

RMT-1

Interface Specification

Release 2.04 - May 31, 2001

RPM Systems Corporation

17371 N.E. 67th Court, Suite A-5

Redmond, Washington, 98052 USA

Ph: (425)869-3901 FAX: (425)883-9552

Contents

1. Introduction	4
1.1 Definition of Terms	4
2. System Overview	5
2.1 Control Connector Signal Descriptions	5
2.2 Signaling Levels and Termination	5
2.3 External Trigger Inputs	5
2.4 Trigger Bus Outputs	6
2.5 Data Communications Protocol	6
2.6 Root 1 Commands and Responses	6
2.7 Modes of Operation	7
2.7.1 Automatic Mode	7
2.8 Root 1 Host Controller Functions	8
3. Root 1 Immediate Mode Commands	9
3.1 Device Request	9
3.2 Port Power On/Off	11
3.3 Global Suspend	12
3.4. Global Resume	12
3.5. Set VCC	12
3.6. Measure Vbus Current	13
3.7. Root_Config	13
3.8 USB_Reset Command	15
3.9 DevTrans Command	15
3.10 Data Port and Trigger Out	17
3.11 Read Root Hub Status	18
4. Asynchronous Responses	20
4.1 Connect Event	20
4.2 Status Event	20
4.3 Data Event	21
4.4 Error Event	21

4.5 Root Fail Event	22
4.6 Command Error	22
4.7 Trigger Event	22
5. RootScript	23
5.1 Program Command	23
5.2 RS_End	24
5.3 Run	25
5.4 Responses during Script Execution	25
5.4.1 ResponseMode Command	26
5.5 Conditions and Flow Control	26
5.5.1 RS_Goto Command	26
5.5.2 RS_IfCommand	27
5.5.3 RS_Check and RS_Cond	27
5.5.4 RS_Timer	28
5.5.5 RS_Call and RS_Return	28
5.6 RS_Message	29
6. Script Management	30
6.1 The Default Script	30
6.2 The Flash Command	30
7. Power	32

1. Introduction

Root 1 is a USB host controller which is targeted for test applications. It is USB 2.0 specification compliant for low-speed and full-speed interfaces, and is designed to be employed in a development or manufacturing test environment for the purpose of testing low-speed and full-speed Universal Serial Bus (USB) peripherals. The device provides a subset of the features typically provided by a USB Host and Root Hub, but its activity is controlled via a serial communications interface. This allows the test engineer to completely control the low-level root hub functions without the interference of a complicated operating system such as would be encountered on a Windows or Mac PC. Root 1 provides the following basic capabilities:

- Full-speed / Low-Speed USB Serial Interface Engine (SIE)
- CRC generation/detection
- USB Transaction handshaking
- Device connect/disconnect detection and reporting
- Automatic device configuration (automatic mode)
- Automatic detection of high/low speed devices
- External hub support (automatic mode)
- Global suspend/resume
- Remote wake-up support
- Automatic frame (SOF) generation
- Automatic Interrupt IN endpoint polling (automatic mode)
- Generates USB Reset on command
- Vbus current measurement
- Vbus voltage control (4.40V ~ 5.25V)
- External Trigger Inputs and Outputs
- 8-bit user Output port
- RootScript scripting language support

1.1 Definition of Terms

The terms defined in this section will be used throughout this document.

Controller The device used to control Root 1 via the serial communications port.

Root Port The USB port on Root 1.

RootScript Root 1's scripting language, which allows pre-written scripts to be downloaded and stored in Root 1 and be subsequently invoked by the Controller.

2. System Overview

This section provides a brief overview of Root 1's capabilities. Many issues are dealt with in more detail in the later sections of the document.

2.1 Control Connector Signal Descriptions

Root 1 communicates with the Controller via a full-duplex RS-232 serial data connection. The serial data format is 19.2K baud with one start bit, eight data bits and one stop bit.

Root 1 provides a 25-pin female D subminiature connector containing the following signals:

<u>Pin #</u>	<u>Signal</u>	<u>Description</u>	<u>In/Out</u>	<u>Notes</u>
1	Aux_TxD	Auxiliary Serial Xmt	Out	Do Not Connect
2	RxD	Serial Receive Data	In	
3	TxD	Serial Transmit Data	Out	
4	Aux_RxD	Auxiliary Serial Rcv	In	Do Not Connect
5~13	GND	Signal Ground		
14	TrigIn0	Trigger Input 0	In	
15	TrigOut0	Trigger Output 0	Out	
16	TrigIn1	Trigger Input 1	In	
17	PD0	Parallel data bit 0 (LSB)	Out	
18	PD1	Parallel data bit 1	Out	
19	PD2	Parallel data bit 2	Out	
20	PD3	Parallel data bit 3	Out	
21	PD4	Parallel data bit 4	Out	
22	PD5	Parallel data bit 5	Out	
23	PD6	Parallel data bit 6	Out	
24	PD7	Parallel data bit 7 (MSB)	Out	
25	Mode	Parallel Data Mode	In	Reserved - Do Not Connect

2.2 Signaling Levels and Termination

The serial communications lines (TxD, RxD, Aux_TxD and Aux_RxD) employ EIA-232 electrical levels. All other signals are 5V logic compatible levels. The trigger output and parallel data lines are source terminated on Root 1 with 47-ohm series resistors. All 5V data and control lines are provided with 4.7K-ohm pullup resistors to 5V on Root 1.

2.3 External Trigger Inputs

Root 1 provides two external, logic-level trigger inputs. The TrigIn0 and TrigIn1 inputs are asserted low, and filtered in hardware to remove noise. A trigger input will be detected and latched by Root 1 on the falling-edge transition. A Trigger input must be driven low and held low for at least 1 uS to guarantee recognition by the

Root 1. If the Root 1 is in immediate-mode (i.e., not executing a RootScript), the trigger event will be reported to the Controller as an Asynchronous Response. During RootScript execution, trigger inputs will be latched, but will not be acted upon until they are specifically addressed by the script (see *RS_Check* command).

2.4 Trigger Bus Outputs

The Trigger Bus Outputs are essentially a parallel data port which can be written directly via a Controller or RootScript command. The data provided by the Controller or RootScript command is driven onto the parallel port, and the output strobe, TrigOut0, is toggled. The data will remain on the output port until a new piece of data is written by the Controller or RootScript. TrigOut0 is a low-going strobe approximately 10uS in duration. Approximately 10uS data setup time is provided prior to the assertion (low) of the strobe. The parallel port data lines are capable of sourcing or sinking up to 8mA each, and may be used to directly drive LED indicators if suitable current-limiting is employed (e.g., 470-ohm or larger value series resistors).

2.5 Data Communications Protocol

The Root 1 employs an escaped binary data communications protocol. Data is transmitted as variable length packets, and packet control is implemented using ASCII escape sequences. All serial data transmissions to and from the Root 1 are framed in the following message protocol:

<SOP><transmission code>{data}<EOP>

SOP is a start-of-packet marker consisting of two ASCII bytes: <Esc>S (0x1B 0x53)

EOP is an end-of-packet marker consisting of two ASCII bytes: <Esc>E (0x1B 0x45)

<transmission code> is a single byte code indicating the nature of the transmission. For commands sent to Root 1, this byte will contain a command code. In responses received from Root 1, this byte will contain a response code.

{data} is a string of binary data bytes 0~4096 bytes in length.

The ASCII Escape character (0x1B) is used as a special character, to implement message control. Since the protocol supports binary data transfer, the data value 0x1B may be encountered in the data stream. The two-byte pattern <Esc><Esc> (0x1B 0x1B) is used to represent the occurrence of the single byte 0x1B if it is encountered in {data}.

