Application Notes

EM198810 RF Transceiver

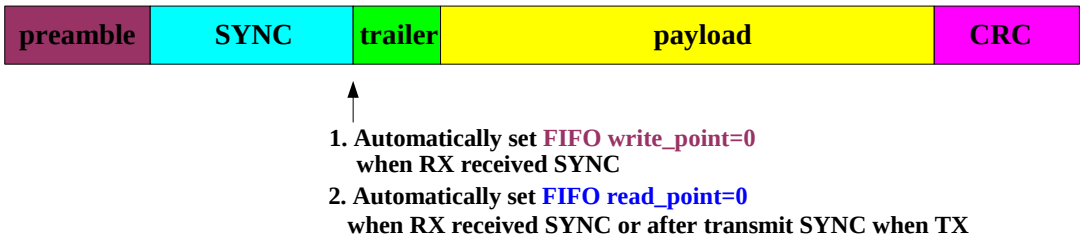
- Framer and Digital Interface Control

Introduction

This Application Note describes the detailed packet handling details pertaining to the Elan EM198810.

For the recommended schematic diagram, please refer to the EM198810 Data Sheet, or consult your Elan representative for assistance.

Packet define and FIFO point clear



- Figure 1 -

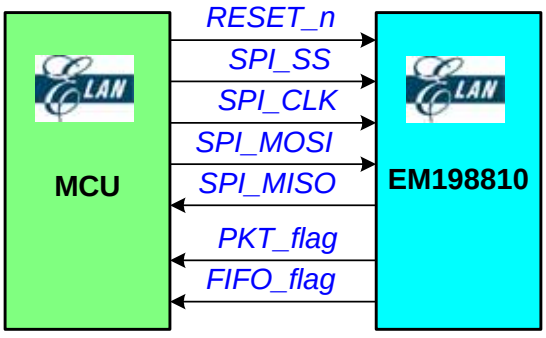
- **Preamble: 1 ~ 8 bytes programmable**
- **SYNC: 32/48/64 bits programmable as device syncword**
- **Trailer: 4~18 bits programmable**
- **Payload: TX/RX data, there are 4 data types: raw data, 8_10 bits, Manchester, interleave, with FEC option**
- **CRC: 16 bit CRC is option**

Note: For transmit, it is required to clear FIFO write point before application writes in data via access reg82[15].

Digital Interface

It is a very simple interface with MCU for application, consisting of SPI interface plus two handshake signals.

The EM198810 SPI can supports slave mode only. It supports two SPI standard formats (CKPHA=0 and CKPHA=1). Please refers to EM198810 datasheet for detail description of CKPHA and the associated data formats.



Pin	Description

SPI_CLK	**SPI clock input**
SPI_SS	**SPI slave select input**
SPI_MOSI	**SPI data in**
SPI_MISO	**SPI data out**
PKT_FLAG	**Packet TX/RX flag**
FIFO_FLAG	**FIFO full/empty**
RESET_n	**Reset input, active low**

- Table 1 -

SPI Timing Diagram

Here is the description of the SPI signals timing diagram. For illustration purpose, CKPHA=0 is shown (SPI_MOSI data clocked in on rising edge of SPI_CLK).

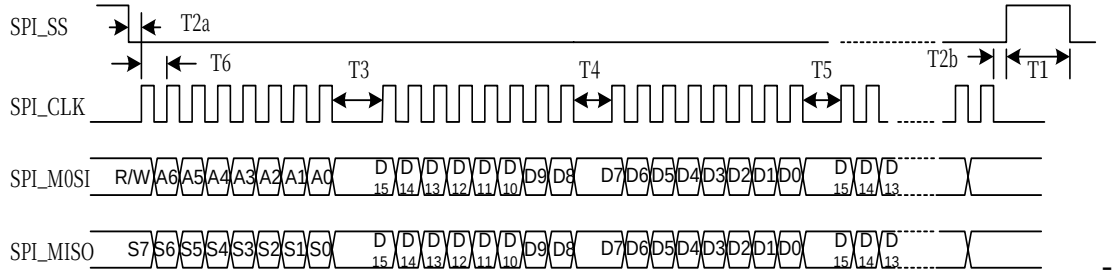


Figure 2 -

name	Min.	Typ.	Max.	Description
T1	250ns			**Chapter 1 Interval between two SPI accesses**
T2a, T2b	41.5ns			**Relationship between SPI_SS & SPI_CLK**
T3	* (1)			**Interval time between two address and data**
T4	* (2)			**Interval time between high byte and low byte data**
T5	* (3)			**Interval time between two register data**
T6	83ns			**SPI_CLK period**

- Table 2 -

* Notes:

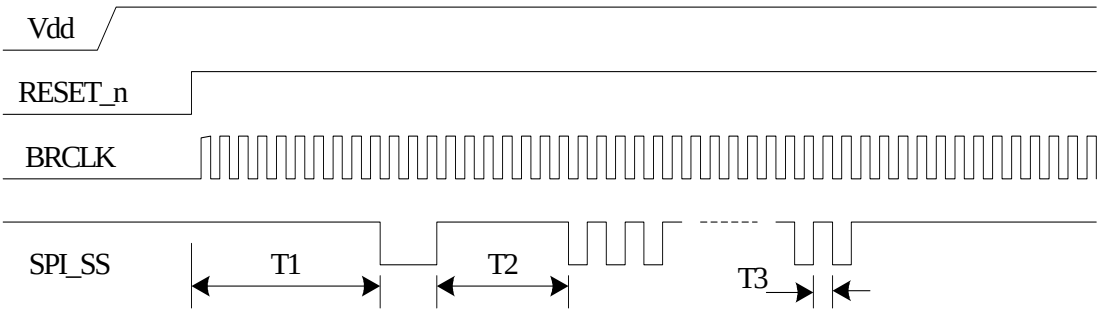
(1) When MCU reads register 0x00~0x1f, at least 3us wait time is required for framer to decode the address and get register (0x00~0x1f) value. When MCU reads FIFO data, at least 450ns wait time is required for framer to get correct FIFO read point. Otherwise, T3min = 41.5ns.

(2) When MCU reads FIFO data, at least 450ns wait time is required for framer to get correct read point. Otherwise, T4min = 41.5ns.

(3) When MCU writes register 0x00~0x1f, at least 3us wait time is required for framer to program register(0x00~0x1f) used by internal BuleRF interface Dbus. If read FIFO data, at least 450ns wait time is required. Otherwise, T5min = 41.5ns.

