Embedded Wireless Transceivers and Applications in Lightweight Wearable Platforms

Mathew Laibowitz and Joseph A. Paradiso
Responsive Environments Group
MIT Media Laboratory

Introduction

Not too long ago, incorporating a wireless RF link into an electronics project implied mastery of the mysteries of RF electronics and layout or the need to incorporate often a fairly cumbersome subsystem into your design. This has all changed enormously over the past few years, with the introduction of embedded modules that encapsulate the radio's electronics onto a compact device, which can easily be mounted onto a small part of your project's circuit card. The embedded RF arena is now exploding with development of new products – although many small hybrid transmitter, receiver, and transceiver modules are on the market, the latest entries include totally integrated RFIC's that put the radio on a CMOS chip with a microcontroller, in some cases adorned with peripherals such as ADC's, DAC's, timers, etc. These elements are intrinsically interconnected, with signal strength, transmit frequency, and other radio parameters digitally accessible as processor registers, and protocols available as subroutines or resident in the device's firmware. Although one still needs to be mindful of proper antenna matching, appropriate bypassing, zero-balancing, interference issues, etc., incorporating radio links into projects has never been simpler. In this article, we present a quick snapshot of the state-of-the-art in embedded RF devices, then discuss an application that we have built around one of them in a wearable badge platform to facilitate social interaction in large events. We conclude by briefly mentioning how we use embedded RF in a dense wearable sensor platform, and point to RFIC's built around heavier protocols, such as Bluetooth [1] and ZigBee [2].

Selecting a Short Range Radio Frequency Device

Bearing in mind that the field is developing so quickly that any survey will rapidly become dated, Table 1 lists and compares a range of currently available devices, including small hybrid, multi-component modules and single integrated CMOS chips (RFICs). Although many of the listed manufacturers provide a range of different offerings that address various requirements, we have selected one representative product to detail from each company.

The selected devices work in the 300 MHz through 1000 Mhz range or at the 2.4 GHz band. The operating frequency is probably the first decision a designer has to make when selecting a RF device. In the United States, the frequency bands 260-470 MHz and 902-928 MHz are license exempt and open for use, provided you transmit at an FCC approved transmission strength and duty cycle. In Europe, the license exempt bands are centered around 433.92 MHz and 868 MHz; The 868 MHz was previously a regulated band in Europe, but has recently become license exempt. In both markets, 2.4 GHz is also license exempt, as are several other higher frequency bands.

In general, devices using frequencies below 1GHz have lower power consumption and longer communication range than devices using 2.4 GHz. They also don't have to worry about interference from now ubiquitous WLAN and Bluetooth networks. However, the 2.4 GHz channel offers higher bandwidth and greater data rates.

Although not all SRDs (short-range devices) come equipped with hardcoded channel-sharing procedures, they all need to have some sort of protocol [3] for working in the chosen frequency band with other devices - either other devices in the same system or unknown devices using the same frequency band. This is

An edited version of this article can be found in Circuit Cellar #163 (February 2004)

especially important in the heavily-used 2.4GHz ISM band. Techniques for sharing the airwaves include Time Division Multiple Access (TDMA), Carrier Sense Multiple Access (CSMA), Frequency Division Multiple Access (FDMA), Frequency Hopping Spread Spectrum (FHSS), and Direct Sequence Spread Spectrum (DSSS).

TDMA is a system by which different devices use the medium at different times, usually done by allocating different time slots to different transmitters. CSMA is a technique where a device first checks whether the medium is available before transmitting. This can still lead to collisions if multiple devices are waiting for the medium to become available, so collision avoidance schemes, such as a random back-off (waiting for a random interval to elapse before transmitting), are usually implemented. The example application shown at the end of this article uses CSMA with such an approach. FDMA is a system where the frequency band is divided into smaller subchannels that are assigned to different devices in the system. Devices operating with the same subchannel cannot be in overlapping physical range space.

Spread Spectrum is a technique where the transmitter rapidly jumps from subchannel to subchannel. In FHSS, each device can have a different sequence of frequencies that it follows, minimizing the probability that two devices will occupy the same frequency simultaneously and interfere. FHSS broadcasts are difficult to listen in on without knowing the hopping code, and frequencies that contain interference can be removed from the hop sequence. Devices with different hopping sequences can coexist on the same channel. DSSS systems are based on spreading the signal energy by XORing the data signal with a pseudorandom spreading code. Different devices in a system can use different spreading codes to support multiple access, which is known as Code Division Multiple Access (CDMA).

It is important to have a general idea of the band sharing scheme that will be used before selecting your RF device. If you are going to use a spread spectrum scheme or FDMA, you must select a chip that will support it in hardware, usually by allowing the frequency to be easily changed in real-time by the application software. If you are going to use a CSMA type scheme, it is a good idea to pick a chip with a Receiver Signal Strength Indicator (RSSI) output that can be polled by an application. If the strength of the signal at the receiver is high, it is usually because another nearby device is transmitting and the medium is busy.

Besides the RF specific parameters, all the standard factors for selecting any electronic component still apply. The power consumption is broken down into three zones: Receiver power consumed while the chip is listening for incoming data, standby current used while the chip is waiting for a carrier, and transmit power consumed while sending. These devices also usually have a sleep mode that draws negligible current. It is important to balance these values against the requirements of your application. If, for example, you need only to transmit, then just the transmit power is important, and the expected power drawn from the RF system is the transmit load weighed by the transmission duty cycle. There are ways of also duty-cycling the receiver – e.g., some devices support power management schemes where the device is mainly in sleep mode and is woken up at periodic intervals to check for the presence of a carrier signal.

It is also important to consider the space requirements of the chip and the ensemble of external components that it requires, as well as any requirements for the layout of the circuit board in order to support the chip. Some manufacturers offer complete modules with their chip and include all required external components on a printed circuit board for easy integration into an application.

Devices have become available recently that integrate both a microcontroller and a RF transceiver in a single CMOS chip. Table 1 includes two such chips, the ChipCon CC1010 and the Nordic nRF24E1 (the rfPIC integrates a transmitter with the processor). These chips contain a flash-programmable 8051 with a full set of peripherals (UARTs, ADC, etc) on the same piece of silicon as the RF transceiver. This reduces component count, board-space, and overall cost. This tight integration between the RF transceiver and the microcontroller allows the transceiver to be controlled by simply writing and reading 8051 registers and by servicing RF specific interrupts. We have recently used the Chipcon CC1010 in the RFRAIN project described below.

