## Document Title

**A7105 reference code for FIFO mode**

## Revision History

| Rev. No. | History | Issue Date | Remark |
|---|---|---|---|
| 0.0 | Preliminary | Oct 15 , 2007 | |
| 0.1 | Preliminary | Mar. 6, 2008 | |
| 0.2 | Modify RF config setting | Apr. 9, 2008 | |
| 0.3 | Add hopping function, modify calibration procedure | Jun, 11, 2008 | |
| 0.4 | Add frequency calibration function, delete section 3.2 | Nov. 18, 2008 | |

## Table of contents

# RF Chip-A7105
# Reference code for FIFO mode

## 1. 簡介

這文件係對 RF chip -A7105 FIFO mode 做一簡單的應用範例程式 ，供使用者能夠快速應用這 RF chip。

## 2. 系統概述

本範例程式使用簡單的跳頻(frequency hopping)機制，時序如下圖：

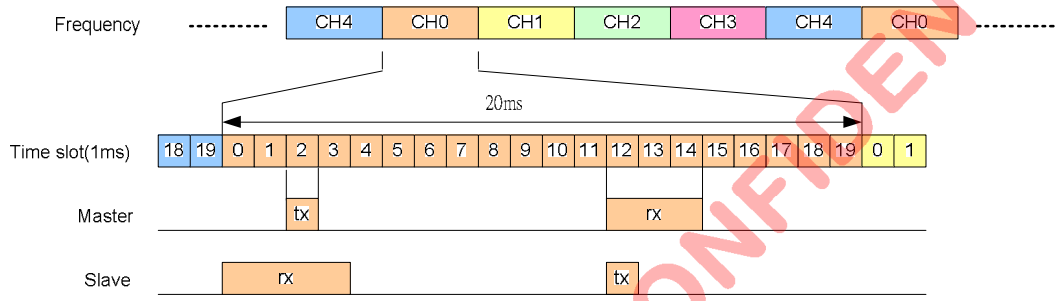


Fig1. 跳頻機制時序圖

程式主要分二個部份，一個爲 master 端，另一個爲 slave 端。

Master 端：power on、initial MCU、RF chip 後，等待 time slot=2 時，進入 TX 狀態，傳送 64 bytes 資料。之後等待 time slot=12 時，再進入 RX 狀態，等待接收。如收到資料，會自動改變下一次的工作頻率序列，重新另一次的時序週期動作。若未收到資料，Master 端會自動改變下一次的工作頻率序列，重新另一次的時序週期動作。

Slave 端：power on、initial MCU、RF chip 及進入頻率校正程序後，等待 time slot=0 時，進入 RX 狀態等待接收。若無收到 Master 端所發送的資料，則會自動改變下一次的工作頻率序列，等待下一次 time slot=0 的時序週期動作。若仍未收到資料有 5 次時序週期，則停止跳頻機制，並回到初始工作頻率，進入 RX 狀態等待接收。若有收到 Master 端所發送的資料，則重新啓動跳頻機制，依時序完成 TX 及 RX 工作。

一旦接收到封包，讀出資料、比對，計算 error bit 後，再發送封包給 Master 端。使用者可依據簡易的計算 error bit 及傳送封包數，得出 BER(bit error rate)，作爲傳輸品質的數據。

## 3. 硬體

### 3.1 系統方塊圖

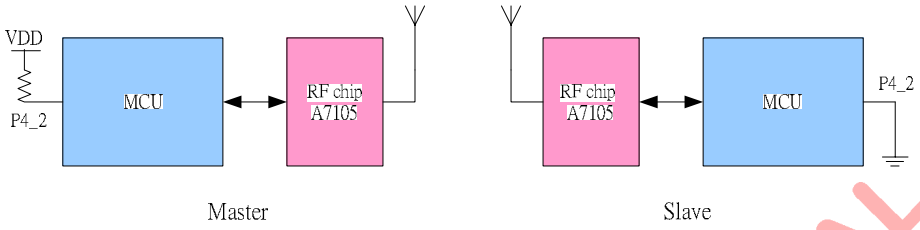


Fig2. 系統方塊圖

MCU 使用 I/O pin 4_2 的設定，判別 Master 端或 Slave 端。

**使用 I/O pin 設定：**
應用範例使用 I/O：
SCS, SCK, SDIO - 這 3 wire 串列介面控制 A7105 內部 register。
GPIO1 - FIFO 動作完成的控制信號，MCU 可檢測該 pin 是否傳送或接收 packet 完成。

**MCU 控制 A7105 RF chip 的 I/O 配置如下圖：**

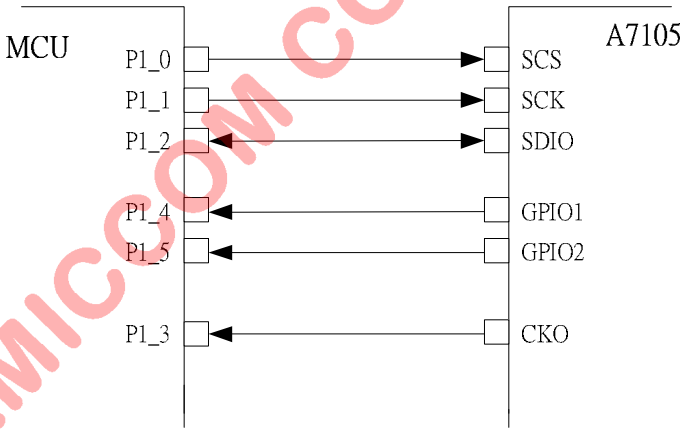


Fig3. I/O 配置圖

## 4. 韌體程式設計:

### 4.1 應用範例概述

首先初始化 Timer0、Uart0，之後判別 Port 4_2 =1 進入 master 端的主程式或 Port 4_2 =0 進入 slave 端的主程式。

**Master 端**：

1) 初始化 A7105RF chip。
2) 等待 timer=2。
3) 進入 TX state，傳送封包。完成傳送後，RF chip 會自動結束 TX state，回復到 Standby state。
4) 等待 timer=12 時，進入 RX 狀態。
5) 進入 RX 狀態。
6) 如 Time>14 時，仍未收到資料，則結束 RX 狀態。變數 seq 值加 1。回到 Step 1 動作，開始下一周期時序(另一工作頻率)的傳送。
7) 如收到封包後，RF chip 會自動結束 RX state，回復到 Standby state。
8) 從 RX FIFO 讀出，並比較 PN9 code 共 64bytes，並計算 error bit 數目。
9) 變數 seq 值加 1，重新回到 Step 1 動作，重新開始下一周期時序工作。
10) 每 500ms，將所計算的 error bit 傳送至 PC。