Power on and register initialize sequence

Figure 3 shows the timing diagram of power on sequence after Vdd is ready.



- Figure 3 -

Power on and register on sequence

1. After Vdd power is ready, make sure have valid reset on pin RESET_n, which is low active. After RESET_n =1, BRCLK will be running at 12Mhz clock.
2. Wait T1 (1~5ms) for crystal to stabilize, then MCU can perform framer register initialization (reg48~reg57). The initialization data can be written in continuously or one by one.
3. After framer register initialization, wait T2 (2ms) for automatic RFIC calibration, then performs RFIC initialization. For RFIC register initialization, the recommended way is to write in data one by one with 3us interval (T3).
4. After RFIC register initialization, wait for additional 2ms, then EM198810 is ready to transmit or receive.

Note: Framer register (reg48~reg57) initialization needs to be done prior to RFIC register (reg-~reg31) initialization.

Enter Sleep and Wake Up

When MCU writes EM198810 register to enter sleep mode and pulls SPI_SS back to high, EM198810 will enter sleep state where the current consumption is extremely low. When SPI_SS is pulled low, EM198810 will automatically wake up from sleep state. MCU needs to keep SPI_SS low a certain time (the time required for RFIC crystal to be stabilized) before driving SPI_CLK and SPI data.

Start TX/RX and stop TX/RX define

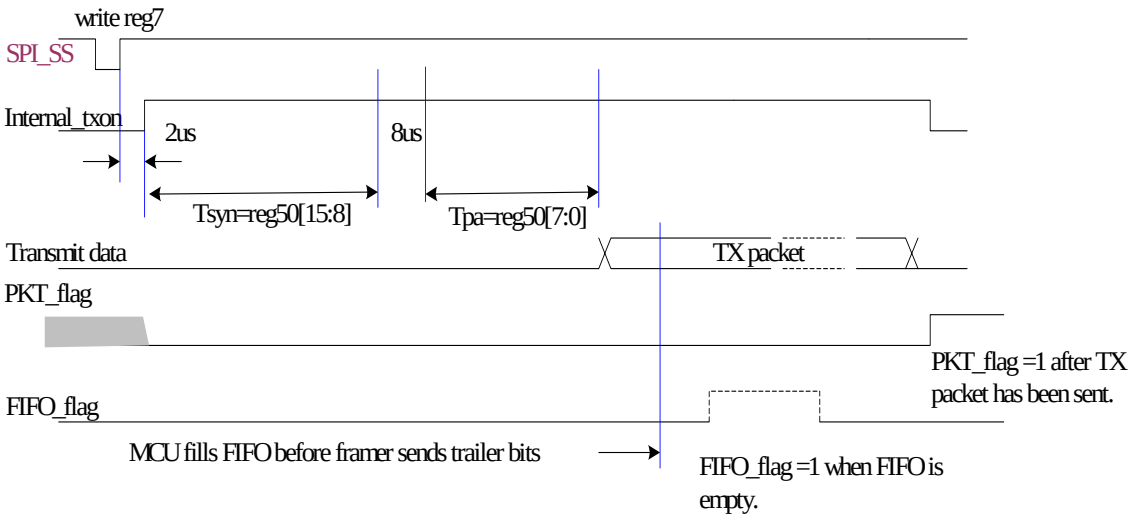
EM198810 provides two ways to handle TX/RX packet length. One is inside framer will detect packet length automatically. Other is framer will keep TX/RX status until MCU terminates TX/RX. The maximum packet length that framer can handle is 255bytes. If TX/RX packet

length is less than 255bytes, we recommend using Framer detect packet length timing diagram. This option is configure via register 57[13]. For detailed information, please refer to EM198810 register definition. Details timing diagram shows as below. All timing diagram set high active for PKT_flag and FIFO_flag, low active is also available via register setting.

TX/RX timing diagram----------Framer detect packet length

Framer of EM198810 will handle packet length by setting Reg57[13]=1. First byte of payload is packet length (this length byte is not counted in the packet length). Maximum allowed packet length is 255 bytes. Framer will handle TX/RX start and stop.

TX timing diagram is shown in figure 4. After MCU writes Reg7[8]=1 and selects transmit channel(refer to Reg7 definition), the framer will automatically process transmit syncword and payload data from FIFO. MCU needs to fill in transmit data before framer sends trailer bits. If packet length is longer than 63 bytes, MCU needs to write in FIFO data multiple times. FIFO_flg indicates whether FIFO is empty in transmit state or not.



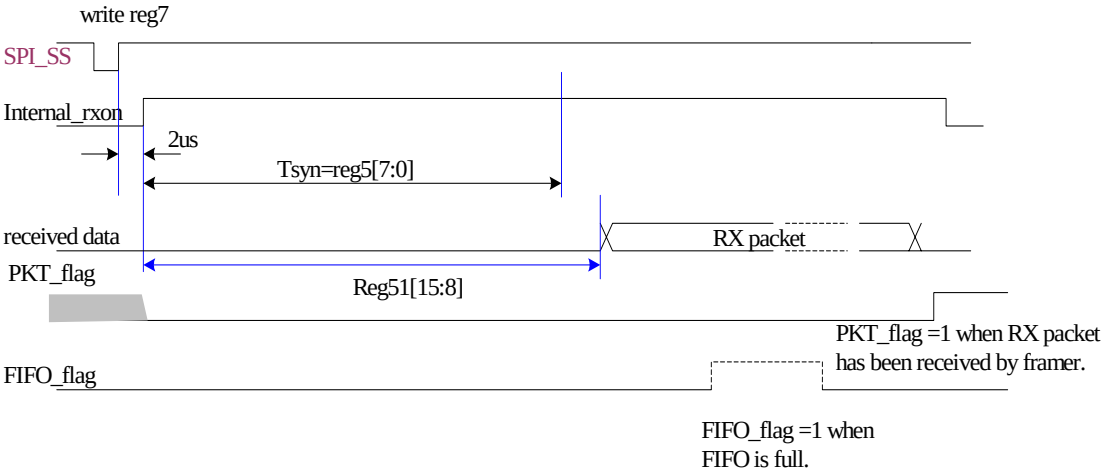
-

Figure 4 -

Figure 4. TX timing diagram when PKT_flag and FIFO_flag are high active

(framer detect packet length)

RX timing diagram is shown in figure 5. When MCU writes Reg7[7]=1 and selects receiving channel, framer will detect valid syncword automatically. When it detects valid syncword, framer will process packet automatically. When received packet process is done, framer will set RF into idle. If received packet length is longer than 63 bytes, FIFO_flg will indicate FIFO full in receiving state, and that means MCU must read out data from FIFO.



