



**VIA Networking Technologies, Inc.**

**VT6651/VT6655/VT6656**

**API Function Specification**

Revision 2.17  
February 15, 2006

**VIA Networking Technologies, INC.**

## Copyright Notice:

Copyright © 2003, VIA Networking Technologies, Incorporated. All Rights Reserved.

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise without the prior written permission of VIA Networking Technologies, Incorporated.



is a registered trademark of VIA Networking Technologies, Incorporated.

All trademarks are the properties of their respective owners.

## Disclaimer Notice:

No license is granted, implied or otherwise, under any patent or patent rights of VIA Networking Technologies Inc. VIA Networking Technologies Inc. makes no warranties, implied or otherwise, in regard to this document and to the products described in this document. The information provided by this document is believed to be accurate and reliable as of the publication date of this document. However, VIA Networking Technologies Inc. assumes no responsibility for any errors in this document. Furthermore, VIA Networking Technologies Inc. assumes no responsibility for the use or misuse of the information in this document and for any patent infringements that may arise from the use of this document. The information and product specifications within this document are subject to change at any time, without notice and without obligation to notify any person of such change.

## Offices:

### USA Office:

940 Mission Court  
Fremont, CA 94539  
USA  
Tel: (510) 683-3300  
Fax: (510) 683-3301 -or- (510) 687-4654  
Web: <http://www.vntek.com>

### Taipei Office:

8th Floor, No. 533  
Chung-Cheng Road, Hsin-Tien  
Taipei, Taiwan ROC  
Tel: (886-2) 2218-2011  
Fax: (886-2) 2219-8461  
Web: <http://www.vntek.com.tw>

### REVISION HISTORY

Document Release	Date	Revision	Initials
1.00	2/26/04	Initial release.	YC
1.01	3/04/04	Functions improvement.	YC
1.02	4/14/04	Document correction. Add a function for checksum calculation.	YC
1.03	4/20/04	Add DUT existence checking function	Kyle Hsu
1.04	5/6/04	Add functions for Carrier Suppression and amount of receiving packets with crc ok. Add UW2451 RF module support for VT6655.	YC
1.05	5/14/04	Add support of VT3253.	YC
1.06	6/03/04		YC
1.07	6/11/04	Add Multi-open DUT support. It needs driver's support.	YC
2.00	2/15/06	Modify API definition for VT6655 and VT6651	Yiching
2.01	7/23/04	Add VT6656 support.	YC
2.02	7/30/04	Add OpenDUTwithDescription(). Add card type "All" to IsDUTExist().	YC
2.03	8/13/04	Add SetTxPowerControlbyDbm() Modified SendPacket(). Add SetPacketTx().	YC
2.05	8/20/04	Add AIROHA RF support	YC
2.06	9/16/04	Add EEPROM Power Control Access. Add VT6655 Default Power Tuning.	YC
2.07	9/21/04	WriteAutoPowerControl() and ReadAutoPowerControl() updated	YC
2.08	11/12/04	Add power tuning support for UBEC RF module. SetTxPowerControlbyPwrMeter() updated.	YC
2.09	11/17/04	Add MAXIM2829 RF module support. WriteCISData() and ReadCISData() updated.	YC
2.11	03/14/05	Add WriteEEPromEX function.	Kyle Hsu
2.12	03/23/05	Fix WriteAutoPowerControl function.	Kyle Hsu
2.14	05/04/05	1. Remove SN number access. 2. Always check sum update with all EEPROM access function. 3. Add power level per channel for 5G band.	Yiching
2.16	10/19/05	1. Add RF type. 2. Modify WriteEEPromEX function. 3. Fix dynamic-linking bug.	Alan C. Lin
2.17	01/26/06	1. Rename Write(Read)AutoPowerControl function Write(Read)PowerControlEx 2. Add Tx/Rx packet sample code.	Lyndon

## TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION .....</b>	<b>3</b>
1.1	GENERAL FUNCTION REQUIREMENT .....	3
1.2	EEPROM FLASH/DOWNLOAD FUNCTION REQUIREMENT .....	7
1.3	PACKET ERROR RATIO TEST FUNCTION REQUIREMENT .....	9
<b>2</b>	<b>CALLING SEQUENCE.....</b>	<b>10</b>
2.1	RECEIVE PACKET CALLING SEQUENCE.....	10
2.2	TRANSMIT PACKET CALLING SEQUENCE .....	11
2.3	CONTINUOUS TRANSMIT CALLING SEQUENCE.....	11
2.4	SINGLE CARRIER TRANSMIT CALLING SEQUENCE .....	12
2.5	CARRIER SUPPRESSION TRANSMIT CALLING SEQUENCE .....	12
2.6	VT6655/VT6656 DEFAULT POWER TUNING GUIDE .....	12
<b>3</b>	<b>SAMPLE CODE .....</b>	<b>13</b>
3.1	TX PACKET SAMPLE CODE .....	13
3.2	RX PACKET SAMPLE CODE.....	14

# 1 INTRODUCTION

## 1.1 General Function Requirement

- **void\* OpenDUT(int Mode, int CardType, unsigned long dwLocationID, unsigned long dwLocationIDMask):**

Open the connection with DUT and get the session/handle of the DUT. This function will return the handle of **AdapterObject** or NULL.

**Mode** : 0 is Non-GoldenDUT, 1 is GoldenDUT,

**CardType** : 0 is VT6651(802.11b), 1 is VT6655 (802.11g) , 2 is VT6656.

**dwLocationID** : LocationID of DUT, you can use **IsDUTExit** function to get LocationID of all DUT on your system. If **dwLocationID** is not zero it will find the matched DUT which (location&**dwLocationIDMask**) equal to **dwLocationID**.

**dwLocationIDMask** : LocationIDMask of DUT, if you have only one DUT on your system you can set **dwLocationID** and **dwLocationIDMask** to zero to ignore DUT location..

- **void\* OpenDUTwithDescription(int Mode, char\* strDescription, unsigned long dwLocationID, unsigned long dwLocationIDMask):**

Open the connection with DUT and get the session/handle of the DUT. This function will return the handle of **AdapterObject** or NULL.

**Mode** : 0 is Non-GoldenDUT, 1 is GoldenDUT,

**strDescription**: Description name of specific device.

**dwLocationID** : LocationID of DUT, you can use **IsDUTExit** function to get LocationID of all DUT on your system. If **dwLocationID** is not zero it will find the matched DUT which (location&**dwLocationIDMask**) equal to **dwLocationID**.

**dwLocationIDMask** : LocationIDMask of DUT, if you have only one DUT on your system you can set **dwLocationID** and **dwLocationIDMask** to zero to ignore DUT location..

- **int CloseDUT(void\* AdapterObject):**

Close the connection with DUT and release the session/handle of the DUT. This function will return an integer to show operation success (1) or fail (0).

- **int IsDUTExist(unsigned long \*LocationLists, int MaxLocationList, int CardType, unsigned long dwLocationID, unsigned long dwLocationIDMask):**

Check if DUT is enable(existent) or disable(nonexistent). The return value is total number of DUTs found on your system.

**LocationLists** : pointer to an array of LocationID which will be set if this function find DUT.

**MaxLocationList** : Total number of **LocationLists**.

**CardType** : 0 is VT6651(802.11b), 1 is VT6655 (802.11g), 2 is VT6656, and -1 is all types.

**dwLocationID** : If **dwLocationID** is not zero it will find the matched DUT which (location&  
**dwLocationIDMask**) equal to **dwLocationID**.

**dwLocationIDMask** : LocationIDMask of DUT, you can set **dwLocationID** and **dwLocationIDMask** to zero to find all DUTs on your system.

– **int ResetTest(void\* AdapterObject) :**

Reset the DUT hardware. It will reload EEPROM settings to DUT. This function will return an integer to show operation success (1) or fail (0).

