Implementierungsalternative für einen flexiblen drahtlosen LAN Transceiver

(Implementation alternatives for a flexible wireless LAN transceiver)

Ir. Roel (R.) Schiphorst, Ir. Fokke (F.W.) Hoeksema and Prof. Dr. Ir. Kees (C.H.) Slump

University of Twente, Department of Electrical
Engineering, Mathematics and Computer Science (EEMCS),
Signals and Systems group (SaS),

P.O. box 217 - 7500 AE Enschede - The Netherlands

Phone: +31 53 489 2770 Fax: +31 53 489 1060

Email: {r.schiphorst, f.w.hoeksema, c.h.slump}@utwente.nl

Abstract—In our software-defined radio project we have implemented two different types of standards, a continuous-phase-modulation (CPM) based standard, Bluetooth, and an OFDM based standard, HiperLAN/2, on a general-purpose processor.

First we describe our baseband software-defined radio testbed for the physical layer of wireless LAN standards. All physical layer functions have been successfully mapped on a Pentium 4 processor that performs these functions in real-time. The testbed consists of a transmitter PC with a DAC board and a receiver PC with an ADC board. Channel selection functionality is performed at the DAC and ADC board, whereas all modulation and demodulation functions are mapped on software running on the CPU.

Then, three implementation alternatives for the digital part of the transceiver are introduced. These include:

- the testbed: a PCI card equipped with analog front-end functionality. All demodulation functions are performed in software running on the CPU of the notebook.
- integration of analog front-end functionality in the chipset of the motherboard and demodulation functions are performed by the processor.
- a low power DSP plus analog front-end functionality mounted on a PCI card.

The alternatives are evaluated with respect to computational-power requirements, power consumption and expected manufacturing costs.

Index Terms:

software-defined radio, testbed, baseband, physical layer, HiperLAN/2, Bluetooth.

I. INTRODUCTION

New wireless communications standards do not replace old ones, instead the number of standards keeps on increasing and by now an abundance of standards already exists, see Table I. Moreover, there is no reason to assume that this trend will ever stop. Therefore the software-radio concept is emerging as a potential pragmatic solution: a software implementation of the user terminal able to dynamically adapt to the radio environment in which the terminal is located [1].

Because of the analog nature of the air interface, a software radio will always have an analog front end. In an ideal software radio, the analog-to-digital converter (ADC) and the digital-to-analog converter (DAC) are positioned directly after the antenna. Such an implementation is not feasible due to the power that such device would consume and other physical limitations [2] [3]. It is therefore a challenge to design a system that preserves most properties of the ideal software radio while being realizable with current-day technology. Such a system is called a software-defined radio (SDR).

Software-defined radio has both advantages for consumers and manufactures because current products support only a fixed number of standards. Figure 1 shows the lifetime of products and wireless standards. One can see that current-day products support a fixed number of standards and in time new standards emerge and old ones disappear, making a product eventually obsolete.

Software-defined radios on the other hand will enable consumers to upgrade their radio with new functionality e.g. required by new standards, just by software updates,

Fig. 1. Standards support by products in time

without the need for new hardware. Moreover, manufacturers can upgrade or improve functionality of consumer-owned products and SDR could result in shorter development time, cheaper production due to less components. However, the downside of SDR is an increased power consumption, because dedicated designs are more power efficient. Especially for mobile applications power consumption is important.

II. SOFTWARE-DEFINED RADIO

Figure 2 depicts the mapping of the different functions as structure by the OSI model on software/hardware in current radio designs. The physical layer is generally implemented in hardware and higher layers are often software based with the Logical Link Control (LLC) and Multiple Access Layer (MAC) layer as transition area. In our SDR project [4] we research whether the lowest layer, the physical layer of wireless standards can be implemented in software running on a general purpose processor and estimate the costs of such an implementation with respect to power consumption and computational power requirements.

Thus, we interpret SDR as an implementation technology which differs from the views in [1] and [5]: flexible, universal, radio systems at each layer of the OSI model where manufacturers, network operators and consumer can benefit. Our interpretation on SDR is more focussed on the physical layer; an implementation technology, invisible for consumers. Moreover, we want to investigate if we can use existing processing capabilities (for example a notebook's CPU) for digital signal processing purposes thereby possibly prolonging the lifetime of a device. Moreover, it saves hardware and Moore's law will lower in time the computational load as a percentage of the computational capacity.

