Создание беспроводной системы мониторинга — первый полет «Бекаса»

Александр Калачев

Введение

Среди множества задач, стоящих за понятием мониторинг аграрных и природных систем, одними из актуальных являются следующие:

- изучение влияния изменений климата на состояние почвенного и растительного покрова;
- мониторинг температурного и водного режимов отдельных территорий.

Методы дистанционного зондирования (как активного, так и пассивного) поверхности в микроволновом и инфракрасном диапазонах позволяют получать данные для огромных территорий, причем с относительно высоким временным и пространственным разрешением. Однако для точной интерпретации этих данных необходимо знание свойств поверхности и ее состояния. Тут на помощь приходят данные о составе, температуре и влажности грунта, а также метеорологических условиях. Такие данные собираются не по всей наблюдаемой территории, а в ряде определенных мест с характерными чертами для данной области.

К сожалению, автоматизация сбора данных на текущий момент времени оставляет желать лучшего — в большинстве случаев это эпизодические полуручные измерения в моменты экспедиций.

От красот западного края — к науке

Разработка комплекса приборов для сбора данных о состоянии почвенного покрова определенной территории призвана отчасти, решить указанную проблему. Кроме того, отдельный интерес представляют суточные и сезонные изменения температуры поверхностного слоя степных, болотистых и солончаковых почв, а также соленых и горько-соленых озер и прилегающих к ним территорий. Наличие растворенных сложных солей и органических наслоений в таких почвах за счет процессов разложения и фазовых переходов создает собственный сложный температурный режим и участвует в формировании микроклимата территории (рис. 1).

В частности, значительная часть запада и юго-запада Алтайского края представляет собой сложное сочетание степных почв, соленых озер и солончаков. Правильное понимание и интерпретация процессов, происходящих на данных территориях, способствуют

Рис. 1. Окрестности Славгорода, солончаки неподалеку от Бурлы, Бурлинское озеро, граница Волчихинского и Романовского районов

формированию оптимального режима земледелия и сохранению уникальных биосистем.

Задача на разработку прибора для мониторинга температурного профиля почвы была поставлена в хоз. договоре с Институтом водных и экологических проблем СО РАН, выполнявшим работы по гранту РФФИ № 13-05-98041 «Исследование сезонных вариаций микроволнового излучения соленых и горько-соленых озер на юге Западной Сибири» (руководитель Романов Андрей Николаевич).

Структура системы мониторинга

Поскольку речь идет о мониторинге значительной территории (т. н. полигона измерений), площадь которой может достигать нескольких квадратных километров, для автоматизации наблюдений выгоднее использовать сеть беспроводных узлов, оснащенных необходимым набором датчиков. На первоначальном этапе развития проекта допустимо полуавтоматическое считывание данных, собранных сетью. Общий вид системы сбора данных для этого случая представлен на рис. 2.

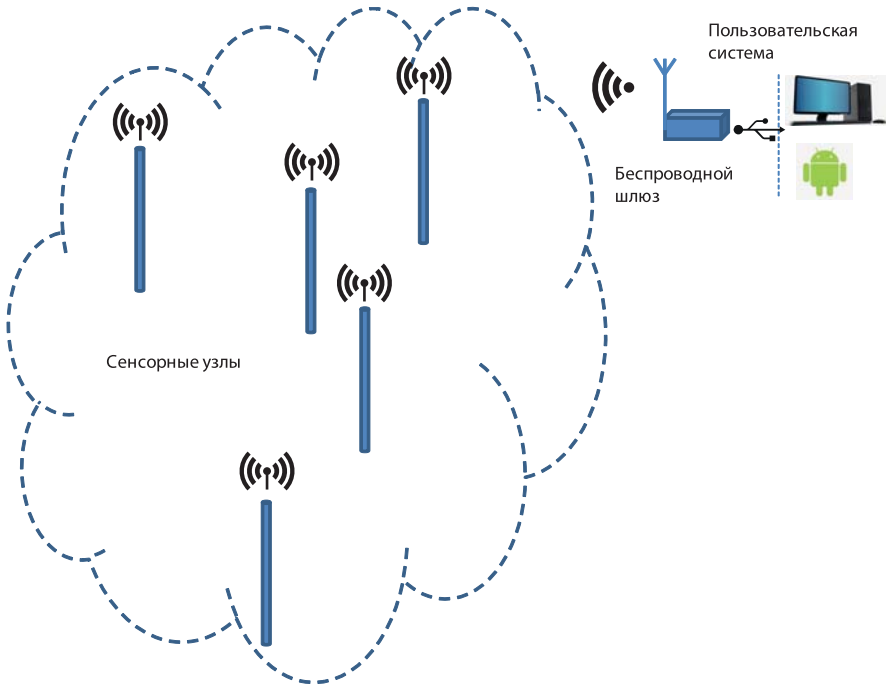

Рис. 2. Общий вид системы сбора данных мониторинга температуры почвенного покрова

В качестве макетного образца измерительной системы для автоматизации мониторинга суточных и сезонных изменений температурного профиля почвы был разработан набор «Бекас» (рис. 3), состоящий из измерительного узла с контроллером и набором датчиков и беспроводным адаптером типа USB-Dongle (в народе — «свисток»).

Состав «Бекаса»:

- щуп на углепластиковой основе с 16 датчиками температуры для измерения профиля температуры почвы/снега/… с платой с беспроводным контроллером и батареей питания — т. н. сенсорный узел;
- выносной датчик-поплавок для измерения температуры поверхностного слоя воды/грунта;
- выносной датчик для измерения температуры воздуха (устанавливается на шест на высоту 2 м);
- USB-Dongle для обеспечения связи с хост-системой, управления устройством и считывания показаний.

Если рассматривать «Бекас» как небольшую сенсорную сеть, то мы имеем один измерительный узел и один узел-координатор, выполняющий еще и роль своеобразного шлюза для взаимодействия с хост-системой, пользователем или приложениями, обрабатывающими данные.

При этом необходимо для сенсорного узла реализовать программу для сбора и хранения показаний датчиков с возможностью изменять параметры процедуры опроса и с удаленным считыванием текущих показаний, а также всего архива измерений. Желательно, чтобы большую часть времени сенсорный узел находился в режиме пониженного энергопотребления.

Для узла-координатора актуален интерфейс с хост-системой, фактически он работает только в моменты считывания данных с сенсорного узла или при настройке процедуры измерений, и для упрощения можно считать, что бюджет энергии для него не ограничен.

Аппаратная часть

С учетом дальнейшего развития системы в распределенную сенсорную сеть, в которой отдельные узлы могут быть разнесены на расстояние до нескольких километров, включая требование о дистанционном считывании показаний, ориентируемся на субгигагерцовый диапазон, в частности — 433 МГц.

В качестве беспроводного контроллера выбран CC430F5137[1] в составе модуля на 433 МГц — TE-CC430F51-433 [2–4]. Интегрированные компоненты радиотракта, компактные размеры и достаточное количество доступных линий ввода/вывода делают данный модуль весьма привлекательным для реализации на нем беспроводных систем различного назначения.

