Organisation européenne de télécommunications par satellite
European Telecommunications Satellite Organization
Tour Maine-Montparnasse 33, av. du Maine - 75755 PARIS Cedex 15 - France

Digital Satellite Equipment Control (DiSEqC)

BUS FUNCTIONAL SPECIFICATION

Version 4.0

March 22, 1996

Reference Documents:

DiSEqC Slave Microcontroller Specification Version 0.2 (March 22, 1996)

Associated Documents:

DiSEqC Update and Recommendations for Implementation (March 22, 1996)
Application Note for LNBs and Switchers (to be issued)
Application Note for Receievers (to be issued)

1. Introduction

This document describes the Functional Specification for the Digital Satellite Equipment Control (DiSEqC) Bus. DiSEqC is an OPEN STANDARD with additions controlled by industry agreement.

The DiSEqC system is a communication bus between satellite receivers and satellite peripheral equipment, using only the existing coaxial cable. DiSEqC can be integrated into consumer satellite installations to replace all conventional analogue (voltage, tone or pulse width) switching and all other control wiring. A "Universal" Slave IC will support multiple applications by link-configuration to identify the peripheral hardware that it is controlling.

The main advantages of DiSEqC are:

- standardised digital system with non-proprietary commands
- enables switching in multi-satellite installations (e.g. European 13°E/19.2°E combi installations)
- backwards compatible with 13/17 volt and 22 kHz tone switching
- potential for reduced power dissipation and thus cost reduction and improved reliability
- elimination of switching problems caused by incompatibility of system components
- easier receiver installation using device recognition

2. System Overview

The DiSEqC concept is based on extending the present 22 kHz tone-signalling method, thus minimising the changes required in Tuner or Integrated Receiver-Decoder (IRD) units, and simplifying backwards compatibility. However, since the full DiSEqC protocol supports a return-signalling path and multiple peripheral devices, it is necessary to more closely define the impedances (at 22 kHz) on the bus, than the simple low impedance drive (supply voltage modulation) commonly employed with the present tone method.

DiSEqC is a single master, single or multi slave system, so communications may be initiated only by the master Tuner/IRD. This avoids the need for the software in the Tuner/IRD to continually monitor the bus (by polling or interrupts) when there may be other tasks in hand. In principle, the master can transmit messages by "chopping" an existing 22 kHz tone, generated either entirely by software or with some hardware support.

The DiSEqC slave function generally will be implemented in a simple microcontroller. When this is dedicated to DiSEqC bus support it is practicable to perform not only the control functions but also the tone decoding and encoding in software, thus eliminating many of the components currently used for 22 kHz tone detection.

It is expected that a single (mask-programmed) microcontroller will be able to support most applications. However, the microcontroller will not have sufficient input/output pins to support all possible requirements concurrently, and must also be able to report back to the master which facilities are actually available. Therefore, at power up or reset, the microcontroller software will scan the peripheral hardware attached to its pins to determine the "family" which it represents, and the "resources" available.

In the more complex installations there is some risk of there being two "identical" slaves (e.g. Switchers or LNBs) on the same bus. The preferred method for dealing with this is a "loop through" structure (see Section 7.2) so that initially there is only one slave on the bus which the master can interrogate. When this device is first accessed, its address can be changed to one which is not normally used. It can then be instructed to loop the bus through to the next device, which can then be installed in a similar way.

To accommodate situations where identically-addressed devices do exist on the bus, a collision-detection and arbitration scheme is defined. Thus, if two devices have the same address, but some difference in their available "resources", then they can be identified by the master on subsequent occasions. However, because the device which "wins" any arbitration is effectively determined at random, then the identity of any truly identical devices will be lost when power is next removed.

3. Slave device architecture

As far as practicable, the intention is to make the DiSEqC Bus Specification independent of the actual slave hardware or microcontroller type employed. However, to assist in understanding the function of the various commands, the general structure of the DiSEqC Slave will be outlined. Full details will be found in the separate DiSEqC Slave Microcontroller Specification.

The prime function of the DiSEqC system is to remotely select between two or more switchable alternatives, such as the Polarisation of received signals, LNB Local Oscillator frequency, etc. The DiSEqC chip therefore has a number of logic output pins to select the various signal types either by electronic or electro-mechanical (relay) methods.