A flexible, all standard radio will always consume more energy than a dedicated radio, thus the first application for a flexible radio will be an application where power consumption is less an issue; an example being a flexible radio in a notebook. This application for SDR has three advantages. First, we can use the processing capabilities of the general purpose processor for digital

Fig. 2. Mapping of protocols in current designs on hardware/software

Fig. 3. Project scope

signal processing purposes. Second, in comparison to SDR for mobile phones, our application can consume much more power (in the order of 1 W). Third, a notebook is very suited for demonstration purposes.

Table I gives an overview of important wireless standards together with the used frequency bands and modulation type. It seems that each standard can be seen as a family of standards, an example being GSM. Thus the number of existing standards that manufacturers have to support is even larger than one would initially expect. However, there are also similarities among them: the used frequency bands are between 0.8 and 6 GHz with dominant frequency bands around the 0.8 GHz, 2 GHz, 2.4 GHz and 5 GHz. In addition three types of modulation are used, CPM (continuous-phase-modulation), OFDM (Orthogonal Frequency Division Multiplexing) and CDMA (Code Division Multiple Access).

In our SDR project we decided not to focus on an all-standard radio but to start with a software-defined radio for wireless LAN standards first. The research is carried out by two chairs of the University of Twente: the IC-Design group which focusses on the RF part and the Signals and Systems group focussing on the baseband part. At the project's start we defined also the scope of project: the physical layer of the OSI model. Recent literature [6] indicates however that especially error correction decoding (Viterbi algorithm) requires most computational power in the lower layers of a system. Figure 3 summarizes the design goal of our project, a notebook with a wideband RF frontend with a software implementation of the physical layer.

Wireless LAN standards use phase modulation or OFDM in the 2.4 GHz or 5 GHz frequency band, so we decided in our project to combine an instance of a continuous-phase modulation standard (Bluetooth) with an OFDM standard (HiperLAN/2).

HiperLAN/2 [7] is a high-speed Wireless LAN (WLAN) standard using OFDM. Its physical layer is very similar to the 802.11a standard. Bluetooth [8] on the other hand is a low cost, low speed standard, designed for replacing fixed cables. Bluetooth uses Gaussian Frequency Shift Keying (GFSK) which is also used by other standards such as HomeRF and DECT.

This paper discusses only the digital baseband part of the project. More information about the total project can be found in [9] or at the project's website [4]. The rest of the paper is organized as follows: First the functional architecture of the physical layer of both standards is discussed, which is followed by a description of the SDR baseband receiver. Then implementation alternatives for the transceiver are introduced and evaluated. The alternatives are:

- 1) The testbed: a PCI card equipped with analog front-end functionality. The analog front-end functionality consists of 2 ADCs and 2 DACs plus analog reconstruction filters and digital anti-aliasing filters. All demodulation functions are performed in software running on the CPU of the notebook.
- 2) Integration of the analog front-end functionality in the chipset of the motherboard and all other functions are performed by a Pentium 4 processor.
- 3) A low power DSP together with the analog front-end mounted on a PCI card; so all receiver functions are performed by the PCI card.

The paper concludes by comparing the alternatives with respect to computational-power requirements, power consumption and expected manufacturing costs.

III. SDR BASEBAND TRANSCEIVER

Although we show that a real-time software implementation of the receiver (and transmitter) functionality is possible using the notebook's processor, it requires besides processing power also a real-time operating system. *Traditional* operating systems such as Windows or Linux are non-real-time, e.g. the latency of the system is undefined and can be up to 100 ms for Linux [10]. So it is possible that our transceiver *program* misses a buffer and data is lost. A solution to this problem is, for example, to apply special patches to the Linux kernel, which reduce this maximal latency to about 5 μ s[10]¹. In our testbed we use large sample buffers of 100 ms to avoid the influence of the operating system but additional research is needed to find the maximal allowable latency which is likely determined by the MAC layer. Furthermore, we have to investigate if this value

¹A HiperLAN/2 MAC frame has a duration of 2 ms and the shortest Bluetooth MAC frame is 0.625 ms long.

Fig. 4. Functional architecture of the Bluetooth transmitter and receiver

can be achieved in our testbed. So at the moment, our transceiver can only be used for continuous transmission of MAC bursts.

