HP-6 血压心率模块通讯协议

密级程度：秘密

版本修改记录

修改日期	版本	修改内容
2016/8/31	V1.0	协议初版，调试使用
2016/9/3	V1.1	修改获取 ADC 数据指令，添加 id 字段

目录

本协议针对 HP-6 适用于 IIC 接口通讯，7bite 硬件地址：0x66；IIC 通讯速率为 100K。

1. 开启血压测量

发送开启测量格式如表 1-1； 返回数据格式如表 1-2.

表 1-1

Header	CMD	Data					Check Sum
Byte 0~3	4	5	6	7	8	9~21	22~23
0xc8d7b6a5	0x90	0x01	测量模式	高压	低压	保留	低位在前

Byte5：参数 0x01~打开血压测量

Byte 6：测量模式，0~通用测量模式，1~私人测量模式（需要设置高压和低压）

Byte 7：高压，80~210（需要设置为私人模式才有效）

Byte 8：低压，45~180（需要设置为私人模式才有效）

表 1-2

Header	CMD	Data			Check Sum
Byte 0~3	4	5	6	6~21	22~23
0xc8d7b6a5	0x90	0x01	0x00~0x01	保留	低位在前

Byte 5：参数 0x01

Byte 6：应答，0~开启失败，1~开启成功

注意：开启成功后，发送表 3-1 指令读取血压测量信息

2. 关闭血压测量

发送关闭测量信息格式如表 2-1；返回数据格式如表 2-2。

表 2-1：

Header	CMD	Data		Check Sum
Byte 0~3	4	5	6~21	22~23
0xc8d7b6a5	0x90	0x00	保留	

Byte 5：参数 0x00

表 2-2：

Header	CMD	Data			Check Sum
Byte 0~3	4	5	6	6~21	22~23
0xc8d7b6a5	0x90	0x00	0x00~0x01	保留	低位在前

Byte 5：参数 0x00

Byte 6：应答，0~关闭失败，1~关闭成功

3. 读取血压测量信息

发送开启测量第 2.56 秒开始，每隔 1.28 秒可以读取一次测量血压信息，发送信息格式如表 3-1.返回信息格式如表 3-2：

表 3-1

Header	CMD	Data		Check Sum
Byte 0~3	4	5	6~21	22~23
0xc8d7b6a5	0x90	0x02	保留	

表 3-2

Header	CMD	Data											Check Sum
Byte 0~3	4	5	6	7	8~9	10	11	12	13~14	15	16~20	21	22~23
0xc8d7b6a5	0x90	0x02	0x00/0x01	测量状态		高压	低压	心率		测量模式		放大倍数	低位在前

Byte 5： 参数 0x02

Byte 6： 应答，0~血压测量未开启，1~血压测量已开启

Byte 7： 测量状态，0~测量中，1~测试完成，2~测试失败

Byte 10：高压测量结果：80~210

Byte 11：低压测量结果：45~180

Byte 12：心率测量结果：30～200

Byte 15：测量模式

Byte 21：传感器信号放大倍数，供测试使用

注：测量完成或者测试失败需要程序去控制关闭测试

4. 开启心率测量

发送开启测量格式如表 4-1； 返回数据格式如表 4-2.

表 4-1

Header	CMD	Data		Check Sum
Byte 0~3	4	5	6~21	22~23
0xc8d7b6a5	0xD0	0x01	保留	低位在前

Byte5：参数 0x01~打开心率测量

表 4-2

Header	CMD	Data			Check Sum
Byte 0~3	4	5	6	6~21	22~23
0xc8d7b6a5	0xD0	0x01	0x00~0x01	保留	低位在前

Byte 5：参数 0x01

Byte 6：应答，0~开启失败，1~开启成功

注意：开启成功后，发送表 6-1 指令读取心率测量信息

5. 关闭心率测量

发送关闭测量信息格式如表 5-1；返回数据格式如表 5-2。

表 5-1：

Header	CMD	Data		Check Sum
Byte 0~3	4	5	6~21	22~23
0xc8d7b6a5	0xD0	0x00	保留	

Byte 5：参数 0x00

表 5-2：

Header	CMD	Data			Check Sum
Byte 0~3	4	5	6	6~21	22~23
0xc8d7b6a5	0xD0	0x00	0x00~0x01	保留	低位在前

Byte 5：参数 0x00

Byte 6：应答，0~关闭失败，1~关闭成功

6. 读取心率测量信息

发送开启测量第 5 秒开始，每隔 1.25 秒可以读取一次测量心率信息，发送

信息格式如表 6-1.返回信息格式如表 6-2：

表 6-1

Header	CMD	Data		Check Sum
Byte 0~3	4	5	6~21	22~23
0xc8d7b6a5	0x90	0x02	保留	

表 6-2

Header	CMD	Data					Check Sum
Byte 0~3	4	5	6	7	8	9~21	22~23
0xc8d7b6a5	0xD0	0x02	0x00~0x01	心率结果	放大倍数	保留	