Структура USB-Dongle, реализованного в «Бекасе», достаточно проста: к модулю TE-CC430F51-433 добавлен преобразователь

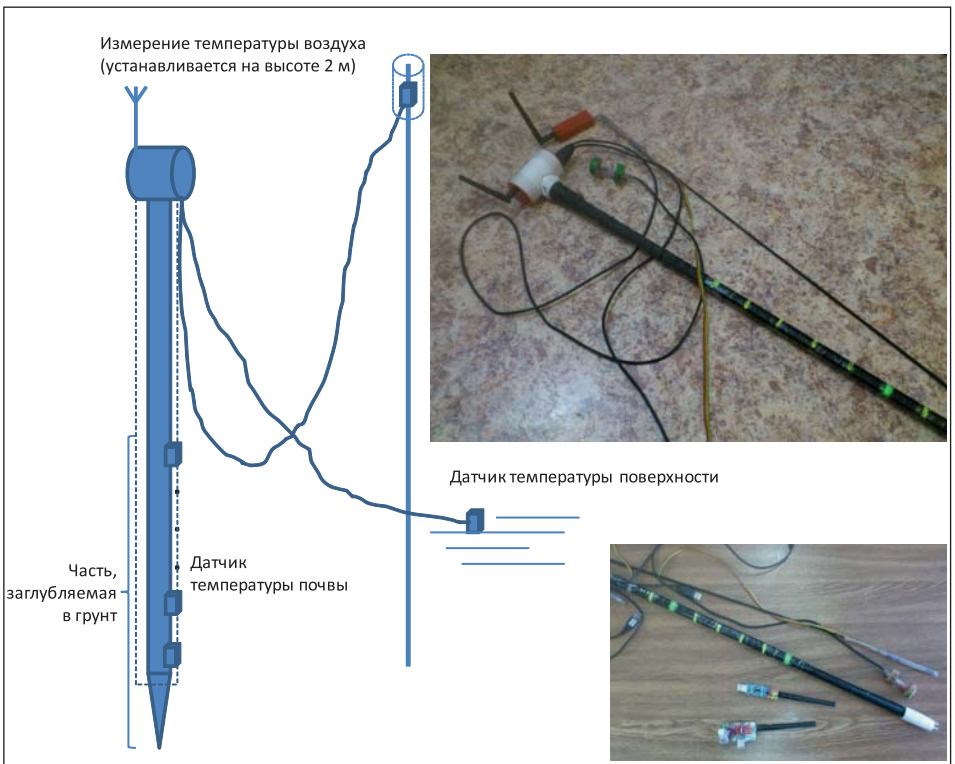

Рис. 3. Состав и внешний вид измерительной системы для автоматизации мониторинга суточных и сезонных изменений температурного профиля почвы «Бекас»

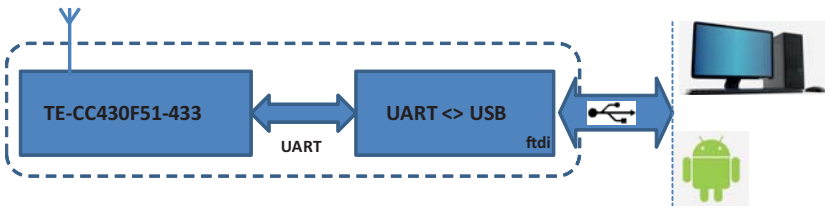

Рис. 4. Структура USB-Dongle, реализованного в «Бекасе»

интерфейсов UART-USB на основе микросхемы от FTDI (рис. 4). Преобразователи FTDI имеют программную поддержку в виде драйверов для большинства операционных систем. В контексте задачи интересны были системы Windows, Android и, как дублирующий вариант, Linux.

В целях поиска оптимального количества требуемых датчиков и их взаимного расположения было принято решение о размещении на щупе сенсорного узла шестнадцати датчиков на расстоянии 5 см друг от друга — суммарно это дало 80 см заглубляемой части. Вместе с выносными получилось 18 датчиков.

Поскольку контроллер CC430F5137 имеет встроенный аппаратный интерфейс I²C, было решено применять температурные датчики с этим же интерфейсом. Выбор пал на цифровые температурные датчики STLM75. За счет наличия адресных выводов их можно размещать до восьми штук на одной I²C шине. В случае с «Бекасом» для корректной работы 18 датчиков потребовалось бы три независимые шины I²C, или дополнительная коммутация датчиков. Структурная схема сенсорного узла «Бекаса» представлена на рис. 5.

Для снижения количества требуемых линий все датчики температуры питаются от одной шины (один общий провод на все датчики и одна питающая линия). Поскольку ток потребления STLM75 мал, возможно их питание от линий порта контроллера — максимальный суммарный потребляемый ими ток всего 2 мА. Дополнительно это дает возможность программно отключать датчики для экономии энергии.

Датчики были разделены на три логических сегмента со своей адресацией (два сегмента по восемь датчиков и один с двумя). Линия данных I2C_DAT для всех датчиков имеет подтягивающий резистор непосредственно на плате сенсорного узла. Каждый из сегментов имеет свою собственную линию тактирования шины — I2C_SCL. Они также имеют отдельные подтягивающие резисторы. При помощи мультиплексора серии 4052 тактовые линии сегментов коммутируются на тактовую линию I2C_SCL от контроллера. Тем самым обеспечивается работа с шиной датчиков только того сегмента, который выбран в данный момент, и исключаются конфликты устройств на шине из-за совпадения адресов. Мультиплексор управляется парой выходных линий контроллера.

Для хранения результатов измерений используется внешняя SPI flash-память объемом 32 Мбит.

Таким образом, один универсальный последовательный интерфейс CC430F5137 работает в режиме I²C, второй — в режиме SPI, при этом сохраняется возможность его работы и в режиме UART.

Поскольку для организации мониторинга территории в рамках поставленной на текущий момент задачи достаточно сети топологии «звезда» или «дерево», то в качестве стека протоколов для нее подойдет простой проприетарный протокол TI SimpliciTI. Он обладает достаточными возможностями для организации обмена данными и при этом имеет небольшой набор API, что в итоге позволяет достаточно быстро разобраться с ним и реализовать собственное сетевое приложение.

Опишем основные алгоритмы функционирования узлов.