A. Functional architecture

Figure 4 depicts the functional architecture of the Bluetooth transmitter and receiver. The first step in the transmitter is to embed the raw bits into MAC bursts which are then BPSK modulated at 1 Mbit/s. The BPSK symbols are filtered by a Gaussian low pass filter and the filtered output is connected to an VCO that translates the amplitude variation into frequency variations. At the receiver side, the first step is to select the wanted Bluetooth channel and suppressing all others which is performed both digitally and by the analog front-end. This is achieved by mixing the wanted channel to zero IF and applying a low-pass filter. The next step is to demodulate the FM signal into an AM signal by taking the derivative of the phase. Because a frequency offset introduces an offset in the AM signal, it has to be corrected before the bit decision.

On the other hand Figure 5 depicts the HiperLAN/2 physical layer architecture which is very different from the Bluetooth architecture one. The HiperLAN/2 transmitter starts with mapping raw bits on QAM symbols (either BPSK, QPSK, 16-QAM or 64-QAM symbols). In the next step, the QAM symbols are mapped on data carriers and an OFDM symbol is constructed by adding pilot carriers, applying an inverse FFT and adding an prefix, which results in a 20 MSPS signal. MAC bursts are then created by adding special symbols, preambles, to the start of the MAC burst.

The HiperLAN/2 receiver starts by searching for the start of a MAC burst. If found, it estimates the frequency offset and channel parameters. After these steps the data OFDM symbols can be demodulated by first correcting the frequency offset, performing an FFT, correcting the channel and detecting and correcting the phase offset

Fig. 5. Functional architecture of the HiperLAN/2 transmitter and receiver

Fig. 6. Functional architecture of the Bluetooth-enabled HiperLAN/2 receiver

by using the pilot tones. The output are QAM symbols which have to be de-mapped into raw bits.

Although the functional architecture of both standards is very different, we have successfully integrated the Bluetooth receiver functionality into the HiperLAN/2 receiver [11] by using a (simplified) maximum a posteriori probability (MAP) receiver which is a more advanced Bluetooth demodulation algorithm [12]. In this testbed, however, we did not implement this receiver (yet) but used instead a conventional receiver such as depicted in Figure 4.

B. Testbed setup

Figure 7 shows the component architecture of our SDR testbed. The testbed consists of four components: a transmitter PC, a DAC PC board, a receiver PC and an ADC PC board.

The transmitter PC continuously generates HiperLAN/2 or Bluetooth MAC bursts which are sent in real-time to the DAC board at 20 MSPS by using an Adlink cPCI-7300 digital I/O PCI card. This DAC board converts the digital signal into a complex analog baseband signal. The ADC board samples the analog signal with 80 MSPS and the onboard Intersil ISL5416 programmable down converter (SRC) decimates the digital signal into a complex 20 MSPS signal in HiperLAN/2 mode and into a 5 MSPS signal (including mixing of

Fig. 7. Component architecture of the SDR testbed (baseband only)

Fig. 8. Implementation alternatives

the wanted channel to baseband) for Bluetooth. This signal is transported to the receiver PC by using another Adlink cPCI-7300 digital I/O PCI card. The receiver PC performs all demodulation functions and demodulates the MAC bursts real-time.

IV. IMPLEMENTATION ALTERNATIVES

This section describes three implementation alternatives (Figure 8) for our transceiver. The first alternative is the testbed: a PCI card equipped with 2 ADCs and 2 DACs plus analog reconstruction filters and digital anti-aliasing filters. Moreover the demodulation functions are performed on the notebook's CPU. This implementation requires no extra DSP hardware which saves costs. However the PCI interface is fully utilized, disabling the use of other applications. In addition, the CPU is very power inefficient relative to an ASIC, although one can argue that the CPU is always consuming power whether it is used for our *program* or not.

To overcome the large load on the PCI bus we propose an implementation alternative where the ADCs, DACs and filters are implemented in the chipset of the notebook. As there is already a large bandwidth available between chipset and CPU, this SDR application can easily be added. Power consumption will be slightly less than the first alternative due less inter-chip communication.

Moreover this integrated solution is more cost effective than separate chips.

The last implementation alternative is a PCI board equipped with a low power DSP (e.g. a TI C64x DSP), ADCs, DACs and filters. So all receiver functions are performed by this PCI card. As DSPs are designed to compute digital signal processing algorithms at a low power consumption ($< 1W$), this solution is the most power efficient of the considered alternatives.

