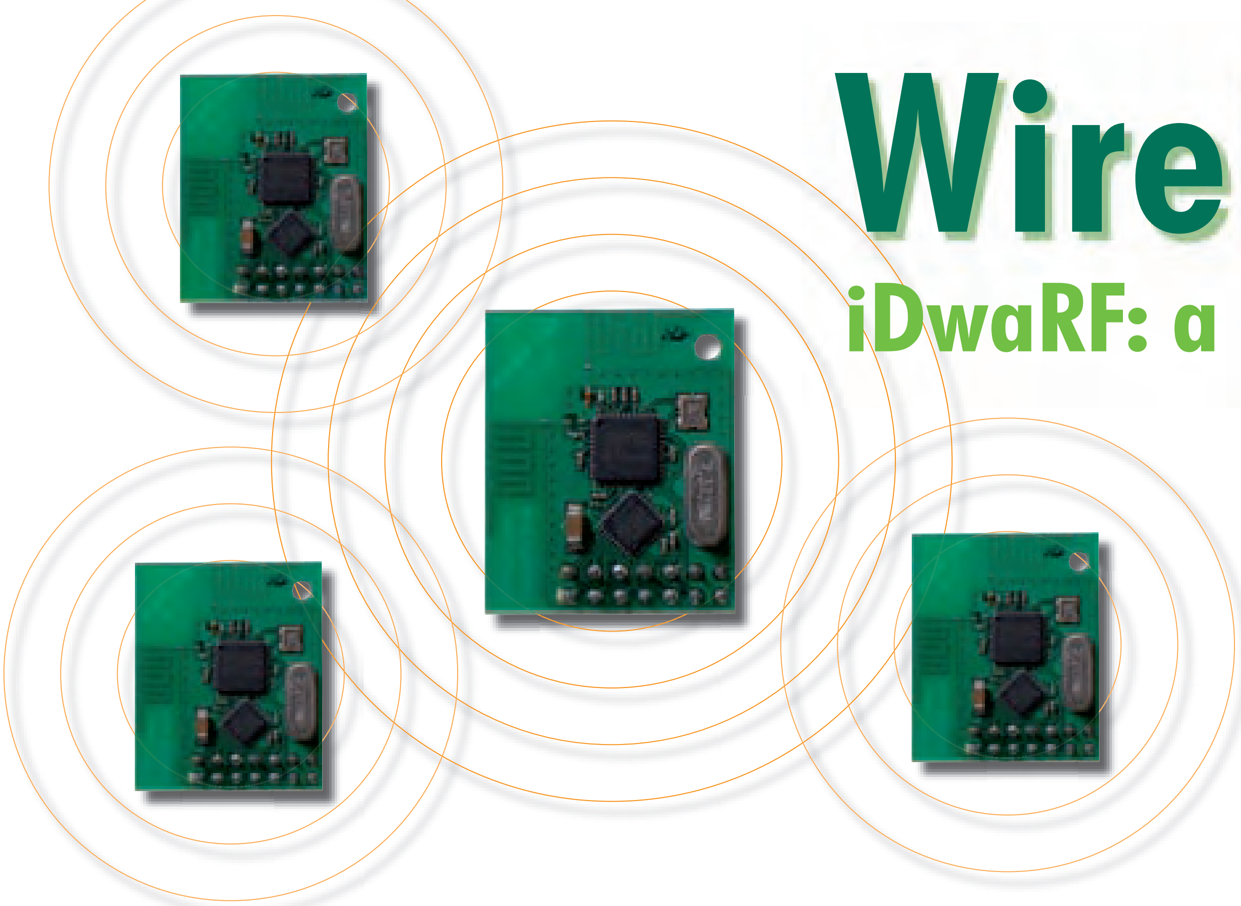

Wireless

iDwaRF: a network

Dr. Erik Lins and Christian Meinhardt

iDwaRF brings together a Cypress WirelessUSB transceiver and an Atmel AVR microcontroller to create a networkable 2.4 GHz radio module featuring a free protocol stack and development environment.

Besides standard applications such as mobile radio, WLAN and Bluetooth, highly-integrated low power radio devices open up many new possibilities, including wireless sensor networks and even radio-controlled robotic football teams able to orchestrate an offside trap in the blink of an eye! And, with iDwaRF, we can do all this without complex protocols or licensing problems.

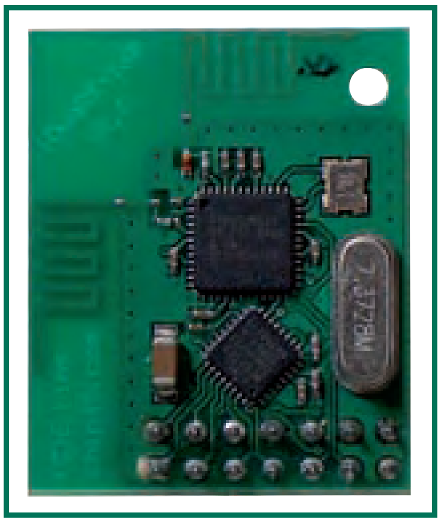

Figure 1. The iDwaRF radio module with 2.4 GHz WirelessUSB transceiver and ATmega168 AVR microcontroller.