2.6 Root 1 Commands and Responses

Commands which are transmitted to the Root 1 from the Controller over the serial port, using the message protocol discussed earlier, are referred to as *interactive commands*. Any properly formatted interactive command will invoke a response from Root 1. The interactive commands, and their responses, are discussed in section 3.

Responses transmitted by the Root 1 to the Controller are of two types: Command Responses and

Asynchronous Responses. Command Responses are transmitted in direct response to a command received from the Controller. Asynchronous Responses are transmitted as the result of asynchronous conditions, for example, a USB device connect. Command Responses are discussed in section 3, in coordination with the commands with which they are associated. Asynchronous responses are discussed in section 4.

2.7 Modes of Operation

Root 1 has two main modes of operation: Interactive Mode and Script Mode. Interactive Mode is the normal mode of operation, in which commands are issued by the Controller and responded to by Root 1. Root 1 may also issue Asynchronous Responses in Interactive Mode to indicate the occurrence of asynchronous conditions. Within Interactive Mode operation, Root 1 supports an additional mode of operation called Automatic Mode, in which many of the functions of a typical USB host controller are performed automatically by Root 1. Automatic Mode is discussed in the next section.

Root 1 supports a scripting language called RootScript, which consists of the standard set of Root 1 Interactive Commands, plus additional scripting commands for program flow control. A RootScript is downloaded into Root 1 in Interactive Mode. Once the script has been loaded into Root 1, it may be executed, causing Root 1 to enter Script mode. Root 1 will remain in script mode, executing RootScript commands, until the script terminates or until a new Interactive Command is received on the serial port. Automatic Mode is not available in Script Mode. RootScripting is discussed in section 5.

At power up, Root 1 typically defaults to Interactive Mode operation with Automatic Mode enabled. However, Root 1 allows for a *default script* to be loaded and saved in on-board Flash memory. Once the default script has been saved, Root 1 can be programmed to enter Script Mode immediately after power up, executing the default script.

Functions intrinsic to the basic operation of the USB, such as SOF (Start of Frame packet) generation, overcurrent protection, etc., are always handled automatically by Root 1 hardware, regardless of mode of operation.

2.7.1 Automatic Mode

The default mode of operation for the Root 1 is Automatic Mode. In this mode, the Root 1 will automatically detect the attachment of a new device to the root port, reset it, enumerate it and configure it. If the attached device is a hub, the Root 1 will read the hub's Hub Descriptor, configure its status endpoint and enable power to its downstream ports. The Root 1 will thereafter routinely poll the hub's status, identify connect events on the hub's downstream ports and reset and enumerate newly connected devices. In addition, non-hub devices connected to downstream ports of the hub device will be automatically reset and enumerated upon attachment. Non-hub Interrupt devices attached either directly to the root port or to the downstream port of an attached hub will be configured for up to four endpoints, in addition to the default endpoint, based on each device's Interface and Endpoint descriptors. Interrupt endpoints will be automatically polled in accordance with their Endpoint and Interface descriptors. Root 1 operation in Automatic mode, with regard to the connection of a new device, is essentially identical to the sequence that a typical USB host would exhibit in enumerating and configuring a new device.

In Automatic Mode, the Root 1 will generate an asynchronous response to the Controller any time an event occurs on the root port or on the downstream port of a hub attached to the root port. Examples of detectable events are Device Connect, Device Disconnect, Remote Wake-up, or a non-NAK response to polling on any endpoint.

Automatic Mode may be enabled and disabled by command from the Controller (*Root_Config* command). With Automatic Mode disabled, the Root 1 will not automatically enumerate new devices, nor will it automatically configure and poll downstream hubs or functions. The Controller then becomes responsible for issuing discrete device requests for all device and hub configuration, including polling hub status for new device connects on its downstream ports and polling downstream devices. Most Root 1 Interactive Commands can be used whether Root 1 is in Automatic Mode or not, however care must be taken that the commands issued do not modify the configuration of devices such that Automatic Mode activity is adversely affected.

2.8 Root 1 Host Controller Functions

Root 1 provides the basic host controller functions necessary for the proper operation of the USB. These include device speed detection on the root port, Vbus power switching and overcurrent protection, SOF (start of frame) generation, CRC generation and checking and packet handshaking. These functions are essentially invisible to the user, with the exception that SOF generation can be controlled by the Suspend and Resume commands. The Root 1 host controller is also responsible for the timing of transactions within a frame and the detection of overrun conditions, IGNORE conditions, etc.

3. Root 1 Immediate Mode Commands

This section details the set of interactive commands supplied by Root 1. The format of each command and its associated response are discussed. All transmissions, commands and responses, are conducted using the protocol discussed in section 2.4.

3.1 Device Request

The DevRqst command conducts a single control transfer, consisting of a valid USB device request, to be issued to any attached device. A valid USB device request is any properly formed request, not necessarily one that the target device is capable of handling. This command may be used with Automatic Mode enabled or disabled to issue any standard, class-specific or vendor-specific device request. When processing a DevRqst, the Root 1 will perform the entire request, including Setup, Data and Status phases. Multiple data transactions will be conducted as necessary to complete the request.

As control transfers, DevRqst commands are always targeted to the control endpoint (endpoint 0) of the device. INs and Outs which are generated by the DevRqst command and are NAK'd by the device are retried until they are successful (ACK'd), or until they time out. Per the USB Specification, a time-out will be generated during the data portion of the control transfer if a device does not return with a non-NAK response after 500mS, or if the duration of an entire device request sequence exceeds 5 seconds.

If the device being addressed by the DevRqst command has been connected and automatically enumerated by Root 1 in Automatic Mode, Root 1 will use the information it has collected during device configuration regarding device speed (full speed or low speed) and maximum packet size (*bMaxPacketSize0* field of the Device Descriptor) to properly conduct the DevRqst command. If the device has not been configured in Automatic Mode, or if the user desires for whatever reason to override the automatic-mode parameters for device speed or *bMaxPacketSize0*, this may be done by setting the <OVRD> flag in the <Address> byte of the DevRqst command, and including the additional <XferConfig> byte.

Note that DevRqst conducts a complete transaction, including Setup Data and Status phases. DevRqst accommodates multiple data transfers if necessary to complete the entire data transaction, and automatically handles data-toggle checking and NAK retries on data and status packets. If the test engineer requires device interaction to be broken down onto a more detailed level than that provided by DevRqst, the DevTrans command should be used. However, conducting transactions using DevTrans requires that the test script or interactive test software handle each packet of the transaction separately, including generating NAK retries.

The reader is referred to Chapter 9 of the USB Specification for information regarding the proper construction of a USB device request. This reference also addresses the details of the Standard USB Device requests. The details of class-specific (e.g., HID class specific or Audio class specific) requests are addressed in the Class specifications.

The form of the DevRqst command is:

<SOP><CMD_DevRqst><Address>{<XferConfig>}<Data><EOP>

<CMD_DevRqst> is the DevRqst command ID, 0x01.

<Address> contains the target device address (0 ~ 127) in bits 0~6, and the <OVRD> flag in bit 7.
For requests to devices which have been configured via normal Automatic Mode operation, OVRD is typically set to 0, and the <XferConfig> byte is omitted from the command. To override the Automatic Mode settings, or to issue requests to a device which has not been configured by Root 1 in Automatic Mode, the OVRD flag is set to 1, and the <XferConfig> byte is included in the command.

