ACS Protocol Specification

As used in the

Genesis Project

Version 1.0
Rev. a

Table of contents

1 Introduction

This document describes the protocol used for the Acustical Communication System (from now ACS) used for communication between LEGO Genesis action figures and other medias.

Chapter 2 tells about the special requirements for communication via sound waves: I.e. the lack of handshake, the need for some sort of Forward Error Correction (from now FEC) etc.

Chapter 3 is describing the low-level part of the protocol i.e. the carrier.

Chapter 4 is describing the low-level part of the protocol including the different methods to secure a safe transmission of the messages i.e the use of Hamming encoding, Modulo-256 checksum, retransmission etc. – all seen from the transmitter (e.g. the Television Channel).

Chapter 5 describes the different modes of transmissions in the protocol: The difference between compact, normal and the extended protocol, timecritical transmissions and also the difference between retransmitted commands and new commands. And how a combination of all these methods should be used to ensure a safe transmission.

Chapter 6 lists all the different methods used in the receiver part of the transmission chain. And how to combine all these methods to predict messages, which contains some failed information/distortion.

2 General description of ACS communication

2.1 Missing handshake

The ACS is a SIMPLEX system (I.e. no validation or software-handshake can be transferred from the receiver to the transmitter. Hence special checks and tricks are going to be used, when the data transmitted is of the “must be received”-type.

Communication via a noisy and not ideal environment normally involves some sort of handshake (i.e. the receiver is able to say: “Please repeat again – I did not understand”. But when transmitting the communicated data elements via sound waves and e.g. through a normal television set, there is no way, the receiver part can interact with the transmitter.

I.e. the transmitting part cannot determine if the receiver part has missed the total message or only received it partly. Some messages are essential for the Genesis concept – e.g. the figure should be triggered with the correct piece of speech and the beginning of this speech should be synchronous with the play on the television or computer screen.

Each package also includes a checksum and some error-correcting algorithms.

The transmitter is also responsible for the correct synchronization of the receiver. The receiver can erroneous detect payload as preamble etc. So the only way the transmitter can be sure the receiver is in a ready mode (synchronized waiting for a new “start of transmission”) is waiting an idle-time with no transmission. This idle-time should be calculated equal to the worst case package time as a maximum.

The size of a given transmission is encoded in the “low-level protocol” and used in both the driver- and the protocol part of the driver/controller stack. The size is adjusted at the low-level, so only the payload bytes and size are made available for the protocol- and controller layer.

- To validate the transmission without any feedback, the single packages are retransmitted a number of times with respect to their importance.
  Retransmission is one of the simplest methods to raise the guaranteed reception quality.
  Packages are sent with an incremental package counter, so the receiver part can determine when new information is available.

- Time-critical parts will be transmitted with decreasing time-stamps. When the signal is understood, the following signals with decreasing time to trigger will automatically be rejected in the Genesis-figure. The Genesis figure will do the final timing internal. However the Genesis figure can use the automatically rejected time-stamps as a continuous check.

- Hamming encoding of the data-bytes is another method (FEC) to raise the “noise-immunity”.

- A modulo-256 (or like) checksum helps validating the transmission.

- At the receiver end all the abovementioned methods can help predicting missing or corrupted data elements.

The Genesis protocol is using a mixture of all the bullet points mentioned above. More exotic FEC will add too much overhead to the current needs (the command size needed is small - download of data-blocks is not a must). However, the protocol will include switches for indicating extended protocol and - FEC.

2.2 Transmission via audible sound waves

The use of audible frequencies for transmitting data in speech and/or music should not reduce the original sound quality (or the impact should be made as small as possible).

- One way to eliminate the distortion of the original sound material is to use data encoding at a sound pressure level as low as possible.
- Using a small bandwidth will also reduce this impact. In this product all the data pulses (sound-burst) are feed through a BlackMann filter algorithm before transmission (i.e. encoded in the signal). This reduced bandwidth of the pulses reduces the needed width of the notch filter used to remove the data-carrier frequencies in the original sound material.
- Using the knowledge of Psycho Acoustic Masking effect is also a great way of reducing the impact heard by the user. (For further details see the document: LAU.doc)
- The transmitted pulse train should also be taken in account. The pulses will form a low-frequency envelope curve in the low end of the audible spectrum.
- Framming error due to signal distortion should also be eliminated. However using start- and stop in every byte as used in RS-232 will lower the total bandwidth. This will normally not effect the transmission rate for small “trigger” signal, but for an eventually extended protocol transfer (i.e. transmission of new function-macros or images), this will lower the total bandwidth by 20%.

