# AT76C902
# RFtool Description

| Version | Date | Author | Changes |
|---|---|---|---|
| 1.0 | 14 July 2005 | fromesis@patras.atmel.com | Original version |
| 1.1 | 12 Aug 2005 | fromesis@patras.atmel.com<br>efilippatos@patras.atmel.com | 1. Examples on bbc, rfc commands added<br>2. Test parameters reordered<br>3. Zero command explained<br>4. Not supported commands updated<br>5. New FER commands explained<br>6. CRXFG statistics |
| 1.2 | 31 Aug 2005 | fromesis@patras.atmel.com | Statistics loop added |

## Table of Contents

# Section 1

# Overview

The RFtool test command is an application that sets the device to testing mode for Radio Frequency Tests (RF Tests). The following tests can be executed:

- Continuous Transmission (not implemented for DSSS but only for OFDM)
- Continuous Packet Transmission
- Carrier Frequency Accuracy
- Carrier Frequency Suppression
- Continuous Reception
- Continuous Reception with Fixed Gain

You can also set the test parameters, the values of the baseband registers, and the values of the radio registers. Statistics for the tests can also be viewed, including the following fields:

- packets transmitted
- packets received
- transmission failures
- transmission corruptions
- transmission FIFO underflows
- FCS errors
- FER

# Section 2

# Commands for the Serial Console

In order to set the device into Test Mode you should type:

`ifconfig eth0 hw ether xx:xx:xx:xx:xx:xx up` : the characters xx is the Mac address of the Wlan interface.

Now a new Wlan Interface has been created.

The commands you can use for the serial console are:

- **rftool**

  **Description**

  This command starts the application for the RF Test. It sets the state of the driver to Test Mode, initializes the test parameters with the default values and enters the test loop.

```
# rftool

Atmel Wlan interface closed
Driver Version : 1.0.0.18
Firmware Version : 1.0.3.25

        cptx            Continuous Packet TX
        ctx             Continuous TX
        cars            Carrier Suppresion
        cara            Carrier Accuracy
        crx             Continuous RX
        crxfg           Continuous RX Fixed Gain
        c               Change test params
        bbc             Change BB Reg <bbc 20 0x48>
        bbr             Read BB Reg <bbr 20>
        bball           Read All BB Reg
        bbs             Change BB Reg Configuration
        rfc             Change Radio register <rfc 20 0x48>
        stop            Stop Test
        s               Statistics
        z               Zero Statistics
        d               Restore Defaults
        h               Show This Message
        q               Quit

Cmd>_
```

**Performing Tests**

In the test loop, you can perform the following tests:

| | |
|---|---|
| **Continuous Packet TX** | (continuous packet transmission) |
| **Continuous TX** | (continuous packet transmission with small interval) |
| **Carrier Suppression** | (checks the transmitted carrier frequency suppression) |
| **Carrier Accuracy** | (checks the transmitted carrier frequency accuracy) |
| **Continuous RX** | (continuous reception) |
| **Continuous RX Fixed Gain** | (continuous reception with fixed gain) |

**Note:** Please stop a test (stop command), before proceeding to the next one.

**Test Parameters**

By using the c command you can change the test parameters, before proceeding to a test.
The test parameters (as defined in `testcmd.h` in `TEST_PARAMS` struct – included in APPENDIX A) are:

| | |
|---|---|
| **ShortPreamble** | (Long - 0 or Short - 1) |
| **Channel** | (transmission channel: 1 - 14) |
| **TxRate** | (transmission rate) |
| **TxPower** | (transmission power: 0x0-0x3f) |
| **Length** | (packet length - bytes) |
| **TxIframe** | (interframe time - µs) |
| **PacketsToTX** | (number of packets to transmit. By using '–1' infinite number of packets will be sent) |
| **Pattern** | (pattern contents: 0x0 - 0xff) |

Also added:

| | |
|---|---|
| **Receive Gain** | (used for CRX Fixed Gain Test. Values from 0x00 to 0x7f ) |
| **Expected Receive Packets** | (number of packets expected to receive when in CRX or CRXFG mode) |

**Parameters Default Initialization**

The default values for the parameters initialization are:

**Preamble** = Long (0)
**Channel** = 1
**TxRate** = 1
**TxPower** = 0x20
**Length** = 0x30
**TxIframe** = 1000
**PacketsToTX** = -1
**Pattern** = 0xaa
**Receive Gain** = 0x70
**Expected Receive Packets** = 1000

By using the d command, you can restore the default test parameters values.