The <XferConfig> parameter is included in the DevRqst command *only* in the <OVRD> bit is set=1 in the <Address> byte. <XferConfig> allows the device's speed and *bMaxPacketSize0* value to be defined for devices which are not configured in Automatic Mode. The format of <XferConfig> is as follows:

<u>Bits 7~3</u>	<u>Bit 2</u>	<u>Bits 1,0</u>
<reserved=0>	<Speed>	<MaxPacketSize>

The <Speed> bit is set=1 if the device being addressed is a full speed device. If the <Speed> bit is set =0 (low speed) and the device attached to Root 1's root port is a high-speed device, Root 1 will assume that the low-speed device being address is downstream of a hub external to Root 1. In this case Root 1 will generate a full-speed PRE PID followed by a low-speed transfer for each packet issued to the device.

<MaxPacketSize> is the maximum data packet size to be used for transfers to and from the device. This information is normally obtained by Root 1 in Automatic Mode by reading the *bMaxPacketSize0* byte of the device's Device Descriptor, and is required to properly conduct some transactions with the device. The <MaxPacketSize> field of the <XferConfig> byte is encoded as follows:

<u>Bit 1</u>	<u>Bit 0</u>	<u>MaxPacketSize</u>
0	0	8 bytes
0	1	16 bytes
1	0	32 bytes
1	1	64 bytes

<Data> consists of binary data bytes comprising the valid USB device request, beginning with *bmRequestType* and ending with any data to be included with the request. At least eight bytes of data are required to make up the contents of the Setup packet. For device-to-host (Setup-IN) transactions, only these eight bytes of data are required. For host-to-device (Setup-OUT) transactions, any additional data to be sent with the request should be included in <Data> following the initial Setup information. Note that <Data> should include only actual data. Root 1 automatically generates PIDs, CRC's, etc, so this information should not be included in the <Data> parameter.

The device request is issued on the bus, and the Root 1 responds with the following:

<SOP><RESP_DevRqst><RespStatus>{<Response>}<EOP>

<RESP_DevRqst> is the DevRqst response ID, 0x81.

<RespStatus> is the response status as indicated in Table 3-1.

<Response> is the data, if any, returned by the target device.

Table 3-1: Response Status Values	
Value	Indication
0x00	Success
0x02	Ack
0x0A	Nak
0x0E	Stall
0x80	Ignore
0x81	Data CRC Error
0x82	Data Toggle Error
0x83	Sync Error
0x84	Babble Error
0x85	PID Error
0x86	ShortPacket Error
0x87	Configuration Error
0x88	Auto Mode Scheduling Error
0x89	USB Transmit Failure

DevRqst can accommodate an overall device response data size (the total number of bytes received in the data stage of a device-to-host request) of up to 4Kbytes on a single request.

Examples:

Read the device descriptor from device at address 2 configured in Automatic Mode:

Command: <SOP><0x01><0x02>{0x80 0x06 0x00 0x01 0x00 0x00 0x12 0x00}<EOP>

Response: <SOP><0x81><0x00>{0x12 0x01 0x01 0x01 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??}<EOP>

Read the device descriptor from a low-speed device at address 0 - force maxPacketSize=8

Command: <SOP><0x01><0x80><0x00>{0x80 0x06 0x00 0x01 0x00 0x00 0x12 0x00}<EOP>

Response: <SOP><0x81><0x00>{0x12 0x01 0x01 0x01 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??}<EOP>

3.2 Port Power On/Off

The Power command allows the Controller to turn Root port Vbus power on or off. Note that the execution of this command *only* switches bus power on or off; it does not affect the Vbus voltage setting, which is controller by the SetVcc command. The format of the command is:

<SOP><CMD_Power><Action><EOP>

<CMD_Power> is the command ID, 0x02.

<Action> is:

- 0x0 = Power Off or
- 0x1 = Power On.

The Root 1 will respond with :

<SOP><RESP_Power><EOP>

<RESP_Power> is the response ID, 0x82.

Example: Turn Vbus power on.

Command: <SOP><0x02><0x01><EOP>

Response: <SOP><0x82><EOP>

Vbus power defaults to OFF.

3.3 Global Suspend

Allows the Controller to command a global suspend, causing the Root 1 to discontinue SOF generation and polling of downstream devices. The format of the command is:

<SOP><CMD_Suspend><EOP>

<CMD_Suspend> is the command ID, 0x03.

The Root 1 will respond with :

<SOP><RESP_Suspend><EOP>

<RESP_Suspend> is the response ID, 0x83.

3.4. Global Resume

Allows the Controller to command a global resume, causing the Root 1 to resume SOF generation and polling of downstream devices. The format of the command is:

<SOP><CMD_Resume><EOP>

<CMD_Resume> is the command ID, 0x04.

The Root 1 will respond with :

<SOP><RESP_Resume><EOP>

<RESP_Resume> is the response ID, 0x84.

3.5. Set VCC

The Vcc command allows the Controller to set the root port Vbus voltage. Note that this command *only* sets the Vbus voltage. The state of the power on/off switch does not change. Vbus voltage may be changed with Vbus power on or off. The format of the command is:

<SOP><CMD_VCC><Vcc_Value><EOP>

<CMD_VCC> is the command ID, 0x05.

<Vcc_Value> is a one byte value in the range 40 ~ 125 decimal, indicating the voltage value for root port Vcc as follows:

$$V_{cc} = 4.00 + \langle V_{cc_Value} \rangle / 100 \text{ in volts}$$

e.g., a <Vcc_Value> of 40 decimal would yield a Vcc of 4.40V, and a Vcc_Value of 125 would yield a Vcc of 5.25V.

Root port Vcc will be accurate to +/- 2% over the range 4.40V ~ 5.25V. Root port Vcc defaults to 5.00V.

The Root 1 will respond with:

<SOP><RESP_VCC><EOP>

<RESP_VCC> is the response ID, 0x85.

Example: Set Vbus voltage to 5.00V.

Command: <SOP><0x05><0x64><EOP>

Response: <SOP><0x85><EOP>

3.6. Measure Vbus Current

The VccMeasI command returns a Root Port Vcc current measurement. The format of the command is:

<SOP><CMD_VccMeasI><EOP>

<CMD_VccMeasI> is the command ID, 0x06.

The Root 1 will respond with:

<SOP><RESP_VccMeasI><I_Value><EOP>

<RESP_VccMeasI> is the response ID, 0x86. <I_Value> is a byte value in the range 0~250 decimal, indicating the measured current drawn from the root port Vcc as follows:

$$\text{Current Drawn} = \langle I_Value \rangle * 3\text{mA.}$$

Current measurement will be accurate to +/-2% +/- 3mA. The Root 1 limits root port current draw to approximately 750mA for protection of the Root 1 itself and the downstream devices.

Example: Measure Vbus current.

Command: <SOP><0x06><EOP>

Response: <SOP><0x86><0x50><EOP>

' Root 1 returns Vbus current of 240mA

3.7. Root_Config

The Root_Config command allows the Controller to set various Root 1 configuration parameters. The parameters currently defined which can be affected by this command are:

Enable/Disable Automatic Mode.

Enable/Disable Trigger Inputs.

Enable/Disable AutoRecovery Mode.

Automatic Mode is described in section 2.7. With Automatic Mode disabled, the user application must handle all device interactions. Root port and external hub status must be polled to detect device connects, devices must be reset enumerated and configured explicitly using Interactive Commands

The external Trigger Inputs are discussed in section 2.3. The Root_Config command allows the two trigger inputs to be individually enabled and disabled. In Interactive Mode (no RootScript running), if a trigger input is enabled, and an external trigger is detected on that input, an Asynchronous Response will be generated by Root 1 indicating the trigger detection. If the trigger input is disabled, inputs on that trigger line will be ignored.