3 Carrier (i.e. the transmission signal)

3.1 The BlackMann-filtered signal for a single bit

The single bit in the Genesis Protocol is formed by 12 sinusoidal cycles at a frequency*) of 9000 [Hz]. In a PC based test setup a frequency of 8820Hz. will be used (due to the fixed 44.1 KHz. Sampling rate of the normal Wave sound reproduction).

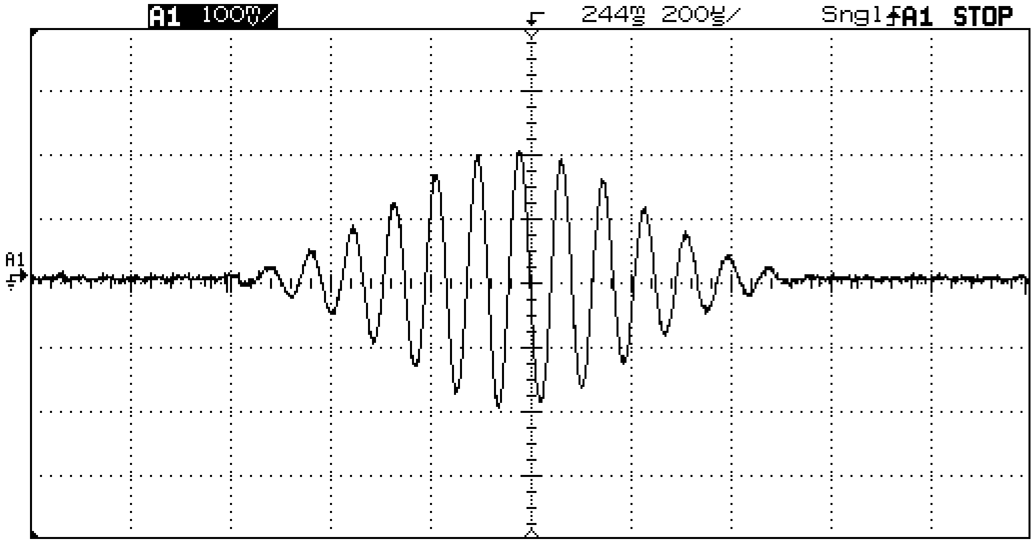
*) This frequency depends of the actual sample rate of the transmitting device. The table values should be able to form a sinusoidal signal starting and ending in zero.

These 12 cycles is BlackMann-filtered to reduce the total bandwidth and hence the impact on the original sound material (the removed part from the original sound should be held as small as possible). The time for one bit is 1.361 [mS] giving a total maximum BAUD rate of 735 bits pr. Sec.

The BAUD rate can be increased by reducing the count of cycles in a single bit-burst, but then the detection at the receiver end will be less reliable. Especially the AGC will have less time to control the amplitude and hence the trigger-level of the rising and falling edge of the envelope (the period of the single bit).

An increase in carrier-frequency can also raise the BAUD-rate, but then the different transmitters (transmission chains and especially the cheap television-sets) will be less reliable.

Oscilloscope screen showing one single bit of information:

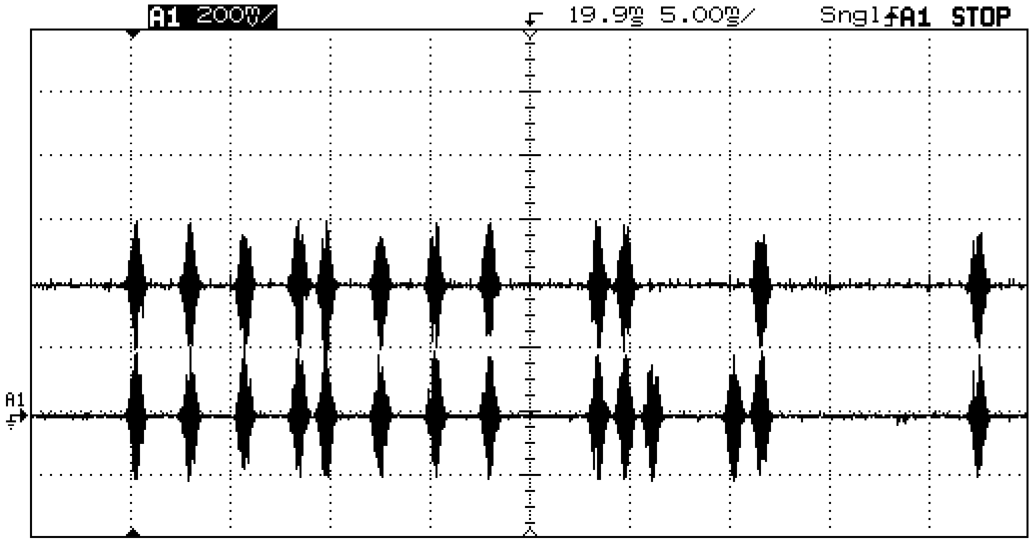


3.2 Byte/Nibble encoding

The upper trace below is showing the transmitted value of 0x55, 0xAA, 0x30, 0x80, 0x01. The lower trace shows the value of 0x55, 0xAA, 0x39, 0x80, 0x01.

