CIRQUE®

Interfacing to the Pinnacle ASIC Using SPI and I2C

Application Note

Document Version 1.6

AUGUST 2017
GT-AN-090620

This document provides the information necessary to interface to a Cirque Pinnacle ASIC. Important features needed to READ, WRITE, and process data are included. Sample firmware is provided to show how to properly communicate with the Pinnacle ASIC using SPI or I2C protocol. This document applies to Cirque's Pinnacle 2.2, and Pinnacle AG 1.4

Document Version History

Date	Current Version	Description
15 JULY 2011	1.0	Documentation creation.
30 JULY 2011	1.2	Modified Firmware ID, added SPI timing note
OCTOBER 2012	1.3	Corrected documentation errors.
MAY 2014	1.4	Updated formatting and edited documentation for accuracy.
JUNE 2015	1.5	Updated document for accuracy
AUGUST 2017	1.6	Added extended registers information and updated sample code.

Notice

Copyright, Trademark, and Patent Information

Table of Contents

1. Introduction

This document describes how to communicate with a Cirque Pinnacle ASIC using either the SPI or I2C protocol. Pinnacle is also capable of communication using PS/2 signals (commonly used for computer peripherals); however, this document focuses solely on design for embedded devices using the built-in support for SPI and I2C communication. A wide variety of microcontrollers can be used with SPI and I2C to interact with Cirque's Pinnacle ASIC.

There are two types of Pinnacle ASICs (Pinnacle version 2.2 and Pinnacle AG version 1.4), both types are described in this manual. Most information is the same for both versions. When differences exist, they are identified in the tables and descriptions.

1.0.0.1 Additional Documentation

Other documents that provide information about Cirque's Pinnacle ASIC:

- AS-150408: Pinnacle ASIC Electrical and Mechanical Specification
- GT-AN-090624: Managing Sensor Compensation Application note
- GT-AN-090625: Using a Stylus with Cirque’s Pinnacle ASIC Application note

2. Pinnacle Overview

The Pinnacle ASIC monitors a capacitive touch sensor. When a finger (or a stylus) is placed on the surface of the sensor Pinnacle will determine the location of the object (user's finger or stylus). The position data is reported to the host as X and Y coordinates. The strength of the signal is reported as the Z coordinate.

3. Pinnacle Operation

Pinnacle has several features and options to achieve varied sensor functionality. This section describes how to establish proper communication with Pinnacle, and how to select the preferred features and options.

The Pinnacle ASIC uses a simple Register Access Protocol (RAP) method to READ and WRITE to the Pinnacle's registers. The primary functions that aid communication with Pinnacle are described in this section.

3.1 Register Access Protocol (RAP)

Pinnacle's registers are read and written to, using a Register Access Protocol (RAP). RAP has only two functions, READ and WRITE.

Note: *See the sections on SPI or I2C for examples of how to perform the READ and WRITE operations.*

Registers are accessed by sending a byte in the format shown in Table 1. The Standard Registers have five-bit addresses that range from 0x00 to 0x1F; see Table 2 on page 6 for the register map.

Follow this process to avoid accidentally replacing existing settings in a register, when performing a WRITE sequence:

1. READ the register.
2. Modify the value using logical operators to set or clear only the intended bits.
3. WRITE the modified value to the register.

Table 1. Register Access Protocol

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
READ (0xAX)	1	0	1	Address 4	Address 3	Address 2	Address 1	Address 0
WRITE (0x8X)	1	0	0	Address 4	Address 3	Address 2	Address 1	Address 0

3.1.1 RAP_WRITE and RAP_READ Sample Code

Table 2. Standard Register Set

Address	Pinnacle 2.2	Pinnacle AG	Description
0x00	Firmware ID	Firmware ID	Firmware ASIC ID.
0x01	Firmware Version	Firmware Version	Firmware revision number.
0x02	Status1	Status1	Contains status flags about the state of Pinnacle.
0x03	SysConfig1	SysConfig1	Contains system operation and configuration bits.
0x04	FeedConfig1	FeedConfig1	Contains feed operation and configuration bits.
0x05	FeedConfig2	FeedConfig2	Contains feed operation and configuration bits.
0x06	RESERVED	RESERVED	RESERVED
0x07	CalConfig1	CalConfig1	Contains calibration configuration bits.
0x08	PS/2 Aux Control	PS/2 Aux Control	Contains Data register for PS/2 Aux Control.
0x09	Sample Rate	Sample Rate	Number of samples generated per second.
0x0A	ZIdle	ZIdle	Number of Z=0 packets sent when Z goes from >0 to 0.
0x0B	Z Scaler	Z Scaler	Contains the pen Z_On threshold.
0x0C	Sleep Interval	Sleep Interval	
0x0D	Sleep Timer	Sleep Timer	
0x0E	RESERVED	RESERVED	RESERVED
0x0F	RESERVED	RESERVED	RESERVED
0x10	RESERVED	PacketByte_0	trackpad Data (Pinnacle AG)
0x11	RESERVED	PacketByte_1	trackpad Data (Pinnacle AG)
0x12	PacketByte_0	PacketByte_2	trackpad Data
0x13	PacketByte_1	PacketByte_3	trackpad Data
0x14	PacketByte_2	PacketByte_4	trackpad Data
0x15	PacketByte_3	RESERVED	trackpad Data (Pinnacle)
0x16	PacketByte_4	RESERVED	trackpad Data (Pinnacle)
0x17	PacketByte_5	RESERVED	trackpad Data (Pinnacle)
0x18	RESERVED	RESERVED	RESERVED
0x19	RESERVED	RESERVED	RESERVED
0x1A	RESERVED	RESERVED	RESERVED
0x1B	RESERVED	RESERVED	RESERVED
0x1C	RESERVED	RESERVED	RESERVED
0x1D	RESERVED	RESERVED	RESERVED
0x1E	RESERVED	RESERVED	RESERVED
0x1F	RESERVED	RESERVED	RESERVED

3.2 Status and Data Ready Signals

3.2.1 Hardware Data Ready

To interrupt or alert the host when data is ready, Pinnacle uses a Hardware Data Ready (HW_DR) signal (active HIGH) that is triggered by either the Command Complete (SW_CC) flag or the Software Data Ready (SW_DR) flag in the Status1 register address 0x02. When either software flag is set, the HW_DR (pin 36 PA2) is also asserted. The SW_DR and SW_CC flags must be manually cleared by the host. The HW_DR signal remains asserted while either SW_DR or SW_CC is asserted (See Table 3 below).

Table 3. Data Ready and Command Complete

Signal	Description
Hardware Data Ready (HW_DR)	Asserted (HIGH) with either SW_DR or SW_CC flag. Cleared when both SW_DR and SW_CC flags are cleared.
Software Data Ready (SW_DR)	Asserted with new data. Remains asserted until cleared by host.
Command Complete (SW_CC)	Asserted after calibration, POR. Remains asserted until cleared by host.