In hardware terms, the actual switching output pins of the DiSEqC slave chip normally will be identical so the actual control functions could be connected in any arbitrary manner. However, an aim of the DiSEqC Bus is to rigidly fix not only the pin functions (e.g. Linear Polarisation) but also their sense (e.g. Enabled = Horizontal Polarisation) so that system installation is greatly simplified. It is expected that the definitions will be observed not only in devices such as LNBs, but also in Switcher and SMATV installations, whenever possible.

In many applications of the DiSEqC slave chip there are further "uncommitted" switching output pins available which can be used for arbitrary switching functions, or, at a later date, may be allocated to any specific new facilities which may become desirable. To reduce the risk of the same pin being allocated to different functions, the pins will be allocated formally in ascending order (i.e. Switch 1 first) and OEM uses should be applied in descending order (i.e. Switch 4, or higher if available, first).

To allow the DiSEqC chip to report back to the master (Tuner/IRD) which facilities are available, the slave software must scan a number of input pins. These pins can be dedicated inputs, or combined input/output pins (as are available in some families of microcontroller) or even "virtual" pins fixed in a particular version of the control software. It is of no concern to the Bus Specification or the Tuner/IRD software how the configuration data is obtained, the master software will simply access a group of formally-defined "status registers" to indicate the available functions and external condition of the slave hardware.

Some peripheral device functions require continuously variable, or analogue, control signals, such as Polarisation Skew or "Installer Aid" signal strength. This information will be passed over the bus usually as a single byte value from 0 to 255. It may then be processed in several different ways, for example by applying the individual bits to 8 separate pins for driving a digital-analogue converter, or driving a single pin with a variable duty cycle Pulse Width Modulation (PWM) waveform which may be low-pass filtered to a steady dc value. Again it is not relevant to the Bus Specification how the actual analogue control level is achieved.

Another peripheral device requiring support is the "Positioner", to move a steerable dish to point in a required direction. Small incremental movements of the dish are normally detected as pulses by a reed switch, optical sensor, etc. on the motor drive. A DiSEqC chip can monitor these pulses, and also limit-switch (endstop) operation, etc. and employ simple software to drive the motor in the appropriate direction to achieve a requested accumulated rotation count. A preliminary set of bus commands will be defined to allow the movement counter to be loaded and read, but it is expected that the Positioner bus commands may be extended as experience with operation over a bus grows. The master-slave protocol demands that the Tuner/IRD must poll a status register at appropriate times to determine the progress of positioning process.

4. Bus Hardware Specification

To permit slave devices to communicate back to the master, it is necessary to make more specific recommendations for the operating impedances of the bus than is required for a simple 22 kHz continuous tone signalling. In addition, it is advantageous to be able to accept a wide range of operating voltages during the DiSEqC introductory phase, to permit compatible and/or hybrid operation with the present signalling methods. The recommendations for operation of the DiSEqC Bus are:

The long-term recommendation for the DC Supply voltage is 12 ±1 volts., but during the introductory phase, voltages up to 18 volts should be tolerated by peripheral devices (i.e. not suffer catastrophic failures). For the short term, some manufacturers may choose to retain compatibility with 13/17 volt signalling levels, and it is proposed to support this facility in the first versions of the DiSEqC Slave integrated circuit.

It is recommended that Tuner/IRDs should support a DC Supply current drain of up to 500 mA for peripheral devices powered by the bus.

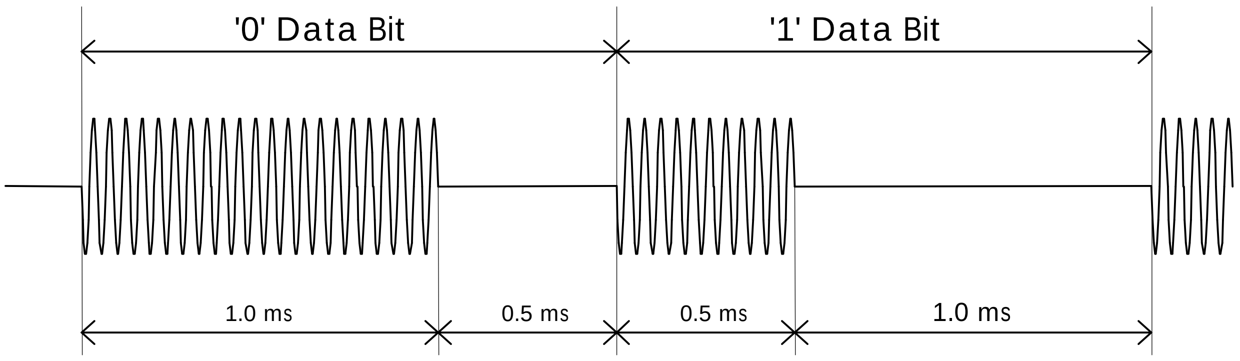
The nominal 22 kHz signalling amplitude is 650 mV peak to peak. To accommodate tolerances and voltage drop in the cable, the detector should respond to amplitudes down to approximately 300 mV (±100 mV). The maximum recommended amplitude to be applied to the bus is 1 volt peak to peak.

