Pers Ubiquit Comput (2003) 7: 299–308
DOI 10.1007/s00779-003-0243-x

ORIGINAL ARTICLE

Christian Plessl · Rolf Enzler · Herbert Walder
Jan Beutel · Marco Platzner · Lothar Thiele
Gerhard Tröster

The case for reconfigurable hardware in wearable computing

Received: 1 February 2003 / Accepted: 2 April 2003 / Published online: 11 September 2003

Abstract Wearable computers are embedded into the mobile environment of their users. A design challenge for wearable systems is to combine the high performance required for tasks such as video decoding with the low energy consumption required to maximise battery runtimes and the flexibility demanded by the dynamics of the environment and the applications. In this paper, we demonstrate that reconfigurable hardware technology is able to answer this challenge. We present the concept and the prototype implementation of an autonomous wearable unit with reconfigurable modules (WURM). We discuss experiments that show the uses of reconfigurable hardware in WURM: ASICs-on-demand and adaptive interfaces. Finally, we present an experiment with an operating system layer for WURM.

Keywords Body area computing system · Embedded systems · Field-programmable gate arrays · Reconfigurable hardware · Wearable computing

1 Introduction

Many current wearable systems are built out of standard components such as personal digital assistants (PDAs) or sub-notebooks. These more or less miniaturised general-purpose computers are very attractive due to their easy availability, low price and matured development tools, including compilers and operating systems. However, they are not optimised for wearable usage. Keyboard or pen-based data entry and LCD display contradict the unobtrusive, hands-free operation paradigm of wearable computing. Moreover, distributed on-body computing systems built out of PDAs and sub-notebooks are highly inefficient due to the lack of specialisation of the single components.

In this paper, we argue that future wearable computing systems should be viewed and designed as *embedded systems*. Due to the movement of its user, a wearable computer is embedded into a mobile environment and needs to interact with this environment. We denote such a computing system as a *body area computing system*. Figure 1 sketches a body area computing system that is composed of a set of distributed nodes and a communication network centred around one general-purpose main module.

Sensors and actors are distributed over the human body. The current efforts in integrating electronics and wires into clothing, shoes and other appliances promote this distribution. For example, Van Laerhoven et al. [1] and Kern et al. [2] attach acceleration sensors to the body to discriminate between user actions. These actions are then used to extract high-level context information. Kymissis et al. [3] present piezo-electric generators integrated into shoes, powering a wearable subsystem when the user is walking. The spatial distribution of sensors and actors implies that computation is also bound to specific locations on the body. Since communication—in particular wireless communication—is rather costly in terms of energy consumption, it is advantageous to reduce the communication demand by moving sensor data preprocessing and actor control very close to the sensors and actors, respectively. For instance, the filtering and compression of sampled audio data is best done directly at the microphone node.

The communication network is a mixture of wireless and wired connections. Presently, wireless is the predominant technology. In the future, conductive textiles might increase the trend toward wired connections for nodes in the same piece of textile, as has been described by Kirstein et al. [4] and Park et al. [5]. External net-

C. Plessl (✉) · H. Walder · J. Beutel · M. Platzner · L. Thiele
ETH Zentrum Office ETZ-G81,
Computer Engineering and Networks Lab (TIK),
Gloriastraße 35, 8092 , Zurich, Switzerland
E-mail: plessl@tik.ee.ethz.ch

R. Enzler · G. Tröster
Electronics Laboratory, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

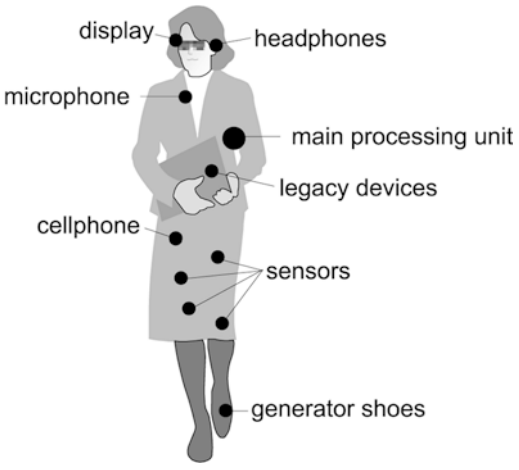

Fig. 1 The body area computing system

works and pervasive computing facilities are also accessed by wireless technologies, such as cellphones or wireless LAN.

As body area computing systems are distributed embedded systems, they are subject to several design constraints, described as follows.

Multi-mode performance: wearable systems are multi-mode systems. They require a fixed baseline performance for running control tasks that neither show high computational demands nor stringent timing constraints. Occasionally, wearable systems execute bursts of computation-intensive tasks that carry real-time constraints [6]. An example for a task that can be served best effort is reading position sensors to update the geographical context. Examples of real-time tasks are audio and video coding. Wearable computing systems must have sufficient performance to satisfy the real-time constraints. Missed deadlines render the system useless.

Energy awareness: the energy awareness of the body area computing system is essential since we want the wearable system to be on and functional the whole day. Designing energy aware wearable systems leads to several challenges, including energy-efficient computation and dynamic power management. *Energy-efficient computation* means that a task has to be run such that the amount of dissipated energy is minimised while the system retains its functionality. It is important to clearly distinguish between low power consumption and energy efficiency. The power consumption is the measured energy dissipation divided by the measurement period. In data sheets, the power consumption is often used like a device parameter without any detailed reference to the specific instructions or tasks executed. In contrast, the energy efficiency relates to the total energy the device needs to compute a specific task. The challenge is to design devices with high energy efficiency for the set of tasks given. For example, it is quite possible that the total dissipated energy for a given task is smaller on a device with a high power consumption which executes quickly, than on a slow device with a lower power consumption. *Dynamic power management* comprises techniques that assign tasks to the most energy-efficient devices available and that force other unused system components into their power-down modes or even shut them off when appropriate. Many applications of wearable computing systems extract and use high-level context information. This facilitates the implementation of advanced dynamic power management techniques that make use of the current context and even predictions of future user actions.

