Si3482

POWER MANAGEMENT CONTROLLER

Features

- Enables use of smaller power supplies for up to 48-port PoE systems with Si3452 PSE interface ICs
- Can operate with or without a host
- Configuration save
- Pin-selectable SPI or UART interface
- Pin-selectable UART data rate
- Fully-compliant with IEEE 802.3 clause 33 for PoE including the 802.3at amendment for higher power (30 W, category 2 ports)
- Supports classification-based and LLDP power negotiation
- Supports individual port priority and port configuration
- Supports Power supply status from up to 3 power supplies
- 24 pin Quad flat pack package (4x4 mm)
  - 4x4 mm PCB footprint; RoHS complaint
- Extended operation range (–40 to +85 °C)

Applications

- Power over Ethernet Endpoint switches and Midspans for IEEE Std 802.3af and 802.3at
- Supports high-power PDs, such as:
  - Pan/Tilt/Zoom security cameras
  - 802.11n WAPs
  - Multi-band, multi-radio WAPs
- Security and RFID systems
- Industrial automation systems
- Networked audio
- IP Phone Systems and iPBXs
- Metropolitan area networked WAPs, cameras, and sensors
- WiMAX, ASN/BTS, and CPE/ODU systems

Description

The Si3452 is capable of delivering over 30 W per port, which means that, in a 24- or 48-port system, a very large power supply would have to be used to avoid overload. Typically, not all ports are used at full power; so, a smaller power supply in the range of 5 W per port can be used along with the Si3482 power management controller.

The Si3482 is a power manager intended for use with the Si3452/3 Power over Ethernet (PoE) controllers for power management of up to 48 ports with three power sources.

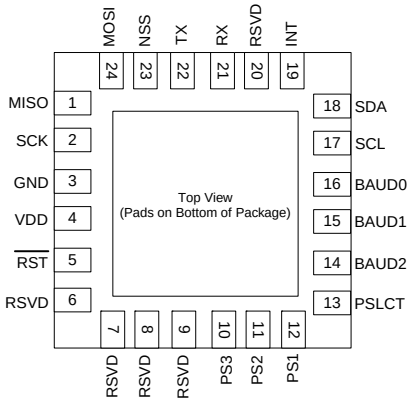
Use of the Si3482 power manager greatly simplifies system implementation of power management. The Si3482 power management controller is programmed via a SPI or UART interface to set the power supply capacity, the port power configuration (Category 1: 15.4 W, or high-power category 2: 30 W) ports, the port priority, the detection timing (Alternative A or Alternative B), and the fault recovery protocol. Once programmed, the configuration data can be saved, and the Si3482 can work without host intervention. If desired, port and overall status information is available and continuously updated.

The Si3482 uses the real-time overload and current monitoring capability of the Si3452 to manage provided power among up to 48 ports. Power management is selectable between grant-based or consumption-based in order to supply power to the greatest number of ports.

In high-reliability systems, multiple power supplies are often connected to provide redundancy, which further increases the power supply requirements. The Si3482 can manage up to three power supplies automatically enabling or disabling ports in priority order when required.

Pin Assignments

24-Pin QFN



See "5. Pin Descriptions" on page 31.

Functional Block Diagram

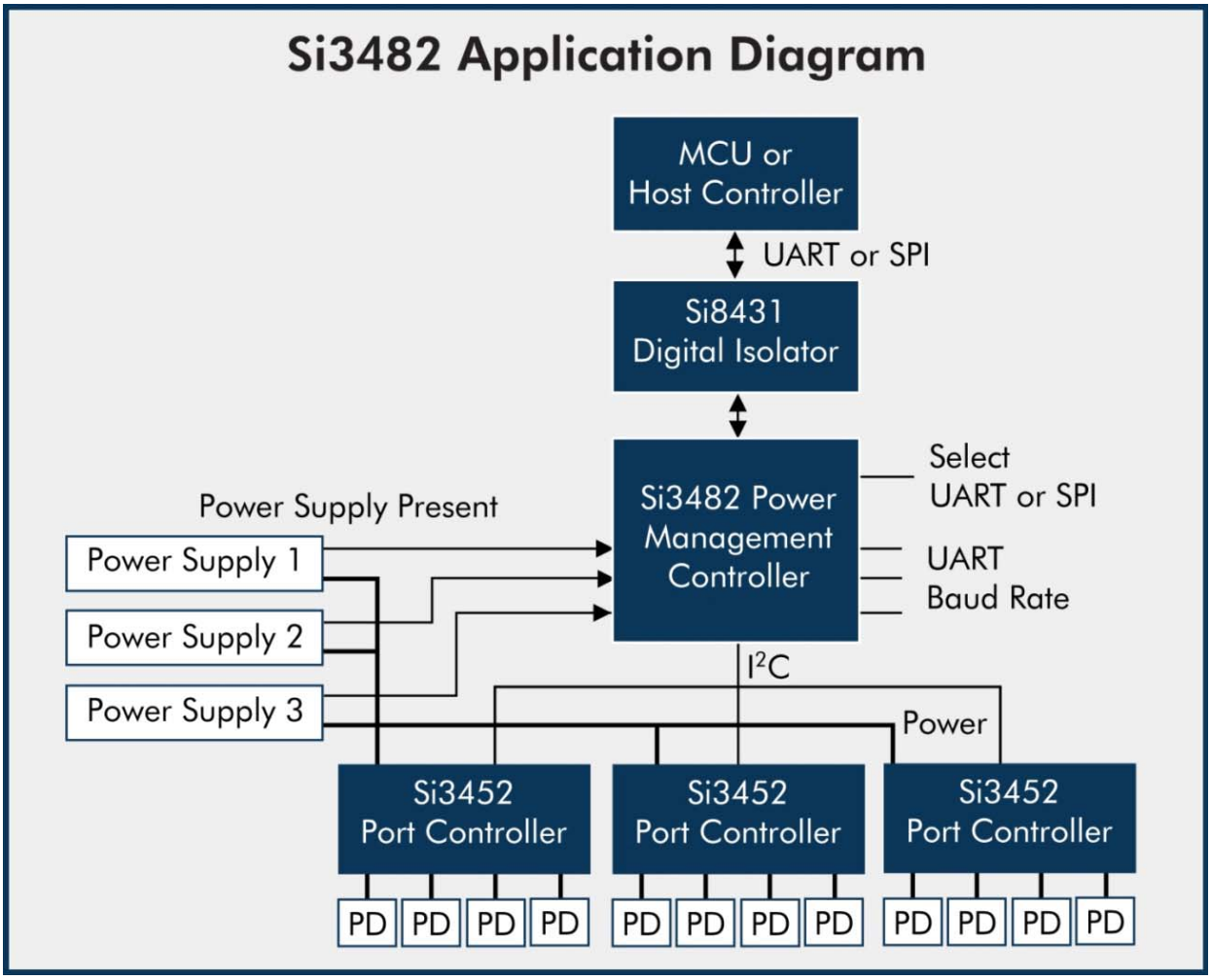


TABLE OF CONTENTS

1. Electrical Specifications

Table 1. Recommended Operating Conditions

Description	Symbol	Test Conditions	Min	Typ	Max	Units
Operating Temperature Range	T_A	No airflow	–40	—	85	°C
V_{DD} Supply Voltage	V_{DD}	All operating modes	2.7	—	3.6	V

Table 2. Absolute Maximum Ratings

Parameter	Conditions	Min	Typ	Max	Units
Ambient Temperature under Bias		–55	—	125	°C
Storage Temperature		–65	—	150	°C
Voltage on any I/O with Respect to GND	V_{DD}>2.2 V	–0.3	—	5.8	V
Voltage on V_{DD} with Respect to GND		–0.3	—	4.2	V

Note: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the devices at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 3. Electrical Characteristics

Description	Symbol	Test Conditions	Min	Typ	Max	Units
Input High	V_{IH}	Input pins: RST, SCK, MOSI, NSS, RX, PSn, BAUDn, SLCTIN, SCL, SDA	2.0	—	—	V
Input Low	V_{IL}		—	—	0.8	V
Input Leakage Current	I_{IL}		—	—	±1	uA
Output Low (MOSI, TX, SCL, and SDA)	V_{OL}	I_{OL} = 8.5 mA	—	—	0.6	V
Output High (MOSI, TX)	V_{OH}	I_{OH} = –3 mA	—	—	V_{DD}–0.7	V
V_{DD} Current	I_{DD}	VDD = 3.0 V* VDD = 3.6 V*	—	—	8.6 12.1	mA

***Note:** V_{DD} = 2.7 to 3.6 V, –40 to 85 °C unless otherwise noted.

Table 4. Timing Requirements

Description	Test Conditions	Min	Max	Units
SPI Timing Requirements (See Figure 1)				
T_{SE}	NSS Falling to First SCK Edge	84	—	ns
T_{SD}	Last SCK Edge to NSS Rising	84	—	ns
T_{SEZ}	NSS Falling to MISO Valid	—	168	ns
T_{SDZ}	NSS Rising to MISO High Z	—	168	ns
T_{CKH}	SCK High Time	210	—	ns
T_{CKL}	SCK Low Time	210	—	ns
T_{SIS}	MOSI Valid to SCK Sample Edge	84	—	ns
T_{SIH}	SCK Sample Edge to MOSI Change	84	—	ns
T_{SCH}	SCK Shift Edge to MISO Change	—	168	ns
F_{MAX}	Maximum SPI Clock Speed	—	1	MHz
UART Requirements (See Figure 2)				
ΔFTx	Deviation of Tx Transmit Speed from Pin-programmed Value.	–3	+3	%
ΔFRx	Deviation of Rx receive Speed from Pin-programmed Value.	–4	+4	%