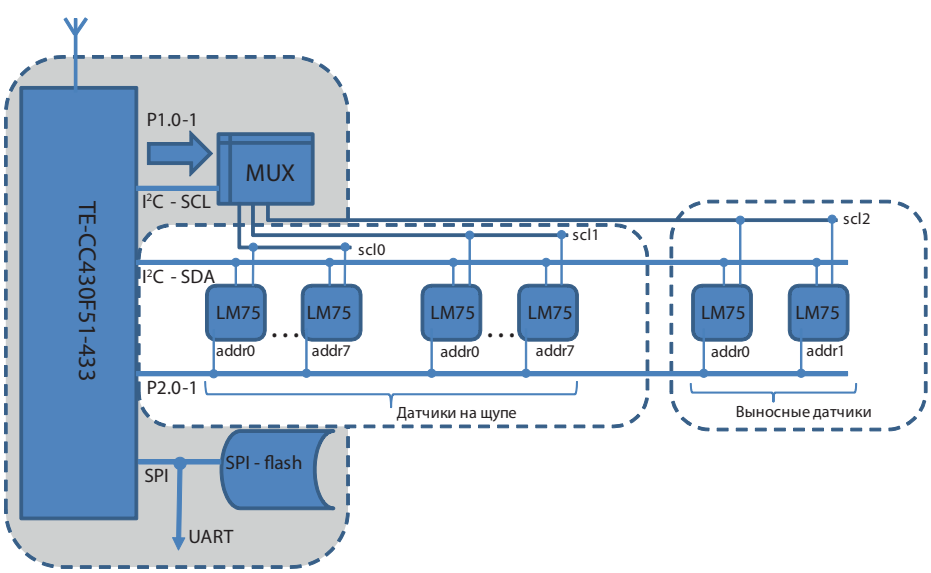

Рис. 5. Структурная схема сенсорного узла «Бекаса»

Сенсорный узел:

- периодически «просыпается», проводит измерения, записывает результаты в память;
- после измерений проверяет наличие подключения к хост-узлу;
- если подключение успешно, переходит на ожидание команд;
- по приходу команд управления выполняет советующие действия;
- если хост не обнаружен, переходит в спящий режим.

Хост-узел (USB-Dongle):

- по включении ожидает запроса на подключение;
- после подключения транслирует команды управления сенсорному узлу;
- передает хост-компьютеру все данные, пришедшие от сенсорного узла.

Программа управления и сбора данных

В качестве основы программного обеспечения возьмем проект приложения UART_Bridge из штатной поставки стека протоколов SimpliciTI [7, 8], как наиболее близкий по структуре к предполагаемым алгоритмам функционирования узлов.

Идея работы программы-примера проста: организуется некое подобие радиоудлинителя последовательного асинхронного порта — беспроводные контроллеры принимают символы по UART-интерфейсам и передают их по беспроводному каналу. Таким образом, хост-системы, подключенные к последовательным портам беспроводных контроллеров, могут обмениваться данными. Разберем исходный код примера:

```
#define INDICATOR_TIME_OUT 250;
// подключение библиотек поддержки ввода/вывода
и платформы
#include<stdio.h>
#include «bsp.h»
#include «mrfi.h»
// подключение библиотек стека протоколов SimpliciTI
#include «nwk_types.h»
#include «nwk_api.h»
#include «nwk_pll.h»
// подключение библиотек UART и ввода/вывода,
специфичных для платформы
#include «bsp_leds.h»
#ifdef MRFI_CC430
#include «uart_intfc_cc430.h»
#else
  #include «uart_intfc.h»
#endif
// основная функция
void main (void)
{
/* holds length of current message */
uint8_t len; // длина текущего сообщения
/* the link token */
linkID_t LinkID = 0; // идентификатор установленной
между узлами логической связи
/* the transmit and receive buffers */
// буферы для принятых и передаваемых сообщений –
размер задаем в соответствии
// с максимальным размером пакета
uint8_t rx[MAX_APP_PAYLOAD], tx[MAX_APP_PAYLOAD];
/* holds led indicator time out counts */
uint16_t led_tmr;
```

```
// инициализируем периферию контроллера
BSP_Init();
// инициализируем радиочасть контроллера и стек протоколов SimpliciTI
SMPL_Init(NULL);
// настраиваем UART
  uart_intfc_init();
/* turn on the radio so we are always able to receive data asynchronously */
// включаем радио на прием
SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, NULL);
// индицируем включением светодиода
/* turn on LED. */
BSP_TURN_ON_LED1();
#ifdef LINK_TO
// код, выполняемый на узле, запрашивающем подключение
{
uint8_t cnt = 0;
// посылаем сообщение в UART об ожидании подключения
tx_send_wait(«Linking to...rn», 15);
// посылаем пакет-запрос на подключение до тех пор,
// пока не будет получен положительный ответ
while (SMPL_SUCCESS!= SMPL_Link(&LinkID))
if(cnt++ == 0)
{
/* blink LED until we link successfully */
// периодически изменяем состояние светодиода
BSP_TOGGLE_LED1();
}
}
#else // ifdef LINK_LISTEN
// код, выполняемый на узле, ожидающем подключение
// посылаем сообщение по UART об ожидании подключения
tx_send_wait(«Listening for Link...rn», 23);
// ожидаем запрос на подключение
while (SMPL_SUCCESS!= SMPL_LinkListen(&LinkID))
{
/* blink LED until we link successfully */
// мигаем светодиодом до тех пор, пока не будет осуществлено подключение
BSP_TOGGLE_LED1();
}
#endif
// когда соединение произошло, посылаем сообщение по UART
tx_send_wait(«Link Established!rnReady...rn», 29);
/* turn off the led */
// выключаем светодиод
BSP_TURN_OFF_LED1();
// переходим в основной бесконечный цикл работы:
// по приходу символов по беспроводному каналу узел транслирует его в UART
// и по приходу символа по UART он передается в эфир
main_loop:
// метка начала основного цикла
// если принят пакет данных, то его содержимое побайтно передается по UART
if(SMPL_Receive(LinkID, tx, &len) == SMPL_SUCCESS)
{
/* blocking call but should be ok if both ends have same uart buad rate */
tx_send_wait(tx, len);
led_tmr = INDICATOR_TIME_OUT; /* update activity time out */
}
FHSS_ACTIVE(if(nwk_pllBackgrounder(false)!= false));
{
/* check to see if the host has sent any characters and if it has
* then send them over the radio link and reset indicator timeout.
*/
// в том случае если по UART приняты символы/данные, передаем их в эфир
len = rx_receive(rx, MAX_APP_PAYLOAD);
if(len!= 0)
{
while(SMPL_Send(LinkID, rx, len)!= SMPL_SUCCESS);
led_tmr = INDICATOR_TIME_OUT; /* update activity time out */
// делаем небольшую паузу после передачи для возможного приема пакетов
// отсутствие паузы может привести к потере символов, переданных вторым узлом по
// радиоканалу
MRFI_DelayMs(5);
}
}
/* manage led indicator */
// периодически мигаем светодиодом
if(led_tmr!= 0)
{
led_tmr--;
BSP_TURN_ON_LED1();
}
else
BSP_TURN_OFF_LED1();
goto main_loop; /* do it again and again and again and... */
}
```

Блок-схема работы программы-примера приведена на рис. 6.

Как видно, для хост-узла исходный алгоритм работы останется практически неизменным.