High flexibility: the body area computing system has to handle highly dynamic situations. Firstly, the application requirements vary strongly with the user's choices, but also with the context and the location. Secondly, as clothes are put on and off, components are dynamically added and removed from the body area computing system. On a longer time scale, deployed wearable systems have to adapt to emerging or changing communication standards and protocols.

Further design criteria includes reliability, availability and form factors such as volume and weight. As wearable systems will eventually become consumer devices, cost and reduced time-to-market will also become important design goals.

In summary, we can state that the flexibility requirements in wearable computing demand a programmable general-purpose computing system, while the high performance and energy awareness requirements demand a specialised computing system. To answer this design challenge, we propose to incorporate *reconfigurable hardware* in the computing nodes of the body area computer system. In this paper, we present wearable nodes comprised of small-to medium-range CPUs and reconfigurable modules. The reconfigurable hardware in these nodes achieves a higher performance and is more energy-efficient than the CPU for computation-intensive real-time tasks.

We first discuss the advantages of reconfigurable hardware in body area computing systems, and then introduce the concept of a wearable unit with reconfigurable modules (WURM). Later, we present a WURM prototype implementation and report on several experiments with the prototype, and then outline future work.

2 The advantages of reconfigurable hardware in wearable computing

In this section, we first review the main characteristics of reconfigurable hardware and compare them to processors and dedicated hardware. Then, we advocate the use of reconfigurable hardware in wearable computing nodes and discuss related work.

2.1 Reconfigurable hardware

The predominant reconfigurable hardware device today is the field-programmable gate array (FPGA) [7].

FPGAs were introduced to the market at the high-end of programmable logic devices (PLDs) in the mid 1980s. FPGAs consist of an array of logic blocks, routing channels to interconnect the logic blocks and surrounding input/output (I/O) blocks. SRAM-based FPGAs use SRAM cells to control the functionality of logic and I/O blocks as well as routing, and can be reprogrammed in-circuit arbitrarily often by downloading a stream of configuration data to the device.

While early FPGA generations were quite limited in their capacities, today's devices feature millions of gates of programmable logic and, additionally, dedicated hardware blocks such as fast embedded memories and fixed-point multipliers. To interface to external components, FPGAs are compliant to a number of high speed I/O standards and can directly interface to DDR RAM, or to the PCI or AGP bus. Current FPGAs have sufficient resources to implement rather complex circuits such as cryptography algorithms, image and video processing functions, networking interfaces and complete CPU cores.

A currently popular trend is to combine FPGAs with CPUs to form hybrid computing systems, or so-called configurable systems on a chip (CSoC). The CPUs in CSoCs can be implemented in two different ways: as soft cores or as hard cores. A soft CPU core is synthesised and mapped to the FPGA's logic resources. A hard CPU core is a dedicated piece of hardware integrated into the FPGA fabric. While a hard CPU core delivers maximum performance, a soft CPU core allows for easy architectural extension. Examples of commercially available hybrid systems that integrate FPGAs with hard CPU cores are Triscend's 8051-based E5 [8] and Xilinx's PowerPC-based Virtex-II Pro [9]. The tight integration of CPU and FPGA allows for low-latency and low-power communication.

SRAM-based FPGAs are configured either statically or dynamically. In the case of static configuration, the FPGA loads the configuration data from an external non-volatile memory at system startup. The configuration does not change during the system's runtime. In the case of dynamic configuration, an external host processor writes the configuration data to the FPGA. This allows to change the FPGA configuration on demand. The configuration data size of a current high-end FPGA is about 1–4 MB, which results in configuration times of about 12.5–50 ms. Some FPGAs offer an advanced reconfiguration mode, partial reconfiguration, which allows the configuration of parts of the device at runtime. Partial reconfiguration enables true multi-tasking and reduces the configuration times for single tasks substantially.

2.2 The efficiency of reconfigurable hardware

With respect to performance, power consumption and flexibility, reconfigurable hardware is positioned between processors and dedicated hardware (an application specific integrated circuit, or ASIC). For a given function, dedicated hardware achieves the highest performance and the lowest power consumption due to its high degree of specialisation. However, dedicated hardware is also inflexible. For a general wearable computing node, programmable solutions such as processors or reconfigurable hardware are required. Compared to reconfigurable hardware, processors pay area and runtime penalties for instruction storage and handling, i.e., fetching and decoding. Advances in semiconductor technology will allow future generations of processors to become more widespread in high-performance and low-power applications. However, since reconfigurable devices also benefit from advances in process technology, their advantages in terms of performance and power will persist.

Several case studies have shown that FPGAs achieve higher throughput and are more energy-efficient than processors, provided that the application matches well the spatial structures of FPGAs and reveals a sufficient amount of parallelism.

Mencer et al. [10] compared different implementations of the IDEA cryptography algorithm on RISC and DSP processors and on an FPGA. Abnous et al. [11] performed similar studies on finite and infinite impulse response filters (FIR, IIR). Table 1 summarises the results and gives the throughput and energy efficiency for each application. For IDEA, the throughput is measured in millions of encrypted bits per second, and for FIR and IIR the throughput is measured in millions of filter taps per second. As Table 1 shows, the FPGAs achieved the highest performance. The energy efficiency relates the performance to the power consumption. For all applications, the FPGAs achieved a better energy efficiency than the embedded RISC processor. The DSPs outperformed the FPGAs in energy efficiency for FIR and IIR, because these filters perfectly match the DSP architectures.