**Slave 端：**

1) 初始化 A7105RF chip。
2) 使用 Link 通道，做頻率校正程序，調整 Master 端與 Slave 端的頻率偏差。
3) 等待 timer=0。
4) 進入 RX 狀態，等待封包收到。
5) 如 timer>3 時，仍未收到資料，則結束 RX 狀態。變數 seq 值加 1，變數 Err_HopCnt 值加 1。
6) 如 Err_HopCnt 值沒有大於 5 次，則重新回到 Step 1 動作，重新開始下一周期時序工作。
7) 如 Err_HopCnt 值大於 5 次，則清除變數 seq 值爲 0，及 Err_HopCnt 值爲 0。使用 seq=0 的工作頻率進入 RX state，等待封包收到。
8) 如收到封包後，RF chip 會自動結束 RX state，回復到 Standby state。
9) 從 RX FIFO 讀出，並比較 PN9 code 共 64bytes，計算 error bit 數目。
10) 等待 timer=12 時，進入 TX 狀態，傳送封包。完成傳送後，RF chip 會自動結束 TX state，回復到 Standby state。
11) 重新回到 Step 1 動作，重新開始下一周期時序工作。
12) 每 500ms，將所計算的 error bit 傳送至 PC。

4.2 範例程式工作基本方塊

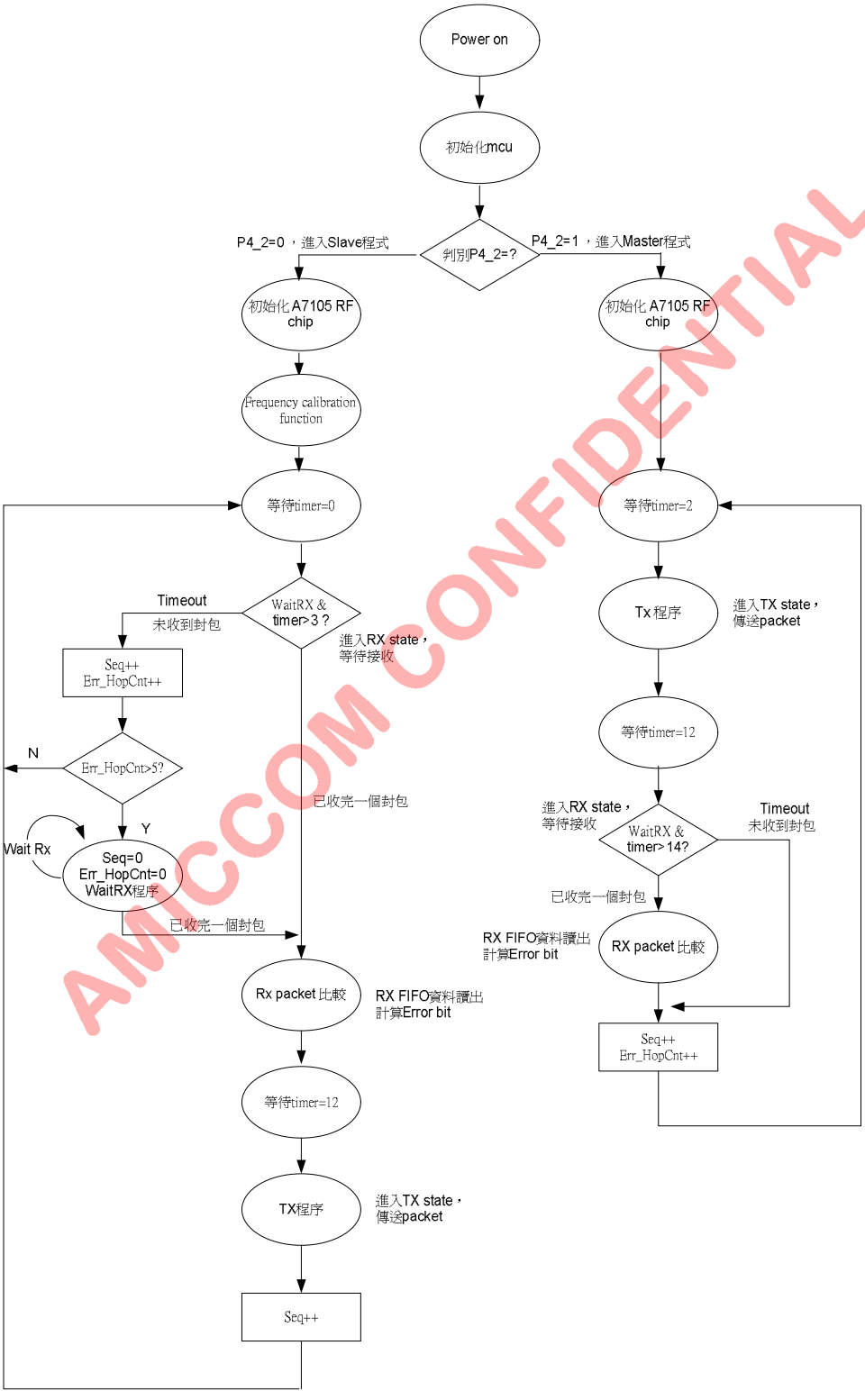


Fig4. 範例程式工作基本方塊

## 5. 程式說明

```
/*****************************************************************************
** Device:     A7105
** File:       main.c
** Author:     JPH
** Target:     Winbond W77LE58
** Tools:      ICE
** Created:    2008-11-18
** Description:
** This file is a sample code for your reference.
**
** Copyright (C) 2008 AMICCOM Corp.
**
*****************************************************************************/
#include "define.h"
#include "w77le58.h"
#include "a7105reg.h"
#include "Uti.h"
```

功能說明：Include 檔宣告，定義常數變數

| 行數 | 說明 |
|---|---|
| 14~17 | 匯入程式庫設定檔 |

```
/*****************************************************************************
** I/O Declaration
*****************************************************************************/
#define SCS          P1_0        //spi SCS
#define SCK          P1_1        //spi SCK
#define SDIO         P1_2        //spi SDIO
#define CKO          P1_3        //CKO
#define GPIO1        P1_4        //GPIO1
#define GPIO2        P1_5        //GPIO2
#define Button       P1_7        //test Button

/*****************************************************************************
** Constant Declaration
*****************************************************************************/
#define TIMEOUT      50
#define t0hrel       1000
```

功能說明：MCU 對 A7105 RF chip I/O 接腳定義，常數定義

| 行數 | 說明 |
|---|---|
| 22~28 | MCU I/O 配置 |
| 33~34 | 常數定義 |

```
 36  /*********************************************************************
37   **  Global Variable Declaration
38   *********************************************************************/
39   Uint8      data    timer;
40   Uint16     idata   RxCnt;
41   Uint32     idata   Err_ByteCnt;
42   Uint32     idata   Err_BitCnt;
43   Uint16     idata   TimerCnt0;
44   Uint8      data    *Uartptr;
45   Uint8      data    UartSendCnt;
46   Uint8      data    CmdBuf[12];
47   Uint8      xdata   tmpbuf[64];
48   Uint16     xdata   AFCBuf[12];
49   Uint8      idata   Err_Frame;
50   Uint8      data    Seq;
51   Uint8      data    Err_HopCnt;
52
53   const Uint8 code BitCount_Tab[16] = {0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4};
54   const Uint8 code ID_Tab[4]={0x54,0x75,0xC5,0x2A}; //ID code
55   const Uint8 code PN9_Tab[]=
56   {   0xFF,0x83,0xDF,0x17,0x32,0x09,0x4E,0xD1,
57       0xE7,0xCD,0x8A,0x91,0xC6,0xD5,0xC4,0xC4,
58       0x40,0x21,0x18,0x4E,0x55,0x86,0xF4,0xDC,
59       0x8A,0x15,0xA7,0xEC,0x92,0xDF,0x93,0x53,
60       0x30,0x18,0xCA,0x34,0xBF,0xA2,0xC7,0x59,
61       0x67,0x8F,0xBA,0x0D,0x6D,0xD8,0x2D,0x7D,
62       0x54,0x0A,0x57,0x97,0x70,0x39,0xD2,0x7A,
63       0xEA,0x24,0x33,0x85,0xED,0x9A,0x1D,0xE0
64   };      // This table are 64bytes PN9 pseudo random code.
```