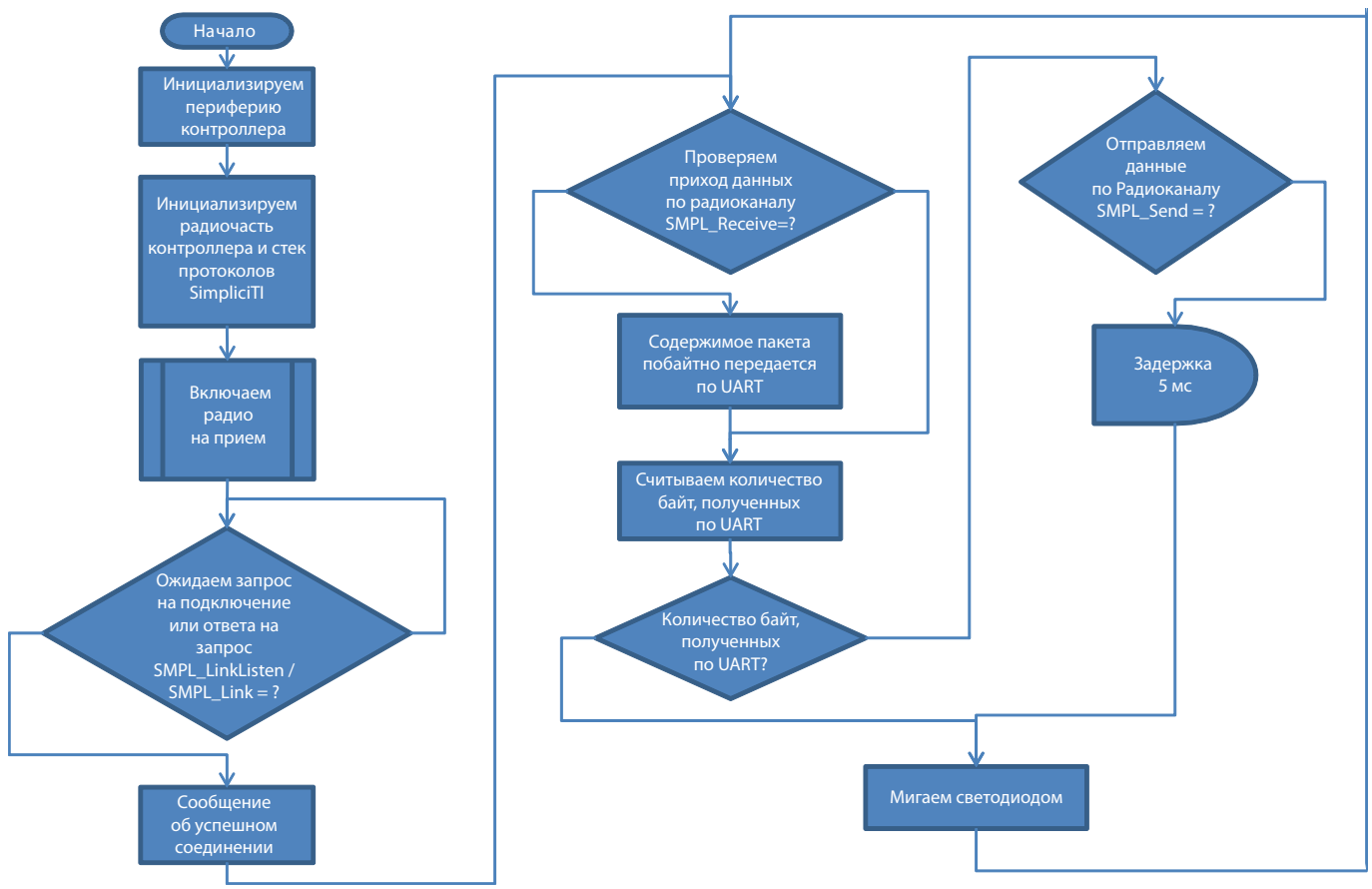

Рис. 6. Блок-схема работы программы-примера UART_Bridge

Учитывая, что для узла, ожидающего подключения, более критичным является наличие постоянного питания, то узлом, выполняющим первоначально процедуру ***LinkListen,*** будет хост-узел. Соответственно, процедуру ***LinkTo*** будет периодически пытаться выполнять сенсорный узел. Блок-схема работы сенсорного узла представлена на рис. 7.

С учетом особенностей IAR-проекта исходного примера, код для узлов «Бекаса» будет следующим:

```
Becas_main
#define INDICATOR_TIME_OUT 250;
#include<stdio.h>
#include «bsp.h»
#include «mrfi.h»
#include «nwk_types.h»
#include «nwk_api.h»
#include «nwk_pll.h»
#include «bsp_leds.h»
#ifdef MRFI_CC430
#include «uart_intfc_cc430.h»
#else
#include «uart_intfc.h»
#endif
// подключаем драйверы периферийных датчиков «Бекаса» и необходимые функции
// работа с шинами I²C, SPI, опрос датчиков и обработчиков прерываний
#include «becas_bsp.h»
#include «becas_bsp.c»
// основной рабочий цикл
void main (void)
{
uint16_t led_tmr;
// инициализируем поддержку платформы в стеке протоколов
BSP_Init();
// инициализируем радиочасть контроллера и стек протоколов SimpliciTI
SMPL_Init(NULL);
```