An alternative to ZigBee

For wireless sensor network applications ZigBee [1] is often the protocol of choice. The protocol is relatively complicated, and only members of the ZigBee Alliance are permitted to use it in commercial products. Cypress [2] offers a simpler alternative in its WirelessUSB technology [3]. The devices are cheap and the radio protocol makes only moderate demands in terms of hardware and memory in the microcontroller. WirelessUSB supports wireless many-to-one links and is thus ideal for use in wireless sensor networks. The protocol details are freely available and can be used without restriction in combination with Cypress radio chips.

iDwaRF

The iDwaRF-Net software that accompanies the iDwaRF-168 module (**Figure 1**) is a port of the Cypress WirelessUSB protocol to the ATmega168 AVR-family microcontroller [4]. The iDwaRF-168 module can be freely reprogrammed and can equally well play the role of hub or sensor in a many-to-one wireless sensor network. It is easy to add extra application-specific functions.

WirelessUSB operates in the 2.4 GHz ISM band. Each WirelessUSB radio network uses a selection from a total of 79 channels: even when multiple WirelessUSB devices are operating simultaneously the protocol will be able to find a free channel to use.
Transmission uses a robust DSSS (direct sequence spread spectrum) modulation scheme [5]. Even at a 10 % error rate the data can still be received correctly, and if there should be long-term interference the protocol provides for

USB in miniature

able WirelessUSB radio module

an automatic change of channel. Like Bluetooth, WirelessUSB comes in short-range (up to 10 m) and long-range (up to 50 m) versions. The latter is used in the iDwaRF-168 module.

Hub and sensors

An iDwaRF module programmed to act as a hub forms the centre point of a star-topology many-to-one radio network, which can consist of many sensors (**Figure 2**). Normally the hub operates continuously and can be connected to a PC or another microcontroller, acting as a host, using a serial link. For simple applications the iDwaRF module itself can be programmed to carry out the required dedicated host functions.

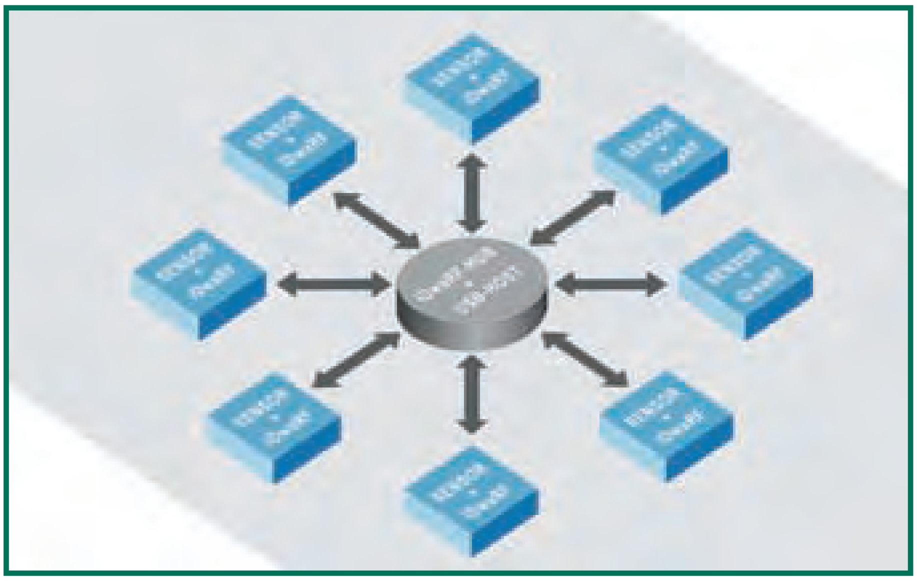

Figure 2. The star-topology radio network consists of a hub and several sensors.