3.2.2 Command Complete

The Software Command Complete (SW_CC) flag (Bit [3] of Register 0x02, Status1) is asserted after successful completion of either power on reset (POR) or calibration. This flag triggers the HW_DR signal, which then alerts the Host. The host must clear the SW_CC flag, by writing 0x00 to the Status1 register (0x02), in order to clear HW_DR (see Table 4).

To clear Command Complete and Software Data Ready flags simultaneously, see the code below in *ClearFlags Example Code*. The RAP Write byte is explained in *Writing to a Register Using SPI* *on page 23*.

3.2.2.1 ClearFlags Example Code

```
void ClearFlags(void) {
    RAP_WriteByte(0x02, 0x00); // Write 0x00 to Status1 register (0x02)
}
```

3.2.3 Software Data Ready

When a touch is detected, Pinnacle loads X and Y position data into the position registers and asserts the SW_DR flag (Bit [2] of Register 0x02, Status 1), which also triggers the HW_DR signal (see Table 4 below). While the finger/stylus is present, the position registers are updated every 10 ms and SW_DR and HW_DR are asserted. The host must clear SW_DR, by writing a value of 0x00, to the Status1 register (0x02), which will clear HW_DR and ensure no data is missed.

Note: *SW_CC and SW_CC flags may be cleared by the same write operation.*

Table 4. Status1 - Register 0x02

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description					Command Complete (SW_CC)	Software Data Ready (SW_DR)		
READ/WRITE					R/W	R/W		
Values					0 = Clear	0 = Clear		
Default					0	0		

3.3 Data Output

To receive position data from Pinnacle, the data feed must be enabled by asserting the Feed Enable flag (Bit [0] Register 0x04, FeedConfig1) (see Table 5). Once enabled, Pinnacle's finger tracking, sampling, and reporting begins.

Pinnacle can report position as relative data or absolute data. Relative data is identical to a mouse or pointer where each position is reported as relative to the last position. Absolute data is a grid with X positions and Y positions (Pinnacle range: X= 0 - 2047, Y= 0 - 1535, Pinnacle AG range: X= 0 - 1919, Y= 0 - 1407).

The absolute data mode allows the designer a wider range of freedom to process the data and design the preferred touch functions. In absolute mode, Y and X data can be inverted to allow different orientations of the trackpad. The data output mode (relative or absolute) is specified using Bit[1] in Register 0x04, FeedConfig1 (see Table 5). Advanced features for relative mode can be set using flags in Register 0x05, FeedConfig2 (see Table 6 on page 9).

Table 5. FeedConfig1 (Data Output Flags) - Register 0x04

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Y data Invert[1]	X data Invert[1]		Y Disable[2]	X Disable[3]	Filter Disable[4]	Data mode	Feed Enable
READ/ WRITE	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Values	1=Y max to 0 0=0 to Y max	1=X max to 0 0=0 to Y max		1=no Y data 0=Y data	1=no X data 0=X data	1=no filter 0=filter	1=absolute 0=relative	1=feed 0=no feed
Default	0	0		0	0	0	0	0

1. Y and X data count invert is only available when in absolute mode.
   (Pinnacle range: X= 0 - 2047, Y= 0 - 1535, Pinnacle AG range: X= 0 - 1919, Y= 0 - 1407)
2. Disabling the Y-axis will not allow regular tracking and is not recommended for typical applications.

3. Disabling the X-axis will not allow regular tracking and is not recommended for typical applications.

4. The Filter disable bit controls whether the filtering algorithm is applied to generated data. By default the hardware filters are enabled. Cirque does not recommend disabling hardware filtering.

Table 6. FeedConfig2 (Feature flags for Relative Mode Only) - Register 0x05

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Swap X & Y			GlideExtend® Disable [1]	Scroll Disable	Secondary Tap Disable [2]	All Taps Disable [3]	Intellimouse Enable [4]
READ/ WRITE	R/W			R/W	R/W	R/W	R/W	R/W
Values	1=90° rotation 0=0° rotation			1= disable 0 = enable	1= disable 0 = enable	1= disable 0 = enable	1= disable 0 = enable	1= enable 0= disable
Default	0			0	0	0	0	0

1. GlideExtend is Cirque's patented motion extender technology that allows the user to continue the drag function when an edge is reached, by lifting and repositioning the finger.

2. Secondary Taps allows a tap in the upper right corner (standard orientation) to simulate an activation of the secondary button.

3. Disabling all taps disables secondary taps, even if secondary tap is explicitly enabled.

4.Intellimouse enabled will change Pinnacle's relative data packet to four bytes rather than three. The fourth byte (PacketByte_3) will report scroll data (referred to as wheel count).

3.3.1 Relative Data Mode Data Packet

Note: *The data packets are reported in different registers for Pinnacle and Pinnacle AG. See Table 2, Standard Register Set on page 6 for the appropriate register address.*

Pinnacle reports a change (or delta) in current position from the previous position in relative mode. The position deltas are stored in PacketByte_1 (X delta) and PacketByte_2 (Y delta). The X and Y data sign (either negative (-) or Positive (+)) indicates the direction of change and is reported in PacketByte_0.

3.3.1.1 Button Data

Relative Data Mode supports three button inputs. Button data can be generated by the actual button or by taps (if enabled) that simulate a button press. Button data is also reported in PacketByte_0. If Intellimouse is enabled (Bit[0] of Register 0x05, FeedConfig2), scroll data (wheel count) is reported in an additional byte, PacketByte_3 (See Table 7).

Table 7. Relative Position Registers Byte Format

Address	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PacketByte_0	X & Y Sign, Button (or tap) data	0	0	Y sign 1=(-) 0=(+)	X sign 1=(-) 0=(+)	1	BTN Auxiliary 1=pressed 0=released	BTN Secondary (or top right corner tap) 1=pressed 0=released	BTN Primary (or tap) 1=pressed 0=released
PacketByte_1	X Delta	X7	X6	X5	X4	X3	X2	X1	X0
PacketByte_2	Y Delta	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
PacketByte_3	Wheel Count	W7	W6	W5	W4	W3	W2	W1	W0

Example: To go from a positive delta to a negative would look like this:

```
3
2
1
0
255(sign bit is set, making -1)
254(sign bit is set, making -2)
252(sign bit is set, making -3)
```

3.3.2 Absolute Data Packet

Note: *The data packets are reported in different registers for Pinnacle and Pinnacle AG. See Table 2 on page 6 and Table 8 on page 11 for the appropriate register addresses.*

Absolute mode is entered by setting Bit[1] in FeedConfig1 (Register 0x04) (see Table 5 on page 8). The Host reads whichever registers meet the application needs. For example, if button status is not needed, the host can read the last four packet bytes.

