

ViPen-2. Описание протокола Bluetooth

v1.24 18.04.2022

AndreySchekalev@vibrocenter.ru – протокол

Файлы:

Types_Vipen2.h – структуры и константы ViPen-2

Цветом выделены отличия от протокола ViPen-1+

Изменения в файле

v1.24 18.04.2022	<p>2. Бекон Advertising -> TVipen2AdvertisingData</p> <p>3. Данные TUserData -> TVipen2UserData</p> <p>Вместо поля Reserv теперь поле с версиями FW</p> <p>uint8_t Firmware;</p> <p>// Версии FW процессоров:</p> <p>// Старшие 4 бита = для SAME70, младшие 4 бита = для CC2640</p>

1. Сервисы ViPen-2

Сервис

(Private service UUID):

413557AA213F42798530D38E41390000

413557AA-213F-4279-8530-D38E41390000

Характеристика для приёма TUserData (п.3)

(Private characteristic UUID):

Read, Notify, 15 байт

42EC1288B8A043DBAE0029F942ED0001

42EC1288-B8A0-43DB-AE00-29F942ED0001

Характеристика для управления прибором

(Private characteristic UUID):

Write 64 байта TVipen2MeasureSetup (п.4)

Read, Notify Статус 2 байта (п.5)

42EC1288B8A043DBAE0029F942ED0002

42EC1288-B8A0-43DB-AE00-29F942ED0002

Характеристика для передачи запроса сигнала (п.6) или текстового лога (п.10)

(Private characteristic UUID):

Write, 2 байта VIPEN2_GET_DATA или VIPEN2_GET_LOG

42EC1288B8A043DBAE0029F942ED0003

42EC1288-B8A0-43DB-AE00-29F942ED0003

Характеристика для приёма сигнала

(Private characteristic UUID):

Indicate, 236 байт

TVipen2_Waveform_Header или TVipen2_Waveform_Data (п.6)

или для приёма текстовой информации (п.10, текстовый массив 234 байта * 20 блоков = 4680 байт)

42EC1288B8A043DBAE0029F942ED0004

42EC1288-B8A0-43DB-AE00-29F942ED0004

2. Бекон Advertising

Обновились ли данные, можно смотреть по полю Timestamp.

Если Timestamp==0 – данных ещё нет.

31 байт, выравнивание на 1 байт

```
#pragma pack(1) // Структуры упакованные
typedef struct stVipen2AdvertisingData
{
    uint8_t Len1;
    uint8_t Type1;
    uint8_t Flag1;

    uint8_t Len9;
    uint8_t Type9;
    char Name9[5];

    uint8_t LenFF;
    uint8_t TypeFF;
    uint16_t ManID;
    uint8_t Addr; // ==0
    uint16_t DeviceNumber; // Номер прибора
    uint32_t TimeStamp; // Счётчик 1024 Гц для проверки, что появились новые данные
    int16_t Values[4]; // Velocity, Value, Excess, Temperature
    uint8_t Battery; // Состояние батареи в процентах 0..100 %
    // Старший бит == 1 - прибор заряжается (зелёный светодиод)
    uint8_t Firmware; // Версии FW процессоров:
    // Старшие 4 бита = для SAME70, младшие 4 бита = для CC2640
} TVipen2AdvertisingData;
#define szTVipen2AdvertisingData sizeof(TVipen2AdvertisingData)
static_assert(szTVipen2AdvertisingData == 31, "");

#define GAP_ADTYPE_FLAGS 1
#define GAP_ADTYPE_LOCAL_NAME_COMPLETE 9
#define GAP_ADTYPE_MANUFACTURER_SPECIFIC 0xFF

#define GAP_ADTYPE_FLAGS_GENERAL 0x02
#define GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED 0x04

#define TI_COMPANY_ID (0x000D) // Company Identifier: Texas Instruments Inc. (13)

// Состояние батареи в процентах 0..100 %
// Старший бит == 1 - прибор заряжается (зелёный светодиод)
#define VIPEN2_CHARGE (0x80)

// бекон по-умолчанию
// бекон
static TVipen2AdvertisingData Vipen2AdvertisingData =
{
    0x02, GAP_ADTYPE_FLAGS, GAP_ADTYPE_FLAGS_GENERAL |
    GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED,
    0x06, GAP_ADTYPE_LOCAL_NAME_COMPLETE, 'V', 'i', 'P', '-', '2',
    0x14, GAP_ADTYPE_MANUFACTURER_SPECIFIC, TI_COMPANY_ID,
    0x00, 0x0001, 0x00000000, 0, 0, -200, 0, 0, 0
};
```