REMARK: Bytes/Nibbles are send as MSB first (I.e. Bit 7, bit 6 ……… bit 0).

Oscilloscope screen showing two different transmissions of byte encoded information:



4 The Genesis Protocol

4.1 High- and Low level part of the protocol

The Genesis protocol is divided into a low-level and high-level part. The low-level part is used in the driver-layer of the Transmitter and in the low-level driver part of the firmware. The high-level part of the protocol is used at the protocol-layer of the software.

Dividing the protocol this way ensures a layered architecture and the communication can be changed at different level without disturbing the other layers. E.g. the low-level part can be changed to other types of pre-amble, error-correction etc. without any changes necessary at a higher layer of the protocol.

Some part of the transmitted information is used both at the high- and low end of the protocol stack.

4.2 General command encoding

The Genesis protocol uses a mixture of Nibble and Byte encoding. The command-set itself uses Byte encoding, so future extensions can be done. The pre-amble, the added Hamming bits, the synchronization and the re-synchronization uses Nibble encoding.
All one (‘1’) encoding are transmitted as bursts of Blackmann filtered carrier frequencies; zeros (‘0’) are encoded as space (no signal).

The nibbles and bytes are transmitted with the most significant bit first (bytes starting with bit 7, bit6……. bit 0 and nibbles starting with bit 3, bit 2… bit 0).

The layout of a transmitted package:

P1	P2	S	TTT SIZE	DCEE NNNN	H	PL	H	PL	H	……	PL	H	Σ
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

═════ Part of the very low-level of the protocol (used as pre-ample and initial syncronization)

_____ Used (or can be used) at both the low- and high-level part of the protocol (i.e. used for data validation and also used for mode etc.)

_____ User-level data. I.e. the data sent to the application.

Px:	Pre-amble

- 1’st pre-amble nibble
- 2’nd –

S:	Synchronization

- Sync character	0b0010	The 1->0 transition used for suncronization at the receiver part.

See also the resyncronization (RS).

TTT:	Protocol Type

- RC platform encoding	**0**??	“Compact ACS”	(See the documentation of the RC platform for further information)
- Extended	**1**00	Serengeti
  101	Genesis
  110
  111	“Escape” future use

SIZE: Package Size

- A package can be from 1 nibble in size (depending on the used protocol encoding) and up to 31 nibbles.

D: Delayed Trigger (Timed transmission):

- 0 Normal transmission, direct command (and downloaded).
- 1 Timed transmission. I.e. a command which should be activated at a later time in the figure (a timed triggering). All commands (except commands relating downloads etc.) can be send with this bit set. I.e. they will have another added byte holding the time-delay until triggering: 1 – 255 sec. (1 sec. – 4 min. 15 sec.)

C: Package type

- Command 0 Normally set to 0 (zero).
- Continued block 1 Data is sent as bulk. I.e. The package does not include any command. Data is collected as bulk and a more relaxed error correction could be used. Data will be send a number of times, and the receiving part could use the repeated data for restoring the original signal.

E: Error Check and Correction

- 00 Modulo-256 checksum only
- 01 Hamming encoding and modulo-256 checksum
- 10 CRC-16 checksum only
- 11 Hamming encoding and CRC-16 checksum

N: Package counter

- Modulo-7 package counter (retransmission check).

Σ: Modulo-256 checksum

- Modulo-256 sum of the Mode byte, all Payload-bytes and all the Hamming code nibbles.

4.3 Pre-amble

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	∑
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

Preamble nibbles:

P1: 0xA

P2: 0xA

The pre-amble is used as a kind of wake-up character. To have a known signal with a known duty-cycle, the 2 pre-amble nibbles (the preamble-“byte”) are encoded as 0xAA. This forms a binary string: 1010101010. This encoding ensures a solid ‘1’ as the first piece of data (i.e. a fixed trigger) and the repeated ‘10’ are used to calculate the real period time and duty-cycle. The length of 2 nibbles is selected so hardware, e.g. AGC, PLL or another H/W-circuitry can settle etc. before the real synchronization.

The encoded BAUD rate can be changed due to the way producers of movies and TV-productions adjust the time of a transmission. I.e. if a movie takes 26 minutes to transmit, the producer can adjust this time to e.g. 25 minutes. This will result in a change in BAUD rate (pulse-width).

