API manual

1. System Message

1.1 System Message Word

typ_msg_word union is the description of the return value of **sys_msg**.

```
typedef union {
    struct {
        unsigned int    n_a_15          : 1;
        unsigned int    n_a_14          : 1;
        unsigned int    alarm_clock     : 1;
        unsigned int    comm_err        : 1;
        unsigned int    comm_sb_empty   : 1;
        unsigned int    comm_data       : 1;
        unsigned int    n_a_9           : 1;
        unsigned int    n_a_8           : 1;
        unsigned int    RF_on_air       : 1;
        unsigned int    RF_new          : 1;
        unsigned int    time_out        : 1;
        unsigned int    card_gone       : 1;
        unsigned int    card_insert     : 1;
        unsigned int    key_available   : 1;
        unsigned int    key_up          : 1;
        unsigned int    key_down        : 1;
    } bits;
    unsigned short int s_word;
} typ_msg_word;
```

Members:

bits: message word intepreted as bit fields, all the n_a_* fields represent the n/a bits.

- alarm_clock : set if alarm clock event occurred
- comm_err: set if UART communication error occurred
- comm_sb_empty: set if UART send buffer empty
- comm_data: set if new data exists in UART receiving FIFO
- RF_on_air: set if RF unit has started a receiving process
- RF_new: set if new data exists in RF unit receiving buffer
- time_out: set if a preset timer has expired
- card_gone: set if the IC card in socket 2 has been removed
- card_insert: set if a IC card has been inserted into socket 2
- key_available: set if new data exists in keyboard/jogdialer input FIFO.(FIFO mode)
- key_up: set if a key has been released (1-KEY mode)
- key_down: set if a key has been pressed(1-KEY mode)

s_word: message word in raw form

1.2 System Message Inquiring

sys_msg is the system message inquiring subroutine which is supposed to be called in the main loop of message handler. System will enter power saving mode unless there is unmasked message available.

```
int sys_msg (unsigned long if_stay_awake);
```

Parameters:

if_stay_awake : set the power-saving mode while there is no unmasked message

`SM_STAY_AWAKE:` system frozen but all the peripherals still working.

`SM_GOTO_SLEEP:` system enters maximum power saving mode but some peripherals such as UART, IC card sockets would not work.

Return value:

System message word is returned, which could be intepreted by **typ_msg_word**

Example:

```
#include   "api.h"
 typ_msg_word   smw;
…….
  sm_setting(MSG_KEY_DOWN | MSG_KEY_UP);
  while (1) {                          /* main loop of message handler */
     smw.s_word = sys_msg(SM_GOTO_SLEEP);
     if (smw.bits.key_available) {
        /* your keydealers here: */
     }
     if(smw.bits.card_insert) {
        /* your IC card 2 dealer here */
     }
  }
```

1.3 System Message Mask Setting

sm_setting sets the message mask so that **sys_msg** would not return on unwanted messages.

```
unsigned long sm_setting(unsigned long msg_msk);
```

Parameters:

msg_msk : message mask is set by bit-OR'ing the following masks:

```
MSK_KEY_DOWN
MSK_KEY_UP
MSK_KEY_AVAILABLE
MSK_CARD_INSERT
MSK_CARD_GONE
MSK_TIME_OUT
MSK_RF_NEW
```

```
MSK_RF_ON_AIR
MSK_COMM_DATA
MSK_COMM_SB_EMPTY
MSK_COMM_ERR
MSK_ALARM_CLOCK
```

Return values:
The same as input value msg_msk

Examples:
c.f 1.2

1.4 Version Check

Check_version performs version check

```
void Check_Version(unsigned char * vbuff);
```

Parameters:
vbuff : pointer to the buffer to store the version data.

Version data is 12-byte long:

```
typedef struct {
        unsigned char           v_month;
        unsigned char           v_day;
        unsigned short int      v_year;
        unsigned short int      v_hour;
        unsigned char           v_min;
        unsigned char           v_sec;
        unsigned char           v_main;
        unsigned char           v_sub;
        unsigned short int      v_rel;
} typ_version_content;
```

Members:

v_month, v_day, v_year	:	last modification date of BIOS;
v_hour, v_min, v_sec	:	last modification time of BIOS;
v_main	:	version number main part;
v_sub	:	version number sub part
v_rel	:	release number

2. Keyboard

2.1 keyboard unit initialization

KEY_init initializes keyboard interface.

```
void    KEY_init(unsigned long key_flag);
```

Parameters:

key_flag: indicates keyboard operation mode. Set by bit-OR'ing the following flags:

KEY_FIFO_MODE:	keyboard as FIFO mode. Keyboard will send message when key FIFO is not empty. Any key pressed is put into FIFO so it would not be lost. Should not be used with KEY_1_KEY_MODE.
KEY_1_KEY_MODE:	keyboard as 1-key mode. Keyboard will send message when a key is pressed or released. Key data should be dealt with soon to prevent it from being overlapped. Should not be used with KEY_FIFO_MODE.
KEY_BEEP_KEY:	beep when key pressed and hold down.
KEY_AUTO_EL_ON:	automatically turn on backlight when a key is pressed

2.2 keyboard reading

2.2.1 FIFO mode reading

Key reading in FIFO mode is implemented by **KEY_read**:

```
int     KEY_read(void);
```

Return value:

KEY_read returns –1 when FIFO is empty; scan code when data exists in FIFO.

2.2.2 One-key mode reading

2 functions are used as key reading in 1-key mode , **KEY_read_up** and **KEY_read_down**:

```
int     KEY_read_up(void);
int     KEY_read_down(void);
```

Return value:

returns –1 when no key is pressed or released; scan code when there is.

