Revised Date: Jan. 22, 2007

QRZ-1000A ZigBee Module

Description

The QRZ-1000A is a miniature 2.4 GHz fully ZigBee compliant module. It includes all RF hardware and a micro-controller to manage the communications link. The micro-controller manages all communications tasks including configuration, data packaging, and clear channel selection. The result is a complete ZigBee protocol wireless data communications solution.

The QRZ-1000A package is unique because of its small form factor (44 x 28 mm^2), It has an on-board chip antenna for minimizing the PCB size. No competitive products can offer a solution as flexible, convenient, and easy to integrate,

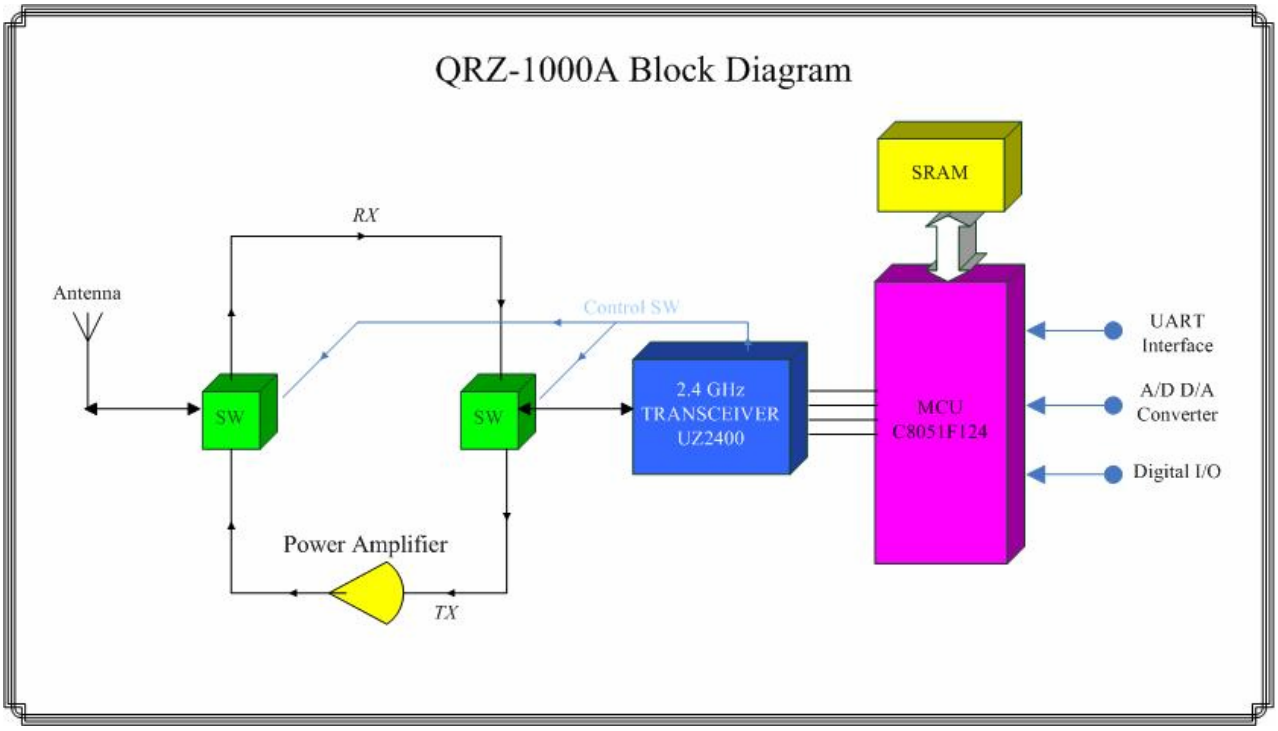
The QRZ-1000A includes an on-board chip antenna, power amplifier. The power amplifier enhances the transmission power. It improves range. Development Kits are available.

Models

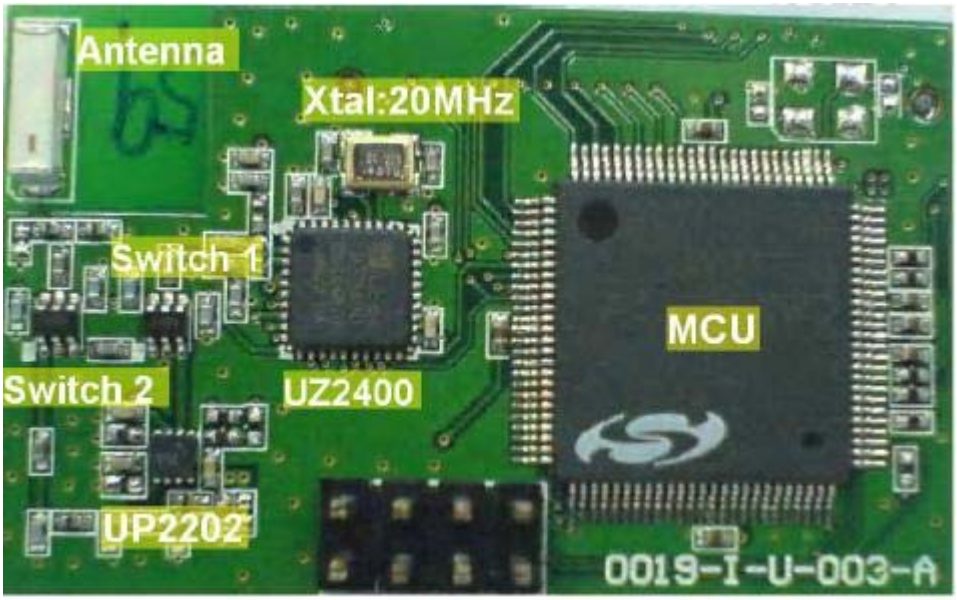
- QRZ-1000A: Includes on-board chip antenna and power amplifier
- QRZ-2200K: 2 nodes Development Kit
- QRZ-2400K: 4 nodes Development Kit

Features

- 44 x 28 mm^2 PCBA package with 2 connectors
- Utilizes globally available 2.4 GHz ISM band
- Control and Configuration compliant with ZigBee API commands.
- 65535 unique node addresses, IDs allow multiple large networks to coexist.
- Programmable Transmit Power Output
- Complete ZigBee and IEEE 802.15.4 spec compliant
- Typical Receiver Sensitivity –95 dBm
- Typical Throughput rate 250,000 bps
- Open air signal range to 500 meters at 15 dBm
- Multiple Low Power Operating modes

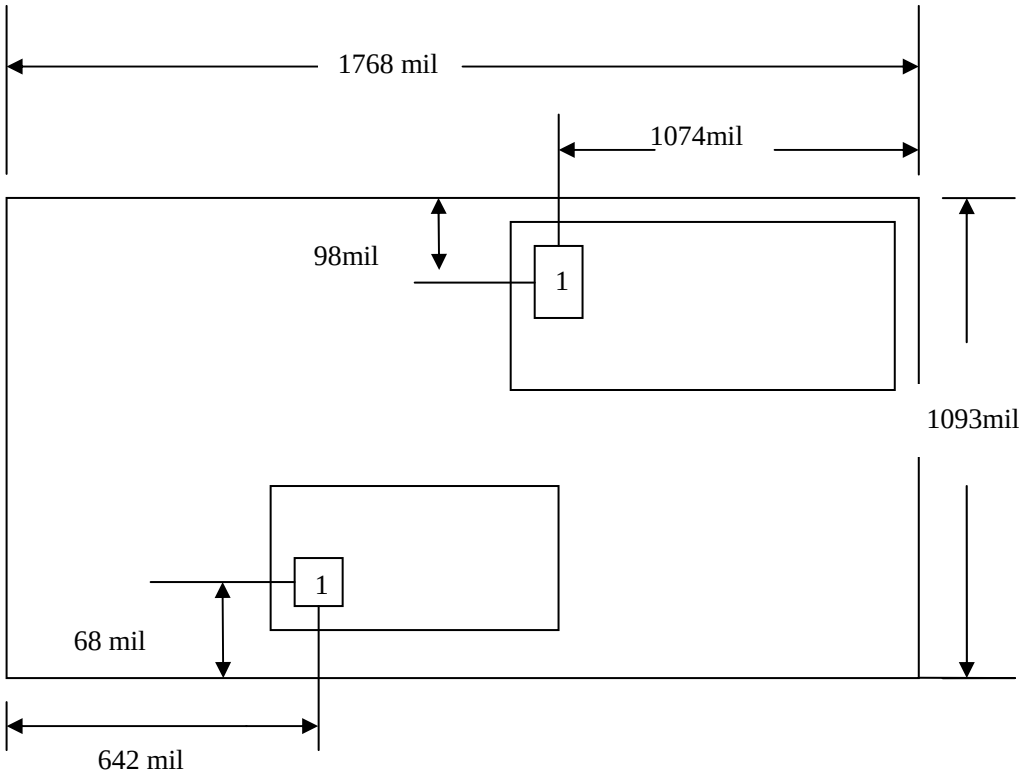


QRZ-1000A PCBA DRAWING



QRZ-1000A MECHANICAL SPECIFICATIONS

Note: The thickness of QRZ-1000A module is 3.64 mm.