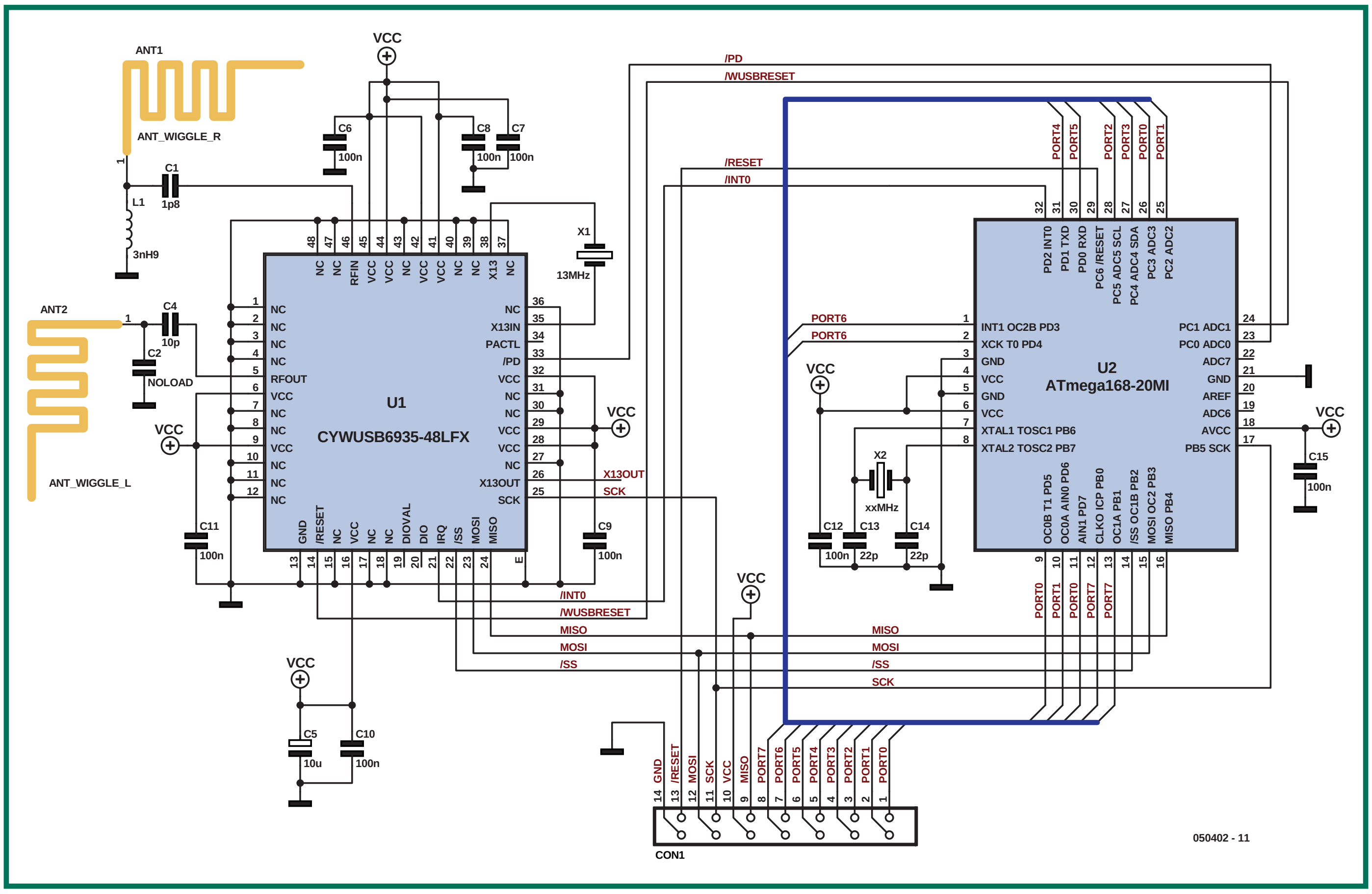

Figure 3. Circuit diagram of the iDwaRF module.

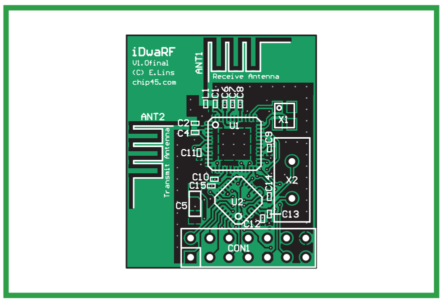

Figure 4. The iDwaRF printed circuit board includes a printed antenna.

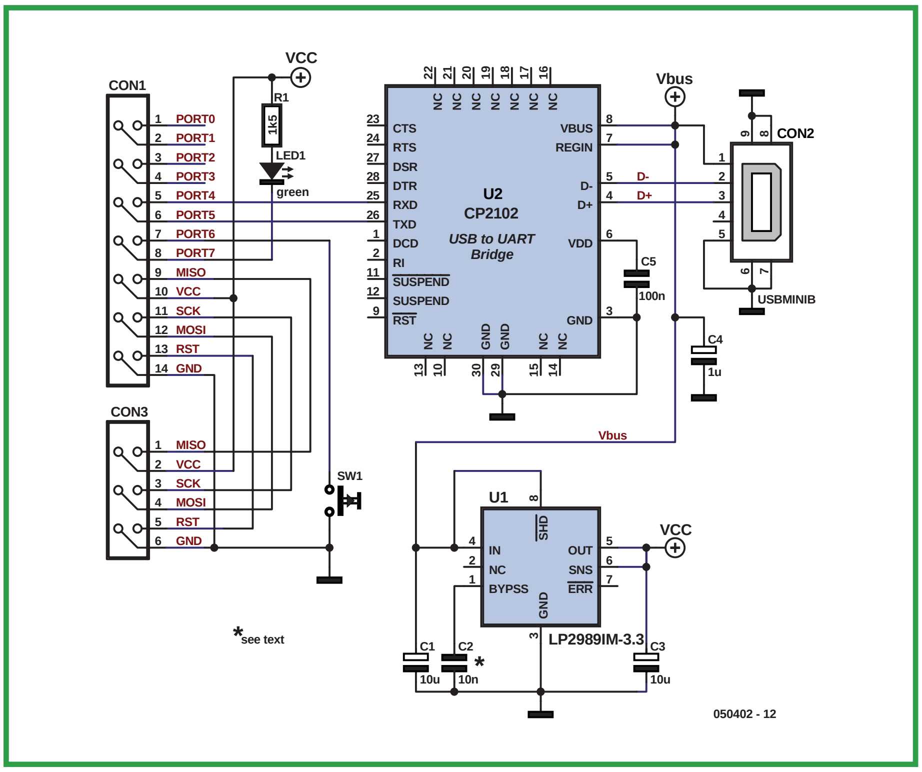

Figure 5. Circuit of the hub board.