Note:

Do not call these 2 functions unless 1-key mode is enabled and key_up or key_down message sent. Invoke KEY_read_up in key_up message dealer and KEY_read_down in key_down message dealer.

2.2.3 Key matrix map reading

KEY_get_status is a low level function to return the key matrix bit map. It could be called anytime to check keyboard status.

```
Unsigned short int   KEY_get_status(void);
```

Return value:

16-bit key matrix bit map representing 4 x 4 key board.

2.2.4 Key auto beep and auto EL masking

Each of the 16 keys could be masked from auto beeping and auto EL individually, i.e. user could specify which keys will beep or turn on EL when pressed.

KEY_mask_set is used to set these two masks:

```
void KEY_mask_set(unsigned long key_mask);
```

Parameters:

key_mask : the 16 MSBs is the mask for beep mask, and the 16 LSBs is the mask for auto EL mask. If any bit in these masks are set (1) , the pressing of corresponding key will trigger beep or EL.

KEY_mask_read is used to get the current setting of these two masks:

```
Unsigned long KEY_mask_read(void);
```

Returned value:

The same as parameter key_mask in **KEY_mask_set**.

3. Buzzer

BEEP_sound is the buzzer function.

```
void       BEEP_sound(unsigned long SREG);
```

Parameters:

SREG: controls buzzer. Only 16 LSBs are used.

Bits	15	14	13	12	11	10	9	8
Name	ON	D						
Bits	7	6	5	4	3	2	1	0
Name	N							

ON : 1 to turn on buzzer and 0 to shut it up

D: 0~127 to set buzzer output dutycycle to D/255 (0~50%)

N: set buzzer output frequency to 32768/N

4. Backlight

4.1 Switching

EL_setting controls backlight.

```
void       EL_setting(unsigned long ONOFF);
```

Parameters:
ONOFF: 1 to turn on and 0 to turn off. or EL_ON to turn on and EL_OFF to turn off.

4.2 Backlight Auto off time setting

Backlight is turned off after a certain period of time if no key is pressed. **EL_set_time_out** sets this time.

```
void       EL_set_time_out(unsigned long TOT);
```

Parameters:
TOT: timeout time for backlight auto off in seconds. only 8 LSBs are used.

5. Battery Check

BATT_check_level performs battery check and returns current battery level.

```
int     BATT_check_level(void);
```

Return value:
100H for full scale (3 Volts).

Note:
1) Return value is not a linear function of battery voltage. Tests should be done to check the return value of certain points.
2) Battery check is a somehow time consuming process. RF information maybe lost in such process.

6. Real Time Clock

6.1 time structure

typ_RTC_time_rec is a union for time accessing.

```
typedef union {
   struct {
      unsigned char hour : 8;
      unsigned char minute : 8;
      unsigned int second:16;
   } fields;
   unsigned long l_word;
} typ_RTC_time_rec;
```

Members:
fields: union intepreted as time fields

hour : hour (0~23)
minute: minute(0~59)
second: second(0~59)

l_word: time data in raw form

6.2 date structure

typ_RTC_date_rec is a union for date accessing.

```
typedef union {
   struct {
      unsigned int year : 8;
      unsigned char month : 8;
      unsigned char day:16;
   } fields;
   unsigned long l_word;
} typ_RTC_date_rec;
```

Members:

fields: union intepreted as date fields

year : year (1999~2098)
month: month(0~11)
day: dat(0~31)

l_word: date data in raw form

6.3 time reading

RTC_read_time reads RTC time

```
unsigned long     RTC_read_time(void);
```

Return value:

Raw time data which could be intepreted by typ_RTC_time_rec

6.4 time setting

RTC_set_time sets RTC time

```
int RTC_set_time(unsigned long time_rec);
```

Parameters:

time_rec : raw time data to set the RTC time to. Macro MAKE_TIME could be used to simplify time setting.

Return value:

-1 when time_rec is not a valid time. Same as time_rec when successful

Example:

```
#include <api.h>

if (RTC_set_time(MAKE_TIME(15,03,28)) == -1) {    /* set current time to 15:03:28 */
   /* put your error handling code here */
}
```

6.5 date reading

RTC_read_date reads RTC date

```
unsigned long     RTC_read_date(void);
```

Return value:

Row date data which could be intepreted by typ_RTC_date_rec

6.6 date setting

RTC_set_date sets RTC date

```
int RTC_set_date(unsigned long date_rec);
```

Parameters:

date_rec : raw date data to set the RTC date to. Macro MAKE_DATE could be used to simplify date setting.

Return value:

-1 when date_rec is not a valid date. Same as date_rec when successful

6.7 alarm time reading

RTC_read_alarm reads RTC alarm time

```
unsigned long     RTC_read_alarm(void);
```

Return value:

Raw time data which could be intepreted by typ_RTC_time_rec

6.8 alarm time setting

RTC_set_alarm sets RTC alarm time

```
int RTC_set_alarm(unsigned long time_rec);
```

Parameters:

time_rec : raw time data to set the RTC alarm time to. Macro MAKE_TIME could be used to simplify time setting.

Return value:

-1 when time_rec is not a valid time. Same as time_rec when successful

6.9 alarm clock on/off setting

RTC_alarm_setting turn alarm clock on/off

```
void       RTC_alarm_setting(unsigned long ONOFF);
```

Parameters:

ONOFF: RTC_ALARM_ON (1) to turn on and RTC_ALARM_OFF (0) to turn off.