Table 8. Absolute Position Registers Byte Format

Pinnacle Address	Pinnacle AG Address	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x12	0x10	Button/Switch Status			SW5	SW4	SW3	SW2	SW1	SW0
0x14	0x11	X-Position Low Byte	X7	X6	X5	X4	X3	X2	X1	X0
0x15	0x12	Y-Position Low Byte	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0x16	0x13	X & Y Position High Bits	Y11	Y10	Y9	Y8	X11	X10	X9	X8
0x17	0x14	Z-Level			Z5	Z4	Z3	Z2	Z1	Z0

3.3.2.1 Example Code

Note: *This code is for Pinnacle 2.2.*

```
uint16_t calculateXPosition(uint8_t * registerVals) {
    return registerVals[0x14] | ((registerVals[0x16] & 0x0F) << 8);
}

uint16_t calculateYPosition(uint8_t * registerVals) {
    return registerVals[0x15] | ((registerVals[0x16] & 0xF0) << 4);
}
```

3.4 Power Modes

Pinnacle has four power modes - Active (touch detected), Idle (no touch), Low Power/ Sleep (lower power after ~ 5 seconds of inactivity) and Shutdown/Standby (no data reported) (See Figure 1 below).

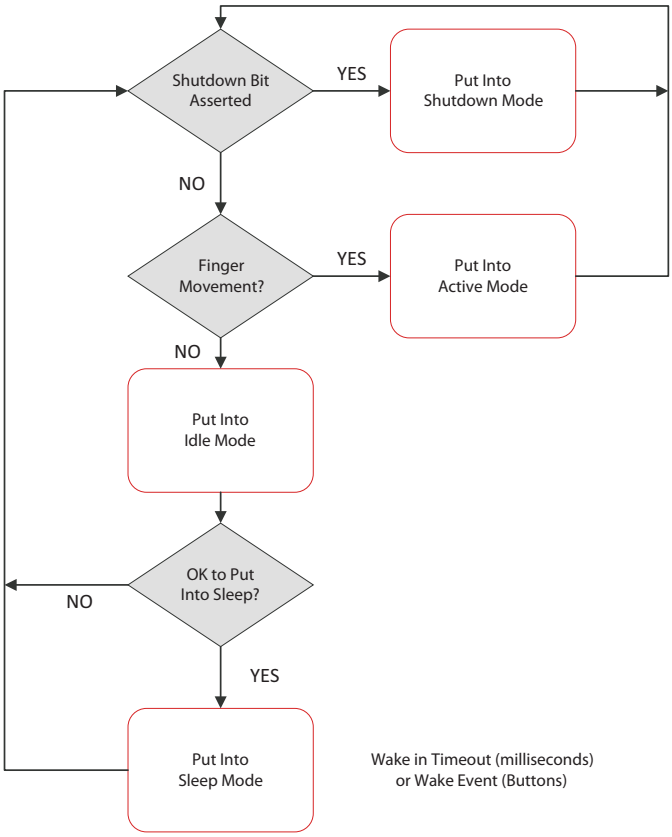

Figure 1. Pinnacle Power Mode State Machine

3.4.1 Active and Idle Mode

By default, Pinnacle toggles between Active and Idle mode. Pinnacle is in Active mode when a touch is detected (that is, a finger or stylus is present and is moving or tapping on the trackpad). The measurement system is active and data packets are being created and then sent to the host. Active mode begins as soon as a touch is detected. Idle mode is entered when the finger has been removed and there are no data packets to be sent. While in Idle mode, Pinnacle wakes every 10 milliseconds to check for a touch.

3.4.2 Low Power Sleep Mode

Enabling sleep mode will cause Pinnacle to go into a low power mode (around 50 µA) within 5 seconds of no touch detection. While in sleep mode, Pinnacle will wake within 300 ms to report any detection of a finger/stylus. To enable sleep mode, assert the Sleep Enable flag, Bit [2] of Register 0x03, SysConfig1 (see Table 9 on page 13). The bit remains asserted and Pinnacle uses low power sleep mode until the Host clears the Sleep Enable flag.

3.4.3 Shutdown/Standby Mode

Shutdown/Standby mode is a very low power mode and Pinnacle does not track touch in this mode. Shutdown/Standby is activated by the host when the Shutdown flag is asserted (Bit [1] Register 0x03, SysConfig1) and deactivated when the bit is cleared.

Table 9. SysConfig1 (Low Power Mode) - Register 0x03

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description						Sleep Enable [1]	Shutdown [1]	Reset
READ/WRITE						R/W	R/W	R
Values						1=low power mode 0=normal mode	1=Shutdown 0=Active	1 = Reset
Default						0	0	0

1.Request additional instructions from Cirque for exiting Sleep and Shutdown modes if your application requires SPI at 1 MHZ or greater.

3.5 Touch Detection (Z Information)

3.5.1 Z Level

The Z level is a measure of how much the capacitive field has changed. When no finger is near, the Z level will be at or near zero (0). The Z level will start increasing as a finger approaches. The Z level continues to increase as more of the surface area of the finger touches the surface of the sensor. Position data is generated for any Z level greater than zero (0). Overlay types (glass vs. plastic) on the sensor and application requirements are factors that the developer needs to consider when setting Z level thresholds for a touch and a release. With absolute data mode enabled, Z-level values are reported in Bits [0:5] of the Z-level Packet Byte (see Table 8 on page 11).

3.5.2 Z-Idle

During Z-idle (no touch detected) and when in absolute data mode, Pinnacle will continue to send empty packets (both X and Y data set to 0x00) every 10 ms. The number of empty packets to be sent is stored in Register 0x0A, Z-idle (see Table 10 below). The default value is 0x1E (30 decimal). This value can be changed by writing to Register 0x0A. When set to zero (0), this register prevents any empty packets from being sent, and the position registers will contain the last sensed location until a new finger presence is detected.

The Z-Idle count can be a helpful design tool. For example, tap-frequency can be determined by counting the number of Z-idle packets reported between a finger lifting off and touching back down (cutting short the stream of Z-idle packets).

Table 10. Z-Idle Count - Register (0x0A)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Z-Idle Count							
READ/WRITE	R/W							
Values	0x00 through 0xFF (0 to 255 decimal)							
Default	0X1E							

3.6 Compensation

To enhance sensor performance, Pinnacle performs a calculation to compensate for the current operating environment. Pinnacle automatically executes compensation/calibration when powered on and when triggered by specific events.

4. Example Start-up Sequence

The following summarizes the typical sequence to use Pinnacle:

1. Power on Reset (POR); Pinnacle is in the default condition.
2. Host clears SW_CC (writes value 0x00 to Register 0x02, Status1), which clears HW_DR.
3. Host configures intended bits of registers 0x03 and 0x05.
4. Host enables the preferred output mode and enables the feed (As explained in *Touch Detection (Z Information)* *on page 14*.).
5. Host continuously monitors HW_DR pin for Pinnacle data ready interrupt.
   a. When received, host reads the appropriate packet data (Registers 0x10-0x17, depending on ASIC and data mode).
   b. Host clears SW_DR (writes value 0x00 to Register 0x02, Status1), which clears HW_DR.
   c. Host uses the data received.