Using the QRZ-1000A Power Saving Modes

The QRZ-1000A includes several low power operating modes to permit the most efficient use of the available power. Below are descriptions of the available selections.

ACTIVE: In Active Mode, all QRZ-1000A circuits are powered and available for immediate action. This includes the RF receiver which actively monitors the air for an incoming communications request. Two sub-modes are classified as TX-ACTIVE and RX-ACTIVE. The current consumption of TX-ACTIVE is 22 mA while RX-ACTIVE is 18 mA.

IDLE: In Idle mode, all QRZ-1000A RF circuits are shut down but the communications controller remains active to accept external commands. The QRZ-1000A cannot respond to incoming RF communications requests in Power-Down mode. If a transmit RF or receive RF command is received, The QRZ-1000A can activate the RF section in under 200 microseconds. Current draw in Idle Mode is less than 7.5 milliamps.

STANDBY: In Standby mode, all QRZ-1000A RF, MAC and Base Band circuits are powered-down with sleep clock remains active to minimize power consumption. The QRZ-1000A cannot accept commands or respond to incoming RF communications requests from Standby mode. Any input on TXD awakens the transceiver. Current draw in Standby Mode is less than 3.5 uA.

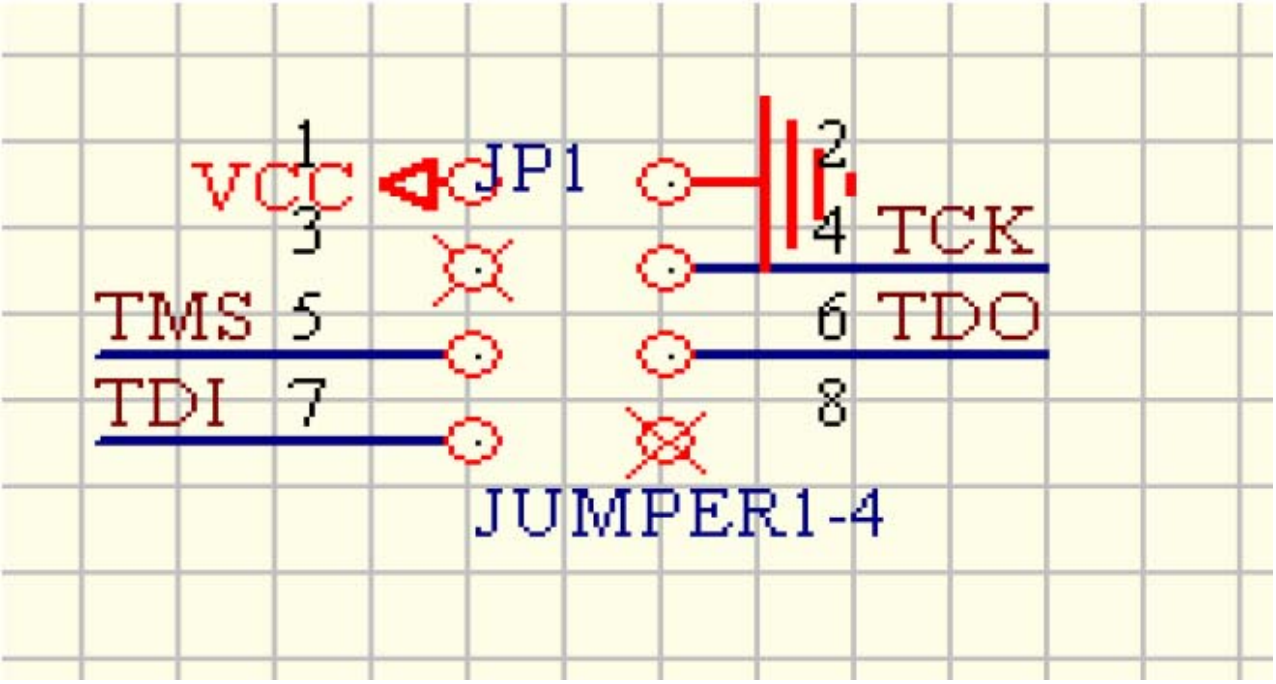
DEEP SLEEP: In Deep Sleep Mode, all QRZ-1000A circuits are shutdown except digital-side power. Current draw in Deep Sleep Mode is only 2 uA.

WAKE-UP: QRZ-1000A has two wake-up modes: timed wake-up and immediate wake-up. Timed wake-up has two sub-modes: beacon and non-beacon modes. However, if programmer sets both “main counter” and “remain counter” zero, QRZ-1000A will not wake up without immediate wake-up. QRZ-1000A can be wakened up externally. It can also be wakened up by setting internal register.

QRZ-1000A CONNECTORS PIN CONFIGURATION

QRZ-1000A uses SiliconLabs 82C51F124 as MCU. It is an 8051 base MCU. C8051F124 is a fully integrated mixed-signal System-on-Chip MCU with 64 digital IO pins. On-board JTAG debug circuitry allows non-intrusive, full-speed, in-circuit debugging using the production MCU installed in the final application. C8051F124, which can be programmed through 4-wired bus connector. The connector pin list is as below:

Connector JP1 Pin Configuration

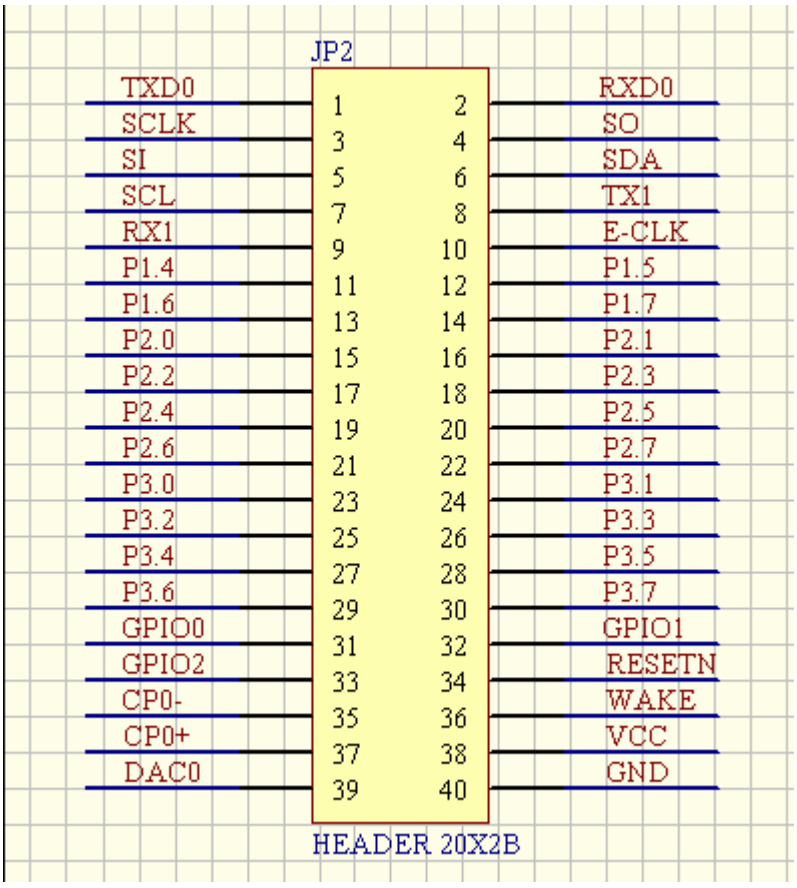


Connector JP1 Pin Description

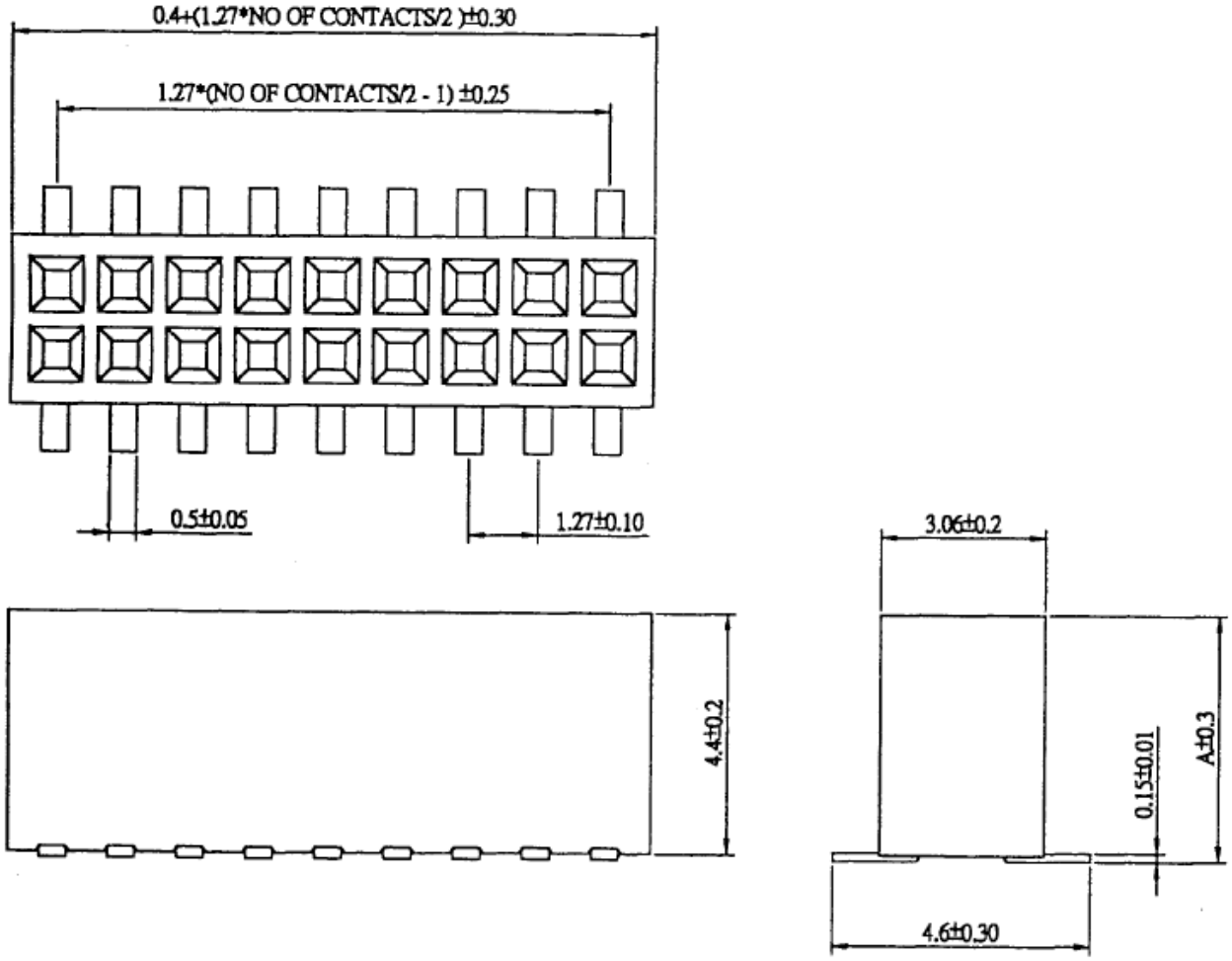
Signal	Pin	Description
VCC	1	3~3.3 Volt power for the QRZ-1000A
GND	2	Common voltage reference for the QRZ-1000A
NC	3	This pin is not connected. Reserve for future use.
TCK	4	Used as JTAG Clock
TMS	5	Used as JTAG TMS
TDO	6	Used as JTAG TDO, data output pin
TDI	7	Used as JTAG TDI, data input pin
NC	8	This pin is not connected. Reserve for future use.

QRZ-1000A can interface other devices such as sensor, LED, host controller, push button, joystick or power relays through a 20x2 connector. This connector interface can be configured to different functions.

Connector JP2 Pin Configuration



Connector JP2 Mechanical Drawing



Connector JP2 Pin Description

Pin	Signal	Function #1	Function #2	Function #3
1	TX0	P0.0	UART0:TX	
2	RX0	P0.1	UART0:RX	
3	SCK	P0.2	SPI:SCK	
4	MISO	P0.3	SPI:MISO	
5	MOSI	P0.4	SPI:MOSI	
6	SDA	P0.6	I2C:SDA	
7	SCL	P0.7	I2C:SCL	
8	TX1	P1.0	UART1:TX	
9	RX1	P1.1	UART1:RX	
10	CLK	P1.2	ECI: external clock	
11	GIO0	P1.4	GPIO0	T1/PWM1

12	GIO1	P1.5	GPIO1	INT1
13	GIO2	P1.6	GPIO2	T2/PWM2
14	GIO3	P1.7	GPIO3	T2EX
15	GIO4	P2.0	GPIO4	T4
16	GIO5	P2.1	GPIO5	T4EX
17	GIO6	P2.2	GPIO6	SYSCLK
18	GIO7	P2.3	GPIO7	CNVSTR0
19	GIO8	P2.4	GPIO8	CNVSTR2
20	G109	P2.5	GPIO9	
21	GIO10	P2.6	GPIO10	
22	GIO11	P2.7	GPIO11	
23	GIO12	P3.0	GPIO12	
24	GIO13	P3.1	GPIO13	
25	GIO14	P3.2	GPIO14	
26	GIO15	P3.3	GPIO15	
27	GIO16	P3.4	GPIO16	
28	GIO17	P3.5	GPIO17	
29	GIO18	P3.6	GPIO18	
30	GIO19	P3.7	GPIO19	
31	GIO20	GPIO0	GPIO20	
32	GIO21	GPIO1	GPIO21	
33	GIO22	GPIO2	GPIO22	
34	RESETN	RESETN		
35	AIN00*	AIN0.0	CP0+	
36	WAKE	WAKE		
37	AIN01*	AIN0.1	CP0-	
38	VDD	VDD		
39	DAC0	DAC0		
40	GND	GND		

* Use 0-Ohm resister to select

** P0.5 connects to SEN, P1.3 connects to INT

*** For additional function, check SiliconLab C8051F124 data sheet for detail.

Absolute Maximum Rating	
VCC	3.6V

Storage temperature	-40°C to +120°C
Operating temperature Range	-30°C to +80°C

WARNING: Exceeding any of these ratings will void the warranty and may damage the device

QRZ-1000A ELECTRICAL SPECIFICATIONS

Parameters	Min	Typ	Max	Units
Supply Voltage for RF, analog and digital circuits	2.4		3.6	V
Supply Voltage for Digital I/O	2.4	3.3	3.6	V
Current Consumption (MCU needs extra30mA in TX and RX modes)				
ACTIVE TX Mode @ 15 dBm		77		mA
ACTIVE RX Mode		18		mA
IDLE Mode		7.5		mA
STANDBY Mode		3.5		uA
DEEP SLEEP Mode		2		uA
Output Power (dBm) (P(dBm)=10*log(P{mW}))	-23.75	15		dBm
Output Power (mW)	0.0065	31.6		mW
Wireless Receive Sensitivity		-95		dBm
Range thru no Physical Obstructions@15dBm		500		meter
Selectable Channels		16		channel
Frequency Band	2.400		2.4835	GHz
Antenna Output Impedance		50		Ohms