6.10 alarm clock on/off reading

RTC_get_alarm_setting reads RTC alarm clock on/off state

```
unsigned long     RTC_get_alarm_setting(void);
```

Return value:

0 for alarm clock off and 1 for on

7. IC Card interface

IC card interface is a set of low level subroutines which is used when a new card type is used. It's recommended that they are not called directly from the application program.

7.1 data structures

7.1.1 ATR data

typ_Card_ATR_param is ATR command & data buffer for IC card ATR communication.

```
typedef struct {
  short int    card_num : 8;
  short int    card_type_7 : 1;
  short int    card_type_6 : 1;
  short int    card_type_5 : 1;
  short int    card_type_active_lo : 1;
  short int    card_type_3 : 1;
  short int    card_type_2 : 1;
  short int    card_type_1 : 1;
  short int    card_type_async : 1;
  short int    ATR_char_cnt : 16;
  unsigned long ATR_async_freq;
  unsigned char * ATR_buff;
} typ_Card_ATR_param;
```

Members:

Fields described here except all card_type_? bit fields are n/a.

card_num : card socket number (0 ~ 2) to perform ATR communication . should be filled before ATR.

card_type_active_lo : card reset type. filled by ATR session.

card_type_async : set if the card is asynchronous. Better be filled before ATR.

ATR_char_cnt :count of ATR data returned.

ATR_async_freq : oscillator frequency to feed into card for asynchronous card. Should be set as one of the following flags before asynchronous card ATR:

CARD_ATR_AFREQ_3571712 : set to 3571712Hz

CARD_ATR_AFREQ_7143424 : set to 7143424Hz

CARD_ATR_AFREQ_8290304 : set to 8290304Hz

ATR_buffer : pointer to ATR data buffer in which the ATR data will be stored in.

7.1.2 I^2C serial data

typ_IIC_param_block is the buffer used for I^2C like synchronous card session.

```
typedef struct {
  short int    delay_unit : 8;
  short int    pin_SCL    : 8;
  short int    pin_SDA    : 8;
  short int    IIC_flag   : 8;
  short int    send_length;
  short int    read_length;
  unsigned char * send_buffer;
  unsigned char * read_buffer;
} typ_IIC_param_block;
```

Members:

delay_unit : specify the period of clock pulse. 0 for fastest.

pin_SCL : clock pin for this session. Should be one of the following pins:

CARD_PIN_C8 : C8 pin of card socket

CARD_PIN_C4 : C4 pin of card socket

CARD_PIN_CLK : C3 pin of card socket

CARD_PIN_IO : C7 pin of card socket

Pin_SDA : data pin for this session. Should be one of the pins as above.

IIC_flag : detailed description of the I^2C like synchronous card session. Set by bit-OR'ing the following flags:

CARD_IIC_LSB_1 : data is LSB first.

CARD_IIC_WITH_ACK : send and check ACK

CARD_IIC_END_NOTACK: end w/o sending ACK

	CARD_IIC_WITH_STOP : send STOP condition
	CARD_IIC_WITH_START : send START condition
send_length	: length of data to send. For command mostly.
read_length	: length of data to read. For data mostly
send_buffer	: pointer to buffer storing data to send.
read_buffer	: pointer to buffer storing data read. Filled after the session

Remarks:

SCL: 

SDA:

7.2 Common card routines

7.2.1 Card interface initialization

Card_init is used to initialize card interface

```
void   Card_init( void );
```

7.2.2 ATR

Card_ATR activates an IC card and waits for the ATR.

```
int Card_ATR(typ_Card_ATR_param * pCAP);
```

Parameters:

pCAP : pointer to typ_Card_ATR_param which stored card parameters and ATR data.(c.f. typ_Card_ATR_param)

Returned value:

ATR length. Set as the same as pCAP->ATR_char_cnt

7.2.3 Card deactivate

Card_deactivate is used to power off all 2-card sockets and the card interface unit.

```
void   Card_deactivate( void );
```

7.2.4 Socket 2 status

Card2_stat shows if there is card in socket #2

```
int Card2_state( void );
```

Returned value:

Zero : no card in socket #2

none zero : card present in socket #2

7.2.5 Card interface power on

Card_power_on is used to power on the entire card interface unit but leave the card sockets unpowered.

```
void    Card_power_on(void);
```

7.2.6 Select card socket

Card_select_sock selects a card socket. The following card operations will be carried out on the card selected. The status of the card selected before Card_select_sock is left unchanged (idle if it's powered).

```
void    Card_select_sock(unsinged long int socknum);
```

Parameters:

socknum: socket number to select. 1 for socket #1, and 2 for socket #2. 0 is also implemented to hang all 2 sockets. Because the system clock may change during card operations, it's extremely important for the system to switch back to it's normal speed, without losing control of the cards.(c.f.developers manual).

7.2.7 Deactivating card socket

Card_deact_curr_sock is used to power off the selected card socket. Only the selected card socket will be powered off, the others are not affected, while

```
void Card_deact_curr_sock(void);
```

7.2.8 Get current socket number

Card_get_curr_sock gets the number of the socket currently selected.

```
int  Card_get_curr_sock(void);
```

Return value:

1,2 : currently selected cardsocket number.

-1 : no card socked is selected.

7.3 synchronous card routines

7.3.1 Direct pin controls

7.3.1.1 RST pin

Macro **Card_pin_RST_hi** sets the RST pin of the IC card in the current activated socket, and macro **Card_pin_RST_lo** resets it.

7.3.1.2 C6 pin

Macro **Card_pin_C6_hi** sets the C6 pin of the IC card in the current activated socket, and macro **Card_pin_C6_lo** resets it.

