Terminology: bluetooth module - the whole module which includes HM11 and microcontroller.

The package send from the device to android phone is not yet define, I'm currently sending a simple data payload which starts and ends with '|', the code in the bluetooth module looks like:

```
if (crc[0] == rMessage.crc[0] && crc[1] == rMessage.crc[1])
{
    if (g_HM11_Conn)
    {
        HM11_SendByte('|');
        for (uint8_t i = 0; i < rMessage.payload_length; i++)
        {
            HM11_SendByte(rMessage.payload[i]);
        }
        HM11_SendByte('|');
    }
}
```

After receiving data from other modules rMessage the crc is checked and if it is corret the received message is broadcasted using bluetooth (g_HM11_Conn is set to 1 if the HM11 is connected to any bluetooth device).

From the point of view bluetooth module it does not now which it is receiving from other modules - it is just broadcasting it.

The message consists of device identifier (REG_MODULE_ID_[1:3] 3 byte data which defines every possible module), module address which is used to send data to selected device (its like i2c address) and the rest of the data is a payload.

```
typedef enum {
    REG_MODULE_ID_1             = 0,
    REG_MODULE_ID_2             = 1,
    REG_MODULE_ID_3             = 2,

    REG_MODULE_ADDR             = 3,

    REG_SI7013_HUM_HIGH         = 4,
    REG_SI7013_HUM_LOW          = 5,

    REG_SI7013_TEMP_HIGH        = 6,
    REG_SI7013_TEMP_LOW         = 7,

    REG_SI114x_ALS_HIGH         = 8,
    REG_SI114x_ALS_LOW          = 9,

    REG_SI114x_UV_HIGH          = 10,
    REG_SI114x_UV_LOW           = 11,

    REG_BMP180_TEMP_HIGH        = 12,
    REG_BMP180_TEMP_LOW         = 13,

    REG_BMP180_PRESS_XHIGH      = 14,
    REG_BMP180_PRESS_HIGH       = 15,
    REG_BMP180_PRESS_LOW        = 16,

    REG_HMC5883L_X_HIGH         = 17,
    REG_HMC5883L_X_LOW          = 18,
    REG_HMC5883L_Y_HIGH         = 19,
    REG_HMC5883L_Y_LOW          = 20,
    REG_HMC5883L_Z_HIGH         = 21,
    REG_HMC5883L_Z_LOW          = 22,
```

Description how to calculated real data from the message received from a module is written in moduleDescription.xml:

```
<Modules>
    <Module id="19779" name="Bluetooth module">
        <Variable name="test"      equation="d[10]+d[11]/100"></Variable>
    </Module>
    <Module id="4543310" name="Environmental sensor">
        <Variable name="Humidity"        equation="d[4]+d[5]/100"></Variable>
        <Variable name="Temperature"     equation="d[6]+d[7]/100"></Variable>
        <Variable name="Ambient light"   equation="d[8]*256+d[9]"></Variable>
        <Variable name="UV index"        equation="d[10]*256+d[11]"></Variable>
        <Variable name="Pressure"        equation="d[14]*1000+d[15]*10+d[16]/10"></Variable>
        <Variable name="BMP085 temp"     equation="d[12]+d[13]/10"></Variable>
        <Variable name="Compass X"       equation="d[17]*256+d[18]"></Variable>
        <Variable name="Compass Y"       equation="d[19]*256+d[20]"></Variable>
        <Variable name="Compass Z"       equation="d[21]*256+d[22]"></Variable>
        <Variable name="Accel X"         equation="d[23]*256+d[24]"></Variable>
        <Variable name="Accel Y"         equation="d[25]*256+d[26]"></Variable>
        <Variable name="Accel Z"         equation="d[27]*256+d[28]"></Variable>
        <Variable name="Gyro X"          equation="d[29]*256+d[30]"></Variable>
        <Variable name="Gyro Y"          equation="d[31]*256+d[32]"></Variable>
        <Variable name="Gyro Z"          equation="d[33]*256+d[34]"></Variable>
        <Variable name="Pedometer count" equation="d[35]*16777216+d[36]*65536+d[37]*256+d[38]"></Variable>
        <Variable name="Pedometer time"  equation="d[39]*16777216+d[40]*65536+d[41]*256+d[42]"></Variable>
    </Module>
    <Module id="4997699" name="LiPol controller">
        <Variable name="Battery voltage"        equation="(d[4]*256+d[5])/65535*6"></Variable>
        <Variable name="Battery temperature"    equation="(d[6]*256+d[7])/65535*600-273.15"></Variable>
        <Variable name="Battery charge"        equation="(65535-(d[8]*256+d[9]))*85/10"></Variable>
        <Variable name="Battery discharge rate" equation="d[10]*256+d[11]"></Variable>
    </Module>
    <Module id="5270628" name="Power LED controller">
        <Variable name="LED on"                 equation="(d[4]%2"></Variable>
        <Variable name="Torch current [mA]"     equation="(((d[4]-d[4]%4)/4)%8)*25+25"></Variable>
        <Variable name="Flash current [mA]"     equation="(((d[5]-d[5]%4)/4)%32)*50+300"></Variable>
        <Variable name="Flash time [ms]"        equation="d[6]*100+100"></Variable>
    </Module>
</Modules>
```