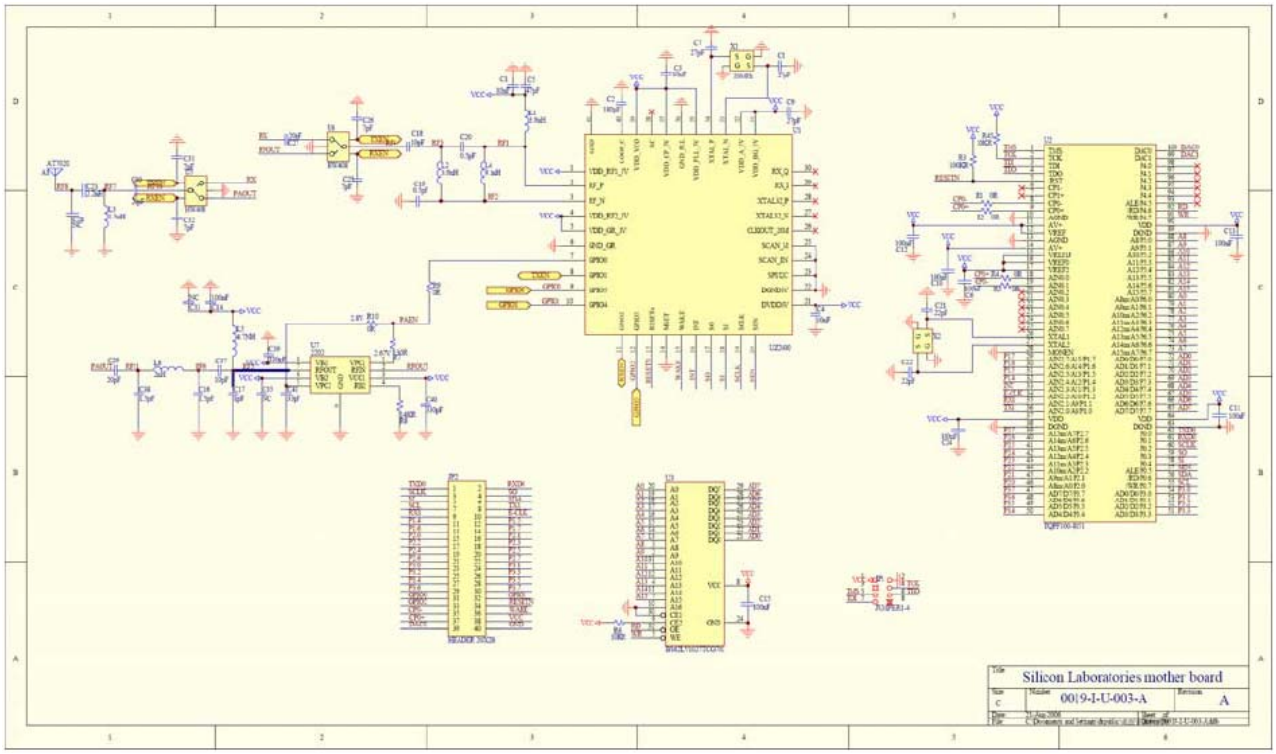
*** For output power setting, QRZ-1000A sets 15 dBm as default setting. To change setting, user can set RF transceiver LREG03 through SPI interface, the setting table shows as below. Check Ubec UZ-2400 data sheet for detail.

LREG03: RFCTRL3

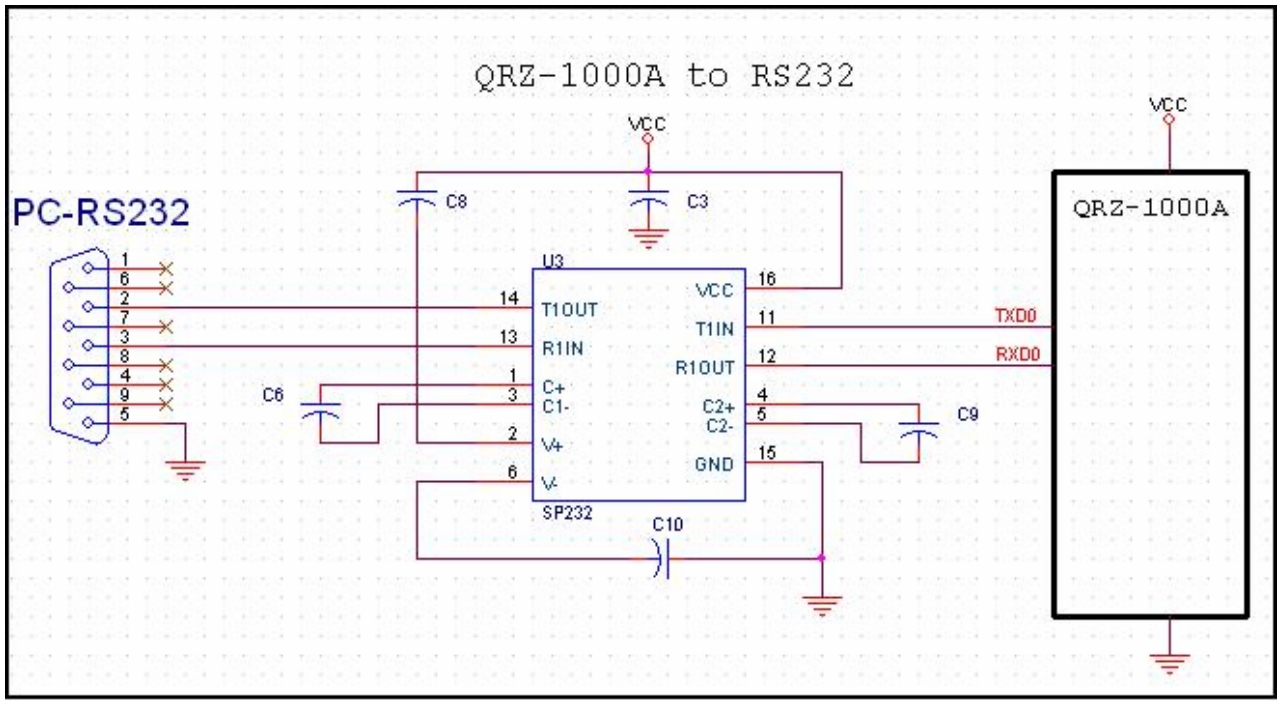
Offset: 0x03

Bits	Name	Description	Reset Value	R/W
7-6	**TXG_F**	Large scale control for Tx power in dB 00: 0; 01: -10; 10: –20; 11: -30	00	R/W
5-3	**TXG_B**	Fine scale control for Tx power in dB 000: 0; 001: -1.25; 010: -2.5; 011: -3.75; 100: -5; 101: -6.25; 110: -7.5; 111: -8.75	000	R/W
2-0	**Reserved**	Do NOT change the default value	000	----

QRZ-1000A CIRCUIT DIAGRAM



QRZ-1000A APPLICATION CIRCUIT DIAGRAM

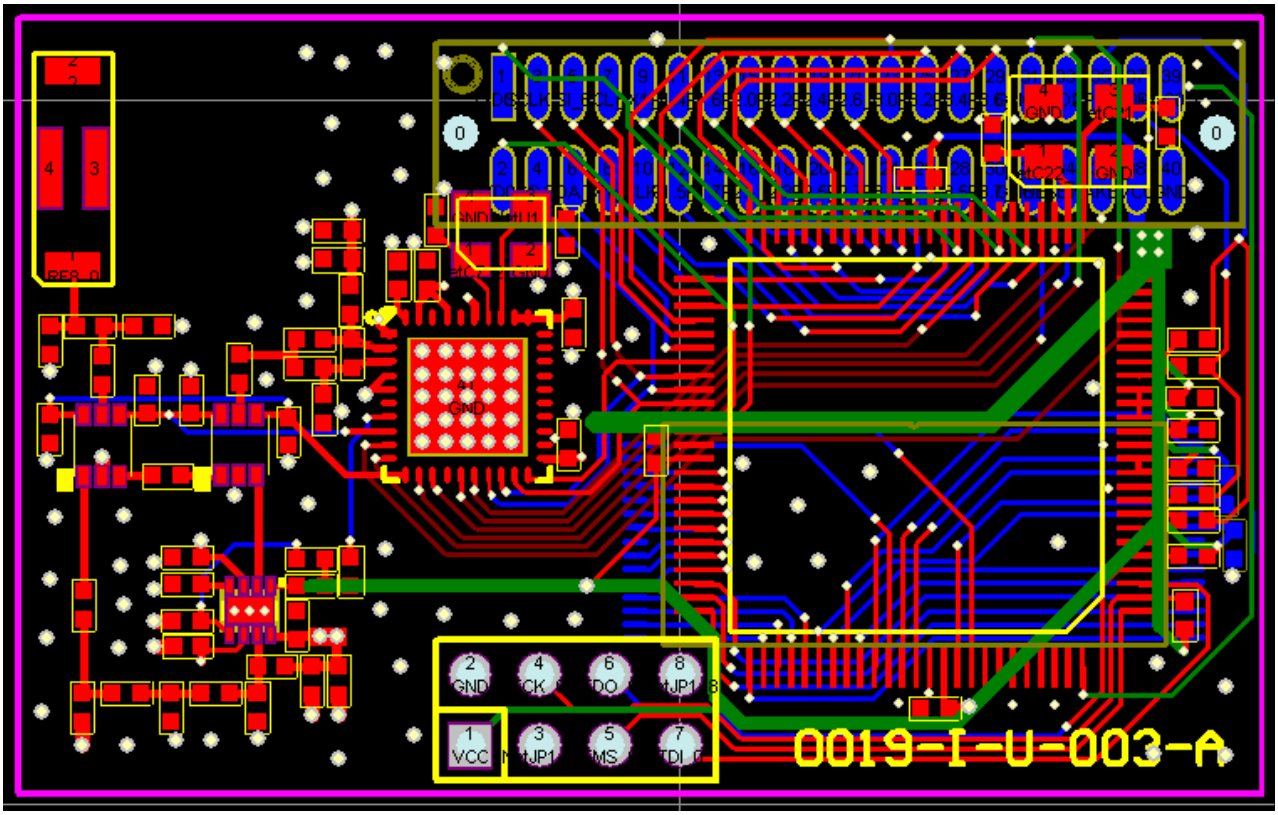


PART LIST OF QRZ-1000A CIRCUIT DIAGRAM				
Item	Quantity	Reference	Part	Description
1	1	A1	AT7020	LCC8
2	1	C4	100pF	0402
3	1	C7	NC	0402
4	1	C9	10pF	0402
5	2	C10,C12	NC	0402
6	3	C11,C18,C20	330pF	0402
7	1	C13	33pF	0402
8	1	C14	100nF	0402
9	2	C16,C15	1.5pF	0402
10	1	C17	1.8pF	0402
11	2	C21,C54	27pF	0402
12	4	C22,C24,C25,C26	7pF	0402
13	1	C23	3nH	0402
14	2	C27,C29	20pF	0402
15	2	C31,C28	470nF	0402
16	1	C30	20pF	0402
17	1	C32	10nF	0402
18	1	C33	3.3pF	0402
19	2	C37,C43	0.5pF	0402