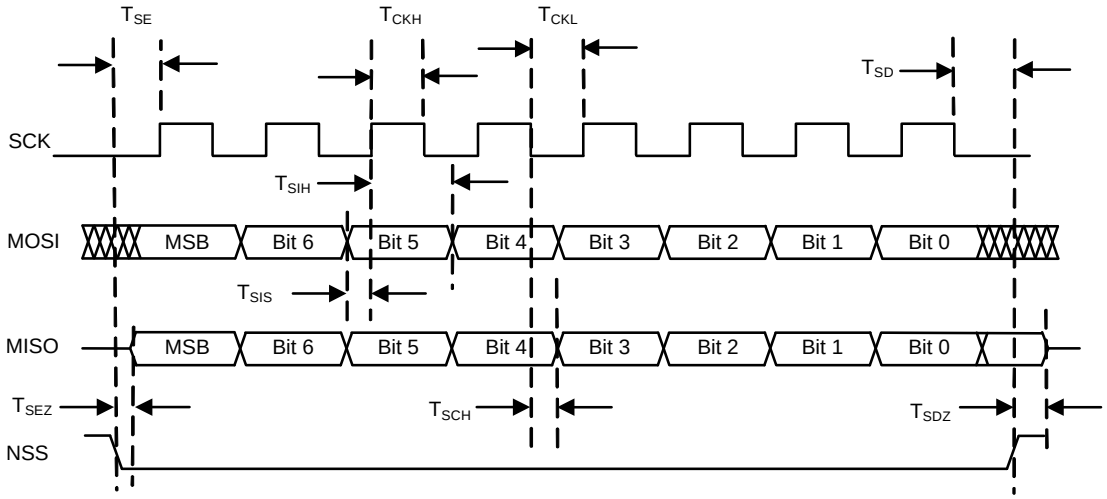


Figure 1. SPI Timing Diagram

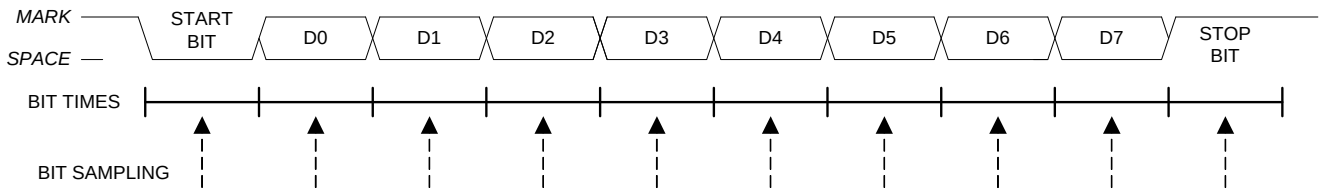


Figure 2. UART Timing Diagram

2. Functional Description

The Si3482 Power Management Controller is the central controller in a Silicon Labs Power over Ethernet (PoE) system. In a PoE system, power is provided by one or more power supplies and is consumed by one or more powered devices (PDs). The Si3482 decides which of the PDs can have power and monitors the amount of power consumed by each.

A host microcontroller unit (MCU) can configure the Si3482 and can query the status of the PDs and the power supplies. The Si3482 stores its configuration in internal flash memory. A host MCU uses a Universal Asynchronous Receiver Transmitter (UART) or a Serial Peripheral Interface (SPI) to communicate with the Si3482. Pins on the Si3482 select which host interface to use and which baud rate to use for the UART interface.

Power supplies may be inserted into bays. The Si3482 supports a system with up to three bays. Power supplies may be inserted or removed from the bays at any time. Each bay provides a signal to the Si3482 that indicates if a power supply is present in the bay. The outputs of the power supplies are ganged together to provide a single power source for the system.

The Si3482 manages a collection of Si3452 Port Controllers. The Si3482 supports a system with up to 12 Si3452s. Each Si3452 has four ports; so, a system may have up to 48 ports. The Si3452 performs low-level port functions, such as detecting and classifying PDs. The Si3482 has a global view of the system and manages power across all ports.

PDs are connected to ports on the Si3452s. PDs may be connected or disconnected from the ports at any time. When a PD is connected to a port, then the PD requests power from the port. The Si3482 determines the amount of power requested from the classification of the PD. If there is enough power remaining, the Si3482 grants the request; otherwise, the Si3482 denies the request.

The host may configure an optional power limit for each port. A power limit restricts the amount of power that the Si3482 grants to a port. If a power request is greater than the power limit, the Si3482 does not fully grant the request, but only grants the amount of the power limit.

The Si3482 supports Link Layer Discovery Protocol (LLDP) agents in the host. An LLDP agent can call a routine in the Si3482 to dynamically adjust the amount of power granted to a PD during the course of a connection.

Several PDs may be connected to a PoE system. The Si3482 may have granted different amounts of power to each PD, and each PD may be consuming different amounts of power. If a PD consumes more power than it is granted (port overload), the Si3482 turns off the PD.

There are two approaches that the Si3482 can take when granting requests for power. The granting policy can be grant-based or it can be consumption-based.

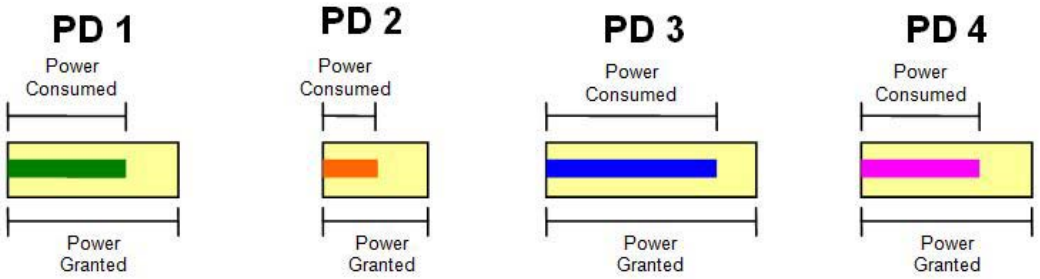


Figure 3. Powered Devices Example

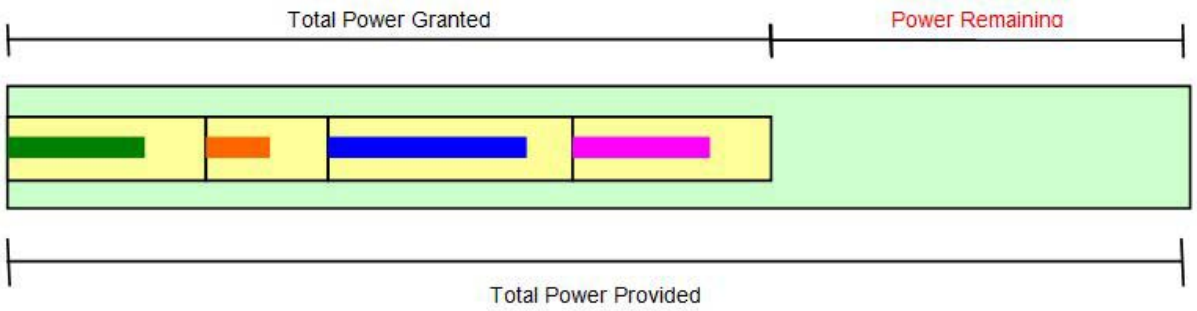


Figure 4. Grant Based Power Management

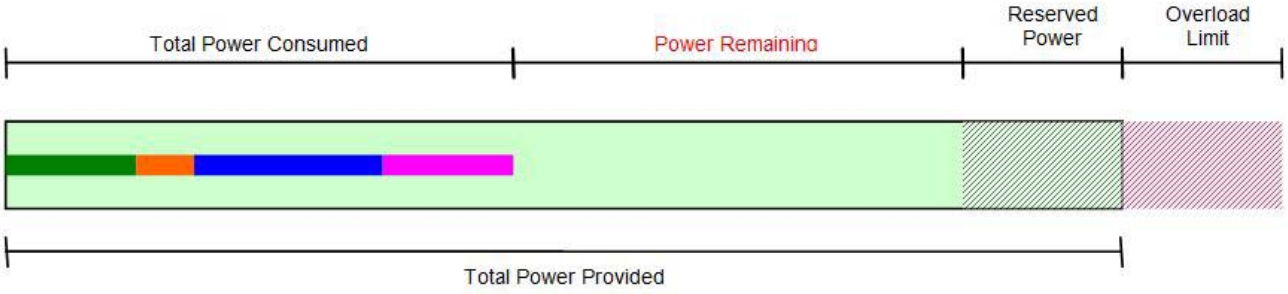


Figure 5. Consumption-Based Power Management

If the granting policy is grant based, then the power remaining for new grants is the total ungranted power. The power remaining is the total power provided minus the total power granted.

The problem with this approach is that much of the provided power is unused because PDs often do not consume all of their granted power.

If the granting policy is consumption-based, then the power remaining for new grants is the total unconsumed power. The power remaining is the total power provided minus the total power consumed (excluding the reserved power). This approach uses more of the provided power, but there is a possibility that the system may consume more power than the power provided (system overload).

To avoid system overloads caused by momentary surges in power consumption, the host can specify that a certain amount of power be held in reserve. The Si3482 does not use the reserved power when granting new requests.

Most power supplies can tolerate a limited amount of overload for a short duration. The host specifies the overload limit of the power supplies to the Si3482. If a system overload is less than the overload limit, the Si3482 turns off ports, one at a time in priority order, until the system is no longer overloaded. If a system overload is greater than the overload limit (severe overload), the Si3482 immediately turns off all low-priority ports. If the system is still overloaded, the Si3482 turns off additional ports, one at a time in priority order, until the system is no longer overloaded. A severe overload is usually caused by removing a power supply.