The receiver part used the known bit-pattern of the 0xAA to calculate the real pulsewidth. This pulsewidth is checked against the original pulsewidth and the default pulsewidth is adjusted to match the transmitted (and frequency-shifted) signal.

4.4 Synchronization

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	Σ
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

Sync. Char:

S: 0x2 (0010_B)

The Genesis protocol uses no start- or stop bit, so a 100% reliable synchronization is required, if framing errors shall be held at an absolute minimum. The pulses are also degraded due to foreign audible noise, variations in SPL (Sound Pressure Level), time compression/decompression or echo.

The Genesis protocol uses single nibbles for synchronization. A well-defined signal, i.e. a binary string of 0010, is used. The ‘1 to 0’ is used as the trigger for the timing of the following bytes (payload, forward error correction code and checksum).
HER_PER
Sampling of data bits is started 1½ bit time after receiving the rising edge of the ‘1’ (this time is either the default or the pulse width calculated from the pre-amble). All bits of the following payload will then be sampled in the middle of the bits.

Packages with payloads greater than 3 bytes (including the Mode byte) will add one or more additional synchronization nibble(s) to the package.

4.5 Re-Synchronization

Layout of a re-synchronized package (> 6 nibbles of payload/Mode):

	SIZE	MODE	H	PL(1)	H	PL(2)	H	**RS**		PL(N)	H	∑
	>11	n1,n2	n3	n4,n5	n6	n7,n8	n9	**n10**		nN,nN+1	nN+2	nN+3,nN+4

Layout of a re-synchronized package (> 21 nibbles of payload/Mode including 1'st resync):

	SIZE	MODE	H	PL(1)		H	**RS**		H	**RS**		PL(N)	H	∑
	>21	n1,n2	n3	n4,n5		n9	**n10**		n19	**n20**		nN,nN+1	nN+2	nN+3,nN+4

The ReSynchronization nibble (RS) is encode as 0001 binary. I.e. giving a solid zero to one transition for easy and safe detection in the software based receiver (pulse-dector).

Payload are divided in 1 byte (2 nibbles) and a following FEC nibble (Hamming encoding).

The use of ReSynchronization can be determined by the following check/calculation:

```
IF (SIZE  > 21) THEN

        Send the nibbles n1 to n9
        Resync (n10)
        Send the nibbles 11 to 19
        Resync (n20)
        Send the nibbles 21 to xx
        Send the Checksum

IF Size (PS) > 11 then

        Send the nibbles n1 to n9
        Resync (n10)
        Send the nibbles 11 to xx
        Send the Checksum
```

4.6 Protocol Type (encoding)

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	Σ
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

Protocol Type:

TTT:	000	Simple	RC platform encoding	“Compact ACS”
	001	-	Can not be used	
	010	-	Can not be used	
	011	-	Can not be used	
	100	Extended	Used “Serengeti”	The Serengeti IR-protocol
	101	-	ACS used in “Genesis”	
	110	-	Free - not used	
	111	-	“Escape” future use	Encoding type in the payload

The ACS system can use the same way of encoding as the RC-protocols (short and extended ones). So the ACS can be used as a transparent link for relaying other kinds of signals. By using different protocols (be able to transfer different protocols) the decoding and encoding software can be held at a minimum in the transmitter and receiving firmware. It is only the hardware near layer of the software and the only part who needs to be changed, if ACS is used for relaying other protocol-formats. This should ensure an easy implementation of ACS in other devices.

Simple: The protocol type used in the small RC platform. The protocol is fixed and limited to the commands described in the document: “Protocol LEGO RC platform 16. January 2001”.

Extended: The protocol is addressed by 2 additional bits. The current protocol has these bits hard-coded as a “101” (one-zero-one) i.e. the normal Genesis protocol described in this document. The other extended protocols will not be used in the current Genesis figure. The extended protocols should be seen as a sort of “safety for the future”.

The TTT flag-bits are used in the “low-level end” of the protocol and lets future ACS applications and –devices use a mixture of both simple and extended command sets. This kind of “protocol-escape” lets also old and future ACS devices be used side by side without any interfering between the devices.

4.7 Package Size

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	∑
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

Package Size:

PS: 0-31

The Package size refers to units of Nibbles. I.e. the Mode byte, the Checksum byte and all Payload bytes counts as 2. The Hamming Encoding nibbles and resynchronization nibbles are also included in this count and counts as 1.

The package size is reduced with the size of the Mode byte, the Hamming nibbles, resynchronization nibble(s) if present and the size of the checksum when stored in the receiver buffer (i.e. when made available for the user-level).

