Document Title

433MHz Band Reference Code for FIFO mode (100Kbps, 100KIF)

Revision History

Rev. No.	History	Issue Date	Remark
0.0	Preliminary	July 19, 2013	

Table of contents

AMICCOM RF Chip - A7139
Reference code for FIFO mode

1. Introduction

This document describes development of simple example procedures by A7139 FIFO mode. It could support user how to implement two-way radio and how to initial A7139.

2. Systems overview

The procedure is divided into two parts, one is Master, and another one is Slave.

Master side ：After power on and initial RF chip procedure, Master will deliver 64 bytes data from TX FIFO, then jump into RX state to wait ACK data from Slave. If Master received the ACK data, it will back to TX state to deliver next 64 byte data. If Master does NOT receive the ACK data, Master will also back to TX state for next 64 byte data delivery after staying in RX state for 100 ms.

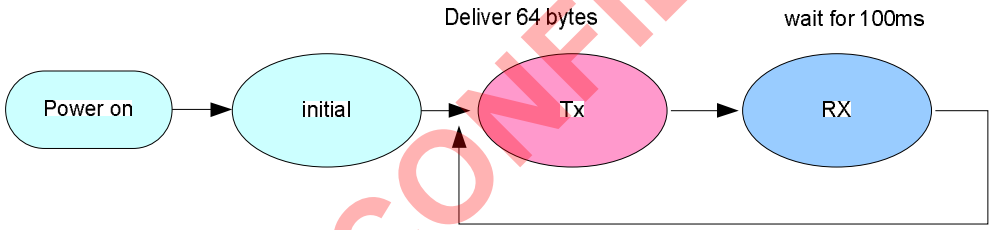

Fig1. The state diagram of master side

Slave side： After power on and initial RF procedure, Slave enters into RX state for receiving data from Master. Slave is set to stay in RX state until it receives the data. If Slave received the data from Master, it will transit to TX state to deliver 64 bytes ACK data and then back to RX state for receiving next 64 byte data from Master.

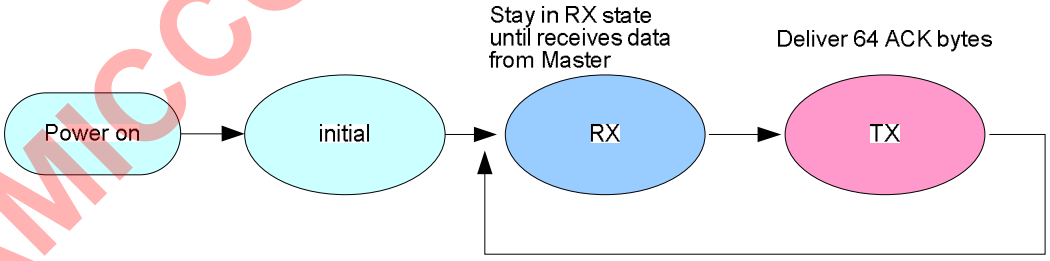

Fig2. The state diagram of Slave side

From Fig3, in Master side, Master enters into RX state to wait 64 byte ACK data once it delivers 64 byte data. If Master does not receive 64 byte ACK data within 100ms, it will back to TX state to deliver next 64 byte data. Once Master received 64 byte packet, this packet will be authenticated and calculated bit error rate. After 100 ms, Master is set to back TX state for next 64 byte delivery.

From Fig3, in Slave side, Slave stays in RX state until it receives 64 byte data from Master. Once Slave receives 64 byte packet, this packet will be authenticated and calculated bit error rate. Then, Slave is set to enter TX state to deliver 64 byte ACK data to Master.
Based on the sample procedures between Master and Slave, user can learn how to implement two-way radio as well as how to calculate BER (bit error rate).

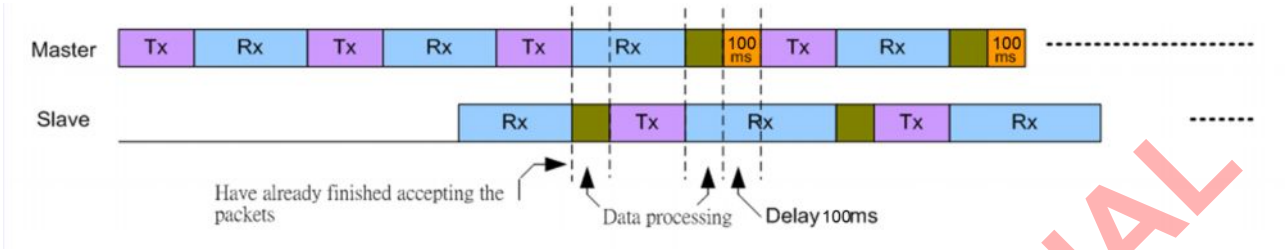

Fig3. Timing chart between Master and Slave

3. Hardware

3.1 System block diagram

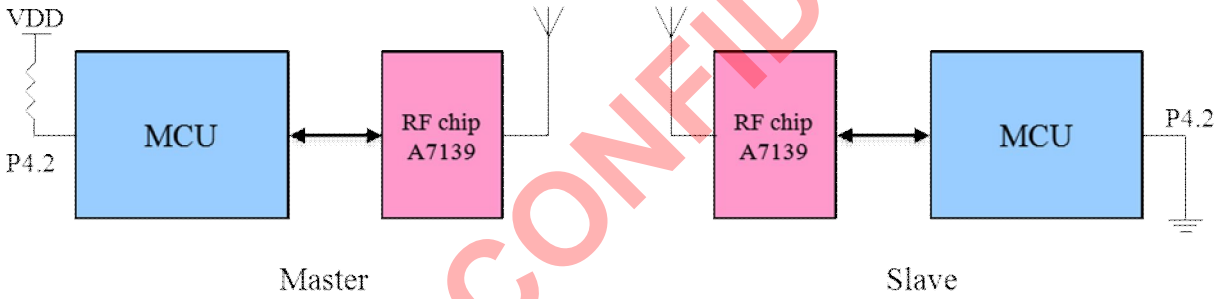

Fig4. System block diagram

MCU I/O Pin Definition:

- Set MCU I/O P4.2 to identify Master (if P4.2 =high) or Slave (if P4.2 =low).
- Set MCU I/O P1.0, P1.1, P1.2 to assign SCS, SCK, SDIO pin.
- Set MCU I/O P1.3, P1.4, P1.5 to assign CKO, GIO1, GIO2 pin.

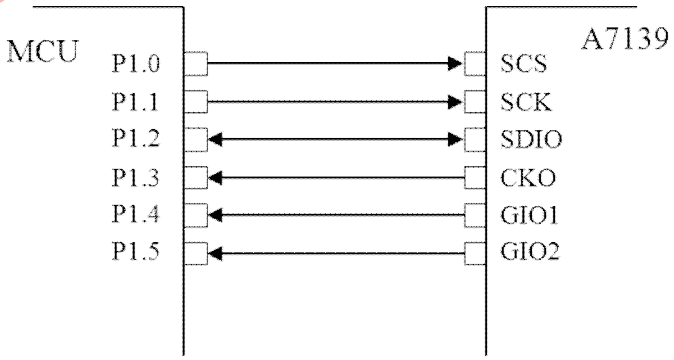

Fig5. Connections between 8051 MCU and A7139

4. Firmware Program

4.1 Introduction

After power on reset, MCU do initialization of its Timer0 and Uart0 as well as A7139. Then, MCU check its P4.2 to identify Master or Slave. If P4.2 = 1, MCU executes Master code in the main program; else, MCU executes Slave code in the main program.

Master code：

1) Writes 64 bytes PN9 code into TX FIFO.
2) A7139 enter TX State to deliver 64 bytes PN9 code. After done, A7139 is auto back to Standby state.
3) A7139 enter RX state to wait 64 bytes ACK data.
4) Enable Timer 0 and clear TimeoutFlag flag
5) If TimeoutFlag = 1 (timeout = 100ms), back to step(1).
6) Once A7139 received the packet, A7139 will be auto back to Standby state.
7) MCU compares received 64 bytes data with PN9 code and calculates BER (Bit Error Rate).
8) MCU calls delay loop for 100 ms, then back to step (1).
9) For each 500 ms, MCU reports BER to personal computer.

Slave code：

1) A7139 enter RX state until it receives 64 byte data from Master.
2) Once A7139 received the packet, A7139 will be auto back to Standby state.
3) MCU compares received 64 bytes data with PN9 code and calculates BER (Bit Error Rate).
4) MCU writes 64 bytes PN9 code into TX FIFO.
5) A7139 enter TX State to deliver 64 byte PN9 code. After done, A7139 is auto back to Standby state.
6) Back to step (1).
7) For each 500 ms, MCU reports BER to personal computer.