功能說明：使用的整體變數宣告，常數變數的宣告

| 行數 | 說明 |
|---|---|
| 39~51 | 程式中使用的變數宣告 |
| 53 | BitCount_Tab 宣告 |
| 54 | ID code 宣告 |
| 55~64 | PN9 data 宣告 |

```
66  const Uint16 code A7105Config[]=
67  {
68      0x00, //RESET register,         only reset, not use on config
69      0x42, //MODE register,
70      0x00, //CALIBRATION register,only read, not use on config
71      0x3F, //FIFO1 register,
72      0x00, //FIFO2 register,
73      0x00, //FIFO register,          for fifo read/write
74      0x00, //IDDATA register,        for idcode
75      0x00, //RCOSC1 register,
76      0x00, //RCOSC2 register,
77      0x00, //RCOSC3 register,
78      0x00, //CKO register,
79      0x01, //GPIO1 register
80      0x21, //GPIO2 register,
81      0x05, //CLOCK register,
82      0x00, //DATARATE register,
83      0x50, //PLL1 register,
84      0x9E, //PLL2 register,          RFbase 2400MHz
85      0x4B, //PLL3 register,
86      0x00, //PLL4 register,
87      0x02, //PLL5 register,
88      0x16, //TX1 register,
89      0x2B, //TX2 register,
90      0x12, //DELAY1 register,
91      0x00, //DELAY2 register,
92      0x62, //RX register,
93      0x80, //RXGAIN1 register,
94      0x80, //RXGAIN2 register,
95      0x00, //RXGAIN3 register,
96      0x0A, //RXGAIN4 register,
97      0x32, //RSSI register,
98      0xC3, //ADC register,
99      0x07, //CODE1 register,
100     0x16, //CODE2 register,
101     0x00, //CODE3 register,
102     0x00, //IFCAL1 register,
103     0x00, //IFCAL2 register,    only read
104     0x00, //VCOCCAL register,
105     0x00, //VCOCAL1 register,
106     0x3B, //VCOCAL2 register,
107     0x00, //BATTERY register,
108     0x17, //TXTEST register,
109     0x47, //RXDEM1 register,
110     0x80, //RXDEM2 register,
111     0x03, //CPC register,
112     0x01, //CRYSTAL register,
113     0x45, //PLLTEST register,
114     0x18, //VCOTEST1 register,
115     0x00, //VCOTEST2 register,
116     0x01, //IFAT register,
117     0x0F, //RSCALE register,
118     0x00  //FILTERTEST
119 };
```

| 功能說明： RF chip 初始設定 | |
|---|---|
| 行數 | 說明 |
| 68~118 | RF chip 的初始設定 |

```
121  const Uint8 HopTab[]=
122  {
123      20, //2410
124      40, //2420
125      80, //2440
126      120,//2460
127      160 //2480
128  };
```

功能說明：Hopping table 宣告

| 行數 | 說明 |
|---|---|
| 123~127 | 自行定義 5 個的 channel. |

```
130  /*********************************************************************
131  **  function Declaration
132  *********************************************************************/
133  void InitTimer0(void);
134  void initUart0(void);
135  void Timer0ISR (void);
136  void Uart0Isr(void);
137  void A7105_Reset(void);
138  void A7105_WriteReg(Uint8, Uint8);
139  Uint8 A7105_ReadReg(Uint8);
140  void ByteSend(Uint8 src);
141  Uint8 ByteRead(void);
142  void A7105_WriteID(void);
143  void A7105_WriteFIFO(void);
144  void initRF(void);
145  void A7105_Config(void);
146  void A7105_Cal(void);
147  void RxPacket(void);
148  void StrobeCmd(Uint8);
149  void SetCH(Uint8);
150  void WaitBit_0(Uint8, Uint8);
151  void SelVCOBand(Uint8, Uint8);
152  void Swap(Uint8, Uint8);
153  void FrequencyCal(void);
```

功能說明：副程式檔頭宣告

| 行數 | 說明 |
|---|---|
| 133~153 | 副程式宣告 |

```
155  /*********************************************************************
156  * main loop
157  *********************************************************************/
158  void main(void)
159  {
160      //initsw
161      PMR |= 0x01; //set DME0
162
163      //initHW
164      P0 = 0xFF;
165      P1 = 0xFF;
166      P2 = 0xFF;
167      P3 = 0xFF;
168      P4 = 0x0F;
169
170      InitTimer0();
171      initUart0();
172      TR0=1;
173      EA=1;
174
175      if ((P4 & 0x04)==0x04) //if P4.2=1, master
176      {
177          initRF();
178          StrobeCmd(CMD_STBY);
179          A7105_WriteFIFO(); //write data to tx fifo
180          Seq=0;
181
182          while(1)
183          {
184              //Tx time-slot
185              while(timer != 2); //wait until timer=2
186              SetCH(HopTab[Seq]);
187              StrobeCmd(CMD_TX); //entry tx & transmit
188              while(GPIO1); //wait transmit completed
189
190              //Rx time-slot
191              while(timer !=12); //wait until timer=15
192              SetCH(HopTab[Seq]-1);
193              StrobeCmd(CMD_RX);
194              while(GPIO1 && timer <= 14);
195
196              if (timer >14)
197              {
198                  //timeout
199                  StrobeCmd(CMD_PLL);
200              }
201              else
202              {
203                  //data procedure
204                  RxPacket();
205              }
206
207              Seq++;
208              if (Seq > 4)
209                  Seq = 0;
210          }
211      }
```

```
212     else //if P4.2=0, slave
213     {
214        initRF();
215
216        //clear variable
217        RxCnt = 0;
218        Err_ByteCnt = 0;
219        Err_BitCnt = 0;
220        Seq=0;
221        Err_HopCnt=0;
222
223        //Frequency calibration with sync data
224        //at default link channel
225        SetCH(HopTab[2]-1);
226        FrequencyCal();
227        StrobeCmd(CMD_STBY);
228
229        while(1)
230        {
231           if (P1_7==0)
232           {
233              RxCnt = 0;
234              Err_BitCnt = 0;
235           }
236
237           //Rx time-slot
238           while(timer!=0);
239           SetCH(HopTab[Seq]-1);
240           StrobeCmd(CMD_RX);
241           while(GPIO1 && timer <= 3); //wait receive completed
242           if (timer > 3)
243           {
244              StrobeCmd(CMD_PLL);
245
246              Seq++;
247              if (Seq > 4)
248                 Seq = 0;
249
250              Err_HopCnt++;
251              if (Err_HopCnt > 5)
252              {
253                 Seq = 0;
254                 Err_HopCnt = 0;
255
256                 SetCH(HopTab[Seq]-1);
257                 StrobeCmd(CMD_RX);
258                 while(1)
259                 {
260                    if (GPIO1==0)
261                    {
262                       break;
263                    }
264                 }
265              }
266                 else
267                 {
268                    continue;
269                 }
270           }
```

```
271
272             timer = 2; //reSync
273             TF0 = 0; // Clear Timer0 interrupt
274             TH0 = (65536-t0hrel)>>8; // Reload Timer0 high byte,low byte
275             TL0 = 65536-t0hrel;
276
277             RxPacket();
278
279             //Tx time-slot
280             while(timer != 12);
281             SetCH(HopTab[Seq]);
282             StrobeCmd(CMD_TX);
283             while(GPIO1);
284
285             Seq++;
286             if (Seq > 4)
287                 Seq = 0;
288         }
289     }
290 }
```