The AutoRecovery mechanism, when enabled, causes Root 1 to automatically attempt to recover from a downstream overcurrent condition. When an overcurrent condition is detected on the root port or on the downstream port of an external hub, power to the port is disabled in order to protect the host or hub. Enabling AutoRecovery causes power to any ports which have been disabled due to overcurrent to be reenabled on a one-second period. If the overcurrent condition has been removed, the port will then remain powered. If the overcurrent condition still exists, the port will be shut down again by the overcurrent detection circuitry. This mechanism effectively results in an automatic one-second retry of any overcurrent-disabled ports.

The format of this command is:

<SOP><CMD_Root_Config><Parameter><Data><EOP>

<CMD_Root_Config> is the command ID, 0x07. <Parameter> is a byte value indicating the parameter to be affected, and <Data> is a byte value indicating the setting to be applied to the parameter, as follows:

<Parameter> = 0: Automatic Mode
 <Data> = 0: Disable
 <Data> = 1: Enable
<Parameter> = 1: Trigger Inputs
 <Data> = 0: TrigIn1: Disabled TrigIn0: Disabled
 <Data> = 1: TrigIn1: Disabled TrigIn0: Enabled
 <Data> = 2: TrigIn1: Enabled TrigIn0: Disabled
 <Data> = 3: TrigIn1: Enabled TrigIn0: Enabled
<Parameter> = 2: Autorecovery Mode
 <Data> = 0: Disable
 <Data> = 1: Enable

At power up, Automatic Mode defaults to Enabled, both Trigger inputs default to Disabled, and AutoRecovery defaults to Disabled.

The Root 1 will respond with :

<SOP><RESP_Root_Config><EOP>

<RESP_Root_Config> is the response I D, 0x87.

Example: Enable Both trigger Inputs.

Command: <SOP><0x07><0x01><0x03><EOP>
Response: <SOP><0x87><EOP>

3.8 USB_Reset Command

The USB_Reset command issues a bus Reset on the root port. In Automatic Mode, this command will result in all device connections being terminated. Asynchronous Responses indicating device disconnects will not be generated. The Controller should assume that all connected devices have been disconnected as a result of this command. The format of this command is:

<SOP><CMD_USB_Reset><EOP>

<CMD_USB_Reset> is the command ID, 0x08.

When the Reset has been completed, the Root 1 will respond with:

<SOP><RESP_USB_Reset><EOP>

<RESP_USB_Reset> is the response ID, 0x88.

3.9 DevTrans Command

This command allows a discrete transaction to be initiated to a user-defined Address and Endpoint. Unlike DevRqst, which completes an entire control transfer in a single command, DevTrans deals with only a single transaction at a time. That is, it issues one IN, OUT or Setup and accepts the device's immediate response. DevTrans is typically used for non-control transfers, such as INs and OUTs to bulk, interrupt or isochronous data endpoints. Control transfers are much more easily handled by the DevRqst command. Because it offers packet level control, however, it may be desirable to use DevTrans to generate control transfers in special situations, such as where the user desires to manage the timing of the transactions within the transfer.

Chapter 8 of the USB Specification describes the construction and handling of transactions on the USB. Note that CRC's and PID check bits are generated and checked automatically by the Root 1, and are not required to be provided by the user. Likewise, Root 1 provides the necessary handshaking to conduct the packet on the bus. The user is required only to provide the address, endpoint, transaction PID and data. Valid PIDs for the DevTrans command are IN, OUT and SETUP. The PID values can be found in Table 8-1 of the USB Specification.

Because of the timing variations associated with the serial communications mechanism, new transactions are typically issued by Root 1 immediately following a SOF. This results in a typical throughput of one transaction per (1mS) frame. The DevTrans command provides a mechanism which allows the user to override this convention, allowing multiple transactions per frame to be conducted. This feature is available only in Script mode, and requires the user to ensure that the transactions fit within the current frame without overrun.

The format of the DevTrans command is:

<SOP><CMD_DevTrans><Address><Endpoint><PID><Control>{<DataPID><OutputData>}<EOP>.

<CMD_DevTrans> is the DevTrans command ID, 0x09.

<Address> is the USB address, 0~127, of the target device.

<Endpoint> is the target endpoint for the request.

<PID> is the 4-bit data PID (i.e., SETUP, IN or OUT) which will be used in the token packet.

The <Control> parameter indicates the direction of the data portion of the transaction, indicates whether the transaction is targeted to a low-speed or full-speed device, and identifies isochronous transactions. Figure 3-1 details the DevTrans Control byte.

Bit 0 of the <Control> parameter is the <direction> bit, and indicates whether the direction of the data portion of the transaction will be In (direction = 0) or Out (direction = 1). If <direction> = 1, such as it would for a SETUP or OUT transaction, <DataPID> will be the data PID, DATA0 or DATA1, to be transmitted with the data packet. Note that Root 1 performs no checking for data toggling when using the DevTrans command. This is the responsibility of the user's application. <OutputData> is the data packet to be transmitted in conjunction with a Setup or Out command, and may be from 0 to 63 bytes in length. <DataPID> and <OutputData> are not included if <Direction> = 0.

Bit 1 of the <Control> parameter is the <FullSpeed> bit. If this bit is set (=1), it indicates to the Root 1 that the function being addressed is a full-speed device. If the FullSpeed bit is =0 (low speed), and the device attached to the Root 1's root port is a high speed device, Root 1 will assume that the device being addressed is a low-speed device attached to the downstream port of a hub, and will generate a low-speed preamble (PRE PID) to indicate to the hub and other high-speed devices downstream that a low-speed packet follows.

Bit 2 of the <Control> parameter is the <Isoch> bit. This bit should be set (=1) if the transaction is an isochronous data packet, in which case no packet handshaking is performed.

Bit 7 of the <Control> parameter is the <Immed> bit. When set, this bit causes the DevTrans packet to be issued immediately, without waiting for the next SOF. This bit is valid only in a RootScript.

The device transaction is conducted on the bus, and the Root 1 responds with the following:

<SOP><RESP_DevTrans><RespStatus>{Device Response}<EOP>

<RESP_DevTrans> is the DevRqst response ID, 0x89. <RespStatus> is the response status as indicated in Table 3-1. In the case of an IN, the {Device Response} is the data packet returned by the target device. In the case of an Output command (Out or Setup), the device's response to the command is completely defined in the RespStatus byte, and {Device Response} is not present.

Care must be taken in using the DevTrans command in conjunction with Automatic Mode, since it allows conditions to be created which will undermine the Root 1's knowledge of the state of downstream devices, causing it to potentially mishandle packets to or from the device.