20	1	C38	10pF	0402
21	4	C39,C52,C58,C63	10nF	0402
22	2	C57,C40	47pF	0402
23	1	C48	180pF	0402
24	1	C53	1uF	0402
25	2	C60,C64	33pF	0402
26	3	C65,C66,C67	1uF	0805
27	1	C68	10nF	0402
28	2	C69,C70	27pF	0402
29	1	D1	LED	0805
30	1	J1	CON5	WAFER-A2501WV-5P
31	1	J2	CON10A	HD5X2_2.54
32	1	L1	1.8nH	0402
33	1	L2	4.7nH	0402
34	2	L5,L3	5.1nH	0402
35	1	L4	9.1nH	0402
36	1	L6	2nH	0402
37	1	L7	33nH	0402
38	1	RFIN_g1	SMA	
39	2	R2,R1	180R	0402
40	1	R3	30R	0402
41	1	R4	10KR	0402
42	1	R5	10MR	0402
43	2	R6,R17	0R	0402
44	1	R7	160R	0402
45	1	R8	24KR	0402
46	1	R10	0R	0402
47	2	R15,R11	10KR	0402
48	2	R12,R13	1KR	0402
49	1	R14	220R	0402
50	1	R16	4.7KR	0402
51	4	R18,R19,R20,R21	560R	0402
52	1	TP1	SMA_ANTENNA	TP24S
53	1	U1	UZ2400	QFP40-0.5
54	1	U3	MPC82E54A	SOP20-7.5_1.27
55	1	U4	UP2202	2_2_8LEAD

56	2	U6,U5	HWS408	AS179
57	1	U7	UA2723	SOT363
58	1	U8	L23SV	SOT23-5
59	1	U9	RT9193-30PB	SOT-23-5
60	1	U10	25AA040_SOP-23	SOT-23-6
61	1	X1	20MHz	CX_101F
62	1	X2	32768Hz	OSC2
63	1	X3	12MHz	XTAL-5X3.2-4P_S

QRZ-3000 PCB LAYOUT



QRZ-1000A API REFERENCE

1. System Initialization

```
void system_init(System_Initial_Parameter_t Params)
```

This subroutine will initialize all system parameters. System characteristic can also be set with this subroutine by assigning values to the parameter. The type definition of

struct System_Initial_Parameter_t:

```
typedef struct System_Initial_Parameters_t {
    byte_t DeviceType;
    word_t PANID;
    word_t ChannelList;
    address_t ExtendedAddress;
} System_Initial_Parameters_t;
```

2. Automatic Network Configuration

```
void Network_Init()
```

Network_Init() is a set of startup procedures for standard ZigBee network. A proper procedure will be applied according to the characteristic of the device.

1. Coordinator: Network Formation → Permit Joining
2. Router: Network Discovery → Join → Start-Router → Permit Joining
3. End Device: Network Discovery → Join

3. Register Services for Profile

In a ZigBee profile, services are registered as endpoints. If you want to write a new profile, you may register new services to endpoints by calling ***add_SimpleDescriptor()***.

```
void add_SimpleDescriptor(
    uint8_t Endpoint,
    uint16_t ProfileId,
    int8_t NumInputClusters,
    byte_t *InputClusters,
    int8_t NumOutputClusters,
    byte_t *OutputClusters
)
```

4. Send MSG Frames

Subroutine ***Send_MSG_Frame()*** will send one MSG frame or a bunch of MSG frames to the specified destination.

```
void Send_MSG_Frame (
    int8_t DstAddrMode,
    address_t* DstAddress,
    uint8_t DstEndpoint,
    uint16_t ProfileId,
    uint8_t ClusterId,
    uint8_t SrcEndpoint,
    uint8_t MSGLength,
    byte_t* MSGFrame,
    byte_t TxOptions,
    int8_t DiscoverRoute,
    word_t RadiusCounter
)
```

5. Receive MSG Frames

In order to receive MSG Command, callback functions must be registered first by calling ***Register_MSG_Handler()***.

```
void Register_MSG_Handler (
    uint8_t Endpoint,
    void* MSGHandler
)
```

The format of callback function is:

```
void function_name(uint8_t, int8_t, byte_t *)
```

The first parameter is cluster identifier, the second parameter is transaction count, and the third parameter is pointer to the frame payload.

6. Assign buffer for transactions

```
void NewMSGOutputTransactions(byte_t *Transactions_p)
```

The first step for setting up transaction is to assign enough buffer space. This subroutine will assign buffer space allocated by the user application to the transaction set for further operation.

7. Add frame to transaction buffer

```
void AddMSGOutputTransactions(MSG_Frame_t * MSGFrame)
```

This subroutine will add a MSG frame into the transaction set. You may set multiple MSG frames to one transaction set, but you must be aware of the buffer size limitation.

8. Acquire transaction handler

```
byte_t *GetMSGOutputTransactions()
```

This function will return the handler of the transaction set, which would be assigned to the ***MSGFrame*** parameter of ***Send_MSG_Frame()*** subroutine.

9. Acquire transaction length

```
int8_t GetMSGOutputTransactionsLength();
```

This function will return the length of transaction set, which would be assigned to the ***MSGLength*** parameter of ***Send_MSG_Frame()*** subroutine.

10. Send KVP Frame

Send_KVP_Frame() will send one KVP frame or a bunch of KVP frames to the specified destination.

```
void Send_KVP_Frame (
    int8_t DstAddrMode,
    address_t* DstAddress,
    uint8_t DstEndpoint,
    uint16_t ProfileId,
    uint8_t ClusterId,
    uint8_t SrcEndpoint,
    uint8_t KVPLength,
    byte_t* KVPFrame,
    byte_t TxOptions,
    int8_t DiscoverRoute,
    word_t RadiusCounter
)
```

11. Receive KVP Commands

In order to receive KVP Command, callback functions must be registered first by calling ***Register_KVP_Handler()***. The registered handler will be called when receiving KVP frames.

```
void Registe_KVP_Handler (
    uint8_t endpoint,
    void* KVPHandler
)
```

The format of callback function is:

```
void function_name(uint8_t, int8_t, byte_t *)
```

The first parameter is cluster identifier, the second parameter is transaction count, and the third parameter is pointer to the frame payload.

12. Assign buffer for transactions

```
void NewKVPOutputTransactions(byte_t *Transactions_p)
```

The first step for setting up transaction is to assign enough buffer space. This subroutine will

assign buffer space allocated by the user application to the transaction set for further operation.

13. Add frame to transaction buffer

```
void AddKVPOutputTransactions(MSG_Frame_t * MSGFrame)
```

This subroutine will add a KVP frame into the transaction set. You add multiple KVP frames to one transaction set, but you must be aware of the buffer size limitation.

14. Acquire transaction handler

```
byte_t *GetKVPOutputTransactions()
```

This function will return the handler of the transaction set, which would be assigned to the ***KVPFrame*** parameter of ***Send_KVP_Frame()*** subroutine.

15. Acquire transaction length

```
int8_t GetKVPOutputTransactionsLength()
```

This function will return the length of transaction set, which would be assigned to the ***KVPLength*** parameter of ***Send_KVP_Frame()*** subroutine.