7.3.1.3 C3/C4/C7/C8 pin

Card_pin_session is used to control/read C3/C4/C7/C8 pins.

```
int Card_pin_session( unsigned long PFLAG );
```

Parameters:

PFLAG : set by bit-OR'ing following flags

a) pin flags : one of the following pin specifier

CARD_PIN_C8	: C8 pin of card socket
CARD_PIN_C4	: C4 pin of card socket
CARD_PIN_CLK	: C3 pin of card socket
CARD_PIN_IO	: C7 pin of card socket

b) operation flags

CARD_PIN_READ	: read specified pin
CARD_PIN_SET	: write 1 to specified pin
CARD_PIN_CLR	: write 0 to specified pin
CARD_PIN_SWITCH_IN	: switch specified pin to input mode
CARD_PIN_SWITCH_OUT	: switch specified pin to output mode

Returned value:

Valid only when CARD_PIN_READ is specified.

Zero	: specified pin is 0
None zero	: specified pin is 1

7.3.2 I^2C serial communication

Card_IIC_session initiates an I^2C like communication session.

```
int Card_IIC_session( typ_IIC_param_block * pIPB );
```

Parameters:

pIPB : pointer to typ_IIC_param_block which contains descriptions of the session

Returned value:

Zero	: session ended successfully.
None zero	: abnormal ends. Mostly because of the invalid ACK bit.

7.4 asynchronous card routines

7.4.1 T= 0

Card_exchange_T0_header is the T=0 data exchanger.

```
void Card_exchange_T0_header(unsigned char * pbuff);
```

Parameters:

pbuff : pointer to buffer that stores conversation flag and T=0 TPDU data when entering the routine, and response data length and data after the routine returns.

Data buffer:

1) TPDU for receiving data

Before calling the routine:

```
struct {
    unsinged short int   is_TPDU_for_receiving;
    unsigned char CLS;
    unsigned char INS;
    unsigned char P1;
    unsigned char P2;
    unsigned char Le;
    unsigned char rec_buff[Le+2]
}
```

where is_TPDU_for_receiving is not 0, Le is the expected length of response data. The buffer should be large enough to hold the data received.

Upon returning of the routine:

```
struct {
    unsinged short int  received_data_length;
    unsigned char CLS;
    unsigned char INS;
    unsigned char P1;
    unsigned char P2;
    unsigned char Le_left;
    unsigned char rec_buff[received_data_length]
}
```

where received_data_length is the actual length of the data received, including the SW; if Le_left is not 0, the T=0 data exchanging was not completed. rec_buff holds the data received and SW.

2) TPDU for sending data

Before calling the routine:

```
struct {
    unsinged short int   is_TPDU_for_receiving;
    unsigned char CLS;
    unsigned char INS;
    unsigned char P1;
    unsigned char P2;
    unsigned char Lc;
    unsigned char send_buff[Lc];
```

```
            unsigned char rec_buff[2];
        }
        where is_TPDU_for_receiving is 0, Lc is length of the
        data to be sent. The buffer should be large enough to
        hold the data received.

    Upon returning of the routine:
        struct {
            unsinged short int  received_data_length;
            unsigned char CLS;
            unsigned char INS;
            unsigned char P1;
            unsigned char P2;
            unsigned char Lc_left;
            unsigned char send_buff[Lc];
            unsigned char rec_buff[2];
        }
        where received_data_length is the actual length of the
        data received, i.e., SW; if Lc_left is not 0, the T=0
        data exchanging was not completed. rec_buff holds SW.
```

7.4.2 T= 1

Card_exchange_T1_frame is the T=1 frame exchanger.

```
void Card_exchange_T1_frame(unsigned char * pbuff);
```

Parameters:

pbuff : pointer to buffer that stores packet length and T=1 frame data.

```
Data buffer:
   struct {
     unsinged short int   pack_length;
     unsigned char frame_data[pack_length];
   }
```

After the routine returns, pack_length in data buffer will be set to the length of the response packet, and frame_data will be set to response packet data.

8. UART peripherals

8.1 data structures

8.1.1 UART status

typ_UART_stat_word reflects the detailed status of UART.

```
typedef union {
  struct {
    unsigned int in_FIFO_full : 1;
    unsigned int in_FIFO_half : 1;
```

```
    unsigned int in_data_ready: 1;
    unsigned int in_old_data  : 1;
    unsigned int in_over_run  : 1;
    unsigned int in_frame_err : 1;
    unsigned int in_break     : 1;
    unsigned int in_parity_err: 1;
    unsigned int out_FIFO_empty    : 1;
    unsigned int out_FIFO_half     : 1;
    unsigned int out_FIFO_available : 1;
    unsigned int out_send_brk      : 1;
    unsigned int out_no_CTS        : 1;
    unsigned int out_busy          : 1;
    unsigned int out_CTS_stat      : 1;
    unsigned int out_CTS_delta     : 1;
    unsigned int n_a_7  : 1;
    unsigned int n_a_6  : 1;
    unsigned int n_a_5  : 1;
    unsigned int n_a_4  : 1;
    unsigned int n_a_3  : 1;
    unsigned int buff_data_available  : 1;
    unsigned int buff_overflow       : 1;
    unsigned int UART_on            : 1;
    unsigned char n_a;
  } bits;
  unsigned long l_word;
} typ_UART_stat_word;
```

Members:

bits: UART status word intepreted as bit fields, all the n_a_* fields represent the n/a bits. All except buff_data_available, buff_overflow and UART_on are low level status indicators and it's highly recommended that user do not use them.