4.0.0.1 Example Code

```
void StartupSequence(void)
{
    ClearFlags();
    RAP_WriteByte(0x03, 0x00); // Configures SysConfig1(normal mode, active)
    RAP_WriteByte(0x05, 0x1F); // Configures FeedConfig2(disable relative mode features)
    RAP_WriteByte(0x04, 0x03); // Configures FeedConfig1(absolute mode, enable feed)
}
```

5. Serial Peripheral Interface (SPI)

This section describes the basic operation, timing of SPI, as well as how it is used to communicate with Pinnacle.

SPI communication is a four-wire bus that includes a slave select line for communicating with multiple devices on the bus. Pinnacle is designed as a slave device on the SPI bus with a select line (SS), two data lines (MOSI and MISO), and a clock line (SCK), see Table 11 below.

Table 11. Pinnacle SPI Signals

Signal	Description
SPI	
SS	SPI Select (SS=0 for slave to be active and send data to master)
SCK	SPI Clock (CPOL=0=idle) (CPHA=1, latches on falling edge)
MOSI	SPI Master Out Slave In
MISO	SPI Master In Slave Out
Pinnacle	
HW_DR	Indicates Pinnacle has Data Ready (Active High) or a command is complete.
GPIO	General Purpose Input/Output
GND	Ground
VDD	Power Supply

SPI signal operation:

- A slave device will not communicate on the bus unless its slave select line (SS) is pulled low.
- Data sent by the master is placed on the Master Out, Slave In (MOSI) line.
- Data returned by Pinnacle is placed on the Master In, Slave Out (MISO) line.
- Both sets of data are latched on the falling edge of SCK.
- Data is presented Most Significant Byte first (MSB).
- Pinnacle supports data rates up to 13 MHz.

A basic example of interconnections between a single master and a single slave is to have the MOSI pins connected together and the MISO pins connected together. In this way, the data is transferred serially between master and slave (most significant bit first). The master always starts the data exchange. When the master device transmits data to a slave device on the MOSI pin, the slave sends data to the master on the MISO pin at the same time. This is full-duplex communication. All the data is synchronized using the SCK pin with a clock signal from the master device. Slave Select (SS) must be LOW in order for Pinnacle to be active and send data to the Master.

5.1 SPI Timing

Timing for the SPI data lines is shown below (see Figure 2, Figure 3 below, and Table 12 on page 18). The data exchange is started when the master pulls the Slave Select (SS) line low. Due to the full duplex nature of the SPI bus, Pinnacle always returns data at the same time that it is receiving data. The returned byte may be data from the previous command or it may be a filler byte. For both Master and Slave, data is changed on the rising edge of the clock and latched on the falling edge.

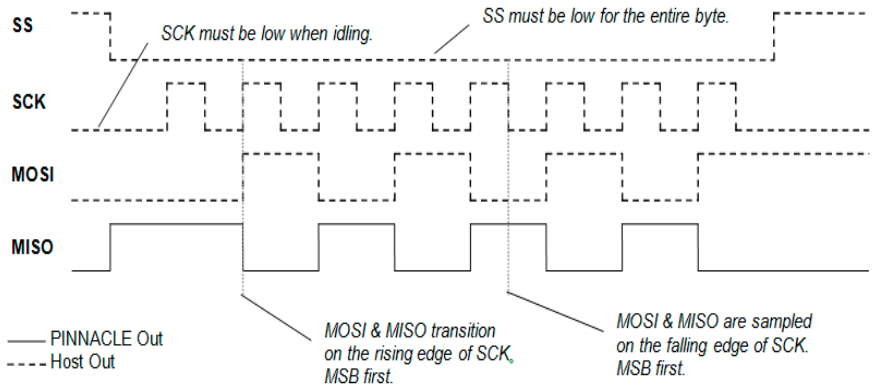

Figure 2. SPI Signals

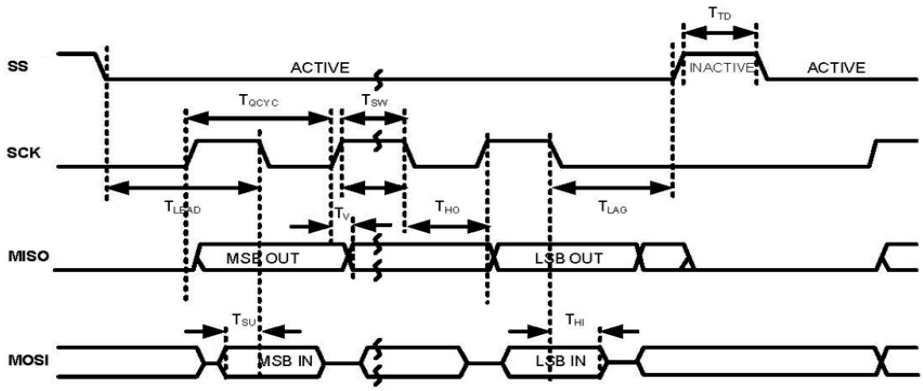

Figure 3. SPI Timing

Table 12. SPI Timing Explanation

Ref	Parameter	Symbol	Minimum	Maximum	Unit
1	Period	T_{QCYC}	76	N/A	ns
2	Clock (SCK) High or Low Time	T_{SW}	$T_{QCYC}/2$	N/A	
3	Chip Select Lag Time (last clock edge to slave select de-asserted)	T_{LAG}	30	N/A	
4	Inter-Message Transfer Delay (required by slave)	T_{TD}	50	N/A	
5	Chip Select Lead Time (slave select asserted to first SCK falling)	T_{LEAD}	T_{QCYC}	N/A	
6	Master Data Setup Time (data valid to SCK falling)	T_{SU}	$T_{QCYC}/4$	N/A	
7	Master Data Hold time (SCK falling to data invalid)	T_{HI}	$T_{QCYC}/4$	N/A	
8	Slave Data Valid Time (SCK rising to data valid)	T_{V}	N/A	30	
9	Slave Data Hold Time (SCK falling to data invalid)	T_{HO}	$T_{QCYC}/2$	N/A	

6. Register Access Using SPI

Communicating with Pinnacle requires RAP, which has only READ and WRITE commands. The byte format is repeated in Table 13.

Table 13. Register Access Protocol

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
READ (0xAX)	1	0	1	Address 4	Address 3	Address 2	Address 1	Address 0
WRITE (0x8X)	1	0	0	Address 4	Address 3	Address 2	Address 1	Address 0

6.1 Reading a Register Using SPI

SPI requires two exchanges when reading a register.