– **int GetDeviceVendorID (void\* AdapterObject, unsigned long \*DeviceVendorID) :**

Get device and Vender ID. This function will return an integer to show operation success (1) or fail (0).

**DeviceVenderID**: the device and vender ID returned.

– **int SetDataRate(void\* AdapterObject, int Rate):**

Set the transmit data rate of the DUT. This function will return an integer to show operation success (1) or fail (0).

**Rate** : 0 is 1Mbps, 1 is 2Mbps, 2 is 5.5Mbps, 3 is 11Mbps, 4 is 6Mbps, 5 is 9Mbps, 6 is 12Mbps, 7 is 18Mbps, 8 is 24Mbps, 9 is 36Mbps. 10 is 48Mbps, and 11 is 54Mbps.

– **int SetNetworkType(void\* AdapterObject, int Mode):**

Set the 802.11a/b/g network type of the DUT. This function will return an integer to show operation success (1) or fail (0).

**Mode** : 0 is 802.11b, 1 is 802.11a, 2 802.11g.

– **int GetNetworkType(void\* AdapterObject, int \*Mode):**

Get the 802.11a/b/g network type of the DUT. This function will return an integer to show operation success (1) or fail (0).

**Mode** : 0 is 802.11b, 1 is 802.11a, 2 802.11g.

– **int SetChannelDirectCall(void\* AdapterObject, int Channel):**

Set the DUT to transmit and receive on a specific channel (2.4GHz : 1~14, 5GHz: 15~56). This function will return an integer to show operation success (1) or fail (0).

**Channel** :

Value	Frequency	Value	Frequency	Value	Frequency
1	FR_2412MHZ	2	FR_2417MHZ	3	FR_2422MHZ
4	FR_2427MHZ	5	FR_2432MHZ	6	FR_2437MHZ,
7	FR_2442MHZ	8	FR_2447MHZ	9	FR_2452MH

10	FR_2457MHZ	11	FR_2462MHZ	12	FR_2467MHZ
13	FR_2472MHZ	14	FR_2484MHZ	15	FR_4915MHZ
16	FR_4920MHZ	17	FR_4925MHZ	18	FR_4935MHZ
19	FR_4940MHZ	20	FR_4945MHZ	21	FR_4960MHZ
22	FR_4980MHZ	23	FR_5035MHZ	24	FR_5040MHZ
25	FR_5045MHZ	26	FR_5055MHZ	27	FR_5060MHZ
28	FR_5080MHZ	29	FR_5170MHZ	30	FR_5180MHZ
31	FR_5190MHZ	32	FR_5200MHZ	33	FR_5210MHZ
34	FR_5220MHZ	35	FR_5230MHZ	36	FR_5240MHZ
37	FR_5260MHZ	38	FR_5280MHZ	39	FR_5300MHZ
40	FR_5320MHZ	41	FR_5500MHZ	42	FR_5520MHZ
43	FR_5540MHZ	44	FR_5560MHZ	45	FR_5580MHZ
46	FR_5600MHZ	47	FR_5620MHZ	48	FR_5640MHZ
49	FR_5660MHZ	50	FR_5680MHZ	51	FR_5700MHZ
52	FR_5745MHZ	53	FR_5765MHZ	54	FR_5785MHZ
55	FR_5805MHZ	56	FR_5825MHZ		

– **int SetSleepModeDirectCall(void\* AdapterObject):**

Set the DUT to the sleep mode for testing. This function will return an integer to show operation success (1) or fail (0).

– **int SetWakeModeDirectCall(void\* AdapterObject):**

Set the DUT wake up from the sleep mode. This function will return an integer to show operation success (1) or fail (0).