in_FIFO_full	:set if input FIFO is full
in_FIFO_half	:set if input FIFO reaches half capacity
in_data_ready	:set if input FIFO is not empty
in_old_data	:set if data in FIFO is old (i.e. UART has been in idle mode for a while)
in_over_run	: set if an overrun error has occurred
in_frame_err	: set if an frame error has occurred
in_break	: set if break condition detected
in_parity_err	: set if parity error has occurred
out_FIFO_empty	: set if output FIFO is empty
out_FIFO_half	: set if output FIFO reaches half capacity
out_FIFO_available	: set if output FIFO is available (not full)
out_send_brk	: set if initiating a break condition

out_no_CTS : set if CTS is ignored
out_busy : set if UART is sending a character
out_CTS_stat : CTS state
out_CTS_delta : set if CTS state changed
buff_data_available : set if there is data stored in UART buffer
buff_overflow : set if UART buffer has overflowed
UART_on : set if UART is on

l_word: UART status word in raw form

8.1.2 fcntl word

typ_UART_f_word is the device control word used by UART_fcntl.

```
typedef union {
  struct {
    unsigned int tx_enable           : 1;
    unsigned int send_brk            : 1;
    unsigned int ignore_cts          : 1;
    unsigned int force_parity_err     : 1;
    unsigned int loop_test           : 1;
    unsigned int n_a_10              : 1;
    unsigned int rts_auto            : 1;
    unsigned int rts_value           : 1;
    unsigned int rx_polarity         : 1;
    unsigned int tx_polarity         : 1;
    unsigned int n_a_5               : 1;
    unsigned int n_a_4               : 1;
    unsigned int n_a_3               : 1;
    unsigned int n_a_2               : 1;
    unsigned int n_a_1               : 1;
    unsigned int n_a_0               : 1;
  } bits;
  unsigned short int s_word;
} typ_UART_f_word;
```

Members:

bits: UART device control word intepreted as bit fields, all the n_a_* fields represent the n/a bits.
tx_enable : set if UART transmission is enabled
send_brk : set if initiating a break condition
ignore_cts : set if CTS is ignored
force_parity_err : set if forcing a parity error
loop_test : set if UART is in intermal loop testing mode
rts_auto : set if rts is set automatically

rts_value : RTS level
rx_polarity : set if inverse RX polarity is used
tx_polarity : set if inverse TX polarity is used

s_word: UART status word in raw form

8.2 initialization

UART_init is used to turn on/off UART.

```
int UART_init( unsigned long int uflags);
```

Parameters:

uflags : UART_OFF to turn off all UART related peripherals including modem, infrared and RS232 interface. UART is turn on by bit-OR'ing UART_ON and one or more of following flags:

a) peripheral flags:
one and only one of following 3 peripherals should be specified
UART_232_ON : set if RS232 is used for UART
UART_IRDA_ON : set if infrared interface is used for UART
UART_MODEM_ON : set if internal modem is used for UART

b) no parity bit is present if none of the 2 flags in this group is specified. Only one of the following 2 flags should be specified
UART_ODD_PARITY : set if odd parity bit is used
UART_EVEN_PARITY: set if even parity bit is used

c) UART_2_STOP_BITS : specified if 2 stop bits used. 1 stop bit by default.

d) UART_8_DATA_BITS : specified if 8 data bits used. 7 data bits by default

e) Baud rate flags:
One and only one of following flags should be specified to set the baud rate of UART communication:
UART_BAUD_115200
UART_BAUD_57600
UART_BAUD_28800
UART_BAUD_14400
UART_BAUD_38400
UART_BAUD_19200
UART_BAUD_9600
UART_BAUD_4800
UART_BAUD_2400
UART_BAUD_1200
UART_BAUD_600
UART_BAUD_300

f) None integer prescaler setting:

UART_NIP_ON : internal use only.

8.3 Read

UART_get_char is used to read data from UART data buffer. It's hightly recommended that it is called in comm_data message dealer.

```
int UART_get_char(void);
```

Returned value

-1 if no data available ;

0~0xFF for data received.

8.4 Write

UART_send_char is used to write data to UART output buffer.

```
int UART_send_char(unsigned long ch);
```

Parameters:

ch : data to send. Only 8 LSB's are used.

Returned value

0 if succeeded ;

-1 if failed. Call UART_stat to see the reason why it failed.

8.5 device control

UART_fcntl is used to perform device control on UART.

```
unsigned short int      UART_stat(unsigned long CWord);
```

Parameters:

Cword : set to UART_F_INQ to read current device control setting or set current device control by bit-OR'ing one or more of following flags

a) UART_F_TX_POL : set if inverse TX polarity

b) UART_F_RX_POL : set if inverse RX polarity

c) RTS control flags. Only one of 3 flags should be specified

UART_F_RTS_HIGH : force RTS high

UART_F_RTS_LOW : force RTS low

UART_F_RTS_AUTO : RTS is set automatically

d) UART_F_LOOP_TEST : force internal loop back

e) UART_F_FORCE_PARITY_ERR : force parity bit error

f) UART_F_NO_CTS : ignore CTS

g) UART_F_SEND_BREAK : initiate a break condition

h) UART_F_TX_ENABLE : TX is disabled if not specified

Returned value

Status word with bit fields described as typ_UART_stat_word.

Example:

```
#include <api.h>
/* UART_fcntl to ignore CTS pin and all the other fcntl bits unchanged */
void main(void) {
……
    UART_fcntl(UART_fcntl(UART_F_INQ) | UART_F_NO_CTS);
}
```

8.6 status reading

UART_stat is used to get UART status.

```
unsigned long    UART_stat(void);
```

Returned value

Status word with bit fields described as typ_UART_stat_word.