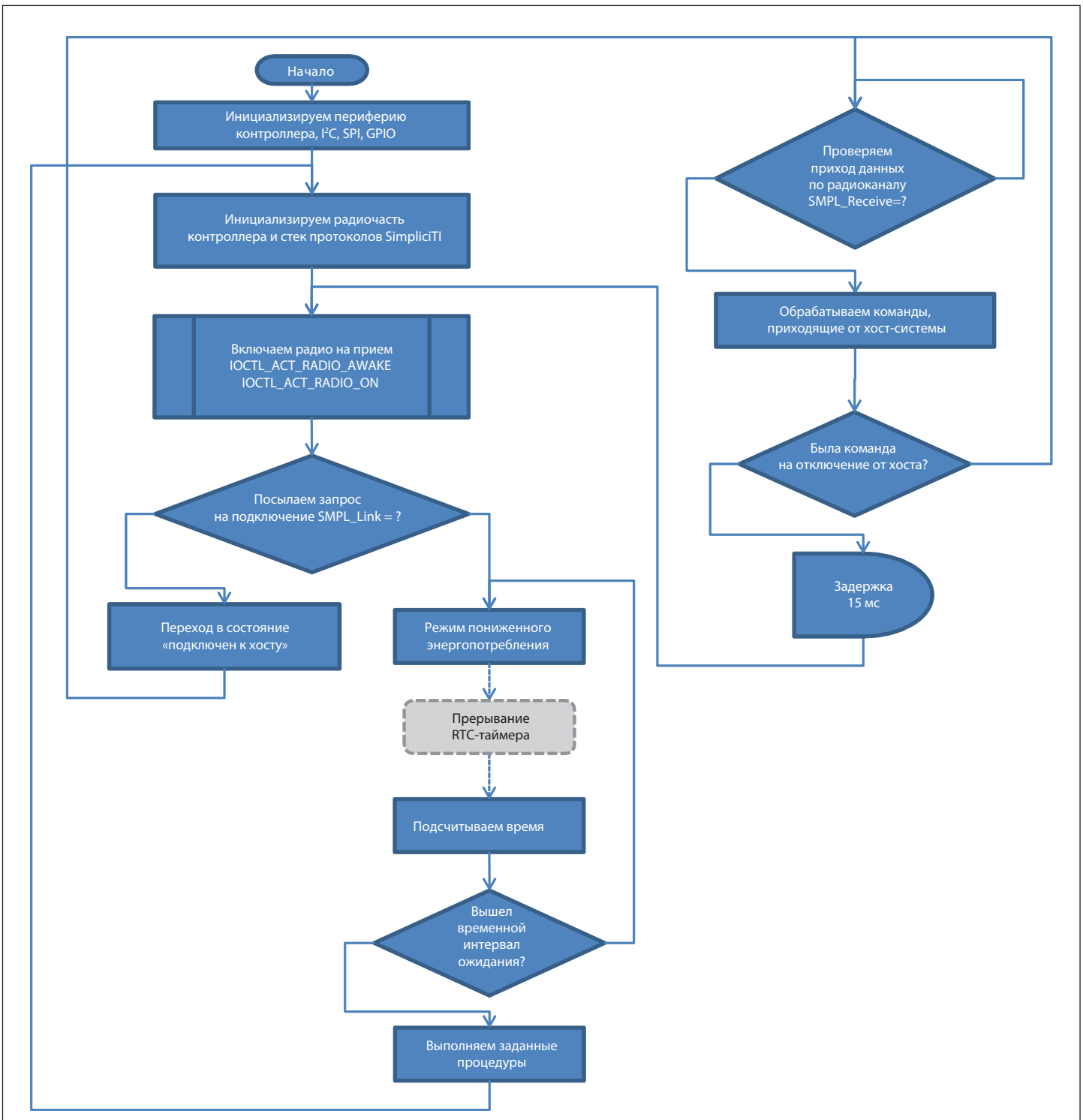

Рис. 7. Блок-схема алгоритма работы сенсорного узла

```
#ifdef LINK_TO
// задаем переменные состояния устройства
radio_of_state=-1;
device_state=measure;
// инициализируем периферийные устройства «Бекаса»
// настраиваются выводы, шины
setup_becas_bsp();
// запускаем таймер реального времени (срабатывание
каждые 2 секунды)
setup_RTC_timer();
// проверяем наличие флэш-памяти и ее тип
spi_read_id();
// проверяем наличие и адреса датчиков
read_sensors();
// ищем адрес последней записи в SPI флэш-памяти
search_last_note();
begin_addr=last_addr;
// задаем интервал опроса
interval=300;
#else // ifdef LINK_LISTEN
// наш хост-узел
// инициализируем UART
uart_intfc_init();
#endif
#ifdef LINK_TO
begin_listen:
// активируем радиочасть на прием
SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_
AWAKE, NULL);
SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_
RXON, NULL);
// посылаем запрос на установление соединения
if (SMPL_SUCCESS!= SMPL_Link(&LinkID))
{
// в случае неудачи отключаем радио,
// переходим в режим пониженного энергопотребления
SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_
SLEEP, NULL);
__bis_SR_register(LPM0_bits + GIE); // enter LPM0
__no_operation();
// здесь контроллер периодически выходит из спящего
режима
// по прерыванию RTC таймера в том случае,
// если истек заданный временной интервал
goto begin_listen;
}
else
{
// в случае успешного соединения изменяем переменные
состояния
device_state=host_connect;
radio_of_state=-1;
};
#else // ifdef LINK_LISTEN
// включаем радиочасть на прием
SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_
RXON, NULL);
// индицируем включением диода
BSP_TURN_ON_LED1();
{
uint8_t cnt = 0;
// делаем небольшую паузу перед началом ожидания
подключения
MRFI_DelayMs(10000);
// сообщение в UART об ожидании подключения
tx_send_wait(«Looking for....rn», 15);
// ждем запроса на подключение
while (SMPL_SUCCESS!= SMPL_LinkListen(&LinkID))
// периодически мигаем диодом (от скуки...)
if(cnt++ == 0)
{
BSP_TOGGLE_LED1();
tx_send_wait(«.»,1);
}
}
#endif
#ifdef LINK_TO
#else
tx_send_wait(«Link Established!rnReady...rn», 29);
/* turn off the led */
BSP_TURN_OFF_LED1();
#endif
// основной рабочий цикл
main_loop:
#ifdef LINK_TO
// действия по приему байта данных
if(SMPL_Receive(LinkID, tx, &len) == SMPL_SUCCESS)
{
switch (tx[0])
{
case 'd': // data
device_state=nothing;
send_data();
break;
case 'g': // go
device_state=measure;
cr();
while(SMPL_Send(LinkID, «Started:», 8)!= SMPL_
SUCCESS){};
break;
case 's': //stop
device_state=nothing;
cr();
while(SMPL_Send(LinkID, «Stopped», 7)!= SMPL_
SUCCESS){};
cr();
break;
case 'r': // read data in real-time
device_state=host_connect;
cr();
while(SMPL_Send(LinkID, «Data:», 5)!= SMPL_SUCCESS)
{};
cr();
break;
case 'q': // get selected data
device_state=nothing;
send_selected_data();
break;
case 'E': //!!! ERASE all flash
device_state=nothing;
cr();
while(SMPL_Send(LinkID, «Erasing», 7)!= SMPL_SUCCESS)
{};
spi_write_en();
spi_erase_all();
MRFI_DelayMs(30000);
while(SMPL_Send(LinkID, «...Erased», 9)!= SMPL_
SUCCESS){};
cr();
last_addr=0x0;
begin_addr=last_addr;
n_of_measures=0;
break;
case 't': //time&date set
device_state=nothing;
cr();
while(SMPL_Send(LinkID, «Date&Time:», 10)!= SMPL_
SUCCESS){};
cr();
for(char i=0; i<12; i++)
{
while(SMPL_Receive(LinkID, tx, &len)!= SMPL_SUCCESS)
{};
time_array[i]=tx[0]-0x30;
while(SMPL_Send(LinkID, tx, 1)!= SMPL_SUCCESS){};
};
set_RTC_time();
MRFI_DelayMs(10);
while(SMPL_Send(LinkID, «__done», 6)!= SMPL_
SUCCESS){};
emit_num(year); emit('/'); // год
emit_num(month); emit('/'); // месяц
emit_num(day); bl();//день
emit_num(hh); emit(':');//час
emit_num(mm); emit(':');//минута
emit_num(ss); //секунды
tics=ss+mm*60+hh*3600;
cr();
break;
case 'i': // interval set
device_state=nothing;
cr();
while(SMPL_Send(LinkID, «Interval:», 9)!= SMPL_
SUCCESS){};
cr();
for(char i=0; i<3; i++)
{
while(SMPL_Receive(LinkID, tx, &len)!= SMPL_SUCCESS)
{};
time_array[i]=tx[0]-0x30;
//echo
while(SMPL_Send(LinkID, tx, 1)!= SMPL_SUCCESS){};
};
MRFI_DelayMs(10);
while(SMPL_Send(LinkID, «_done_», 6)!= SMPL_
SUCCESS){};
interval=time_array[0]*100+time_array[1]*10+time_
array[2];
emit_num(interval); // interval
cr();
break;
case 'o': //off radio
device_state=measure;
cr();
while(SMPL_Send(LinkID, «Link off», 8)!= SMPL_SUCCESS){};
cr();
radio_of_state=0;
break;
}
}
if(radio_of_state==0)
{
SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_
SLEEP, NULL);
MRFI_DelayMs(15);
goto begin_listen;
};
#else // ifdef LINK_LISTEN
// проверяем приход данных по радиоканалу
if(SMPL_Receive(LinkID, tx, &len) == SMPL_SUCCESS)
{
/* blocking call but should be ok if both ends have same uart
buad rate */
// отправляем полученные данные в UART
tx_send_wait(tx, len);
led_tmr = INDICATOR_TIME_OUT; /* update activity time
out */
}
FHSS_ACTIVE(if(nwk_pllBackgrounder(false)!= false));
{
```

```
// проверяем количество байт, полученных по UART
len = rx_receive(rx, MAX_APP_PAYLOAD);
if(len!= 0)
{
// передаем данные в эфир
while(SMPL_Send(LinkID, rx, len)!= SMPL_SUCCESS);
led_tmr = INDICATOR_TIME_OUT; /* update activity time
out */
// небольшая пауза
MRFI_DelayMs(5);
}
}
// мигаем светодиодом
if(led_tmr!= 0)
{
led_tmr--;
BSP_TURN_ON_LED1();
}
else
BSP_TURN_OFF_LED1();
#endif
// замыкаем цикл работы
goto main_loop;
}
```