Implementation Example – the RFRAIN

The RFRAIN device (standing for RF Random Access Integrated Node) was developed by the Responsive Environments group at the MIT Media Lab as an original part of the UbER-Badge (described below), a compact platform designed to explore applications at the convergence of wearable and social computing. It was decided that the RF subsystem developed for the badge had considerable standalone utility for other projects, hence it was designed as a separate daughterboard called the RFRAIN. A website dedicated to the RFRAIN is located at http://www.media.mit.edu/resenv/rfrain.

The RFRAIN board, with schematic shown in Figure 1 and assembled in Photo 1, contains the CC1010, an 8MHz crystal, a 32kHz crystal for low-power operations, an antenna matching/filter circuit, an inductor for the RF VCO, 2 choices of antennas, a 4-way selector switch, 4 LEDs, and connectors breaking out all external i/o connections to the CC1010. The hardware is based on ChipCon's reference design available at their website and the component values for the antenna circuit and VCO were calculated by SmartRF Studio, also downloadable from ChipCon's website (www.chipcon.com).

The software on the RFRAIN board running in the integrated microcontroller handles the media access using CSMA and the protocol needed for random-access, peer-to-peer networking. It also buffers incoming and outgoing data and provides a serial interface to a connected application microprocessor, although stand-alone applications can also be developed directly on the RFRAIN board. The software for the RFRAIN project was based on ChipCon's Simple Packet Protocol. The packet structure was modified to increase the address space from 8-bits to 16-bits and a packet ID was added. Chipcon's SPP software also contained code to handle Acknowledgments and Retries. The Retry code was removed here, as this was handled at a higher level to support the CSMA backoff scheme.

Listing 1 gives sample code that illustrates the transmission, retry, and backoff protocol. This code section resides in the main application loop and is executed while the receiver is looking for a preamble of a transmission. If the receiver is not idly looking for a preamble, this code does not get executed.

If there is a packet to transmit, *tx_outgoing* will be set and this code will begin to execute. The first thing that happens is that the Receive Signal Strength Indicator is read. This detects whether some sort of carrier is being received. If there is a carrier signal present, execution of this code stops, cycles around the main loop, then returns to this same check. When this check shows that the channel is empty, execution continues with a random wait. This prevents all the devices waiting to transmit on the same channel from being in sync and colliding with each other. If *tx_attempts* is equal to 0, we are about to transmit a new packet; if it is greater than 0, we are about to send a retry because a previous transmission did not receive an ACK. If it is a new packet, we load the packet into the transmit buffer, increment the packet ID, and set *tx_attempts* to the number of attempts that will be tried. After the packet is loaded and ready to go, the channel is checked again to see if it is clear. If it is clear we send the packet, and wait for the ACK if requested. If the channel is busy, we start all over again, wait for the channel to become clear, and perform the random wait. If an ACK is not received, we start the loop again, but without loading a new packet into the transmit buffer, and continue until we run out of retry attempts or an ACK is received. In between retries, we return to receive mode to check for incoming packets.

The reception and transmission of bytes in each packet is handled by an Interrupt Service Routine (ISR). Whenever the RF transceiver finishes receiving or transmitting a byte, it triggers the RF interrupt. The ISR related to this interrupt executes a Finite State Machine (FSM), the code for which is shown in Listing 2. The first function, *RF_ISR*, is the main interrupt service routine that the code vectors to upon completion of transmission or reception of a single byte. This function calls a callback function *sppRFStateFunc()*, which executes a function for the particular state of the finite state machine. Example FSM functions for receiving a packet are given following *RF_ISR*.

Each of these functions represents a single state in the FSM for reception. In general, these functions store the received byte appropriately, check if the packet is still valid, and set the callback function to the next state. The first state is executed after reception of the sync byte, upon which this byte is dropped and the state machine is advanced to the first Destination Address Byte. After receiving this byte, the firmware checks if the address is the receiving node's address or the broadcast address. If it is not a valid address, it sets a flag and continues listening. It does not stop receiving if the packet is not intended for it; instead it continues receiving, and drops the entire packet at the end. This keeps the node from transmitting while the channel is active (e.g., another two devices are communicating). This procedure, combined with the check of the signal strength indicator, works well to avoid interference.

The reception state machine then continues by receiving the lower byte of the destination address and repeating the check. It then receives the two byte source address, the packet ID (which is used elsewhere to determine if the packet is a repeat transmission), the length of the data payload, a byte containing flags, and a CRC8 byte as a check on the header. If the CRC8 fails, then the packet is dropped and the receiver is reset. If this check is successful, the receiver acquires the data payload and a CRC16 check on the data. If the data is valid, the function checks the flags to see if an acknowledgement is requested. If it is, the state machine will go to transmit mode and begin the acknowledgement process. If not, the state machine returns to idle, and the main application will be notified that a new packet has arrived.

We have written a series of similar state machine functions for sending the acknowledgement, sending a packet, and receiving an acknowledgment. This code is available on the RFRAIN website at http://www.media.mit.edu/resenv/rfrain.

Host Systems

A wide range of applications can be supported by the RFRAIN, as well as by similar circuits hosting embedded wireless devices. The RFRAIN was originally developed for use in a device called the UbER (Ubiquitous Experimental Research) Badge (Photo 2) - the latest in a series of badge platforms [4] developed at the MIT Media Lab. The UbER-Badge is designed to be worn as a digital nametag at large events such as trade shows and conventions. In addition to the RF link provided by the RFRAIN card, the UbER-Badge features line-of-sight IR transceivers for face-face communication, a 5x9 LED matrix display capable of presenting bitmap graphics and scrolling text that users in the vicinity can read, a 2x2 brightness-controllable blue LED matrix for indicating badge status, a 3-state pressable thumbwheel for user input, an onboard microphone sampled into 12 bits, a 12-bit audio output, a pager motor vibrator for tactile feedback, 3 onboard processors, capacity for up to 256MB of flash memory for storing audio or user data, provisions for connecting two LCD displays, and connectors that mate into the Stack Sensor platform (see below), allowing integration of a wide variety of different sensors. The RFRAIN system is used to network these badges in an ad-hoc, infrastructure-less way, allowing message passing, location of colleagues and points of interest, and transfer of contact information, all across a wide vicinity without the line-of-sight restriction imposed by the IR system, which talks only to a badge or IR beacon that's directly facing you. The UbER-badge uses the RFRAIN and its embedded microcontroller to handle all of its wireless communication activity. The RFRAIN stores incoming and outgoing packets and provides a simple serial interface to the main application processor on the badge, a Texas Instruments MSP430F149 16-bit microcontroller. Details on the UbER-Badge can be found at http://www.media.mit.edu/resenv/badge.