**Baseband Registers**

You can read or change the values of the baseband registers, using the appropriate commands:

**bbc** (change baseband registers)
(i.e. bbc "register address" "register value")
(i.e. bbc 0x4000 0x110)
**bbr** (read baseband registers)
(i.e. bbr "register address")
(i.e. bbr 0x4000)
**bball** (read all baseband registers – not implemented)
**bbs** (change baseband registers configuration – not implemented)

**Radio Register**

You can change the value of the radio registers, using the command:

**rfc** (change radio registers)
(i.e. rfc "register address" "register value")
(i.e. rfc 10 0xdbba)

***Note:*** *Please stop a test (stop command), before changing a baseband/radio register value.*

**Statistics**

By using the s command, you can view statistics regarding the performing or completed tests. The following statistics are being shown, depending on the type of the test:

**In Idle Mode:**
TX
PacketsTx (number of packets transmitted)
Tx Fails (transmission failures)
Tx Corrupt (transmission corruptions)
Tx FIFO Underflow (transmission underflows)

RX
PacketsRx (number of packets received with correct CRC)
FCSError (number of packets received with wrong CRC)
FER

Expected Received Packets
Total FER

**In CPTX Test Mode:**
PacketsTx (number of packets transmitted)
Tx Fails (transmission failures)
Tx Corrupt (transmission corruptions)
Tx FIFO Underflow (transmission underflows)

**In CTX Test Mode:**

No statistics

**In CRX Test Mode:**

| | |
|---|---|
| PacketsRx | (number of packets received with correct CRC) |
| FCSError | (number of packets received with wrong CRC) |
| Differential FER | (FER between two consecutive statistic measurements) |
| Cumulative FER | (FER between the start of crx command and the last statistic measurement) |

**In CRXFG Test Mode:**

| | |
|---|---|
| PacketsRx | (number of packets received with correct CRC) |
| FCSError | (number of packets received with wrong CRC) |
| Differential FER | (FER between two consecutive statistic measurements) |
| Cumulative FER | (FER between the start of CRX command and the last statistic measurement) |

**Usage**:
Type s to enter the statistics loop.
Press ENTER to get the next statistics measurement.
Type z to reset the statistics measurements and exit the statistics loop.
Press q to exit the statistics loop.
You can also reset the statistics measurements by pressing z outside the statistics loop (z command).

# Section 3

# APPENDIX A

- **The structure of the Test Parameters**

```
typedef struct _TEST_PARAMS{
UINT8       Pattern;
UINT8       TxRate;
UINT16      TxIframe;
UINT32      Length;
UINT8       Channel;
UINT8       Antenna;
UINT8       Tx_Filter;
UINT8       TxPower;
UINT8       ShortPreamble;
UINT32      PacketsToTX;
UINT32      RegAddr;
UINT32      RegValue;
UINT32      StatPos;
UINT32      Reserved;
} TEST_PARAMS;
```

- **The structure used for Statistics**

We use some members of the STATISTICS_MIB structure to hold and get the statistics values:

```
typedef struct __STATISTICS_MIB{    // used for (statistics)
// Tx Packets
UINT32      UnicastPacketsTx;       //PacketsTx
UINT32      BroadcastPacketsTx;     //Tx FIFO Underflow
UINT32      MulticastPacketsTx;
UINT32      BeaconsTx;              //Tx Fails
UINT32      AckPacketsTx;
UINT32      RTSPacketsTx;
UINT32      CTSPacketsTx;           //TxCorrupt
// Rx Packets
UINT32      UnicastPacketsRx;       //PacketsRx
UINT32      BroadcastPacketsRx;
UINT32      MulticastPacketsRx;
UINT32      BeaconsRx;
UINT32      AckPacketsRx;           //Packets Rx with CRC OK
UINT32      RTSPacketsRx;           //Register Value
UINT32      CTSPacketsRx;
// failure
UINT32      ACKFailureCount;
UINT32      CTSFailureCount;        //Packets  Rx  with  CRC  Error
(FCSError)
} STATISTICS_MIB;
```