To permit back-channel signalling, the Master transmitter (in the Tuner/IRD) should present a nominal source impedance of 15Ω to the bus, at 22 kHz. This termination will typically consist of a resistor, a parallel inductor to support the DC power supply current, and a capacitor (to ground) to shape the 22 kHz signal when the cable length and termination capacitance are both small.

To permit transport of the 22 kHz signal, it is recommended that the total load capacitance at the far end of the bus (cable) should not exceed 250 nF (0.25 µF). This value is chosen to tolerate one LNB of existing design for a cable length of up to 50 meters. True DiSEqC peripherals should not load the bus by typically more than 100 nF, and for certain classes of device such as SMATV nodes and Installer Aids, a much lower value is preferred. It is expected that installation tables will be made available to indicate the maximum acceptable loading for various combinations of terminating device(s) and cable lengths.

5. Method of Data-Bit Signalling

DiSEqC uses base-band timings of 500µs (± 100µs) for a one-third bit PWK (Pulse Width Keying) coded signal period on a nominal 22 kHz (±20%) carrier. The end of each DiSEqC message is signalled by a minimum of 6 ms of silence. The following diagram shows the 22 kHz time envelope for each bit transmitted, with nominally 22 cycles for a Bit '0' and 11 cycles for a Bit '1':



5.1 Backwards Compatibility

DiSEqC commands are able to co-exist on the bus (i.e. the satellite downlead cable) with the established voltage and 22 kHz tone switching methods. Thus the DiSEqC data may be superimposed on a supply voltage of either 13 or 17 volts (for H/V polarisation switching) and also may be inserted between a continuous 22 kHz tone (for Local Oscillator frequency or Satellite selection), provided there are short "quiet" gaps before and after the DiSEqC command burst. The recommended timing periods are defined in the next section.

There are two aspects to this compatibility:

Firstly, it is possible to add a new DiSEqC-controlled switch to select between one or more existing LNBs. The Tuner/IRD sends a DiSEqC message to the switch to select the required input, and then the Tuner/IRD can send the appropriate voltage and/or continuous tone to select the required output from the LNB.

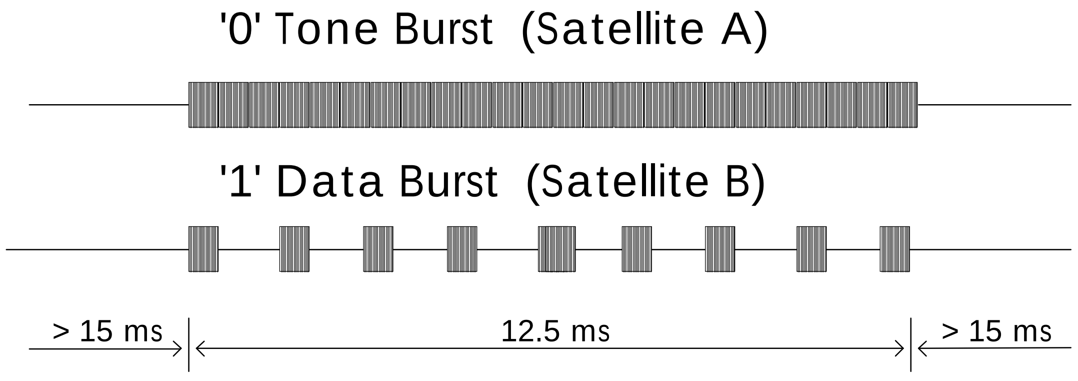
Secondly, the introductory versions of the DiSEqC Slave microcontroller will initially (until they receive a DiSEqC command) respond to the established signalling methods so that they may be used under the control of non-DiSEqC Tuner/IRDs. This reduces the need for manufacturers to produce different versions of LNBs and Switchers for "old" and "new" systems, and increases the chances that a user will already have some DiSEqC accessories installed when he upgrades to a DiSEqC Tuner/IRD at some later date.