4.8 “Future” Escape-flag

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	∑
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

“Delayed triggering”-flag:

D:	0	Normal transmission.
	1	Delayed triggered transmission. I.e. HER_PER.

When set the protocol and pay-load will be decoded using bits and bytes following the Mode-byte.

4.9 Package type

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	Σ
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

Package type: (low-level)

• Command	0
• Continued block (BULK)	1

Transmissions via the Genesis protocol can be either short commands or “BULK”-transmissions. Commands are very important and should be decoded and error-corrected with care. Bulk transmissions e.g. icon-data are less error-sensitive because they can be transmitted over and over again.

The blocks can be tested against previous transmitted data until the package and/or the total checksum is compared OK.

Block numbering and/or interleave encoding is part of the pay-load.

4.10 Checksum type

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	Σ
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

Checksum type: (low-level)

EE

- **0**0 **Modulo-256** checksum only
- **0**1 Hamming encoding and **modulo-256** checksum
- **1**0 **CRC-16** checksum only
- **1**1 Hamming encoding and **CRC-16** checksum

When this bit is reset (“0”) a simple modulo-256 checksum is used. This checksum uses a summarization of all the payload nibbles within one byte. I.e. summarization in one byte without any carry. The single elements is summed as they are encoded: bytes summed as a byte, nibbles (i.e. the Hamming encoded bits) are summed as a byte with the nibble information.

Σ = (The byte of modeflags[T..N] + H1 + PL2 + H2…………PLn + Hn) MOD 256

4.11 Hamming encoding (on/off)

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	Σ
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

Hamming encoding on/off: (low-level)

E**E**

- 0**0** Modulo-256 checksum only
- 0**1** **Hamming encoding** and modulo-256 checksum
- 1**0** CRC-16 checksum only
- 1**1** **Hamming encoding** and CRC-16 checksum

When this bit is set to 1 each byte is followed by a 4 bit Hamming encoded parity nibble. This bit is used together with the checksum bit. Remember Hamming encoding can only flag uneven bit errors (1, 3, 5 or 7 bits wrong) and repair 1 bit error. Other error-counts will be detected using the Modulo-256 checksum.

4.12 Retransmission counter (package counter)

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	Σ
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

Package counter: (low-level)

NNNN:	1 - 15	Increasing package number (held constant if retransmitted)

The retransmission counter is used as a simplex handshake. I.e. the transmitter is incrementing this number, when a new command transmission is started. Retransmissions will keep this number constant, so the receiver part will be aware of the difference between retransmissions and new commands. A package number set to “0000” (all zeros) has a special function: It is the command for resetting and synchronization the package counter compare register in the Genesis figure.

It is added for situations, where a Genesis figure used for both television and PC-game reception. The figure will reject packages sent by another media if the packages starts at a package counter value equal to the package counter of the last transmitting media.

4.13 Hamming encoding

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H	……	PL	H	Σ
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?	……	0x??	0x?	0x??

The Genesis Protocol uses Hamming encoding with an encoding table. Encoding byte by byte is selected as the most efficient use of the bandwidth versus safety. Furthermore the selected encoding and decoding is very easy to implement in microprocessors low-level assembly code.

Using row and column parity can also detect parity errors but sending 1 to 8 bytes add at least 2 bytes to the package. But only 1 bit error can be detected. Using the byte-wise and table-encoded Hamming encoding 1 byte adds 4 bit and 8 bytes adds 4 bytes, but one should remember, when sending 8 bytes all parity-checked by Hamming encoding, one bit-error per byte can be detected and corrected (up to 8 bit-errors in total).

Encoding table:

Bit:	
8:	1011 1011
9:	1101 1001
10:	1110 1100
11:	1111 0110

The byte to encode is AND’ed row by row with the entries [0..11] in the encoding table. Each result is checked for parity. The parity of each result is the code-word, MSB first.

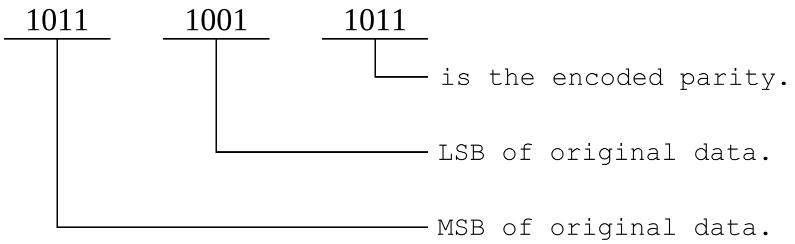
Example of encoding:

Byte to transmit: 0xB9 1011 1001

Encoding table	**Byte to encode**	**Result**	**Parity**

1011 1011	&	1011 1001	=	1011 1001	=>	1
1101 1001	&	1011 1001	=	1001 1001	=>	0
1110 1100	&	1011 1001	=	1010 1000	=>	1
1111 0110	&	1011 1001	=	1011 0000	=>	1

The resulting code-word is: 1011 1001 1011
Where:



Decoding table:

1011 1011 1000
1101 1001 0100
1110 1100 0010
1111 0110 0001

The code-word is AND'ed with all 4 rows in the decoding table. The 4 results are then checked for parity. The parity bits of the results are used to point to bit-errors. A result of 0000 flags no error.