In the next section, the computational power requirements plus power consumption of the alternatives are assessed.

V. COMPUTATIONAL POWER REQUIREMENTS

A. User scenarios

For both standards, Bluetooth and HiperLAN/2, we derived a user scenario to estimate and measure the computational requirements, assuming continuous transmission. This scenario can be compared with a realistic scenario that includes the influences of the higher OSI layers on the physical layer.

1) *Bluetooth user scenario:* The Bluetooth symbol duration is $1 \mu s$ and data is transmitted in time slots with a duration of $625 \mu s$ [8]. For estimating computational requirements, we assume maximal transfer rate. In this mode, Bluetooth uses a packet which spans 5 time slots and 1 time slot is used for uplink communication.

2) *HiperLAN/2 user scenario:* A HiperLAN/2 MAC frame consists of 5 parts and has a maximal duration of 2 ms [7]. We assume that all parts have equal duration and that we have to demodulate 2 parts (one common and one user part).

B. Requirements

We used the user scenarios of both standards for the implementation of the transmitter and receiver. This section presents the required computational power for each function that is mapped on the Pentium 4 processor and the low-power TI C64x DSP. The computational requirements for the Pentium 4 are determined by measuring the number of cycles per second for each transceiver function. For the TI DSP on the other hand we assessed the load of each function by using benchmark results.

1) *Software:* The source code of the Bluetooth and HiperLAN/2 transmitter and receiver is written in C and compiled with the Intel compiler 7.1 under Linux, using floating-point precision because floating-point operations are as fast as fixed-point operations on a Pentium 4².

²We expect that floating-point instructions will have a higher power consumption than similar fixed-point instructions. However, we expect that this difference is very small compared with the total power consumption of a Pentium 4.

Moreover we used the open-source FFTW library [13] for computing the inverse FFT and FFT. As a DAC requires fixed-point numbers, the transmitter has to convert the floating-point numbers into fixed point. The receiver, on the other hand, receives fixed-point numbers from the ADCs, so it has to do the inverse process. It was observed that these conversions take a long time to compute and therefore special SIMD (Single Input Multiple Data) instructions [14] are used for acceleration.

2) *Measurement method used in the testbed:* Time measurements were performed on a Pentium 4 at 2.8 GHz by counting the number of cycles for each function. A Pentium 4 processor is a very complex design and therefore the number of cycles needed for computing a particular function, is influenced by many parameters such as cache misses, memory alignment, etc. It is for that reason that we used average values in these time measurements. The number of cycles required for the whole receiver or transmitter function (total values) is measured separately and not determined by summing up the measurement results for all individual components.

3) *Estimation method used for the TI C64x DSP:* The TI C64x DSP is a fixed point DSP family, so all computations are performed 16-bit or 32-bit fixed point. Simulations of our HiperLAN/2 transceiver implementation in 64-QAM mode revealed that the receiver requires at least 7-bit quantized input values for error-free reception. So we expect that 16-bit fixed-point calculations can be used. We estimated the computation power for each transceiver function by using available benchmarks for the TI C64x DSP [15], [16] or if there were no benchmarks available for the used algorithms, we used the instruction set manual [17] for estimation. The total computational load is estimated by the sum of all functions, multiplied with 25 % for overhead costs.

C. Results

Table II and Table III list for each function of the Bluetooth transmitter and receiver, the number of required operations³ (multiplications, additions, etc.) and how much cycles this function needs on a Pentium 4 and TI C64x DSP. Especially the GFSK modulation, conversion to fixed point numbers of the Bluetooth transmitter and FM-to-AM conversion of the receiver require most cycles. In the GFSK modulation function a 60-tap Gaussian filter is used that requires 1000 million additions plus multiplications per second. In our implementation we replaced this filter by lookup tables as the output value of the filter depends on the

³These values are derived by looking at the used algorithms in each part of the transceiver and is not determined by the C-code.

last 4 BPSK symbols. This optimization reduces the amount of computations significantly. Furthermore the estimated number of cycles on a TI C64x DSP is almost a factor 10 smaller than the measured number of cycles on the Pentium 4. This could have several reasons, the first one is that a DSP is designed for digital signal processing algorithms for embedded low-power systems and a Pentium 4 not. Furthermore several steps in our transceiver code operate on large buffers, which reduce the performance gain of the CPU cache.