4.2 Example State Diagram

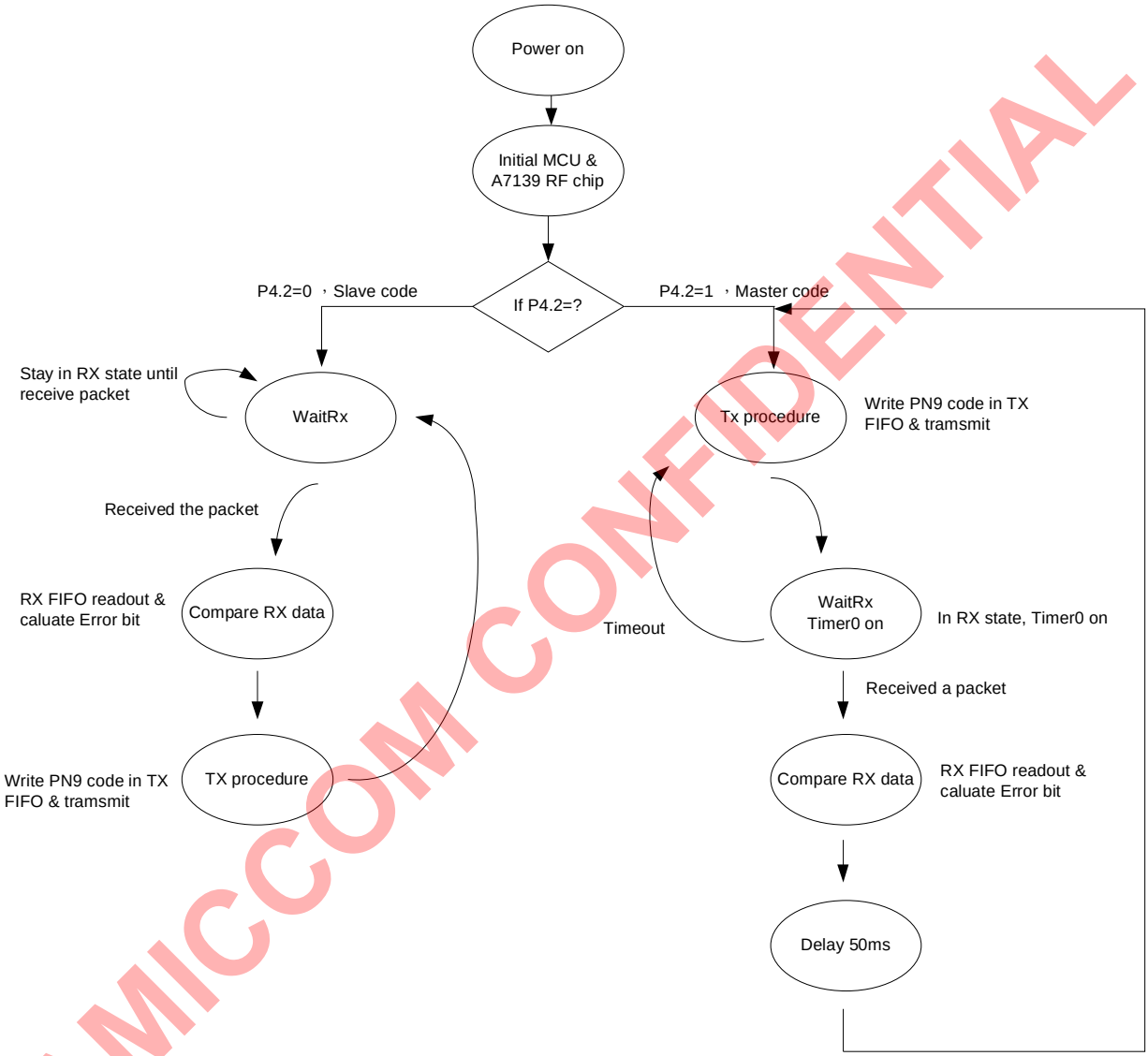

Fig6 State diagram of Master code and Slave code

5. Explanation of reference code

```
 1 /*********************************************************************
 2 **  Device:       A7139
 3 **  File:         main.c
 4 **  Target:       Winbond W77LE58
 5 **  Tools:        ICE
 6 **  Updated:      2013-07-19
 7 **  Description:
 8 **  This file is a sample code for your reference.
 9 **
10 **  Copyright (C) 2011 AMICCOM Corp.
11 **
12 *********************************************************************/
13 #include "define.h"
14 #include "w77le58.h"
15 #include "A7139reg.h"
16 #include "Uti.h"
```

Function: Include file declaration	
Line	Description
13~16	Include the link files

```
18 /*********************************************************************
19 **  I/O Declaration
20 *********************************************************************/
21 #define SCS             P1_0    //SPI SCS
22 #define SCK             P1_1    //SPI SCK
23 #define SDIO            P1_2    //SPI SDIO
24 #define CKO             P1_3    //CKO
25 #define GIO1            P1_4    //GIO1
26 #define GIO2            P1_5    //GIO2
27
28 /*********************************************************************
29 **  Constant Declaration
30 *********************************************************************/
31 #define TIMEOUT         100     //100ms
32 #define t0hrel          1000    //1ms
```

Function: Define the I/O Port of A7139 RF chip by MCU, Define the constant parameter	
Line	Description
21~26	MCU I/O definition
31~32	Constant declaration

```
34 /*********************************************************************
35 **  Global Variable Declaration
36 *********************************************************************/
37 Uint8   data    timer;
38 Uint8   data    TimeoutFlag;
39 Uint16  idata   RxCnt;
40 Uint32  idata   Err_ByteCnt;
41 Uint32  idata   Err_BitCnt;
42 Uint16  idata   TimerCnt0;
43 Uint8   data    *Uartptr;
44 Uint8   data    UartSendCnt;
45 Uint8   data    CmdBuf[11];
46 Uint8   idata   tmpbuf[64];
47
48 const Uint8 code BitCount_Tab[16]={0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4};
49 const Uint8 code ID_Tab[8]={0x34,0x75,0xC5,0x8C,0xC7,0x33,0x45,0xE7};   //ID code
50 const Uint8 code PN9_Tab[]=
51 {   0xFF,0x83,0xDF,0x17,0x32,0x09,0x4E,0xD1,
52     0xE7,0xCD,0x8A,0x91,0xC6,0xD5,0xC4,0xC4,
53     0x40,0x21,0x18,0x4E,0x55,0x86,0xF4,0xDC,
54     0x8A,0x15,0xA7,0xEC,0x92,0xDF,0x93,0x53,
55     0x30,0x18,0xCA,0x34,0xBF,0xA2,0xC7,0x59,
56     0x67,0x8F,0xBA,0x0D,0x6D,0xD8,0x2D,0x7D,
57     0x54,0x0A,0x57,0x97,0x70,0x39,0xD2,0x7A,
58     0xEA,0x24,0x33,0x85,0xED,0x9A,0x1D,0xE0
59 };// This table are 64bytes PN9 pseudo random code.
```