-

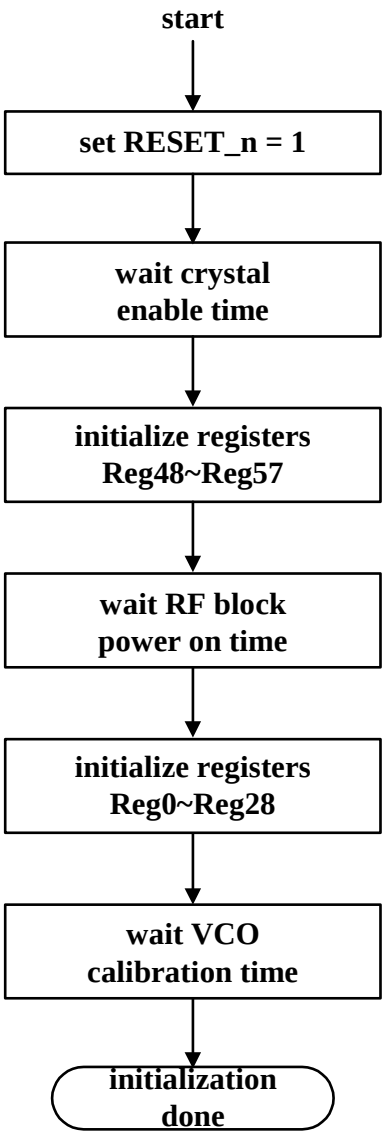
Figure 5 -

Figure 5. RX timing diagram when PKT_flag and FIFO_flag are high active

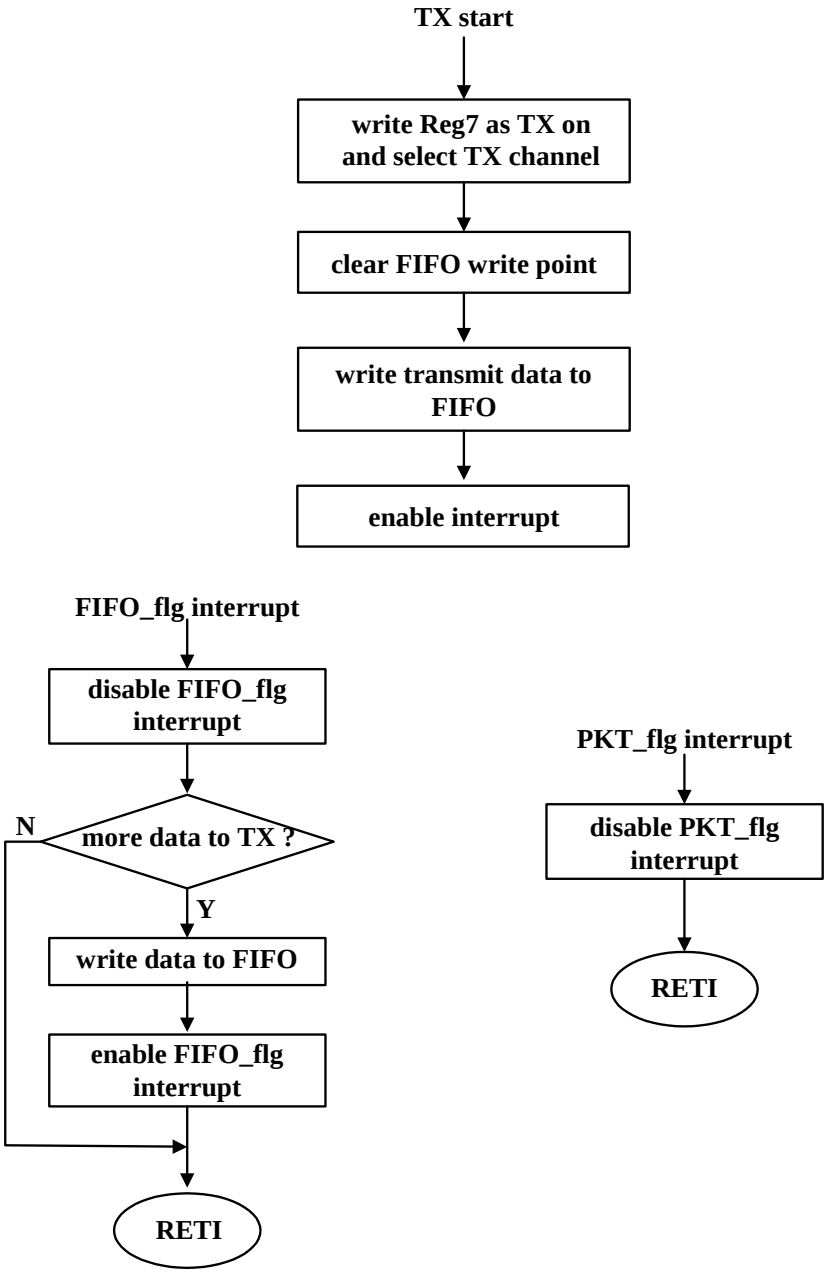
(framer handle packet length)