– **int SetPacketTx(void\* AdapterObject, int Mode):**

This function sets the DUT to start(**Mode** is 1) or stop(**Mode** is 0) the transmission mode. This function will return an integer to show operation success (1) or fail (0). The following tells the calling sequence to send packets.

```
SetPacketTx(AdapterObject, 1); // Tx mode ON
```

```
SendPacket(AdapterObject, ...);
```

```
SetPacketTx(AdapterObject, 0); // Tx mode OFF
```

– **int SetTxContinuousDirectCall(void\* AdapterObject, int Mode):**

This function starts(**Mode** is 1) or stops(**Mode** is 0) the DUT to perform continuous transmission. This function will return an integer to show operation success (1) or fail (0).

– **int SetSingleCarrierTxContinuous(void\* AdapterObject, int Mode):**

This function starts(**Mode** is 1) or stops(**Mode** is 0) the DUT to perform single carrier continuous transmission. This function will return an integer to show operation success (1) or fail (0).

– **int SetCarrierSuppressionTx(void\* AdapterObject, int Mode):**

This function starts(**Mode** is 1) or stops(**Mode** is 0) the DUT to perform carrier suppression transmission. This function will return an integer to show operation success (1) or fail (0).

– **int SetPreamble(void\* AdapterObject, int Mode):**

Set the DUT preamble to long(**Mode** is 1) or short(**Mode** is 2). This function will return an integer to show operation success (1) or fail (0).

– **int SetTxPowerControl(void\* AdapterObject, int Value):**

Set the DUT Control Register corresponding to the MANUAL POWER CONTROL to the desired value. This function will return an integer to show operation success (1) or fail (0). The power control ranges are as followed,

**VT6651** with **AIROHA** RF : 0 ~ 30.

**VT6655/VT6656** with **RFMD** RF : 0 ~ 30.

**VT6655/VT6656** with **AIROHA** RF : 0 ~ 63.

**VT6655/VT6656** with **MAXIM** RF : 0 ~ 63.

– **int GetTxPowerControl(void\* AdapterObject, int \*Value):**

This is only used for VT6651 device. Get the DUT Control Register value corresponding to the MANUAL POWER CONTROL. The return value is from 0 to 30. The return value –1 means **Unavailable**. This function will return an integer to show operation success (1) or fail (0).

– **int SetAntennaBB(void\* AdapterObject, int Mode):**

Set the antenna mode of the DUT. This function will return an integer to show operation success (1) or fail (0).

**Mode** : 0 is both TX and RX use antenna A, and 1 is both TX and RX use antenna B.

– **int SetTxPowerControlbyPwrMeter(void\* AdapterObject, int InitLv1, int MeasuredDbm1, int InitLv2, int MeasuredDbm2, int TargetDbm, int\* TargetLv):**

This is used for VT6655/VT6656 devices. Set the DUT Control Register corresponding to the MANUAL POWER CONTROL to the desired value. This function will return an integer to show operation success (1) or fail (0). Please reference 2.6 VT6655/VT6656 Default Power Tuning Guide.

**InitLv1** : First initial input power lcontrol.

**MeasuredDbm1** : The power value (**0.01 dBm**) getting from Power Meter after setting power control to **nitLv1**.



**InitLv2** : Second initial input power lcontrol.

**MeasuredDbm2** : The power value (**0.01 dBm**) getting from Power Meter after setting power control to **InitLv2**.

**TragetDbm** : Target power value (**0.01 dBm**).

**TargetLv** : The return target power control.

## 1.2 EEPROM Flash/Download Function Requirement

### – int SetFilterBB(void\* AdapterObject, int Zone):

This function sets the DUT to use USA(0), JPN(1) or EUR(2) zone filter. In 2.4GHz frequency, the channel range for USA zone type are ch1 ~ ch11, JPN zone type are ch1 ~ch14, and EUR type are ch1~ch13. Default zone type is USA. This function will return an integer to show operation success (1) or fail (0).

**Zone** : 0 is USA, 1 is JPN, and 2 is EUR.

### – int GetFilterBB(void\* AdapterObject, int\* Zone):

Get the current zone type for this DUT. This function will return an integer to show operation success (1) or fail (0).