Function: Declaration of the parameter

Line	Description
37~46	Declare parameters that are used in the procedure
48	Declare BitCount_Tab
49	Declare ID_Tab for ID code
50~59	Declare PN9_Tab of 64 bytes PN9 code

```
 61 const Uint16 code A7139Config[]=            //433MHz, 100kbps (IFBW = 100KHz, Fdev = 37.5KHz)
 62 {
 63     0x0021,     //SYSTEM CLOCK register,
 64     0x0A21,     //PLL1 register,
 65     0xDA05,     //PLL2 register,     433.301MHz
 66     0x0000,     //PLL3 register,
 67     0x0A20,     //PLL4 register,
 68     0x0024,     //PLL5 register,
 69     0x0000,     //PLL6 register,
 70     0x0011,     //CRYSTAL register,
 71     0x0000,     //PAGEA,
 72     0x0000,     //PAGEB,
 73     0x18D4,     //RX1 register,      IFBW=100KHz
 74     0x7009,     //RX2 register,      by preamble
 75     0x4000,     //ADC register,
 76     0x0800,     //PIN CONTROL register,          Use Strobe CMD
 77     0x4C45,     //CALIBRATION register,
 78     0x20C0      //MODE CONTROL register,        Use FIFO mode
 79 };
 80
 81 const Uint16 code A7139Config_PageA[]=   //433MHz, 100kbps (IFBW = 100KHz, Fdev = 37.5KHz)
 82 {
 83     0xF706,     //TX1 register,      Fdev = 37.5kHz
 84     0x0000,     //WOR1 register,
 85     0xF800,     //WOR2 register,
 86     0x1107,     //RFI register,      Enable Tx Ramp up/down
 87     0x0170,     //PM register,
 88     0x0201,     //RTH register,
 89     0x400F,     //AGC1 register,
 90     0x2AC0,     //AGC2 register,
 91     0x0045,     //GIO register,      GIO2=WTR, GIO1=FSYNC
 92     0xD181,     //CKO register
 93     0x0004,     //VCB register,
 94     0x0A21,     //CHG1 register,    430MHz
 95     0x0022,     //CHG2 register,    435MHz
 96     0x003F,     //FIFO register,    FEP=63+1=64bytes
 97     0x1507,     //CODE register,    Preamble=4bytes, ID=4bytes
 98     0x0000      //WCAL register,
 99 };
100
101 const Uint16 code A7139Config_PageB[]=   //433MHz, 100kbps (IFBW = 100KHz, Fdev = 37.5KHz)
102 {
103     0x0337,     //TX2 register,
104     0x8400,     //IF1 register,      Enable Auto-IF, IF=200KHz
105     0x0000,     //IF2 register,
106     0x0000,     //ACK register,
107     0x0000      //ART register,
108 };
```

Function: Configure of A7139's control registers

Line	Description
61~108	Configure of A7139's control registers

```
110 /*********************************************************************
111  **  function Declaration
112  *********************************************************************/
113 void Timer0ISR(void);
114 void UART0Isr(void);
115 void InitTimer0(void);
116 void InitUART0(void);
117 void InitRF(void);
118 void A7139_Config(void);
119 void A7139_WriteID(void);
120 void A7139_Cal(void);
121 void StrobeCMD(Uint8);
122 void ByteSend(Uint8);
123 Uint8 ByteRead(void);
124 void A7139_WriteReg(Uint8, Uint16);
125 Uint16 A7139_ReadReg(Uint8);
126 void A7139_WritePageA(Uint8, Uint16);
127 Uint16 A7139_ReadPageA(Uint8);
128 void A7139_WritePageB(Uint8, Uint16);
129 Uint16 A7139_ReadPageB(Uint8);
130 void A7139_WriteFIFO(void);
131 void RxPacket(void);
132 void Err_State(void);
```

Function: Declaration of function

Line	Description
113~132	Subroutine declarations

```
/*******************************************************************
 * main loop
 *******************************************************************/
void main(void)
{
    //initsw
    PMR |= 0x01;        //set DME0

    //initHW
    P0 = 0xFF;
    P1 = 0xFF;
    P2 = 0xFF;
    P3 = 0xFF;
    P4 = 0x0F;

    InitTimer0();
    InitUART0();
    TR0=1;    //Timer0 on
    EA=1;     //enable interrupt

    if((P4 & 0x04)==0x04)         //if P4.2=1, master
    {
        InitRF(); //init RF

        while(1)
        {
            A7139_WriteFIFO();    //write data to TX FIFO
            StrobeCMD(CMD_TX);
            Delay10us(1);
            while(GIO2);              //wait transmit completed
            StrobeCMD(CMD_RX);
            Delay10us(1);

            timer=0;
            TimeoutFlag=0;
            while((GIO2==1)&&(TimeoutFlag==0));   //wait receive completed
            if(TimeoutFlag)
            {
                StrobeCMD(CMD_STBY);
            }
            else
            {
                RxPacket();
                Delay10ms(10);
            }
        }
    }
    else        //if P4.2=0, slave
    {
        InitRF(); //init RF

        RxCnt = 0;
        Err_ByteCnt = 0;
        Err_BitCnt = 0;

```

```
189         while(1)
190         {
191             StrobeCMD(CMD_RX);
192             Delay10us(1);
193             while(GIO2);                //wait receive completed
194             RxPacket();
195
196             A7139_WriteFIFO();    //write data to TX FIFO
197             StrobeCMD(CMD_TX);
198             Delay10us(1);
199             while(GIO2);                //wait transmit completed
200
201             Delay10ms(9);
202         }
203     }
204 }
```

Function: Main loop.