2.1. Host Interface

The Si3482 has a UART interface and an SPI interface for communicating with the host MCU, but only one interface is used at a time. The PSLCT (protocol select) pin selects which interface is used.

2.1.1. UART Interface

If the PSLCT pin is tied high, then the Si3482 uses the UART interface to communicate with the host MCU. The Si3482 uses the TX and RX pins to send and receive serial data. The BAUD0, BAUD1, and BAUD2 pins select the baud rate for the UART interface.

Table 5. Baud Rates

BAUD2	BAUD1	BAUD0	Baud Rate (bps)
H	L	L	19200
H	L	H	38400
H	H	L	57600
H	H	H	115200

The UART interface uses eight data bits, no parity, and one stop bit.

2.1.2. SPI Interface

If the PSLCT pin is tied low, then the Si3482 uses the SPI interface to communicate with the host MCU. The Si3482 is an SPI slave device. Therefore, it receives data on the MOSI pin and sends data on the MISO pin. The host MCU drives the NSS and SCK pins.

The SPI interface uses an active-high clock (CKPOL = 0). The clock line is low in the idle state, and the leading edge of the clock goes from low to high. The SPI interface samples the data on the leading edge of the clock (CKPHA = 0). The SPI interface transfers the most-significant bit first, and the maximum bit rate is 1 Mbps

2.2. Hardware Only Mode

The host interface (SPI or UART) and the UART baud rate are pin-configured. The Si3482 reads the pin configuration at power up, and it cannot be changed after power up. The hardware designer only needs to decide which interface to use and, if UART is selected, which BAUD rate to use.

In general, the host interface must be electrically isolated from the host MCU using an appropriate electrical isolator for either SPI or UART signals as well as power supply status signals as needed.

The Si3482 backs up its configuration to internal flash memory. Once the Si3482 is configured, it is possible to disconnect the host interface and use the Si3482 without a host MCU.

To configure the Si3482 when a host MCU is not required or for initial system debug, the USB adapter and Power Manager GUI supplied with the Si3482 evaluation system, “SMARTPSE24-KIT”, can be adapted to set the device configuration as desired.

3. Serial Packet Protocol

The Si3482 contains the Power Manager component and the interface to the Power Manager is a collection of routines known as the Power Manager application programming interface (API).
The Power Manager API is described later in this manual. The host MCU contains a User Interface component, which calls the routines in the Power Manager API to get status information and configure and control the Power Manager.

The Serial packet protocol (SPP) is a remote procedure call (RPC) mechanism that allows a User Interface component to call routines in the Power Manager, even though the User Interface component and the Power Manager are on different MCUs. The Serial Packet Protocol is implemented by a Serial Packet Client in the host MCU and the Serial Packet Server in the Si3482.

A Serial Packet Client is a collection of stub routines with the same names and parameters as the routines in the Power Manager API. A User Interface component calls a stub routine, and the stub routine uses the host interface to send a packet to the Serial Packet Server. The Serial Packet Client receives a packet back and then returns to the User Interface component.

The Serial Packet Server receives a packet from a Serial Packet Client and then calls the specified routine in the Power Manager. When the Power Manager routine returns, the Serial Packet Server sends a packet back to the Serial Packet Client.

Silicon Labs provides a Serial Packet Client Software Development Kit (SDK) that implements SPP and is portable to any MCU. The customer only needs to write some small support routines, which map SDK routines to RTOS routines.

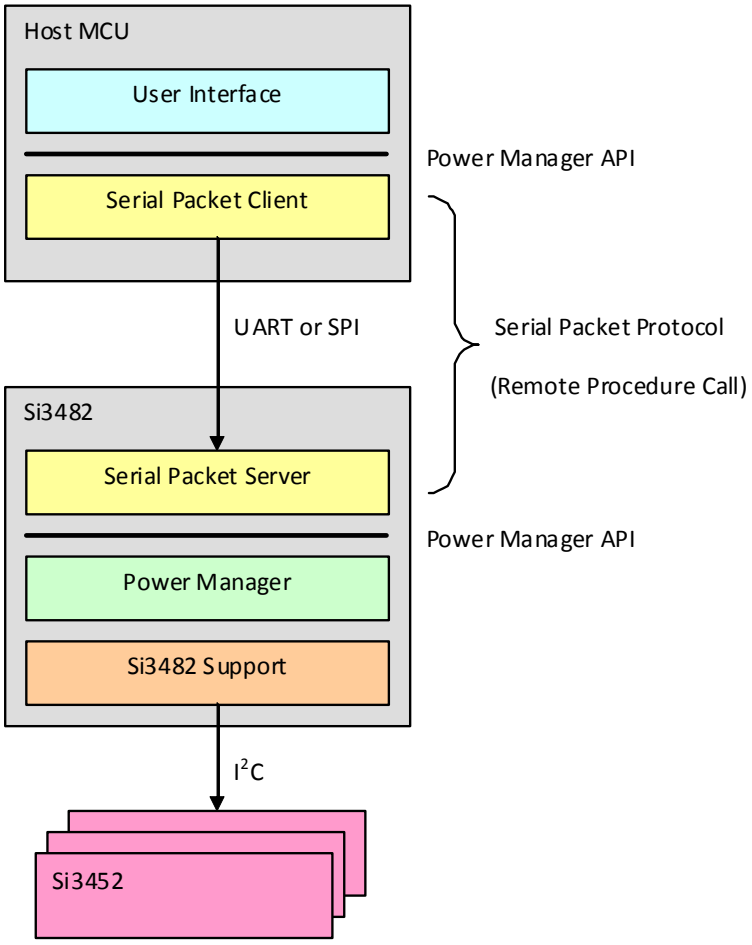


Figure 6. Serial Packet Protocol

3.1. Packet Format

A packet is a sequence of fields sent together as a unit. Figure 7 shows the SPP packet format.

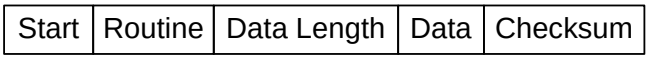


Figure 7. Packet Format

Each field is a single byte except for the Data field. The Data field may be from zero to 255 bytes.

3.1.1. Start Field

The Start field marks the beginning of a packet and always contains the Start-of-Packet (SOP) character (0xAC). If data is lost on the host interface, the Serial Packet Server and the Serial Packet Client use the Start field to resynchronize. A “receive packet” routine starts by receiving and discarding bytes until the SOP character is found.

3.1.2. Checksum Field

The Checksum field is used to verify that the packet was not corrupted during transmission. The sender of a packet calculates the checksum and writes it into the Checksum field. The receiver of a packet verifies that the checksum is correct. The Checksum field should contain the value such that all the bytes in the packet, except for the Start field, add up to zero.

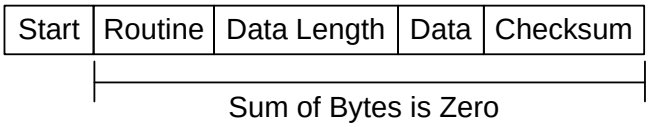


Figure 8. Packet Checksum

To calculate the checksum, the sender uses an 8-bit variable to sum up the bytes of the Routine field through the end of the Data field. The sender adds one to the one's complement of this sum and stores the result in the Checksum field.

$$Checksum = (\sim Sum) + 1$$

To verify the checksum, the receiver uses an 8-bit variable to sum up the bytes of the Routine field through the Checksum field. The sum should be zero.

3.1.3. Routine Field

The Routine field identifies a routine in the Power Manager API.

The client uses the Routine field to specify which routine to call. The client should verify that the Routine field in a received packet matches the Routine field in the sent packet.

Table 6. Routine Field

Routine	Symbol	Value
GetSystemStatus()	RTN_GETSYSTEMSTATUS	1
GetSystemInfo()	RTN_GETSYSTEMINFO	2
GetTotalPowerConsumed()	RTN_GETTOTALPOWERCONSUMED	3
GetTotalPowerGranted()	RTN_GETTOTALPOWERGRANTED	4
GetTotalPowerProvided()	RTN_GETTOTALPOWERPROVIDED	5
GetPortCount()	RTN_GETPORTCOUNT	6
GetPortStatus()	RTN_GETPORTSTATUS	7
GetPortInfo()	RTN_GETPORTINFO	8

Table 6. Routine Field (Continued)

Routine	Symbol	Value
GetPortPriorityStatus()	RTN_GETPORTPRIORITYSTATUS	9
GetPortPowerConsumed()	RTN_GETPORTPOWERCONSUMED	10
GetPortPowerGranted()	RTN_GETPORTPOWERGRANTED	11
GetPortPowerRequested()	RTN_GETPORTPOWERREQUESTED	12
GetPortPowerAvailable()	RTN_GETPORTPOWERAVAILABLE	13
ResetSystem()	RTN_RESETSYSTEM	14
RestoreFactoryDefaults()	RTN_RESTOREFACTORYDEFAULTS	15
SetPortControl()	RTN_SETPORTCONTROL	16
AdjustPortPower()	RTN_ADJUSTPORTPOWER	17
SetPowerProvided()	RTN_SETPOWERPROVIDED	18
GetPowerProvided()	RTN_GETPOWERPROVIDED	19
SetReservedPower()	RTN_SETRESERVEDPOWER	20
GetReservedPower()	RTN_GETRESERVEDPOWER	21
SetOverloadLimit()	RTN_SETOVERLOADLIMIT	22
GetOverloadLimit()	RTN_GETOVERLOADLIMIT	23
SetGrantingPolicy()	RTN_SETGRANTINGPOLICY	24
GetGrantingPolicy()	RTN_GETGRANTINGPOLICY	25
SetRetryPolicy()	RTN_SETRETRYPOLICY	26
GetRetryPolicy()	RTN_GETRETRYPOLICY	27
SetPowerLocation()	RTN_SETPOWERLOCATION	28
GetPowerLocation()	RTN_GETPOWERLOCATION	29
SetPortEnable()	RTN_SETPORTENABLE	30
GetPortEnable()	RTN_GETPORTENABLE	31
SetPortCapability()	RTN_SETPORTCAPABILITY	32
GetPortCapability()	RTN_GETPORTCAPABILITY	33
SetPortPriority()	RTN_SETPORTPRIORITY	34
GetPortPriority()	RTN_GETPORTPRIORITY	35
SetPortPowerLimit()	RTN_SETPORTPOWERLIMIT	36
GetPortPowerLimit()	RTN_GETPORTPOWERLIMIT	37
SetPowerSupplyStatus()	RTN_SETPOWERSUPPLYSTATUS	38
GetPowerSupplyStatus()	RTN_GETPOWERSUPPLYSTATUS	39
GetEvents()	RTN_GETEVENTS	40

3.1.4. Data Length Field

The DataLength field specifies the number of bytes in the Data field. The number of bytes may be from zero to 255.