5.2 Simple Toneburst Control Signal

To avoid the need to use a complete DiSEqC Slave microcontroller just to add a simple two-way switch to an existing system, a simple "toneburst" command is being added to the DiSEqC Specification, to permit detection by simple analogue hardware. The toneburst uses a 22 kHz carrier and is sent after any normal DiSEqC data burst, but before the resumption of the continuous tone (if this is being used for backwards-compatible signalling). Two types of toneburst are used, one Unmodulated to select "Satellite Position A" , and the other Modulated to select "Satellite Position B".

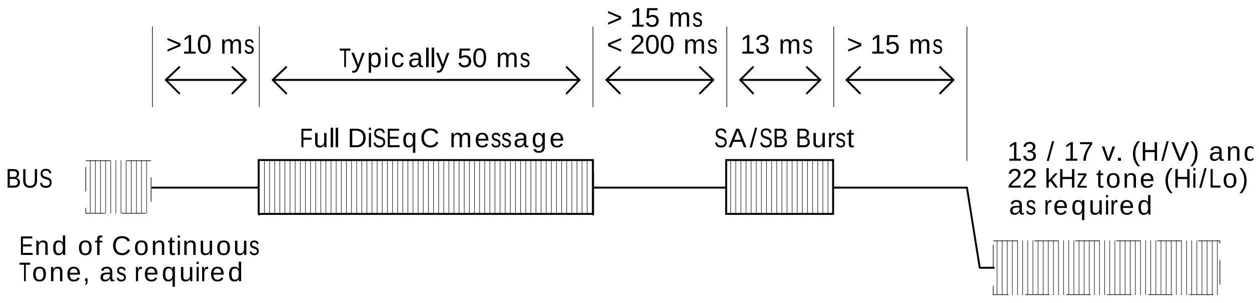
The nominal Modulated toneburst corresponds to a single-byte DiSEqC command of nine '1's (i.e. 'FFh'), which has a 1 : 3 duty cycle, so low-pass filtering of the detected modulated carrier gives a d.c. level which is approximately 33% of the peak carrier. The nominal burst duration is 12.5 ms (nine 0.5 ms pulses with eight 1.0 ms gaps) with normal DiSEqC modulation tolerances of ±20%.

The Unmodulated toneburst has a similar duration to the modulated burst and can be detected as a d.c. level approximately 3 times larger than the modulated burst. The detection circuitry may use either this difference in levels, or may detect the presence or absence of the modulation tone itself. In either case, the termination of the toneburst can cause the appropriate control value to be latched into a flip flop.



5.3 Combined Transmission of DiSEqC and Backwards-Compatible Signals

It is possible to combine both types of signalling on the bus, either to specifically control accessories of both types, or simply to avoid the need to configure the Tuner/IRD to a particular command protocol. Generally the control data carried in the two protocols will be the same, but this is not essential because the DiSEqC slave microcontroller ignores any backwards compatible signalsone it has received a valid DiSEqC command.



The above figure shows the typical timing of the combined signals. The minimum time gaps must be observed to avoid the risk of malfunctions (particularly of simple analogue decoder hardware) but time intervals longer than the maximum should only produce minor temporary "glitches" in the switch lines. In two-way DiSEqC, the master may need to leave a gap of greater than 150 ms between the message and the burst to wait for the slave's reply. However, the reply normally will be received much sooner and then the toneburst can be sent after a pause of 15 ms.

6. Message Data Format

DiSEqC messages consist of one or more bytes of data, each formed from 8 bits as defined above, and with each byte followed by an odd parity bit. The ordering of the bits is that the most-significant bit is transmitted first, and then in descending sequence to the least-significant bit last.

The detailed message format is described in later sections, but the basic command structure (from the Tuner/IRD master) consists of 3 bytes: a start and run-in byte which includes some high-level control, a slave address byte and a command byte. For some commands, additional data is carried in a variable number of subsequent bytes.

The return of a "reply" message from the slave is at the discretion of the master so that the bus can be kept clear if the master is unable to receive a reply (e.g. "one-way" DiSEqC 1.0) or wishes to send rapid updates (e.g. of an analogue "Skew" value). However, it is expected that most unique-address commands will request an acknowledgement because arbitration and communications validation cannot be performed without one. The reply will normally consist of a single acknowledge byte, which will be followed by one or more additional bytes when the command involves the return of data (or to supply an error-code). The slave should reply within a maximum of 150 ms from the end of the command message.