Stitt et al. [12] studied the energy efficiency of hybrid CPUs for embedded systems and evaluated a set of benchmarks that are relevant to wearable computing.

Table 1 A comparison of throughput and energy efficiency between RISC, DSP and FPGA for the IDEA cryptography algorithm, FIR and IIR digital filters

Type	Device	IDEA [10]		FIR [11]		IIR [11]	
RISC	StrongARM SA-110	32.0	32.0	9.9	26.7	1.5	3.6
DSP	TI TMS320C6x	53.1	8.9	–	–	–	–
	TI TMS320C2xx	–	–	20.0	769.2	1.0	52.4
FPGA	Xilinx XC4020XL	528.0	167.6	–	–	–	–
	Xilinx XC4003A	–	–	30.0	454.5	2.1	9.7

On the Triscend E5 device, they measured an energy savings of 71% on average by moving application kernels to the FPGA instead of running the applications solely on the CPU.

2.3 Reconfigurable hardware in wearables

We see two main uses of reconfigurable hardware in a wearable computing node: ASIC-on-demand and adaptive interfaces.

2.3.1 ASIC-on-demand

Many computational-intensive functions can be more efficiently executed in reconfigurable hardware than on a processor. We denote such functions, which are loaded into reconfigurable hardware as needed, as ASICs-on-demand. Small wearable nodes with attached sensors, especially, benefit greatly from ASICs-on-demand. Such nodes often require computational-intensive preprocessing to reduce data rates. This is motivated by the fact that wireless communication is more expensive than computation in terms of energy [13].

Figure 2 outlines a typical scenario for a wearable sensor node. The node is attached to a microphone and preprocesses the raw audio data in different ways. In a voice recording application, high-quality audio samples are recorded, compressed and transmitted to the recorder's database, presumably located on the main module. In a feature extraction application, we are interested in deriving information useful for the context engine rather than in high audio quality. For example, [14] describes a feature extractor that is based on a dozen features and discriminates between five types of TV programs. The feature extraction and analysis is done by means of a neural network. Both audio compression and feature extraction are computationally demanding and would require a high-performance CPU. It is more energy-efficient to use a small-range CPU for control and communication and to execute the preprocessing in reconfigurable hardware.

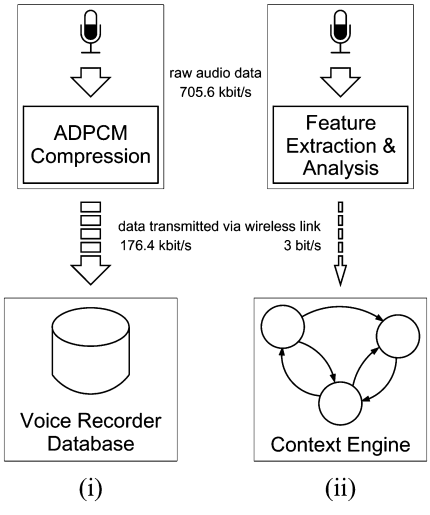

Fig. 2 Two audio applications running on the same wearable node. **a** Voice recorder **b** Context recognition

2.3.2 Adaptive interfaces

For economic reasons, we are interested in reusing the same wearable node for many applications. This is facilitated by adaptive interfaces. A node can be equipped with a rich set of generic interface pins. The reconfigurable hardware connects these pins with the processing modules on demand.

Again, nodes attached to sensors are the prime example. Sensors come with a multitude of different interfaces, ranging from simple two-port analog and digital interfaces to more complex interfaces, e.g., SPI or I2C bus. Often, data from several sensors is fused to derive high-level context information. The numbers and types of sensors required varies with the application and the position on the human body. In one scenario, a leg-bound sensor node might process the signals of dozens of acceleration sensors for accurately determining the leg's movement [15]. In another scenario, a chest-bound node that discriminates between indoor and outdoor might have only one or two light sensors attached to it.

The two uses of reconfigurable hardware in wearable computing are combined when reconfigurable hardware computes interface conversions and protocol stacks to relieve the CPU from such tasks.

2.4 Related work

Scalera et al. [16] suggest using reconfigurable hardware for sensor data preprocessing and present the CAμS (common architecture for microsensors) platform. Acoustic signal detection is discussed as an example application. Significant improvements are reported in terms of system size, power consumption and weight, compared to a CPU based predecessor system. In contrast to our work, the reconfigurable devices are mainly used as an alternative to ASICs; the dynamic *re*configuration capabilities are not exploited. The system functionality is configured once and then remains unchanged. We consider the loading of different tasks on-demand at runtime.

The use of reconfigurable technology for mobile, networked appliances was investigated by Mignolet et al. [17]. Their demonstration system uses an FPGA board with two Virtex-II FPGAs attached to the expansion slot of a Compaq iPaq PDA. The hardware and software resources are managed by a modified version of the real-time Linux operating system running on the PDA. A middleware layer overlays the operating system and schedules tasks according to quality-of-service parameters. This work shares many similarities with our approach. However, we stress small-scale embedded

nodes and the distributed nature of wearable computing systems.

Yau and Karim [18] proposed a reconfigurable middleware layer for autonomous decentralised systems, called reconfigurable context-sensitive middleware (RCSM). The goal of RCSM is to facilitate applications that require context-awareness and/or spontaneous and ad-hoc communication between heterogeneous mobile nodes. The computational intensive parts of RCSM are implemented in FPGAs in order to achieve the required performance. In contrast to RCSM, our approach does not restrict reconfigurable hardware to the middleware layer.

3 WURM

This section presents the WURM architectural concept for a basic node of the body area computing system and its envisioned integration into a body area computing system.