3.1.5. Data Field

The Data field is used to pass data to and from the Si3482. The Data field may contain four different types of data:

- Parameters
- System Information
- Port Information
- Events

The Data field has a different format for each type of data. In almost all packets (sent and received), the Data field has the Parameters format. The only exceptions are the packets that are received back after calling the GetSystemInfo(), GetPortInfo(), and GetEvents() routines. The Data fields for these packets are in the System Information format, Port Information format, and Events format.

3.1.5.1. Parameters Format

The Parameters format of the Data field is used to pass parameters to Power Manager routines. In most cases, the Parameters format is also used to return data from the routines.

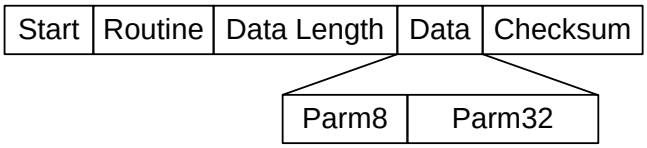


Figure 9. Parameters Format

The Parameters format has an 8-bit Parm8 field followed by a 32-bit Parm32 field (see Table 7). Depending on the routine being called, Parm8, Parm32, or both fields are used. Sometimes, neither field is used. However, both fields are always sent and received. The DataLength field contains five.

Table 7. Use of Parameters

Routine	Parameters in Sent Packet		Parameters in Received Packet	
	Parm8	**Parm32***	**Parm8**	**Parm32***
GetSystemStatus()			SystemStatus	
GetSystemInfo()			Uses System Information Format	
GetTotalPowerConsumed()				PowerConsumed
GetTotalPowerGranted()				PowerGranted
GetTotalPowerProvided()				PowerProvided
GetPortCount()			PortCount	
GetPortStatus()	Port		PortStatus	
GetPortInfo()	Port		Uses Port Information Format	
GetPortPriorityStatus()	Port		PortPriorityStatus	
GetPortPowerConsumed()	Port			PowerConsumed
GetPortPowerGranted()	Port			PowerGranted
GetPortPowerRequested()	Port			PowerRequested
***Note:** The Parm32 field is big endian; therefore, the most significant byte is first.				

Table 7. Use of Parameters (Continued)

Routine	Parameters in Sent Packet		Parameters in Received Packet	
	Parm8	**Parm32***	**Parm8**	**Parm32***
GetPortPowerAvailable()	Port			PowerAvailable
ResetSystem()			Result	
RestoreFactoryDefaults()				
SetPortControl()	Port	Control	Result	
AdjustPortPower()	Port	PortPower	Result	
SetPowerProvided()	PowerSupply	PowerProvided	Result	
GetPowerProvided()	PowerSupply			PowerProvided
SetReservedPower()	ReservedPower		Result	
GetReservedPower()			ReservedPower	
SetOverloadLimit()	OverloadLimit		Result	
GetOverloadLimit()			OverloadLimit	
SetGrantingPolicy()	GrantingPolicy		Result	
GetGrantingPolicy()			GrantingPolicy	
SetRetryPolicy()	RetryPolicy		Result	
GetRetryPolicy()			RetryPolicy	
SetPowerLocation()	PowerLocation		Result	
GetPowerLocation()			PowerLocation	
SetPortEnable()	Port	Enable	Result	
GetPortEnable()	Port		Enable	
SetPortCapability()	Port	Capability	Result	
GetPortCapability()	Port		Capability	
SetPortPriority()	Port	Priority	Result	
GetPortPriority()	Port		Priority	
SetPortPowerLimit()	Port	PowerLimit	Result	
GetPortPowerLimit()	Port			PowerLimit
SetPowerSupplyStatus()	PowerSupply	Status	Result	
GetPowerSupplyStatus()	PowerSupply		Status	
GetEvents()			Uses Events Format	

***Note:** The Parm32 field is big endian; therefore, the most significant byte is first.

3.1.5.2. System Information Format

The System Information format of the Data field is used to return system information to the client. System information is returned after calling the GetSystemInfo() routine.

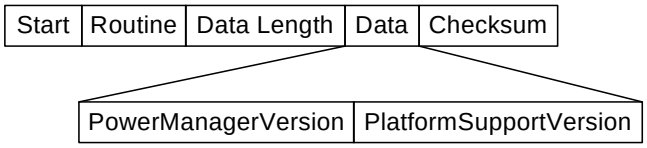


Figure 10. System Information Format

The System Information format has a PowerManagerVersion field followed by a PlatformSupportVersion field. Both of these fields are eight bytes long and contain a version string that is a zero-terminated ASCII string. A version string may be from one to seven characters long. The Routine field contains RTN_GETSYSTEMINFO, and the DataLength field contains 16.

3.1.5.3. Port Information Format

The Port Information format of the Data field is used to return port information to the client. Port information is returned after calling the GetPortInfo() routine.

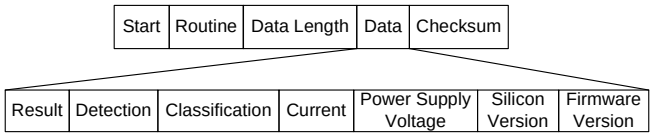


Figure 11. Port Information Format

In C, the Port Information format is:

```
typedef struct
{
     INT8   Result;
     UINT8  Detection;
     UINT8  Classification;
     UINT16 Current;
     UINT16 PowerSupplyVoltage;
     char   SiliconVersion[2];
     char   FirmwareVersion[8];
} DATA_PORTINFO;
```

The Port Information format is a sequence of fields as shown above. For more information, read the description of the GetPortInfo() routine in the Power Manager API Section. The Routine field contains RTN_GETPORTINFO, and the DataLength field contains 17.

The Result field contains the return code from the GetPortInfo() routine, and, if Result is not SUCCESS (0), the remaining fields should be ignored.

The Current and PowerSupplyVoltage fields are big endian. Therefore, the most significant byte comes first.

3.1.5.4. Events Format

The Events format of the Data field is used to return events to the client. Events are returned after calling the GetEvents() routine.

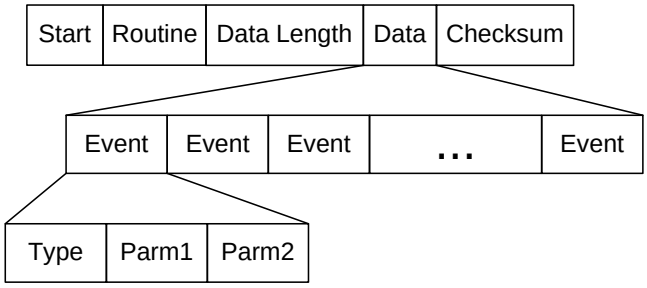


Figure 12. Events Format

In the Si3482, the Serial Packet Server internally receives events from the Power Manager and stores them in a circular event queue. If the event queue becomes full, newer events overwrite older events.

If a client wishes to receive events, it should periodically get the events from the Serial Packet Server. The client gets the events by sending a packet with the Routine field set to RTN_GETEVENTS. The Serial Packet Server returns all the events from the event queue in a single packet with the Data field in the Events format.

The Data field does not have a fixed length. The length of the Data field depends on the number of events that are returned. An event is three bytes long; so, the number of events in the Data field is DataLength divided by three. If there are no events to return, then DataLength is zero, and the Data field is empty. The maximum number of events that can be returned is 85.

3.2. SPP Error Handling

There are many reasons why a client may not receive back a packet. Perhaps the Si3482 is not running or perhaps the serial data was corrupted or lost during transmission (in either direction). In any case, it is not prudent for a Serial Packet Client to call a serial receive routine that blocks forever until data is received. If the serial receive routine does not have a timeout option, the client should not call the receive routine unless it knows that received data is available. If a client does not receive a packet within one second of sending a packet, then the client should assume that there has been a communications error. The client should resend the original packet or simply give up (but do not wait forever to receive a packet).

When the Serial Packet Server receives a packet, it validates the packet. If the checksum is bad or the Routine field is invalid, the Serial Packet Server ignores the packet and does not send back a packet in response. After one second, the client should realize that a packet has not been received and should resend the original packet.