The handling of errors can be largely at the discretion of the master (Tuner/IRD) control software. In general it is recommended that any command which fails to receive a reply should be repeated once, but a different number of repeats might be used during the installation process (and as a result of the behaviour of the slave at this time).

To be compatible with possible future extensions, the message bytes normally should be fully decoded from their definitions in section 8. However, as an introduction, their overall structure is outlined below.

6.1 Framing (first) byte

The first four bits of the first byte will be initially reserved as a "run in" and "framing" pattern '1110' for DiSEqC commands. It is possible that future extensions may make use of these bits, but it may be assumed that the first two bits always will be identical (i.e. contribute even parity) for error-detection purposes.

The fifth bit initially will be reserved as '0' but may be used in the future for "open" extensions to the protocol.

The sixth bit is cleared to '0' if the message originates from the master and is set to '1' if it is a reply from a slave.

The seventh bit is set to '1' if a response is required and cleared to '0' if no reply is expected. In the case of a slave, the set bit can request a re-transmission because an error has been detected.

In command messages, the eighth bit is set to '1' if the message is a re-transmission of a previous message (for which a reply was expected and not received). In reply messages the eighth bit is used to qualify the reason for failure to execute a command (either erroneous or unsupported command).

6.2 Address (second) byte

The second byte indicates which slave device the message is intended for, and is divided into two nibbles of 4 bits each. The first nibble indicates the overall "family" to which the slave belongs (e.g. LNB, Positioner, etc.) and the second nibble allows the family to be divided into sub-types for some applications. In each nibble the value '0' (binary '0000') indicates a "don't care" situation and can command any slave within the family and/or sub-type.

Original Equipment Manufacturers may apply for an OEM address byte, which may then be followed by a moderate number of bytes (provisionally a maximum of 6), using any protocol which does not conflict with normal operation of the DiSEqC Bus.

6.3 Command (third) byte

The third byte specifies the required action(s) to be taken by the addressed slave(s). The byte generally should be fully decoded, although some related commands (by function or format) are grouped together to simplify decoding.

6.4 Data (subsequent) bytes

The fourth and subsequent command bytes (from the master) and second and subsequent reply bytes (from the slave) contain numeric data when necessary. This may be an "analogue" value (e.g. for a Polariser), a number (e.g. related to the LNB Local Oscillator frequency or a pulse count for Positioner operation, etc.) or a group of up to 8 separate flag bits.

7. Initialisation

When the master applies power to the bus, any slave which supports backwards compatibility takes up a state which is consistent with the voltage and tone signals on the cable. Then, if the slave detects a valid DiSEqC command on the bus, it abandons the backwards compatible mode and sets up its control lines in response to the DiSEqC commands. Although some slaves may take up a defined state at power-up, the master control software must NOT make any assumptions, but should issue ALL relevant commands during the initialisation procedure.

7.1 Power (Standby) Control

Since there may be more than one peripheral device powered from the bus, DiSEqC slave devices should ideally start up in the power-down (standby) state to avoid overloading the master's power supply. However, backwards-compatible devices must become fully active without receiving any DiSEqC commands. A solution to this dilemma is for the slave to initially start in the standby state, but automatically switch to full operation after typically one second if **no** DiSEqC command is received. Thus, early in the initialisation procedure, the master could issue a general power-down command to prevent multiple slaves overloading the power supply, and then later enable the power supply in the appropriate peripheral device.

Because the preferred start-up state for slaves is into standby, it is important that **all master software** (even simple "one-way" DiSEqC 1.0, with no reply) **must issue a power-on command during the initialisation procedure**.

7.2 Bus Arbitration

The preferred method for preventing contention on the bus during initialisation is to avoid having more than one device with the same family address "listening" to the bus at the same time. This can be achieved by using a "loop through" architecture, which is particularly relevant to switcher devices, but can include LNBs. In this architecture, one slave device connects directly to the bus (going to the master) and other slaves are then connected to the bus via the first device. At power-up (initialisation), the first device responds to the bus, but blocks the DiSEqC commands from propagating to the further devices (either specifically, or in conjunction with the normal IF signal switching). When the first device is being initialised, it can be allocated a unique address and then commanded to switch the bus through to the next device in the chain. This process could be continued for several levels, if the master's control software is sufficiently sophisticated, but aspects of power management and response times will need careful consideration.