Table IV and Table V show for HiperLAN/2, the number of required operations and cycles for each function of the transmitter and receiver.

Computational intensive parts are the conversion to floating-point precision, FFT and 64-QAM de-mapping in the receiver and conversion to fixed-point numbers in the transmitter. Although more bits are transmitted by the HiperLAN/2 transmitter, it requires less computational power than the Bluetooth transmitter. The HiperLAN/2 receiver requires on the other hand more cycles per second than the Bluetooth receiver, but the latter operates at a lower sample rate. Again, the estimated number of cycles on the TI C64x DSP is almost a factor 10 smaller than the measured number of cycles on the Pentium 4.

1) *Experiments*: Baseband experiments have been performed with the setup of Figure 7. In both Bluetooth and HiperLAN/2 mode, successful transmission and reception of continuously transmitted MAC bursts is achieved.

D. Power consumption

This section assesses the power consumption for the implementation alternatives. All alternatives share a common part which is the interface to the analog domain. This common part consists of 2 ADCs, 2 DACs and filters which is estimated to consume about 0.5 W⁴

Alternative 1, the testbed, and alternative 2, integration in the chipset, will have similar power consumption because they both use the Pentium-4 for the demodulation functions. Typical power consumption for a Pentium 4 is about 35 W at 2.4 GHz clock frequency [19]. So the power consumption will vary between $\frac{437}{2400} * 35 + 0.5 \approx 7$ W (Bluetooth receiver program) to $\frac{1225}{2400} * 35 + 0.5 \approx 19$ W (HiperLAN/2 receiver program). This power consumption can be reduced significantly if a low-power CPU, for example the Pentium-M [14] is used. Typical power

consumption of a Pentium-M at 1.2 GHz is 12 Watt. Because the architecture of this processor is much more efficient than a normal Pentium 4 CPU, it is not possible to use the values of Tables II, III, IV and V for this processor.

Alternative 3 uses a low power DSP instead for the demodulation algorithms. Typical power consumption for a TI C64x DSP running at 500 MHz is 1.0 W [15]). In this case the power consumption varies between $\frac{54}{500} * 1.0 + 0.5 \approx 0.6$ W (Bluetooth receiver program) to $\frac{203}{500} * 1.0 + 0.5 \approx 1$ W (HiperLAN/2 receiver program). This alternative is more than a factor 10 energy efficient, but requires additional hardware which increases manufacturing costs⁵.

VI. CONCLUSIONS

This paper describes a software-defined radio testbed for wireless LAN standards. The physical layer of the HiperLAN/2 and Bluetooth standard has been implemented in software running real-time on a normal PC and baseband experiments have verified the system both functionally and with respect to real-time continuous transmission and reception. Our testbed can easily be extended to other standards, because the only limitation in our testbed is the maximal channel bandwidth of 20 MHz and of course the processing capabilities of the used PC.

In addition, this paper evaluates implementation alternatives for the digital part of our testbed:

- 1) the testbed: a PCI card equipped with analog front-end functionality and all demodulation functions are performed by the Pentium 4 CPU.
- 2) integration of analog front-end functionality in the chipset of the motherboard.
- 3) a low power DSP plus analog front-end functionality mounted on a PCI card

The first alternative requires no extra DSP hardware which reduces manufacturing costs. However the PCI interface is fully utilized, disabling other applications. Therefore we proposed an second alternative where the analog front-end is integrated in the chipset of the notebook which has no load on the PCI interface. This solution is due to the integration more cost effective. The last implementation alternative is a PCI board equipped with a low power DSP (a TI C64x DSP) and analog front-end.

The last alternative has a maximal power consumption of 1 W whereas the first 2 solutions consume almost 19 W. So the DSP solution is almost 20 times more

⁴We have assessed that 12-bit 40 MSPS ADCs and 10-bit 20 MSPS DACs can be used. Current ADCs, for example the dual AD9238 of Analog Devices consumes about 400 mW (200 mW per channel) [18] and current DACs such as the AD9740 DAC consumes about 35 mW per channel [18].