Wireless Sensing devices are another common use of these short-range radio modules, which have enabled compact sensor platforms such as the Berkeley Motes [5] and the Smart-Its (developed by a European university consortium) [6], through which researchers prototype sensor networks and ubiquitous computing

environments. Our group has developed several dense, multisensor wireless platforms for wearable applications. One example, shown in Photo 3, is card that acquires 16 different tactile and free-gesture sensor parameters from a dancer's shoe, producing data which are then interpreted by a rulebase running on a PC generating interactive music, putting the dancer in control of the composition. Data is acquired by a PIC 16C711 microcontroller and transmit right from the shoe by a Radiometrix TX-series transmitter. See http://www.media.mit.edu/resenv/danceshoe.html and [7] for more information. A more recent example in this area is our Stack Sensor Architecture [8], shown in Photo 4. The Stack is a series of compact interchangeable circuit boards, each performing a specific function such as 6-axis inertial measurement sensing, sonar proximity measurement, tactile interfaces (bend, pressure, capacitive sensors), and processing and wireless communication. The wireless board contains a RFM wireless transceiver and a Cygnal 8051-based processor that handles a TDMA communication protocol. It snaps together with any combination of sensor boards to form a compact wireless sensing suite. We are currently using this device in another shoe, this one designed in collaboration with the Biomotion Group at the Massachusetts General Hospital to acquire data for diagnosis of gait defects and to develop interactive therapy for patients with difficulty walking properly. For more information about the stack sensor architecture, please see http://www.media.mit.edu/resenv/Stack; and for more information about the Gait Sensing Shoe, please see http://www.media.mit.edu/resenv/gaitshoe.html.

Wireless Networking Standards and Upcoming Devices

Several higher-level standards have been established atop the low-level protocols that we have overviewed in the "Selecting a Short Range Radio Frequency Device" section. Although many manufactures provide chipsets that implement 802.11 (i.e., "Wi-Fi"), its power requirements and associated complexity often preclude adoption in small battery-operated devices such as described above, which don't often need its high data rates and heavy protocol stack. A significantly simpler and less power-hungry option is provided by Bluetooth, a well-established standard that describes hardware and a software protocol for wireless networking. By conforming to a standard such as Bluetooth, different manufacturers can insure that their devices will be able to communicate with one another. Several manufacturers, including Cambridge Silicon Radio (www.csr.com), SiliconWave (www.siliconwave.com), and Zeevo (www.zeevo.com), offer single chip solutions that not only perform the RF communication but also handle the high-level protocol described in the Bluetooth standard. These chips offload much of the required processing and allow the use of a low power microcontroller to handle the application software without sacrificing board space, power consumption, or cost. Furthermore, single chip solutions integrating the protocol stack, the RF hardware, and a microprocessor are starting to emerge from manufacturers such as Motorola and National Semiconductor.

With a maximum data rate of 1Mb/s and power consumption at 0.3mA in standby, 30mA maximum while transferring at full speed, Bluetooth is intended for high data rate applications such as streaming audio and video in battery-powered devices. A new standard, developed under IEEE 802.15.4, called ZigBee is starting to emerge that aims to do the same thing for lower bandwidth, lower power, longer range applications, such as home automation and sensor telemetry. ChipCon has recently announced an RFIC (the CC2420) that's ZigBee enabled, and Motorola is starting to enable devices such as sensors with ZigBee to provide single chip solutions that perform a task and handle all the communication hardware and software – the soon-to-be-released Motorola NeuRFon chips (e.g., the MC13192) will integrate an 802.15.4 radio with a digital state-machine to handle very low-level ZigBee protocol.

References

The authors wish to thank their colleagues in the Responsive Environments Group, in particular Ari Benbasat and Stacy Morris, for their contribution to several of the projects introduced in this article. We are grateful for the support of the *Things That Think* Consortium and other sponsors of the MIT Media Laboratory.

[1] J. C. Haartsen, “The Bluetooth Radio System,” *IEEE Personal Communications* 7, No. 1, 28-36 (2000).

[2] J. Adams, "Meet the ZigBee Standard," *Sensors Magazine*, June 2003. See also http://www.zigbee.org.

[3] Rappaport, T.S., **Wireless Communications: Principles and Practice (2'nd Edition)**, Prentice-Hall, NJ, 2002.

[4]. Borovoy, M. McDonald, F. Martin, M. Resnick, "Things That Blink: Computationally Augmented Nametags," *IBM Systems Journal*, Vol 35, Nos. 3 & 4, 1996. See also: http://web.media.mit.edu/~borovoy/.

[5] See: http://webs.cs.berkeley.edu/tos/hardware/hardware.html

[6] Holmquist, L.E., et. al., "Building Intelligent Environments with Smart-Its," to appear in *IEEE Computer Graphics and Applications Magazine*, December 2003.

[7] J. Paradiso, K. Hsiao, A. Benbasat, Z. Teegarden, "Design and Implementation of Expressive Footwear," *IBM Systems Journal*, Volume 39, Nos. 3 & 4, October 2000, pp. 511-529.

[8] A.Y. Benbasat, S.J. Morris, and J.A. Paradiso, "A Wireless Modular Sensor Architecture and its Application in On-Shoe Gait Analysis," in the Proceedings of the 2003 IEEE International Conference on Sensors, October 21-24.

Bios

Mat Laibowitz holds a diploma from Columbia University in Electrical Engineering and Computer Science. He is currently a second-year graduate student at the MIT Media Lab in the Responsive Environments Group, and a Motorola Fellow. Prior to joining the Media Lab he worked for 3 years at IBM Research in Yorktown Heights, NY, and worked for 2 years at XanBoo Inc, in New York City, where he developed wireless products for internet-controlled home automation. He has also studied film and animation at NYU and is active in the electronic music scene as a performer and instrument designer.