The Si3482 checks the configuration every 30 seconds to see if it has changed. If the configuration has changed, the Si3482 backs up the configuration to internal flash memory. While the Si3482 is writing to flash memory, it cannot send or receive packets on the host interface. If a host MCU sends a packet to the Si3482 while it is backing up the configuration, the packet is lost. If a host MCU does not receive a packet back within one second, the host MCU should resend the original packet.

4. Power Manager API

User Interface components call the routines in the Power Manager API to get status information and configure and control the Power Manager. The Power Manager API has routines for:

- Management
- System Status
- Port Status
- System Control
- Port Control
- System Configuration
- Port Configuration
- Power Supply Status
- Events

4.1. Management

The Management routines allow a User Interface component to:

- Initialize the Serial Packet Client
- Close the Serial Packet Client

4.1.1. InitPowerManager

Initialize the Serial Packet Client.

Prototype: `INT8 InitPowerManager ( EVENT_HANDLER EventHandler )`

Parameters: `EventHandler` A pointer to an event handler routine.

Return Value: Zero (for success) or an error code.

This is the first call that a User Interface component makes to the Serial Packet Client. The Serial Packet Client does everything that is required to bring itself to an operational state and is ready to execute all other Power Manager API routines.

If a User Interface component would like to receive events, it should pass the address of an event handler to this routine. If a User Interface component does not want to receive events, it should pass a null address to this routine.

An event handler is located in a User Interface component and has the following prototype:

Prototype: `void EventHandler ( EVENT *Event )`

Parameters: `Event` A pointer to an EVENT structure, where event information is provided.

Return Value: None

An `EVENT` structure has the following layout:

```
typedef struct
{
      UINT8  Type;
      INT8   Parm1;
      UINT8  Parm2;
} EVENT;
```

Table 8 lists the event type values.

Table 8. Event Types

Event Type	Value	Symbol	Parm1	Parm2
System	1	SYSTEM_EVENT	System Status	(not used)
Port	2	PORT_EVENT	Port Status	Port Number
Power Supply	4	POWER_SUPPLY_EVENT	Power Supply Status	Power Supply Number
Error	8	ERROR_EVENT	Error Code	(error specific)
Information	16	INFO_EVENT	Information Code	(not used)

See the GetSystemStatus(), GetPortStatus(), and GetPowerSupplyStatus() routines for a listing of the status values. See the end of this section for a listing of error codes and information codes.

The Serial Packet Client calls the event handler whenever there is a change in system status, port status, or power supply status. The event handler is also called when an error occurs, such as port overload. If the first parameter (Parm1) is negative, this means there is an error.

4.1.2. ClosePowerManager

Shut down the Serial Packet Client.

Prototype: `void ClosePowerManager ( void )`

Parameters: None

Return Value: None

This function shuts down the Serial Packet Client and frees all allocated resources.

4.2. System Status

The System Status routines allow a User Interface component to get the following information:

- System Status
- System Info
- Total Power Consumed
- Total Power Granted
- Total Power Provided

4.2.1. GetSystemStatus

Get the status of the system.

Prototype: `INT8 GetSystemStatus( void )`

Parameters: None

Return Value: System status value.

Table 9 lists the system status values. The system status is the overall status of the system. A negative system status value is an error that is not specific to a particular port.

Table 9. System Status

Status	Value	Symbol
OK	0	STATUS_SYSTEM_OK
Initialization Failed	–1	STATUS_SYSTEM_INIT_FAIL
Under Voltage	–2	STATUS_SYSTEM_UNDER_VOLT
Over Temperature	–3	STATUS_SYSTEM_OVER_TEMP
Communications Lost	–4	STATUS_SYSTEM_COMM_LOST

4.2.2. GetSystemInfo

Get information about the system.

Prototype: `void GetSystemInfo ( SYSINFO *SystemInfo )`

Parameters: `SystemInfo` A pointer to a SYSINFO structure, where the system information is returned.

Return Value: None

A `SYSINFO` structure has the following layout:

```
typedef struct
{
      char  *PowerManagerVersion;
      char  *PlatformSupportVersion;
} SYSINFO;
```

The SYSINFO structure contains the version of the Power Manager and the version of the Platform Support component as zero-terminated strings.

4.2.3. GetTotalPowerConsumed

Get the power consumed by all PDs.

Prototype: `INT32 GetTotalPowerConsumed ( void )`

Parameters: None

Return Value: Total power consumed in milliwatts.

4.2.4. GetTotalPowerGranted

Get the power granted to all PDs.

Prototype: `INT32 GetTotalPowerGranted ( void )`

Parameters: None

Return Value: Total power granted in milliwatts.

4.2.5. GetTotalPowerProvided

Get the power provided by all power supplies.

Prototype: `INT32 GetTotalPowerProvided ( void )`

Parameters: None

Return Value: Total power provided in milliwatts.

4.3. Port Status

The Port Status routines allow a User Interface component to get the following:

- Port Count
- Port Status
- Port Info
- Port Priority Status
- Port Power Consumed
- Port Power Granted
- Port Power Requested
- Port Power Available

4.3.1. GetPortCount

Get the number of ports in the system.

Prototype: `UINT8 GetPortCount ( void )`

Parameters: None

Return Value: Number of ports in the system.

When the Power Manager starts up, it discovers the number of ports in the system by searching for port controllers.

4.3.2. GetPortStatus

Get the status of a port.

Prototype: `INT8 GetPortStatus ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Port status value or an error code.

Table 10 lists the Port status values.

Table 10. Port Status Values

Status	Value	Symbol	Description
Disabled	0	STATUS_PORT_DISABLED	The port is off because it is not allowed to turn on.
Powered On	1	STATUS_PORT_POWERED_ON	A PD is connected and receiving power.
Powered Off	2	STATUS_PORT_POWERED_OFF	The port is off because a PD is not connected.
Denied	3	STATUS_PORT_DENIED	The port is off because there is not enough power remaining to grant the power request.
Blocked	4	STATUS_PORT_BLOCKED	The port is off because of a port overload.
Forced On	5	STATUS_PORT_FORCED_ON	The user forced the port on.
Forced Off	6	STATUS_PORT_FORCED_OFF	The user forced the port off.

If a port is blocked, then the PD consumed more power than it was granted (port overload), and the retry policy is “retry after reconnect”. To remove the block, the user must physically disconnect the PD from the port. Another way to remove the block is to disable and then reenable the port.

4.3.3. GetPortInfo

Get low-level port information.

Prototype: `INT8 GetPortInfo ( UINT8 Port, PORTINFO *PortInfo )`

Parameters: `Port` The port number (1–48)

`PortInfo` A pointer to a PORTINFO structure, where the port information is returned.

Return Value: Zero (for success) or an error code

A `PORTINFO` structure has the following layout:

```
typedef struct
{
      UINT8   Detection;
      UINT8   Classification;
      UINT16  Current;              /* in mA */
      UINT16  PowerSupplyVoltage;   /* in mV */
      char   *SiliconVersion;
      char   *FirmwareVersion;
} PORTINFO;
```

Table 11 lists Detection values.

Table 11. Detection Values

Detection	Value	Symbol
Unknown	0	DETECT_UNKNOWN
Short	1	DETECT_SHORT
Low	3	DETECT_LOW
Good	4	DETECT_GOOD
High	5	DETECT_HIGH
Open	6	DETECT_OPEN

Table 12 lists Classification values.

Table 12. Classification Values

Classification	Value	Symbol
Unknown	0	CLASS_UNKNOWN
Class 1	1	CLASS_1
Class 2	2	CLASS_2
Class 3	3	CLASS_3
Class 4	4	CLASS_4
Fingers Not Equal	5	CLASS_UNEQ_FINGERS
Class 0	6	CLASS_0
Overload	7	CLASS_OVERLOAD

4.3.4. GetPortPriorityStatus

Get the priority status of a port.

Prototype: `INT8 GetPortPriorityStatus ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Port priority status value or an error code.

Table 13 lists the Port Priority Status values.

Table 13. Port Priority Status Values

Port Priority Status	Value	Symbol
Low	0	PRIORITY_LOW
High	1	PRIORITY_HIGH
Forced	2	PRIORITY_FORCED
Critical	3	PRIORITY_CRITICAL

The priority status of a port is the currently-active priority and may be different than the configured priority of the port. If a port is forced on or off and the configured priority is low or high, the priority status is elevated to the forced priority. If a forced port is returned to automatic control, the Power Manager returns the priority status to the configured priority.

4.3.5. GetPortPowerConsumed

Get the power that a PD is currently using.

Prototype: `INT32 GetPortPowerConsumed ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Port power consumed in milliwatts or an error code.

4.3.6. GetPortPowerGranted

Get the power that is allocated to a PD.

Prototype: `INT32 GetPortPowerGranted ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Port power granted in milliwatts or an error code.

4.3.7. GetPortPowerRequested

Get the power that a PD says it needs.

Prototype: `INT32 GetPortPowerRequested ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Port power requested in milliwatts or an error code.

4.3.8. GetPortPowerAvailable

Get the maximum request that would be successfully granted.

Prototype: `INT32 GetPortPowerAvailable ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Port power available in milliwatts or an error code.

An LLDP agent calls this routine to ask the question, “What is the maximum power that you would give me if I asked for it?”

If a port has a power limit, then the power limit is returned. If a port does not have a power limit and the port can supply high power, then maximum power (40 W) is returned; otherwise, low power (15.4 W) is returned.

4.4. System Control

The System Control routines allow a User Interface component to

- Reset the System
- Restore Factory Defaults

4.4.1. ResetSystem

Reset the system.

Prototype: `INT8 ResetSystem ( void )`

Parameters: None

Return Value: Zero (for success) or an error code.