Example of decoding a OK received data byte:

Byte received: 1011 1001 1011

Decoding table		**Byte to encode**		**Result**	**Parity:**
1011 1011 1000	&	1011 1001 1011	=	1011 1001 1000	**0**
1101 1001 0100	&	1011 1001 1011	=	1001 1001 0000	**0**
1110 1100 0010	&	1011 1001 1011	=	1010 1000 0010	**0**
1111 0110 0001	&	1011 1001 1011	=	1011 0000 0001	**0**

Parity = 0000 so the byte is received OK!

Example of decoding a FAILED data byte:

Byte received [FAIL]: 1011 1**1**01 1011 (should be 1011 1**0**01 1011)

Encoding table		**Byte to encode**		**Result**	**Parity:**
1011 1011 1000	&	1011 1101 1011	=	1011 1001 1000	**0**
1101 1001 0100	&	1011 1101 1011	=	1001 1001 0000	**0**
1110 1100 0010	&	1011 1101 1011	=	1010 1100 0010	**1**
1111 0110 0001	&	1011 1101 1011	=	1011 0100 0001	**1**

Byte received FAIL, the 0011 points to bit-column with incorrect polarity. If this information is used, the original byte can be reproduced (Remember – only <u>one</u> bit per byte can be fixed this way!)

Decoding table	Column pointer
1011 1**0**11 1000	**0**
1101 1**0**01 0100	**0**
1110 1**1**00 0010	**1**
1111 0**1**10 0001	**1**

Received byte :	1011 1101
Bit-pointer:	0000 0100
Corrected byte:	1011 1001

Which matches the transmitted byte!

4.14 Payload

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H		……	PL	H	∑
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?		……	0x??	0x?	0x??

PL: 0x??

Payload is encoded in 8 bit data bytes. I.e. there is no start or stop bit. Both commands and parameters are part of the payload.

4.15 Modulo-256 checksum

P1	P2	S	TTT SIZE	DCEENNNN	H	PL	H	PL	H		……	PL	H	∑
0xA	0xA	0x2	101 0-31	0001 0000	0x?	0x??	0x?	0x??	0x?		……	0x??	0x?	0x??

Σ: The Modulo-256 sum of all the bytes and nibbles (P1, P2, S and the Type and Size byte are not included).

Σ = (The byte of modeflags[F..N] + H1 + PL2 + H2…………PLn + Hn) MOD 256

The modulo-256 checksum is used to detect errors not seen or corrected by the Hamming encoding. It is important to notice, that a modulo-256 checksum cannot be used for synchronization. Retransmission of modulo-256 blocks without any change i.e. no change in length, retransmission counter or contents, will always show an ok modulo-256 checksum (last part + checksum + first part instead of total part + checksum).

5 Transmission enhancement and safety

There is different “handles” the programmer of the application software can use to ensure a safe and stable communication. Remember there is no hardware or software handshake (simplex), so the only way to ensure a correct transmission, is to prevent the signal from getting interrupted and/or distorted.

5.1 Protocol included signal error correction and detection

The producer of the application software can enable or change some of the protocol specific error detection and error correction methods. The protocol will always use a checksum calculation but this calculation is changeable between Modulo-256 and CRC-16.

The forward error correction (FEC) is also user selectable.

5.1.1 Hamming encoding

To help validating the transmitted data, the use of Hamming encoding (FEC) should always be used. The reduced bandwidth is regained as the need for retransmissions is decreasing when a Hamming encoding is used.

The Hamming Encoding will help in an environment where noise is of the “pulse”-type. For long term noise the Hamming Encoding does not help. The maximum error correction is one bit per transmitted payload byte.

5.1.2 Checksum calculation

Checksum calculation is always used, but the user can select either Modulo-256 or CRC-16. The checksum is used for detecting 2 or more error bits in one or more of the transmitted bytes.

5.2 User methods for improving the signal reliability

5.2.1 Retransmission of packages

The most important way of keeping a reliable transmission is the use of retransmissions. The producer can set the retransmission rate low or high corresponding to the importance of the packages. The rate is automatically transmitted to the receiver (i.e. the retransmit-counter shows the actual package no.)