7.3 Bus Collision detection

The single-master protocol and the limited number of slave devices expected on the bus should give a low risk of data collisions. However, to simplify the initialisation process, it is proposed to initially specify a relatively small number of different device families and device types. Also, because of the "mechanical" aspects of satellite position and plane of polarisation there may be legitimate occasions when slaves with identical addresses appear on the bus. A simple strategy for resolving conflicts is proposed:

Before a slave device replies to any command to its address (at least after power-up until completion of the system initialisation process) it should monitor the bus for any other slave responses, for a random time of between typically 15 ms and 115 ms. If it detects another DiSEqC reply message it should cancel its own attempt to reply and set an internal "bus conflict" flag. Whilst this "conflict" flag remains set, the slave should either not reply at all, or wait for a period of typically 130 ms (i.e. slightly longer than the maximum random delay) and reply only if no other response is detected. This extended delay avoids any collisions with the reply from the slave which "won" the arbitration, but does not cause a slave to "disappear" from the system if the contention flag becomes set accidentally.

The master can detect the existence of the contention in various ways. It may be able to detect the extended response time from the slave, and/or issue a command which applies only to a device with its contention flag set (i.e. "return address"). Alternatively, the master can change the address of any slave known to it, and then test for any further responses. The "change address" command is considered more useful than the "sleep" commands defined in previous specifications.

If the master does send a command to change the address of a slave, it should preferably checkfor a reply from the new address. To attempt to retain the identity of the conflicting devices after a power-down, the master should ideally interrogate the available "resource" registers of each device and store any distinguishing feature in its Non-Volatile Memory.

8. Definition of DiSEqC Bus Commands

The following tables list the currently-defined DiSEqC Bus Control Commands. Commands from the DiSEqC Master consist of 3 bytes, plus any ancillary data bytes, each followed by an odd parity check bit. Slave Reply messages consist of 1 byte, plus any ancillary data bytes, each followed by an odd parity check bit.

Master Command :

FRAMING	P	ADDRESS	P	COMMAND	P	DATA	P

Slave Reply :

FRAMING	P	DATA	P	DATA	P

8.1 Framing Byte

The following framing bytes have so far been defined:

Hex Byte	Binary	Framing byte Function
E0	1110 0000	Command from Master, No reply required, First transmission
E1	1110 0001	Command from Master, No reply required, Repeated transmission
E2	1110 0010	Command from Master, Reply required, First transmission
E3	1110 0011	Command from Master, Reply required, Repeated transmission
E4	1110 0100	Reply from Slave, OK, no errors detected
E5	1110 0101	Reply from Slave, Command not supported by slave
E6	1110 0110	Reply from Slave, Parity Error detected - Request repeat
E7	1110 0111	Reply from Slave, Command not recognised - Request repeat

8.2 Address Byte

The address byte is divided into two nibbles of four bits to define a family and sub-type:

FAMILY	SUB-TYPE

The following table lists the addresses which so far have been defined:

Hex Byte	Binary	Family and Sub-type
00	0000 0000	All Families (Don't care address)
10	0001 0000	Any LNB Switcher or SMATV
11	0001 0001	LNB
12	0001 0010	LNB with Loop-through
14	0001 0100	Switcher
15	0001 0101	Switcher with Loop-through
18	0001 1000	SMATV
20	0010 0000	Any Polariser
21	0010 0001	Full Skew (Linear Polarisation) control
22	0010 0010	Skew trimming adjustment
30	0011 0000	Any Positioner
40	0100 0000	Any Installation Aid
41	0100 0001	Signal-strength alignment aid
60	0110 0000	Family reserved for address re-allocations
70	0111 0000	Intelligent slave interface for proprietary Multi-Master bus
Fx	1111 xxxx	OEM Extensions

8.3 Command Byte

The command bytes define the actions required of the addressed slave(s). The table below lists the commands which have so far been defined. The first column indicates the relative priority which should be placed on incorporating the commands in first generation products. 'M' indicates commands which are considered mandatory for inclusion in even minimum implementation ("one-way" DiSEqC 1.0) master devices (Tuner/IRDs). 'R' recommends commands for use in "two-way" (DiSEqC 2.0) systems and 'S' suggests commands which it is believed will be of value in the relatively short term. The final column defines the reply data bytes which would be expected to be received from the addressed slave.