4.4.2. RestoreFactoryDefaults

Restore the configuration to factory default values.

Prototype: `void RestoreFactoryDefaults ( void )`

Parameters: None

Return Value: None

The Power Manager also resets the system after setting the configuration to default values.

4.5. Port Control

The Port Control routines allow a User Interface component to

- Set Port Control
- Adjust Port Power

4.5.1. SetPortControl

Set how the port is turned on and off.

Prototype: `INT8 SetPortControl ( UINT8 Port, UINT8 Control )`

Parameters:

`Port`	The port number (1–48)
`Control`	How the port is turned on and off.

Return Value: Zero (for success) or an error code.

Table 14 lists Port Control values.

Table 14. Port Control Values

Control	Value	Symbol
Automatic	0	`PORT_CTRL_AUTOMATIC`
Force On	1	`PORT_CTRL_FORCE_ON`
Force Off	2	`PORT_CTRL_FORCE_OFF`

If the port control is automatic, the Power Manager automatically turns the port on and off when a PD is connected and disconnected from the port.

If the port control is forced on, the port's priority is boosted to the forced priority level. This usually results in the port turning on. However, a forced port cannot cause a critical priority port to turn off in order to turn on the forced port. If a forced port is granted power, the Power Manager turns on a forced port even if no PD is detected.

If the port control is forced off, the port is unconditionally turned off and held off. A forced-off port is considered to be temporarily off, while a disabled port is considered to be permanently off.

4.5.2. AdjustPortPower

Adjust the power granted to a PD.

Prototype: `INT8 AdjustPortPower ( UINT8 Port, INT32 Power )`

Parameters:

`Port`	The port number (1–48)
`Power`	Requested port power in milliwatts.

Return Value: Zero (for success) or an error code.

An LLDP agent calls this routine to reallocate the power granted to a PD. The agent can request more power than is currently granted or it can request less power than is currently granted. This routine allows an LLDP agent to dynamically change the amount of power granted to a PD during the course of a connection. A port must be on before its power can be adjusted.

4.6. System Configuration

The System Configuration routines allow a User Interface component to set and get

- Power Provided
- Reserved Power
- Overload Limit
- Granting Policy
- Retry Policy
- Power Location

4.6.1. SetPowerProvided

Set the amount of power that is output from a power supply.

Prototype: `INT8 SetPowerProvided ( UINT8 PowerSupply, INT32 Power )`

Parameters:

`PowerSupply`	The power supply number (1–3)
`Power`	Power provided by the power supply in milliwatts.

Return Value: Zero (for success) or an error code.

4.6.2. GetPowerProvided

Get the amount of power that is output from a power supply.

Prototype: `INT32 GetPowerProvided ( UINT8 PowerSupply )`

Parameters:

`PowerSupply`	The power supply number (1–3)

Return Value: Power provided by the power supply in milliwatts or an error code.

4.6.3. SetReservedPower

Set the percentage of power that is reserved from granting.

Prototype: `INT8 SetReservedPower ( INT8 Reserved )`

Parameters:

`Reserved`	Reserved power as a percentage of the total power provided.

Return Value: Zero (for success) or an error code.

If the granting policy is consumption-based, the Power Manager holds this amount of power in reserve. The Power Manager does not use the reserved power to grant new requests. This creates a power buffer that reduces the likelihood of system overloads caused by momentary surges in consumption.

4.6.4. GetReservedPower

Get the percentage of power that is reserved from granting.

Prototype: `INT8 GetReservedPower ( void )`

Parameters: None

Return Value: Reserved power as a percentage of the total power provided or an error code.

4.6.5. SetOverloadLimit

Set the maximum system overload that the power supplies can tolerate.

Prototype: `INT8 SetOverloadLimit ( INT8 Limit )`

Parameters: `Limit` Overload limit as a percentage of the total power provided.

Return Value: Zero (for success) or an error code.

The overload limit is the maximum system overload that the power supplies can tolerate. It is expressed as a percentage of the total power provided. If a system overload is less than the overload limit, the ports are turned off one at a time. If a system overload is greater than the overload limit (severe overload), all of the low-priority ports are immediately turned off.

4.6.6. GetOverloadLimit

Get the maximum system overload that the power supplies can tolerate.

Prototype: `INT8 GetOverloadLimit ( void )`

Parameters: None

Return Value: Overload limit as a percentage of the total power provided or an error code.

4.6.7. SetGrantingPolicy

Set the granting policy.

Prototype: `INT8 SetGrantingPolicy ( INT8 GrantingPolicy )`

Parameters: `GrantingPolicy` How requests for power are granted.

Return Value: Zero (for success) or an error code.

Table 15 lists the Granting Policy values.

Table 15. Granting Policy Values

Granting Policy	Value	Symbol
Grant-based	0	`GRANT_POLICY_GRANT_BASED`
Consumption-based	1	`GRANT_POLICY_CONSUMPTION_BASED`

The granting policy is used by the Power Manager when deciding if a request for power should be granted. If the granting policy is grant based, the remaining power is considered to be the total ungranted power. If the granting policy is consumption-based, the remaining power is considered to be the total unconsumed power (excluding the reserved power). If the remaining power is greater than or equal to the requested power, then the Power Manager grants the request.

Grant based: `PowerRemaining = TotalPowerProvided – TotalPowerGranted`

Consumption based: `PowerRemaining = TotalPowerProvided – TotalPowerConsumed – ReservedPower`

4.6.8. GetGrantingPolicy

Get the granting policy.

Prototype: `INT8 GetGrantingPolicy ( void )`

Parameters: None

Return Value: The granting policy value.

4.6.9. SetRetryPolicy

Set the retry policy.

Prototype: `INT8 SetRetryPolicy ( INT8 RetryPolicy )`

Parameters: `RetryPolicy` When to retry after a port overload.

Return Value: Zero (for success) or an error code.

Table 16 lists the Retry policy values.

Table 16. Retry Policy Values

Retry Policy	Value	Symbol
Immediate	0	RETRY_IMMEDIATELY
Reconnect	1	RETRY_AFTER_RECONNECT
Reenable	2	RETRY_AFTER_REENABLE

The retry policy specifies when the Power Manager tries again to power a port that is turned off because of a port overload. A port overload is when the power consumed by a PD is greater than the power granted to that PD. If the retry policy is “immediate”, the Power Manager tries to turn the port back on immediately.

If the retry policy is “reconnect”, the Power Manager waits until the PD is disconnected and then reconnected before it tries again to power the port. The Power Manager must detect an open circuit on the port before retrying.

If the retry policy is “reenable”, the Power Manager disables the port when a port overload occurs. The user must reenable the port before the Power Manager tries to power the port again.

4.6.10. GetRetryPolicy

Get the retry policy.

Prototype: `INT8 GetRetryPolicy ( void )`

Parameters: None

Return Value: The retry policy value.

4.6.11. SetPowerLocation

Set the location of the power source.

Prototype: `INT8 SetPowerLocation ( INT8 Location )`

Parameters: `Location` Where the power source is located.

Return Value: Zero (for success) or an error code.

Table 17 lists the Location values.

Table 17. Power Location Values

Location	Value	Symbol
Endpoint	0	LOCATION_ENDPOINT
Midspan	1	LOCATION_MIDSPAN

If the power source is within an Ethernet switch, the location is an “endpoint”. If the power source is inserted between an Ethernet switch and a PD, the location is a “midspan”. The Power Manager uses different back-off timings for different locations. The Power Manager assumes that an endpoint device uses the Alternative A pinout and that a midspan device uses the Alternative B pinout.

4.6.12. GetPowerLocation

Get the location of the power source.

Prototype: `INT8 GetPowerLocation ( void )`

Parameters: None

Return Value: The location value.

4.7. Port Configuration

The Port Configuration routines allow a User Interface component to set and get:

- Port Enable
- Port Capability
- Port Priority
- Port Power Limit

4.7.1. SetPortEnable

Set if a port is allowed to turn on.

Prototype: `INT8 SetPortEnable ( UINT8 Port, INT8 Enable )`

Parameters:

`Port`	The port number (1–48)
`Enable`	Specifies if the port is allowed to power on.

Return Value: Zero (for success) or an error code

If `Enable` is zero, the port is disabled and is not allowed to power on. If `Enable` is one, the port is enabled and is allowed to power on.

4.7.2. GetPortEnable

Get if a port is allowed to turn on.

Prototype: `INT8 GetPortEnable ( UINT8 Port )`

Parameters:

`Port`	The port number (1–48)

Return Value: Zero for disabled, one for enabled, or an error code

4.7.3. SetPortCapability

Set if a port can supply high power.

Prototype: `INT8 SetPortCapability ( UINT8 Port, INT8 Capability )`

Parameters:

`Port`	The port number (1–48)
`Capability`	Specifies if the port can supply high power.

Return Value: Zero (for success) or an error code

Table 18 lists the Port Capability values.

Table 18. Port Capability Values

Capability	Value	Symbol
Low Power	0	`CAPABILITY_LOW_POWER`
High Power	1	`CAPABILITY_HIGH_POWER`

If the port hardware is designed to supply high power (PoE+), set `Capability` to one. Otherwise, set `Capability` to zero.

A port's capability cannot be changed while the port is on. This is a limitation of the Si3452 port controller.

4.7.4. GetPortCapability

Get if a port can supply high power.

Prototype: `INT8 GetPortCapability ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Capability value or an error code

4.7.5. SetPortPriority

Set the priority of a port.

Prototype: `INT8 SetPortPriority ( UINT8 Port, INT8 Priority )`

Parameters: `Port` The port number (1–48)
`Priority` How important it is for the port to be powered.