Поддержка периферийных устройств

Рассмотрим основные интересные моменты реализации простейших драйверов периферийных устройств и доступа к датчикам.

Запись результатов измерений ведется во внешнюю flash-память с интерфейсом SPI. Процедуры инициализации шин SPI и I²C были взяты из библиотеки CC430x513x_Code_Examples [2, 9] (архив *slac458c.zip*) и немного адаптированы с учетом реально задействованных линий портов ввода/вывода.

В частности, для инициализации SPI был взят участок кода из файла-примера *cc430x513x_uscia0_spi_09*:

```
void init_spi(void)
{
// настраиваем выводы
PMAPPWD = 0x02D52; // Get write-access to port
mapping regs
P1MAP6 = PM_UCA0SIMO; // Map UCA0SIMO output to
P1.6
P1MAP5 = PM_UCA0SOMI; // Map UCA0SOMI output to
P1.5
P1MAP4 = PM_UCA0CLK; // Map UCA0CLK output to
P1.4
PMAPPWD = 0; // Lock port mapping registers
P1DIR |= BIT7; // cs - output
P1OUT |= 0x80; // set high
P1DIR |= BIT6 + BIT5 + BIT4;
P1SEL |= BIT6 + BIT5 + BIT4; // P2.0,2,4 for debugging
purposes.
UCA0CTL1 |= UCSWRST; // **Put state machine in
reset**
UCA0CTL0 |= UCMST+UCSYNC+UCCKPL+UCMSB; //
3-pin, 8-bit SPI master // Clock polarity high, MSB
UCA0CTL1 |= UCSSEL_2; // SMCLK
UCA0BR0 = 0x02; // /2
UCA0BR1 = 0; //
UCA0MCTL = 0; // No modulation
UCA0CTL1 &= ~UCSWRST; // **Initialize USCI state
machine**
cs_high();
}
```

Инициализация шины I²C построена на базе примера *cc430x513x_uscib0_i2c_04.c*:

```
void init_i2c(void)
{
// настраиваем выводы
PMAPPWD = 0x02D52; // Get write-access to port
mapping regs
P1MAP3 = PM_UCB0SDA; // Map UCB0SDA output to
P1.3
P1MAP2 = PM_UCB0SCL; // Map UCB0SCL output to P1.2
PMAPPWD = 0; // Lock port mapping registers
P1SEL |= BIT2 + BIT3; // Select P1.2 & P1.3 to I2C function
// настраиваем регистры и режим работы
UCB0CTL1 |= UCSWRST; // Enable SW reset
UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC; // I2C
Master, synchronous mode
UCB0CTL1 = UCSSEL_2 + UCSWRST; // Use SMCLK,
keep SW reset
UCB0BR0 = 12; // fSCL = SMCLK/12 = ~100kHz
UCB0BR1 = 0;
// наши адреса 0b1001000 --- 0b1001111
UCB0I2CSA = 0x48; // Slave Address is...
UCB0CTL1 &= ~UCSWRST; // Clear SW reset, resume
operation
}
```

Как упоминалось выше, все датчики сенсорного узла разнесены по трем сегментам, на которые коммутируется тактовый сигнал шины I²C. На управление мультиплексором выделено две линии, и, при необходимости, выдается код нужного нам сегмента датчиков:

```
void set_sens_seg (char seg)
{
// устанавливаем нужный сегмент датчиков (0--3)
P1OUT &= (~0x03); // переключаем на 0-й
P1OUT |= (seg);
}
```

Два вывода второго порта работают на линию питания температурных датчиков (выходной сигнал портов через диоды поступает на линию положительного питания датчиков). Следующие две функции включают или выключают питание датчиков соответственно:

```
void temp_sens_on(void)
{
// настраиваем порты ввода/вывода для питания линии
датчиков температуры
// в нашем случае это линии 0 и 1 порта 2
P2DIR |= 0x03;
// включаем линии (подаем высокий уровень на них)
P2OUT |= 0x03;
}
void temp_sens_off(void)
{
// настраиваем порты ввода/вывода для питания линии
датчиков температуры
// в нашем случае это линии 0 и 1 порта 2
P2DIR |= 0x03;
// выключаем линии (подаем низкий уровень)
P2OUT &= ~0x03;
}
```

На текущий момент в проекте применяются датчики температуры STLM75, данные с которых можно получать практически сразу после включения, без каких-либо процедур инициализации или настройки режимов работы. Для получения температуры необходимо всего лишь провести последовательное чтение двух байт с датчика с нужным адресом на шине:

```
int read_temperature (unsigned char addr)
{
int temperature=0;
unsigned char temp1, temp2;
UCB0I2CSA = addr;
UCB0CTL1 &= ~UCSWRST; // Clear SW reset, resume
operation
UCB0CTL1 &= ~UCTR; // Clear UCTR
while (UCB0CTL1 & UCTXSTP); // Ensure stop condition
got sent
UCB0CTL1 |= UCTXSTT; // I2C start condition
while (UCB0CTL1 & UCTXSTT); // Start condition sent?
// ждем приема данных, проверяем приход ACKa
while ((UCB0IFG&UCRXIFG)!=(0xFF&UCRXIFG));
// считали первый байт
temp1=UCB0RXBUF;
// ждем приема данных, проверяем приход ACKa
while ((UCB0IFG&UCRXIFG)!=(0xFF&UCRXIFG));
UCB0CTL1 |= UCTXSTP; // I2C stop condition
while (UCB0CTL1 & UCTXSTP); // Ensure stop condition
got sent
// считали второй байт
temp2=UCB0RXBUF;
// формируем данные температуры
temperature=temp1<<1;
temperature=temperature|(temp2>>7);
if(temperature&0x100) //check temp sign
{
temperature=(~temperature)&0xFF;
temperature=(temperature+1)*(-5);
}
else
{
temperature=temperature*5;
}; // */
// возвращается температура в градусах Цельсия, умно-
женная на 10
// например, 366 – будет соответствовать температуре
36,6°C
return temperature;
}
```