This file is store on the Internet (http://student.agh.edu.pl/~dkalicki/moduleDescription.xml) which should be download on the application start and store in the internal memory for the future need if the internet connection is not available.

Module id from xml file is the same as REG_MODULE_ID_[1:3].
d[X] stands for data at position X received from device.
Variable name stands for a variable identifier which the producent selected.
The equation is an instruction how to calculate the real parameter and should be done before printing the variable on the screen.
Additionaly there should be an another field name - units which will store the variable units for example mA, lux, % etc. <Variable name=”Torch current” unit=”mA” equation=”.....”>

Example data:
Consider system with:

1. Bluetooth module - master
2. Sensor module - periodically sends data
3. Led module - periodically sends data and is capable of receiving commands from the user.
4. LiPol module - periodically sends data
5. User interface module - sends interrupt base data and receives long complex data packages.

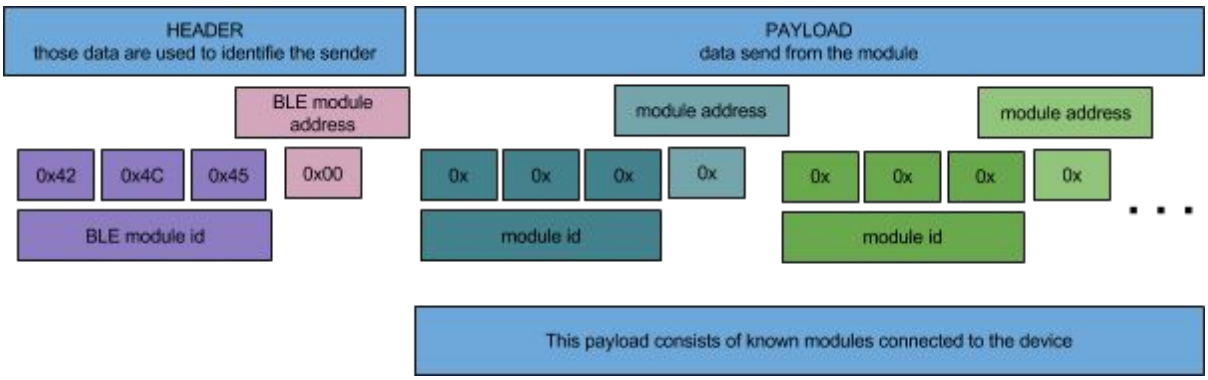
All send and received bytes are stored in InternalRegisters.h file for every module (currently with the exception of Led module which has them implemented in main.c function and bluetooth module).

Currently no CRC mechanism is implemented in the message

Data send from the bluetooth module to Android software consists of a simple (not changed) broadcast command.
To send data to selected module from Android Software a module address needs to be added in front - in this example 0x06 for sensor module, 0x1E for led module, 0x05 for lipol module, 0x02 for user interface module. The rest is a simple data payload as described bellow.