**Zone** : 0 is USA, 1 is JPN, and 2 is EUR.

### – int WriteMACAddress(void\* AdapterObject, unsigned char\* MACAddress):

Write the MAC address to the DUT EEPROM. This function will return an integer to show operation success (1) or fail (0).

### – int ReadMACAddress(void\* AdapterObject, unsigned char\* MACAddress):

Read the MAC address from the DUT EEPROM. This function will return an integer to show operation success (1) or fail (0).

### – int WriteCISData(void\* AdapterObject, unsigned char\* CISData, int Length):

Write the CIS data to the DUT EEPROM. The length should be not more than 128 bytes. This function will return an integer to show operation success (1) or fail (0). For VT6651 and VT6656, the length should be 127 bytes. For VT6655, the length should be 82 bytes.

### – int ReadCISData(void\* AdapterObject, unsigned char\* CISData, int Length, int\* pLengthOut):

Read the CIS data from the DUT EEPROM. This function will return an integer to show operation success (1) or fail (0).

### – int WritePowerDataArray(PVOID AdapterObject, char\* PowerDataArray):

This is only used for VT6651 device. Write the power control of all the channel(1-14) into EEPROM. PowerControl is a 28-byte char array to store the power control of all the channel. Please reference ReadPowerControl( ) for detail.

– **int ReadPowerDataArray(PVOID AdapterObject, char\* PowerDataArray):**

This is only used for VT6651 device. Read the power control of all the channel(1-14) from EEPROM. PowerControl is a 28-byte char array to store the power control of all the channel, and it contains {Ch1 for Rate1&2, Ch2 for Rate1&2, ..., Ch14 for Rate1&2, Ch1 for Rate5.5&11, Ch2 for Rate5.5&11, ..., Ch14 for Rate5.5&11}. Each item is one-byte long.

– **int WritePowerControl(void\* AdapterObject, int Channel, int RateSelect, int PowerControl):**

This is only used for VT6651 device. Write the power control into EEPROM. This function will return an integer to show operation success (1) or fail (0).

**Channel** : 1 ~ 14.

**RateSelect** : 0 means 1Mbps & 2 Mbps, 1 means 5 Mbps & 11 Mbps.

**PowerControl** : Power control value.

– **int ReadPowerControl(void\* AdapterObject, int Channel, int RateSelect, int\* PowerControl):**

This is only used for VT6651 device. Read the power control from EEPROM. This function will return an integer to show operation success (1) or fail (0).

**Channel** : 1 ~ 14.

**RateSelect** : 0 means 1Mbps & 2 Mbps, 1 means 5 Mbps & 11 Mbps.

**PowerControl** : Return power control value.

– **int WritePowerControlEx(void\* AdapterObject, int Mode, int Channel, int PowerControl, int PowerOutput):**

This is used for VT6655/VT6656 devices. Write the power control into EEPROM. This function will return an integer to show operation success (1) or fail (0).

**Mode** : 0 is CCK and 1 is OFDM

**Channel** : 2.4GHz : 1 ~ 14, 5GHz : 15 ~ 56.

**PowerControl** : Power Level form 0 to 63.

**PowerOutput** : Output Power \* 16 (dBm).

– **int ReadPowerControlEx(void\* AdapterObject, int Mode, int Channel, int\* PowerControl, int\* PowerOutput):**

This is used for VT6655/VT6656 devices. Read the power control from EEPROM. This function will return an integer to show operation success (1) or fail (0).

**Mode** : 0 is CCK and 1 is OFDM

**Channel** : 2.4GHz : 1 ~ 14, 5 GHz : 15 ~ 56.

**PowerControl** : Power Level form 0 to 63.

**PowerOutput** : Output Power \* 16 (dBm).

– **int WriteEeprom(void\* AdapterObject, unsigned char\* Eeprom, int Length):**

Write the Eeprom content into DUT EEPROM. The length of "Eeprom" should be 256 bytes. This function will return an integer to show operation success (1) or fail (0).