功能說明：主程式 main loop。Port4_2=1，進入 master 迴圈，行數 162~188 爲 master 程式。Port4_2=0，進入 slave 迴圈，行數 239~261 爲 slave 程式。

| 行數 | 說明 |
|---|---|
| 161 | 啓用 MCUon chip data SRAM |
| 164~168 | 初始化 MCU I/O Port |
| 170 | 呼叫副程式 initTimer0，致能中斷 |
| 171 | 呼叫副程式 initUart0，初始 Uart0 |
| 172~173 | 啓動 Timer0 ，致能中斷開啓 |
| 175 | 判別 port4_2=1，進入 master 迴圈。Port4_2=0，進入 slave 迴圈。 |
| 176~211 | Master 迴圈程式 |
| 177 | 呼叫副程式 initRF，初始化 A7105 chip |
| 178 | 使用 Strobe command，進入 Standby mode 模式 |
| 179 | 呼叫副程式 A7105_WriteFIFO，將 data 寫入 TX FIFO |
| 180 | 清除變數 seq=0 |
| 185 | 等待 timer=2 |
| 186 | 呼叫副程式 SetCH ，依 HopTab 查表設置工作頻率 |
| 187 | 使用 Strobe command，進入 TX 模式，發送資料 |
| 188 | 判別 I/O GPIO1 等待是否完成資料傳送 |
| 191 | 等待 timer=12 |
| 192 | 呼叫副程式 SetCH ，依 HopTab 查表設置工作頻率 |
| 193 | 使用 Strobe command，進入 RX 模式 |
| 194 | 等待是否收妥資料或是 timer>14 |
| 196~200 | 判別是否 timer>14。如是，則使用 Strobe command，進入 PLL state |
| 202~205 | 已收妥資料，呼叫 RxPacket 副程式，從 RX FIFO 讀出資料、比對、計算 error bit 數 |
| 207~209 | 變數 seq 加 1，判別變數 seq 是否大於 4。如是，則清爲 0 |
| 213~289 | Slave 迴圈程式 |
| 214 | 呼叫副程式 initRF，初始化 A7105 chip |
| 217~221 | 清除變數 RxCnt, Err_ByteCnt, Err_BitCnt, seq, Err_HopCnt 值 |
| 225 | 設定 Link channel，做頻率校正的通道。 |
| 226 | 呼叫副程式 FrequencyCal，校正頻率偏移 |
| 227 | 使用 Strobe command，進入 Standby 模式 |
| 231~235 | 判別 MCU pin P1_7 是否爲 0。如是，則清除變數 seq, Err_HopCnt 值 |
| 238 | 等待 timer=0 |

| | |
|---|---|
| 239 | 呼叫副程式 SetCH ，依 HopTab 查表設置工作頻率 |
| 240 | 使用 Strobe command，進入 RX 模式 |
| 241 | 等待資料的收妥或是 timer>3 |
| 242 | 判別變數 timer 是否大於 3 |
| 244 | 使用 Strobe command，進入 PLL state |
| 246~248 | 變數 seq 加 1，判別變數 seq 是否大於 4。如是，則清爲 0 |
| 250 | 變數 Err_HopCnt 加 1 |
| 251 | 判別 Err_HpoCnt 是否大於 5 |
| 253~254 | 清除變數 seq=0, Err_HopCnt=0 |
| 256 | 呼叫副程式 SetCH ，設置工作頻率 |
| 257 | 使用 Strobe command，進入 RX 模式 |
| 258~264 | 等待資料進入 |
| 268 | 變數 Err_HopCnt 值沒有大於 5 次時，則回到 223 行，重新另一次周期的工作 |
| 272 | 設置變數 timer=2，重新同步 |
| 273~275 | 清除 TF0 旗標，重新設置 TH0, TL0 值 |
| 277 | 已收妥資料，呼叫 RxPacket 副程式，從 RX FIFO 讀出資料、比對、計算 error bit 數 |
| 280 | 等待 timer=12 |
| 281 | 呼叫副程式 SetCH，設置工作頻率 |
| 282 | 使用 Strobe command，進入 TX 模式，發送資料 |
| 283 | 等待是否完成資料傳送 |
| 285~287 | 變數 seq 加 1，判別變數 seq 是否大於 4。如是，則清爲 0 |

```
292  /*********************************************************************
293  **  init Timer0
294  *********************************************************************/
295  void InitTimer0(void)
296  {
297      TR0 = 0;
298      TMOD =(TMOD & 0xF0)|0x01; //timer0 mode=1
299      TH0 = (65536-t0hrel)>>8; // Reload Timer0 high byte,low byte
300      TL0 = 65536-t0hrel;
301      TF0 = 0; // Clear any pending Timer0 interrupts
302      ET0 = 1; // Enable Timer0 interrupt
303  }
```

功能說明：初始化 Timer0 程序

| 行數 | 說明 |
|---|---|
| 297 | 關閉 Timer0 計時動作 |
| 298 | 設置 Timer0 在 mode 1 模式 |
| 299~300 | 設置 TH0,TL0 的初始值 |
| 301 | 清除 Timer0 中斷旗標 |
| 302 | 致能 Timer0 中斷 |

```
305  /*********************************************************************
306  **  Timer0ISR
307  *********************************************************************/
308  void Timer0ISR (void) interrupt 1
309  {
310     TF0 = 0; // Clear Timer0 interrupt
311     TH0 = (65536-t0hrel)>>8; // Reload Timer0 high byte,low byte
312     TL0 = 65536-t0hrel;
313
314     timer++;
315     if (timer>=20)
316     {
317        timer=0;
318        P3_5= ~P3_5;
319     }
320
321     TimerCnt0++;
322     if (TimerCnt0 == 500)
323     {
324        TimerCnt0 = 0;
325        CmdBuf[0] = 0xF1;
326
327        memcpy(&CmdBuf[1], &RxCnt, 2);
328        memcpy(&CmdBuf[3], &Err_ByteCnt, 4);
329        memcpy(&CmdBuf[7], &Err_BitCnt, 4);
330        memcpy(&CmdBuf[11], &Err_Frame, 1);
331
332        UartSendCnt = 12;
333        Uartptr =& CmdBuf[0];
334        SBUF = CmdBuf[0];
335     }
336  }
```