Under normal circumstances (Immed bit = 0), Root 1 always begins a new bus transaction immediately after a SOF. This ensures that the transaction will not overrun the current frame. The <Immed> bit allows

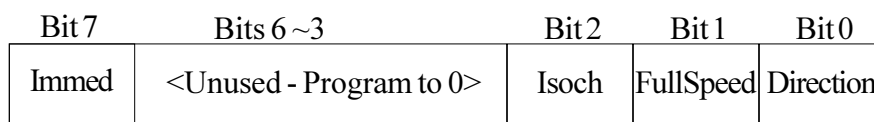


Figure 3-1 DevTrans Control Byte

RootScripts to circumvent this safeguard. In using the <Immed> bit, the script writer must take responsibility for avoiding the condition in which traffic is generated on the bus which overruns the end of the frame. In general, the Root 1 has an overhead of approximately 230uS associated with each bus transaction. This includes the time required to fetch a new script command, setup and conduct the transaction and verify the response, exclusive of the actual data transmission. One could, for example, expect to issue up to three (64-byte) bulk transfers per frame as follows:

```
{Packet 1} Immed = 0; DevTrans - OUT - 64 bytes // sync to SOF
{Packet 2} Immed = 1; DevTrans - OUT - 64 bytes //transfer immediately
{Packet 3} Immed = 1; DevTrans - OUT - 64 bytes //transfer immediately
{Packet 4} Immed = 0; DevTrans - OUT - 64 bytes //resync to SOF
{Packet 5} Immed = 1; DevTrans - OUT - 64 bytes //transfer immediately
.
.
```

In this example, each transfer accounts for approximately 62uS of data transfer time (header packet + data packet) plus 230uS of Root 1 overhead, or approximately 292uS per transfer. After the third transfer, it is necessary to wait again until the start of the next frame.

3.10 Data Port and Trigger Out

The DataPort command is used to strobe data onto the external data port. The data can be written using one of two methods: direct or masked. Using the direct method, the user simply supplies an 8-bit data value which is written to the data port. Using the masked method, the user supplies an 8-bit AND mask, and an 8-bit OR mask. The AND mask is first applied to the current data port data value. The result of this operation is then OR'd with the OR mask, and the resultant data is written to the data port.

Direct Method: DataPort <- NewData

Masked Method: DataPort <- ((DataPort & ANDMask) | ORMask)

The data written to the data port will be driven onto the Root 1's parallel data lines, and the TrigOut strobe will be toggled as described in section 2.4. Note that, while the TrigOut strobe only toggles once per execution of the command, the value written to the data port will continue to be driven on the outputs until a new value is written.

This command is useful in generating strobed data outputs, such as may be used as an input to a logic analyzer, using the TrigOut strobe as a clock. The command is also useful in generating static outputs on the data lines to be used, with the proper interface circuitry, in driving external indicators or controlling external actuators.

The format of the direct command is:

<SOP><CMD_DataPort><Data_Value><EOP>

<CMD_DataPort> is the command ID, 0x0A.

<Data_Value> is a one byte value to be written to the data port.

The format of the masked command is:

<SOP><CMD_DataPort><AND_Mask><OR_Mask><EOP>

<CMD_DataPort> is the command ID, 0x0A.

<AND_Mask> is a one byte AND mask.

<OR_Mask> is a one byte OR mask.

Root 1 distinguishes between the direct and masked methods simply based on the length of the command string. In either case, the Root 1 will respond with :

<SOP><RESP_DataPort><EOP>

<RESP_DataPort> is the response ID, 0x8A.

Example: Direct Method

Command: <SOP><0x0A><0x55><EOP> 'Parallel data output after command = 0x55
Response: <SOP><0x8A><EOP>

Example: Masked Method. Value of data port prior to command = 0x0F

Command: <SOP><0x0A><0x0C><0x81><EOP> 'Parallel data output after command = 0x8D
Response: <SOP><0x8A><EOP>

3.11 Read Root Hub Status

The Get_RootStatus command returns a byte containing various information regarding the status of the Root hub. The format of the command is:

<SOP><CMD_Get_RootStatus><EOP>

<CMD_Get_RootStatus> is the command ID, 0x0B.

The Root 1 will respond with :

<SOP><RESP_Get_RootStatus><Data_Value><EOP>

<RESP_Get_RootStatus> is the response ID, 0x8B.

The one byte value returned in <Data_Value> is encoded as follows:

Bits 1:0 - Connect Status
 0:0 No devices connected
 0:1 Low speed device connected to Root port
 1:0 Full speed device connected to Root port
 1:1 <Unused>

Bit 2 - Power State
 0 Root Port Power Off
 1 Root Port Power On

Bit 3 - Suspend State
 0 Root Port Active (not suspended)
 1 Root Port Suspended

Bit 4- Root Port Enabled State

0	Root Port Disabled
1	Root Port Enabled

Bits 7:5 - <Unused>

The Power and Suspend State values returned by this command can be useful in the event that these factors change as a result of something other than a command from the test controller. For instance, in the event of an overcurrent condition on the Root port, Vbus power will be switched off. Similarly, if the Root port is Suspended by command from the controller, it can be brought out of Suspend by Resume signaling from a downstream device.

Bit 4, the Root Port Enabled status, will typically be true (enabled) when power is on and a device is connected and operating properly. Certain circumstances, such as a device babbling on the bus, will cause the Root 1 host controller to disable the root port. In this instance, it is necessary to disable then reenale root port power to reenale traffic on the root port.

4. Asynchronous Responses

Asynchronous responses are generated by Root 1 to call attention to asynchronous events. The Transmission codes used in Asynchronous Responses are unique, allowing them to be easily differentiated from command responses.

4.1 Connect Event

When a new device connect is detected in Automatic mode, Root 1 resets the device, enumerates it and places it in its initial configuration. At this time, a Connect Event is issued to the Controller. The address assigned to each device is predictable, to facilitate the writing of test applications. A device plugged into the Root port will always be assigned address 2 by the host controller in Automatic mode. If an external hub is connected to the Root port, the address of the hub will be 2, and devices connected to the downstream ports of the hub will be enumerated beginning with the hub's downstream port 0 at address 3, port 1 at address 4, etc. Note that the address assignment in Automatic mode is determined by the physical port of the hub, regardless of the order in which devices are attached. For instance, a devices attached to physical ports 1 and 2 of an external hub will be assigned addresses 4 and 5, respectively, regardless of the order in which the devices are connected. Note that Root 1 does not support multiple levels of hubs. Hub support is limited to the hub connected directly to Root 1.

When the device subsequently disconnects, a Connect Event is issued to the Controller indicating that the device has disconnected. The disconnecting device is identified by the address which was assigned to it at connect time.

The format of the Connect Event is:

<SOP><RESP_Connect><Action><Address> {<Device Class><Vendor ID><Product ID>}<EOP>

<RESP_Connect> is the response ID, 0x90.

<Action> is a byte value 0x0 for a connect, or 0x1 for a disconnect.

<Address> is the USB address assigned to the device by the host controller.

<Device Class>, <Vendor ID> and <Product ID> are provided only in the case of a Connect, and are taken directly from the device's Device Descriptor.

<Vendor ID> and <Product ID> are two-byte values, transmitted LSB first. <Device Class> is a 1-byte value.

4.2 Status Event

A Status Event is an asynchronous response generated by the Root 1 in Automatic Mode to inform the Controller of a status change, other than a connect or disconnect, on the bus. This type of event is typically generated as the result of polling the status-change endpoint of a hub on the root port. The format of the re-

sponse is:

<SOP><RESP_Status><Hub Address><Port #><Status_Value><EOP>

<RESP_Status> is the response ID, 0x91. <Hub Address> is the enumerated address of the hub reporting the status, and <Port #> is the number of the port on that hub reporting the status change. <Status_Value> is the 16-bit Port Status value elicited from the hub for that port, per the USB Specification Rev 1.1, Table 11-5.

4.3 Data Event

In Automatic Mode, Root 1 automatically polls any interrupt IN endpoints defined in initial configuration for each device. Examples are a standard HID keyboard's data endpoint or a hub's status change endpoint. A Data Event is an asynchronous response generated by the Root 1 in Automatic Mode to inform the Controller that data has been returned by a downstream device in response to this polling. The format of the response is:

<SOP><RESP_Data><Address><Endpoint>{Data}<EOP>

<RESP_Data> is the response ID, 0x92.