Joseph Paradiso, Sony Career Development Associate Professor at the MIT Media Lab, directs the Responsive Environments Group, which develops new sensor architectures for interactive systems. He received his Ph.D. in physics from MIT in 1981 and BS in electrical engineering and physics from Tufts University in 1977. Before coming to the Media Lab, he worked at Draper Laboratory in Cambridge and ETH in Zurich on high-energy physics detectors, spacecraft control systems, and underwater sonar, and has long been active in the electronic music community as a designer of synthesizers and musical interfaces.

Tables and Figures

	Aerocomm AC4490-1x1	Chipcon CC1010	Linx Technologies TR-916-SC-S	Maxstream 24XStream	Microchip rfPIC12C509AG	Nordic nRF24E1
Frequency Range	902-928 MHz	300-1000 MHz	916.5 MHz	2.4 GHz	310-480MHz	2.4 GHz
Methodology	FHSS, FSK	FSK	FSK	FHSS, FSK	OOK, FSK	FSK
Maximum Bit Rate	115.2 kb/s	76.8 kb/s	33.6 kb/s	20 kb/s	40 kb/s	1000 kb/s
Standby Power	30 mA	1.3 mA	50 uA	10 uA	0.1 uA	2 uA
Receiver Power	30 mA	9.1 mA	15 mA	50 mA	N/A	19 mA
Transmit Power	35 mA @ 2dBM	8.9mA @ -5dBM	29 mA @4dBM	150 mA @ 17dBM	11.5 mA @ 2dBM	10.5mA @ -5dBM
Receiver Sensitivity	-96 dBM	-107 dBM	-95 dBM	-102 dBM	N/A	-90 dBM
RSSI Output	N/A	Yes	Yes	N/A	N/A	No
Package and Size	OEM Module 26mm x 26mm	64TQFP 12mm x 12mm	Module 33mm x 37mm	Module 40mm x 68mm	18SSOP 10mm x 12mm	36QFN 6mm x 6mm
External Components (not including antenna)	None	RX Match, TX Match, VCO Inductor, Crystals for uC	None	None	Crystal for transmitter, Passives	EEPROM for code, Crystal for μC
OEM Module Available	Yes	No	Yes	Yes	No	No
Comments	Frequency Hopping Spread Spectrum performed by module. Embedded protocol handles headers, acks, rts/cts for streaming data, addressing, error checking.	Integrated 8051, hardware DES, supports frequency hopping, supports two crystals for low power operation	Ready-made keychain remotes and other compatible devices are available	Frequency Hopping Spread Spectrum performed by module. Integrated wire antenna available. Embedded protocol handles acks, retries, and modem-like communications.	Transmitter only – receiver is a separate device, the rfRXD0420	Integrated 8051, power ratings are for RF and don't include mcu, supports frequency hopping
Microcontroller Specifications	N/A	8051 based 22 MHz 32kb flash 2kb ram 10 bit ADC 2 UARTs SPI, WDT, PWM	N/A	N/A	4 MHz 1024 bytes OTP 41 bytes ram WDT	8051 based 16MHz Requires external flash for program storage 4kb RAM 12 bit ADC SPI, WDT, PWM
Website	www.aerocomm.com	www.chipcon.com	www.rfdigital.com	www.maxstream.com	www.microchip.com	www.nvlsi.no

	Radiometrix BiM3	Rayming RE-99, TX-99	RF Monolithics TR1100	RFWaves RFW102-M	Texas Instruments TRF6901	Xemics XE1203
Frequency Range	869/914 MHz	300 MHz	916.5 MHz	2.4 GHz	860-930MHz	433Mhz, 868Mhz, and 915Mhz
Methodology	OOK	OOK	OOK	DSSS, OOK	OOK, FSK	FSK
Maximum Bit Rate	64 kb/s	1.2 kb/s (see comments)	1000 kb/s	1000 kb/s	64 kb/s	152.3 kb/s
Standby Power	11.5 mA	1.6 mA	0.7 uA	2.6 uA	4 uA	1.10 mA
Receiver Power	11.5 mA	1.6 mA	8 mA	38 mA	21 mA	17 mA
Transmit Power	11 mA @ 3dBM	1.6 mA	12mA @ 0dBM	21mA @ 2dBM	40 mA @ 8dBM	40mA @ 5dBM
Receiver Sensitivity	-100 dBM	50 ft range guaranteed, 100 ft expected	-87 dBM	-80 dBM	-86 dBM	-100 dBM
RSSI Output	Yes	No	No	No	Yes	Yes
Package and Size	Module 33mm x 23mm	OEM modules 30mm x 60mm	SM-20H 8mm x 11mm	Module 11mm x 16mm	48PQFP 9mm x 9mm	48VQFN 7mm x 7mm
External Components	None	None	Antenna Filter, Passives	None	Crystal, Antenna Matching Circuitry, Passives (LC Tank circuit)	RX Match, TX Match, VCO Tank, Freq Synth Filter, Crystal
OEM Module Available	Yes	Yes, only	Yes (dr3000-2)	Yes	No	Yes
Comments	Two versions available for different operating frequencies. Drop-in OEM module. Many other products available from Radiometrix.	Transmitted and Receiver pair. Switch Encoders/Decoders are available. They have several ready made keychain remotes available.	OEM Module is recommended which requires no external components	3 chip solution on module, Built-in DSSS protocol, simple interface to main processor	Simple 3-wire serial interface	Several other chips available.
Microcontroller Specifications	N/A	N/A	N/A	N/A	N/A	N/A
Website	www.radiometrix.com	www.rayming.com	www.rfm.com	www.rfwaves.com	www.ti.com	www.xemics.com

