frame Check Sequence low byte high byte
		f8	
		7e	End of HDLC frame
ESC	**Enc.**	**Data**	**Remarks**
Enc. = Encapsuled data byte (+ 20 hex)			

Consider all data bytes (bold) in the white fields to determine the FCS for sending a frame. When analyzing received frame, take all data bytes in the white and red fields. This must result in f0b8 hex for a uncorrupted frame.

Examples for Calculation

Understanding and calculating the checksums is a bit tricky and not

always really obvious. However, if you follow «The calculation loop step by step», you can calculate a small frame by hand, using a hex-calculator. I have done so for the «Example HDLC Frame for PPP» and for an example from a correspondent, who did not quite got it right. If you are not yet enlightened, perhaps you try to follow the examples yourself.

a. Example HDLC Frame for PPP Example 1 (PDF, 1 page, 17 KB).

b. Some small Example 2 (PDF, 1 page, 14 KB).

References

i. RFC 1661 and RFC 1662 «The Point-to-Point Protocol»; *W. Simpson*, Editor.
(Nothing works without THE reference.)

ii. «A painless guide to CRC error detection algorithms»; *Ross N. Williams*
www.geocities.com/SiliconValley/Pines/8659/crc.htm.
(Very painfull and incomprehensible for me.)

iii. CMCRC10 assembler listing; *Celso Minnitti, Jr.*
(Extremly helpfull to me.)

iv. «File Verification using CRC»; *Mark R. Nelson* in Dr. Dobb's Journal, May 1992.
(As usual, the DDJ got it right.)

4.- FICHERO DE FABRICACIÓN

Las muestras que se presenten para cada matrícula, deben de ir acompañadas del correspondiente fichero de fabricación, que será un fichero Excel.xls, grabado en un CD, que contendrá una fila por cada tarjeta de la muestra y de acuerdo con el formato que se recoge en las siguientes tablas:

4.1.- MATRICULA 207

Nº CHIP	Nº CLIENTE	Nº MATRICULA	TIPO DE TARJETA	FECHA DE FABRICACION	FABRICANTE
XXXXXXXX	XXXXXXXXXXX	207	30/31	17-10-21	54

4.2.- MATRICULA 208

Nº CHIP	Nº CLIENTE	Nº MATRICULA	TIPO DE TARJETA	FECHA DE FABRICACION	FABRICANTE
XXXXXXXX	XXXXXXXXXXX	208	28/31	17-10-21	55

4.3.- MATRICULA 209

Nº CHIP	Nº CLIENTE	Nº MATRICULA	TIPO DE TARJETA	FECHA DE FABRICACION	FABRICANTE
XXXXXXXX	XXXXXXXXXX	209	27	17-10-21	56

4.4.- MATRICULA 210

Nº CHIP	Nº MATRICULA	TIPO DE TARJETA	FECHA DE FABRICACION	FABRICANTE
XXXXXXXX	210	26	17-10-21	57

Mª Victoria Andrade Blanco
Gerente de Instalaciones