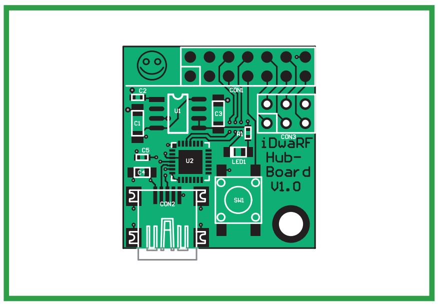

Figure 6. The iDwaRF module is mounted on the hub printed circuit board.

A sensor unit consists of a suitably-programmed iDwaRF module with sensors attached. To save power we use the AVR's internal RC oscillator. Also, the module is only activated at intervals (as determined by the 'beacon time'). Communications are initiated by the sensor and terminated by the hub; a reverse channel (transmitting information from hub to sensor) is also available.

Module printed circuit board

Figure 3 shows the circuit diagram of the iDwaRF-168 module. The Cypress CYWUSB6935-LR transceiver is connected to an ATmega168 microcontroller over SPI, using the MISO, MOSI, SCK and /SS (chip select) signals. An interrupt signal (/INT0) from the radio device indicates the reception of data. The ATmega168 can put the radio chip into power-down mode using an I/O pin connected to the /PD signal, and can reset it using the /WUSBRESET signal. The radio chip needs just an external 13 MHz crystal (X1) and decoupling capacitors (C6 to C11) for operation. The transmit and receive antennas are separate from one another and integrated directly into the circuit board layout as meander lines (**Figure 4**). Having separate antennas gives greater range and sim-

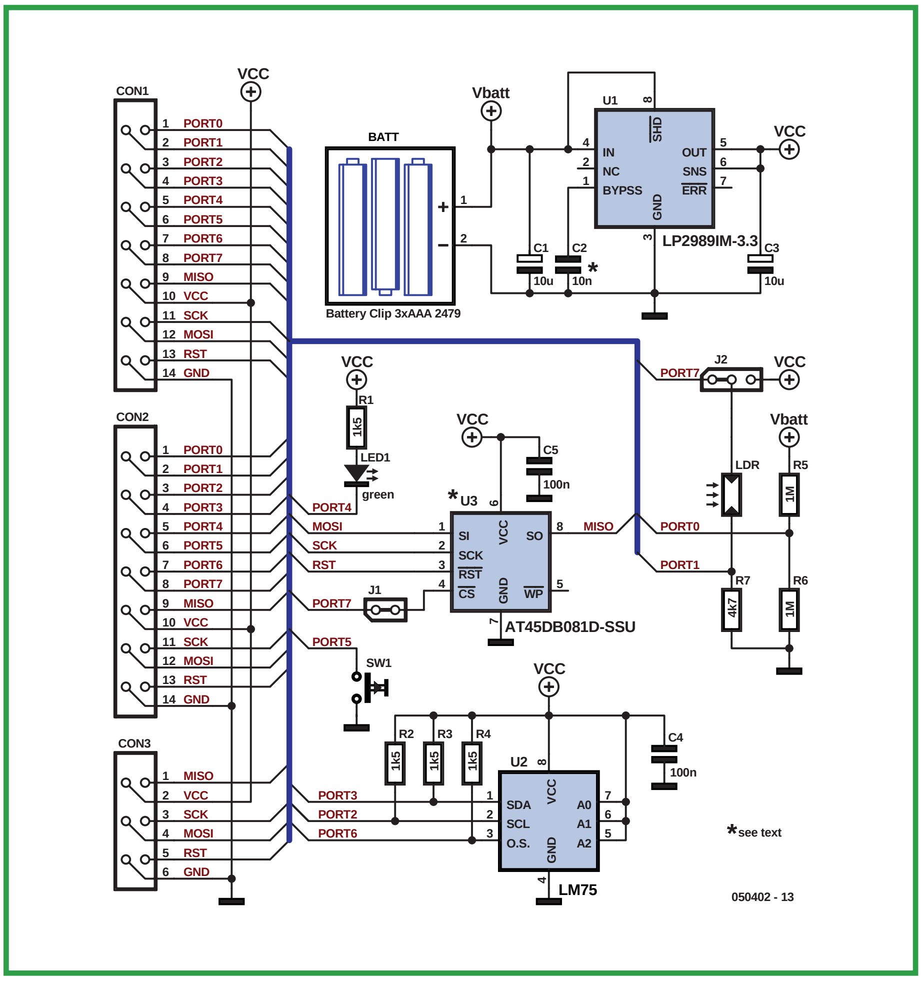

Figure 7. Circuit diagram of the node board with temperature and light sensors.

Table 1. CON1 connection groups

iDwaRF-168 CON1 (port pin)	**First connection** (ATmega168 pin number)	**Second connection** (ATmega168 pin number)	**Third connection** (ATmega168 pin number)
PORT0	OC0B / T1 / PD5 (9)	AIN1 / PD7 (11)	ADC3 / PC3 (26)
PORT1	OC0A / AIN0 / PD6 (10)	ADC2 / PC2 (25)	-
PORT2	SCL / ADC5 / PC5 (28)	-	-
PORT3	SDA / ADC4 / PC4 (27)	-	-
PORT4	TXD / PD1 (31)	-	-
PORT5	RXD / PD0 (30)	-	-
PORT6	INT1 / OC2B / PD3 (1)	XCK / T0 / PD4 (2)	-
PORT7	CLKO / ICP / PB0 (12)	OC1A / PB1 (13)	-

plifies matching (L1, C1 and C4). The antennas are laid out in the manner recommended by Cypress.