16. Retrieve Network-Address

16.1. Send Request

```
void send_NWK_addr_req(NWK_addr_req_t *msg)
```

Parameter definition:

```
typedef struct NWK_addr_req_t {
    address_t IEEEAddr;
    int8_t RequestType;
    int8_t StartIndex;
} NWK_addr_req_t;
```

16.2. Set Response

```
void send_NWK_addr_rsp(NWK_addr_rsp_t *msg)
```

Parameter definition:

```
typedef struct NWK_addr_rsp_t {
    zdp_status_t Status;
    address_t IEEEAddrRemoteDev;
    int8_t NWKAddrRemoteDev;
    qword_t NumAssocDev;
    int8_t StartIndex;
    int16_t NWKAddrAssocDevList[MAX_DEVICE_ADDRESS_LIST_SIZE]
} NWK_addr_rsp_t;
```

17. Retrieve IEEE-Address

17.1. Send Request

```
void send_IEEE_addr_req(IEEE_addr_req_t *msg)
```

Parameter definition:

```
typedef struct IEEE_addr_req_t {
    word_t NWKAddrOfInterest;
    int8_t RequestType;
    int8_t StartIndex;
} IEEE_addr_req_t;
```

17.2. Send Response

```
void send_IEEE_addr_rsp(IEEE_addr_rsp_t *msg)
```

Parameter definition:

```
typedef struct IEEE_addr_rsp_t {
    zdp_status_t Status;
    address_t IEEEAddrRemoteDev;
    word_t NWKAddrRemoteDev;
    int8_t NumAssocDev;
    int8_t StartIndex;
    int16_t NWKAddrAssocDevList[MAX_DEVICE_ADDRESS_LIST_SIZE];
} IEEE_addr_rsp_t;
```

18. Retrieve Node-Descriptor

18.1. Send Request

```
void send_Node_Desc_req(Node_Desc_req_t *msg);
```

Parameter definition:

```
typedef struct Node_Desc_req_t {
    word_t NWKAddrOfInterest;
} Node_Desc_req_t;
```

18.2. Send Response

```
void send_Node_Desc_rsp(Node_Desc_rsp_t *msg);
```

Parameter definition:

```
typedef struct Node_Desc_rsp_t {
    zdp_status_t Status;
    word_t NWKAddrOfInterest;
    NodeDescriptor_t NodeDescriptor;
} Node_Desc_rsp_t;
```

19. Retrieve Power-Descriptor

19.1. Send Request

```
void send_Power_Desc_req(Power_Desc_req_t *msg)
```

Parameter definition:

```
typedef struct Power_Desc_req_t {
    word_t NWKAddrOfInterest;
} Power_Desc_req_t;
```

19.2. Send Response

```
void send_Power_Desc_rsp(Power_Desc_rsp_t *msg)
```

Parameter definition:

```
typedef struct Power_Desc_rsp_t {
    zdp_status_t Status;
    word_t NWKAddrOfInterest;
    PowerDescriptor_t PowerDescriptor;
} Power_Desc_rsp_t;
```

20. Retrieve Simple-Descriptor

20.1. Send Request

```
void send_Simple_Desc_req(Simple_Desc_req_t *msg)
```

Parameter definition :

```
typedef struct Simple_Desc_req_t {
    word_t NWKAddrOfInterest;
    uint8_t endpoint;
} Simple_Desc_req_t;
```

20.2. Send Response

```
void send_Simple_Desc_rsp(Simple_Desc_rsp_t *msg)
```

Parameter definition:

```
typedef struct Simple_Desc_rsp_t {
    zdp_status_t Status;
    word_t NWKAddrOfInterest;
    int8_t Length;
    SimpleDescriptor_t SimpleDescriptor;
} Simple_Desc_rsp_t;
```

21. Retrieve Active-Endpoint list

21.1. Send Request

```
void send_Actvie_EP_req(Active_EP_req_t *msg)
```

Parameter definition:

```
typedef struct Active_EP_req_t {
    word_t NWKAddrOfInterest;
} Active_EP_req_t;
```

21.2. Send Response

```
void send_Actvie_EP_rsp(Active_EP_rsp_t *msg)
```

Parameter definition:

```
typedef struct Active_EP_rsp_t {
    zdp_status_t Status;
    word_t NWKAddrOfInterest;
    int8_t ActiveEPCount;
    byte_t ActiveEPList[MAX_SIMPLE_DESCRIPTORS_SIZE];
} Active_EP_rsp_t;
```

22. Retrieve Match-Descriptor

22.1. Send Request

```
void send_Match_Desc_req(Match_Desc_req_t *msg)
```

Parameter definition:

```
typedef struct Match_Desc_req_t {
    word_t NWKAddrOfInterest;
    uint16_t ProfileID;
    byte_t Clusters[34];
} Match_Desc_req_t;
```

22.2. Send Response

```
void send_Match_Desc_rsp(Match_Desc_rsp_t *msg)
```

Parameter definition:

```
typedef struct Match_Desc_rsp_t {
    zdp_status_t Status;
    word_t NWKAddrOfInterest;
    int8_t MatchLength;
    byte_t MatchList[MAX_MATCH_LIST];
} Match_Desc_rsp_t;
```

23. Retrieve User-Descriptor

23.1. Send Request

```
void send_User_Desc_req(User_Desc_req_t *msg)
```

Parameter definition:

```
typedef struct User_Desc_req_t {
    word_t NWKAddrOfInterest;
} User_Desc_req_t;
```

23.2. Send Response

```
void send_User_Desc_rsp(User_Desc_rsp_t *msg)
```

Parameter definition:

```
typedef struct User_Desc_rsp_t {
    zdp_status_t Status;
    word_t NWKAddrOfInterest;
    int8_t Length;
    UserDescriptor_t UserDescriptor;
} User_Desc_rsp_t;
```

24. Discovery registration

24.1. Send Request

```
void send_Discovery_Register_req(Discovery_Register_req_t *msg)
```

Parameter definition:

```
typedef struct Discovery_Register_req_t {
    word_t NWKAddr;
    address_t IEEEAddr;
} Discovery_Register_req_t;
```

24.2. Send Response

```
void send_Discovery_Register_rsp(Discovery_Register_rsp_t *msg)
```

Parameter definition:

```
typedef struct Discovery_Register_rsp_t {
    zdp_status_t Status;
} Discovery_Register_rsp_t;
```

25. End-device announcement

```
void send_End_Device_annce(End_Device_annce_t *msg)
```

Parameter definition:

```
typedef struct End_Device_annce_t {
    word_t NWKAddr;
    address_t IEEEAddr;
} End_Device_annce_t;
```

26. User-Descriptor configuration

26.1. Send Request

```
void send_User_Desc_set(User_Desc_set_t *msg)
```

Parameter definition:

```
typedef struct User_Desc_set_t {
    word_t NWKAddrOfInterest;
    byte_t UserDescription[UD_USER_DESCRIPTION_LENGTH];
} User_Desc_set_t;
```

26.2. Send Response

```
void send_User_Desc_conf(User_Desc_conf_t *msg)
```

Parameter definition:

```
typedef struct User_Desc_conf_t {
    zdp_status_t Status;
} User_Desc_conf_t;
```

27. End-Device binding

27.1. Send Request

```
send_End_Device_Bind_req(send_End_Device_Bind_req_t *msg)
```

Parameter definition:

```
typedef struct End_Device_Bind_req_t {
    word_t BindingTarget;
    uint8_t Endpoint;
    uint16_t ProfileID;
    byte_t Clusters[34];
} End_Device_Bind_req_t;
```

27.2. Send Response

```
void send_End_Device_Bind_rsp(send_End_Device_Bind_rsp_t *msg)
```

Parameter definition:

```
typedef struct End_Device_Bind_rsp_t {
    zdp_status_t Status;
} End_Device_Bind_rsp_t;
```

28. Binding process

28.1. Send Request

```
void send_Bind_req(Bind_req_t* msg)
```

Parameter definition:

```
typedef struct Bind_req_t {
    address_t SrcAddress;
    uint8_t SrcEndp;
    uint8_t ClusterID;
    byte_t DstAddress;
    uint8_t DstEndp;
} Bind_req_t;
```

28.2. Send Response

```
void send_Bind_rsp(Bind_rsp_t* msg)
```

Parameter definition:

```
typedef struct Bind_rsp_t {
    zdp_status_t Status;
} Bind_rsp_t;
```

29. Unbinding process

29.1. Send Request

```
void send_Unbind_req(Unbind_req_t* msg)
```

Parameter definition:

```
typedef struct Unbind_req_t {
    qword_t SrcAddress;
    uint8_t SrcEndp;
    uint8_t ClusterID;
    qword_t DstAddress;
    uint8_t DstEndp;
} Unbind_req_t;
```

29.2. Send Response

```
void send_Unbind_rsp(Unbind_rsp_t* msg)
```

Parameter definition:

```
typedef struct Unbind_rsp_t {
    zdp_status_t Status;
} Unbind_rsp_t;
```