⁵The price of a TI C64x DSP varies from \$ 25 to \$ 150 [15].

energy efficient than a Pentium 4 based implementation. This has several reasons, the first one is that a DSP is designed for digital signal processing algorithms used in embedded low-power applications and a Pentium 4 is (only) optimized for performance. Code optimizations and the use of a processor designed for low-power applications such as the Pentium-M CPU can reduce the power consumption significantly. The architecture of the Pentium-M is compared with the Pentium-4 CPU very different. Therefore, additional experiments have to be carried out in order to measure the load of our transceiver program on this low-power CPU.

ACKNOWLEDGMENT

We thank our colleagues from the IC-Design group for their work on the analog part of the front-end and for interesting discussions. Moreover we thank Henny Kuipers and Geert-Jan Laanstra for building the DAC and ADC board.

This research is supported by the PROgram for Research on Embedded Systems & Software (PROGRESS) of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the technology foundation STW.

REFERENCES

- [1] J. Mitola. *Software Radio Architecture: Object-Oriented Approaches to Wireless Systems Engineering*. Wiley, 2000.
- [2] R.H. Walden. *Performance trends for analog-to-digital converters*. IEEE Communications Magazine, pages 96–101, February 1999.
- [3] V.J. Arkesteijn, E.A.M. Klumperink, and B. Nauta. *An Analogue Front-End Architecture for Software Defined Radio*. 13th proRISC workshop, November 2002.
- [4] *The Bluetooth-HiperLAN/2 SDR receiver project website*. <http://www.sas.el.utwente.nl/home/SDR/>.
- [5] W. Tuttlebee. *Software Defined Radio: Origins, Drivers and International Perspectives*. John Wiley & Sons, 2002.
- [6] K. Masselos, S. Blionas, and T. Rautio. *Reconfigurability requirements of wireless communication systems*. IEEE heterogeneous reconfigurable systems-on-chip workshop, 2004.
- [7] ETSI. *Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer*. Technical Specification ETSI TS 101 475 V1.2.2 (2001-02), ETSI, February 2001.
- [8] BluetoothSIG. *Specification of the Bluetooth System - Core*. Technical Specification Version 1.1, 'Bluetooth SIG', February 2001.
- [9] V.J. Arkesteijn, R. Schiphorst, F.W. Hoeksema, E.A.M. Klumperink, B. Nauta, and C.H. Slump. *A combined receiver front-end for Bluetooth and HiperLAN/2*. 4nd PROGRESS workshop on Embedded Systems and Software, 2003.
- [10] I. Ripoll, P. Pisa, L. Abeni, P. Gai, A. Lanusse, S. Saez, and B. Privat. *Deliverable D1.1 - RTOS Analysis*. <http://www.mnis.fr/opensource/ocera/rtos/>, 2002.
- [11] R. Schiphorst, F.W. Hoeksema, and C.H. Slump. *A Bluetooth-enabled HiperLAN/2 receiver*. Proceedings of the VTC Fall 2003, October 2003.
- [12] R. Schiphorst, F.W. Hoeksema, and C.H. Slump. *A (simplified) Bluetooth Maximum A posteriori Probability (MAP) receiver*. Proceedings of IEEE SPAWC2003, June 2003.
- [13] <http://www.fftw.org/>.
- [14] <http://www.intel.com>.
- [15] <http://www.ti.com/>.
- [16] J. Edward (Blue Wave Systems). *Performance Evaluation of the TMS320C64x Architecture and C Compiler*. <http://mcg.motorola.com/us/general/bwswp.pdf>, 2000.
- [17] Texas Instruments. *TMS320C6000 CPU and Reference Guide (SPRU189F)*, 2000.
- [18] <http://www.analog.com/>.
- [19] <http://www.sandpile.org/>.
- [20] <http://www.semiconductors.philips.com/comms>.