Examples of flow chat

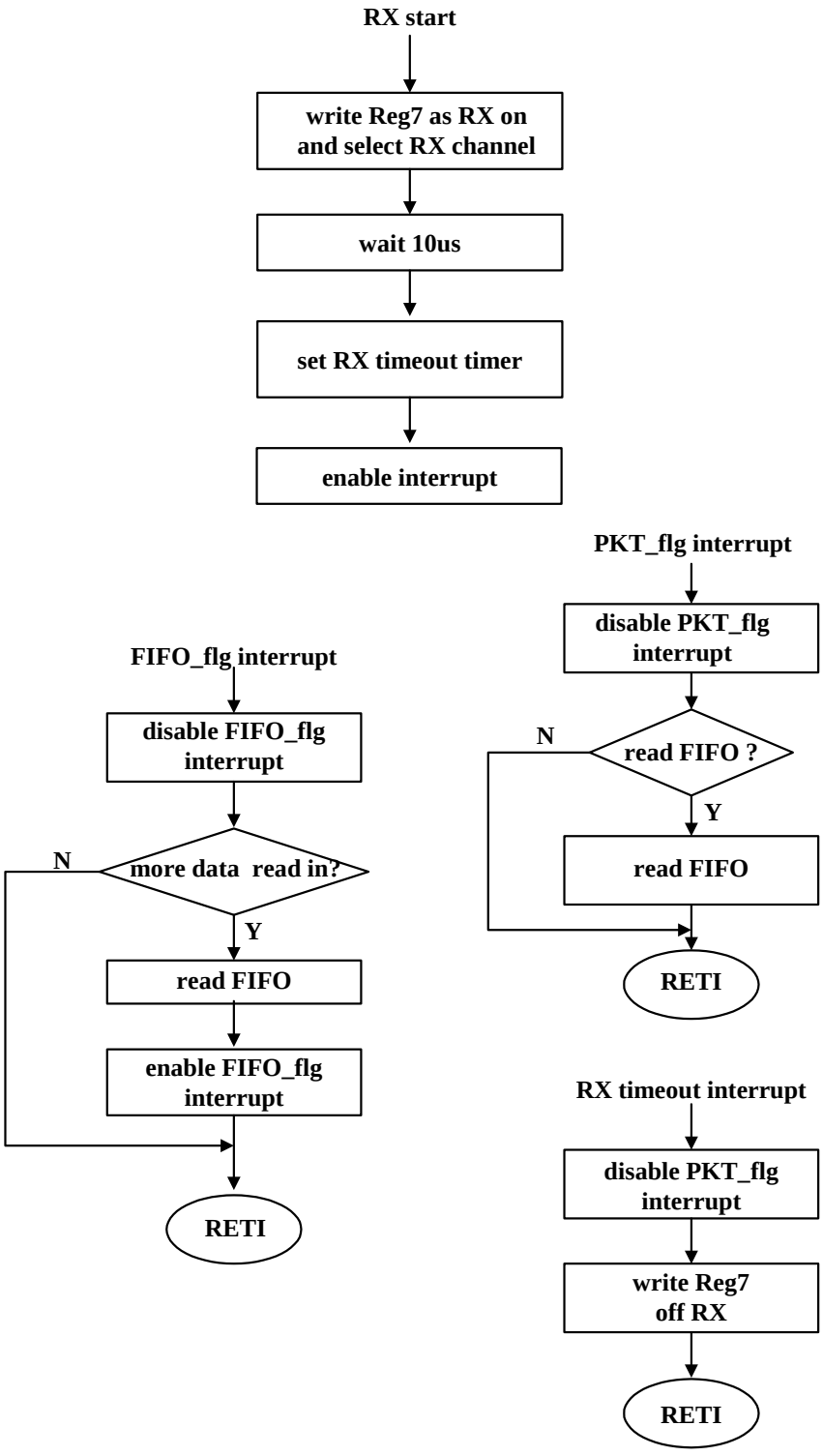
1. Example of initial flow



2. Example of TX flow. FIFO_flg and PKT_flg are interrupt signals to MCU.



3. Example of RX flow. FIFO_flg and PKT_flg are interrupt signals to MCU.



TX/RX timing diagram----------MCU handle packet length

When Reg57[13]=0, Framer of EM198810 always follow MCU's instruction. It means MCU determinates TX/RX start and stop. For more easy to using EM198810, there is another option to determinate TX. If Reg57[8]=1, framer keep detecting FIFO write point and FIFO read point at TX status. If MCU stop write data to FIFO, when framer detects no data to send (it means FIFO is fully empty), EM198810 will exit TX automatically. The timing diagram is shown in Figure 6, If Reg57[8]=0, framer never stop TX until MCU write Reg7 to idle status even FIFO is fully empty. The timing diagram is shown in Figure 7.

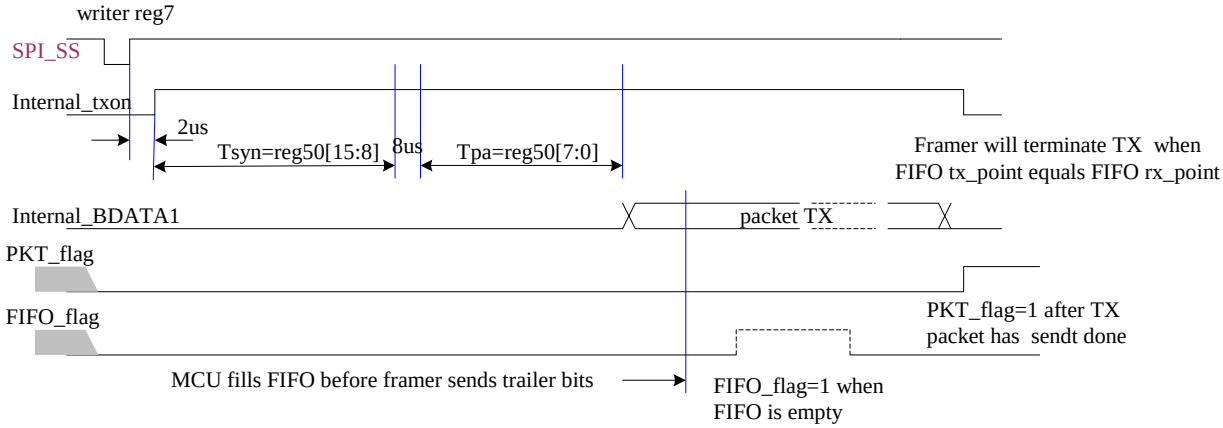


Figure 6. TX timing diagram when PKT_flag and FIFO_flag are high active

(MCU handle packet length, framer determinate TX)

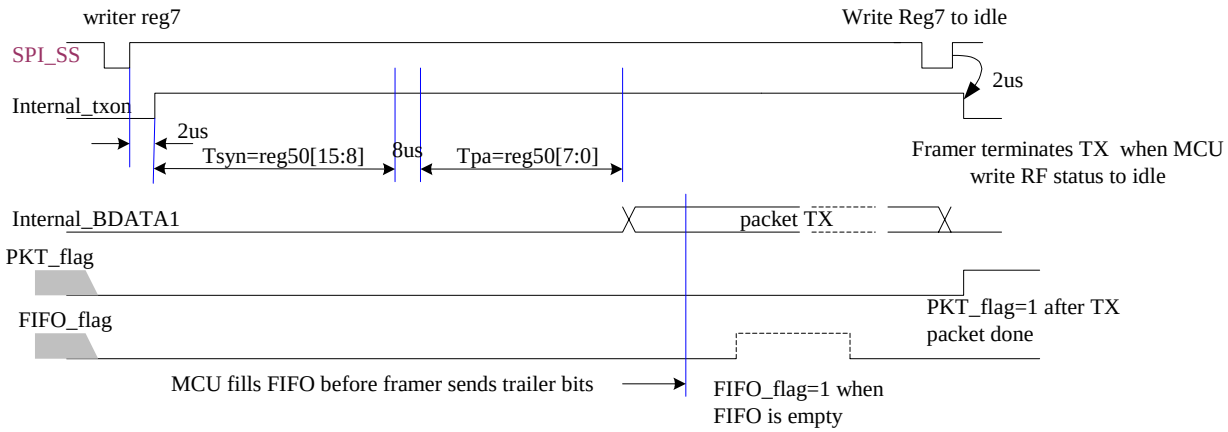


Figure 7. TX timing diagram when PKT_flag and FIFO_flag are high active

(MCU handle packet length, framer doesn't detect FIFO W/R point)