– **int ReadEeprom(void\* AdapterObject, unsigned char\* Eeprom, int Length, int\* pLengthOut):**

Read the Eeprom content from DUT EEPROM. The length of "Eeprom" should be 256 bytes. This function will return an integer to show operation success (1) or fail (0).

– **int WriteEepromEX(void\* AdapterObject, unsigned char\* pbyInBuffer, unsigned char\* pbyOutBuffer, unsigned long uICmd):**

Write the Eeprom content into DUT EEPROM. The length of "pbyInBuffer" and "pbyOutBuffer" should be 256 bytes. This function will return an integer to show operation success (1) or fail (0).

**pbyInBuffer:** The buffer that user want to write to EEPROM.

**pbyOutBuffer:** The function will copy the final content writtrn to EEPROM to the buffer.

**UICmd:** The command defines the items that don't need to write to EEPROM.

<b>EELOAD_IGNORE_ADDR</b>	<b>0x00000001</b>	: Ignore MAC address
<b>EELOAD_IGNORE_PWR</b>	<b>0x00000002</b>	: Ignore Power setting
<b>EELOAD_IGNORE_ZONE</b>	<b>0x00000008</b>	: Ignore Zone Type setting
<b>EELOAD_IGNORE_RFTYPE</b>	<b>0x00000010</b>	: Ignore RF Type setting
<b>EELOAD_IGNORE_ANTENNA</b>	<b>0x00000020</b>	: Ignore Antenna setting
<b>EELOAD_IGNORE_RADIOCTL</b>	<b>0x00000040</b>	: Ignore Radio Control setting

### 1.3 Packet Error Ratio Test Function Requirement

– **int SendPacket(void\* AdapterObject, int iRepeat, int iPktPerGrp, int iGrpDelay, unsigned short wPatternType, unsigned int uPayloadLen, unsigned char\* pbyDestAddress):**

Start sending packets. This function will return an integer to show operation success (1) or fail (0).

**iRepeat** : total packet number to be sent.

**iPktPerGrp** : packet number in a group.

**iGrpDelay** : group delay time in milli-sec.

**wPatternType** : The lower byte of wPatternType presents the PATTERN to be stored in the payload, and the upper byte of wPatternType presents PatternType (0: Fixed PATTERN is used, 1: Increment from 0x00, 2: Random Select, PATTERN is not used).

**uPayloadLen** : Length of packet payload.

**pbyDestAddress** : destination address of packet.

The following tells the calling sequence to send packets.

```
SetPacketTx(AdapterObject, 1); // Tx mode ON
```

```
SendPacket(AdapterObject, ...);
```

```
SetPacketTx(AdapterObject, 0); // Tx mode OFF
```

– **int ReceivePacket(void\* AdapterObject, int Mode, int Filter):**

**Mode** : Start(1) or Stop(0) receiving packets. This function will return an integer to show operation success (1) or fail (0).

**Filter** : 0 is all MAC Address, and 1 is only DUT MAC Address.

– **int ResetTxPacketSent(void\* AdapterObject):**

Reset the transmitted packet count for the adapter. This function will return an integer to show operation success (1) or fail (0).

– **int ResetRxPacketReceived(void\* AdapterObject):**

Reset the received packet count for the adapter. This function will return an integer to show operation success (1) or fail (0).

– **int QueryTxPacketSent(void\* AdapterObject, int \* TxPacketCount):**

Query for the packet amount transmitted for the packet error ratio test. This function will return an integer to show operation success (1) or fail (0).

– **int QueryRxPacketReceive(void\* AdapterObject, int\* RxPacketCount, int\* RxPacketCrcOKCount):**

Query for the packet amount received. RxPacketCrcOKCount only counts packets with CRC ok. This function will return an integer to show operation success (1) or fail (0).

– **int ResetTxRxPacketStatistic(void\* AdapterObject):**

Reset Both the transmitted packet count and received packet count for the DUT. This function will return an integer to show operation success (1) or fail (0).