9. User Timer

9.1 set user timer

SPT_set sets user timer period. A time_out message will be sent when this period of time expires and automatically turn off the timer.

```
void      SPT_set(unsigned long duration);
```

Parameters:

duration: time out duration of user timer in 1/64 seconds. only 16 LSBs are used.

9.2 read user timer

SPT_read reads user timer.

```
unsigned short int   SPT_read(void);
```

Return value:

Time left to time_out event in 1/64 seconds.

9.3 register user timer callback routine

SPT_register_call_back set a callback routine which is called each time a time_out event occurs

```
void SPT_register_call_back( void (* CBroutine)(void));
```

Parameters:

CBroutine: callback routine name

Note:
The callback routine is called before the time_out message is sent.

Example:

```
#include <api.h>

/* this example shows how to use user timer callback routine          */
/* cb_spt will be called for the first time 4 seconds after SPT_set in  */
/* the main function , and every 1 second after that                  */

void cb_spt(void) {
   SPT_set(64);          /* reinitialize user timer */
}
main() {
……
   SPT_register_call_back(cb_spt);
    SPT_set(256);
}
```

10. Display

10.1 Initialization

Disp_init controls the LCD display panel.

```
void Disp_init(unsigned long PowerFlag);
```

Parameters:
PowerFlag: one of the following flags:

DISP_INIT_ON: turn on LCD panel. The entire display panel is accessible.

DISP_INIT_OFF: turn off LCD panel. The entire display panel is inaccessible

DISP_INIT_POWER_SAVE: turn off the main display portion of the LCD panel. Only the static icon (bull's eye) is accessable. Anything written to the panel at this time will be displayed the next time LCD is turned on.

10.2 character displaying

10.2.1 Font setting

Disp_set_font_attribute sets current font and character attribute.

```
void Disp_set_font_attribute(unsigned long FA_FLAG);
```

Parameters:
FA_FLAG: set by bit-OR'ing the attribute flag and the font type flag

Attribute flag : not specified – normal display

DISP_FONT_ATTR_INV - inversed display

Font type flag set as one of the following fonts:

DISP_FONT_TYPE_7x9 : 7x9 ASCII font displayed as 8x16

DISP_FONT_TYPE_16x16_SYMBOL : 16x16 Chinese char font , internal code 2121H~297EH

DISP_FONT_TYPE_16x16_CCHAR: 16x16 Chinese char font , internal code 3021H~777EH

DISP_FONT_TYPE_5x7_ENG: 5x7 ASCII font displayed as 6x8

DISP_FONT_TYPE_5x7_RUS: 5x7 Cyrillic font displayed as 6x8

DISP_FONT_TYPE_Xx8_SPEC: Xx8 special symbols

DISP_FONT_TYPE_LOGO: Xx16 special symbols for logos.

10.2.2 Character displaying

Disp_write_str and **Disp_write_char** are used to display characters on main display using current font and attributes. Cursor is moved to the end of the last written character.

```
Void   Disp_write_char(unsigned long ch);
Void   Disp_wirte_str(char * dstr);
```

Parameters:

ch : character to be displayed. 8 LSBs are valid.

dstr : string to be displayed

10.2.3 Clear screen

Macro **clr_scr()** cleans the matrix display portion of the LCD

10.2.4 Cursor setting

Macro **goto_xy()** controls the cursor.

```
goto_xy(unsigned char x, unsigned char y)
```

Parameters:

x : x coordinate of the cursor. Range is 0~127 in pixels.

y : y coordinate of the cursor. Range is 0~7 in ASCII lines (8 pixels)

Note:

No range check is performed by this macro so the user has to ensure the validity of the parameters.

10.3graphic displaying

10.3.1 clear graphic windows

macro **clr_win()** is used to clear a graphic window.

```
clr_win(
  unsigned char x1, unsigned char y1,
  unsigned char x2, unsigned char y2
 )
```

Parameters:

(x1,y1) and (x2,y2) are coordinates of two diagonal corners of the window to be cleared.

x1,x2 : Range is 0~127 in pixels

y1,y2 : Range is 0~63 in pixels.

10.3.2 pixel based graphics

10.3.2.1 pixel reading

macro **get_pixel()** is used to read pixel.

```
int   get_pixel(unsigned char x, unsigned char y)
```

Parameters:

x : x coordinate of the pixel to be read. Range is 0~127 in pixels

y : y coordinate of the pixel to be read. Range is 0~63 in pixels.

Return value:

0 if the pixel is blank(background) and 1 if the pixel is set (black)

Note:

No range check is performed by this macro so the user has to ensure the validity of the parameters.

10.3.2.2 pixel writing

macro **put_pixel()** is used to write pixel.

```
void put_pixel(
  unsigned char x,
  unsigned char y,
  unsigned char p,
  unsigned char mode
)
```

Parameters:

x : x coordinate of the pixel to be read. Range is 0~127 in pixels

y : y coordinate of the pixel to be read. Range is 0~63 in pixels.

p : pixel value to be written

mode : one of following flags for putting pixel

DISP_PUT_MODE_PUT : just write this pixel

DISP_PUT_MODE_OR: pixel is written as bit inclusive OR'ing p and the previous value of this location.

DISP_PUT_MODE_XOR: pixel is written as bit exclusive OR'ing p and the previous value of this location.

DISP_PUT_MODE_AND: pixel is written as bit AND'ing p and the

previous value of this location.