功能說明：初始化 Timer0 的中斷副程式

| 行數 | 說明 |
|---|---|
| 310~312 | 清除 Timer0 中斷旗標，設置 TH0,TL0 的啓始值 |
| 314 | 變數 timer 加 1 |
| 315~319 | 判別變數 timer 是否等於 20ms。如是，清除變數 timer=0，pin P3_5 信號反向. |
| 321 | 變數 TimerCnt0 加 1 |
| 322 | 判別變數 TimerCnt0 是否等於 500(即 500ms) |
| 324 | 清除變數 TimerCnt0 |
| 325 | CmdBuf[0]設置 0xF1 爲傳送啓始位元識別碼 |
| 327 | CmdBuf[1] 、CmdBuf[1]設置變數 RxCnt 的值 |
| 328 | CmdBuf[3] 、CmdBuf[4]、CmdBuf[5] 、CmdBuf[6]設置變數 Err_ByteCnt 的值 |
| 329 | CmdBuf[7] 、CmdBuf[8]、CmdBuf[9] 、CmdBuf[10]設置變數 Err_BitCn 的值 |
| 330 | CmdBuf[11]設置變數 Err_Frame 的值 |
| 332 | 設置變數UartSendCnt=12 |
| 333 | 設置指標變數 Uartptr 指到變數 CmdBuf[0]的啓始位址 |
| 334 | 傳送 SBUF 至 PC |

```
338  /*******************************************************************************
339  **  Init Uart0
340  *******************************************************************************/
341  void initUart0(void)
342  {
343      TH1 = 0xFD; //BaudRate 9600;
344      TL1 = 0xFD;
345      SCON = 0x40;
346      TMOD = (TMOD & 0x0F) | 0x20;
347      REN = 1;
348      TR1 = 1;
349      ES = 1;
350  }
```

功能說明：初始化 Uart0 的程序

| 行數 | 說明 |
|---|---|
| 343~345 | 初始 TL1,TH1,SCON1 值，設置爲 9600bps @xtal=11.0592MHz |
| 346 | 設置 Timer1 爲 mode 2 |
| 347~349 | 設置 REN,TR1,ES 爲 1，啓用 Uart0 的功能 |

```
352  /*******************************************************************************
353  **  Uart0 ISR
354  *******************************************************************************/
355  void Uart0Isr(void) interrupt 4 using 3
356  {
357      if (TI==1)
358      {
359          TI=0;
360          UartSendCnt--;
361          if(UartSendCnt !=0)
362          {
363              Uartptr++;
364              SBUF = *Uartptr;
365          }
366      }
367  }
```

功能說明：初始化 uart0 的中斷副程式

| 行數 | 說明 |
|---|---|
| 357 | 判別 TI 旗標是否爲 Uart 已傳送完成 1byte |
| 359 | 清除 TI 旗標 |
| 360 | 變數 UartSendCnt 減 1 |
| 361 | 判別變數 UartSendCnt 是否爲 0。如不爲 0，則繼續傳送下一個資料 |
| 363~364 | 指標變數 Uartptr 加 1，並將其位址的資料，使用 Uart0 送至 PC |

```
369  /*******************************************************************************
370  **  Reset_RF
371  *******************************************************************************/
372  void A7105_Reset(void)
373  {
374      A7105_WriteReg(MODE_REG, 0x00); //reset RF chip
375  }
```

功能說明：A7105 RF chip Reset 程序

| 行數 | 說明 |
|---|---|
| 374 | 對 register 位址 0 ，寫入 0x00，重置 RF chip。 |

```
377  /*********************************************************************
378  **  WriteID
379  *********************************************************************/
380  void A7105_WriteID(void)
381  {
382      Uint8 i;
383      Uint8 d1,d2,d3,d4;
384      Uint8 addr;
385
386      addr = IDCODE_REG; //send address 0x06, bit cmd=0, r/w=0
387      SCS = 0;
388      ByteSend(addr);
389      for (i=0; i < 4; i++)
390          ByteSend(ID_Tab[i]);
391      SCS = 1;
392
393      //for check
394      addr = IDCODE_REG | 0x40; //send address 0x06, bit cmd=0, r/w=1
395      SCS=0;
396      ByteSend(addr);
397      d1=ByteRead();
398      d2=ByteRead();
399      d3=ByteRead();
400      d4=ByteRead();
401      SCS=1;
402  }
```

功能說明：寫入 ID 的程序。

| 行數 | 說明 |
|---|---|
| 386 | 計算變數 addr 的值 |
| 387 | SCS=0，設置控制暫存器讀寫功能 |
| 388~390 | 寫入 ID 控制暫存器的位址，及 4 bytes 的 ID code |
| 391 | SCS=1，清除 SPI 讀寫功能 |
| 394 | 計算讀出 ID code 的變數 addr 值 |
| 395 | SCS=0，設置控制暫存器讀寫功能 |
| 396~400 | 讀出 ID code |
| 401 | SCS=1，清除控制暫存器讀寫功能 |

```
/*********************************************************************
**  A7105_WriteReg
*********************************************************************/
void A7105_WriteReg(Uint8 addr, Uint8 dataByte)
{
    Uint8 i;

    SCS = 0;
    addr |= 0x00; //bit cmd=0,r/w=0
    for(i = 0; i < 8; i++)
    {
        if(addr & 0x80)
            SDIO = 1;
        else
            SDIO = 0;

        SCK = 1;
        _nop_();
        SCK = 0;
        addr = addr << 1;
    }
    _nop_();

    //send data byte
    for(i = 0; i < 8; i++)
    {
        if(dataByte & 0x80)
            SDIO = 1;
        else
            SDIO = 0;

        SCK = 1;
        _nop_();
        SCK = 0;
        dataByte = dataByte<< 1;
    }
    SCS = 1;
}
```

功能說明：對 A7105 控制暫存器(Control Register)寫入動作

| 行數 | 說明 |
|---|---|
| 411 | SCS=0，致能控制暫存器讀寫功能 |
| 412 | 將 address Or 寫入控制暫存器命令。 |
| 413~424 | 寫入 address 的程序 |
| 428~439 | 寫入 data byte 的程序 |
| 440 | SCS=1，清除控制暫存器讀寫功能 |

```
443  /*********************************************************************
444  **  A7105_ReadReg
445  *********************************************************************/
446  Uint8 A7105_ReadReg(Uint8 addr)
447  {
448      Uint8 i;
449      Uint8 tmp;
450
451      SCS = 0;
452      addr |= 0x40; //bit cmd=0,r/w=1
453      for(i = 0; i < 8; i++)
454      {
455
456          if(addr & 0x80)
457              SDIO = 1;
458          else
459              SDIO = 0;
460
461          _nop_();
462          SCK = 1;
463          _nop_();
464          SCK = 0;
465
466          addr = addr << 1;
467      }
468
469      _nop_();
470      SDIO = 1;
471
472      //read data
473      for(i = 0; i < 8; i++)
474      {
475          if(SDIO)
476              tmp = (tmp << 1) | 0x01;
477          else
478              tmp = tmp << 1;
479
480          SCK = 1;
481          _nop_();
482          SCK = 0;
483      }
484      SCS = 1;
485      return tmp;
486  }
```