## 2 CALLING SEQUENCE

### 2.1 Receive Packet Calling Sequence

1. ResetRxPacketReceive (...)
2. SetNetworkType(...)
3. SetAntennaBB (...)
4. SetChannelDirectCall (...)

5. ReceivePacket (... , 1) //Start receiving.
6. QueryRxPacketReceive(...)
- .....
7. ReceivePacket (... , 0) //Stop receiving.

## 2.2 Transmit Packet Calling Sequence

1. ResetTxPacketSent (...)
2. SetNetworkType(...)
3. SetAntennaBB (...)
4. SetChannelDirectCall (...)
5. SetDataRate (...)
6. SetPreamble (...)
7. SetTxPowerControl (...)
8. SetPacketTx(..., 1) // Set Tx mode On
9. SendPacket (...)
- .....
10. QueryTxPacketSent (...)
11. SetPacketTx(..., 0) // Set Tx mode Off

## 2.3 Continuous Transmit Calling Sequence

1. ResetTxPacketSent (...)
2. SetNetworkType(...)
3. SetAntennaBB (...)
4. SetChannelDirectCall (...)
5. SetDataRate (...)
6. SetPreamble (...)
7. SetTxPowerControl (...)
8. SetTxContinuousDirectCall (... , 1) //Start Continuous Tx

.....

9. SetTxContinuousDirectCall (... , 0) //Stop Continuous Tx

## 2.4 Single Carrier Transmit Calling Sequence

1. ResetTxPacketSent (...)

2. SetNetworkType(...)

3. SetAntennaBB (...)

4. SetChannelDirectCall (...)

5. SetPreamble (...)

6. SetTxPowerControl (...)

7. SetSingleCarrierTxContinuous (... , 1) //Start Single Carrier Tx

.....

8. SetSingleCarrierTxContinuous (... , 0) //Stop Single Carrier Tx

## 2.5 Carrier Suppression Transmit Calling Sequence

1. ResetTxPacketSent (...)

2. SetNetworkType(...)

3. SetAntennaBB (...)

4. SetChannelDirectCall (...)

5. SetPreamble (...)

6. SetTxPowerControl (...)

7. SetCarrierSuppressionTx (... , 1) //Start Carrier Suppression Tx

.....

8. SetCarrierSuppressionTx (... , 0) //Stop Single Suppression Tx

## 2.6 VT6655/VT6656 Default Power Tuning Guide

1. Config DataRate and Channel

- SetChannelDirectCall(AdapterObject, Channel);

- SetDataRate(AdapterObject, DataRate);
- 2. Set Power Control to the first initial power control, **DefaultLv1**.
  - SetTxPowerControl(AdapterObject, **DefaultLv1**);
- 3. Do ContinuousTx Test
  - SetTxContinuousDirectCall(void\* AdapterObject, 1); // Cont TX ON
  - Get **MeasuredPowerDbm1** from power meter.
  - SetTxContinuousDirectCall(void\* AdapterObject, 0); // Cont TX OFF
- 4. Set Power Control to the second initial power control, **DefaultLv2**.
  - SetTxPowerControl(AdapterObject, **DefaultLv2**);
- 5. Do ContinuousTx Test
  - SetTxContinuousDirectCall(void\* AdapterObject, 1); // Cont TX ON
  - Get **MeasuredPowerDbm2** from power meter.
  - SetTxContinuousDirectCall(void\* AdapterObject, 0); // Cont TX OFF
- 6. Get the **TargetPwrLv** for the **TragetDbm**.
  - SetTxPowerControlbyPwrMeter (AdapterObject, **DefaultLv1**, (int)MeasuredPowerDbm1\*100, **DefaultLv2**, (int)MeasuredPowerDbm2\*100, **TragetDbm\*100**, &TargetPwrLv);
- 7. Use the **TargetPwrLv** to test if **Signal Spectrum Mask** and **EVM** are pass or failure.
- 8. Save **TargetPwrLv** to EEPROM if test is pass.
  - WritePowerControlEx(AdapterObject, Mode, Channel, **TargetPwrLv**, **TragetDbm\*16**);