Short and less important commands (e.g. small phrases) are sent with a low (or no) retransmission rate. Large and more important packages are sent with a retransmission rate depending on the calculated time and the importance (remember the time using a filtered masking sound should be as short as possible)

5.2.2 Timed transmissions

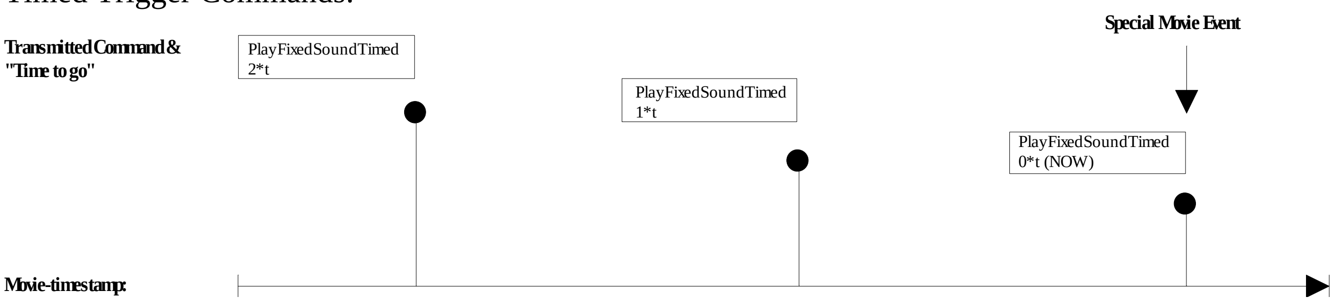
For the most important transmissions timed transmissions should be used (e.g. spoken phrases in synchronization with an on-screen event).

The time between the single transmissions and the number of transmissions should be calculated with respect to the importance of the command and the available time slice.

Timed transmissions will never be retransmitted. I.e. they will always be transmitted with a new and incremented package number.

If transmission of timed commands is done with a slow repetition rate, the command is less sensitive to environmental noise.

Timed Trigger Commands:



If the first transmitted “timer stamp” is received correctly, the following “time stamps” are used to validate the command. Timing will be done internally in the Genesis figure if no “KillTimedTriggers” is received.

6 Receiver methods for safe reception

First of all, the signal feed to the firmware should be stable in volume and timing. The volume can be held stable, if a digital interface is chosen, but then the timing still can be a problem (i.e.. the trigger-levels used for the digitizing of the incoming audio-frequencies).

A software only solution can also be used, if the used DSP and/or <u>fast</u> microprocessor have MIPS for make the frequency detection and filtering in software.

6.1 Preamble Identify

The pre-amble is used for signal detection, system front-end settling etc. (wake-up for data).

6.2 Frequency Detection

The pulse repetition rate is used for determine the correct pulsewidth (i.e. compensate for the time compression[1)] or decomression[1)] added by the producer).

Detecting more pulses in the pre-amble will increase the accuracy of the measured pulse-width. Both the positive and the negative pulses are measured.

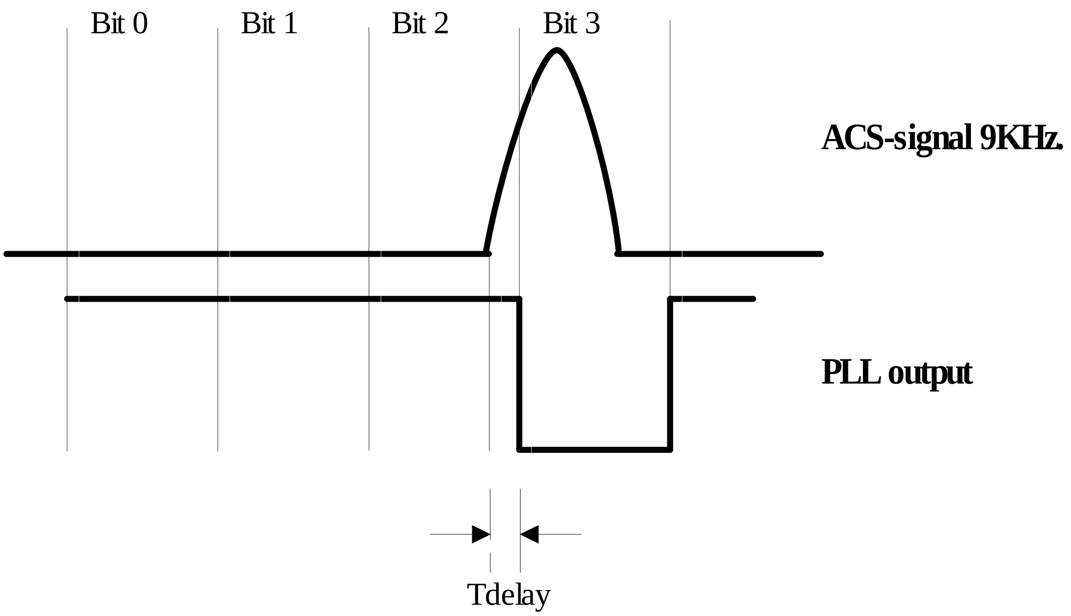
If 2 pulse-pairs (2 positive and 2 negative) are measured only the last pair is used (stable data).