Bluetooth module

This module sends it own packages like:



For example for this configuration it will be:

0x42 0x4C 0x45 0x01 0x45 0x53 0x45 0x06 0x50 0x6C 0x64 0x1E 0x4C 0x42 0x43 0x05 0x4C 0x68 0x74 0x02

Header Sensor module code Led module LiPol module User interface module

Sensor module

This module broadcasts its state every 500ms.
Bytes send from sensor module are organized in:

```
typedef enum {
   REG_MODULE_ID_1          = 0, 0x45
   REG_MODULE_ID_2          = 1, 0x53
   REG_MODULE_ID_3          = 2, 0x45

   REG_MODULE_ADDR          = 3, 0x06

/* Si70xx humidity   ->   e.g. hum=56.25%  [REG_SI7013_HUM_HIGH]=56 [REG_SI7013_HUM_LOW]=25 */
   REG_SI7013_HUM_HIGH       = 4,
   REG_SI7013_HUM_LOW        = 5,

/* Si70xx temperature ->  e.g. temp=25.23*C [REG_SI7013_TEMP_HIGH]=25 [REG_SI7013_TEMP_LOW]=23 */
   REG_SI7013_TEMP_HIGH      = 6,
   REG_SI7013_TEMP_LOW       = 7,

/* Si114x ambient light in lux ->  [REG_SI114x_ALS_HIGH] -> ambient light High byte  [REG_SI114x_ALS_LOW] ->
ambient light Low byte */
   REG_SI114x_ALS_HIGH       = 8,
   REG_SI114x_ALS_LOW        = 9,

/* Si114x Uv_index*100 stored in binary form */
   REG_SI114x_UV_HIGH        = 10,
   REG_SI114x_UV_LOW         = 11,

/* BMP180 temperature if it is measured, else it should be 0xFFF -> e.g. temp=25.23*C
[REG_BMP180_TEMP_HIGH]=25 [REG_BMP180_TEMP_LOW]=23 */
   REG_BMP180_TEMP_HIGH      = 12,
   REG_BMP180_TEMP_LOW       = 13,

/* BMP180 pressure  -> e.g. pressure=101399.8 [REG_BMP180_PRESS_XHIGH]=101
[REG_BMP180_PRESS_HIGH]=39 [REG_BMP180_PRESS_LOW]=98 */
   REG_BMP180_PRESS_XHIGH    = 14,
   REG_BMP180_PRESS_HIGH     = 15,
   REG_BMP180_PRESS_LOW      = 16,

/* Binary stored data from the compass */
   REG_HMC5883L_X_HIGH       = 17,
   REG_HMC5883L_X_LOW        = 18,
   REG_HMC5883L_Y_HIGH       = 19,
   REG_HMC5883L_Y_LOW        = 20,
   REG_HMC5883L_Z_HIGH       = 21,
   REG_HMC5883L_Z_LOW        = 22,

/* Binary stored data from accelerometer */
   REG_MPU6050_X_ACCEL_HIGH  = 23,
   REG_MPU6050_X_ACCEL_LOW   = 24,
   REG_MPU6050_Y_ACCEL_HIGH  = 25,
   REG_MPU6050_Y_ACCEL_LOW   = 26,
   REG_MPU6050_Z_ACCEL_HIGH  = 27,
   REG_MPU6050_Z_ACCEL_LOW   = 28,

/* Binary stored data from gyroscope */
   REG_MPU6050_X_GYRO_HIGH   = 29,
   REG_MPU6050_X_GYRO_LOW    = 30,
   REG_MPU6050_Y_GYRO_HIGH   = 31,
   REG_MPU6050_Y_GYRO_LOW    = 32,
   REG_MPU6050_Z_GYRO_HIGH   = 33,
   REG_MPU6050_Z_GYRO_LOW    = 34,

/* Binary stored data indicating step count */
   REG_MPU6050_X_PEDO_COUNT_1  = 35,
   REG_MPU6050_X_PEDO_COUNT_2  = 36,
```

```
    REG_MPU6050_X_PEDO_COUNT_3  = 37,
    REG_MPU6050_X_PEDO_COUNT_4  = 38,

/* Binary stored data indicating step time in ms */
    REG_MPU6050_X_PEDO_TIME_1   = 39,
    REG_MPU6050_X_PEDO_TIME_2   = 40,
    REG_MPU6050_X_PEDO_TIME_3   = 41,
    REG_MPU6050_X_PEDO_TIME_4   = 42,

/* Binary stored data from RGB sensor not normalized! */
    REG_MAX4400X_CLEAR_HIGH     = 43,
    REG_MAX4400X_CLEAR_LOW      = 44,
    REG_MAX4400X_RED_HIGH       = 45,
    REG_MAX4400X_RED_LOW        = 46,
    REG_MAX4400X_GREEN_HIGH     = 47,
    REG_MAX4400X_GREEN_LOW      = 48,
    REG_MAX4400X_BLUE_HIGH      = 49,
    REG_MAX4400X_BLUE_LOW       = 50,
    REG_MAX4400X_IR_HIGH        = 51,
    REG_MAX4400X_IR_LOW         = 52,
    REG_MAX4400X_TEMP_HIGH      = 53,
    REG_MAX4400X_TEMP_LOW       = 54,

/* This will be removed in later revision - it will be change to interrupt based commands */
    REG_MPU6050_X_TAP_EVENT_1   = 55,
    REG_MPU6050_X_TAP_EVENT_2   = 56,
    REG_MPU6050_X_TAP_EVENT_3   = 57,
    REG_MPU6050_X_TAP_EVENT_4   = 58,
    REG_MPU6050_X_TAP_EVENT_5   = 59,
    REG_MPU6050_X_TAP_EVENT_6   = 60,
    REG_MPU6050_X_TAP_EVENT_7   = 61,
    REG_MPU6050_X_TAP_EVENT_8   = 62,
    REG_MPU6050_X_TAP_EVENT_9   = 63,
    REG_MPU6050_X_TAP_EVENT_10  = 64
} Sensor_Module_Broadcast_Register_t;
```