Pri-ority	Hex Byte	Command Name	Command Function	Total trans. Bytes	Reply Data byte(s)
R	00	Reset	Reset DiSEqC microcontroller	3	
R	01	Clr Reset	Clear the "Reset" flag	3	
R	02	Standby	Switch peripheral power supply off	3	
M	03	Power on	Switch peripheral power supply on	3	
	04	Set Contend	Set Contention flag	3	
S	05	Contend	Return address if Contention flag is set	3	Address
S	06	Clr Contend	Clear Contention flag	3	
S	07	Address	Return address if Contention flag is clear *	3	Address
	08	Move C	Change address if Contention flag is set	4	
R	09	Move	Change address if Contention flag is clear *	4	
R	10	Status	Read Status register flags	3	Status
R	11	Config	Read Configuration flags (peripheral hardware)	3	Config.
R	14	Switch 0	Read Switching state flags (Committed port)	3	Switches
R	15	Switch 1	Read Switching state flags (Uncommitted port)	3	Switches
	16	Switch 2	expansion option	3	
	17	Switch 3	expansion option	3	
R	20	Set Lo	Select the Low Local Oscillator frequency	3	
R	21	Set VR	Select Vertical Polarisation (or Right circular)	3	
R	22	Set Pos A	Select Satellite position A *	3	
R	23	Set S0A	Select Switch Option A (e.g. Linear Pol.) *	3	
R	24	Set Hi	Select the High Local Oscillator frequency	3	
R	25	Set HL	Select Horizontal Polarisation (or Left circular)	3	
R	26	Set Pos B	Select Satellite position B *	3	
R	27	Set S0B	Select Switch Option B (e.g. Circular pol.)*	3	
R	28	Set S1A	Select switch S1 input A (deselect input B)	3	
R	29	Set S2A	Select switch S2 input A (deselect input B)	3	
R	2A	Set S3A	Select switch S3 input A (deselect input B)	3	
R	2B	Set S4A	Select switch S4 input A (deselect input B)	3	
R	2C	Set S1B	Select switch S1 input B (deselect input A)	3	
R	2D	Set S2B	Select switch S2 input B (deselect input A)	3	
R	2E	Set S3B	Select switch S3 input B (deselect input A)	3	

R	2F	Set S4B	Select switch S4 input B (deselect input A)	3	
	30	Sleep	Ignore all bus commands except "Awake"	3	
	31	Awake	Respond to future bus commands normally	3	
M	38	Write N0	Write to Port group 0 (Committed switches)	4	
M	39	Write N1	Write to Port group 1 (Uncommitted switches)	4	
	3A	Write N2	expansion option	4	
	3B	Write N3	expansion option	4	
S	40	Read A0	Read Analogue value A0 *	3	byte value
S	41	Read A1	Read Analogue value A1 *	3	byte value
R	48	Write A0	Write Analogue value A0 (General)	4	
S	49	Write A1	Write Analogue value A1	4	
	4F	Write A7	Write Analogue value A7 (expansion option)	4	
S	50	LO string	Read current frequency [Reply = BCD string]	3	BCD bytes
R	51	LO now	Read current frequency table entry number	3	F number
S	52	LO Lo	Read Lo frequency table entry number	3	F number
S	53	LO Hi	Read Hi frequency table entry number	3	F number
	58	Write Freq	Write channel frequency (BCD string)	6	
	59	Ch. No.	Write (receiver's) selected channel number	5	
	60	Halt	Stop Positioner movement	3	
	61	Go E	Drive Positioner motor East	3	
	62	Go W	Drive Positioner motor West	3	
	64	P Status	Read Positioner status register	3	Pos Status
	65	Read Pos	Read Positioner counter [Reply = Hi, Lo]	3	Count (2)
	6C	Goto	Drive Positioner motor to counter value, Hi, Lo	5	
	6D	Write Pos	Write Positioner counter, Hi, Lo	5	

* Amended from Bus Specification 3.1

9. Description of DiSEqC Bus Commands

The following sections describe specific aspects of the commands defined in Section 8.

9.1 Read Frequency Table entry number (LO x commands)

The Local Oscillator frequency is normally returned as a single byte which represents the entry number in the following list. The list may be added to in the future up to entry number 239. Should the need arise for further expansion then the values 240 to 255 represent pointers to up to 16 auxiliary tables of 256 entries each. An additional byte will indicate the entry number in the appropriate auxiliary table.