Table 1: Basic specifications for several currently-available small-footprint, short-range RF devices

```
if (tx_outgoing == 1) {                         // IF WE HAVE A PACKET TO SEND

        tmp_RSSI = rfdcReadRSSI();              // READ SIGNAL STRENGTH

        if (tmp_RSSI < RSSI_THRESHOLD){         // CARRIER SENSE
                // NO CARRIER DETECTED
                halWait(rand(), CC1010DC_CLKFREQ);    // RANDOM WAIT

                // CHECK IF WE HAVE MORE ATTEMPTS TO MAKE ON LAST TX
                if (txAttempts == 0){
                        // NO MORE ATTEMPTS
                        // LOAD NEXT PACKET FROM BUFFER

                        . . .

                        txAttempts = myRetryCount;     // RESET ATTEMPTS
                        TXI.pkid = ++my_pkid;          // INC PACKET ID
                }

                tmp_RSSI = rfdcReadRSSI();     // READ SIGNAL STRENGTH

                // CHECK CARRIER SENSE AND THAT RF TRANSCEIVER IS NOT CURRENTLY RECEIVING A PACKET
                if ((RXI.status == SPP_RX_WAITING) && (tmp_RSSI < RSSI_THRESHOLD)) {
                        // MEDIA FREE
                        sppReset();            // RESET RECEIVER

                        // TRANSMIT PACKET
                        if (sppSend(&TXI) == SPP_TX_STARTED) {
                                BLED = LED_ON;                // INDICATE TRANSMISSION

                                // TRANSMISSION IS INTERRUPT DRIVEN
                                // ISR WILL UPDATE SPP_STATUS
                                // WAIT UNTIL STATUS CHANGES
                                do {  } while (SPP_STATUS() != SPP_IDLE_MODE);
                                // CHECK TRANSMIT STATUS
                                if (TXI.status == SPP_TX_FINISHED) {
                                        // PACKET SENT SUCCESSFULLY, ACK RECEIVED IF ACK REQUESTED
                                        GLED = LED_ON;
                                        // SEND RETURN CODE
                                        . . .
                                } else {
                                        // PACKET FAILED, COLLISION OR NO ACK
                                        if (--txAttempts == 0) {
                                                // NO MORE RETRIES
                                                RLED = LED_ON;
                                                // CHECK IF THERE ARE ANY OTHER PACKETS TO SEND
                                                if(tx_buffer_empty()) tx_outgoing = 0;
                                                // SEND RETURN CODE
                                                . . .
                                        } // ELSE A RETRY WILL OCCUR ON NEXT LOOP
                                }
                        }
                        // RESTART RECEIVING ALLOWING CARRIER SENSE FOR NEXT RETRY OR NEW PACKET
                        sppReceive(&RXI);
                }
        }
}
```

Listing 1: Embedded C Code for the RFRAIN CSMA Transmission

```
//-------------------------------------------------------------------------
//  void RF_ISR (void) interrupt INUM_RF
//
//  Description:
//      RF interrupt service routine
//              Runs the SPP finite state machine
//
//      The packet sequence bit:
//              TX: Toggle when finished (sppIntData.pTXI->status = SPP_TX_FINISHED).
//              RX: The user application must examine the sequence bit.
//-------------------------------------------------------------------------
void RF_ISR (void) interrupt INUM_RF {

        INT_GLOBAL_ENABLE (INT_OFF);
        INT_SETFLAG (INUM_RF, INT_CLR);

        if (sppIntData.mode != SPP_IDLE_MODE) {
                sppRFStateFunc();
        }

        INT_GLOBAL_ENABLE (INT_ON);
        return;
} // RF_ISR

//-------------------------------------------------------------------------
// RX FUNCTIONS
//-------------------------------------------------------------------------
void RX_WAIT (void) {
        // Drop the sync byte
        RF_LOCK_AVERAGE_FILTER(TRUE);
        sppIntData.mode = SPP_RX_MODE;
        sppRFStateFunc = RX_DABH;
}

void RX_DABH (void) {
        sppIntData.pRXI->status = SPP_RX_RECEIVING;
        FAST_CRC8_INIT(sppIntData.crc8);
        FAST_CRC8(RF_RECEIVE_BYTE(), sppIntData.crc8);
        if ((RF_RECEIVE_BYTE() == sppSettings.myAddressH) || (RF_RECEIVE_BYTE() == SPP_BROADCAST_H)) {
                sppIntData.rx_address_valid = 1;
        } else {
                sppIntData.rx_address_valid = 0;
        }
        sppRFStateFunc = RX_DABL;
}

void RX_DABL (void) {
        sppIntData.pRXI->status = SPP_RX_RECEIVING;
        FAST_CRC8(RF_RECEIVE_BYTE(), sppIntData.crc8);
        if ((RF_RECEIVE_BYTE() == sppSettings.myAddressL) || (RF_RECEIVE_BYTE() == SPP_BROADCAST_L)) {
        } else {
                sppIntData.rx_address_valid = 0;
        }
        sppRFStateFunc = RX_SABH;
}

void RX_SABH (void) {
        sppIntData.pRXI->sourceH = RF_RECEIVE_BYTE();
        FAST_CRC8(RF_RECEIVE_BYTE(), sppIntData.crc8);
```

```
        sppRFStateFunc = RX_SABL;
}

void RX_SABL (void) {
        sppIntData.pRXI->sourceL = RF_RECEIVE_BYTE();
        FAST_CRC8(RF_RECEIVE_BYTE(), sppIntData.crc8);
        sppRFStateFunc = RX_PKID;
}

void RX_PKID (void) {
        sppIntData.pRXI->pkid = RF_RECEIVE_BYTE();
        FAST_CRC8(RF_RECEIVE_BYTE(), sppIntData.crc8);
        sppRFStateFunc = RX_DATA_LEN;
}

void RX_DATA_LEN (void) {
        sppIntData.pRXI->dataLen = RF_RECEIVE_BYTE();
        FAST_CRC8(RF_RECEIVE_BYTE(), sppIntData.crc8);
        sppRFStateFunc = RX_FLAG;
}

void RX_FLAG (void) {
        sppIntData.pRXI->flags = RF_RECEIVE_BYTE();
        FAST_CRC8(RF_RECEIVE_BYTE(), sppIntData.crc8);
        sppRFStateFunc = RX_CRC8_HEADER;
}

void RX_CRC8_HEADER (void) {
        FAST_CRC8(RF_RECEIVE_BYTE(), sppIntData.crc8);
        sppIntData.counter = 0;

        if (sppIntData.crc8 == CRC_OK) {
                if (sppIntData.pRXI->dataLen == 0 && sppIntData.rx_address_valid == 1) {
                        SPP_DISABLE_TIMEOUT();
                        if (sppIntData.pRXI->flags & SPP_ACK_REQ) {
                                SPP_FAST_POWER_UP_TX();
                                sppRFStateFunc = RXACK_START;
                        } else {
                                sppIntData.pRXI->status = SPP_RX_FINISHED;
                                FSM_RESET();
                        }
                } else if (sppIntData.pRXI->dataLen > sppIntData.pRXI->maxDataLen) {
                        sppIntData.pRXI->status = SPP_RX_TOO_LONG;
                        FSM_RESET();
                } else {
                        sppRFStateFunc = RX_DBX_START;
                }
        } else {
                FSM_RESTART_RX();
        }
}

void RX_DBX_START (void) {
        sppIntData.pRXI->pDataBuffer[ sppIntData.counter]  = RF_RECEIVE_BYTE();
        FAST_CRC16_INIT(sppIntData.crc16);
        FAST_CRC16(RF_RECEIVE_BYTE(), sppIntData.crc16);
        sppIntData.counter = 1;
        if (sppIntData.counter == sppIntData.pRXI->dataLen) {
                sppRFStateFunc = RX_CRC16_DATA_H;
```

```
        } else {
                sppRFStateFunc = RX_DBX;
        }
}

void RX_DBX (void) {
        sppIntData.pRXI->pDataBuffer[ sppIntData.counter]  = RF_RECEIVE_BYTE();
        FAST_CRC16(RF_RECEIVE_BYTE(), sppIntData.crc16);
        sppIntData.counter++;
        if (sppIntData.counter == sppIntData.pRXI->dataLen) {
                sppRFStateFunc = RX_CRC16_DATA_H;
        }
}

void RX_CRC16_DATA_H (void) {
        FAST_CRC16(RF_RECEIVE_BYTE(), sppIntData.crc16);
        sppRFStateFunc = RX_CRC16_DATA_L;
}

void RX_CRC16_DATA_L (void) {
        FAST_CRC16(RF_RECEIVE_BYTE(), sppIntData.crc16);
        if (sppIntData.crc16 == CRC_OK && sppIntData.rx_address_valid == 1) {
                SPP_DISABLE_TIMEOUT();
                if (sppIntData.pRXI->flags & SPP_ACK_REQ) {
                        SPP_FAST_POWER_UP_TX();
                        sppRFStateFunc = RXACK_START;
                } else {
                        FSM_RESET();
                        sppIntData.pRXI->status = SPP_RX_FINISHED;
                }
        } else {
                FSM_RESTART_RX();
        }
}
```