Note: When MCU handle TX/RX packet length, never let FIFO underflow or over flow. It can be controlled via setting Reg56 TX_FIFO_threshold and RX_FIFO_threshold. The value depends on SPI speed.

When MCU write Reg7[7]=1, framer will start to detect SYNC automatically after Tsyn time. When framer detected valid SYNC, it will set PKT_flag active to MCU, than start fill data to FIFO. PKT_flag will keep active until MCU reads out first byte data from FIFO. When MCU read out first byte data, PKT_flag becomes inactive until next TX/RX period. It always needs MCU write Reg7 to stop RX. RX timing diagram is shown in Figure 8

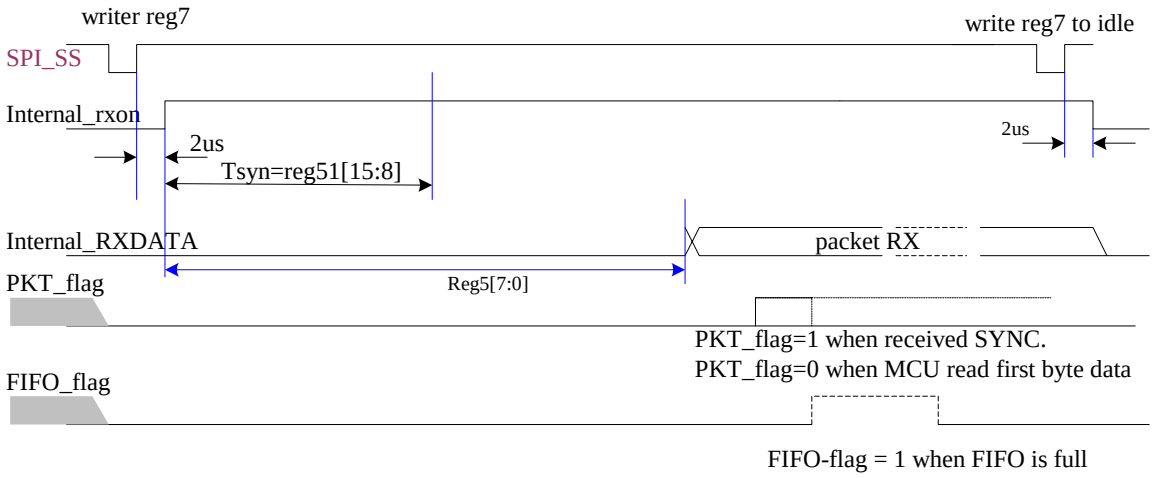
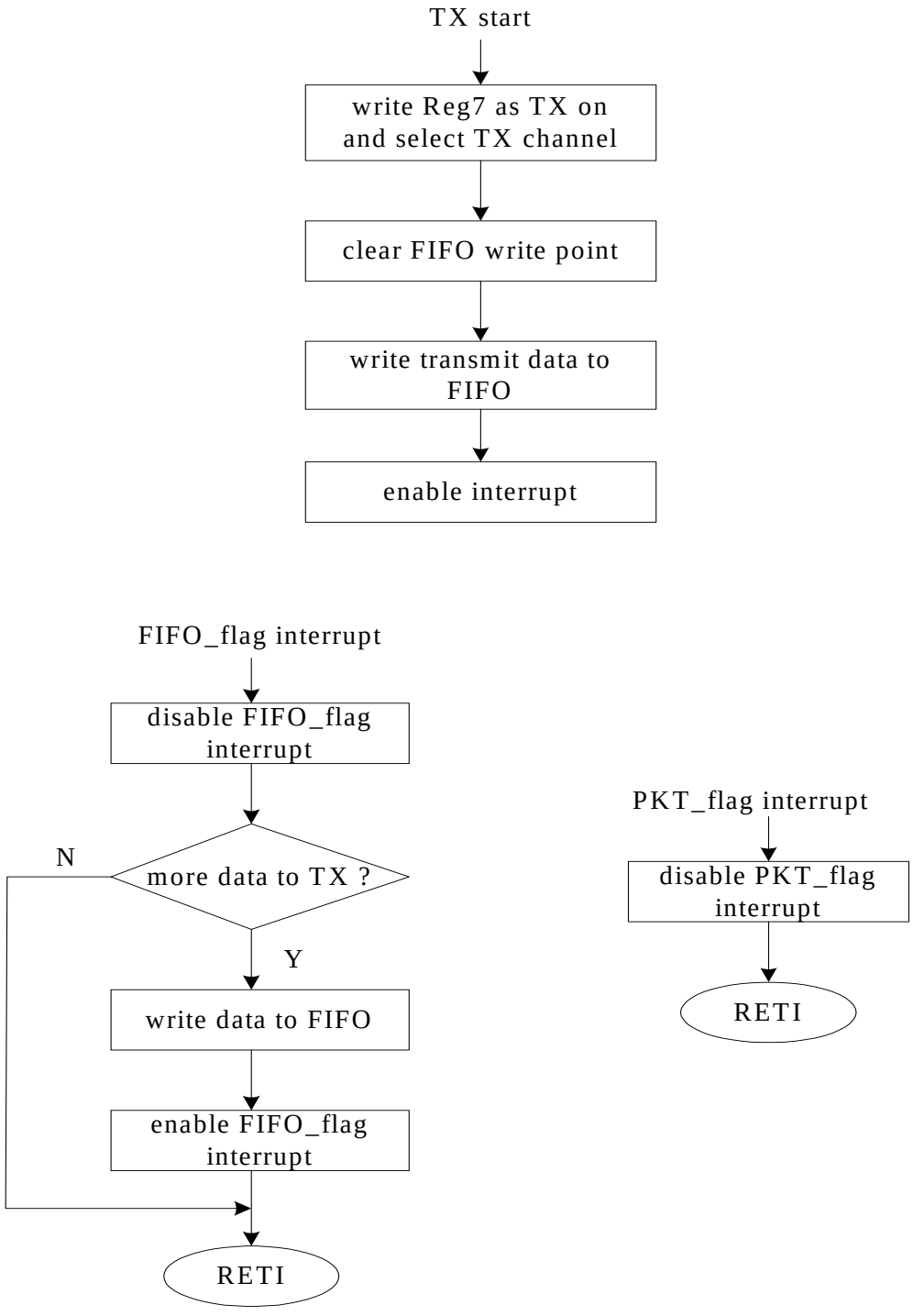