Velocity = Виброскорость, СКЗ 10..1000Гц, мм/с, *100, (7,1 мм/с = 0x02C6)

Value = Значение зависит от типа запрошенного сигнала (TVipen2MeasureSetup.MeasType) *10, (45 м/с² = 0x01C2):

Пик для Ускорения, м/с²

СКЗ для Скорости, мм/с

Размах для Перемещения, мкм

Excess = Экссесс ускорения, о.е., *100, (0,1 = 0x000A, -2,0 = 0xFF38) – для диагностики подшипников

Temperature = Температура, град С, *100, (28,3° = 0x0B0E, -10,0° = 0xFC18)

В беконе в поле Battery хранится состояние батареи в процентах 0..100 %

Старший бит == 1 - прибор заряжается (зелёный светодиод)

Поле Firmware = Справочное поле для проверки версий FW процессоров:

Старшие 4 бита = для SAME70 = (текущая версия 43 & 0x0F) << 4

Младшие 4 бита = для CC2640 = текущая версия 6

Сейчас Firmware == 0xB6 (43 & 0x0F | 0x06)

Когда SAME70 выключен, Firmware == 0x06

3. Данные TUserData

Характеристика для приёма TUserData

Read, Notify, 15 байт

42EC1288B8A043DBAE0029F942ED0001

42EC1288-B8A0-43DB-AE00-29F942ED0001

Повторяют данные бекона Advertising

17 байт, выравнивание на 1 байт

```
typedef struct
{
    uint8_t Addr; // ==0
    uint16_t DeviceNumber; // Номер прибора
    uint32_t TimeStamp; // Счётчик 1024 Гц для проверки, что появились новые данные
    int16_t Values[4]; // Velocity, Value, Excess, Temperature
    uint8_t Battery; // В беконе хранится состояние батареи в процентах 0..100 %
                    // Старший бит == 1 - прибор заряжается (зелёный
                    // светодиод)
    uint8_t Firmware; // Версии FW процессоров:
                    // Старшие 4 бита = для SAME70, младшие 4 бита = для CC2640
} TVipen2UserData;
#define szTVipen2UserData sizeof(TVipen2UserData)
static_assert(szTVipen2UserData == 17, "");

#pragma pack()
```

4. Управление прибором TVipen2MeasureSetup

Характеристика

42EC1288B8A043DBAE0029F942ED0002

42EC1288-B8A0-43DB-AE00-29F942ED0002

Write 64 байта TVipen2MeasureSetup

Позволяет читать состояние прибора и управлять Старт-Стоп-Выключением

Чтобы прибор не выключался через 10 минут, раз в 30 секунд будить его командой

VIPEN2_BT_COMMAND_IDLE

Для команды Старт (VIPEN2_BT_COMMAND_START) заполнить структуру TVipen2MeasureSetup