Led module

This module receives command to change the led brightness or the time in which the module broadcast its state.

Received data is organized in:

```
/* This is a layout which is parsed when data is received from the master by "Write command"
 * Setting any value to 0xFF means the value should not change */
typedef enum
{

   /* Bit [0:1] -> set the on (01b) off(00b) the torch mode value 3 (11b) does not change
    * the actual state of the torch
    * Bit [2:7] -> set the torch current: 0=25mA, 1=50mA, ... 7=200mA
    * Setting this value to 0x3F does not change the current setting */
   REG_TORCH_STATE_RECEIVE        = 0,

   /* Bit [0:1] -> set the on (01b) the flash mode value 3 (11b) does not change
    * the actual state of the flash
    * Bit [2:7] -> set the torch current: 0=300mA, 1=350mA, ... 24=1500mA
    * Setting this value to 0x3F does not change the current setting */
   REG_FLASH_STATE_RECEIVE        = 1,

   /* Set the flash on timer: 0=100ms, 1=200ms, ... 15=1600ms
    * Setting this value to 0xFF does not change the current setting */
   REG_FLASH_TIME_RECEIVE         = 2,

   /* Set to 0x00 to disable the "Broadcast" commands with the led module state
    * Set to X to send a "Broadcast" command every x/2 seconds. By default this value is set to 0x01 - 0.5
second.
    *      Max value 0xFE - 127 seconds.
    * Set to 0xFF to send a single "Broadcast" command from this module */
   REG_MOD_COM_SETTINGS_RECEIVE    = 3
} Led_Module_Receive_Register_t;
```

Led module is broadcasting it state in predefined time intervals (by default 1s):

```
/* Those register are send to the master by a "Broadcast" command */
typedef enum
{
   /* Module id unique identifier */
   REG_MODULE_ID_1          = 0, 0x50
   REG_MODULE_ID_2          = 1, 0x6C
   REG_MODULE_ID_3          = 2, 0x64

   REG_MODULE_ADDR          = 3, 0x1E

   /* Bit [0:1] -> the on (01b) off(00b) the torch mode
    * Bit [2:7] -> the torch current: 0=25mA, 1=50mA, ... 7=200mA */
   REG_TORCH_STATE           = 4

   /* Bit [0:1] -> 0xFF because Flash state cannot be easily checked.
    * Bit [2:7] -> the torch current: 0=300mA, 1=350mA, ... 24=1500mA */
   REG_FLASH_STATE           = 5

   /* the flash on timer: 0=100ms, 1=200ms, ... 15=1600ms */
   REG_FLASH_TIME            = 6,

   /* This value is increased every time a new package is broadcast */
   REG_TEST_COUNTER          = 7
} Led_Module_Broadcast_Register_t;
```