Return Value: Zero (for success) or an error code

Table 19 lists the Port priority values.

Table 19. Port Priority Values

Port Priority	Value	Symbol
Low	0	`PRIORITY_LOW`
High	1	`PRIORITY_HIGH`
Critical	3	`PRIORITY_CRITICAL`

The priority of a port indicates how important it is that the port receives power. If there is not enough power provided for all ports that want power, then the low priority ports are the first ports to be denied. Critical priority ports are the last ports to be denied.

If a port is forced on, then the port's priority is elevated to the forced priority level. Forced priority is between high priority and critical priority and cannot be directly set by the user. When a port is forced on, it may cause a high priority port to be turned off, but it can never cause a critical priority port to be turned off.

If a severe overload occurs, all of the low priority ports are immediately powered off.

4.7.6. GetPortPriority

Get the priority of a port.

Prototype: `INT8 GetPortPriority ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Priority value or an error code

4.7.7. SetPortPowerLimit

Set the power limit of a port.

Prototype: `INT8 SetPortPowerLimit ( UINT8 Port, INT32 Limit )`

Parameters: `Port` The port number (1–48)
`Limit` Maximum power that may be granted in milliwatts.

Return Value: Zero (for success) or an error code.

A power limit restricts the amount of power that may be granted to a port. If a port's power limit is zero, the Power Manager grants the power requested without restriction. If a port's power limit is greater than zero, the Power Manager grants the lesser of the power limit or the power requested. If a power request is greater than the power limit, the Power Manager grants less power than requested.

4.7.8. GetPortPowerLimit

Get the power limit of a port.

Prototype: `INT32 GetPortPowerLimit ( UINT8 Port )`

Parameters: `Port` The port number (1–48)

Return Value: Power limit in milliwatts or an error code

4.8. Power Supply Status

The Power Supply Status routines allow a User Interface component to get:

- Power Supply Status

4.8.1. SetPowerSupplyStatus

Set the status of a power supply.

Prototype: `INT8 SetPowerSupplyStatus (UINT8 PowerSupply, INT8 Status)`

Parameters: `PowerSupply` The power supply number (1–3)
`Status` Power supply status

Return Value: Zero (for success) or an error code.

In platforms that do not have signals to indicate the presence of power supplies, this routine allows a User Interface component to simulate the insertion and removal of power supplies. However, the Si3482 does have power supply signals (PS1, PS2, and PS3). If a User Interface component calls this routine, the Power Manager does nothing and returns SUCCESS.

4.8.2. GetPowerSupplyStatus

Get the status of a power supply.

Prototype: `INT8 GetPowerSupplyStatus ( UINT8 PowerSupply )`

Parameters: `PowerSupply` The power supply number (1–3)

Return Value: The status of the power supply.

Table 20 lists the Power supply status values.

Table 20. Power Supply Status

Power Supply Status	Value	Symbol
Removed	0	STATUS_POWER_SUPPLY_REMOVED
Inserted	1	STATUS_POWER_SUPPLY_INSERTED

A User Interface component calls this function to determine whether a power supply is present in a bay. If the voltage on the specified power supply pin (PS1, PS2, or PS3) is high, this routine returns STATUS_POWER_SUPPLY_INSERTED; otherwise, this routine returns STATUS_POWER_SUPPLY_REMOVED.

4.9. Events

An event is an unsolicited notification from the Power Manager that a status has changed. The Power Manager calls the event handler in the Serial Packet Server whenever there is a change in system status, port status, or power supply status. The Serial Packet Server stores the events in an event queue.

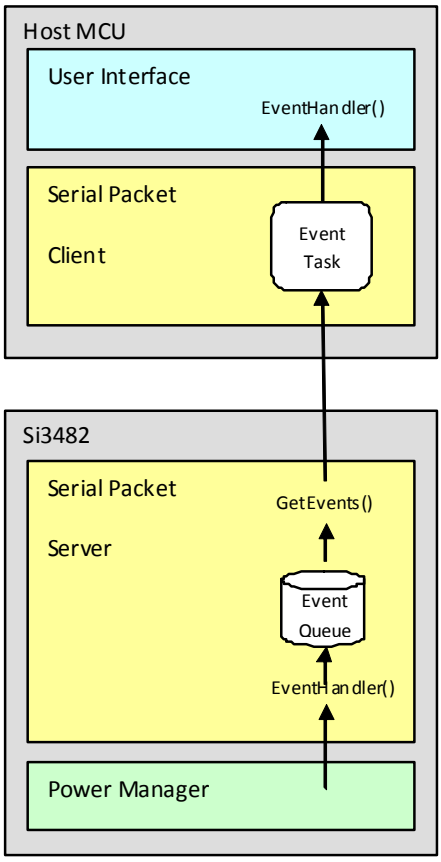


Figure 13. Events Architecture

If a User Interface component wishes to receive events, it provides the address of an event handler when it calls InitPowerManager(). The Serial Packet Client spawns an event task. The event task remotely calls the GetEvents() routine in the Serial Packet Server every 200 ms to retrieve the events from the event queue. For each event that is retrieved, the Serial Packet Client calls the event handler in the User Interface component.

4.9.1. GetEvents

Get the events from the event queue.

Prototype: `void GetEvents ( EVENT *Event )`

Parameters: `Event` Pointer to returned events.

Return Value: None.

This routine exists in the Serial Packet Server and is called by a Serial Packet Client to get events from the event queue. This routine sends back a packet with the Data field in the Events format. This routine may return from zero to 85 events.

4.10. Return Codes

The routines of the Power Manager API return codes to indicate the success or failure of an operation. These codes are also used in Parm1 of error events and information events.

4.10.1. Success Code

A zero code indicates success.

Table 21. Success Code

Success	Value	Symbol
Success	0	SUCCESS

4.10.2. Error Codes

A negative code indicates an error.

Table 22. Error Codes

Error	Value	Symbol
Port number is invalid	–1	ERROR_PORT_INVALID
Power supply number is invalid	–2	ERROR_PWR_SUPLY_INVALID
Parameter is invalid	–3	ERROR_PARAMETER_INVALID
Cannot create resource	–4	ERROR_RESOURCE_CREATE
Resource is invalid	–5	ERROR_RESOURCE_INVALID
Cannot configure resource	–6	ERROR_RESOURCE_CONFIG
Cannot read from resource	–7	ERROR_RESOURCE_READ
Cannot write to resource	–8	ERROR_RESOURCE_WRITE
Cannot find the resource	–9	ERROR_RESOURCE_NOT_FND
Cannot load the configuration	–10	ERROR_CONFIG_LOAD
Cannot save the configuration	–11	ERROR_CONFIG_SAVE
Configuration data is invalid	–12	ERROR_CONFIG_INVALID
Configuration data is corrupt	–13	ERROR_CONFIG_CORRUPT
System overload	–14	ERROR_SYSTEM_OVERLOAD
Port overload	–15	ERROR_PORT_OVERLOAD
Startup overload	–16	ERROR_STARTUP_OVERLOAD

4.10.3. Information Codes

A positive code indicates useful information.

Table 23. Information Codes

Information	Value	Symbol
Restored to factory defaults	1	INFO_DEFAULTS_RESTORED
System reset	2	INFO_SYSTEM_RESET
Configuration saved	3	INFO_CONFIG_SAVED

5. Pin Descriptions

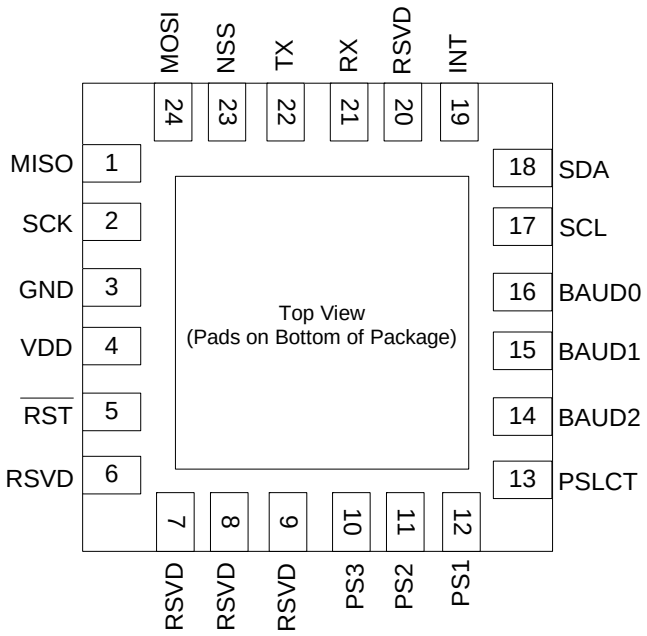


Table 24. Si3482 Pin Descriptions

Pin #	Name	Type	Description
1	MISO	Output	SPI output.
2	SCK	Input	SPI clock.
3	GND	Power	Ground.
4	VDD	Power	VDD.
5	RST	Input	Reset (a low will reset the Si3480).
6	RSVD	Input	Reserved—tie low.
7	RSVD	Reserved	Do not connect.
8	RSVD	Reserved	Do not connect.
9	RSVD	Reserved	Do not connect.
10	PS3	Input	Logic high indicates the power supply is available.
11	PS2	Input	Logic high indicates the power supply is available.
12	PS1	Input	Logic high indicates the power supply is available.
13	PSLCT	Input	Tie high or low to select between SPI and UART interface.
14	BAUD2	Input	Tie high or low to select UART baud rate.
15	BAUD1	Input	Tie high or low to select UART baud rate.