### 3 SAMPLE CODE

#### 3.1 TX Packet sample code

```
void vStartPktTxSample(AdapterObject)
{
    unsigned char abyBroadcastAddr[6]; //{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
    int m_Int_Diag_Ant = 0x01; //0x01 main ANT
    int m_Int_Diag_Chann = 0x06; //channel 6
    int m_Int_Diag_Rate = 0x00; //RATE 1M
    int m_Int_Diag_Pream = 0x01; //long preamble
    int m_Int_packet_count = 0;

    for(int ii = 0 ; ii < 6; ii ++ )
        abyBroadcastAddr[iii] = 0xFF;
```

```

if (ResetTxPacketSent(AdapterObject) == 0) //Reset TX
    vDebugPrint(_T("RestTxPacketSent failed"), DBG_NORMAL);

if (SetAntennaBB(AdapterObject, m_Int_Diag_Ant) == 0)
    vDebugPrint(_T(" SetAntennaBB failed"), DBG_NORMAL);

if (SetChannelDirectCall(AdapterObject, m_Int_Diag_Chan) == 0)
    vDebugPrint(_T(" SetChannelDirectCall failed, please check Zone Type !"), DBG_NORMAL);

if (SetDataRate(AdapterObject, m_Int_Diag_Rate) == 0)
    vDebugPrint(_T(" SetDataRate failed"), DBG_NORMAL);

if (SetPreamble(AdapterObject, m_Int_Diag_Pream) == 0)
    vDebugPrint(_T(" SetPreamble failed"), DBG_NORMAL);

// Use default tx power in EEPROM.
// SetTxPowerControl(AdapterObject, m_iTxPower);
//

if (ResetTxRxPacketStatistic(AdapterObject) == 0)
    vDebugPrint(_T(" RestTxRxPacketStatistic failed"), DBG_NORMAL);

// Set Tx On
if (SetPacketTx(AdapterObject, 1) == 0)
    vDebugPrint(_T(" Start SetPacketTx failed"), DBG_NORMAL);

// Total packet number =1, packet in group = 1, group delay = 0,
// pattern =0x0c, payload len = 128
SendPacket(AdapterObject, 1, 1, 0, 0x0c, 0x080, &abyBroadcastAddr[0]);

// Wait until send operation complete since packet sending process
// is under kernel driver thread.

QueryTxPacketSent(AdapterObject, &m_Int_packet_count);

// Set Tx Stop
if (SetPacketTx(AdapterObject, 0) == 0)
    vDebugPrint(_T(" Stop SetPacketTx failed"), DBG_NORMAL);
}

```

## 3.2 RX Packet sample code

```

void vStartPktRxSample(AdapterObject)
{
    int m_Int_Diag_Ant = 0x01; //0x01 main ANT
    int m_Int_Diag_Chan = 0x06; //channel 6
    int m_Int_rx_count, m_Int_rx_crc_ok;

    if (ResetRxPacketReceive(AdapterObject) == 0)
        vDebugPrint(_T(" RestRxPacketSent failed"), DBG_NORMAL);

    if (SetAntennaBB(AdapterObject, m_Int_Diag_Ant) == 0)
        vDebugPrint(_T(" SetAntennaBB failed"), DBG_NORMAL);

    if (SetChannelDirectCall(AdapterObject, m_Int_Diag_Chan) == 0)

```



```
vDebugPrint(_T(" SetChannelDirectCall failed"), DBG_NORMAL);

// Set start rx
ReceivePacket(AdapterObject, 1, 0);

// ...

// Wait or loop to query rx status
QueryRxPacketReceive(AdapterObject, m_Int_rx_count, m_Int_rx_crc_ok);

// Set Rx stop
if (ReceivePacket(AdapterObject, 0, 0) == 0)
    vDebugPrint(_T(" ReceivePacket Stop failed"), DBG_NORMAL);
}
```

VIA Networking  
Technologies Inc.  
Confidential  
NDA Required