Frequency list number byte	Local Oscillator Frequency	L.O. Offset (relative to carrier)
0	Reserved	
1	Not known	
2	9.750 GHz	Low
3	10.000 GHz	Low
4	10.600 GHz	Low
5	10.750 GHz	Low
6	11.000 GHz	Low
7	11.250 GHz	Low
8	11.475 GHz	Low
9	20.250 GHz	Low
10	5.150 GHz	
11	1.585 GHz	
12	13.850 GHz	High
13	not allocated	
14	not allocated	
15	not allocated	

To provide backwards compatibility with first generation DiSEqC systems, future additions to the frequency list should also be supported in packed BCD string form. In response to the master's request for the frequency string, the slave returns sufficient bytes to define the frequency. The high nibble of the first data byte contains the tens of GHz digit and the low nibble contains the GHz digit in BCD form. Subsequent bytes contain BCD nibbles in descending order until only trailing zeros remain. For example 13.850 GHz would be returned in two bytes:

0001 0011	1000 0101

Note to receiver manufacturers:
It is strongly recommended for future applications to allow the tuner to be controlled via the DiSEqC bus, i.e. rather than take the frequency from the receiver's internal look-up table the tuner will read the frequency returned by the slave on the DiSEqC bus. This could be implemented by a set-up command, for instance in the installation menu of the receiver, which sets the tuner to either internal or external (bus) control. A useful extension of this concept might also allow ALL the programmable features of the receiver to be controlled by the bus so that one receiver can be directly programmed via an intelligent installation aid (even from the roof!).

9.2 Write Port Format (Write Nx commands)

In addition to commands which set or clear individual switch states, there are commands which can control groups of 4 switching lines with only one byte. The new switching combination is defined by a single data byte following the command and is arranged such that any combination of individual switches can be either changed or left in their previous state. This is achieved by allowing the two separate nibbles of the data byte to determine the switches' state in different ways. Any bits set in the high nibble CLEAR the corresponding switch control lines and any bits set in the low nibble SET the corresponding lines. To ensure predictable operation the high nibble should always effectively be applied first, followed immediately by the low nibble. Thus if a group of four lines is initially set to 'WXYZ' and a write data byte '0011 0101' is sent, then the resulting switching lines become 'W101'. To set up all four lines, the high nibble may be set to '1111' and the required switching state defined in the low nibble.

CLEAR	SET
data nibble	data nibble

Initial state	W X Y Z
CLEAR flags	0 0 1 1
Intermediate result	W X 0 0
SET flags	0 1 0 1
Result	W 1 0 1

9.3 Read Status Byte

The Status register contains individual flag bits to indicate the operational conditions of the slave software and the peripheral hardware. The flag bits are allocated from high to low weighting as follows:

Bit Number	Status Function
.7	Bus-Contention flag is set
.6	Standby mode is selected
.5	
.4	Auxiliary power is currently available
.3	
.2	Bus voltage is above 15 volts threshold
.1	
.0	A reset has occurred since this flag was last cleared

9.4 Read Configuration Byte

The Configuration byte contains flags as defined in the following table:

Bit Number	Configuration Function
.7	Device has Analogue output facility
.6	Device has Standby (power-down) facility
.5	Device has Positioner capability
.4	Device has external power detection capability
.3	Device has loop-through facilities
.2	
.1	Device has Switcher capability (including LNBs)
.0	Device can supply LO frequency value(s)

9.5 Read Committed Switches Byte

The low nibble of the Committed Switches byte contains flags to indicate whether each corresponding named signal is controllable, and the high nibble indicates the current switched state. The byte thus contains flags as follows:

Bit Number	Committed switches state
.7	Options switch position 'B' is selected
.6	Satellite Position 'B' is selected
.5	Horizontal Polarisation is selected
.4	High Local Oscillator is selected
.3	Options Switch is available
.2	Two (or more) satellites are switchable
.1	Linear Polarisation is switchable
.0	Local Oscillator is switchable

9.6 Read Uncommitted Switches Byte

The low nibble of the Uncommitted Switches byte contains flags to indicate whether each corresponding switch is controllable, and the high nibble indicates the current switched state. The byte thus contains flags as follows:

Bit Number	Uncommitted switches state
.7	State of Uncommitted switch 4
.6	State of Uncommitted switch 3
.5	State of Uncommitted switch 2
.4	State of Uncommitted switch 1
.3	Uncommitted switch 4 is available
.2	Uncommitted switch 3 is available
.1	Uncommitted switch 2 is available
.0	Uncommitted switch 1 is available

9.7 Read Positioner Status Byte

The Positioner Status register contains individual flag bits to indicate the operational conditions of the slave software and of the peripheral hardware related to dish positioner functions.

Bit Number	Positioner Function
.7	Move command has been completed
.6	
.5	Movement direction is West
.4	Motor is running
.3	
.2	Power is not available
.1	End stop has been hit
.0	Location count register is not valid