Note:

No range check is performed by this macro so the user has to ensure the validity of the parameters.

10.3.3 bitmap based graphics

10.3.3.1 bitmap structure

typ_BMP_rec shows how bitmap is stored in memory.

```
Typedef struct {
   unsigned int  X : 8;
   unsigned int  Y : 8;
   unsigned int  width : 8;
   unsigned int  height : 8;
   unsigned char data;
} typ_BMP_rec;
```

Members:

X,Y	: coordinate of the upper left corner of the bitmap.
width	: width of the bitmap. Will be clipped if X+width > 127
height	: height of th bitmap. Will be clipped if Y+height > 63
data	: a dummy field denoting the start of bitmap data

10.3.3.2 get bitmap

Disp_get_bmp reads a rectangular area of the main display and store it in memory.

```
int Disp_get_bmp(typ_BMP_rec * pb)
```

Parameters:

pb : pointer to buffer with type typ_BMP_rec . fields X, Y, width and height should be filled before calling.

Return value:

0 if successful ; -1 if failed (parameter error).

Examples:

```
#include <api.h>
/*   this demo reads a rectangular area (10,20)-(25,40) into buffer */
unsigned char buffer[1000];
void main(void) {
    typ_BMP_rec * pb = (typ_BMP_rec)buffer;
……
    pb->X = 10;
```

```
    pb->Y = 20;
    pb->width = 15;
    pb->height = 20;
    if (Disp_get_bmp(pb)) {
    /* your error handling routine */
    }
}
```

10.3.3.3 put bitmap

Disp_put_bmp write a rectangular area of the main display with data stored in memory. There're 4 putting modes, which could be specified by **Disp_set_bmp_put_mode**.

```
int Disp_put_bmp(typ_BMP_rec * pb)
```

Parameters:

pb : pointer to buffer with type typ_BMP_rec . fields X, Y should be filled before calling.

Return value:

0 if successful ; -1 if failed (parameter error).

Example:

```
#include <api.h>
/*  this demo writes an arrow to position(0,0) and (15,15)*/
unsigned char buffer[1000];
void main(void) {
    typ_BMP_rec * pb = (typ_BMP_rec)buffer;
……..
    pb->X = pb->Y = 0;
    pb->width = 5;
    pb->height = 8;
    buffer[4] = 0x1f;
    buffer[5] = 0x3e;
    buffer[6] = 0xfc;
    buffer[7] = 0xd8;
    buffer[8] = 0x10;
    Disp_put_bmp(pb);
    pb->X = pb->Y = 15;
    Disp_put_bmp(pb);

}
```

10.3.3.4 bitmap put mode setting

Disp_set_bmp_put_mode sets bitmap putting mode.

```
void Disp_set_bmp_put_mode(unsigned long pmode);
```

Parameters:

pmode : one of putting mode which are the same as in **put_pixel**

10.4 Icons

10.4.1 Customized icon setting

User can control certain icons by obtaining control of them from system. **Disp_icon_customize** is such a funtion.

```
void Disp_icon_customize(unsigned long icflag);
```

Parameters:

icflag : set by bit-OR'ing the following flags or simply use DISP_ICON_C_NONE to give up all user control.

DISP_ICON_C_CARD_SYMS	: user can control the card symbols
DISP_ICON_C_7SEGS	: user can control the four 7-SEG display including the semicolon
DISP_ICON_C_SPEAKERS	: user can control the speaker/spealer disable symbols
DISP_ICON_C_BELL	: user can can control the bell symbol
DISP_ICON_C_ANTENNA	: user can control the antenna symbol
DISP_ICON_C_CONNECT	: user can control the connected symbol
DISP_ICON_C_BATTERY	: user can control the battery level symbols

10.4.2 Set icons

Disp_icon_set is used to set the customized icons.

```
void Disp_icon_set(unsigned long * piflag);
```

Parameters:

piflag : a pointer to a paramblock which is 64 bits long. Each bit stands for a certain segment(even though not all these 64 segments is implemented). A set bit means the corresponding segment is lit up; a cleared bit means it's not.

10.4.3 Read icon settings

Disp_icon_read is used to read the current setting of the customized icons.

```
void Disp_icon_read(unsigned long * piflag);
```

Parameters:

piflag : same as in **Disp_icon_set**.

10.4.4 7-SEG display in icon area

10.5LCD contrast controlling

10.5.1 Read contrast setting

Macro **get_LCD_contrast** reads current LCD contrast setting.

```
char get_LCD_contrast
```

Return value

0~63 representing current LCD contrast level.

10.5.2 Change contrast

Macro **inc_LCD_contrast** and **dec_LCD_contrast** are used to change contrast setting.

```
inc_LCD_contrast
dec_LCD_contrast
```

11. RF Information

11.1data structures

11.1.1 status byte

typ_RFI_stat_flag is the description of the status byte of RF information decoder. Returned by **RFI_read_status**.

```
typedef union {
  struct {
    unsigned int n_a_7: 1;
    unsigned int syn_detected  : 1;
    unsigned int idle_mode     : 1;
    unsigned int preamble_mode : 1;
    unsigned int lock_mode : 1;
    unsigned int write_mode: 1;
    unsigned int receiving_mode : 1;
    unsigned int busy_flag : 1;
  } bits;
  unsigned char byte;
} typ_RFI_stat_flag;
```