The microcontroller is equipped with a crystal in an HC49 package, and so it is straightforward to change it for a different frequency. As supplied the ATmega168 is configured to use its internal RC oscillator.

A 14-way header (CON1) brings out the ISP (in-system programming) signals of the ATmega168 (MOSI, MISO, SCK and /RST), power, and eight spare I/O port pins. Some of the header pins are connected to more than one signal on the microcontroller to allow as many as possible of the peripheral functions of the device to be used. This means that you must ensure that any two microcontroller pins that are connected to the same header pin are never simultaneously configured as outputs, or damage to the microcontroller may result. **Table 1** shows the CON1 pinout and signals in detail.

Application boards

In the simplest wireless network scenario one iDwaRF-168 module is programmed as a hub and one or more modules are programmed as sensors. It is of course necessary to build the necessary interfaces to the sensors themselves and connect them to the modules. To simplify building such systems we have developed three application boards, to each of which can be attached an iDwaRF-168 module: a hub board, a node board and a prototyping board.

The hub board supports the iDwaRF module with a USB interface (the CP2102), a 3.3 V LDO voltage regulator, button and LED (**Figure 5**).

The node board is used to make a sensor unit using an iDwaRF module. The circuit (**Figure 6**) includes an LDR as a light sensor, an LM75 temperature sensor, an (optional) AT45DB801D serial flash memory, a button and an LED. Power is provided by three AAA cells and a 3.3 V LDO voltage regulator.

The two printed circuit boards (**Figure 7** and **Figure 8**) are chiefly populated using SMD components. **Figure 9** shows the boards with iDwaRF-168 modules fitted.

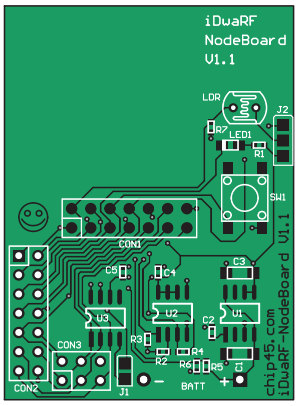

Figure 8. The node board converts the iDwaRF module into a complete sensor unit.

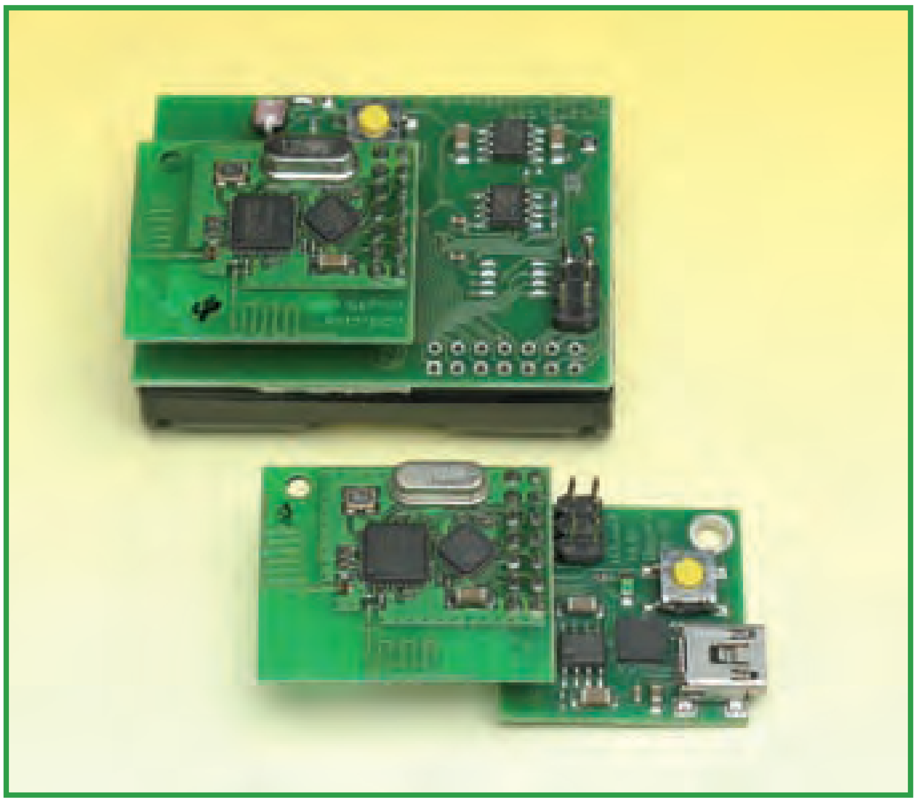

Figure 9. Node board (rear) and hub board (front) with iDwaRF radio modules fitted.

Table 2. Example programs

empty	Empty program: framework for new applications.
chat	Creates a wireless serial connection between two host PCs, allowing 'chatting' between two terminal programs.
tutorial	Example program that switches the LEDs on the hub and sensor modules on and off remotely when a button is pressed. The implementation of this example is explained step-by-step in a separate 'how to' document (see text box).
terminal	This basic sensor network application supports the components on the node board or iDwaRFSensorBox. Data packets (including battery voltage, potentiometer setting, button state and temperature) from several sensors are displayed in plain ASCII text. The terminal program can also send data to individual sensors.
quad_adc	This program reads four ADC channels and transmits the readings to the hub at regular intervals.