Listing 2: Embedded C Code for the RFRAIN Reception FSM and ISR

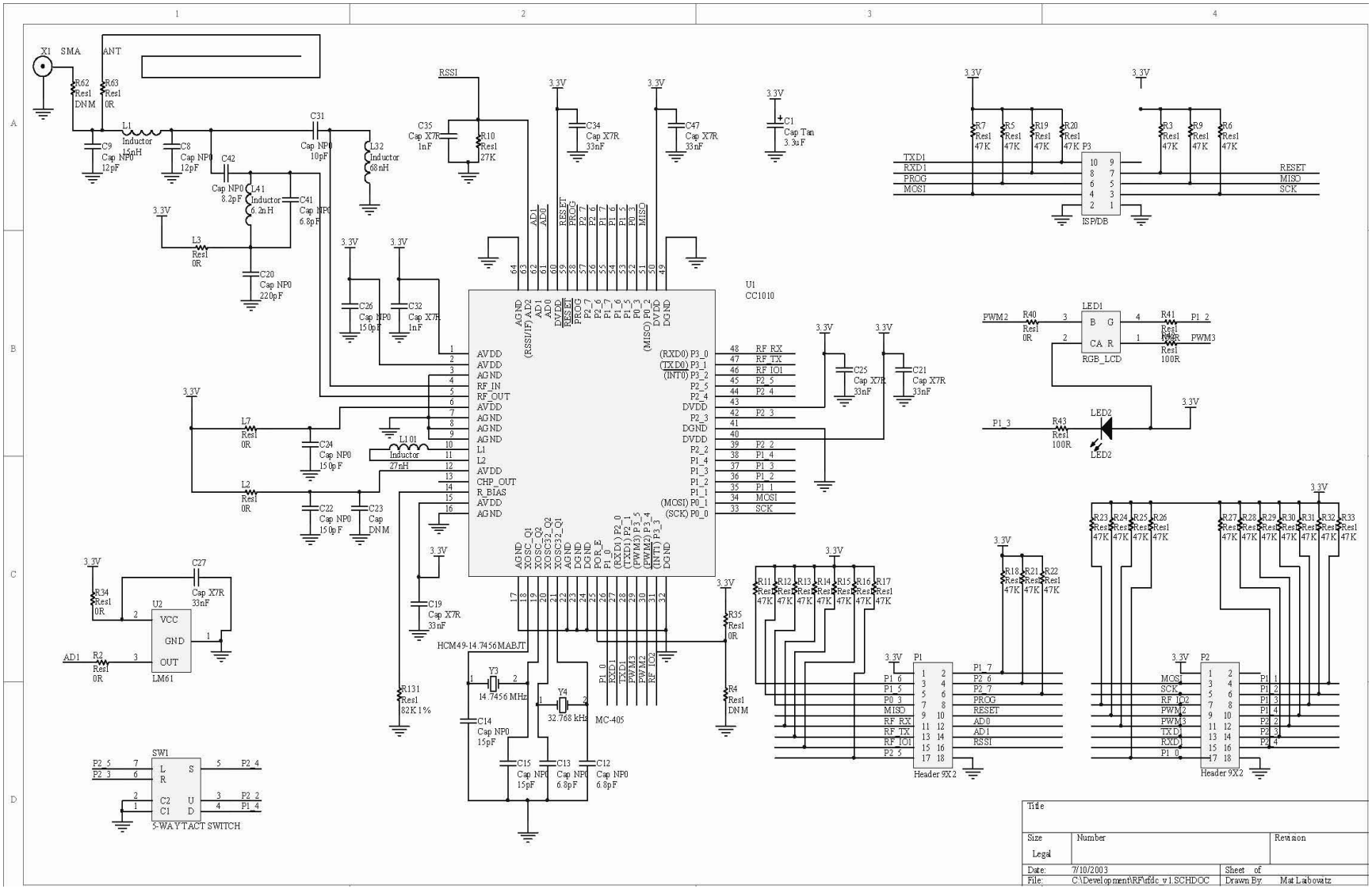

Figure 1: Schematic Diagram for the RFRAIN

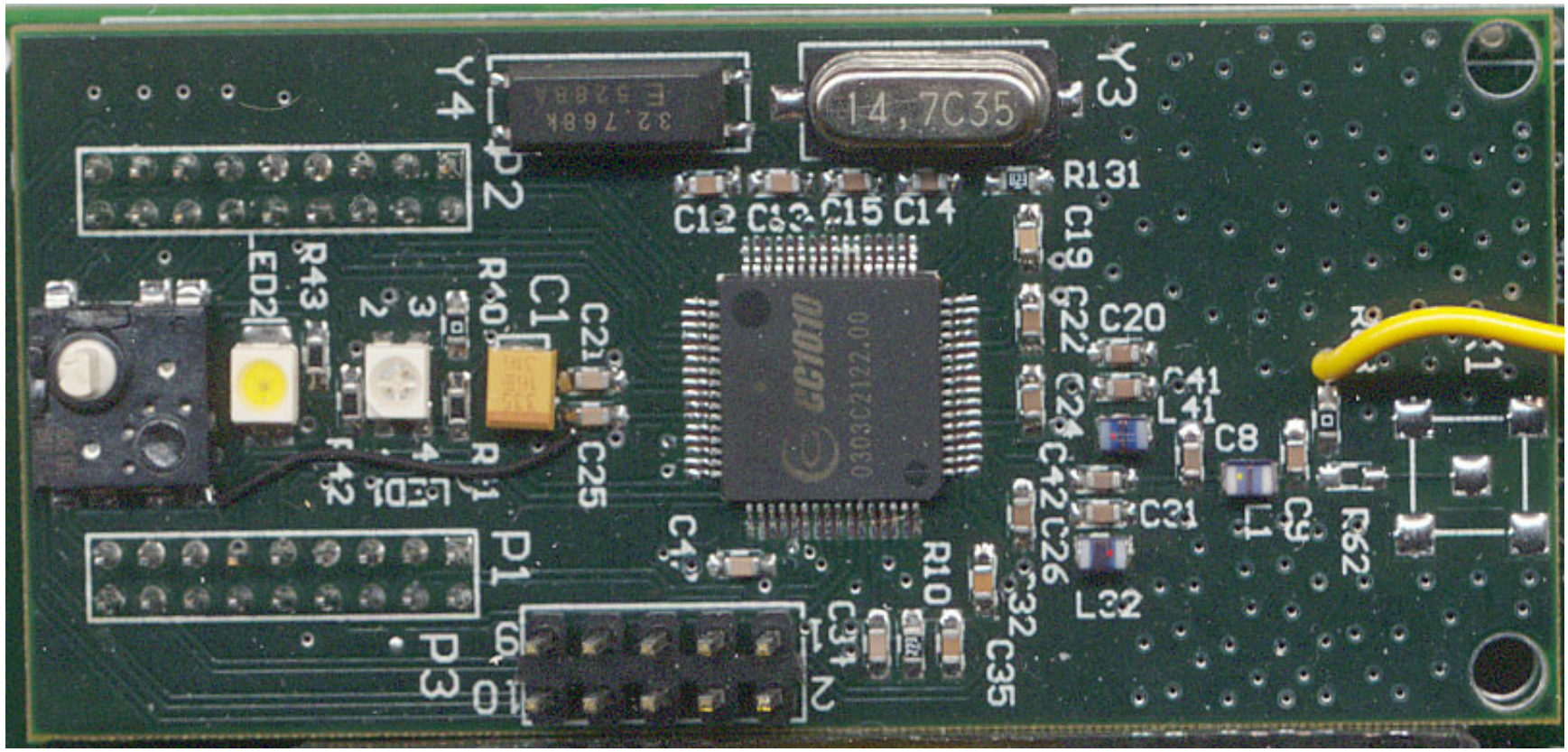