3.1 The WURM hardware architecture

Figure 3 shows the WURM hardware architecture. A WURM node consists of a CPU, a reconfigurable hardware unit, memory, a set of I/O interfaces that connect to sensors and actors and a wireless interface for communication with other nodes.

The CPU handles control tasks and tasks that need low to medium processing power. The reconfigurable hardware unit executes tasks with high computation demands, but can also run communication protocol functions to relieve the CPU. The reconfigurable hardware is further used for interfacing to external sensors and actors. Both the CPU and the reconfigurable hardware unit have power save modes.

WURM is a concept and different wearable nodes can have different realisations of the WURM architecture. Nodes can differ in their CPUs and in the capacity of the reconfigurable hardware. An audio preprocessing node, for example, uses only one sensor interface to connect to the microphone and would employ a mid-range CPU and reconfigurable hardware. A WURM node for leg movement analysis will interface to many acceleration sensors, but a low-end CPU plus a smaller amount of reconfigurable hardware resources might be sufficient.

3.2 The WURM software architecture

Figure 4 shows the WURM software architecture. In this paper, we concentrate on the WURM operating system (WURM-OS) layer. WURM-OS manages the available hardware resources of the node, i.e., the CPU, the reconfigurable hardware unit, the memory and the I/O. The main service WURM-OS performs at higher system layers is the execution of software and hardware tasks. Software tasks run on the node's CPU, hardware tasks execute on the reconfigurable hardware unit.

WURM-OS is based on a standard real-time kernel for microprocessors, but extends the system services to the reconfigurable hardware. The development of operating system layers for reconfigurable hardware is quite a new topic and involves several challenges. While the CPU executes tasks quasi-parallel by pre-emption, the reconfigurable hardware allows for concurrent task execution. Basically, WURM-OS performs task and resource management for the reconfigurable hardware unit which includes:

- loading, executing and removing tasks
- establishing interfaces between the tasks and the I/O
- scheduling and placing of tasks

A central concept in the WURM architecture is the hardware task. A hardware task is described by several structural and timing characteristics which WURM-OS uses for scheduling and placement decisions. The main structural characteristics are *size* and *shape*. The size gives the area requirement of a task in number of logic blocks. The task shapes can reasonably be assumed to be rectangles. A further structural characteristic is a *relocatability*. Relocatable hardware tasks can be placed anywhere in the logic block array. Non-relocatable tasks must be placed exactly on their predefined positions, e.g., to access special resources. The main timing characteristics of a hardware task are the number of *required cycles* and the *maximum clock frequency*. The number of required cycles might or might not be known in advance. Instead of a maximum clock rate, the *clock range* can be given. A task might require a clock rate in a certain

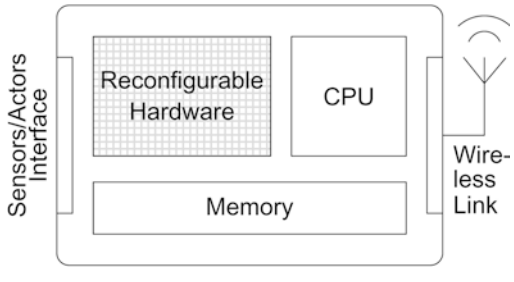

Fig. 3 The WURM hardware architecture

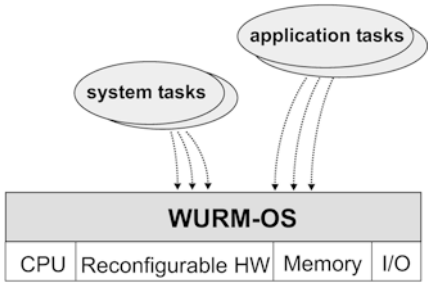

Fig. 4 The WURM software architecture

interval to preserve timing requirements of the I/O devices or the memory.

3.3 The integration into the body area computing system

As outlined in an earlier section, we consider a wearable computer as a distributed, heterogeneous, body-area computing system. Most of the nodes of such a computing system will be WURM nodes, whereas some nodes might be of the sub-notebook type or simply plain sensor/actor nodes with fixed functionality. We propose a distributed operating system layer for such a system, that provides a global view of the computation and communication resources and is aware of the current high-level context as well as the user requests. An important function of this system layer is to run applications on the different nodes. For this, the status of each node must be monitored and hardware and software tasks must be sent to the individual nodes.

As Figure 4 shows, we envision system and application layers on top of the WURM-OS. The system layer in this figure forms a part of the aforementioned distributed operating system. While these higher layers of system software involve a number of challenging research issues on their own, we focus on the lowest software layer (WURM-OS) in this work.

4 The WURM prototype and experiments

We are implementing a WURM prototype to show that reconfigurable hardware enables the construction of small embedded wearable nodes that can deliver the required performance at a high energy efficiency in a flexible way. The WURM prototype is a functional demonstrator and still a work in progress. For deployment in real-world wearable computers, the system can be significantly reduced in size.

In the following sections, we present our prototyping platform and a number of experiments toward ASICs-on-demand and adaptive interfaces. Finally, we present a prototype of WURM-OS for reconfigurable devices.

4.1 The WURM prototyping platform

Figure 5 shows the block diagram of the WURM prototyping platform. The platform is based on the XESS XSV800 board, which contains a Xilinx Virtex XCV800–4 FPGA and a multitude of I/O facilities. To connect sensors and actors, the XESS board offers a rich set of general-purpose I/O (GPIO), serial and parallel interfaces, a stereo audio codec and a video interface. Further, the board provides physical drivers for Ethernet, PS/2 and USB.