Figure 8. RX timing diagram when PKT_flag and FIFO_flag are high active

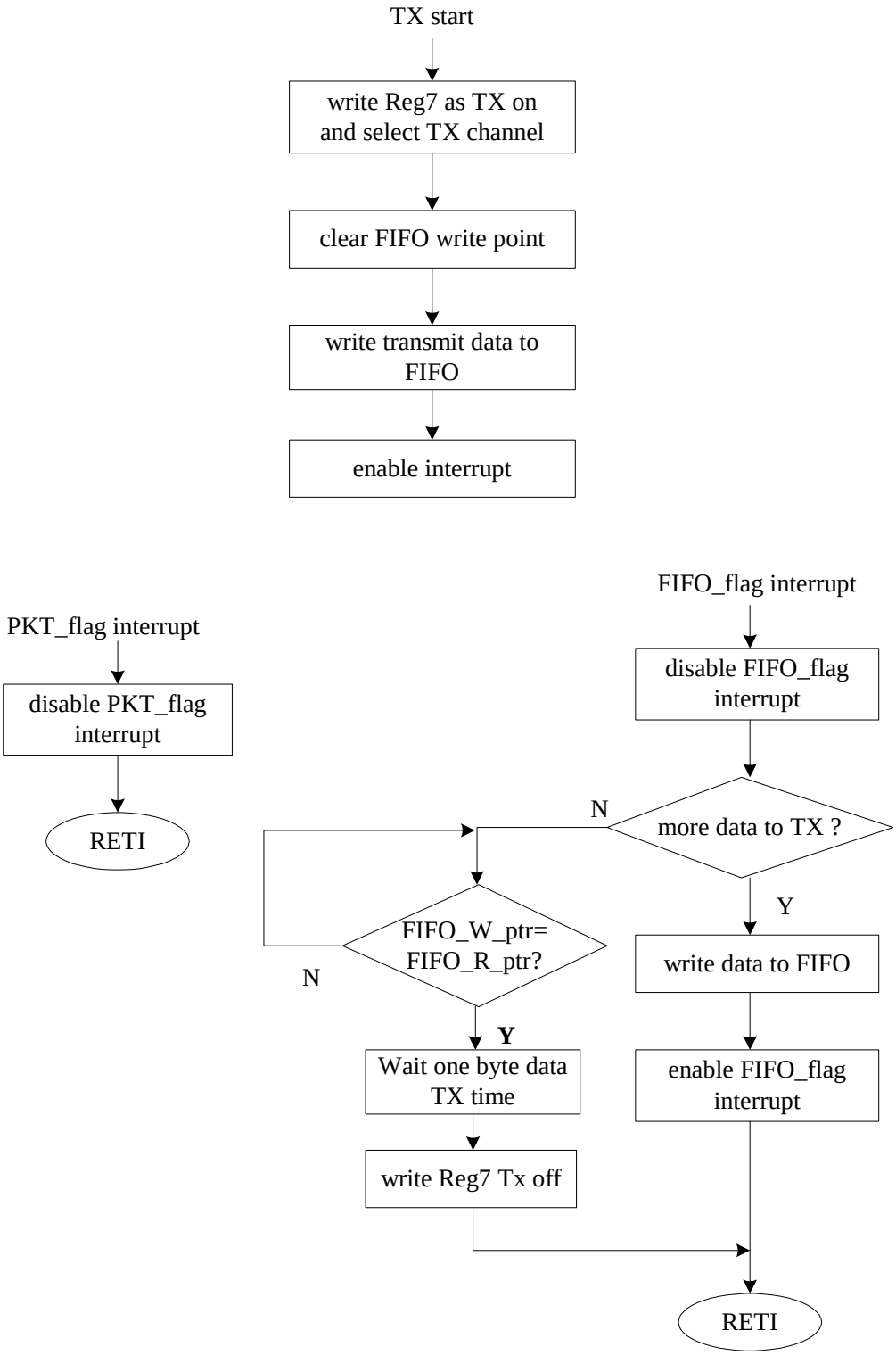
(MCU handle packet length)

Example of flow chat: power on and register initialization is same as framer handle packet length, TX/RX flow chat, PKT_flag and FIFO_flag are defined as MUC's external interrupt source