<Address> and <Endpoint> are 1-byte values indicating the address and endpoint reporting the data

{Data} is the data reported from the device.

4.4 Error Event

An Error Event is an asynchronous response generated by the Root 1 in Automatic Mode to inform the Controller that an error was encountered in communicating with a downstream device. The format of the response is:

<SOP><RESP_Error><Address><Endpoint><Error><EOP>

<RESP_Error> is the response ID, 0x93.

<Address> and <Endpoint> are 1-byte values indicating the address and endpoint on which the error was encountered.

<Error> is a 1-byte value indicating the type of error encountered as shown in Table 3-1.

The majority of the error status responses are self-explanatory. The following paragraphs detail those less obvious responses.

A *PID Error* indicates that a received PID was either an invalid value, or its check bits did not match.

A *Babble Error* is returned if a device begins transmitting on the bus unexpectedly, or continues to transmit after an EOP.

The *Short Packet* response indicates that a received data packet was shorter than the shortest valid packet. The shortest valid data packet consists of a SYNC, a data PID and a 16-bit data CRC. Any received data packet which is shorter than this minimum will be flagged with this error.

The *Configuration Error* response is returned in Automatic Mode only, and indicates that Root 1 was not able to successfully parse the Device, Interface and Endpoint Descriptors of an attached device in order to configure the device. This typically indicates a descriptor formatting error in the device.

4.5 Root Fail Event

Root Fail is an asynchronous response generated by the Root 1 in response to a failure condition on the root port itself. Currently, the only Root Fail condition is an overcurrent detection on the root port. The format of the response is:

<SOP><RESP_Fail><Error><EOP>

<RESP_Fail> is the response ID, 0x94.

<Error> is an error code indicating the nature of the fault. Currently, the only defined <Error> value is 0x01, indicating a root port overcurrent. Once this condition is detected and reported, power to the root port is disabled automatically by the Root 1.

4.6 Command Error

Command Error is an asynchronous response generated by the Root 1 in response to an unrecognized or improperly-formatted command. After generating this response, Root 1 will seek for a new <SOP> sequence to begin parsing the next command. The format of the Command Error response is:

<SOP><RESP_CmdError><EOP>

<RESP_CmdError> is the response ID, 0x95.

4.7 Trigger Event

The Trigger Event is an asynchronous response generated by the Root 1 in response to an external trigger on the TrigIn0 or TrigIn1 lines on the Control connector. Trigger activity will only be recognized, and this response will only be generated, if the trigger input has been enabled using the Root_Config command. The format of the response is:

<SOP><RESP_Trigger><Source><EOP>

<RESP_Trigger> is the response ID, 0x96.

<Source> is a value 0x0 or 0x1 indicating the trigger source (TrigIn0 or TrigIn1, respectively).

5. RootScript

Command execution on the Root 1 as described thus far is referred to as "immediate mode" execution. Each command is received, executed immediately, and its response is returned to the Controller. RootScript provides a non-immediate mode of execution, or "script mode", in which a sequence of Root 1 and RootScript commands (a script) is loaded into Root 1, then executed as a program. Scripting allows commands to be executed in much more rapid succession than is possible in immediate mode. Scripts are also useful in that they can provide a more autonomous testing mechanism - one which does not rely on constant interaction with the test host.

The RootScript program language incorporates the normal set of Root 1 commands, plus additional RootScript commands which are available only in RootScript programs. Some RootScript commands are related to the definition of the script itself (e.g., program start and end commands) and some are included to provide simple program flow control such as branching and conditional execution.

A RootScript program is loaded into the Root 1 as a sequence of Root 1 and RootScript commands framed between a *Program* command and an *RS_End* command. Once loaded, program execution is initiated by a *Run* command. Each command in a RootScript is assigned an index, or line number, based on the order of the commands in the script. The first command in the script is Index #0, the second Index #1, etc.

Unlike normal Root 1 commands, which may be used either in immediate mode or in a RootScript, RootScript commands are not valid in immediate mode - they may be used only in a RootScript.

Like normal Root 1 commands, RootScript commands are framed between <SOP> and <EOP> markers, and consist of a 1-byte command ID followed, if necessary, by parameters. A RootScript program may consist of up to 1000 commands, not including the Program command, but including *RS_End*.

5.1 Program Command

Program is an immediate-mode command used to initiate loading of a new script into Root 1. Receipt of the Program command will cause any script currently existing in Root 1 memory to be erased, and will place Root 1 in "program load" mode. The format of the Program command is:

<SOP><CMD_Program><EOP>

<CMD_Program> is the command ID, 0x0C

The Root 1 will respond with:

<SOP><RESP_Program><EOP>

<RESP_Program> is the response ID, 0x8C

Once a Program command has been received and acknowledged by Root 1, all subsequent commands received from the host controller will be considered part of the newly defined script, until either an *RS_End* command is encountered, or a new Program command is received. Each command, as it is received, is checked for syntax, assigned an index, and stored. Each script command is acknowledged as it is received, as follows:

<SOP><RESP_Script><Idx><Command_ID><EOP>

<RESP_Script> is a load-acknowledge byte of value 0xA0.

<Command_ID> is the command byte of the acknowledged command. For instance, if the script command being acknowledged is a DevTrans command, then <Command_ID> will be CMD_DevTrans.

<Idx> is a 16-bit value indicating the index at which the command is loaded, and will increment by 1 for each subsequent command in the script.

If a new Program command is received prior to an RS_End, the script in process of being loaded will be erased, and recording will begin again at index 0.

Commands are checked during script loading. If a command error is encountered during program load, Root 1 will respond to the offending command with RESP_CmdError, rather than a RESP_Script. If the program being loaded exceeds the RootScript program limit of 1000 commands, or if the script being loaded exceeds the available program memory in Root 1 (approximately 180Kbytes), Root 1 will respond with a RESP_ScriptOvfl response code, 0x97. Once a Command Error or Script Overflow has been generated during a program load, Root 1 will respond to all commands with RESP_CmdError until either an RS_End is encountered or until a new Program command is received. Any command error encountered during a script download will result in no valid script being available for execution.

A very simple RootScript loading sequence, including the Program and RS_End commands, is shown below. Root 1 responses are indicated in color.

```
<SOP><CMD_Program><EOP>  
<SOP><RESP_Program><EOP>  
<SOP><CMD_VCC><0x64><EOP>  
<SOP><RESP_Script><0x0000><CMD_VCC><EOP>  
<SOP><CMD_Power><0x01><EOP>  
<SOP><RESP_Script><0x0001><CMD_Power><EOP>  
<SOP><RS_End><EOP>  
<SOP><RESP_Script><0x0002><RS_End><EOP>
```

When executed, this RootScript would cause the root port Vcc to be set to 5.00V, and root port Vbus power to be switched ON.

Any RootScript command (program-mode command) received by Root 1 when not in program-load mode will elicit a Command Error response.

5.2 RS_End

The RS_End command terminates the "program load" mode, completing the load of a RootScript into Root 1 memory. The format of the RS_End command is:

```
<SOP><RS_End><EOP>
```

<RS_End> is the command ID, 0x21.

If Root 1 is in program mode, it will respond with the acknowledge:

```
<SOP><RESP_Load><RS_End><EOP>
```

By definition, it is possible to have only one RS_End command in a script.