По истечении заданного интервала, в зависимости от текущего режима работы прибора, в обработчике прерывания таймера будет вызвана одна из функций, представленных ниже:

- чтение показаний датчиков и из памяти;
- чтение показаний, запись в память и передача показаний в эфир в текстовом виде;
- пустая функция.

```
void default_read(void)
{
read_sensors();
write_data_line();
}
#include «display_data.h»
#include «display_data.c»
void host_is(void)//host_connect
{
read_sensors();
write_data_line();
// строка данных
emit_num(0); bl(); //идентификатор устройства
```

```
emit_num(year); bl(); //emit('/');// год
emit_num(month); bl(); //emit('/');// месяц
emit_num(day); bl();//день
emit_num(hh); bl(); //emit(':');//час
emit_num(mm); bl(); //emit(':');//минута
emit_num(ss); bl();//секунды
emit_num(18); bl();// количество датчиков
bl(); bl();
for(char i=0; i<18;i++){
emit_num_dot(RxTemp[i],1); bl();
};
cr();
}
void do_nothing(void)
{
__no_operation();
}
```

Общая функция инициализации и настройки аппаратной части сенсорного узла:

```
void setup_becas_bsp (void)
{
WDTCTL = WDTPW + WDTHOLD; // Stop WDT
#ifdef LINKTO
// включаем линии, отвечающие за управление коммутатором сегментов датчиков
P1DIR |= 0x03;
// инициализируем spi
init_spi();
//разрешаем запись во все сектора
for (long int i=0;i<64; i++)
{
// команда на разрешение записи
spi_write_en();
// разблокируем i-й сектор
spi_unprotect_sector (i*0x10000);
};
// инициализируем шину i2c
init_i2c();
// инициализируем массив функций, вызываемых в прерывании в зависимости от
// текущего режима работы
timer_event[measure]=&default_read;
timer_event[host_connect]=&host_is;
timer_event[nothing]=&do_nothing;
n_of_measures=0;
#else
uart_init();
#endif
}
```

Настройка RTC-таймера и обработка его прерывания построены на базе примера *cc430x513x_RTC_01.c*. Используется самый простой режим его работы — как таймера, и прерывание генерируется по переполнению. Настройки тактирования таймера и его предделителей таковы, что прерывание генерируется каждые 2 с.

```
// работа с RTC
// настройка RTC таймера – режим таймера,
// срабатывание по переполнению каждые 2 с
void setup_RTC_timer(void)
{
// Setup RTC Timer
RTCCTL01 = RTCTEVIE + RTCSSEL_2 + RTCTEV_0; // Counter Mode, RTC1PS, 8-bit ovf // overflow interrupt enable
RTCPS0CTL = RT0PSDIV_2; // ACLK, /8, start timer
RTCPS1CTL = RT1SSEL_2 + RT1PSDIV_4; // out from RT0PS, /16, start timer
}
// задаем обработчик прерывания RTC-таймера
#pragma vector=RTC_VECTOR
__interrupt void RTC_ISR(void)
{
switch(__even_in_range(RTCIV,16))
{
case 0: break; // No interrupts
case 2: break; // RTCRDYIFG
case 4: // RTCEVIFG
// выполняем функции часов-календаря
rtc_routine(); //do calendar functions
// в том случае, если временной интервал между измерениями не вышел,
// обработка прерывания таймера закончится переходом контроллера в режим
// пониженного энергопотребления
if(tic_count>=interval)
{
// если прошел интервал задержки, фиксируем время, производим измерения
// выходим из режима пониженного энергопотребления
ss=tics%60;
mm=tics/60;
hh=tics/3600;
// в зависимости от текущего режима работы выполняем необходимые действия
(*timer_event[device_state])();
// обнуляем интервальный счетчик
tic_count=0;
// указываем, что по завершении обработки прерывания
// контроллер должен перейти в активный режим работы
__bic_SR_register_on_exit(LPM0_bits); // Exit active CPU
  };
break;
case 6: break; // RTCAIFG
case 8: break; // RT0PSIFG
case 10: break; // RT1PSIFG
case 12: break; // Reserved
case 14: break; // Reserved
case 16: break; // Reserved
default: break;
  }
}
```

Из недостатков представленных выше алгоритмов работы и опроса датчиков можно отметить несколько, которые хоть практически и не влияют на работу опытного образца, но тем не менее подлежат коррекции в свете развития системы.

Сенсорный узел:

- Режим энергопотребления при радиообмене с хост-узлом не оптимален: при периодической передаче данных измерений хосту в промежутках между передачами радиочасть постоянно работает в режиме приема.
- Возможна дальнейшая оптимизация режима питания датчиков и выбора режима пониженного потребления.

Хост-узел:

- После отключения сенсорного узла повторная связь с ним возможна только после перезапуска хост-узла.

Заключение

В качестве заключения приведены краткие технические характеристики опытного образца и некоторые результаты его работы:

- диапазон измеряемых температур –40…+70 °С, точность не хуже 0,5 °С;