1. The host starts by sending the READ command byte. The simultaneous response byte may be data from the previous command or it may be a filler byte.
2. Three additional bytes (“filler bytes”) with a value of 0xFB, must be sent after the READ command byte to allow Pinnacle to respond. The response will be sent during the transmission of the final byte (see Table 14 below).

6.1.1 SPI Read Example

Read register 0x02 (Status1)

1. Use the OR operand to combine the READ command (0xAX) with the register address to be read. For this example, use register 0x02.

   ```
   0xA0|0x02 = 0xA2
   ```

2. Send 0xA2 followed by three-filler bytes (0xFB). The response to the third byte will be the contents of register 0x02.

Table 14. Example SPI READ Sequence

Byte	Command (MOSI)	Response (MISO)
1	0xA2	Response to a previous command or a filler byte.
2	0xFB	0xFB
3	0xFB	0xFB
4	0xFB	Content of register 0x02

6.1.1.1 SPI Read Example Code

```
void RAP_ReadByte(uint8_t address, uint8_t * data) {
    Assert_CS();
    SPI_Transfer(address | 0xA0); // Send register address OR'd with read-mask (0xA0)
    SPI_Transfer(0xFB); // Filler-byte
    SPI_Transfer(0xFB); // Filler-byte
    *data = SPI_Transfer(0xFB); // Contents are received on 3rd filler-byte
    DeAssert_CS();
}
```

6.2 Pipelining Multiple Reads

Multiple registers can be read by sending READ commands back-to-back. For greater throughput, back-to-back READ commands can overlap the last filler byte with the next READ command. Alternatively, an auto-increment command (0xFC) can be used in place of the filler byte (0xFB). Examples are provided for each method below.

6.2.1 Reading SPI Back-to-Back

Read register 0x02 (Status1) and 0x17 (Z Level) (see Table 15).

1. Use OR operand to combine the READ command (0xAX) with the register addresses to be read. For this example, use registers 0x02 and 0x17.

```
0xA0    | 0x02 =0xA2
0xA0    | 0x17 =0xB7
```

2. Send 0xA2 followed by three-filler bytes (0xFB). The response to the third byte will be the contents of register 0x02.

3. Send 0xB7 followed by three-filler bytes (0xFB). The response to the third byte will be the contents of register 0x17.

Table 15. Example of Back-to-Back SPI READ Commands

Byte	Command (MOSI)	Response (MISO)
1	0xA2	Response to a previous command or a filler byte.
2	0xFB	0xFB
3	0xFB	0xFB
4	0xFB	Content of register 0x02
5	0xB7	0xFB
6	0xFB	0xFB
7	0xFB	0xFB
8	0xFB	Contents of register 0x17

6.2.2 SPI Auto-Increment READ for Sequential Addresses Example

Read registers 0x14 through 0x16 (see Table 16).

1. Use OR operand to combine the READ command (0xAX) with the register addresses to be read.

```
0xA0    | 0x14 =0xB4
0xA0    | 0x15 =0xB5
0xA0    | 0x16 =0xB6
```

2. Send 0xB4 followed by four Auto-Increment bytes (0xFC) and end with a filler byte (0xFB) to read three sequential registers. The response to the third 0xFC will be the contents of register 0x14, the next response will be the data in register 0x15, then register 0x16 (see Table 16).

Table 16. Example of SPI Auto-Incremented READ Command Sequence

Byte	Command (MOSI)	Response (MISO)
1	0xB4	Response to a previous command or a filler byte.
2	0xFC	0xFB
3	0xFC	0xFB
4	0xFC	Contents of register 0x14
5	0xFC	Contents of register 0x15
6	0xFB	Contents of register 0x16

6.2.2.1 SPI Auto-Increment READ for Sequential Addresses Example Code

```
// (note that this function can be used to read a single byte or multiple; making the
RAP_ReadByte() function redundant)
void RAP_ReadBytes(uint8_t address, uint8_t * data, uint8_t count) {
uint8_t i = 0;

    Assert_CS();
    SPI_Transfer(address | 0xA0); // Send register address OR'd with read-mask (0xA0)
    SPI_Transfer(0xFC); // Special auto-increment filler byte
    SPI_Transfer(0xFC); // Special auto-increment filler byte
    for(; i < count; i++) {
    data[i] = SPI_Transfer(0xFC); // Each transfer gets the next register's contents
    }
    DeAssert_CS();
}

// RAP_ReadBytes() example to read the entire register set into an array
uint8_t registerContents[0x17];
RAP_ReadBytes(0x00, registerContents, 0x17); // Reads 0x17 (quantity) registers, start-
ing at address 0x00
```

6.3 Writing to a Register Using SPI

Writing to a register requires two SPI exchanges. The host starts by sending the WRITE command byte (0x8X) with the desired register address. The simultaneous response byte from Pinnacle may be data from the previous command or it may be a filler byte (0xFB). The host then sends the value to be written to the register in the next byte. The simultaneous response byte is 0xFB. Multiple writes are sent as consecutive single writes (with a repeating sequence of command byte/data byte).

6.3.1 SPI Write Example

WRITE the value 0x02 to register 0x03 (SysConfig1 Shutdown Bit) (see Table 17).

1. Use OR operand to combine the WRITE command (0x8X) with the register addresses to be read.

   `0x80 | 0x03 =0x83`

2. Send 0x83. The response will be a filler byte or the response to the previous command.
3. Send the value 0x02. The value 0x02 will then be written to register 0x03. The response will be a filler byte.

Table 17. Example of SPI WRITE Command

Byte	Command (MOSI)	Response (MISO)
1	0x83	Response to a previous command or a filler byte
2	0x02	0xFB (value 0x02 is written to register 0x03)

6.3.1.1 SPI Write Example Code

```
void RAP_WriteByte(uint8_t address, uint8_t data) {
    Assert_CS();
    SPI_Transfer(address | 0x80); // Send register address OR'd with write-mask (0x80)
    SPI_Transfer(data); // Send data to be written
    DeAssert_CS();
}
```

7. Inter Integrated Circuit (I^2C) Protocol

This section provides a brief overview of the I^2C protocol as well as how to communicate to Pinnacle using I^2C.

7.1 I^2C Operation

I^2C communication is a two-wire bus that uses module addressing for communicating with multiple devices on the bus. The I^2C bus contains a data signal (SDA) and a clock signal (SCL), see Table 18. The clock is provided by the master and is an input to all slave devices. The data line is both an input and an output (bidirectional).