Для остальных команд данные из структуры не используются.

```
// Команды, приходящие с Bluetooth
const uint32_t VIPEN2_BT_COMMAND_NONE = (0); // Нет команды
const uint32_t VIPEN2_BT_COMMAND_START = (1); // Старт чтения, в структуре - параметры
чтения
const uint32_t VIPEN2_BT_COMMAND_STOP = (2); // Стоп чтения
const uint32_t VIPEN2_BT_COMMAND_IDLE = (3); // Пока не отключаться
const uint32_t VIPEN2_BT_COMMAND_OFF = (4); // Выключиться

typedef uint32_t TBluetoothCommand;

// Что измеряем ?
#define MEAS_TYPE_SPECTRUM (0) // Спектр: Стандартный канал: Ускорение
10-10000 Гц; Скорость 10-1000 Гц; Перемещение 10-200 Гц;
#define MEAS_TYPE_WAVEFORM (1) // Сигнал: Стандартный канал: Ускорение
10-10000 Гц; Скорость 10-1000 Гц; Перемещение 10-200 Гц;
#define MEAS_TYPE_SPECTRUM_SLOW (2) // Спектр: Тихоходный канал: 0,5-50 Гц;
Ускорение, Скорость, Перемещение
#define MEAS_TYPE_WAVEFORM_SLOW (3) // Сигнал: Тихоходный канал: 0,5-50 Гц;
Ускорение, Скорость, Перемещение
#define MEAS_TYPE_SPECTRUM_ENV (4) // Спектр огибающей Ускорения 0,5-10 кГц;
Только канал Ускорения
#define MEAS_TYPE_WAVEFORM_ENV (5) // Сигнал огибающей Ускорения 0,5-10 кГц;
Только канал Ускорения
#define MEAS_TYPE_COUNT (6)

#define MEAS_TYPE_SPECTRUM_MASK (0)
#define MEAS_TYPE_WAVEFORM_MASK (1)

#define MEAS_UNITS_ACCELERATION (0) // Ускорение
#define MEAS_UNITS_VELOCITY (1) // Скорость
#define MEAS_UNITS_DISPLACEMENT (2) // Перемещение
#define MEAS_UNITS_COUNT (3)

// Отсчётов в сигнале
#define SETUP_ALLX_256 (0) // 256
#define SETUP_ALLX_1K (1) // 1024
#define SETUP_ALLX_2K (2) // 2048
#define SETUP_ALLX_8K (3) // 8192
```

```

#define SETUP_ALLX_COUNT    (4)

// Отсчётов в спектре
#define SETUP_ALLF_100      SETUP_ALLX_256      // 100 + 1
#define SETUP_ALLF_400      SETUP_ALLX_1K // 400 + 1
#define SETUP_ALLF_800      SETUP_ALLX_2K // 800 + 1
#define SETUP_ALLF_3200     SETUP_ALLX_8K // 3200 + 1
#define SETUP_ALLF_COUNT    SETUP_ALLX_COUNT

// Частота семплирования в сигнале
#define SETUP_DX_256_HZ      (0)    // 256 Гц
#define SETUP_DX_640_HZ      (1)    // 640 Гц
#define SETUP_DX_2560_HZ     (2)    // 2560 Гц
#define SETUP_DX_6400_HZ     (3)    // 6400 Гц
#define SETUP_DX_25600_HZ    (4)    // 25600 Гц
#define SETUP_DX_COUNT       (5)

// Верхняя частота в спектре
#define SETUP_FN_100_HZ      SETUP_DX_256_HZ      // 100 Гц
#define SETUP_FN_250_HZ      SETUP_DX_640_HZ      // 250 Гц
#define SETUP_FN_1000_HZ     SETUP_DX_2560_HZ     // 1000 Гц
#define SETUP_FN_2500_HZ     SETUP_DX_6400_HZ     // 2500 Гц
#define SETUP_FN_10000_HZ    SETUP_DX_25600_HZ    // 10000 Гц
#define SETUP_FN_COUNT       SETUP_DX_COUNT

const uint32_t SetupMeasAllX[SETUP_ALLX_COUNT] = { 256, 1024, 2048, 8192 };
const float SetupMeasdX[SETUP_DX_COUNT] = { 1.0f / 256.0f, 1.0f / 640.0f, 1.0f / 2560.0f,
1.0f / 6400.0f, 1.0f / 25600.0f };

// Усреднения для спектров
#define SETUP_AVG_NO         (0)    // Нет усреднений
#define SETUP_AVG_4_STOP     (1)    // Усреднить 4 спектра и остановить измерение
#define SETUP_AVG_10_STOP    (2)    // Усреднить 10 спектров и остановить измерение
#define SETUP_AVG_999        (3)    // Усреднять спектры всё время, пока не нажмут
Стоп
#define SETUP_AVG_COUNT      (4)

Эти поля только для производителя – выставять в 0:

#define SETUP_EXTERNAL_SENSOR (0)    // Сигнал подаётся с датчика
#define SETUP_INTERNAL_DAC    (1)    // Сигнал подаётся с внутреннего ЦАП -
используется для внутренних тестов

#define SETUP_READ_MODE        (0)    // Режим измерения
#define SETUP_CALIBRATION_MODE (1)    // Режим калибровки - только для производителя

#pragma pack(4) // Структуры выровнены на 4 байта

// Текущие настройки чтения
typedef struct
{
    TBluetoothCommand Command;

    uint32_t MeasType; // MEAS_TYPE_xxx
    uint32_t MeasUnits; // MEAS_UNITS_xxx

    uint32_t AllX; // SETUP_ALLX_xxx или SETUP_ALLF_xxx
    uint32_t dX; // SETUP_DX_xxx или SETUP_FN_xxx

    uint32_t Avg; // SETUP_AVG_xxx

```

```
uint32_t InternalDAC; // Переключение канала: 0 - Вход с датчиков, 1 - с DAC
uint32_t CalibrationMode; // 0 - Работа, 1 - Калибровка

uint32_t Reserv[8];

} TVipen2MeasureSetup;
#define szTVipen2MeasureSetup sizeof(TVipen2MeasureSetup)
static_assert(szTVipen2MeasureSetup == 64, "");
```

5. Статус прибора

Характеристика

42EC1288B8A043DBAE0029F942ED0002

42EC1288-B8A0-43DB-AE00-29F942ED0002

Read, Notify Статус 2 байта

```
// Биты состояния, Read, Notify:
#define VIPEN_STATE_STOPED (0<<0) // Прибор стоит
#define VIPEN_STATE_STARTED(1<<0) // Прибор в режиме измерения
#define VIPEN_STATE_NODATA (0<<1) // Данных нет (после инициализации)
#define VIPEN_STATE_DATA   (1<<1) // Есть данные
```


6. Получение сигналов/спектров

236 байт, , выравнивание на 4 байта

Порядок байт в int16_t – Low8, High8 (little-endian, интеловский)

Всего в сигнале до 8192 отсчётов по 2 байта = 1 заголовочный + до 71 блока
(TVipen2_Waveform_Header.ViPen2_Data_Blocks) по 236 байт (по 117 отсчётов).