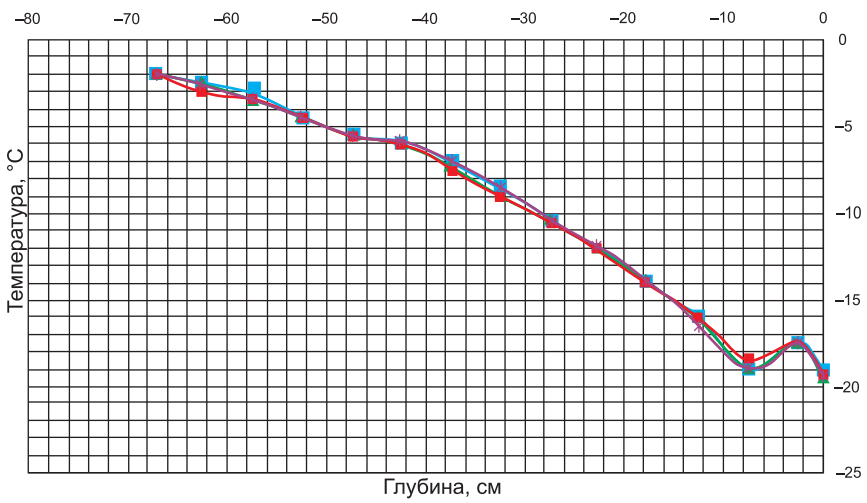

Рис. 8. Тестирование «Бекаса» — правый берег р. Барнаулка, побережье

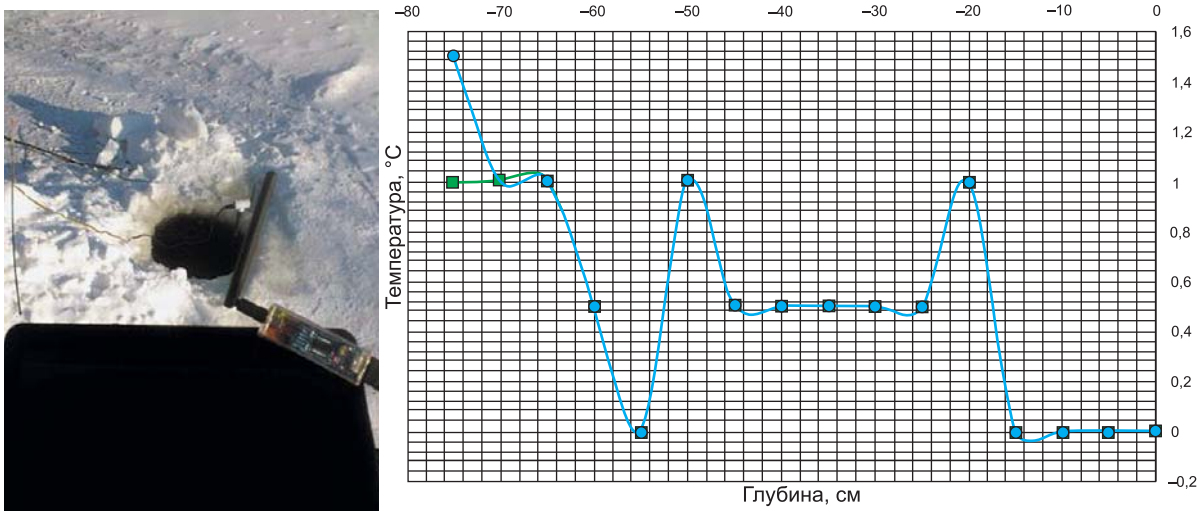

Рис. 9. Тестирование «Бекаса» — левый берег р. Барнаулка, в непосредственной близости от русла

- степень защищенности — не ниже IP67;
- дистанционное считывание показаний и удаленное управление;
- время автономной работы не менее 48 ч;
- настраиваемое время снятия показаний — периодически, по показаниям часов реального времени, по внешнему запросу;
- хранение истории измерений;
- 16 датчиков температуры на щупе с расстоянием в 5 см;
- выносной датчик температуры воздуха для установки на высоте 2 м;
- выносной датчик-поплавок для измерения температуры непосредственно на поверхности воды или почвы.

Зарядка встроенной батареи производится по кабелю mini-USB (разъем со стороны антенны) от любого источника постоянного напряжения в диапазоне 4–5 В.

Устройство тестировалось с Li-ion аккумуляторной батареей емкостью 650 мА/ч. В автономном режиме работы, без передачи данных и связи с хост-компьютером при постоянно включенных датчиках и периодическом мигании индикационного светодиода это дало время работы порядка 360–400 ч.

По результатам тестирования, желательно, чтобы в непосредственной близости от антенн USB-Dongle и сенсорного узла не находилось проводящих предметов — это не лучшим образом сказывается на дальности уверенной связи и качестве приема данных.

Результаты условно-полевого тестирования многоканального цифрового запоминающего термометра «Бекас» представлены на рис. 8–10.

В целом, разработка приложения под CC430 с использованием стека протоколов SimpliciTI не представляет непреодолимых сложностей и достаточно легко дается даже специалисту, не имеющему большого опыта работы с данной аппаратной платформой. Библиотека примеров кода CC430F513x Code Examples представляется полезной при разработке драйверов периферийных устройств и читабельной (наличие комментариев, прозрачная для понимания структура кода).

Литература

1. CC430F5137 — Texas Instruments. www.ti.com/product/cc430f5137
2. CC430F5137 — универсальное решение для радиоустройств 433 и 868 МГц. www.compel.ru/2014/10/13/cc430f5137-universalnoe-reshenie-dlya-radioustroystv-433-i-868-mgts/
3. TE-CC430F51-433, RUSSIA. www.terraelectronica.ru/catalog_info.php?CODE=1048344
4. TE-CC430F51-433 (Evaluation Modules & Boards). www.ti.com/devnet/docs/catalog/thirdpartydevtoolfolder.tsp?actionPerformed=productFolder&productId=14260
5. FTDI Products. www.ftdichip.com/FTProducts.htm
6. STLM75Digital temperature sensor and thermal watchdog. www.st.com/web/catalog/sense_power/FM89/SC294/PF121768
7. SimpliciTI — RF Made Easy. www.ti.com/corp/docs/landing/simpliciTI/
8. Протокол SimplciTI // Новости электроники. 2008. № 14.
9. CC430F513x Code Examples. www.compel.ru/wordpress/wp-content/uploads/2014/10/slac458c.zip

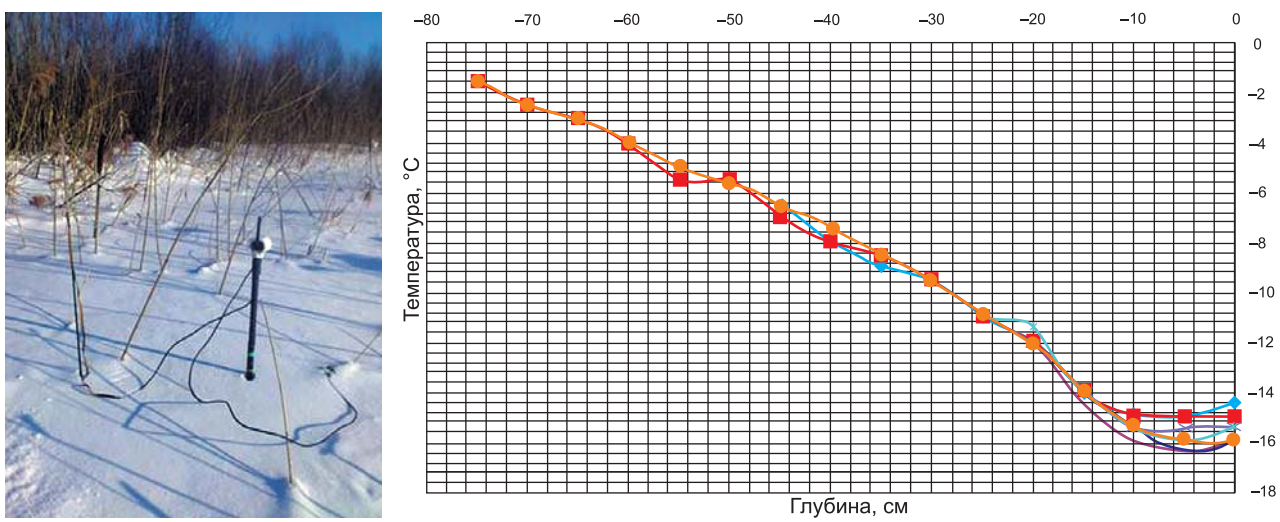

Рис. 10. Тестирование «Бекаса» — прорубь на р. Барнаулка.