Table 18. Pinnacle I^2C Signals

Signals	Description
I^2C	
SDA	Serial Data line for both input and output for master and slave.
SCL	I^2C Serial Clock Line provided by the master as an input to all slave devices.
Pinnacle	
HW_DR	Indicates Pinnacle has Data Ready to send
GPIO	General Purpose Input/Output
GND	Ground
VDD	Power supply

Pinnacle is designed as a slave device on the I^2C bus. As per the I^2C protocol, each slave is assigned a unique, seven-bit slave address. All slave devices on the bus receive all commands, but only the addressed slave device can respond to communication on the bus. The default slave address for Pinnacle is 0x2A.

I^2C slave addresses are only seven bits long to allow the eighth bit to signify a READ or WRITE command. In the first byte sent by the host, the slave address comes first, followed by the READ/WRITE bit (at the lowest significant bit). Therefore, the hexadecimal value for the first byte is the slave address (0x2A) shifted up one bit (0x54), plus the READ/WRITE bit. The final value is either 0x54 for WRITE or 0x55 for READ (see Figure 4).

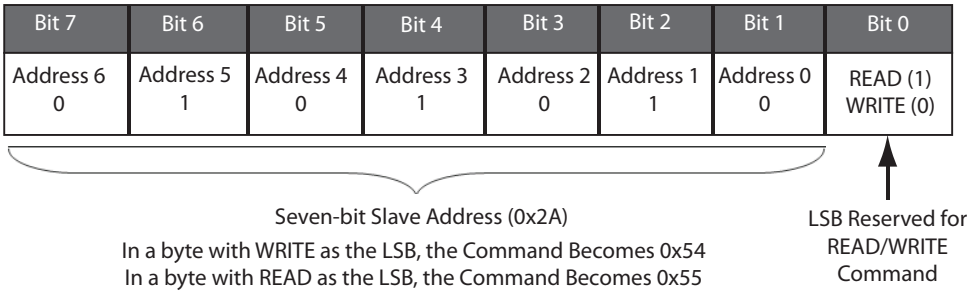

Figure 4. I^2C READ or WRITE Command with Pinnacle's Address

When the bus is idle, the SDA and SCL lines are high due to pull-up resistors on each line. The Master starts communication with a start condition, which is a high to low transition on the SDA line while the SCL line remains high. After the start signal, Data is placed on the SDA line when the SCL line is low. The receiving device latches the data when the SCL line is high. All 8-bits are transmitted and a 9th bit is designated as the acknowledge bit (ACK [0] /NACK [1]). The first byte transmitted will always be a 7-bit address of the target Slave and a READ/WRITE bit indicating the direction of data flow. Any bytes following the Slave address byte are only for the addressed Slave. When reading from the Slave, the Master must NACK the last byte to indicate to the Slave that it was the last byte to be exchanged. After at least one data byte is sent or received, the Master can cause a stop condition to free the bus. The stop condition requires SDA to transition from low to high, while SCL is high.

I^{2}C is half-duplex, and therefore responses must be read after a transmission rather than during a transfer. After a READ command, the slave hardware will ACK/NACK on the 9th bit of a byte. A NACK (9th bit asserted by slave) will denote a BUSY condition or an ERROR condition, and the slave will follow it with a status byte to indicate the error or response. A NACK from the slave requires the host to end the transfer and read a status byte carrying a response. When writing, the Slave is not required to NACK the last byte because the Slave does not know how many bytes it is to receive. A Stop Condition occurs when the SDA line rises, while the SCL line is high.

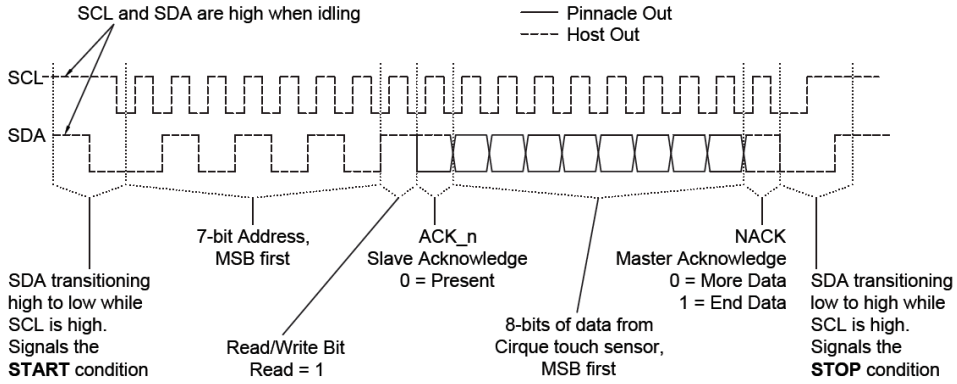

Figure 5. I^{2}C Signals - READ

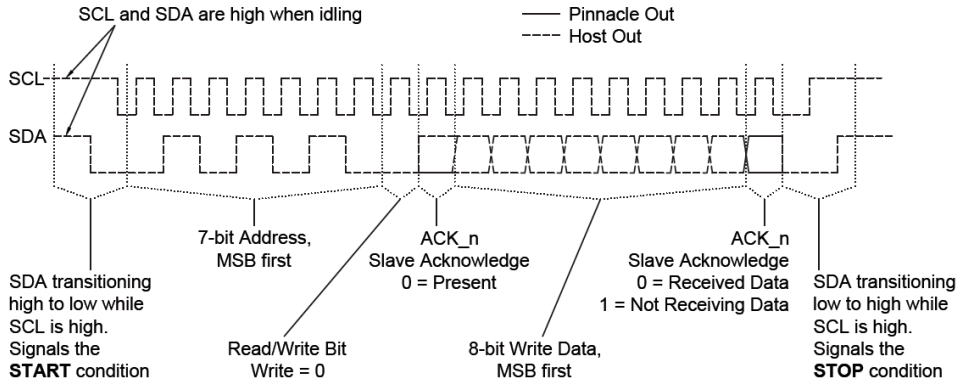

Figure 6. I^{2}C Signals - WRITE

Note: *Pinnacle is an I^{2}C Fast-Mode device. Timing information is available in the official I^{2}C specification from NXP, which can be found at this link: www.nxp.com/documents/user_manual/UM10204.pdf*

8. Register Access Protocol Using I²C

To communicate with Pinnacle using I²C requires RAP, which has only READ and WRITE commands with the byte format repeated in Table 19.

Table 19. Register Access Protocol

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
READ (0xAX)	1	0	1	Address 4	Address 3	Address 2	Address 1	Address 0
WRITE (0x8X)	1	0	0	Address 4	Address 3	Address 2	Address 1	Address 0

8.1 Reading a Register with I²C

Pinnacle stores a 'Current Read Address' (CRA) for I²C, which allows for efficient read operations. With I²C, the Host sends an I²C WRITE command to Pinnacle as the slave. The host must then send a RAP READ command to access Pinnacle registers. When the RAP READ command is received, Pinnacle stores the address it contains as the 'Current Read Address'. The host can then send I²C READ commands that will start at the 'Current Read Address' and then automatically increment to the next address until a stop condition is received. When the stop condition is received, the 'Current Read Address' is reset to the address received in the original RAP READ command.