В спектре до 3201 линий по 2 байта

Лишние отсчёты (8192-ой и дальше) == 0 – их не учитывать.

Запрос измеренных данных:

Write команду в характеристику VIPEN2_GET_DATA

```
42EC1288B8A043DBAE0029F942ED0003
42EC1288-B8A0-43DB-AE00-29F942ED0003
```

```
// Команда для запроса сигнала/спектра
#define VIPEN2_GET_DATA (0x0010) // запросить сигнал/спектр
```

Получение сигнала:

Читать характеристику по Indicate

```
42EC1288B8A043DBAE0029F942ED0004
42EC1288-B8A0-43DB-AE00-29F942ED0004
```

Прибор посылает до 72 блоков по 236 байт по Indicate.

Блок 0 – заголовок TVipen2_Waveform_Header

1..71 – данные TVipen2_Waveform_Data

TVipen2_Waveform_Header.ViPen2_Data_Blocks = количество блоков для приёма, включая блок
TVipen2_Waveform_Header = 2..72

Смотреть по полю Hdr.ViPen2_Get_Wave_ID == Block.ViPen2_Get_Wave_ID, что сигнал не
обновился во время передачи

Смотреть блоки по полю ViPen2_Get_Data_Block, что пришли все блоки

```
#pragma pack(1) // Структуры выровнены на 1 байт
```

```
#define VIPEN2_BLUETOOTH_PACKET_DATA_SIZE (236) // длина блока данных
```

```
#define VIPEN2_STAMPS_IN_BLOCK ((VIPEN2_BLUETOOTH_PACKET_DATA_SIZE-2)/2) // 117 отсчётов  
в блоке
```

```

#define VIPEN2_MAX_DATA_BLOCKS (VIPEN2_WAVEFORM_COUNT / VIPEN2_STAMPS_IN_BLOCK + 1) //
Длина сигнала = до 71 блока

typedef struct
{
    uint8_t ViPen2_Get_Data_Command; // Команда
    uint8_t ViPen2_Get_Data_Block; // Номер блока
    uint8_t ViPen2_Get_Wave_ID; // Счётчик, позволяет проверить, что
качаем тот-же замер // Увеличивается на
1, при запросе заголовка
    uint8_t ViPen2_Data_Blocks; // Количество блоков с данными, до 72: 0 =
Hdr; 1..71 = отсчёты

    uint32_t Timestamp; // Счётчик 1024 Гц, совпадает с User_Data. Timestamp
    float Coeff; // Коэф перевода данных int16_t в float, 4 байта

    uint32_t DataType; // MEAS_TYPE_xxx
    uint32_t DataUnits; // MEAS_UNITS_xxx
    uint32_t DataLen; // Отсчётов
    float DataDX; // Шаг между отсчётами, сек или Гц

    int32_t SpectrumAvg;
    int32_t SpectrumAvgMax;

    int16_t Values[4]; // Velocity, Value, Excess, Temperature

    uint8_t Reading; // 1 - идёт измерение; 0 - стоит
    uint8_t Align1[3];

    uint8_t Reserv2[VIPEN2_BLUETOOTH_PACKET_DATA_SIZE - 12 * 4];
} TVipen2_Waveform_Header;
#define szTVipen2_Waveform_Header sizeof(TVipen2_Waveform_Header)
static_assert(szTVipen2_Waveform_Header == VIPEN2_BLUETOOTH_PACKET_DATA_SIZE, "");

typedef struct
{
    uint8_t ViPen2_Get_Data_Block; // Номер блока, 0..71
    uint8_t ViPen2_Get_Wave_ID; // Счётчик, позволяет проверить, что качаем тот-
же замер
    int16_t Wave[VIPEN2_STAMPS_IN_BLOCK]; // отсчёты = 2 байта знаковое * 117
отсчётов в блоке
} TVipen2_Waveform_Data;
#define szTVipen2_Waveform_Data sizeof(TVipen2_Waveform_Data)
static_assert(szTVipen2_Waveform_Data == VIPEN2_BLUETOOTH_PACKET_DATA_SIZE, "");

```