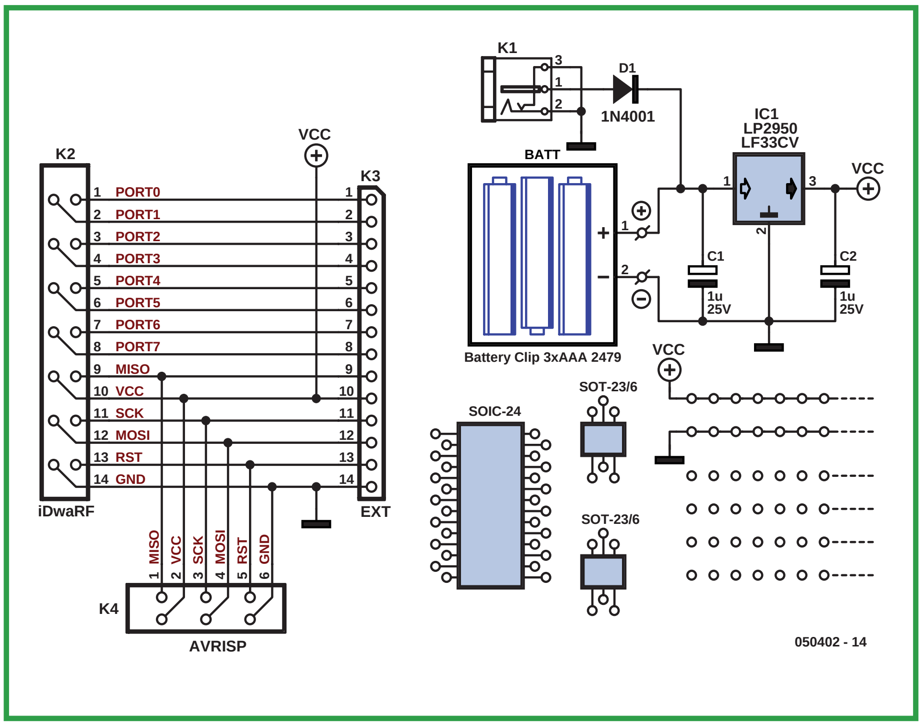

Figure 10. Circuit diagram of the prototyping board for dedicated iDwaRF module applications.

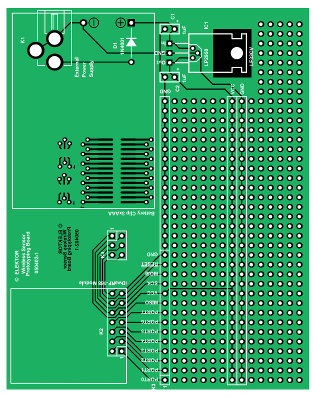

Figure 11. The prototyping board has an experimentation area with SMD footprints on the reverse.

The prototyping board has a generous area for building your own circuits and is suitable for creating more specialised applications using the iDwaRF-168 module. There are SMD pads on the reverse of the board. The only active component in the circuit (**Figure 10**) is the LDO voltage regulator, in a TO-92 or TO-220 package according to the expected current draw. As well as the regulator and socket to accept the iDwaRF-168 module there is an AVR programming connector, a battery holder and, as an alternative, a socket for a mains adaptor with a reverse polarity protection diode. The printed circuit board (**Figure 11**) is half-Eurocard sized and can be used in the place of a node board. For reasons of space we have made the parts lists for the circuit boards and layouts available for download from the *Elektor Electronics* website.

Software

The iDwaRF-Net software package [6] includes a library of firmware for use in hub and sensor modules with corresponding header files (in the 'iDwaRF' directory), along with a few ancillary functions, for example to support serial communications ('USART' directory). There are also four example programs which can either be used as they stand or form the basis for dedicated applications. **Table 2** gives more details of these example programs.

Each example program consists of hub source code (userMain_hub.c) and sensor source code (userMain_sensor.c). The firmware provides the facility to register so-called 'callback' functions, which the firmware calls regularly in the course of normal operation. These functions can be used for application-specific code. The most important callback functions are explained in the text box.

Ready, steady…

Assembling the hardware is relatively straightforward. The SMD printed circuit boards (the iDwaRF module, the node board and the hub board) are available as ready-made units (see the 'Elektor Shop' pages at the back of this issue). A kit of parts is available for the prototyping board. Separately-ordered iDwaRF-168 modules are supplied unprogrammed and without header or crystal fitted in order to give the user maximum flexibility.

The firmware for programming a hub or sensor module is freely available [6]. Modules ordered bundled with a node or hub board come ready-programmed. A crystal is always required for baud rate generation on the hub board (a 7.3728 MHz crystal is supplied as standard), and the microcontroller must be suitably programmed for crystal operation. The correct values for the ATmega168 with a crystal oscillator are: extended byte, 0xF9; high byte, 0xDF; and low byte, 0xFC. It is recommended to use the same values and same crystal frequency for the sensor, and the firmware is currently set up to work on this assumption. If the frequency is to be changed (using a different crystal or the internal RC oscillator) the relevant #define in the firmware must be changed and the code compiled afresh.