8.1.1 I²C READ Example

Read the contents of register 0x12 (Button Status)

1. Send the start condition.
2. Send the I²C WRITE command (7-bit slave address as top 7 bits and a 0 for WRITE). This means, shift the address (0x2A) one bit.

   ```
   0x2A << 1 = 0x54
   ```

3. Use the OR operand to combine the RAP READ command (0xAX) with the register address to be read. For this example, 0x12 is used.

   ```
   0xA0|0x12=0xB2
   ```

4. Send 0xB2.
5. Send a stop condition. Pinnacle sets the “Current Read Address” to 0x12 and is now setup to respond with the contents.
6. Send another start condition, followed by the I²C READ command. (Pinnacle slave address shifted, and OR'd with a 1 to read.)

   ```
   0x2A << 1=0x54
   0x54|0x01=0x55
   ```

7. Pinnacle will respond with the contents of register 0x12.
8. After the receiving all 8 bits, send a stop condition to end communication.

8.1.2 I²C Multiple READ Example

Read the contents of register 0x14 through 0x16

1. Send the start condition.
2. Send the I²C WRITE command. Shift address (0x2A) one bit.

   ```
   0x2A << 1 = 0x54
   ```

3. Use the OR operand to combine the RAP READ command (0xAX) with the first address to be read. For this example, 0x14 is used.

   ```
   0xA0|0x14 = 0xB4
   ```

4. Send 0xB4.
5. Send a stop condition.

   Pinnacle sets the Current Read Address (CRA) to 0x14 and is now setup to respond with the contents.

6. Send another start condition, followed by the I²C READ command.

   (Pinnacle slave address shifted, and OR'd with a 1 to read.)

   ```
   0x2A << 1 = 0x54
   0x54 | 0x01 = 0x55
   ```

7. Pinnacle will respond with the contents of register 0x14 and increment CRA.
   - After the receiving all 8 bits, send another I²C READ command (0x55). Pinnacle will respond with the contents of register 0x15 and increment CRA.
   - After the receiving all 8 bits, send another I²C READ command (0x55). Pinnacle will respond with the contents of register 0x16 and increment CRA.
   - Send the stop condition to end communication. Pinnacle will reset the Current Read Address (CRA) to 0x14.

I2C Read Register

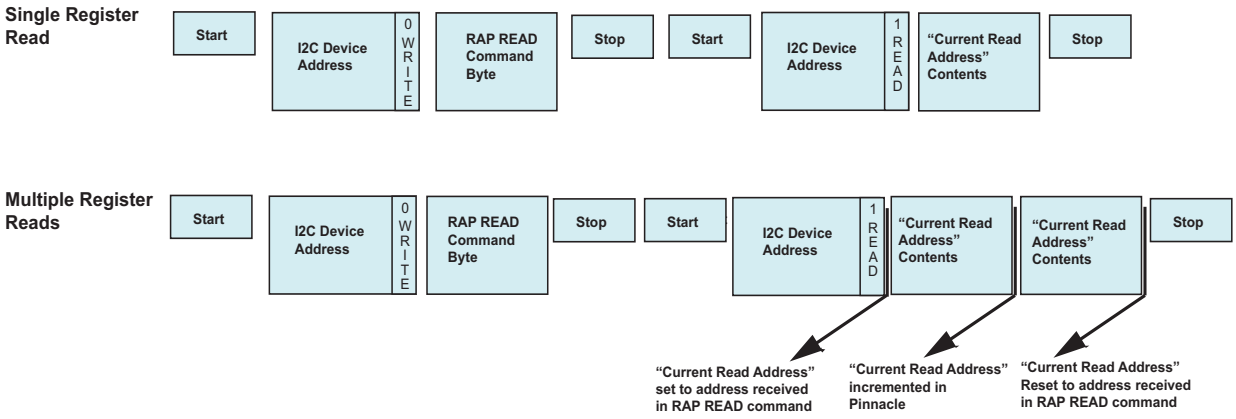

Figure 7. I²C READ Register Sequence

8.2 Writing to a Register with I²C

Writing to a register using I²C requires sending two bytes, the WRITE Command Byte and the WRITE Value Byte. The master MUST NOT issue a stop condition between the WRITE Command Byte and the WRITE Value Byte (see Figure 8).

8.2.1 I²C WRITE Example

WRITE value 0x02 to register 0x03 (SysConfig1 Shutdown Bit)

1. Send the START condition.
2. Send the I²C WRITE command (7-bit slave address as top 7 bits and a 0 for WRITE)

   Shift address (0x2A) one bit.

   ```
   0x2A << 1 = 0x54
   ```

3. Use the OR operand to combine the RAP WRITE command (0x80) with the address to be written. For this example, 0x03 is used.