Photo 1: Photograph of a functional RFRAIN card (wire antenna protrudes from the right)

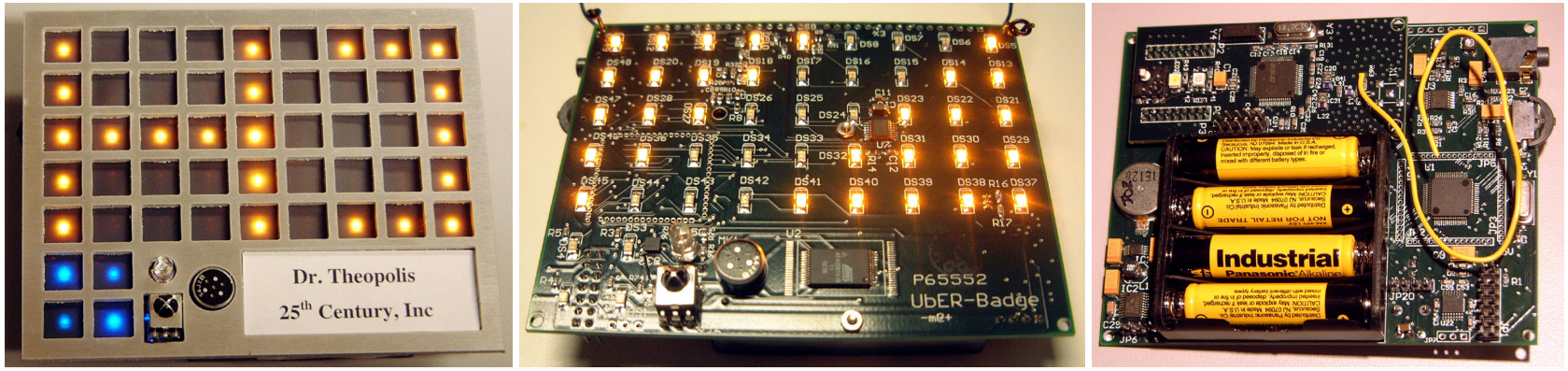

Photo 2: The UbER-Badge - with cover (left), without cover (center), and rear (right)