Line	Description
140	Enable the MCU on chip data SRAM
143~147	Initial MCU I/O Port
149	Call subroutine of initTimer0 and enable interrupt.
150	Call subroutine of initUart0.
151	Enable Timer0
152	Enable interrupt
154	Check P4.2. If P4.2=1, MCU executes Master code. Else, MCU executes Slave code.
156~179	Master loop
156	Call subroutine of initRF to initial A7139 chip
160	Call subroutine of A7139_WriteFIFO to write data into TX FIFO
161	Send strobe command to enter TX mode, transmit data.
163	wait transmit completed.
164	Send strobe command to enter RX state.
167~168	Clean timer0 variable and Timeout Flag
169	Wait for received data or Timeout
170~173	Check Timeout Flag. if Timeout, send strobe command to entry standby mode.
176	Call subroutine of RxPacket to read data from RX FIFO, do authentication and calculate BER
177	Delay 100ms.
183~202	Slave loop
183	Call subroutine of initRF to initial A7139 chip.
185~187	Clear RxCnt, Err_ByteCnt and Err_BitCnt.
191	Send strobe command to enter RX state.
193	Wait for received data
194	Call subroutine of RxPacket to read data from RX FIFO, do authentication and calculate BER
196	Call subroutine of A7139_WriteFIFO to write data into TX FIFO
197	Send strobe command to enter TX mode, transmit data.
199	wait transmit completed
201	Delay 90ms.

```
206 /*******************************************************************************
207 **  Timer0ISR
208 *******************************************************************************/
209 void Timer0ISR (void) interrupt 1
210 {
211     TH0 = (65536-t0hrel)>>8;// Reload Timer0 high byte,low byte
212     TL0 = 65536-t0hrel;
213
214     timer++;
215     if (timer >= TIMEOUT)
216     {
217         TimeoutFlag=1;
218     }
219
220     TimerCnt0++;
221     if (TimerCnt0 == 500)
222     {
223         TimerCnt0=0;
224         CmdBuf[0]=0xF1;
225
226         memcpy(&CmdBuf[1], &RxCnt, 2);
227         memcpy(&CmdBuf[3], &Err_ByteCnt, 4);
228         memcpy(&CmdBuf[7], &Err_BitCnt, 4);
229
230         UartSendCnt=11;
231         Uartptr=&CmdBuf[0];
232         SBUF=CmdBuf[0];
233     }
234 }
```

Function: Subroutine of Timer0 interrupt

Line	Description
211~212	setup initial value of TH0 and TL0.
214	Increase timer (timer is a variable).
215~218	Check if timer= TIMEOUT. If yes, set TimeoutFlag =1.
220	Increase timerCnt0 (timer is a variable).
221	Check if TimerCnt0 = 500, if yes, 500ms delay is done.
223	Clear TimerCnt0
224	CmdBuf[0] is set to 0xF1 for identify code
226	CmdBuf[1] is used to set up parameter RxCnt
227	CmdBuf[3] 、CmdBuf[4]、CmdBuf[5] 、CmdBuf[6] are used to set up parameter Err_ByteCnt
228	CmdBuf[7] 、CmdBuf[8]、CmdBuf[9] 、CmdBuf[10] are used to set up parameter Err_BitCnt
230	Set UartSendCnt=11.
231	Set pointer Uartptr to indicate location of CmdBuf [0].
232	Deliver SBUF to PC for BER information.

```
236 /*******************************************************************************
237 **  Uart0ISR
238 *******************************************************************************/
239 void Uart0Isr(void) interrupt 4 using 3
240 {
241     if (TI==1)
242     {
243         TI=0;
244         UartSendCnt--;
245         if(UartSendCnt !=0)
246         {
247             Uartptr++;
248             SBUF = *Uartptr;
249         }
250     }
251 }
```

Function: Initial the Uart0 interrupt subroutine

Line	Description
241	Check TI flag, if TI=1, one byte of UART transmission is done.
243	Clear TI flag
244	Decrease UartSendCnt (UartSendCnt is a variable)
245	Check UartSendCnt == 0, if not, continue UART transmission until done.
247~248	Increase pointer Uartptr to assign address of next data via UART0 to PC

```
253  /*******************************************************************************
254 **  init Timer0
255 *******************************************************************************/
256 void InitTimer0(void)
257 {
258     TR0 = 0;
259     TMOD =(TMOD & 0xF0)|0x01;       //timer0 mode=1
260     TH0 = (65536-t0hrel)>>8;        //setup Timer0 high byte,low byte
261     TL0 = 65536-t0hrel;
262     TF0 = 0;                        //Clear any pending Timer0 interrupts
263     ET0 = 1;                        // Enable Timer0 interrupt
264 }
```

Function: Subroutine of InitTimer0

Line	Description
258	Disable Timer0
259	Setup Timer0 in mode1 mode
260~261	Setup the initial value of TH0,TL0
262	Clean Timer0 interrupt flag
263	Enable Timer0 interrupt

```
266 /*******************************************************************************
267 **  Init Uart0
268 *******************************************************************************/
269 void initUart0(void)
270 {
271     TH1 = 0xFD;         //BaudRate 9600;
272     TL1 = 0xFD;
273     SCON = 0x40;
274     TMOD = (TMOD & 0x0F) | 0x20;
275     REN = 1;
276     TR1 = 1;
277     ES = 1;
278 }
```

Function: Initial Uart0 procedure	
Line	Description
271~273	Initial TL1, TH1, SCON1 value, sets up for 9600bps @xtal =11.0592MHz
274	Set Timer1 is mode 2.
275~277	Set up REN1, TR1, and ES1 is 1. Enable the function of UART.

```
280 /*******************************************************************************
281 ** Strobe Command
282 *******************************************************************************/
283 void StrobeCMD (Uint8 cmd)
284 {
285     Uint8 i;
286
287     SCS = 0;
288     for(i = 0; i < 8; i++)
289     {
290         if(cmd & 0x80)
291             SDIO = 1;
292         else
293             SDIO = 0;
294
295         _nop_();
296         SCK = 1;
297         _nop_();
298         SCK = 0;
299         cmd <<= 1;
300     }
301     SCS = 1;
302 }
```