LiPol module

This device broadcasts its state (by default every 1 second) using a structure defined as:

```
/* Those register are send to the master by a "Broadcast" command */
typedef enum {
    REG_MODULE_ID_1           = 0,
    REG_MODULE_ID_2           = 1,
    REG_MODULE_ID_3           = 2,

    REG_MODULE_ADDR           = 3,

    /* Binary values */
    REG_BAT_VOLT_1            = 4,
    REG_BAT_VOLT_2            = 5,
    REG_BAT_TEMP_1            = 6,
    REG_BAT_TEMP_2            = 7,
    REG_BAT_CHRG_1            = 8,
    REG_BAT_CHRG_2            = 9,
    REG_BAT_DISCHR_1          = 10,
    REG_BAT_DISCHR_2          = 11,

    /* ADP5063 internal state - refer to the ic datasheet for more informations */
    REG_ADP5063_CHARGER_ST_1    = 12,
    REG_ADP5063_CHARGER_ST_2    = 13,

    /* Bit 0 is set to 0 if LTC4411 forward diode mode is enable (CTR pin settings)
     *              1 if LTC4411 forward diode mode is disable (CTR pin settings)
     * Bit 1 is set to 1 if LTC4411 STATE pin is high
     *              0 if LTC4411 STATE pin is low */
    REG_LTC4411_STATE         = 14

} LiPol_Module_Broadcast_Register_t;
```

The module can receive commands to change the broadcast timer or to change the LTC4411 state.

```
/* This is a layout which is parsed when data is received from the master by "Write command"
 * Setting any value to 0xFF means the value should not change */
typedef enum
{
    /* Bit [0:1]
     *      Set to 0 to disable LTC4411 forward diode
     *      Set to 1 to enable LTC4411 forward diode
     *      Setting to 0x02 or 0x03 does not change the current settings
     * Bit [2:3]
     *      Set to 0 to disable the charging process
     *      Set to 1 to enable the charging process
     *      Setting to 0x02 or 0x03 does not change the current settings
     *      */
    REG_MODULE_STATE_RECEIVE      =   0,

    /* Set to 0x00 to disable the "Broadcast" commands with the lipol module state
        * Set to X to send a "Broadcast" command every x/2 seconds. By default this value is set to 0x01 -
0.5 second.
        *      Max value 0xFE - 127 seconds.
        * Set to 0xFF to send a single "Broadcast" command from this module */
    REG_MOD_COM_SETTINGS_RECEIVE    = 1
} LiPol_Module_Receive_Register_t;
```

User interface module

This module can receive a big configuration data for example it blinking rate can be set independently for every led, because of such a large data package an additional addressing was implemented.

The first byte of the write command is used to set the starting byte.

For example to set REG_ADP8866_SCxON (number 0x04) and REG_ADP8866_SCxOFF_1 (number 0x05) without channing previous bytes a write commands starting with 0x04 should be send as: 0x04 [value to set the selected byte] [next byte settings etc]

For example to turn on the vibration motor send:
0x31(starting byte) 0x10 (this will turn on the default vibration mode 0x10 times).

Additionally with a capacitive touch interrupt a module sends a broadcast message as:

```
/* Those register are send to the master by a "Broadcast" command */
typedef enum
{
   /* Module type unique identifier */
   REG_MODULE_ID_1          = 0,
   REG_MODULE_ID_2          = 1,
   REG_MODULE_ID_3          = 2,

  REG_MODULE_ADDR           = 3,

/* Binary stored touch position - every bit corresponds to a selected pad, for example the closes to the left is
0x0001 (bit 0 set to 1) and the closest to the right is 0x0200 (bit 9 set to 1) */
   REG_TOUCH_INT_1          = 4,
   REG_TOUCH_INT_2          = 5
} User_Interface_Module_Broadcast_Register_t;
```