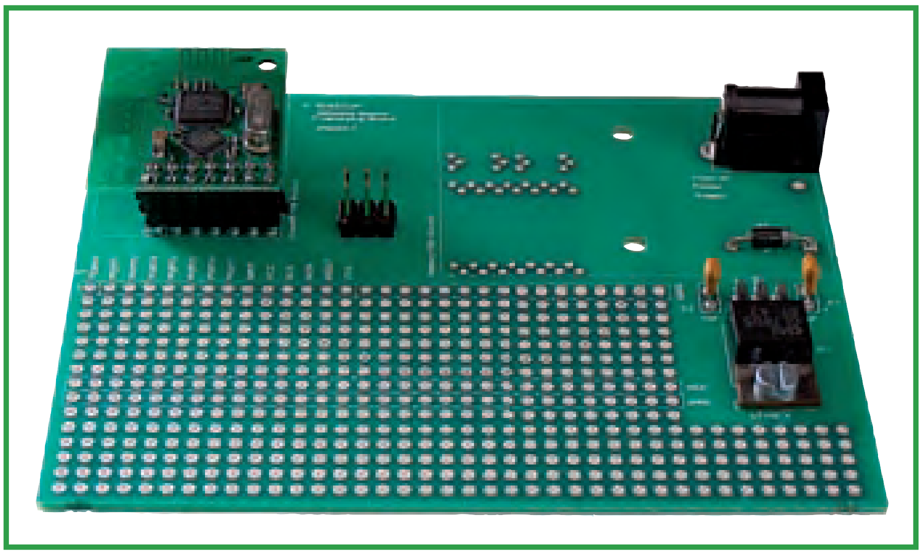
Figure 12. Prototyping board with iDwaRF radio module plugged on.

The AVR ISP connectors have to be soldered on to the node board and hub board, and the node board also needs to be connected to the battery holder. Finally the iDwaRF module can be fitted and (in the case of the node board) the batteries inserted. The USB connection on the hub board requires the corresponding CP2102 virtual COM port driver to be downloaded and installed [6]. Drivers are available for both Windows and Mac OS X. A CP210x module is provided as standard in current Linux kernels, allowing the iDwaRF module to be used with non-Windows PCs.

The pre-compiled hex files [6] for hub and sensor are downloaded using the six-pin AVR ISP connector. Note that the supply voltage for the iDwaRF module on the node and hub boards is only 3.3 V, and so care must be taken to ensure that the programming adaptor also works at this voltage. Simple STK200-compatible programming adaptors that connect to the PC's printer port, which draw power from the target device, do not always work reliably at 3.3 V. Modern USB programming adaptors (compatible with the STK500-V2) such as the CrispAVR-USB work without problems.

Table 3. Commands available in 'terminal' program

Command	Description
rst	Restarts the hub firmware. All sensors are unregistered and lose their ID codes, and must register again with the hub.
gps	Returns an internal Cypress firmware state variable.
bon	Activates automatic bind mode on the hub. New sensors are probed for using PN code 0 and channel 0. In normal use automatic bind mode is activated.
bof	Deactivates automatic bind mode. New sensors can no longer register with the hub.
enu	Displays a list of the currently registered sensors on the hub. The list includes the sensor ID assigned during registration and the unique manufacturing ID stored in the radio chip.
cln	Cleans up the hub's list of registered sensors. Sensors which have registered more than once with the hub and which have therefore been assigned different ID codes are removed from the list and only their current ID code remains valid.
cnf	Configures network parameters. There are 8 different PN codes and channel subsets, and this command allows the hub to be switched to a new PN code and channel. The facility to change channel number is for test purposes only, as the channel is changed automatically in normal operation. The PN code, however, remains fixed. The format is cnf <pncode> <channel> where 0 < pncode < 9 and 0 < channel < 80. The cnf command automatically deletes all registered sensors from the list.
snd	Sends beacon time and other data to a sensor. The data packet is buffered in the hub and when the sensor in question next makes a transmission the packet is sent back in the back channel to the sensor. All parameters are given in decimal. The format is snd <sensorid> <beacon time> <data0> <data1> <data2> <data3> ... A beacon time of -1 indicates that the beacon time is not to be changed.
del	Removes a sensor from the list of registered sensors in the hub. The format is del <sensorid>
hex	Causes sensor data to be displayed in hex rather than as plain text.

The principal callback functions

In the hub:

cbSensorPacketReceived(): called when a packet is received from a sensor. Direct access to the current sensor data is possible, although if lengthy processing is to be carried out the data should be copied into a global buffer and the work done in the main program.

cbSerialDataReceived(): called when a byte is received over the serial interface. Usually the byte is simply stored in a global buffer and its reception signalled using a flag, so that more time-consuming processing can be carried out in the main program.

cbProcessRxData(): further processes the bytes received by cbSerialDataReceived(), for example to implement a complex communications protocol with the host PC. In the 'terminal' example the commands entered on the host PC are parsed and processed in this function.

In the sensor:

cbConfigForSleep(): called shortly before the sensor switches into power-down mode. This allows for particular sensor devices to be switched off to reduce power consumption.

cbExitFromSleep(): called when the sensor leaves power-down mode. At this point particular sensor devices can be powered up again.

cbTxProcess(): assembles the data packet to be sent to the hub. At this point sensors can be read and the readings stored in the global transmit buffer. The current data packet is then automatically sent to the hub.

cbBackchannelProcess(): called with data received from the hub in the reverse channel. This can be used to create an output signal on the sensor, or to generate an analogue voltage using PWM. The 'terminal' example switches on the LED when a data packet is received.

... go!