Instead of a dedicated processor, we use a soft CPU core that is configured into the FPGA. This simplifies the prototype and gives us the flexibility to experiment

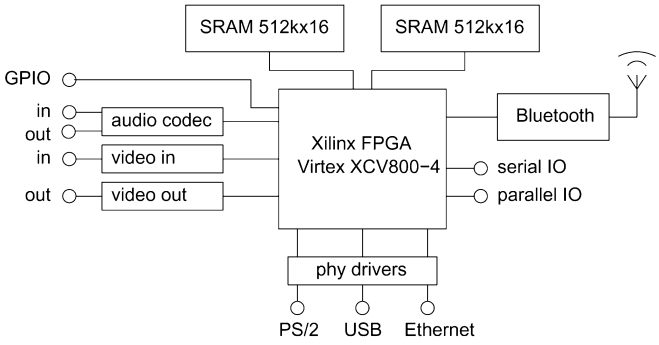

Fig. 5 A block diagram of the WURM prototyping platform

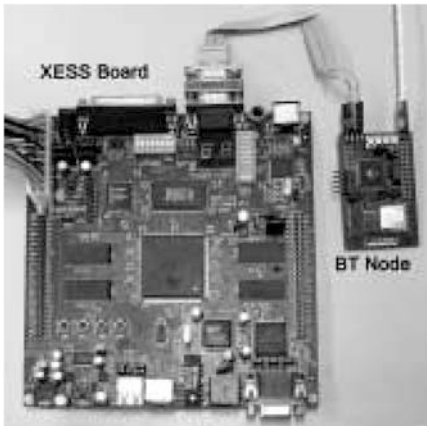

Fig. 6 The WURM prototyping platform

with different ways of coupling the CPU with the reconfigurable modules. We can select from a broad range of soft CPUs, from 8-bit microcontrollers [19] to 32-bit RISCs running at 125 MHz [20]. While the CPU is configured into the FPGA at power-on, the hardware tasks are dynamically configured on demand. In our current prototype, the configuration is controlled by a host system via the parallel interface. In the future, we plan to switch to one of the hybrid devices that integrate a dedicated CPU and a reconfigurable array in a system-on-chip, e.g., Xilinx Virtex-II Pro [9].

For wireless communication between the WURM nodes, we have developed the Bluetooth module BTnode [21]. The BTnode runs firmware that processes the lower levels of the Bluetooth protocol stack autonomously and thus relieves the CPU and the reconfigurable hardware unit from implementing this protocol. Figure 6 shows a picture of the WURM prototyping platform.

4.2 ASIC-on-demand

The computationally demanding applications in wearable computing typically come from the domains of multimedia, cryptography and communication. We have analysed a set of 29 benchmark programs (MCCmix) from these domains with a cycle-accurate CPU simulator

and have derived instruction class mix and execution statistics. The instruction class mix characterises the programs according to the frequency of different instruction types during execution. We have classified the instructions according to their functionality into eight instruction classes. Figure 7 illustrates the results for the two instruction classes ''integer arithmetic'' and ''logic''. The graphics show the percentages of the instructions with respect to the total instruction count. The application domains multimedia (MM), cryptography (CR) and communication (CO) are highlighted and the averages for these domains are drawn. The programs from the three domains show quite different characteristics:

– The multimedia applications use many integer arithmetic operations but few logic operations.
– The cryptography applications use many logic operations and few branches.
– The communication applications use many branches and few shift operations.

The different characteristics demonstrate that having a computing element that flexibly adapts to the characteristic of the running program is beneficial.

In order to identify the portions of code that should go into the reconfigurable hardware, we have determined the kernels of the programs. Figure 8 shows the relative runtimes of the two most runtime-intensive functions for each application. It is notable that on average the MCCmix applications spend 59% of their runtime in the most runtime-intensive function, as opposed to only 19% for the SPECint95 benchmarks. The detailed results of our analysis have been published in [22].

For a detailed case study [23], we have chosen adaptive differential pulse code modulation (ADPCM) from our benchmark set. Decoding streams of audio and video data is an important task required in many wearable systems. Decoding, i.e., decompressing, such streams is computationally expensive. Moreover, there are many different audio and video formats and compression algorithms in use. This combination of high performance requirements and flexibility requires ASICs-on-demand.

In the case study, we have designed a minimal embedded WURM based on a general-purpose CPU core, memories, a network interface and two ASICs-on-demand, or coprocessors, for the playback of compressed audio streams. Audio streams encoded in different formats are sent as UDP packets via Ethernet to the WURM node. The Ethernet media access controller (MAC), implemented as a hardware task, receives and buffers the packets. The UDP protocol is executed on the CPU as a software task. The CPU recognises the used encoding format and initiates the dynamic configuration of the suitable audio decoding coprocessor into the reconfigurable hardware unit. When the audio coprocessor is in place, it receives the encoded audio stream from the CPU. The audio