功能說明：A7105 控制暫存器(Control Register)讀出動作

| 行數 | 說明 |
|---|---|
| 451 | SCS=0，致能控制暫存器讀寫功能 |
| 452 | 將 address Or 讀出控制暫存器命令。 |
| 453~467 | 寫入 address 的程序 |
| 470 | 設置 SDIO 爲輸出模式 |
| 473~483 | 讀出資料 |
| 484 | SCS=0，清除控制暫存器讀寫功能 |
| 485 | 回傳 1 byte 的讀值 |

```
/*******************************************************************************
**  ByteSend
*******************************************************************************/
void ByteSend(Uint8 src)
{
    Uint8 i;

    for(i = 0; i < 8; i++)
    {
        if(src & 0x80)
            SDIO = 1;
        else
            SDIO = 0;

        _nop_();
        SCK = 1;
        _nop_();
        SCK = 0;
        src = src << 1;
    }
}
```

| 功能說明：寫入 1 byte 的程序 | |
|---|---|
| 行數 | 說明 |
| 495~507 | 寫入 1 個 byte 的程序 |

```
/*******************************************************************************
**  ByteRead
*******************************************************************************/
Uint8 ByteRead(void)
{
    Uint8 i,tmp;

    SDIO = 1; //sdio pull high
    for(i = 0; i < 8; i++)
    {
        if(SDIO)
            tmp = (tmp << 1) | 0x01;
        else
            tmp = tmp << 1;

        SCK = 1;
        _nop_();
        SCK = 0;
    }
    return tmp;
}
```

| 功能說明：讀出 1byte 的程序 | |
|---|---|
| 行數 | 說明 |
| 517~528 | 讀出 1 個 byte 的程序 |
| 529 | 返回 8 bit 的讀值 |

```
532  /*********************************************************************
533  **  Send4Bit
534  *********************************************************************/
535  void Send4Bit(Uint8 src)
536  {
537      Uint8 i;
538
539      for(i = 0; i < 4; i++)
540      {
541          if(src & 0x80)
542              SDIO = 1;
543          else
544              SDIO = 0;
545
546          _nop_();
547          SCK = 1;
548          _nop_();
549          SCK = 0;
550          src = src << 1;
551      }
552  }
```

功能說明：寫入 4 bit 的程序

| 行數 | 說明 |
|---|---|
| 539~550 | 寫入 1 個 byte 的程序 |

```
554  /*********************************************************************
555  ** SetCH
556  *********************************************************************/
557  void SetCH(Uint8 ch)
558  {
559      A7105_WriteReg(PLL1_REG, ch);
560  }
```

功能說明：設置頻道的程序

| 行數 | 說明 |
|---|---|
| 559 | 呼叫副程式 A7105_WriteReg，對 PLL1 控制暫存器寫入工作頻道值。 |

```
562  /*********************************************************************
563  ** initRF
564  *********************************************************************/
565  void initRF(void)
566  {
567      //init io pin
568      SCS = 1;
569      SCK = 0;
570      SDIO = 1;
571      CKO = 1;
572      GPIO1 = 1;
573      GPIO2 = 1;
574
575      A7105_Reset(); //reset A7105 RF chip
576      A7105_WriteID(); //write ID code
577      A7105_Config(); //config A7105 chip
578      A7105_Cal(); //calibration IF,vco, vcoc
579  }
```

功能說明：初始化 Master 端的 RF chip

| 行數 | 說明 |
|---|---|
| 5683~573 | 設置 RF chip 介面 I/O 初始值 |
| 575 | 呼叫副程式 A7105_Reset，重置 RF chip |
| 576 | 呼叫副程式 A7105_WriteID，寫入 ID code 4ytes |
| 577 | 呼叫副程式 A7105_Config，初始 RF 端的控制暫存器 |
| 578 | 呼叫副程式 A7105_Cal，IF, VCO, VCO current 的校準程序 |



```
581  /*********************************************************************
582  ** A7105_WriteFIFO
583  *********************************************************************/
584  void A7105_WriteFIFO(void)
585  {
586      Uint8 i;
587      Uint8 cmd;
588
589      cmd = FIFO_REG; //send address 0x05, bit cmd=0, r/w=0
590      SCS=0;
591      ByteSend(cmd);
592      for(i=0; i <64; i++)
593          ByteSend(PN9_Tab[i]);
594      SCS=1;
595  }
```

功能說明：Tx FIFO 寫入資料的程序

| 行數 | 說明 |
|---|---|
| 589 | 將 FIFO 控制暫存器位址與 cmd bit, r/w bit 作運算，寫入 TX FIFO 控制暫存器命令 |
| 590 | SCS=0，致能控制暫存器讀寫功能 |
| 591 | 送出 TX FIFO 寫入命令 |
| 592~593 | 寫入 64 bytes 的資料 |
| 594 | SCS=1，清除控制暫存器讀寫功能 |

```
597  /*********************************************************************
598  ** Strobe Command
599  *********************************************************************/
600  void StrobeCmd(Uint8 cmd)
601  {
602      SCS = 0;
603      Send4Bit(cmd);
604      SCS = 1;
605  }
```

功能說明：Strobe 命令寫入的程序。

| 行數 | 說明 |
|---|---|
| 602 | SCS=0，致能控制暫存器讀寫功能 |
| 603 | 呼叫副程式 Send4Bit，將控制指令寫入 |
| 604 | SCS=1，清除控制暫存器讀寫功能 |

```
607  /*********************************************************************
608  ** RxPacket
609  *********************************************************************/
610  void RxPacket(void)
611  {
612      Uint8 i;
613      Uint8 recv;
614      Uint8 tmp;
615      Uint8 cmd;
616
617      RxCnt++;
618      cmd = FIFO_REG | 0x40; //address 0x05, bit cmd=0, r/w=1
619
620      SCS=0;
621      ByteSend(cmd);
622      for(i=0; i <64; i++)
623      {
624          recv = ByteRead();
625          tmpbuf[i]=recv;
626          if((recv ^ PN9_Tab[i])!=0)
627          {
628              tmp = recv ^ PN9_Tab[i];
629              Err_BitCnt += (BitCount_Tab[tmp>>4] + BitCount_Tab[tmp & 0x0F]);
630          }
631      }
632      SCS=1;
633  }
```