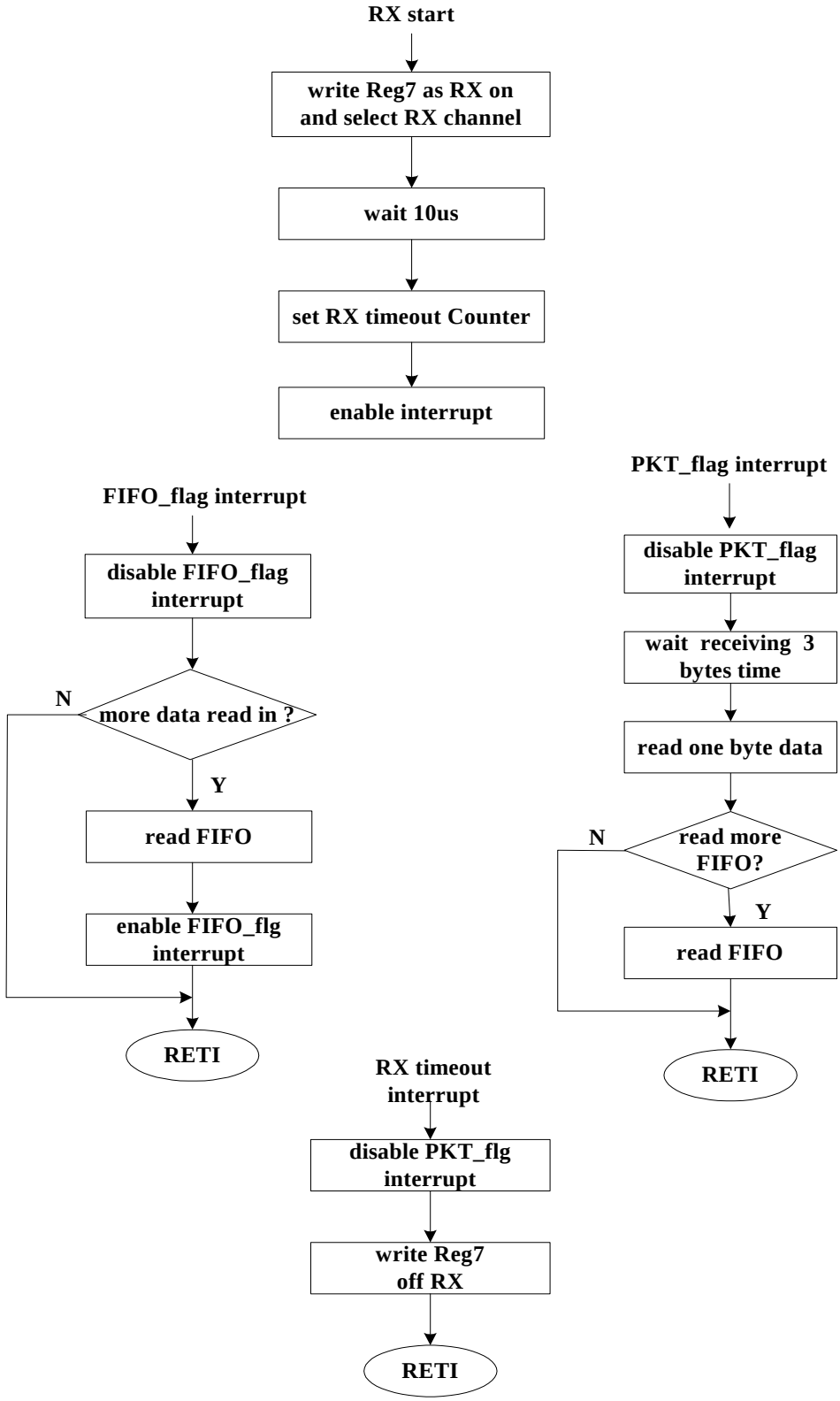
TX flow chart (framer detect FIFO TX/RX point)



TX flow chart (framer doesn't detect FIFO TX/RX point)



. RX flow chart



Typical Register Values

The following register values are recommended for most of customers.

Reg. address	Read/Write	Default value (Hexadecimal)	Recommend value (12MHz crystal frequency) (Hexadecimal)
0	R/W	0000	CD51
2	R/W	00C1	0061
4	R/W	0688	3CD0
5	R/W	0041	00A1
9	R/W	0003	3003
14	R/W	6617	6697
16	R/W	0000	F000
18	R/W	FC00	E000
19	R/W	0014	2114
20	R/W	8103	819C
21	R/W	0962	6962
22	R/W	2602	0402
23	R/W	2602	0802
24	R/W	30C0	B080
25	R/W	3814	7819
26	R/W	5304	6704
48	R/W	1800	5800
51	R/W	4000	A000
56	R/W	4407	4407
57	R/W	B000	E000*

- Table 3 -

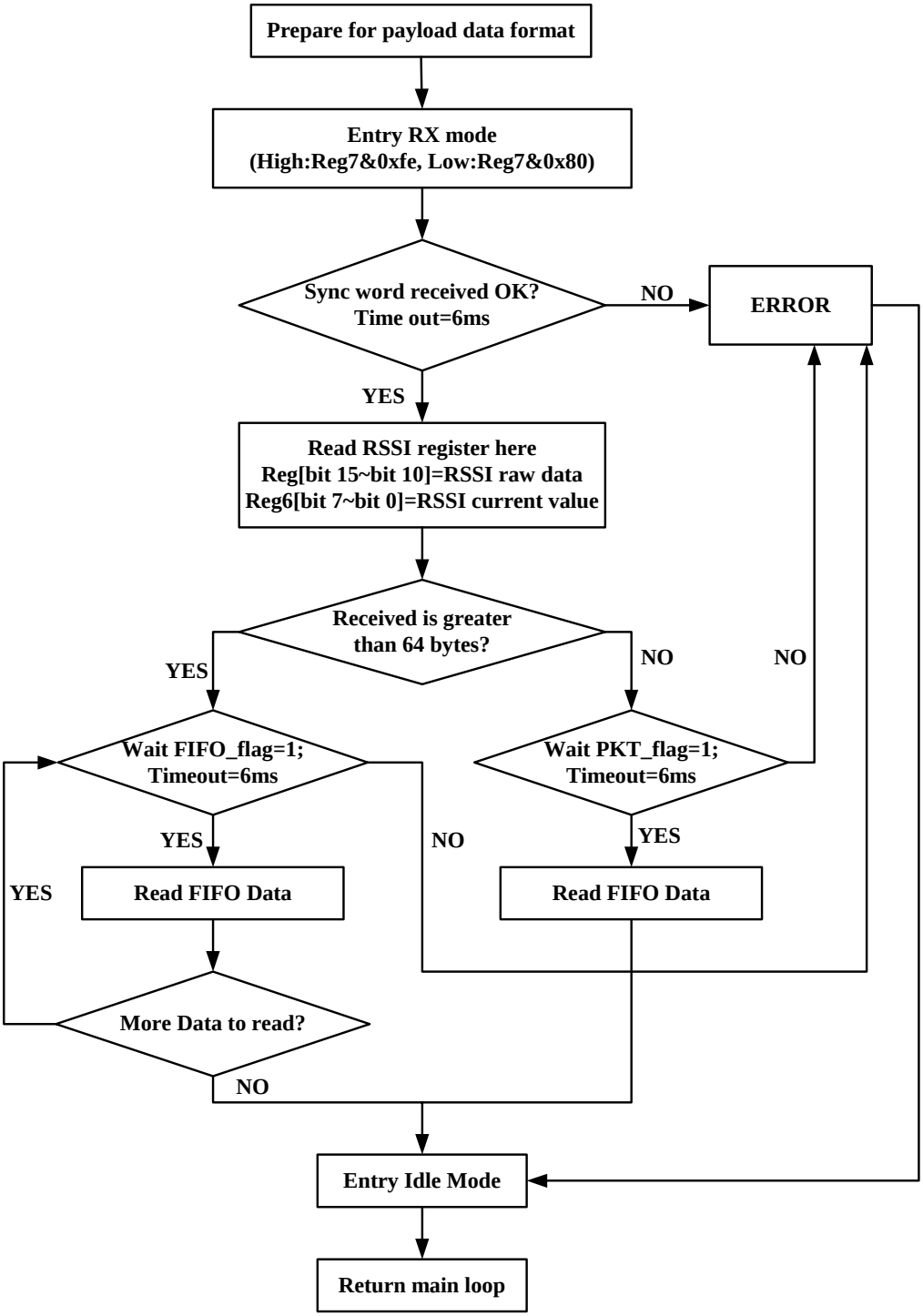
Recommend Registers Setting

- Reg57, if MCU handle packet length and framer detect FIFO fully empty, Reg57=0xC080
- Reg57, if MCU handle packet length and terminates TX done, Reg57=0xC000