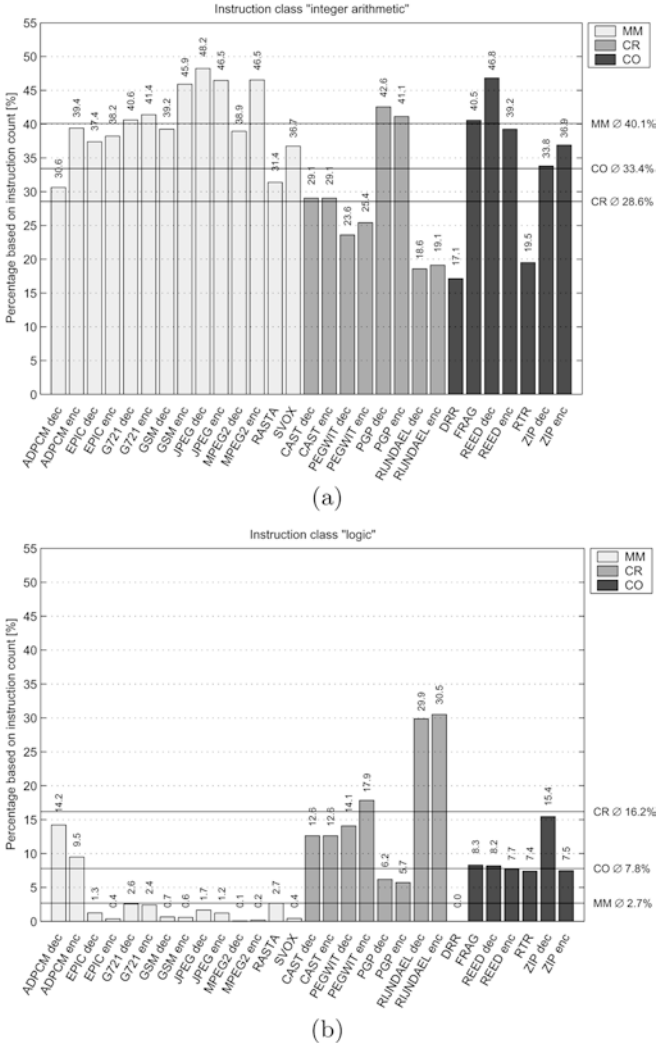

Fig. 7 Percentages of the instruction classes. **a** ''Integer arithmetic'' **b** ''Logic'' with respect to the total instruction count. The application domains multimedia (MM), cryptography (CR), and communication (CO) are highlighted

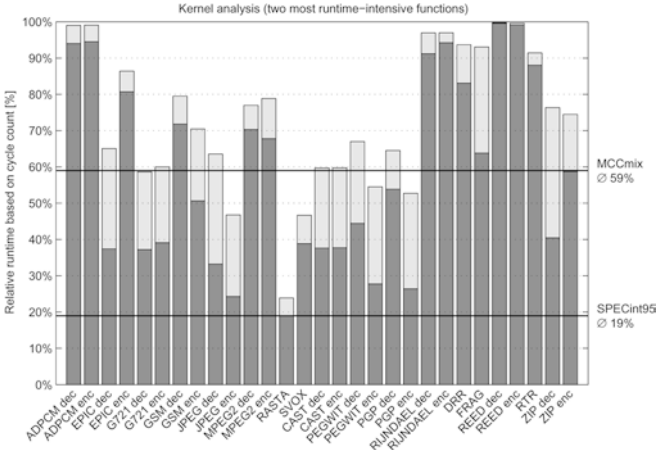

Fig. 8 Kernel identification for wearable applications

coprocessor decodes the audio stream and sends the raw audio data for playback to the on-board digital-to-analog converter. The CPU and coprocessor characteristics are described as follows.

The soft CPU core used for this experiment is the SPARC V8 compatible 32-bit LEON CPU [24]. The CPU implements 2 KB of separated data and instruction caches, a 256 byte internal boot-ROM and uses a 32-bit external memory interface. The CPU core requires 3865 Virtex slices which amounts to 41% of the FPGA's logic resources, and 14 dedicated RAM blocks which equals 50% of the FPGA's memory resources. Without hand optimisation, the CPU runs at 25 MHz. The software tasks are executed under the RTEMS real-time operating system kernel running on the LEON CPU.

We have implemented two audio decoding coprocessors, a PCM decoder and an Intel/DVI compliant ADPCM decoder. The ADPCM core runs at 50 MHz and uses 430 Virtex slices, or 4.5% of the FPGA's resources. The PCM decoder fits into 35 slices, or 0.4% of the FPGA's resources. All circuit design was done in VHDL. Synopsys FPGA Express 3.6 and Xilinx Foundation 4.1i tools were used for synthesis.

The software implementation of the ADPCM decoder needs 70 CPU cycles per sample. At a 25 MHz clock frequency, this results in a decoding performance of 350 Ksamples/sec. The ADPCM hardware task decodes one sample in five cycles. At a clock frequency of 25 MHz, this results in a sustained decoding performance of 5 Msamples/sec. Hence, running the ADPCM decoder in reconfigurable hardware yields a speedup of 14.

4.3 Adaptive interfaces

We have implemented several communication interfaces as hardware tasks. These tasks differ strongly in their complexity, ranging from simple UART and synchronous serial interface (SSI) interfaces to an Ethernet media access controller. Table 2 lists the typical sizes of these tasks, measured in the number of Virtex slices. For our WURM prototype, these cores occupy 0.4–7.3% of the reconfigurable resources.

To demonstrate the feasibility of protocol stack processing in reconfigurable hardware, we have devised a minimal IP stack that handles the ARP, IP and UDP protocols autonomously without intervention of the CPU. Figure 9 shows the floor plan of the IP stack in the FPGA.

As Figure 9 indicates, the IP stack splits into several tasks. Each task implements a part of the protocol handling. Due to their stringent timing requirements, some tasks need to be permanently resident in the FPGA while others can be loaded by the WURM-OS on demand. For instance, the ''Ethernet receiver'' (Task 1 in Figure 9) is highly time-critical as an average IP packet of 500 bytes payload comes in within about 40 μs (100 Mbit/s Ethernet). With partial configuration times in the range of milliseconds, the task could not be loaded fast enough to capture the packet. In contrast, tasks handling higher protocol layers such as ICMP and ARP are not time-critical and can be loaded on demand.