The register map for receiving packages as defined as:

```
/* This is a layout which is parsed when data is received from the master by "Write command"
 * Setting any value to 0xFF means the value should not change */
typedef enum
{
   /* The master can start writing to registers from a specific
    * index set by REG_START_IDX. */
   REG_START_IDX            = 0x00,

   /* Set led bit to 1 to enable the led,
    * Set led bit to 0 to disable the led,
    * Setting led bit to 2 or 3 does not change its current state
    *
    * REG_ADP8866_LED_ONOFF_3[3:5] additionally sets the Led Fade patterns:
    *  000b = square law DAC, linear time steps.
    *  001b = square law DAC, linear time steps.
    *  010b = square law DAC, nonlinear time steps (Cubic 10).
    *  011b = square law DAC, nonlinear time steps (Cubic 11)
    *  1xxb = does not change the Led Fade pattern settings
    *
    * REG_ADP8866_LED_ONOFF_1:
    *   9     8     7    6     -> Led number
    *  # # | # # | # # | # # |  -> led bit
    *
    *  REG_ADP8866_LED_ONOFF_2:
    *   5     4     3    2     -> Led number
    *  # # | # # | # # | # # |  -> led bit
    *
    *  REG_ADP8866_LED_ONOFF_3:
    *   1    Led Fade pattern     unused
    *  # # |   #    #     #   |  # # # |  -> led bit
    */
```

```
REG_ADP8866_LED_ONOFF_1    = 0x01,
REG_ADP8866_LED_ONOFF_2    = 0x02,
REG_ADP8866_LED_ONOFF_3    = 0x03,

/* Set the SCxON timer - 0=0s, 1=0.05s, 2=0.10s, ... 16=0.75s
 * Setting this reg to 0x1F does not affect the SCxON timer
 *
 * REG_ADP8866_SCxON:
 *     SCxON     SCxOFF5
 *   # # # # # | # # # |
 */
REG_ADP8866_SCxON          = 0x04,

/* Set the SCx off time for led 5 - 1:
 *   000b = off time disabled
 *   001b = 0.6 sec.
 *   010b = 1.2 sec.
 *   011b = 1.8 sec
 *   1xxb = does not change the current settings
 *
 *   REG_ADP8866_SCxOFF_1:
 *    4       3     unused
 *   # # # | # # # |  # #  |
 *
 *   REG_ADP8866_SCxOFF_2:
 *    2       1     unused
 *   # # # | # # # |  # #  |
 */
REG_ADP8866_SCxOFF_1       = 0x05,
REG_ADP8866_SCxOFF_2       = 0x06,

/* Set the Led 6 - 9 SCxOFF timer:
 * 0=disable, 1=0s, 2=0.1s, 3=0.2s, ... 126=12.5s, 127=off.
 * Setting to 255 (0xFF) does not change the SCxOFF settings.
 */
REG_ADP8866_SCxOFF6        = 0x07,
REG_ADP8866_SCxOFF7        = 0x08,
REG_ADP8866_SCxOFF8        = 0x09,
REG_ADP8866_SCxOFF9        = 0x0A,

/* Set Fade out time: 0=disable, 1=0.05s, 2=0.10s, ... 16=1.75s
 * Setting to 0xFF does not change the SCFO value
 */
REG_ADP8866_FADE_SCFO      = 0x0B,
/* Set Fade out time: 0=disable, 1=0.05s, 2=0.10s, ... 16=1.75s
 * Setting to 0xFF does not change the SCFI value
 */
REG_ADP8866_FADE_SCFI      = 0x0C,

/* Setting to 0xFF does not change the current settings */
REG_ADP8866_LED1_CURRENT   = 0x0D,
REG_ADP8866_LED2_CURRENT   = 0x0E,
REG_ADP8866_LED3_CURRENT   = 0x0F,
REG_ADP8866_LED4_CURRENT   = 0x10,
REG_ADP8866_LED5_CURRENT   = 0x11,
REG_ADP8866_LED6_CURRENT   = 0x12,
REG_ADP8866_LED7_CURRENT   = 0x13,
REG_ADP8866_LED8_CURRENT   = 0x14,
REG_ADP8866_LED9_CURRENT   = 0x15,

/* Set led bits to 0 to disable heartbeat mode,
 * Set led bits to 1 to enable heartbeat mode,
 * Setting led bits to 2 or 3 does not change the current settings
 *
 * REG_ADP8866_HEARTBEAT_EN:
 *  9     8     7     6
 * # # | # # | # # | # # |
 */
REG_ADP8866_HEARTBEAT_EN   = 0x16,
```

```
    /* Setting to 0xFF does not change the current settings */
    REG_ADP8866_HB_LED6_CURR   = 0x17,
    REG_ADP8866_HB_LED7_CURR   = 0x18,
    REG_ADP8866_HB_LED8_CURR   = 0x19,
    REG_ADP8866_HB_LED9_CURR   = 0x1A,

    /* Setting to 0xFF does not change the current settings */
    REG_ADP8866_HB_LED6_SCxOFF = 0x1B,
    REG_ADP8866_HB_LED7_SCxOFF = 0x1C,
    REG_ADP8866_HB_LED8_SCxOFF = 0x1D,
    REG_ADP8866_HB_LED9_SCxOFF = 0x1E,

    /* Set the SCxON timer - 0=0s, 1=0.05s, 2=0.10s, ... 16=0.75s
     * Setting this reg to 0x1F does not affect the SCxON timer
     *
     * REG_ADP8866_HB_SCxON:
     *    unused      SCxON
     *   # # #  | # # # # # |
     */
    REG_ADP8866_HB_SCxON       = 0x1F,

    /* Setting to 0xFF does not change the current settings */
    REG_ADP8866_LED6_DELAY     = 0x20,
    REG_ADP8866_LED7_DELAY     = 0x21,
    REG_ADP8866_LED8_DELAY     = 0x22,
    REG_ADP8866_LED9_DELAY     = 0x23,



    /* This register is used to set actually used haptic pattern
     * Setting to 0xFF does not change current settings
     * For the pattern list with corresponding functions see
     *      BD6210_SetPatternBasedOnInternalRegister()
     * Set this value to 0xFE to use user defined pattern
     */
    REG_BD6210_SET_PATTERN      = 0x30,
    /* Registers REG_BD6210_START_PATTERN_[1:2] are used to set the
     *      number of repeat cycles of the haptic pattern.
     *      If both registers are given by the "Write" command
     *      the maximum number of cycles equals 0xFFFD.
     *      If only the first register is provided (the message
     *      ends after it) its value will be used as a single
     *      byte - value 0x00 disables actual played pattern
     *          -       0xFF does not affect current value
     *          -       0xFE pattern is playing indefinitely
     *          -       other values are used to set the repeat
     *                    cycles number.
     *  ------------------------------------------------------------
     * Set to 0x0000 to disable actually played pattern
     * Setting to 0xFFFF does not change current settings
     * Set to 0xFFFE to play the pattern indefinitely
     * Other values will set the repeat number of cycles
     */
    REG_BD6210_START_PATTERN_1   = 0x31,
    REG_BD6210_START_PATTERN_2   = 0x32

} User_Inteface_Module_Receive_Register_t;
```