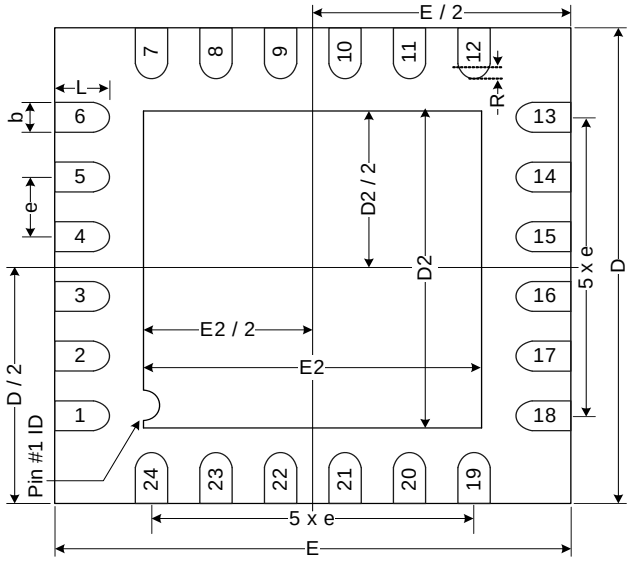
Table 24. Si3482 Pin Descriptions (Continued)

Pin #	Name	Type	Description
16	BAUD0	Input	Tie high or low to select UART baud rate.
17	SCL	Open Collector	Connect to Si3452 SCL and pull up resistor.
18	SDA	Open Collector	Connect to Si3452 SDA and pull up resistor.
19	INT	Input	Connect to Si3452 INT and pull up resistor.
20	RSVD	Reserved	Do not connect.
21	RX	Input	UART receive.
22	TX	Output	UART transmit.
23	NSS	Input	SPI select.
24	MOSI	Input	SPI input.

6. Package Outline: 24-Pin QFN

The Si3482 is packaged in an industry-standard, RoHS-compliant 6 x 6 mm^2, 24-pin QFN package.

Bottom View



Side View

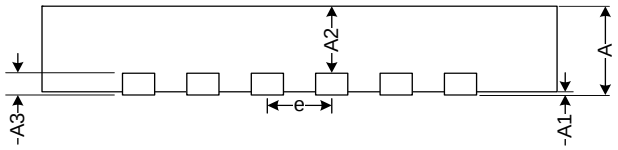


Figure 14. 24-Pin QFN Mechanical Diagram

Table 6.1. QFN-24 Package Dimensions

	MM		
	Min	**Typ**	**Max**
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A2	—	0.50	—
A3	—	0.25	—
b	0.18	0.25	0.30
D	—	4.00	—
D2	2.50	2.60	2.70
E	—	4.00	—
E2	2.50	2.60	2.70
e	—	0.50	—
L	0.35	0.40	0.45
N	—	24	—
ND	—	6	—
NE	—	6	—
R	0.09	—	—

7. PCB Land Pattern

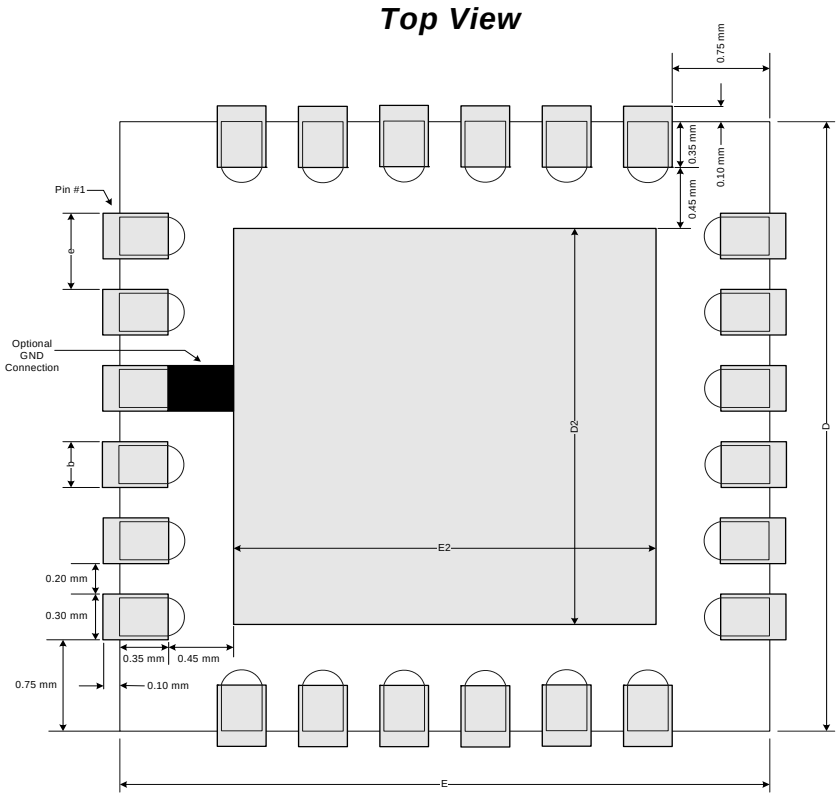


Figure 15. Typical QFN-24 PCB Land Pattern

8. Solder/Paste Recommendation

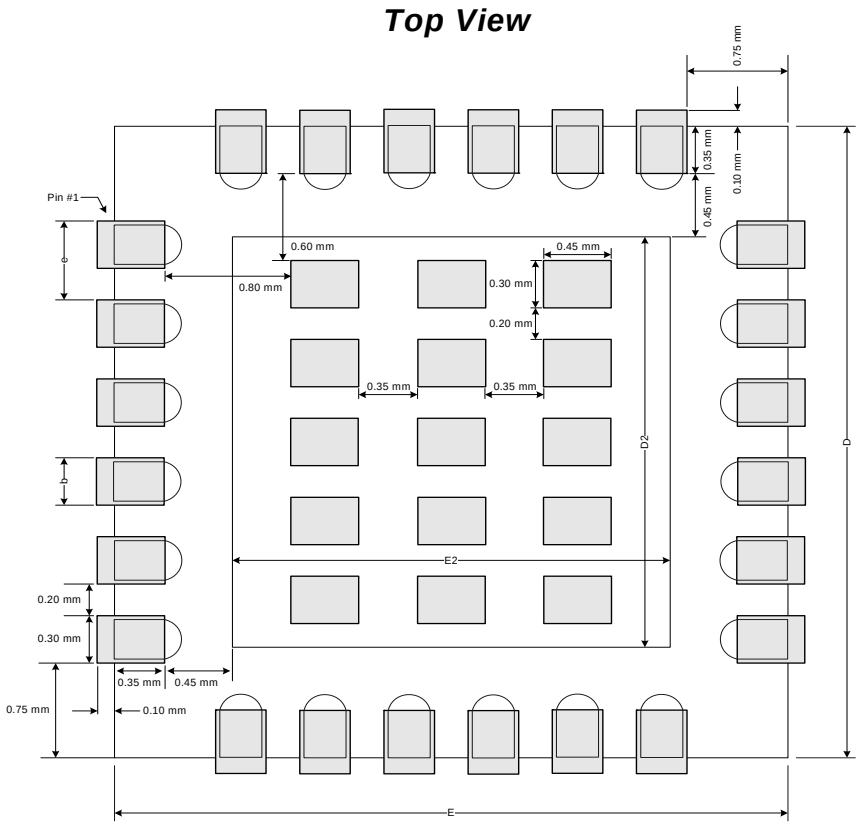


Figure 16. QFN-24 Solder Paste Recommendation

9. Top Marking Diagram

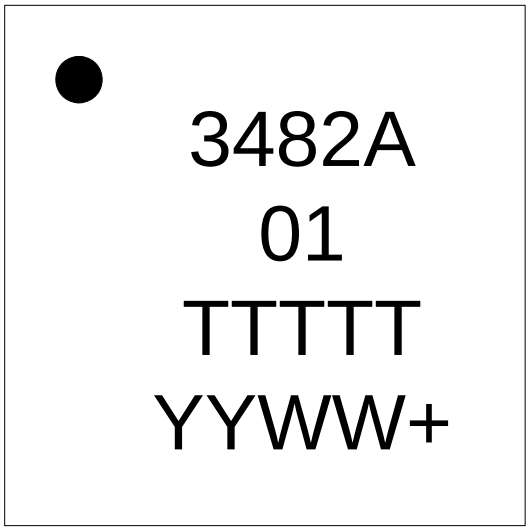


Figure 17. Top Marking Diagram

Table 25. Top Marking Explanation

Line 1 Marking:	Pin 1 Identifier	Circle, 0.25 mm diameter
	Product ID	3482A
Line 2 Marking:	Firmware revision	01 = Firmware revision 01
Line 3 Marking:	TTTTT = Trace Code	Manufacturing code characters from the Markings section of the Assembly Purchase Order form
Line 4 Marking:	YYWW+ = Date Code	YY = Last two digits of current year WW = Current Work Week
	Lead Free Designator	+

10. Ordering Guide

Table 26. Si3482 Ordering Guide

Ordering Part Number	Description	Package Information
Si3482-A01-GM	Power management controller	24-pin 4x4 mm QFN RoHS compliant
SMARTPSE24-KIT	An evaluation kit with the Si3482, six Si3452 controllers, and the Si3500 for generating the 3.3 V supply from the PoE supply.	Evaluation Board

Notes:
1. Add “R” to the part number to denote tape and reel option (Si3482-A01-GMR).
2. The ordering part number is not the same as the device mark. See "6. Package Outline: 24-Pin QFN" on page 33 for device marking information

CONTACT INFORMATION

Silicon Laboratories Inc.

400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Please visit the Silicon Labs Technical Support web page:
https://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.