Table 2 Area requirements for communication tasks

Hardware task	Area
UART [25]	100
SSI (PCM serialiser) [26]	38
Ethernet-MAC [25]	628
ARP (part of Ethernet-MAC) [25]	417
IP (Internet protocol) [25]	690
UDP (user datagram protocol) [25]	365

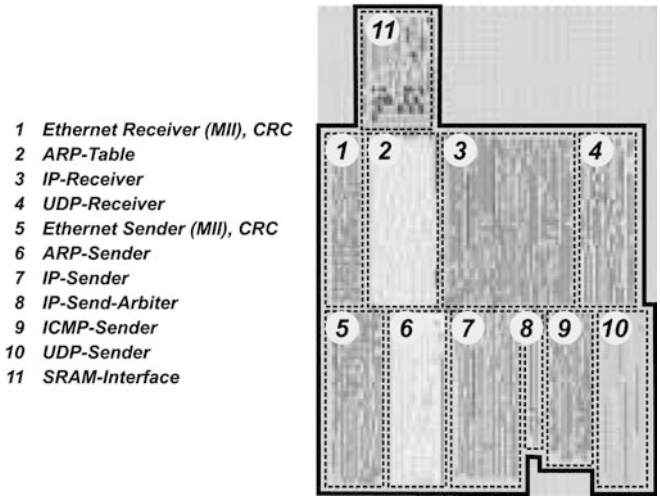

Fig. 9 The floorplan of the minimal IP stack

In order to test the various communication interfaces and protocol stacks, we have developed a WURM that implements a reconfigurable Bluetooth/Ethernet bridge [25]. The bridge provides an Internet access point for Bluetooth devices. The node executes a number of hardware tasks, among them a UART interface, an Ethernet MAC and the minimal IP protocol stack shown in Figure 9.

4.4 Toward a WURM-OS

WURM-OS divides the surface of the reconfigurable device into two different types of regions: *OS frame* and *task slots*. Figure 10 shows an example with four task slots. The OS frame is loaded into the FPGA at system startup and remains unchanged during the system's operation. The task slots are placeholders for application tasks. The WURM-OS is partitioned between hardware and software and is constituted by the circuits running in the OS frame of the FPGA and control software running on the attached CPU. The main function of the control software is to schedule and load tasks into the task slots during runtime. As all task slots have the same size, a task can be loaded into any slot.

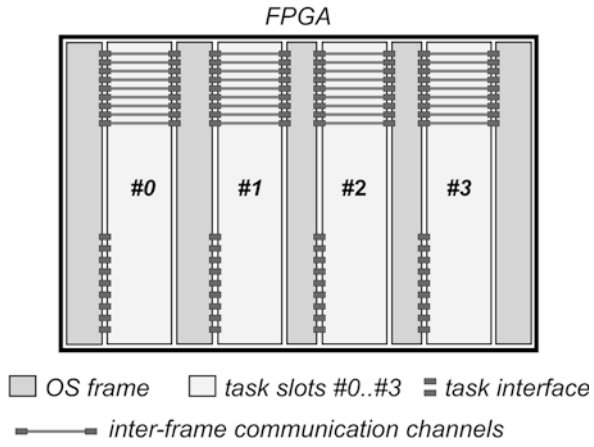

Fig. 10 The partitioning of the FPGA surface into OS frame and task slots

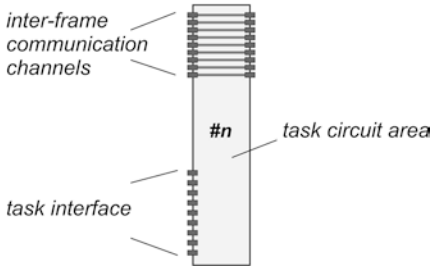

Fig. 11 The WURM-OS task template

To ensure the interoperability between the tasks and the operating system, our design flow includes the automatic generation of *task templates*. As shown in Figure 11, a task template consists of three elements: the task interface, the inter-frame communication channels and the task circuit area. The task interface connects the OS frame with the tasks. WURM-OS controls the tasks via the task interface by issuing commands such as reset, start, or stop. On the other hand, the tasks use the task interface to demand WURM-OS services such as access to buffers and connectivity to external devices. The inter-frame communication channels route signals across the device. Finally, the task circuit area is available for application circuits.

To test the WURM-OS prototype, we have implemented an application that processes a flow of incoming IP data packets on a reconfigurable device with two task slots. The application implements the minimal IP stack and the audio decoder coprocessors described in the previous sections as well as AES encryption and decryption coprocessors. Figure 12 shows the networking part of the application. As shown in the previous section, the IP protocol handling splits into several tasks that are coupled with FIFO buffers. Due to their stringent timing constraints and importance to many applications, we have decided to make the tasks T_1 (Ethernet receiver) and T_5 (Ethernet sender) part of the OS (similar to device drivers in a software OS). Consequently, these tasks are implemented inside the OS frame.

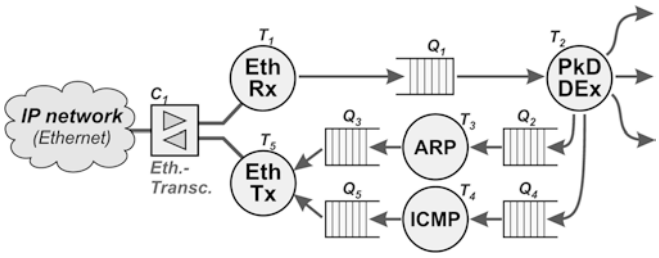

Fig. 12 The networking section of the WURM-OS test application

Incoming IP packets are captured and checked by T_1. If the MAC address and the frame checksum are correct, the Ethernet receiver forwards the packets to queue Q_1. This generates an event to the WURM-OS which subsequently leads to the loading of task T_2 (the packet discriminator/data extractor) into a free task slot. T_2 analyses the packet header. If the packet contains an ARP request, the data is forwarded to queue Q_2 and task T_3 is loaded. In case of an ICMP request, the data is forwarded to queue Q_4 and task T_4 is loaded. Similarly, data for the encryption, decryption and audio decoder coprocessors is forwarded to the appropriate queues.