Some examples of configuring this module a module address is already added [0x02] so the second byte is the starting byte:

```
 * Test values to check if the "Write" to internal regs parsing is
 * working correctly
 *
 * 02 01 00 00 00 - all leds turn off
 *
 * 02 01 55 55 55 - all leds turn on
```

```
*
* 02 04 7B - blinking led5
*
* 02 04 7B 44 - led5,4,3 blinking in differente rates
*
* 02 04 7b 6c 6c - led 5,4,3,2,1 blinking at the same rate. Timer needs to be
*                  clean to makes sure that all leds starts blinking at the
*                  same time. Clean command: 02 05 00 00 00
*
* 02 04 78 | 02 07 0A - first command sets the SCxON timer to maximum
*                       and the second sets the Led6 blinking rate
*
* 02 04 78 |       - first command sets the SCxON timer to maximum
* 02 07 0a 0a 0a 0a- all leds6-9 blinking at the same rate.
*
* 02 04 78 |       - first command sets the SCxON timer to maximum
* 02 07 0a 0a 0a | - all leds6-9 blinking at the same rate.
* 02 0b 0f         - fade out set to maximum for led6-9
* 02 0c 0f         - fade in set to maximum
*
* 02 12 40 20 10 30 - led6-9 set with different currents.
*
* Heartbeat test:
* 02 12 40 20 10 30 - initial led settings
* 02 04 78
* 02 07 0a 0a 0a 0a
* 02 16 55          - heartbeat mode enable for all leds.
*       7f 7f 7f 7f - leds current set to maximum
*       0a 0a 0a 0a - timer off rate
*       05          - timer on rate
*       64          - delay between heartbeat mode
*/
```