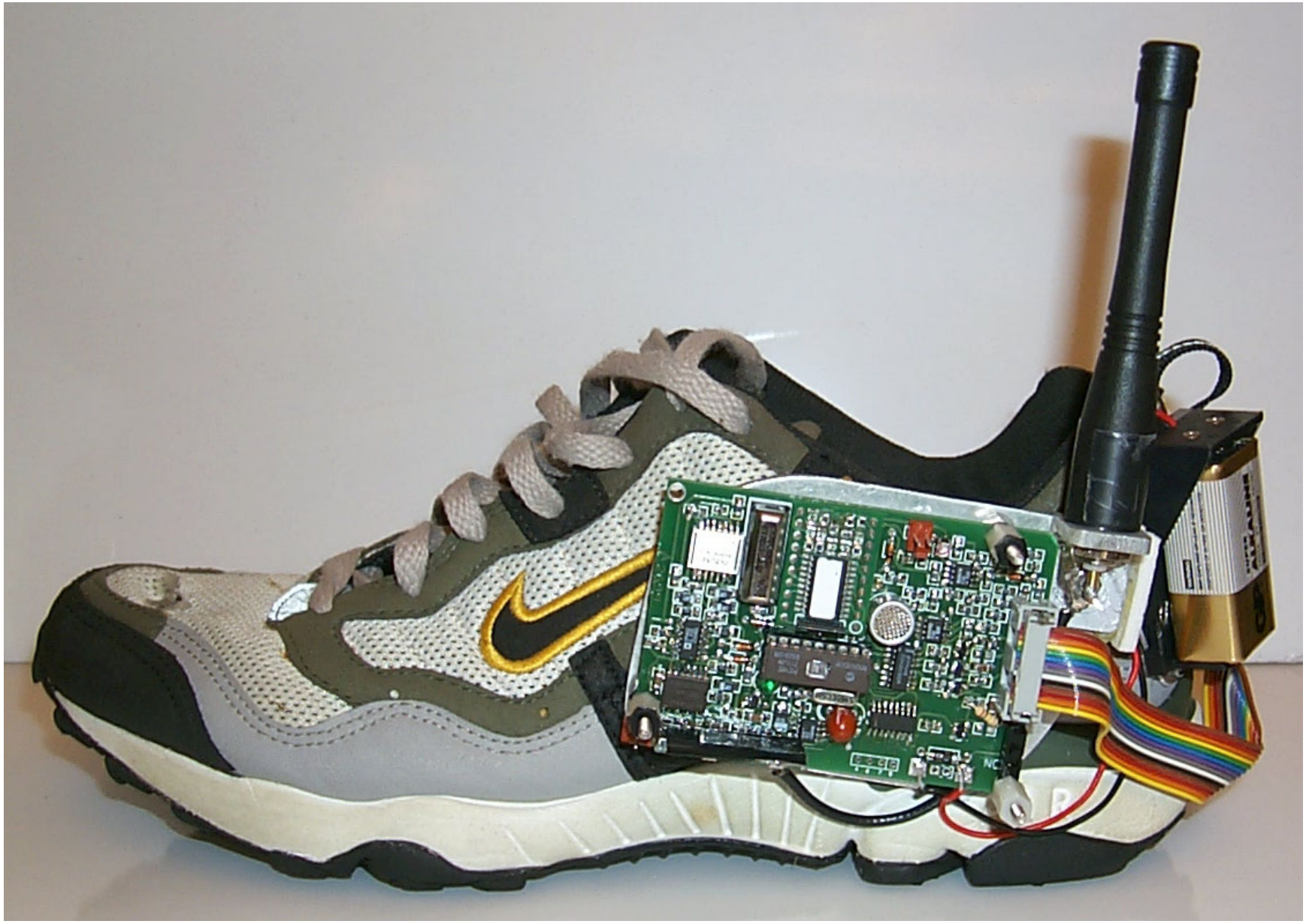

Photo 3: The Expressive Footwear - 16 diverse sensor channels wirelessly transmit from the foot of a performer for producing dance-driven interactive music

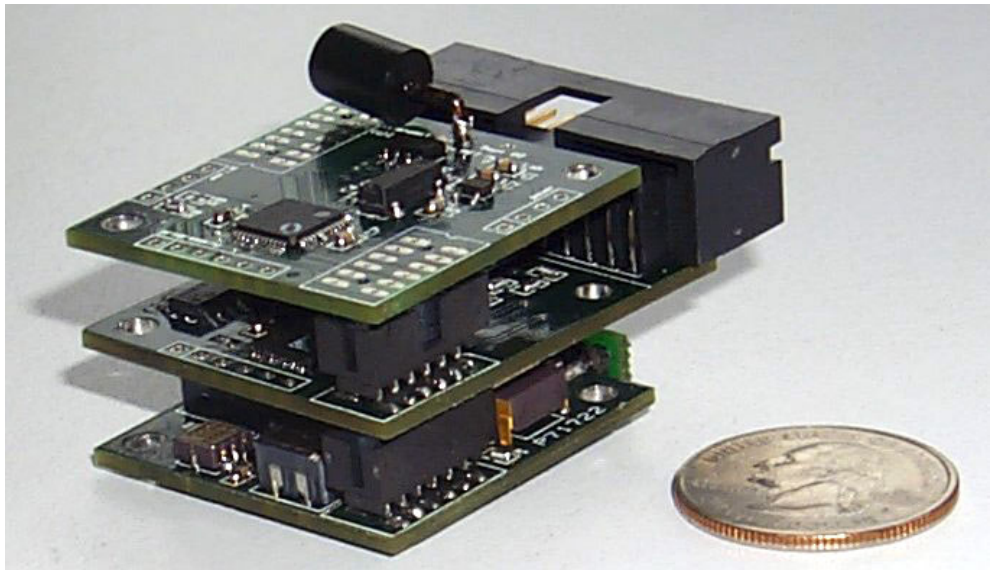

Photo 4: The Stack Sensor Architecture - a compact, configurable, multimodal wireless sensing platform (top) and its application in a shoe to monitor gait characteristics (bottom)