If more than 2 pulse-pairs (>2 positive and >2 negative) are measured, the shortest and maybe also widest pulse pair should be removed.

[1)] When movies are transmitted via television, the producer can compress the movie concerning the total transmission time (everything happens in a shorter time – shorter pulses). However the software should also be able to compensate for increased transmission time (Longer Pulse-Width).

6.3 Syncronization

To allow the receiver part to be synchronized a known and well defined “sync-nibble” is used. The binary-value 0001 will result in a well defined negative transition from the Analog circuitry e.g. a PLL-circuitry (Phase-Locked_Loop).

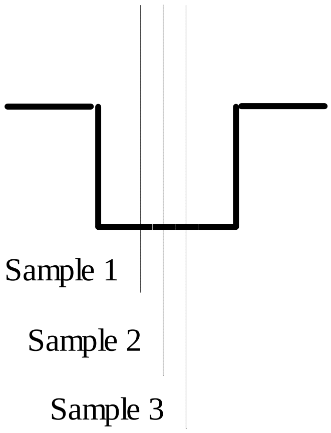


T_{delay} is the delay time from 9 KHz. Presented on input of PLL until valid output. The negative pulse of the PLL is used as the overall trigger for the internal timer in the software UART.

This delay time is also depending on the actual burst amplitude, so a stable AGC (Automatic Gain Control) should be added before the PLL circuitry.

6.4 Oversampling

In order to remove noise (“spikes”) in the signal the UART should use a 3 times “oversampling” I.e. make 3 samples with some time in between:



The firmware then calculates the real value using the following pseudo-code:

```
If (HighSamples >= 2) Then
        BitValue = LogicOne
Else
        BitValue = LogicZero
```

This way of sampling should minimize sampling of spike-errors.

6.5 Hamming Error Correction

Even though the Hamming Encoding will correct some noisy bytes (with single spikes), the firmware should also use the Hamming Encoding correction statistic to tell the user about the signal-quality.

6.6 Checksum Calculation

The Hamming Encoding can only regain 1 (one) bit per encoded byte. But it will flag all sizes of uneven errors (1, 3, ..). The checksum is used for checking errors larger than 1 bit and/or errors in the Hamming bits.

6.7 Retransmission

The FirmWare can use the forthcoming retransmissions to collect a valid data-package. If the checksum is equal in two retransmissions, and the Hamming Encoding flag errors in different nibbles, then all the other nibbles and bytes can be used to form a valid package. This feature should be selectable.

```
IF (LAST_VALID_NO = = CURRENT_RX_NO) THEN
  REJECT CURRENT_RX_DATA                                   //Already rec. OK

IF ((LAST_DATA & CURRENT_RX_DATA) = = VALID_DATA) THEN     //Can LAST &
    SET VALID_DATA_FLAG;                                   // CURRENT
                                                           //be combined as a
                                                           //valid package?
LAST_DATA = CURRENT_RX_DATA;
```

6.8 Statistic

By using all the different error-flags and retransmission rates, the firmware can calculate a number representing the actual signal-quality. This value can be used to make some guidance for the user. E.g. “I can not hear”, “Move closer”, “What’s that noise!?” etc.

Everything depending on the actual kind of noise:

- A lot of Hamming Corrections shows a signal full of spikes. I.e. there is a lot of foreign noise in the input signal.

- Many retransmissions with checksum errors show a weak signal. Too far distance, too low a setting of the volume control on the transmitter or a foreign continued sound at frequencies near the carrier.

Document revision List

24-11-2000 Document initiated.

08-12-2000 Name changed from Genesis Protocol Design to Genesis Protocol Specification

11-12-2000 References to 12 KHz. And/or 11.636 Hz. Changed to 9 KHz.

18-12-2000 Package size added. Protocol layer ref. Added to main protocol drawing.

20-12-2000 Document finished as draft

25-01-2001 Some small changes added: No retransmission of timed packages. Low-and High level references of driver, controller and protocol changed/fixed.

13-06-2001 Protocol specific drawings/tables changed. Frequency for Genesis changed to 8820 Hz. Version set to 1.0 rev. a