Function: subroutine of StrobeCmd	
Line	Description
287~301	Write one byte data of strobe command procedure.

```
304 /*********************************************************************
305 **  ByteSend
306 *********************************************************************/
307 void ByteSend(Uint8 src)
308 {
309     Uint8 i;
310
311     for(i = 0; i < 8; i++)
312     {
313         if(src & 0x80)
314             SDIO = 1;
315         else
316             SDIO = 0;
317
318         _nop_();
319         SCK = 1;
320         _nop_();
321         SCK = 0;
322         src <<= 1;
323     }
324 }
```

Function: Subroutine of ByteSend	
Line	Description
311~323	Write one byte data procedure.

```
326 /*********************************************************************
327 **  ByteRead
328 *********************************************************************/
329 Uint8 ByteRead(void)
330 {
331     Uint8 i, tmp;
332
333     //read data code
334     SDIO = 1;       //SDIO pull high
335     for(i = 0; i < 8; i++)
336     {
337         if(SDIO)
338             tmp = (tmp << 1) | 0x01;
339         else
340             tmp = tmp << 1;
341
342         SCK = 1;
343         _nop_();
344         SCK = 0;
345     }
346     return tmp;
347 }
```

Function: subroutine of ByteRead	
Line	Description
335~345	Read one byte data procedure.
346	Return tmp value.

```
349 /*********************************************************************
350 **  A7139_WriteReg
351 *********************************************************************/
352 void A7139_WriteReg(Uint8 address, Uint16 dataWord)
353 {
354     Uint8 i;
355
356     SCS = 0;
357     address |= CMD_Reg_W;
358     for(i = 0; i < 8; i++)
359     {
360         if(address & 0x80)
361             SDIO = 1;//bit=1
362         else
363             SDIO = 0;//bit=0
364
365         SCK = 1;
366         _nop_();
367         SCK = 0;
368         address <<= 1;
369     }
370     _nop_();
371
372     //send data word
373     for(i = 0; i < 16; i++)
374     {
375         if(dataWord & 0x8000)
376             SDIO = 1;
377         else
378             SDIO = 0;
379
380         SCK = 1;
381         _nop_();
382         SCK = 0;
383         dataWord <<= 1;
384     }
385     SCS = 1;
386 }
```

Function: Write operation for A7139 control register

Line	Description
356	Set SCS=0 to Enable SPI interface.
357	Enable write operation of control registers.
358~369	Assign control register's address by input variable addr.
373~384	Assign control register's value by input variable dataByte
385	Set SCS=1 to disable SPI interface.

```
388 /*********************************************************************
389 **  A7139_ReadReg
390 *********************************************************************/
391 Uint16 A7139_ReadReg(Uint8 address)
392 {
393     Uint8 i;
394     Uint16 tmp;
395
396     SCS = 0;
397     address |= CMD_Reg_R;
398     for(i = 0; i < 8; i++)
399     {
400         if(address & 0x80)
401             SDIO = 1;
402         else
403             SDIO = 0;
404
405         _nop_();
406         SCK = 1;
407         _nop_();
408         SCK = 0;
409
410         address <<= 1;
411     }
412     _nop_();
413
414     //read data code
415     SDIO = 1;       //SDIO pull high
416     for(i = 0; i < 16; i++)
417     {
418         if(SDIO)
419             tmp = (tmp << 1) | 0x01;
420         else
421             tmp = tmp << 1;
422
423         SCK = 1;
424         _nop_();
425         SCK = 0;
426     }
427     SCS = 1;
428     return tmp;
429 }
```

Function: Read operation for A7139 control register	
Line	Description
396	Set SCS=0 to enable SPI write function.
397	Enable read operation of control registers.
398~411	Write address procedure.
416~426	Read data word procedure.
427	Set SCS=1 to disable SPI interface.
428	Return tmp value

```
431 /*********************************************************************
432 **  A7139_WritePageA
433 *********************************************************************/
434 void A7139_WritePageA(Uint8 address, Uint16 dataWord)
435 {
436     Uint16 tmp;
437
438     tmp = address;
439     tmp = ((tmp << 12) | A7139Config[CRYSTAL_REG]);
440     A7139_WriteReg(CRYSTAL_REG, tmp);
441     A7139_WriteReg(PAGEA_REG, dataWord);
442 }
```

Function: Subroutine of A7139 page A register writing operation.

Line	Description
438~440	Set the writing-intended page A register address to register CRYSTAL bit[15:12]
441	Write data to dedicated page A register. (By address CRYSTAL bit[15:12])

```
444 /*********************************************************************
445 **  A7139_ReadPageA
446 *********************************************************************/
447 Uint16 A7139_ReadPageA(Uint8 address)
448 {
449     Uint16 tmp;
450
451     tmp = address;
452     tmp = ((tmp << 12) | A7139Config[CRYSTAL_REG]);
453     A7139_WriteReg(CRYSTAL_REG, tmp);
454     tmp = A7139_ReadReg(PAGEA_REG);
455     return tmp;
456 }
```

Function: Subroutine of A7139 page A register reading operation.