Standard	Frequency band	Modulation type
CT2	864/868 MHz	GFSK
CT2+	944/948 MHz	GFSK
DECT	1880-1900 MHz	GFSK
PHS	1893-1920 MHz	DQPSK
IEEE 802.15.4	2402 - 2480 MHz (N. America) 2412 - 2472 MHz (Europe) 2483 MHz (Japan)	GFSK
Bluetooth	2402 - 2480 MHz (N. America & Europe) 2447 - 2473 MHz (Spain) 2448 - 2482 MHz (France) 2473 - 2495 MHz (Japan)	GFSK
HomeRF	2402 - 2480 MHz (N. America & Europe) 2447 - 2473 MHz (Spain) 2448 - 2482 MHz (France) 2473 - 2495 MHz (Japan)	GFSK
IEEE 802.11a	5150 - 5250 MHz (USA) 5250 - 5350 MHz (USA) 5725 - 5825 MHz (USA)	OFDM: 2/4/16/64 QAM
IEEE 802.11b	2410 - 2462 MHz (N. America) 2412 - 2472 MHz (Europe) 2483 MHz (Japan)	GFSK/DBPSK/DQPSK/QPSK
HiperLAN/2	5150 - 5250 MHz (USA) 5250 - 5350 MHz (USA) 5725 - 5825 MHz (USA) 5150 - 5350 MHz (Europe) 5470 - 5725 MHz (Europe) 5725 - 5875 MHz (Europe) 5150 - 5250 MHz (Japan)	OFDM: 2/4/16/64 QAM
IS-54/IS-136	824 - 849 MHz 869 - 894 MHz 1850 - 1910 MHz 1930 - 1990 MHz	CDMA: $\pi/4$ DQPSK
IS-95	824 - 849 MHz 869 - 894 MHz 1850 - 1910 MHz 1930 - 1990 MHz	CDMA: QPSK/OQPSK
IMT-2000/UMTS	1920 - 1980 MHz (Asia only) 2110 - 2170 MHz (Asia only) 1920 - 1980 MHz 2110 - 2170 MHz 1900 - 1920 MHz 2010 - 2025 MHz	CDMA: QPSK
GSM	824 - 849 MHz 869 - 894 MHz 880 - 915 MHz 925 - 960 MHz 1710 - 1785 MHz 1805 - 1880 MHz 1850 - 1910 MHz 1930 - 1990 MHz	GMSK
PDC	810 - 826 MHz 940 - 956 MHz 1429 - 1441 MHz 1453 - 1465 MHz 1477 - 1489 MHz 1501 - 1513 MHz	$\pi/4$ DQPSK

TABLE I
OVERVIEW OF WIRELESS STANDARDS

Bluetooth TX function	M operations/s	M cycles/s [Pentium 4]	M cycles/s [TI C64x DSP]
MAC burst generation	1	1	1
GFSK modulation	2100	440	50
float-to-int conversion	80	320	-
total	2181 (analyzed)	714 (measured)	64 (estimated)

TABLE II
COMPUTATIONAL LOAD OF BLUETOOTH TRANSMITTER FUNCTIONS

Bluetooth RX function	M operations/s	M cycles/s [Pentium 4]	M cycles/s [TI C64x DSP]
int-to-float conversion	20	88	-
FM-to-AM conversion	75	261	40
synchronization	1	27	2
freq. offset corr.	1	not implemented	not implemented
bit decision	3	33	1
total	100 (analyzed)	437 (measured)	54 (estimated)

TABLE III
COMPUTATIONAL LOAD OF BLUETOOTH RECEIVER FUNCTIONS

HiperLAN/2 TX function	M operations/s	M cycles/s [Pentium 4]	M cycles/s [TI C64x DSP]
QAM mapping	38	85	15
iFFT	230	128	32
float-to-int conversion	80	215	-
total	348 (analyzed)	500 (measured)	59 (estimated)

TABLE IV
COMPUTATIONAL LOAD OF HIPERLAN/2 TRANSMITTER FUNCTIONS MAPPED FOR 64-QAM MODE

HiperLAN/2 RX function	M operations/s	M cycles/s [Pentium 4]	M cycles/s [TI C64x DSP]
int-to-float conversion	80	351	-
synchronization and parameter estimation	4	60	8
freq. offset corr.	39	120	21
FFT	230	128	32
channel equalization	39	79	21
phase offset det. and corr.	40	127	32
64-QAM de-mapping	77	204	48
total	509 (analyzed)	1225 (measured)	203 (estimated)

TABLE V
COMPUTATIONAL LOAD OF HIPERLAN/2 RECEIVER FUNCTIONS MAPPED FOR 64-QAM MODE

LIST OF FIGURES

1	Standards support by products in time	2
2	Mapping of protocols in current designs on hardware/software	2
3	Project scope	2
4	Functional architecture of the Bluetooth transmitter and receiver	3
5	Functional architecture of the HiperLAN/2 transmitter and receiver	4
6	Functional architecture of the Bluetooth-enabled HiperLAN/2 receiver	4
7	Component architecture of the SDR testbed (baseband only)	4
8	Implementation alternatives	4