30. Network Discovery

30.1. Send Request

```
void send_Mgmt_NWK_Disc_req(Mgmt_NWK_Disc_req_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_NWK_Disc_req_t {
    dword_t ScanChannel;
    int8_t ScanDuration;
    int8_t StartIndex;
} Mgmt_NWK_Disc_req_t;
```

31. Network Discovery

31.1. Send Request

```
void send_Mgmt_NWK_Disc_req(Mgmt_NWK_Disc_req_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_NWK_Disc_req_t {
    dword_t ScanChannel;
    int8_t ScanDuration;
    int8_t StartIndex;
} Mgmt_NWK_Disc_req_t;
```

31.2. Send Response

```
void send_Mgmt_NWK_Disc_rsp(Mgmt_NWK_Disc_rsp_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_NWK_Disc_rsp_t {
    zdp_status_t Status;
    int8_t NetworkCount;
    int8_t StartIndex;
    int8_t NetworkListCount;
    net_desc_info NetworkList[3];
} Mgmt_NWK_Disc_rsp_t;
```

32. Retrieve LQI value

32.1. Send Request

```
void send_Mgmt_Lqi_req(Mgmt_Lqi_req_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Lqi_req_t {
    int8_t StartIndex;
} Mgmt_Lqi_req_t;
```

32.2. Send Response

```
void send_Mgmt_Lqi_rsp(Mgmt_Lqi_rsq_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Lqi_rsp_t {
    zsp_status_t Status;
    int8_t NeighborTableEntries;
    int8_t StartIndex;
    int8_t NeighborTableListCount;
    NeighborTableList_t NeighborTableList[2];
} Mgmt_Lqi_rsp_t;
```

33. Retrieve Routing Table

33.1. Send Request

```
void send_Mgmt_Rtg_req(Mgmt_Rtg_req_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Rtg_req_t {
    int8_t StartIndex;
} Mgmt_Rtg_req_t;
```

33.2. Send Response

```
void send_Mgmt_Rtg_rsp(Mgmt_Rtg_rsp_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Rtg_rsp_t {
    zdp_status_t Status;
    int8_t RoutingTableEntries;
    int8_t StartIndex;
    int8_t RoutingTableListCount;
    RoutingTableList_t RoutingTableList[8];
}Mgmt_Rtg_rsp_t;
```

34. Retrieve Binding Table

34.1. Send Request

```
void send_Mgmt_Bind_req(Mgmt_Bind_req_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Bind_req_t {
    int8_t StartIndex;
}Mgmt_Bind_req_t;
```

34.2. Send Response

```
void send_Mgmt_Bind_rsp(Mgmt_Bind_rsp_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Bind_rsp_t {
    zdp_status_t Status;
    int8_t BindingTableEntries;
    int8_t StartIndex;
    int8_t BindingTableListCount;
    BindingTableList_t BindingTableList[3];
}BindingTableList_t;
```

35. Leave process

35.1. Send Request

```
void send_Mgmt_Leave_req(Mgmt_Leave_req_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Leave_req_t {
    address_t DeviceAddress;
} Mgmt_Leave_req_t;
```

35.2. Send Response

```
void send_Mgmt_Leave_rsp(Mgmt_Leave_rsp_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Leave_rsp_t {
    zdp_status_t Status;
} Mgmt_Leave_rsp_t;
```

36. Direct-Join process

36.1. Send Request

```
void send_Mgmt_Direct_Join_req(Mgmt_Direct_Join_req_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Direct_Join_req_t {
    address_t DeviceAddress;
    byte_t CapabilityInformation;
} Mgmt_Direct_Join_req_t;
```

36.2. Send Response

```
void send_Mgmt_Direct_Join_rsp(Mgmt_Direct_Join_rsp_t* msg)
```

Parameter definition:

```
typedef struct Mgmt_Direct_Join_rsp_t {
    zdp_status_t zdp_status_t Status;
} Mgmt_Direct_Join_rsp_t;
```

37. Network Layer NLME Commands

37.1 NLME-NETWORK-DISCOVERY.request

```
void nlme_network_discovery_request (
    dword_t ScanChannels,
    int8_t ScanDuration
)
```

37.2. NLME-NETWORK-FORMATION.request

```
void nlme_network_formation_request (
    dword_t ScanChannels,
    int8_t ScanDuartion,
    int8_t BeaconOrder,
    int8_t SuperframeOrder,
    int8_t PANId,
    int8_t BatteryLifeExtension
)
```

37.3. NLME-PERMIT-JOINING.request

```
void nlme_permit_joining_request (
    byte_t PermitDuration
)
```

37.4. NLME-START-ROUTER.request

```
void nlme_start_router_request (
    int8_t BeaconOrder,
    int8_t SuperframeOrder,
    bool_t BatteryLifeExtension
)
```

37.5. NLME-JOIN.request

```
void nlme_join_request (
    word_t PANId,
    bool_t JoinAsRouter,
    bool_t RejoinNetwork,
    dword_t ScanChannels,
    int8_t ScanDuration,
    int8_t PowerSource,
    int8_t RxOnWhenIdle,
    int8_t MACSecurity
)
```

37.6. NLME-DIRECT-JOIN.request

```
Void nlme_direct_join_request (
    address_t* DeviceAddress,
    byte_t CapabilityInformation
)
```

37.7. NLME-LEAVE.request

```
void nlme_leave_request (
    address_t* DeviceAddress,
    bool_t RemoveChildren,
    bool_t MACSecurityEnable
)
```

37.8. NLME-RESET.request

```
void nlme_reset_request ( void )
```

37.9. NLME-SYNC.request

```
void nlme_sync_request (
    bool_t Track
)
```

38. NLME Confirms

The NLME confirm events will be passed to the user application through callback functions. Corresponding handlers must be registered by calling ***`Register_NWK_Handler()`***.

```
void Register_NWK_Handler(
    void* NWKHandler
)
```

38.1. NLME Confirm ID

NLME_NETWORK_DISCOVERY_CONFIRM
NLME_NETWORK_FORMATION_CONFIRM
NLME_START_ROUTER_CONFIRM
NLME_PERMIT_JOINING_CONFIRM
NLME_JOIN_CONFIRM
NLME_JOIN_INDICATION
NLME_DIRECT_JOIN_CONFIRM
NLME_LEAVE_CONFIRM
NLME_LEAVE_INDICATION

NLME_RESET_CONFIRM
NLME_SYNC_CONFIRM
NLME_SYNC_INDICATION
NETWORK_FINISH

38.2. NLME-NETWORK-DISCOVERY.confirm

```
typedef struct nlme_network_discovery_confirm_t {
    byte_t NetworkCount;
    net_desc_info NetworkDescriptor[NEIGHBORTABLENUM];
    mac_status_t status;
} nlme_network_discovery_confirm_t;
```

38.3. NLME-NETWORK-FORMATION.confirm

```
typedef struct nlme_network_formation_confirm_t {
    uint8_t status;
} nlme_network_formation_confirm_t;
```

38.4. NLME-START-ROUTER.confirm

```
typedef struct nlme_start_router_confirm_t {
    mac_status_t status;
}nlme_start_router_confirm_t;
```

38.5. NLME-PERMIT-JOINING.confirm

```
typedef struct nlme_permit_joining_confirm_t {
    mac_status_t status;
} nlme_permit_joining_confirm_t;
```

38.6. NLME-JOIN.confirm

```
typedef struct nlme_join_confirm_t {
    word_t PANId;
    mac_status_t status;
} nlme_join_confirm_t;
```

38.7. NLME-JOIN.indication

```
typedef struct nlme_join_indication_t {
    word_t ShortAddress;
    address_t ExtendedAddress;
    byte_t CapabilityInformation;
    bool_t secureJoin;
} nlme_join_indication_t;
```

38.8. NLME-DIRECT-JOIN.confirm

```
typedef struct nlme_direct_join_confirm_t {
    address_t DeviceAddress;
    mac_status_t status;
} nlme_direct_join_confirm_t;
```

38.9. NLME-LEAVE.confirm

```
typedef struct nlme_leave_confirm_t {
    qword_t DeviceAddress;
    mac_status_t status;
} nlme_leave_confirm_t;
```

38.10. NLME-LEAVE.indication

```
typedef struct nlme_leave_indication_t {
    address_t DeviceAddress;
} nlme_leave_indication_t;
```

38.11. NLME-RESET.confirm

```
typedef struct nlme_reset_confirm_t {
    mac_status_t status;
} nlme_reset_confirm_t;
```