Line	Description
451~453	Set the writing-intended page A register address to register CRYSTAL bit[15:12]
454	Read data from dedicated page A register. (By address CRYSTAL bit[15:12])
455	Return tmp value

```
458 /*********************************************************************
459 **  A7139_WritePageB
460 *********************************************************************/
461 void A7139_WritePageB(Uint8 address, Uint16 dataWord)
462 {
463     Uint16 tmp;
464
465     tmp = address;
466     tmp = ((tmp << 7) | A7139Config[CRYSTAL_REG]);
467     A7139_WriteReg(CRYSTAL_REG, tmp);
468     A7139_WriteReg(PAGEB_REG, dataWord);
469 }
```

Function: Subroutine of A7139 page B register writing operation.

Line	Description
465~467	Set the writing-intended page B register address to register CRYSTAL bit[9:7]
468	Write data to dedicated page B register. (By address CRYSTAL bit[9:7])

```
471 /*********************************************************************
472 **  A7139_ReadPageB
473 *********************************************************************/
474 Uint16 A7139_ReadPageB(Uint8 address)
475 {
476     Uint16 tmp;
477
478     tmp = address;
479     tmp = ((tmp << 7) | A7139Config[CRYSTAL_REG]);
480     A7139_WriteReg(CRYSTAL_REG, tmp);
481     tmp = A7139_ReadReg(PAGEB_REG);
482     return tmp;
483 }
```

Function: Subroutine of A7139 page B register reading operation.

Line	Description
478~480	Set the writing-intended page B register address to register CRYSTAL bit[9:7]
481	Read data from dedicated page B register. (By address CRYSTAL bit[9:7])
482	Return tmp value

```
485 /*********************************************************************
486 ** initRF
487 *********************************************************************/
488 void InitRF(void)
489 {
490     //initial pin
491     SCS = 1;
492     SCK = 0;
493     SDIO= 1;
494     CKO = 1;
495     GIO1= 1;
496     GIO2= 1;
497
498     StrobeCMD(CMD_RF_RST);          //reset A7139 chip
499     A7139_Config();                 //config A7139 chip
500     Delay100us(8);                  //delay 800us for crystal stabilized
501     A7139_WriteID();                //write ID code
502     A7139_Cal();                    //IF and VCO calibration
503 }
```

Function: Initial RF chip

Line	Description
491~496	Set up I/O initial value.
498	Send strobe command to reset A7139
499	Call subroutine of A7139_Config to initial RF chip.
500	Delay and waiting for crystal stabilized.
501	Call subroutine of A7139_WriteID to write 4 bytes ID code.
502	Call subroutine of A7139_Cal to do calibration procedures.

```
505 /*********************************************************************
506 ** A7139_Config
507 *********************************************************************/
508 void A7139_Config(void)
509 {
510     Uint8 i;
511     Uint16 tmp;
512
513     for(i=0; i<8; i++)
514         A7139_WriteReg(i, A7139Config[i]);
515
516     for(i=10; i<16; i++)
517         A7139_WriteReg(i, A7139Config[i]);
518
519     for(i=0; i<16; i++)
520         A7139_WritePageA(i, A7139Config_PageA[i]);
521
522     for(i=0; i<5; i++)
523         A7139_WritePageB(i, A7139Config_PageB[i]);
524
525     //for check
526     tmp = A7139_ReadReg(SYSTEMCLOCK_REG);
527     if(tmp != A7139Config[SYSTEMCLOCK_REG])
528     {
529         Err_State();
530     }
531 }
```

Function: Initial the registers of RF chip

Line	Description
513~517	Write initial value into A7139 register.
519~520	Write initial value into A7139 page A register.
522~523	Write initial value into A7139 page B register.
526~530	Read register(00h) for check

```
533 /*********************************************************************
534 **  WriteID
535 *********************************************************************/
536 void A7139_WriteID(void)
537 {
538     Uint8 i;
539     Uint8 d1, d2, d3, d4;
540
541     SCS=0;
542     ByteSend(CMD_ID_W);
543     for(i=0; i<4; i++)
544             ByteSend(ID_Tab[i]);
545     SCS=1;
546
547     SCS=0;
548     ByteSend(CMD_ID_R);
549     d1=ByteRead();
550     d2=ByteRead();
551     d3=ByteRead();
552     d4=ByteRead();
553     SCS=1;
554
555     if((d1!=ID_Tab[0]) || (d2!=ID_Tab[1]) || (d3!=ID_Tab[2]) || (d4!=ID_Tab[3]))
556     {
557         Err_State();
558     }
559 }
```