5.3 Run

Once a script has been loaded into Root 1, it may be invoked by the *Run* command. Run is an immediate mode command; it is not part of the script itself, and can be issued any time after a script has been successfully loaded. The format of this command is:

<SOP><CMD_Run><EOP>

<CMD_Run> is the command ID, 0x0D.

Before beginning execution of the script, the Root 1 will respond to the *Run* command as follows:

<SOP><RESP_Run><EOP>

<RESP_Run> is the response ID, 0x8D.

If no script is currently loaded, the *Run* command will return a Command Error response.

Once a script has begun execution, it will execute either to completion (RS_End) or until it is terminated by another command from the test host. ANY traffic received from the Controller during script execution will terminate the script immediately.

5.4 Responses during Script Execution

Root 1 provides two modes of response handling during script execution: Full-Response mode and Quiet mode. In Full-Response mode, each Root 1 command executed as part of the script will return its normal response over the serial port to the test controller, prepended with a RESP_Script byte and the program index. In addition, encountering an RS_End command will cause the Root 1 to generate a script termination message consisting of the RESP_Script byte, the program index, the RESP_End byte and a termination index. The termination index is the index of the last instruction executed prior to the RS_End. For example, execution of the sample script in section 5.1 would result in the following messages being generated in Full Response mode:

<SOP><RESP_Script><0x0000><RESP_VCC><EOP>
<SOP><RESP_Script><0x0001><RESP_Power><EOP>
<SOP><RESP_Script><0x0002><RESP_End><0x0001><EOP>

<RESP_Script> is the script-response ID byte (0xA0), indicating that the message is a script-command response. The index words indicate the program index associated with the command that generated the response. Byte RESP_End, value 0xA1, indicates execution of the *RS_End* command, after which Root 1 is returned to immediate mode.

In Quiet mode, all responses associated with Root 1 commands executed as part of the script will be suppressed, with the exception of RS_Message and RESP_End responses. Execution of the sample script in section 3.1 running in Quiet mode would result in only a RESP_End response, with the VCC and Power responses being suppressed. The advantage to quiet mode is that it greatly reduces the amount of traffic which the host controller must handle. In addition, although the Root 1 buffers its serial transmit data, even a simple script executing at high speed can generate a substantial amount of response traffic, most of which the host controller does not require access to, and this traffic will eventually impact the execution speed of the script. The termination index in the RESP_End response can be useful in quiet mode as an indication of which code path led to script termination.

Note that only normal Root 1 commands and the *RS_End* commands generate execution responses, even in full-response mode. RootScript commands (with the exception of *RS_End* and *RS_Message*) do not generate responses.

5.4.1 ResponseMode Command

The response mode during script execution is determined using the ResponseMode command. The format of this command is as follows:

<SOP><RS_Response><Mode><EOP>

<RS_Response> is the RootScript ResponseMode command ID, 0x22.

<Mode> is a 1-byte parameter = 0 for Full Response mode or =1 for Quiet mode.

As a RootScript command, ResponseMode is not available in immediate mode. The *RS_Run* command sets the response mode to Quiet, so it must be explicitly changed using a ResponseMode command if Full Response execution is desired. The response mode may be modified dynamically, switching between quiet mode or full-response mode as necessary to report to the test controller only that information which is desired by the script writer.

5.5 Conditions and Flow Control

RootScript provides a set of commands relating to program flow control and the evaluation of certain conditions which may be used by the script writer to control program flow. Program flow is controlled by allowing the script to transfer execution control to a new execution index, either explicitly as in the case of a Goto, or implicitly as in the case of Return. Program indexes are always represented in RootScripts as 2-byte values stored most-significant byte first (big-endian). These commands are described in the following sections.

5.5.1 RS_Goto Command

The *RS_Goto* command is the simplest mechanism available in RootScript to change program flow, allowing program control to be transferred to a new program index. The format of the command is:

<SOP><RS_Goto><Idx><EOP>

<RS_Goto> is the command ID, 0x23.

<Idx> is the 16-bit index to which control is to be transferred.

An index value which is outside the scope of the script (i.e., greater than the index of the *RS_End* command) will cause control to transfer to *RS_End*. In order to provide a simple mechanism for the script writer to indicate "goto end", index 0xFFFF is reserved as a forced end-of-script. That is, "*RS_Goto* 0xFFFF" will always terminate the script. Note that the index is represented MSB first, so an *RS_Goto* index 0x0002 would be represented as:

<SOP><RS_Goto><0x0><0x2><EOP>

5.5.2 RS_IfCommand

The RS_If command is used to conditionally transfer program control to a new program index if a certain condition is found to be true. The conditions upon which RS_If can operate are given in Table 3-1. The scope of RS_If is limited to the most-recently completed USB transaction. The format of the command is:

<SOP><RS_If><Cond><Idx><EOP>

<RS_If> is the command ID, 0x24.

<Idx> is the 16-bit index to which control will conditionally be transferred.

Like RS_Goto, RS_If identifies index 0xFFFF as a default "end-of-script" index. <Cond> is a one-byte parameter indicating the condition to be checked. Valid <Cond> values are those listed in Table 3-1.

5.5.3 RS_Check and RS_Cond

The RS_Check command is used to transfer program control to a new index depending selected conditions. The conditions upon which RS_Check depends are programmed individually by the RS_Cond command.

Command RS_Cond is used to set a program index which will be associated with each of the possible conditions, and to which control will be transferred upon execution of the RS_Check command if the condition is true. Once an index is set by RS_Cond for a given condition, the index will not change unless a new RS_Cond command is issued for that condition. The format of the RS_Cond command is:

<SOP><RS_Cond><Cond><Idx><State><EOP>

<RS_Cond> is the command ID, 0x25.

<Idx> is the 16-bit program index to which control will be passed by RS_Check if the condition is true. Again, a program index of 0xFFFF is used as default end-of-script.

<Cond> is the condition 1-byte ID, as shown below:

<u><Cond></u>	<u>Condition</u>
0	Connect
1	Disconnect
2	<Not Used>
3	Resume
4	TriggerIn0
5	TriggerIn1
6	Timer Timeout

<State> is a 1-byte value =1 to enable the condition, or =0 to disable it. If the condition is being disabled, the value of <idx> is unimportant. All conditions default to the OFF state upon execution of RS_Run.

When the RS_Check command is executed, it polls the enabled conditions until one of them is found to be true, at which time control is transferred to the associated index. Note that execution of an RS_Check command with no conditions enabled will cause RS_Check to wait indefinitely. The conditions are polled by RS_Check in the order they are listed in the above table, so an enabled Connect condition will take precedence

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<Unused>	Timeout	Clear TrigIn1	Clear TrigIn0	Resume	<Unused>	Dis-Connect	Connect

Figure 5-1 RS_Check <Inits> Byte

over an enabled Timer Timeout, for instance.

The format of the RS_Check command is:
 <SOP><RS_Check><Inits><EOP>

<RS_Check> is the command ID byte, 0x26. <Inits> is a one-byte value which indicates whether or not certain latched conditions should be cleared prior to execution of RS_Check. The bits of <Inits> correspond to the RS_Cond conditions listed above (See figure 5-1), however only the TriggerIn0, TriggerOut0 and SOF conditions are latched conditions, so only these conditions are affected by the <Inits> byte. If for instance, a TriggerIn0 has been latched by the Root 1 prior to the execution of RS_Check, and its associated bit in the <Inits> byte is set, the latched TriggerIn0 condition will be cleared prior to execution of RS_Check - so RS_Check will wait for the next TriggerIn0 transition before succeeding for TriggerIn0. If a TriggerIn0 condition has been latched but is *not* cleared by <Inits>, RS_Check will succeed for TriggerIn0 immediately.