The 'terminal' example program is the best one to use to demonstrate all the important functions of a wireless sensor network. iDwaRF modules programmed as sensors automatically connect to the module programmed as the hub and transfer data. With one node board and a hub board connected to a PC it is possible to see immediately on the PC's screen when light falls on or is shaded from the light sensor, when the button is pressed, or when the temperature sensor is warmed or cooled. Setup proceeds as follows.

Hub board
The virtual COM port driver creates a virtual COM port when the USB cable is plugged in. The number of the port can be obtained from the Device Manager (reached from the Control Panel).

Now we can run a terminal program, set to the relevant port, and talk to the hub in plain ASCII. If there are no sensors there will initially be no reports from the hub. The hub is reset by typing the command 'rst' and pressing 'enter': the hub will then emit its start-up message. It is best to configure the terminal program to expect CR+LF at the end of each line and to enable local echo, as the hub does not echo the characters it receives.

Data format

S5:	ID 0	ldr 212	temp 22.5°C	batt 2.9V	button OFF	(5 6 7 8 9 10)	:	11
a)	b)	c)	d)	e)	f)	g)		h)

Legend:

a) Packet type: S0 (BIND_REQUEST); S1 (BIND_RESPONSE); S2 (PING, hub only); S3 (ACK); S4 (DATA, hub only); S5 (DATA, sensor only)

b) Sensor ID

c) ADC value from the light intensity sensor

d) Temperature

e) Battery voltage

f) Button state (ON or OFF)

g) Six unused bytes displayed as decimal indices

h) Data byte count

The values from c) to g) above form the packet data payload. The standard packet size is set to 17 bytes, of which six are protocol overhead, leaving 11 bytes of payload.

Sensor
If a sensor is switched on the data packets received will be displayed line-by-line in the terminal window. The first packet is called a 'bind request' where the sensor registers with the hub and, in return, receives an ID code assignment and a value called the 'beacon time'. This period, which has a default value of five seconds, is the interval between the transmission of successive data packets. The format of the data packets is described in detail in the text box 'Data format'. A brief flash of the LED on the sensor board shows when it is active; the LED is extinguished when the sensor returns to power-down mode.

If the node board is moved to a warmer location, or if the ambient light intensity changes, the data packets displayed will reflect the new sensor values. If the button on the node board is pressed a data packet is transmitted immediately: this shows that it is possible to react immediately to external events, without waiting for the preset beacon time to elapse.

Command line
Commands can be typed into the terminal window at any time. Typing 'enu' ('enumerate') lists the sensors that are registered with the hub, showing their ID code and unique serial number.

Our first sensor will appear in this list with ID '0'. To send data to this sensor we use the 'snd' command. In its simplest form this has two parameters: 'snd 0 40' sends the value 40 to the sensor with ID code '0'. The first data byte is always interpreted by the sensor as a new beacon time, in units of 125 ms. In this example, the value of 40 corresponds to the default beacon time of five seconds. So, if we type 'snd 0 8' we will set the beacon time for sensor 0 to one second, and data packets from this sensor will be displayed in the terminal window at this rate. The command 'snd 0 40 1' sets the beacon time back to five seconds and also sends an extra data byte with value '1', which will cause the LED on the node board to light. In the terminal example the code in the sensor simply checks whether there is an extra data byte beyond the beacon time or not, and sets the LED on or off accordingly. The actual value of the extra byte is not taken into account. The complete set of commands provided by the hub in the 'terminal' example is listed in **Table 3**.
If now a further sensor is activated another 'bind request' packet will appear among the data packets being received from the first sensor, followed by a series of data packets. The 'enu' command can be used to list the registered sensors, and should now display two entries with ID codes '0' and '1'. We can test the new sensor by adjusting its beacon time: 'snd 1 8' will set it to one second, and we should now receive packets from sensor 1 at five times the frequency of those from sensor 0. If an 'x' is used in place of the ID code in the send command, the beacon times for all sensors are set simultaneously. Using 'snd x 40' we can therefore reset the beacon times for both sensors to five seconds.

If the data packets are to be processed on the host PC, the 'hex' command can be used to switch the display from readable form to pure hex values. These can easily be read by another application, for example using the scanf() function.

Room for expansion

The supplied programs can form the basis of dedicated applications using iDwaRF modules: the possibilities are endless. Often a couple of extra components are all that is needed, for example to make measurements using an ADC, generate PWM waveforms, scan keys or drive an LCD.

The iDwaRF-Net software can in theory work with up to 255 sensors, although at present the limit is 32. For larger sensor networks the xHub is planned, using an ATmega128 with more flash memory and an external SRAM. An external antenna will increase the range of the hub.

Users of the iDwaRF radio module [7] and the iDwaRF-Net firmware can use a forum [8] organised by the author. This is in addition to the forum on the *Elektor Electronics* website. Further example programs will of course appear for download at [6] as soon as they become available.

(050402-I)

References and links:

[1] www.zigbee.org

[2] www.cypress.com

[3] Thomas Biel: 'Wireless USB', Elektor Electronics, September 2004, p. 8

[4] www.atmel.com/avr

[5] Stefan Tauschek: 'Wireless Connectivity', Elektor Electronics, February 2005, p. 14

[6] www.elektor-electronics.co.uk or www.chip45.com/iDwaRF-168_Downloads

[7] www.chip45.com/iDwaRF-168

[8] www.chip45.com/iDwaRF-Net