功能說明：從 RX  FIFO 讀出資料，比對資料的程序

| 行數 | 說明 |
|---|---|
| 617 | 變數 RxCnt 加 1 |
| 618 | 將 FIFO 控制暫存器位址與 cmd bit, r/w bit 作運算，讀出 RX FIFO 控制暫存器命令 |
| 620 | SCS=0，致能控制暫存器讀寫功能 |
| 621 | 呼叫副程式 ByteSend，送出控制命令 |
| 622~631 | 讀出 data ，比較 data 的正確性，計算出 error bit |
| 632 | SCS=1，清除控制暫存器讀寫功能 |

```
/*********************************************************************
** Err_State
*********************************************************************/
void Err_State(void)
{
    //ERR display
    //Error Proc...
    //...
}
```

功能說明：Error State 處理程序

| | |
|---|---|
| 640~641 | Error 處理程序，使用者自訂 |

```
/*********************************************************************
**  WaitBit_0
*********************************************************************/
void WaitBit_0(Uint8 reg, Uint8 nbit)
{
    do {
    } while(A7105_ReadReg(reg) & nbit);
}
```

功能說明：WaitBit 程序

| | |
|---|---|
| 650~651 | 呼叫 A7105_ReadReg 副程式，讀取該暫存器中，判定某 bit 值是否清爲 0. |

```
/*********************************************************************
** SelVCOBand
*********************************************************************/
void SelVCOBand(Uint8 vb1, Uint8 vb2)
{
    Uint8 diff1,diff2;

    if (vb1>=4)
        diff1 = vb1-4;
    else
        diff1 = 4-vb1;

    if (vb2>=4)
        diff2 = vb2-4;
    else
        diff2 = 4-vb2;

    if (diff1 == diff2 || diff1 > diff2)
        A7105_WriteReg(VCOCAL1_REG, (vb1 | 0x08));//manual setting vb1 value
    else
        A7105_WriteReg(VCOCAL1_REG, (vb2 | 0x08));//manual setting vb2 value
}
```

功能說明：Select VCOBand 處理程序

| | |
|---|---|
| 661~662 | 決定變數 vb1 值與 band 4 的差值. |
| 666~667 | 決定變數 vb2 值與 band 4 的差值. |
| 671~674 | 判別 diff1 與 diff2 值是否相等或是找出與 band 4 相差較大的 band 值，並將該 band 值以手動設定 vco band 值. |

```
677  /*********************************************************************
678  ** calibration
679  *********************************************************************/
680  void A7105_Cal(void)
681  {
682      Uint8 tmp;
683      Uint8 fb,fbcf;
684      Uint8 vb1,vbcf1,dvt1;
685      Uint8 vb2,vbcf2,dvt2;
686
687      StrobeCmd(CMD_STBY); //calibration @STB state
688
689      //calibration IF procedure
690      A7105_WriteReg(CALIBRATION_REG, 0x01);
691      WaitBit_0(CALIBRATION_REG, 0x01);
692
693      //for check
694      tmp = A7105_ReadReg(IFCAL1_REG);
695      fb = tmp & 0x0F;
696      fbcf = (tmp >>4) & 0x01;
697
698      if (fbcf ==1)
699      {
700          Err_State();
701          while(1);
702      }
703
704      //calibration vco procedure
705      A7105_WriteReg(VCOCCAL_REG, 0x13); //manual VCOC=3
706      A7105_WriteReg(VCOCAL2_REG, 0x3B); //VTL=3, VTH=7
707
708      SetCH(0); //setting 2400MHz
709      A7105_WriteReg(CALIBRATION_REG, 0x02);
710      WaitBit_0(CALIBRATION_REG, 0x02);
711
712      tmp = A7105_ReadReg(VCOCAL1_REG);
713      vb1 = tmp & 0x07;
714      vbcf1 = (tmp >> 3) & 0x01;
715      dvt1 = (tmp >> 4) & 0x03;
716
717      SetCH(160); //setting 2480MHz
718      A7105_WriteReg(CALIBRATION_REG, 0x02);
719      WaitBit_0(CALIBRATION_REG, 0x02);
720
721      tmp = A7105_ReadReg(VCOCAL1_REG);
722      vb2 = tmp & 0x07;
723      vbcf2 = (tmp >> 3) & 0x01;
724      dvt2 = (tmp >> 4) & 0x03;
725
726      SelVCOBand(vb1, vb2);
727
728      if (vbcf1==1 && vbcf2==1)
729      {
730          Err_State();
731          while(1);
732      }
733  }
```

功能說明：IF, VCO, VCO current 校準程序

| 行數 | 說明 |
|---|---|
| 687 | 使用 Strobe 命令，設置 RF chip 進入 Standby state |
| 690 | 設置 calibration control register 中 bit FBC=1。 |

| | |
|---|---|
| 691 | 讀出 calibration control register，並判別 bit FBC 是否爲 0。如爲 0，則跳出等待迴圈 |
| 694~696 | 讀出 IF Calibration I 控制暫存器，並檢示其值 |
| 698~702 | 判別 fbcf 旗標是否爲 0。如不爲 0，則判定 RF chip 爲 fail。 |
| 705 | 設置 VCO current Band 爲 3。 |
| 706 | 設置 VTH=7, VTL=3。 |
| 708 | 設置頻率在 2400MHz。 |
| 709 | 設置 calibration control register 中 bit VBC=1。 |
| 710 | 讀出 calibration control register，並判別 bit VBC 是否爲 0。如爲 0，則跳出等待迴圈 |
| 712~715 | 讀出 IF Calibration I 控制暫存器，並檢示其值 |
| 717 | 設置頻率在 2480MHz。 |
| 718 | 設置 calibration control register 中 bit VBC=1。 |
| 719 | 讀出 calibration control register，並判別 bit VBC 是否爲 0。如爲 0，則跳出等待迴圈 |
| 721~724 | 讀出 IF Calibration I 控制暫存器，並檢示其值 |
| 726 | 呼叫副程式 SelVCOBand，設置最佳 VCO band 值 |
| 728~732 | 判別 vbcf1,vbcf2 是否爲 1。如爲 1，則判定 fail. |

```
735  /*********************************************************************
736  ** A7105_Config
737  *********************************************************************/
738  void A7105_Config(void)
739  {
740      Uint8 i;
741
742      //0x00 mode register, for reset
743      //0x05 fifo data register
744      //0x06 id code register
745      //0x23 IF calibration II, only read
746      //0x32 filter test register
747
748      for (i=0x01; i<=0x04; i++)
749          A7105_WriteReg(i, A7105Config[i]);
750
751      for (i=0x07; i<=0x22; i++)
752          A7105_WriteReg(i, A7105Config[i]);
753
754      for (i=0x24; i<=0x31; i++)
755          A7105_WriteReg(i, A7105Config[i]);
756  }
```

功能說明：初始 RF chip Master 端的程序

| 行數 | 說明 |
|---|---|
| 748~749 | 呼叫副程式 A7105_WriteReg，寫入控制暫存器位址 0x01~0x04 |
| 751~752 | 呼叫副程式 A7105_WriteReg，寫入控制暫存器位址 0x07~0x22 |
| 754~755 | 呼叫副程式 A7105_WriteReg，寫入控制暫存器位址 0x24~0x31 |

```
758  /*********************************************************************
759  ** Swap
760  *********************************************************************/
761  void Swap(Uint8 idx1, Uint8 idx2)
762  {
763      Uint16 tmp;
764
765      tmp= AFCBuf[idx1];
766      AFCBuf[idx1] = AFCBuf[idx2];
767      AFCBuf[idx2]= tmp;
768  }
```