Appendix1: Reading the RSSI value & RSSI reference table

(1) Reading the RSSI value:

For reading RSSI value flow chart: (TX side must be in transmission continuously)



For reading RSSI value, the sample code is as follow: (Using Keil C51)

```
unsigned int RX_packet(bit EnableTimout,unsigned int bytes,bit rssi)
{
    bit Timex= 0, TimeOut = 0;                                  unsigned int beri, reg6;
    unsigned char i, j, k,ErrCnt=0;
    unsigned int raw_rssi,d_rssi;
    j=(bytes+1)/64;
    k=(bytes+1)%64;
  GenerateData();          // prepare the data
    Reg_write16(RF_REG_07, (gReg7_high & 0xfe), (gReg7_low | 0x80));  //enter RX mode
    if(rssi)
    {
        Timer4_Init(CLK_6MS);
        while(((0x0400 & Reg_read16(MAIN_STATUS)) != 0x0400) && (timer4_flag==ING));
        if(timer4_flag)
        {
            Timex = 1;
            goto Err;
        }
        reg6 = Reg_read16(RF_REG_06);
        goto Lb;
    }
    Timer4_Init(CLK_25us);          //make sure PKT has time to go low
    while((timer4_flag==ING) && (PKT==1))
        ;
Lb: if(j==0)
    {
        if (EnableTimout)
        {
            Timer4_Init(CLK_6MS);
            while ((PKT==0) && ((Timex= timer4_flag) == ING))
                ;
            if(Timex)
                goto Err;
        }
```

```
        else
            while((PKT==0) && (SW2 == 0))
                ;
        read_FIFO(Temp,(bytes+1));
    }
    else if(j>0)
    {
        if(k==0)
        {
            for(i=0;i<j;i++)
            {
                if(EnableTimout)
                {
                    Timer4_Init(CLK_6MS);
                    while((FIFO_flag==0) && ((TimeOut=timer4_flag) == ING))
                        ;
                    if(TimeOut)
                    {
                        ErrCnt = i+1;
                        if(ErrCnt==1)
                        {
                            Timex = 1;
                            goto Err;
                        }
                        else
                        {
                            read_FIFO((Temp+i*64),Temp[0]%64);
                            Timex = 0;
                            goto Err;
                        }
                    }
                }
                else
                    while(FIFO_flag==0)
                        ;
                read_FIFO((Temp+i*64),64);
```

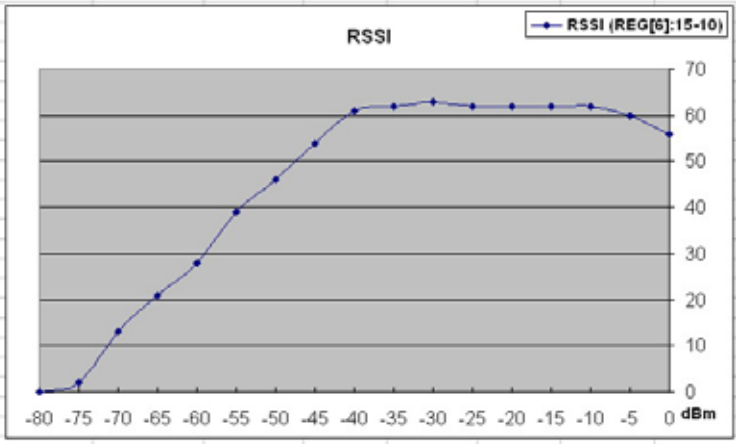
```
        }
    }
    else if(k>0)
    {
        for(i=0;i<j;i++)
        {
            if(EnableTimout)
            {
                Timer4_Init(CLK_6MS);
                while((FIFO_flag==0) && ((TimeOut=timer4_flag) == ING))
                    ;
                if(TimeOut)
                {
                    ErrCnt = i+1;
                    if(ErrCnt==1)
                    {
                        Timex = 1;
                        goto Err;
                    }
                    else
                    {
                        read_FIFO((Temp+i*64),Temp[0]%64);
                        Timex = 0;
                        goto Err;
                    }
                }
            }
            else
                while(FIFO_flag==0)
                    ;
            read_FIFO((Temp+i*64),64);
        }
        if(EnableTimout)
        {
            Timer4_Init(CLK_6MS);  //This not work at all, because timeout is impossible. So you
            can del this code
```

```
                    while((PKT==0) && ((Timex=timer4_flag) == ING))
                            ;
                    if(Timex)
                        goto Err;
                }
                else
                    while((PKT==0)&&(SW2==0))
                            ;
                read_FIFO((Temp+j*64),k);
            }
        }
Err:
        Reg_write16(RF_REG_07, (gReg7_high & 0xfe), (gReg7_low & 0x7f));      //enter idle mode
        if (Timex)
          beri= 0xffff;
        else
          beri = verify_FIFO_data( gDAT,Temp, gLength);    // == 64, SPI takes 113uS
        if(rssi)
        {
            raw_rssi = (reg6 & 0xfc00) >>10;
            d_rssi = reg6 & 0x00ff;
            printf("\rRAW_RSSI = %03u    D_RSSI = %03u",raw_rssi,d_rssi);
        }
        return beri;
}
```

RSSI reference table

Table 1. PA V.S. RSSI value

Input Power (dBm)	RSSI (REG[6]:15-10)
0	56
-5	60
-10	62
-15	62
-20	62
-25	62
-30	63
-35	62
-40	61
-45	54
-50	46
-55	39
-60	28
-65	21
-70	13
-75	2
-80	0



※ Golden range input power is recommend -60dBm.