In addition, each time RS_Check succeeds on a latched condition (SOF, TriggerIn0 or TriggerIn1) the condition is cleared. Two subsequent RS_Check's for SOF, for instance, would impose a delay of one SOF time between detections. Likewise, two subsequent RS_Check's for TriggerIn0 would require two transitions of the trigger line to traverse.

5.5.4 RS_Timer

The RS_Timer command is used to set a counter which is decremented by the Root 1's 1mS-tick timer. Upon reaching 0, the counter will create a timer timeout condition which may be used by RS_Cond and RS_Check. Once the counter reaches its timeout condition, it will remain at 0 until reloaded by the RS_Timer command. The format of this command is:

<SOP><RS_Timer><Count><EOP>

<RS_Timer> is the command ID byte, 0x27.

<Count> is a 32-bit value which will be loaded into the timeout counter. <Count> is represented most significant byte first (bigendian).

Loading the timer with a 0 will result in a timeout condition being True. Note that, as <Count> is a 32-bit value, extremely long timeout periods, on the order of 50 days, can be generated.

5.5.5 RS_Call and RS_Return

RS_Call and RS_Return together provide call return functionality in RootScript. RS_Call pushes the index

of the command immediately following it onto a program stack, then transfers program control to the specified index. The format of this command is:

<SOP><RS_Call><Idx><EOP>

<RS_Call> is the command ID, 0x29.

<Idx> is the 16-bit index to which control is to be transferred. An index value which is outside the scope of the script (i.e., greater than the index of the RS_End command) will cause control to transfer to RS_End.

Execution of an RS_Return command causes the last index pushed onto the program stack to be popped, and control to be transferred to that index. The format of the command is:

<SOP><RS_Return><EOP>

<RS_Return> is the command ID, 0x2A.

RootScript provides a maximum stack depth of 256 calls. In the event that the stack is overflowed (the number of RS_Calls exceeds the number of RS_Returns by more than 256 at any given time) or underflowed (more RS_Returns than RS_Calls are executed), the script will terminate immediately at RS_End. The termination index will point to the RS_Call command which caused the overflow, or the RS_Return command which caused the underflow.

5.6 RS_Message

The RS_Message command forces a response to the test controller, regardless of the response mode. The format of the RS_Message command is:

<SOP><RS_Message><data><EOP>

<RS_Message> is the command ID, 0x28. <data> is a data string of 0 to 63 binary bytes which will be included in the forced response. The format of the RS_Message forced response is:

<SOP><RESP_Script><Idx><RESP_Message><Timer><data><EOP>

<RESP_Script> is the normal script-response ID byte.

<Idx> is the index of the RS_Message command responsible for the message.

<RESP_Message> is an RS_Message ID byte, 0xA8.

<Timer> is a 32-bit value indicating the remaining count in the timeout counter.

<data> is the data programmed with the RS_Message command.

6. Script Management

Scripts are normally loaded into RAM on the Root 1 using the Program command, and executed directly from RAM using the Run command. Root 1 offers the option, however, to load a script into on board Flash memory, and enable that script to be executed after power up rather than entering Immediate Mode. This is referred to as a "default script". The default script is loaded into Root 1 RAM in the normal fashion, using the Program command. Once the script is successfully loaded, it can be burned into Flash using the Flash command. The Flash command can also be used to enable and disable the default script, without removing it from Flash, and to determine the status of the default script.

6.1 The Default Script

If a default script is present in Flash, and is enabled, Root 1 will, after power up initialization, load the default script into RAM and execute it. If the script terminates, or if any traffic is received on the serial port, Root 1 will abort the script and return to Immediate Mode operation.

6.2 The Flash Command

The Flash command is an immediate mode command used to manage the programming and status of the default script. It allows the script to be written to Flash, allows it to be enabled or disabled, and allows its status to be queried. The format of the Flash command is:

<SOP><CMD_Flash><Script_ID><Action>{<Name>}<EOP>

<CMD_Flash> is the Flash command ID, 0x31.

<Script_ID> is the script identifier. The only valid value is 0, indicating the default script.

<Action> indicates the action to be taken by the Flash command. Valid Actions are:

- 0: Disable Script
- 1: Enable Script
- 2: Burn script in RAM
- 3: Return status of Script

<Name> is present only if <Action>=2. <Name> must be exactly 8 bytes in length and, while it may contain any value, it is typically used to hold an ASCII description of the default script.

The Flash-Enable, Flash-Disable and Flash-Burn commands, if successful, return the following:

<SOP><RESP_Flash><EOP>

<RESP_Flash> is the Flash command response code, 0xB1.

If there is no default script programmed, the Flash-Enable and Flash-Disable commands have no effect, and a Command Error event is returned.

In order for a Flash-Burn command to be successful, a script must have been loaded into Root 1 RAM

using the Program command. In addition, the script must fit in the Flash memory area available for default script storage (approximately 160Kbytes). If either of these conditions is not met, a Command Error event is returned.

The Flash-Status command returns the following:

<SOP><RESP_Flash><Status>{<Name>}<EOP>

<RESP_Flash> is the Flash command response code, 0xB1.

<Status> is a byte indicating the status of the default script, as follows:

0: Default Script Disabled

1: Default Script Enabled

0xFF: Default Script Undefined

A status of Disabled indicates that a script is programmed into Flash, but is currently disabled, versus a status of Undefined, which indicates that no default script is programmed in Flash.

<Name> is returned only if <Status> is = 0 or 1, and contains the 8-byte Name field which was programmed with the script.

7. Power

The Root 1 requires an external 16~24 VDC power supply which is capable of providing 13W. The power connector pinout is shown in Figure 7-1. The mating connector and terminals required for connection to the Root 1 power connector is as follows:

Molex P.N. 03-06-2044 Plug Housing (1 required)

Molex P.N. 02-06-2103 .062" Male crimp terminal (3 required)

RPM offers a universal input (100~240VAC 50/60Hz) wall-mount power supply as well as a universal input (90~260VAC 50/60Hz) miniature desktop supply, both of which attach directly to the Root 1 power connector. These are RPM Part #'s R1-PSW and R1-PSD, respectively. R1-PSW has US-style prongs for direct plug in to a wall outlet. R1-PSD has an IEC standard universal cord adapter for attachment of US or international power cords.

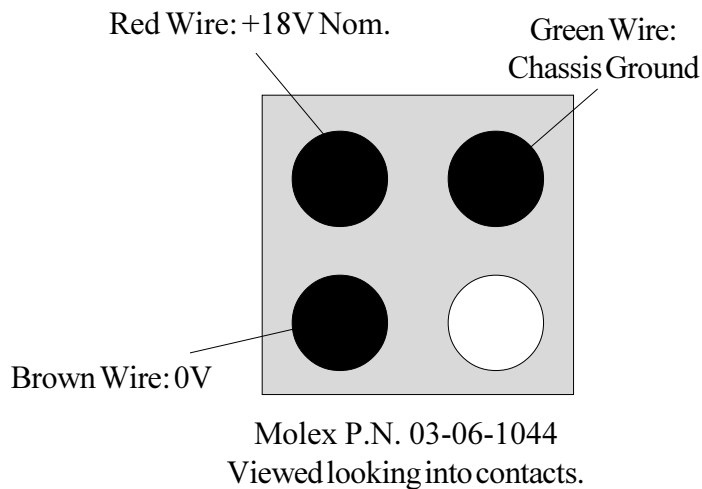


Figure 7-1 - Root-1 Power Connector Pinout