功能說明：二個暫存器值互換的程序

| 行數 | 說明 |
|---|---|
| 765~767 | 將二個暫存器值互換 |

```
770  /*********************************************************************
771  ** FrequencyCal
772  *********************************************************************/
773  void FrequencyCal(void)
774  {
775      Uint8 tmp_h, tmp_l;
776      Uint32 tmp;
777      Uint16 tmp1,tmp2;
778      Uint8 i,j;
779      Uint8 cnt;
780      Uint8 flag_1n,flag_2n;
781      Uint8 pll3,pll4,pll5;
782      Uint16 offsetValue,pll;
783      Uint8 sign;
784
785      //Step1. enable AFC bit
786      A7105_WriteReg(RX_REG, 0x72);
787
788      //Step2. receive data with sync
789      cnt=0;
790      while(1)
791      {
792          StrobeCmd(CMD_RX);
793          while(GPIO1);
794
795          cnt++;
796          if (cnt>3)//ignore AFCcnt 1~3
797          {
798              tmp_h = A7105_ReadReg(PLL4_REG) & 0x7F;
799              tmp_l = A7105_ReadReg(PLL5_REG);
800              tmp1 = (Uint16)tmp_h * 256 + (Uint16)tmp_l;
801              AFCBuf[cnt-4]=tmp1;
802          }
803
804          if (cnt>=15)
805              break;
806
807          Delay1ms(1);
808      }
```

```
809
810     //Step3. bubble sort
811     for(i = 0; i < 12-1; i++)
812     {
813         for(j = i+1; j < 12; j++)
814         {
815             if(AFCBuf[i] & 0x4000)
816             {
817                 flag_1n=1;
818                 tmp1= ~AFCBuf[i] & 0x7FFF;
819             }
820             else
821             {
822                 flag_1n=0;
823                 tmp1= AFCBuf[i];
824             }
825
826             if(AFCBuf[j] & 0x4000)
827             {
828                 flag_2n=1;
829                 tmp2= ~AFCBuf[j] & 0x7FFF;
830             }
831             else
832             {
833                 flag_2n=0;
834                 tmp2= AFCBuf[j];
835             }
836
837             if (flag_1n==0 && flag_2n==0)
838             {
839                 if (tmp1 > tmp2)
840                     Swap(i, j);
841             }
842
843             if(flag_1n==1 && flag_2n==1)
844             {
845                 if (tmp1 < tmp2)
846                     Swap(i, j);
847             }
848
849             if (flag_1n==0 && flag_2n==1)
850                 Swap(i, j);
851         }
852     }
853
854     //Step4. ignore AFCBuf[0~1], AFCBuf[10~11], average AFCBuf[2~9]
855     //Plus, sign operation
856     tmp=0;
857     for(i=2; i<=9; i++)
858     {
859         if (AFCBuf[i] & 0x4000)
860             tmp = tmp + (AFCBuf[i] & 0x3FFF);
861         else
862             tmp = tmp + AFCBuf[i] + 0x4000;
863     }
```

```
864
865     tmp = tmp /8;
866     if (tmp & 0x4000)
867         offsetValue = tmp & 0x3FFF;
868     else
869         offsetValue = tmp | 0x4000;
870
871     //Step5. disable AFC bit
872     A7105_WriteReg(RX_REG, 0x62);
873
874     //Step6. update PLL register
875     pll3 = A7105Config[PLL3_REG];
876     pll4 = A7105Config[PLL4_REG];
877     pll5 = A7105Config[PLL5_REG];
878
879     pll = (Uint16)pll4*8 + (Uint16)pll5;
880
881     if (offsetValue & 0x4000)
882     {
883         sign=1;
884         offsetValue = (~offsetValue + 1) & 0x7FFF;
885     }
886     else
887     {
888         sign=0;
889     }
890
891     if (sign)
892     {
893         pll = pll - offsetValue;
894         if (CY)
895             A7105_WriteReg(PLL3_REG, pll3-1);
896
897         A7105_WriteReg(PLL4_REG, pll>>8);
898         A7105_WriteReg(PLL5_REG, pll);
899     }
900     else
901     {
902         pll = pll + offsetValue;
903         if (CY)
904             A7105_WriteReg(PLL3_REG, pll3+1);
905
906         A7105_WriteReg(PLL4_REG, pll>>8);
907         A7105_WriteReg(PLL5_REG, pll);
908     }
909 }
```

功能說明：頻率偏移修正程序
** a) 抓取 15 次 Master 端 TX 頻率偏差值。
** b) 刪除前 3 次偏差值。
** c) 將偏移值做大小排序,刪除最大、最小偏差值各二筆。
** d) 計算 8 次平均偏差值。
** e)修正設定 Slave 端 PLL control register 的值。

| 行數 | 說明 |
|---|---|
| 786 | 開啓 bit AFC=1 |
| 789 | 清除變數 cnt =0 |
| 790~808 | 接收 15 次，在有 RX sync 動作時，記錄其 compensate 的值 |
| 792 | 使用 Strobe command，進入 Rx mode 模式 |
| 793 | 等待 GIO1 爲 0 |
| 795 | 變數 cnt 加 1 |

| | |
|---|---|
| 796 | 判斷變數 cnt 是否大於 3。如不是，則忽略不運算。 |
| 798 | 讀取控制暫存器 PLL4 的值，將值存入變數 temp_h |
| 799 | 讀取控制暫存器 PLL5 的值，將值存入變數 temp_l |
| 800 | 計算 tmp1 值 |
| 801 | 將所計算的值，存入 AFCBuf[xx]中。 |
| 804 | 判斷變數 cnt 是否爲 15。如果是，則離開這迴圈。 |
| 807 | 呼叫副程式 Delay1ms，延遲 1ms。 |
| 811~852 | 對 AFCBuf[]做大小排序。 |
| 856 | 清除變數 tmp=0 |
| 857~863 | 對 AFCBuf[2]~AFCBuf[9]做有號數的相加 |
| 865~869 | 變數 tmp 除 8，做平均值，計算出頻率偏移值。 |
| 872 | 關閉 bit AFC=0 |
| 875 | 讀取初始 PLL3 的值，存入變數 pll3 |
| 876 | 讀取初始 PLL4 的值，存入變數 pll4 |
| 877 | 讀取初始 PLL5 的值，存入變數 pll5 |
| 879 | 計算變數 pll 值，即 RF chip 中的 FP 值。 |
| 881~889 | 判斷變數 offsetValue 的值是正向偏移或是負向偏移。 |
| 892~899 | 如 offsetValue 爲負向偏移，則做控制暫存器 PLL3, PLL4,PLL5 值的修正 |
| 901~908 | 如 offsetValue 爲正向偏移，則做控制暫存器 PLL3, PLL4,PLL5 值的修正 |