Function: subroutine of setting ID code registers

Line	Description
541	Set SCS=0 to enable SPI interface.
542	Sent Write ID Command.
543~544	Write ID_Tab Table into A7139 ID Code registers.
545	Set SCS=1 to disable SPI interface.
547	Set SCS=0 to enable SPI interface.
548	Sent Read ID Command.
549~552	Read ID code 4bytes
553	Set SCS=1 to disable SPI interface.
555~558	Check ID code

```
561 /*********************************************************************
562 ** A7139_Cal
563 *********************************************************************/
564 void A7139_Cal(void)
565 {
566     Uint8 fb, fcd, fbcf;            //IF Filter
567     Uint8 vb,vbcf;                  //VCO Current
568     Uint8 vcb, vccf;                //VCO Band
569     Uint16 tmp;
570
571     //IF calibration procedure @STB state
572     A7139_WriteReg(MODE_REG, A7139Config[MODE_REG] | 0x0802);    //IF Filter & VCO Current Calibration
573     do{
574         tmp = A7139_ReadReg(MODE_REG);
575     }while(tmp & 0x0802);
576
577     //for check(IF Filter)
578     tmp = A7139_ReadReg(CALIBRATION_REG);
579     fb = tmp & 0x0F;
580     fcd = (tmp>>11) & 0x1F;
581     fbcf = (tmp>>4) & 0x01;
582     if(fbcf)
583     {
584         Err_State();
585     }
586
587     //for check(VCO Current)
588     tmp = A7139_ReadPageA(VCB_PAGEA);
589     vcb = tmp & 0x0F;
590     vccf = (tmp>>4) & 0x01;
591     if(vccf)
592     {
593         Err_State();
594     }
595
596
597     //RSSI Calibration procedure @STB state
598     A7139_WriteReg(ADC_REG, 0x4C00);                    //set ADC average=64
599     A7139_WriteReg(MODE_REG, A7139Config[MODE_REG] | 0x1000);    //RSSI Calibration
600     do{
601         tmp = A7139_ReadReg(MODE_REG);
602     }while(tmp & 0x1000);
603     A7139_WriteReg(ADC_REG, A7139Config[ADC_REG]);
604
605
```

```
606     //VCO calibration procedure @STB state
607     A7139_WriteReg(PLL1_REG, A7139Config [PLL1_REG]);
608     A7139_WriteReg(PLL2_REG, A7139Config [PLL2_REG]);
609     A7139_WriteReg(MODE_REG, A7139Config[MODE_REG] | 0x0004);      //VCO Band Calibration
610     do{
611        tmp = A7139_ReadReg(MODE_REG);
612     }while(tmp & 0x0004);
613
614     //for check(VCO Band)
615     tmp = A7139_ReadReg(CALIBRATION_REG);
616     vb = (tmp >>5) & 0x07;
617     vbcf = (tmp >>8) & 0x01;
618     if(vbcf)
619     {
620        Err_State();
621     }
622 }
```

Function: VCO,IF and RSSI calibration procedure

Line	Description
572	The calibration process suggested to run at standby or PLL state. Set mode control register, bit FBC=1 and VCC=1.
573~575	Read value of mode control register, and check bit FBC and VCC. If bit FBC=0 and VCC=0, exit the loop.
578~585	Read value of calibration register and check it. If bit fbcf=1, jump to Err_State procedure.
588~594	Read value of vco current register and check it. If bit vccf=1, jump to Err_State procedure.
598	Set ADC average=64.
599	Set mode control register, bit RSSC=1.
600~602	Read value of mode control register, and check bit RSSC. If bit RSSC=0, exit the loop.
603	Set original ADC average.
607~621	VCO calibration procedure
607~608	Settling frequency.
609	Set mode control register, bit VBC=1.
610~612	Read value of mode control register and check bit VBC. If bit VBC=0, exit the loop.
615~621	Read value of calibration register, and check it. If bit vbcf=1, jump to Err_State procedure.

```
624 /*********************************************************************
625 ** A7139_WriteFIFO
626 *********************************************************************/
627 void A7139_WriteFIFO(void)
628 {
629     Uint8 i;
630
631     StrobeCMD(CMD_TFR);              //TX FIFO address pointer reset
632
633     SCS=0;
634     ByteSend(CMD_FIFO_W);            //TX FIFO write command
635     for(i=0; i <64; i++)
636         ByteSend(PN9_Tab[i]);
637     SCS=1;
638 }
```

Function: Subroutine of write TX FIFO

Line	Description
631	Send Strobe command to reset write pointer of TX FIFO.
633	Set SCS=0 to enable SPI interface.
634	Sent Write TX FIFO Command.
635~636	Write PN9_Tab into TX FIFO, total 64 bytes.
637	Set SCS=1 to disable SPI interface.

```
640 /*********************************************************************
641 ** RxPacket
642 *********************************************************************/
643 void RxPacket(void)
644 {
645     Uint8 i;
646     Uint8 recv;
647     Uint8 tmp;
648
649     RxCnt++;
650
651     StrobeCMD(CMD_RFR);    //RX FIFO address pointer reset
652
653     SCS=0;
654     ByteSend(CMD_FIFO_R); //RX FIFO read command
655     for(i=0; i <64; i++)
656     {
657         tmpbuf[i] = ByteRead();
658     }
659     SCS=1;
660
661     for(i=0; i<64; i++)
662     {
663         recv = tmpbuf[i];
664         tmp = recv ^ PN9_Tab[i];
665         if(tmp!=0)
666         {
667             Err_ByteCnt++;
668             Err_BitCnt += (BitCount_Tab[tmp>>4] + BitCount_Tab[tmp & 0x0F]);
669         }
670     }
671 }
```

Function: Read received data from RX FIFO

Line	Description
649	Increase parameter RxCnt
651	Send Strobe command to reset read pointer of RX FIFO.
653	Set SCS=0 to enable SPI interface.
654	Sent Read RX FIFO Command.
655~658	Read 64 bytes RX FIFO
659	Set SCS=1 to disable SPI interface.
661~670	compare received data with PN9_Tab and calculate BER

```
673 /*********************************************************************
674 ** Err_State
675 *********************************************************************/
676 void Err_State(void)
677 {
678     //ERR display
679     //Error Proc...
680     //...
681     while(1);
682 }
```

Function: Error state Process

Line	Description
678~681	User define process of error state