Byte 5： 参数 0x02

Byte 6： 应答，0~心率测量未开启，1~心率测量已开启

Byte 7： 心率结果

Byte 8： 放大倍数，测试使用

7. 读取 ADC 数据指令

血压 AD 数据每次更新大小为 0.5K，分为 32 包，每次读取 16byte；心率 AD 数据每次更新大小为 100byte，分为 7 包，0～5 包每次读取 16byte，第 6 包读取 4byte；发送信息格式如表 7-1.返回信息格式如表 7-2：

表 7-1：

Header	CMD	Data			Check Sum
Byte 0~3	4	5	6	7~21	22~23
0xc8d7b6a5	0x91	包号（0~31/0xFF）	区别 ID	保留	低位在前

Byte 6：区别 ID 值～1—255，同一周期 ID 一样

表 7-2：

Header	CMD	Data		Check Sum
Byte 0~3	4	5	6~21	22~23
0xc8d7b6a5	0x91	包号（0~31/0xFF）	AD 数据	低位在前

包号： 0xFF～表示请求错误或者没有新的数据

注 1：测量信息和 ADC 数据是实时更新的，请在发送开启血压测量第 2.56 秒开始（前几个周期的波形相对不稳定，），每隔 1.28 秒（血压采样周期），先读取血压测量信息，再读取血压 ADC 数据；请在发送开启心率测量第 6 秒开

始，每隔 1.25 秒（心率采样周期），先读取心率测量信息，再读取心率 ADC 数据。

注 2：读取 ADC 数据对于主控的读取时间有明确的要求；必须严控时间点，否则导致读取回来的数据不完整，无法将波形重现。

注 3：一个 ADC 数据 12 位有效，使用两个字节表示。读取返回的数据，byte6 是 ADC 数据低 8 位，byte7 是 ADC 数据的高 8 位。依次类推。

8. ADC 获取代码参考

```
static uint8_t tx_buf[24] = {0};
static uint8_t rx_buf[24] = {0};
uint8_t *ad_buf;
tx_buf[4] = 0x91;
timeout_cnt = 0;
//1 血压大包 ADC 数据分 32 次读取
for(i=0; i<32; )
{

    tx_buf[5] = i;     //包号
    tx_buf[6] = id;    //需要按协议有求填写
    crc = Crc16(&tx_buf[4],18);         //数据校验
    *(uint16_t*)(&tx_buf[22]) = crc;
    I2cWrite(tx_buf,24); //发送命令到血压模块
    delay_ms(5); //读写间隔延时
    I2cRead(rx_buf, 24); //读取返回值
    crc = *(uint16*)(&rx_buf[22]); //校验确定都回来的是否为有效数据
    check_sum = Crc16(&rx_buf[4], 18);
    if(check_sum == crc) //如果读取上来的数据是正确的
    {
        //这里需要判断返回的包续是否为 0xFF，具体见协议内容
        for(j=0; j<16; j++)
        {
            ad_buf[j+(i*16)] = rx_buf[6+j]; //提取 ADC 数据保存的 ad_buf
        }
        i++;
```

```
            timeout_cnt = 0;
        }
        else
        {
            timeout_cnt ++;
        }

        if(timeout_cnt>10)
        {
            break;
        }
        delay_ms(5);//延时
    }
```

9. 设置模块为低功耗模式

发送设置模块为低功耗指令格式如表 6-1；返回数据格式如表 6-2.

表 6-1

Header	CMD	Data		Check Sum
Byte 0~3	4	5	6~21	22~23
0xc8d7b6a5	0x70	0x01	保留	低位在前

Byte 5： 参数 0x01

表 6-2

Header	CMD	Data			Check Sum
Byte 0~3	4	5	6	6~21	22~23
0xc8d7b6a5	0x70	0x01	0x00~0x01	保留	低位在前

Byte 5： 参数 0x01

Byte 6： 应答，0~设置失败，1~设置成功

注意：模块设置为低功耗模式之后，模块不再受指令控制。模块进入低功耗模式之后，保持给模块供电。再次测量时，通过给模块断电再上电来唤醒模块，模块唤醒后，可以接收测量指令。

10. 读取模块的软件版本号

读取软件版本协议指令见表 10-1，返回见 10-2

表 10-1

Header	CMD	Data		Check Sum
Byte 0~3	4	5	6~21	22~23
0xc8d7b6a5	0xA2	0x02	保留	低位在前

表 10-2

Header	CMD	Data					Check Sum
Byte 0~3	4	5	6~9	10~13	14~17	18~21	22~23
0xc8d7b6a5	0xA2	0x02	软件版本	保留	保留	保留	低位在前

注：如果软件版本为 01.01.01.04，那么 byte6~byte9 读取会来的内容为 0x01010104

11. 校验和算法

Check Sum 使用 CRC16：

```
const uint16_t crc16_tab[256] =
{
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
```

```
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040
};
uint16_t Crc16(uint8_t *data,uint16_t len)
{
    uint16_t crc16 = 0xFFFF;
    uint32_t uIndex ;
    while (len --)
    {
        uIndex = (crc16&0xff) ^ ((*data) & 0xff) ;
                data = data + 1;
        crc16 = ((crc16>>8) & 0xff) ^ crc16_tab[uIndex];
    }
    return crc16 ;
}
```