- **The SET_TESTMODE structure**

You should use this structure to pass the values of the test parameters to the driver. The Type member is being used for the type of the test parameters we pass to the driver,

according the test we intend to perform, as defined in `TestMode.h`. The `Data` member is being used for the values of the parameters (associated to a `TEST_PARAMS` structure)

```
typedef struct SET_TESTMODE
{
      UINT8 Type;
      UINT8 Size;
      UINT16      Reserved;
      UINT8 Data[72];

} SET_TESTMODE;
```

- **Definitions**

We use the following definitions in `TestMode.h`:

```
used in 'type' member of the SET_TESTMODE structure:
#define Command_Set_ContTx                          0x01
#define Command_Set_ContRx                          0x02
#define Command_Set_ContRxFG                        0x0d
#define Command_Set_ContTx_woModulation                   0x03
#define Command_Set_Idle                            0x04
#define Command_Carrier_Accuracy                    0x05
#define Command_Carrier_Suppression                 0x06
#define Command_Packet_Tx                           0x07
#define Command_Set_CR_Values                       0x08
#define Command_SetBB_Reg                           0x09
#define Command_ReadBB_Reg                          0x0a
#define Command_SetAiroha_Reg                       0x0b
#define Command_Reset_Stats                               0x0c


#define LEFT_ANTENNA       0x0
#define RIGHT_ANTENNA            0x1
#define A_DIVERSITY        0x3

#define Baseband_11b       1
#define Baseband_11a       2
```

# Section 4

# APPENDIX B

We use the following APIs in order to set the driver mode and pass the test parameters to the VNET driver:

- **ioctl SET_TEST_MODE**

You should use this `ioctl` in order to set the driver to Test Mode. If the state value passed using the `wrq.u.data.pointer` to the driver is 1, then the driver is set to Test Mode. If the state value is 0, the driver is being reset to normal operation.

**Example:**

```
int SetTMState(int state)
{
      struct iwreq      wrq;
      int fd __attribute__((unused));

    memset(&wrq, 0, sizeof(wrq));
    strncpy(wrq.ifr_name, "eth0", IFNAMSIZ);

      /*set the driver in test mode*/
      wrq.u.data.pointer = (caddr_t) &state;
      wrq.u.data.length  = sizeof(int);

      if (ioctl(test_inet, SET_TEST_MODE, &wrq) < 0) {
            printf("Setting Test Mode failed!!!\n");
            //inTestMode = 0;
            return -EIO;
      }

      inTestMode = state;
      return 0;

}
```

- **ioctl SET_TEST_MODE_COMMAND**

This `ioctl` is being used in order to pass the test parameters values to the driver, in order to perform a test. The parameters values are being passed to the `wrq.u.data.pointer` using the `SET_TESTMODE` structure (defined in `testcmd.h`).

**Example:**

```
int SetTestMode(SET_TESTMODE *params)
{
 struct iwreq     wrq;
 int fd __attribute__((unused));

 /*set the driver in test mode*/
    memset(&wrq, 0, sizeof(wrq));
    strncpy(wrq.ifr_name, "eth0", IFNAMSIZ);
 wrq.u.data.pointer = (caddr_t)params;
```

```
        wrq.u.data.length  = sizeof(SET_TESTMODE);

        if (ioctl(test_inet, SET_TEST_MODE_COMMAND, &wrq) < 0) {
        printf("Setting Test Mode Command failed!!!\n");
              return -EIO;
        }

        return 0;

}
```