Members:

bits : RFI decoder status byte intepreted as bit fields.(c.f. POCSAG standard & sm8212 user manual)

n_a_7	: N/A bit
syn_detected	: SYNC word detected
idle_mode	: RFI unit in idle state
preamble_mode	: RFI unit in preamble detecting state
lock_mode	: RFI unit in SYNC lock state

write_mode : RFI unit is ready to be programmed
receiving_mode : RFI unit is in data receiving state
busy_flag : RFI unit is busy with internal affairs, no time for talking

byte: status byte in raw data form

11.1.2 Subsystem configuration byte

typ_RFI_configuration_flag describes the configuration byte of RFI message receiving subsystem. Returned by **RFI_read_configuration**.

```
typedef union {
  struct {
    unsigned int n_a_7  : 1;
    unsigned int n_a_6  : 1;
    unsigned int n_a_5  : 1;
    unsigned int n_a_4  : 1;
    unsigned int msg_coming: 1;
    unsigned int msg_ended : 1;
    unsigned int word_mode : 1;
    unsigned int RFI_on    : 1;
  } bits;
  unsigned char byte;
} typ_RFI_configuration_flag;
```

Members:

bits : RFI subsystem configuration byte intepreted as bit fields. All the n_a_* bits denote n/a bits.

msg_coming : set if new RFI message is being received
msg_ended : set if a RFI message receiving session has ended
word_mode : set if RFI subsystem works in word mode
RFI_on : set if RFI subsystem is working

byte: configuration byte in raw data form

11.1.3 message data buffer

typ_RFI_message is the description of the message data returned when RFI unit is working in message mode.

```
typedef {
   unsigned int isAlphanumeric : 1;
   unsigned int n_a_bits : 2;
   unsigned int addr : 3;
   unsigned int func : 2;
   unsigned char data;
} typ_RFI_message
```

Members: (c.f. POCSAG standard)

isAlphanumeric	: set if current message is alphanumeric.
n_a_bits	: N/A
addr	: address on which current message is received
func	: fuction code of current message.
data	: dummy field denoting the start of message data.

Data format of message data:

```
typdef {
   unsigned int err   : 1;
   unsigned int data :7
} typ_RFI_msgdata;
```

err is set if transmission error occurred when receiving this byte; data is the data received, which will be 0~15 for numeric message and 0~127 for alphanumeric message

11.2 Initialization

RFI_init is the main switch of the RFI unit.

```
void RFI_init (unsigned long * pFB)
```

Parameters:

pFB : pointer to RFI decoder parameter data block. If it's NULL, the RFI unit will be turned off so no RFI message will be received, and the power consumption of the whole system would be reduced significantly.(c.f. sm8212 manual)

11.3 message reading

RFI_read_msg is the subroutine to read a received RFI message. It's highly recommended that it's called in RF_new message dealer when RFI unit is in message mode. And it sould not be used when RFI unit is working in word mode.

```
unsigned long RFI_read_msg(typ_RFI_message * pBuff);
```

Parameters:

pBuffer : pointer to data buffer which the received message will be stored in.

Returned value:

Length of received message data.

11.4 status reading

RFI_read_status is the subroutine to read the status byte of the RFI information decoder.

```
unsigned short RFI_read_status ( void ) ;
```

Returned value:

RFI decoder status byte which could be intepreted by **typ_RFI_stat_flag**.

11.5configuration reading

RFI_read_configuration is the subroutine to read the configuration byte of the RFI message receiving subsystem.

```
unsigned short RFI_read_configuration ( void ) ;
```

Returned value:

RFI message receiving subsystem configuration byte which could be intepreted by **typ_RFI_configuration_flag**.

11.6register RF information callback routine

RFI_register_call_back set a callback routine which is called each time a RFI decoder data ready interrupt occurs. It should not be used when RFI unit works in message mode.

```
void RFI_register_call_back(
  void (* CBroutine)(unsigned long msgword)
);
```

Parameters:

CBroutine: callback routine name. The parameter of this routine is the dataword returned by RFI decoder.(c.f. sm8212 manual)

Note:

Using this funtion and RFI word mode is not recommended unless the programmer is experienced with pager programming.

11.7set receiving method

RFI_set_msg_level determines what mode the RFI is working in.

```
void       RFI_set_msg_level(unsigned long msgmode);
```

Parameters:

msgmode: RFI_MSG_LEVEL_MSG for message mode and RFI_MSG_LEVEL_WORD for word mode.

12. Download

enter_download_mode is used to enter download mode.

```
void enter_download_mode (void)
```

Note:

This is a one-way subroutine that never returns, the only way to exit download mode is to reset the system by pressing reset button or a reset command received from UART.

13. Flash memory

There are 2 low level functions to access flash memory. It's highly recommended that user should not call these functions directly. Any invalid address accessing will cause unwanted exceptions.

13.1erasing block

FLASH_erase_block erases a 64kbyte block at a given address in flash memory

```
int FLASH_erase_block(void * blockaddress)
```

Parameters:

blockaddress : pointer to denote the address of the block to be erased. Recommended to be aligned to 64k boundary.

Return value:

0x80 if succeeded, else if error occurred .

13.2 writing data

FLASH_write_record is used to write a record (data with certain length) to a specified flash memory area.

```
int FLASH_write_record(FLASH_wr_param * pfwp)
```

Parameters:

pfwp : pointer to a flash write param block which is defined as

```
typedef {
    void * ptr_buffer;
    void * ptr_FLASH_addr;
    unsigned long data_length
} FLASH_wr_param
```

Members of FLASH_wr_param:

ptr_buffer : pointer to the data buffer containing data to write

ptr_FLASH_addr: pointer to destination address in flash memory

data_length : numbers of data to write , in characters.

Return value:

0x80 if succeeded, else if error occurred .

Note:

Data written should not cross the block boundary.