38.12. NLME-SYNC.confirm

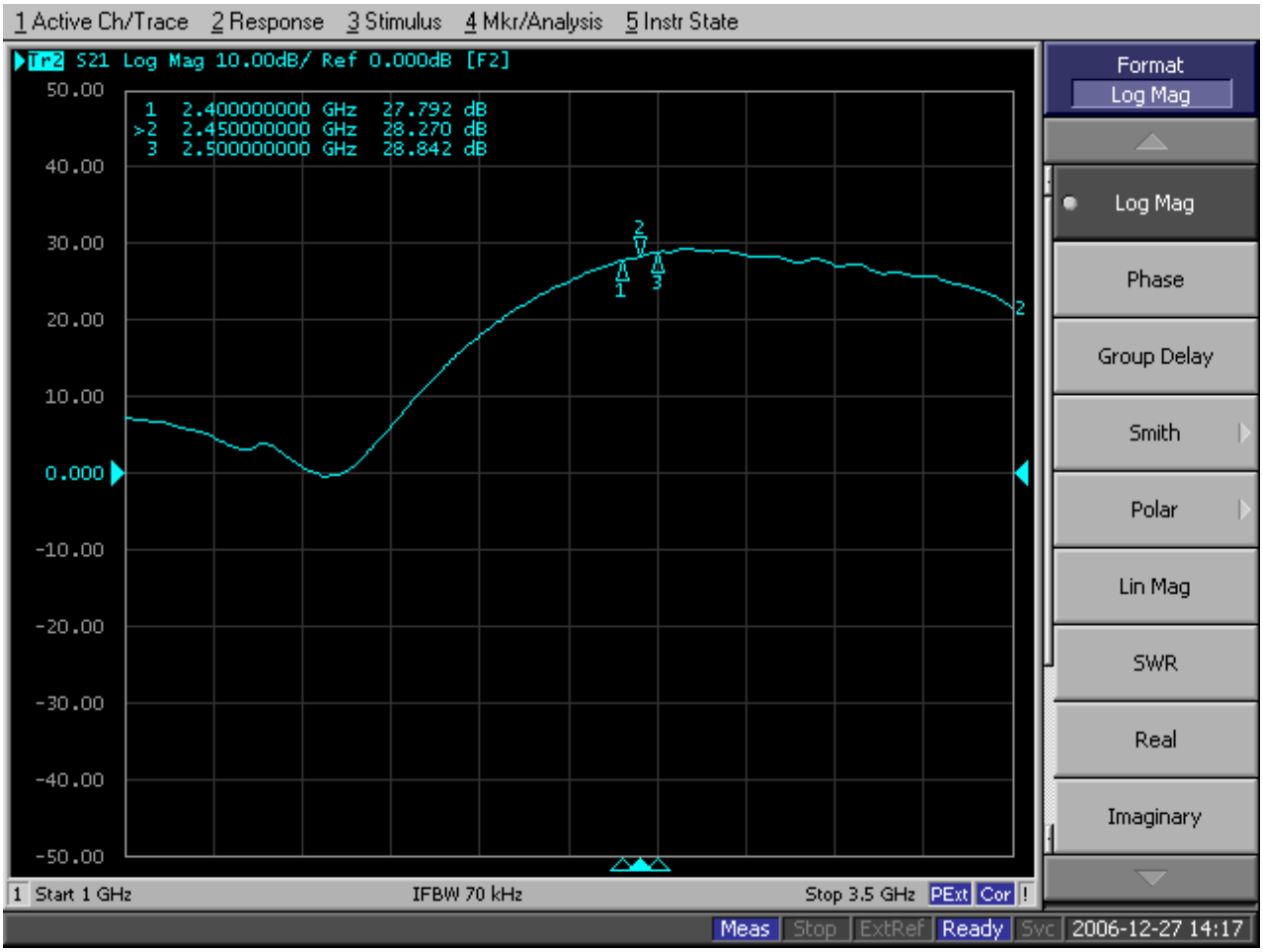
```
typedef struct nlme_sync_confirm_t {
    nwk_status_t status;
} nlme_sync_confirm_t;
```

RF PERFORMANCE

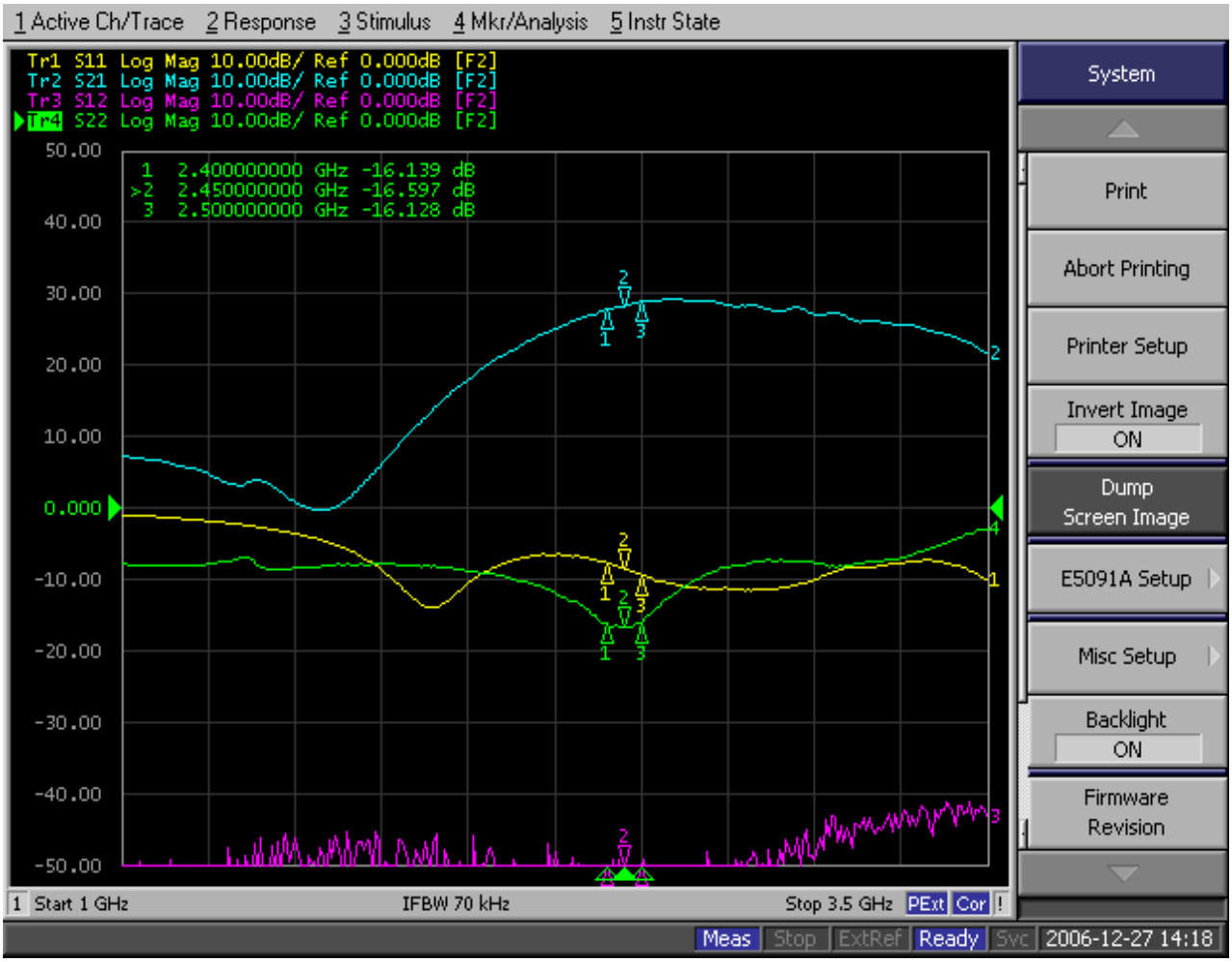
For an RF power amplifier, the performance degrades when output power reaches the non-linearity region. The maximum linear power of PA depends on P1dB point and signal complexity such as modulation, data length and data rate. For UP2202, linear gain is around 27dB and P1dB output power is 26 dBm. To optimize the maximum output power, the output power of UZ2400 (that is, the input power of UP2202) should be adjusted for various system needs.

RF PERFORMANCE TEST DATA

Maximum PA output power 28.79dB



Return Loss: -16.5dB



QuadRep Electronics [T] Ltd.

5F-13, No. 79, Hsin Tai Wu Rd, Sec.1, His-Chih, Taipei, Taiwan

TEL: +886-2-26989933

FAX: +886-2-26989911

http:// www.quadrep.com.tw

http:// www.quadrep.com.cn