5 Conclusions and future work

In this paper, we have made the case for reconfigurable hardware in body area computing systems. We have presented WURM, a concept and a prototype of a small embedded wearable node that combines a CPU with a reconfigurable hardware unit. Future work includes the development of:

- a fully autonomous WURM node that receives tasks via the wireless interface
- an extended WURM-OS that includes task and resource management for hardware and software tasks, and
- a prototypical body area computing system consisting of a main module and a number of WURM nodes.

Acknowledgements This work is being carried out in cooperation with the Wearable Computing Laboratory at ETH Zurich and is supported by ETH Zurich (Wearable Computing) and the Swiss National Science Foundation (NCCR MICS).

References

1. van Laerhoven K, Schmidt A and Gellersen HW (2002) Multi-sensor context aware clothing. In: Proceedings of the 6th International Symposium on Wearable Computers (ISWC), Seattle, WA, 7–10 October 2000
2. Kern N, Schiele B, Junker H, Lukowicz P and Tröster G (2002) Wearable sensing to annotate meeting records. In: Proceedings of the 6th International Symposium on Wearable Computers (ISWC), Seattle, WA, 7–10 October 2000

3. Kymissis J, Kendall C, Paradiso J and Gershenfeld N (1998) Parasitic power harvesting in shoes. In: Proceedings of the 2nd International Symposium on Wearable Computers (ISWC), Pittsburgh, PA, 19–20 October 1998
4. Kirstein T, Cottet C, Grzyb J and Tröster G (2002) Textiles for signal transmission in wearables. In: Proceedings of the Workshop on Modeling, Analysis and Middleware Support for Electronic Textiles (MAMSET), San Jose, CA, 6 October 2002
5. Park S, Mackenzie K and Jayaraman S (2002) The wearable motherboard: a framework for personalized mobile information processing (PMIP). In: Proceedings of the 39th Design Automation Confererence (DAC), New Orleans, LA, 10–14 June 2002
6. Herring C (2000) Microprocessors, microcontrollers, and systems in the new millennium. IEEE Micro 20(6):45–51
7. Brown S, Rose J (1996) FPGA and CPLD architectures: a tutorial. IEEE Des Test Comp 13(2):42–57
8. Triscend Corp. (2001) Triscend E5 configurable system-on-chip platform
9. Xilinx, Inc. (2002) Virtex-II Pro platform FPGA handbook
10. Mencer O, Morf M and Flynn MJ (1998) Hardware software tri-design of encryption for mobile communication units. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Seattle, WA, 12–15 May 1998
11. Abnous A, Seno K, Ichikawa Y, Wan M and Rabaey J (1998) Evaluation of a low-power reconfigurable DSP architecture. In: Proceedings of the 5th Reconfigurable Architectures Workshop (RAW), Springer Lecture Notes in Computer Science 1388:55–60
12. Stitt G, Grattan B, Villarreal J and Vahid F (2002) Using on-chip configurable logic to reduce embedded system software energy. In: Proceedings of the 10th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, 22–24 April 2002
13. Rabaey JM, Ammer MJ, da Silva Jr. JL, Patel D, and Roundy S (2000) PicoRadio supports ad hoc ultra-low power wireless networking. IEEE Comp 33(7):42–48
14. Liu Z, Wang Y and Chen T (1998) Audio feature extraction and analysis for scene segmentation and classification. J VLSI Sig Process 20(1/2):61–79
15. van Laerhoven K, Aidoo KA and Lowette S (2001) Real-time analysis of data from many sensors with neural networks. In: Proceedings of the 5th International Symposium on Wearable Computers (ISWC), Zurich, Switzerland, 8–9 October 2001
16. Scalera S, Falco M and Nelson B (2000) A reconfigurable computing architecture for microsensors. In Proceedings of the 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, 17–19 April 2000
17. Mignolet JY, Vernalde S, Verkest D and Lauwereins R (2002) Enabling hardware-software multitasking on a reconfigurable computing platform for networked portable multimedia appliances. In: Proceedings of the 2nd International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), Las Vegas, NV, 24–27 June 2002
18. Yau SS, Karim F (2001) Reconfigurable context-sensitive middleware for ADS applications in mobile ad hoc network environments. In: Proceedings of the 5th International Symposium on Autonomous Decentralized Systems (ISADS), Dallas, TX, 26–28 March 2001
19. Xilinx, Inc. (2002) PicoBlaze 8-bit microcontroller for Virtex devices. Application Note XAPP213
20. Xilinx, Inc. (2002) MicroBlaze hardware reference guide
21. Beutel J, Kasten O, Ringwald M, Siegemund F and Thiele L (2003) Bluetooth smart nodes for ad-hoc networks. TIK Technical Report No. 167, Computer Engineering and Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich
22. Enzler R, Platzner M, Plessl C, Thiele L and Tröster G (2001) Reconfigurable processors for handhelds and wearables: application analysis. In: Proceedings of SPIE 4525:135–146
23. Dyer M, Plessl C and Platzner M (2002) Partially reconfigurable cores for Xilinx Virtex. In: Field-programmable logic and applications, Springer Lecture Notes in Computer Science 2438:292–301
24. Gaisler J (2001) The LEON processor user's manual, Version 2.3.7, Gaisler Research
25. Lerjen M, Zbinden C (2002) Reconfigurable Bluetooth–Ethernet bridge. Master's thesis, Swiss Federal Institute of Technology (ETH)
26. Dyer M, Wirz M (2002) Reconfiguranle system on FPGA. Master's thesis, Swiss Federal Institute of Technology