   ```
   0x80|0x03=0x83
   ```

4. Send 0x83.
5. Send the value 0x02.
6. Send the stop condition to end communication.

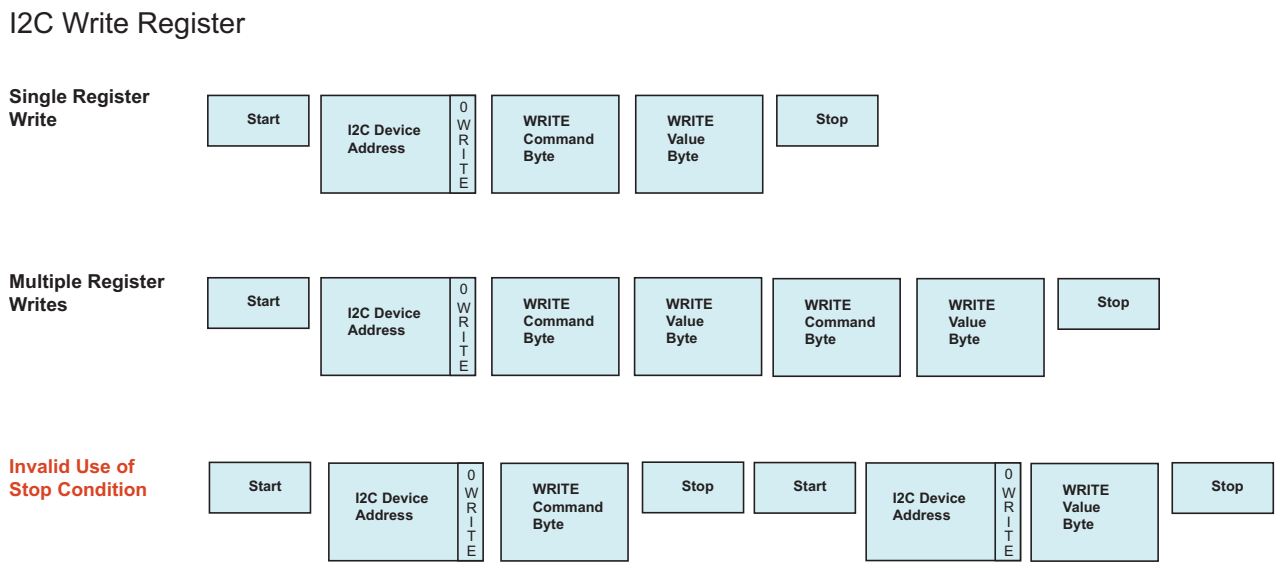

Figure 8. I²C WRITE Register Sequence

9. Appendix: Extended Register Access

Using four standard RAP registers, the host can gain Extended Register Access (ERA) to Pinnacle memory. Register 0x1B is used for the value to be written or read (see Table 20). Register 0x1C is the high byte and register 0x1D is the low byte of the address to be read or written to (see Table 21 and Table 22). Register 0x1E, the ERA control register, specifies READ or WRITE and the optional Auto-Incremented READ or WRITE for sequential commands, as well as a WRITE/Verify option if both the READ and WRITE flags are set (see Table 23 on page 30). The control register value returns to 0x00 to indicate a command is complete. Use standard Register Access Protocol (RAP) to access these four registers.

It is important to note that accessing the extended registers will assert the command complete (SW_CC) flag and force the hardware data ready (HW_DR) pin high. The Pinnacle data feed should be disabled before accessing extended registers, and SW_CC should be cleared when the host is finished accessing extended registers.

Table 20. Extended Register Access Value - (RAP register 0x1B)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Value 7	Value 6	Value 5	Value 4	Value 3	Value 2	Value 1	Value 0
Read/Write	R/W							
Values	0x00 - 0xFF							
Default	0							

Table 21. Extended Register Access Address High Byte - (RAP register 0x1C)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Address15	Address14	Address13	Address12	Address11	Address10	Address9	Address8
Read/Write	R/W							
Values	0x00 - 0xFF							
Default	0							

Table 22. Extended Register Access Address Low Byte - (RAP register 0x1D)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Address 7	Address 6	Address 5	Address 4	Address 3	Address 2	Address 1	Address 0
Read/Write	R/W							
Values	0x00 - 0xFF							
Default	0							

Table 23. Extended Register Access Control - (RAP register 0x1E)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description					WRITE Auto-Increment	READ Auto-Increment	Write	Read
Read/Write					R/W	R/W	R/W	R/W
Values					1=enabled 0=disabled	1=enabled 0=disabled	1 = WRITE	1 = Read
Default					0	0	0	0

This register value returns to 0x00 to indicate a completed command

Asserting both Bit[1] and Bit[0] indicates a WRITE/Verify

9.1 Extended Register Access Examples

Using standard RAP to send READ and WRITE commands to Pinnacle, the following examples demonstrate the proper sequence to read and write to Pinnacle's extended registers.

Example: READ an Extended Register

1. WRITE the high byte of the 16-bit extended register address to RAP Register 0x1C (ERA High Byte).
2. WRITE the low byte of the 16-bit extended register address to RAP Register 0x1D (ERA Low Byte).
3. WRITE 0x01 (ERA READ flag) to RAP Register 0x1E (ERA Control).
4. READ the RAP Register 0x1E (ERA Control) until it contains 0x00.
5. READ the new value in RAP Register 0x1B (ERA Value).
6. WRITE 0x00 to RAP Register 0x02 (Status1) to clear Command Complete (SW_CC).

Example: READ an Extended Register with Address Increment

1. WRITE the high byte of the 16-bit extended register address to RAP Register 0x1C (ERA High Byte).
2. WRITE the low byte of the 16-bit extended register address to RAP Register 0x1D (ERA Low Byte).
3. WRITE 0x05 to RAP Register 0x1E (ERA Control) to specify auto-increment read.

   Repeat 4, 5, and 6 as needed
4. READ the RAP Register 0x1E (ERA Control) until it contains 0x00.
5. READ the new value in RAP Register 0x1B (ERA Value).
6. WRITE 0x00 to RAP Register 0x02 (Status1) to clear Command Complete (SW_CC).

   Extended Register Address is incremented. Repeat steps 4, 5, and 6 to reach the desired address.

Example: WRITE to an Extended Register

1. WRITE the value to be written in RAP Register 0x1B (ERA Value)
2. WRITE the high byte of the 16-bit extended register address to RAP Register 0x1C (ERA High Byte).
3. WRITE the low byte of the 16-bit extended register address to RAP Register 0x1D (ERA Low Byte).
4. WRITE 0x02 (ERA WRITE flag) to RAP Register 0x1E (ERA Control).
5. READ the RAP Register 0x1E (ERA Control) until it contains 0x00.
6. WRITE 0x00 to RAP Register 0x02 (Status1) to clear Command Complete (SW_CC).

Example: WRITE to an Extended Register with Address Increment

1. WRITE the value to be written in RAP Register 0x1B (ERA Value)
2. WRITE the high byte of the 16-bit extended register address to RAP Register 0x1C (ERA High Byte).
3. WRITE the low byte of the 16-bit extended register address to RAP Register 0x1D (ERA Low Byte).
4. WRITE 0x0A (ERA auto-increment WRITE) to RAP Register 0x1E (ERA Control).

   Repeat Steps 5 and 6 as needed
5. READ the RAP Register 0x1E (ERA Control) until it contains 0x00.
6. WRITE 0x00 to RAP Register 0x02 (Status1) to clear Command Complete (SW_CC).

   Extended Register Address is incremented. Repeat steps 5 and 6 to reach the desired address.

Example: WRITE to an Extended Register with Verification

1. WRITE the value to be written in RAP Register 0x1B (ERA Value)
2. WRITE the high byte of the 16-bit extended register address to RAP Register 0x1C (ERA High Byte).
3. WRITE the low byte of the 16-bit extended register address to RAP Register 0x1D (ERA Low Byte).
4. WRITE 0x03 (write and read) to RAP Register 0x1E (ERA Control).
5. READ the RAP Register 0x1E (ERA Control) until it contains 0x00.
6. READ the value in RAP Register 0x1B (ERA Value) to verify.
7. WRITE 0x00 to RAP Register 0x02 (Status1) to clear Command Complete (SW_CC).

9.2 Example Code

Example code for reading and writing extended registers is available on Cirque's online code repository located at GitHub. Follow the link below for access:

https://github.com/cirque-corp/Pinnacle_1CA027/blob/master/Circular_Trackpad/TM040040/Example_Code/SPI_Demo/TM040040_SPI.ino

10. Contact Information

Contact a Cirque sales representative for a complete list of Cirque's OEM products.

In United States & Canada	(800) GLIDE-75 (454-3375)
Outside US & Canada	(801) 467-1100
Fax	(801) 467-0208
Web site	http://www.cirque.com